

# The use of Self-Organizing Maps in Recommender Systems

A survey of the Recommender Systems field and a presentation of a State of the Art Highly Interactive Visual Movie Recommender System

*Sam Gabrielsson and Stefan Gabrielsson*



UPPSALA  
UNIVERSITET

*A Uppsala Master's Thesis in Computer Science 20p, submitted on August 2006 to the Department of Information Technology at the Division of Computer Systems for the degree of Master of Science in Computer Science at Uppsala University.*

Reviewer: Olle Gällmo  
Supervisor: Filip Mican, Discshop Svenska Näthandel AB  
Examiner: Anders Jansson





## Abstract

*“Is one’s entire psyche’s most secret landscape really a fairly public thing, given just a few pieces of information?”* – Douglas R. Hofstadter

This is a thesis about *recommender systems*, and the different *approaches* recommender systems use to solve the *information overload problem*. Our focus lies on two common approaches, *collaborative filtering* and *content based filtering*. Both of these approaches have their weaknesses and strengths. To overcome the weaknesses of each approach, various *hybrid filters* have been developed. We will start by analyzing these three approaches based on previous research literature and will then proceed to implement different variants of these approaches, including our own filtering approach for the movie domain. These implementations will be done in Java and open sourced for further development by other researchers in this area. The results will be *evaluated* and compared against previous research in this area in order to validate our implementations. Evaluation will be done by using *standard metrics* that are commonly used for evaluating the *accuracy* of recommender systems.

Various algorithms from the machine learning community have been used in the effort to improve and solve some of the problems in the previously mentioned approaches. We will concentrate on one such algorithm, Kohonen’s *self-organizing map* algorithm. The self-organizing map algorithm is an unsupervised learning algorithm which we believe is suitable for recommender systems in the movie domain. Our implementation of this algorithm will be used together with collaborative filtering approaches in the effort of designing a recommender system for movies. The result of this approach will be evaluated and compared against the results from our previous implementations and discussed in the context of previous results from the recommender systems research community.

Evaluating the effectiveness of recommender systems is often done by analyzing the accuracy of the recommendations produced by the *techniques* used to implement the different approaches. However, the goal for a recommender system is not only to give accurate recommendations but also to conceive to the user *trust* and encourage the user to *explore* the recommendations. This is more of an interface issue than an algorithmic issue, we have chosen to call this the *recommendation interface problem*. Similar conclusions have been drawn by other researchers and different attempts to solve this has been done. We will summarize and discuss proposed solutions. We will introduce and describe what we call *visual recommendations*, and show how this approach solves the recommendation interface problem by creating a *visual recommender system* called MOV SOM.

The testing and evaluation will be done on the well used MovieLens dataset, as well on a larger dataset taken from an e-commerce site selling DVDs, together with movie attributes provided by the IMDb.

Our empirical evaluation results shows that MOV SOM produces recommendations of movies that are comparable to state of the art techniques and with the combination of our solution to the recommendation interface problem we believe that this approach has a very promising future as a recommender system for movies.

*“I love deadlines. I like the whooshing sound they make as they fly by.”* – Douglas Adams

*To all the makers of TV-series and movies in the world!*

*Thanks for the encouragement, Bill!  
...thanks for introducing us to the SOM, Olle!  
...thanks for the grammar checks, Linda!  
and a thanks to Joss Whedon! ;)*

# Contents

<b>Introduction</b>	<b>2</b>
Contributions . . . . .	3
Outline . . . . .	3
 <b>I Recommender Systems</b>	 <b>5</b>
<b>1 Historical development and related work</b>	<b>6</b>
1.1 Information Retrieval . . . . .	6
1.2 Information Filtering . . . . .	7
1.3 Collaborative Filtering . . . . .	8
1.3.1 Recommendation techniques . . . . .	10
1.3.2 Similarity measures . . . . .	13
1.3.3 Recommendations . . . . .	16
1.4 Content Based Filtering . . . . .	18
1.5 Hybrid Filters . . . . .	21
1.6 Recommender Systems . . . . .	24
1.6.1 Recommender systems in e-commerce . . . . .	24
1.6.2 Trust in recommender systems . . . . .	25
1.6.3 Recommendation approaches . . . . .	26
1.6.4 Problems and challenges of the recommendation approaches . . . . .	28
1.7 Case studies of recommender systems in the movie domain . . . . .	29
1.7.1 MovieLens . . . . .	30
1.7.2 Internet Movie Database . . . . .	33
1.7.3 All Movie Guide . . . . .	35
1.7.4 Amazon . . . . .	38
1.7.5 Recommendations for the movie The Thing . . . . .	44
1.7.6 Conclusions . . . . .	46
1.8 Patents . . . . .	47
1.9 Where are we today? . . . . .	47
 <b>2 Implementation and evaluation of common recommendation techniques</b>	 <b>48</b>
2.1 Recommendation profiles . . . . .	49
2.2 Information analysis . . . . .	49
2.3 Profile representation . . . . .	49
2.3.1 User rating models . . . . .	49
2.3.2 User transaction models . . . . .	49
2.3.3 Item rating models . . . . .	50
2.3.4 Item transaction models . . . . .	50
2.3.5 Item attribute models . . . . .	50
2.4 Recommendation techniques . . . . .	50
2.4.1 Baseline Statistical Filtering (BASELINESF) . . . . .	50
2.4.2 User based Collaborative Filtering (UCF) . . . . .	52
2.4.3 Random Neighbors User based Collaborative Filtering (RNUCF) . . . . .	53
2.4.4 Trust User based Collaborative Filtering (TRUSTUCF) . . . . .	54
2.4.5 Common Items Prioritizing User based Collaborative Filtering (CIPUCF) . . . . .	56

2.4.6	Item based Collaborative Filtering (ICF)	56
2.4.7	Random Neighbors Item based Collaborative Filtering (RNICF)	59
2.4.8	Personalized Content Based Filtering (PCBF)	59
2.4.9	Common attribute similarity algorithm	60
2.4.10	Top-N User based Collaborative Filtering (TOPNUCF)	61
2.4.11	Top-N Item based Collaborative Filtering (TOPNICF)	61
2.4.12	Top-N Personalized Content Based Filtering (TOPNPCBF)	61
2.4.13	Top-N Content Based Filtering (TOPNCBF)	61
2.4.14	Transaction based Top-N User based Collaborative Filtering (TBTOPNUCF)	62
2.4.15	Transaction based Top-N Item based Collaborative Filtering (TBTOPNICF)	63
2.5	Evaluation of recommender systems	64
2.5.1	Evaluation data	65
2.5.2	Evaluation protocols	65
2.5.3	Splitting evaluation data	66
2.5.4	Prediction evaluation metrics	66
2.5.5	Ranking evaluation metrics	70
2.6	Evaluation data	74
2.6.1	Consistent MovieLens rating dataset (CML)	74
2.6.2	Discshop rating dataset (DS)	78
2.6.3	Discshop rating dataset 2 (DS2)	82
2.6.4	Discshop transaction dataset (DS-T)	86
2.6.5	Discshop attribute dataset (DS-A)	88
2.6.6	IMDb attribute dataset (IMDB)	89
2.7	Evaluation results	90
2.7.1	Accuracy of predictions	90
2.7.2	Relevance of Top-N ranking lists	93
2.8	Summary and conclusions	95

## II Self-Organization applied to Recommender Systems 101

<b>3</b>	<b>Artificial neural networks</b>	<b>102</b>
3.1	The brain	102
3.2	The biological neuron	103
3.3	The artificial neuron	103
3.4	Formal definition of artificial neural networks	105
3.5	Artificial neural network structures and learning algorithms	106
3.5.1	Feedforward networks	106
3.5.2	Recurrent networks	107
3.6	Learning Rules	107
3.6.1	Error-correction learning	108
3.6.2	Hebbian Learning	111
3.6.3	Competitive learning	111
3.7	Learning paradigms	112
3.8	Learning protocols	113
3.9	A brief history of artificial neural networks	113
<b>4</b>	<b>Self-Organizing Maps</b>	<b>117</b>
4.1	Brain maps	118
4.2	Requirements for self-organization	119
4.3	The Mexican Hat	120
4.4	The SOM paradigm	122
4.5	The incremental SOM algorithm	122
4.5.1	Selection of the best matching node ("Winner-takes-all")	123
4.5.2	Adaptation (Updating of the weight vectors)	123
4.6	Selection of parameters	124
4.7	Incomplete data	125

4.8	Quality measure of the SOM . . . . .	125
4.8.1	Average quantization error . . . . .	125
4.8.2	Topographic error . . . . .	125
4.9	Summary of the SOM algorithm . . . . .	126
4.10	The Dot-Product SOM algorithm . . . . .	126
4.11	The batch SOM algorithm . . . . .	126
4.12	The SOM as a clustering and projection algorithm . . . . .	127
4.12.1	Vector quantization . . . . .	128
4.12.2	Vector projection . . . . .	129
4.12.3	Comparison of the SOM to other VQ and VP algorithms . . . . .	130
4.13	Visualizing the SOM . . . . .	132
4.14	Properties of the SOM . . . . .	133
4.14.1	Approximation of the input space . . . . .	134
4.14.2	Topological ordering . . . . .	134
4.14.3	Density matching . . . . .	136
4.15	Theoretical aspects of the SOM algorithm . . . . .	137
4.16	SOM Toolbox for Matlab . . . . .	138
4.17	SOM based applications . . . . .	138
4.18	Applications of the SOM in recommender systems . . . . .	138
<b>5</b>	<b>Implementation and evaluation of SOM based recommendation techniques</b>	<b>140</b>
5.1	SOM implementation . . . . .	140
5.2	SOM based recommendation techniques . . . . .	143
5.2.1	SOM User based Collaborative Filtering (SOMUCF) . . . . .	143
5.2.2	SOM Item based Collaborative Filtering (SOMICF) . . . . .	144
5.2.3	Item SOM User based Collaborative Filtering (ISOMUCF) . . . . .	145
5.2.4	Model Predicting SOM User based Collaborative Filtering (MPSOMUCF) . . . . .	146
5.2.5	SOM Goodness Collaborative Filtering (SOMGOODNESSCF) . . . . .	146
5.2.6	Top-N SOM User based Collaborative Filtering (TOPNSOMUCF) . . . . .	147
5.2.7	Top-N SOM Item based Collaborative Filtering (TOPNSOMICF) . . . . .	147
5.2.8	Top-N SOM Goodness Collaborative Filtering (TOPNSOMGOODNESSCF) . . . . .	147
5.3	Evaluation data . . . . .	147
5.4	Evaluation results . . . . .	148
5.4.1	Accuracy of predictions . . . . .	148
5.4.2	Relevance of Top-N ranking lists . . . . .	151
5.5	Conclusions . . . . .	152
<b>III</b>	<b>The recommendation interface problem</b>	<b>155</b>
<b>6</b>	<b>Trust in recommender systems</b>	<b>156</b>
6.1	Transparency of recommendations . . . . .	156
6.2	Explanation of recommendations . . . . .	157
6.3	Interactive design of recommender systems . . . . .	159
<b>7</b>	<b>Visual recommendations</b>	<b>160</b>
<b>8</b>	<b>MOVSOM - State of the art</b>	<b>170</b>
8.1	MOVSOM map . . . . .	170
8.2	MOVSOM interface . . . . .	171
8.3	MOVSOM architecture . . . . .	174
8.4	Challenges . . . . .	176
8.4.1	Introduction of new movies and new users . . . . .	176
8.4.2	Retraining the SOM . . . . .	176
	<b>Conclusions</b>	<b>177</b>
	<b>Future work</b>	<b>178</b>
	<b>References</b>	<b>179</b>



## Abbreviations

**BMN** Best matching node  
**CBF** Content based filtering  
**CCA** Curvilinear component analysis  
**CF** Collaborative filtering  
**IDF** Inverse document frequency  
**MDS** Multidimensional scaling  
**PCA** Principal component analysis  
**SOM** Self-Organizing Map  
**VP** Vector projection  
**VQ** Vector quantization  
**VQ-P** Vector quantization-projection

# Introduction

*“The most exciting phrase to hear in science, the one that heralds new discoveries, is not ‘Eureka!’ (I found it!) but ‘That’s funny...’” – Isaac Asimov*

*Recommender systems* are one solution to the information overload problem. Other solutions can for example be found in the field of information retrieval and information filtering, where search engines like Google and various text-retrieval applications are successful applications developed to deal with the problem. Recommender systems emerged as an independent research area around the mid 90’s and has since then become a broad research topic with substantial integration of machine learning and information filtering. E-commerce sites use recommender systems for making recommendations that are tailored to a user’s preferences while at the same time increasing their own sales. Although recommendations can be *non-personalized*, like “customers who bought this item also bought these items” or popularity based, e.g. top lists, resulting in the same recommendations to all customers, recommender systems try to make recommendations as *personalized* as possible.

The main goal for a recommender system is to provide the user with recommendations that reflect the user’s personal taste and to convince the user to trust and explore the given recommendations. Applications of this has been successfully used on the Internet by e-commerce sites like Amazon.com, that offers millions of products to its customers, and by communities in the entertainment domain like MovieLens<sup>1</sup>, a research project that runs a website where people can become members and receive recommendations for movies.

Recommender systems that make personal recommendations achieve their goal by maintaining profiles for the users that consists of their preferences. The profiles are used as filters, only items that match user’s preferences will slip through and be presented as recommendations. Depending on the domain, items can be books, movies, WebPages, Usenet news articles, etc.

Throughout the text we will be referring to the user that gets a recommendation as the *active user* and the item that is recommended as the *active item*.

Two main filtering approaches for making personalized recommendations that have emerged over the years are *collaborative filtering* (CF) and *content based filtering* (CBF). CF uses the opinions of other users, that are similar to the active user, as a filter. CBF only uses the preferences of the active user as a filter. In this thesis we will try out different established techniques that use both of the approaches, as well as some techniques suggested by ourselves. We will for example develop our own CBF for the movie domain where movies will be represented with attributes such as directors, actors, genres, etc. Movies will be matched against movies in a user’s profile, where similarity is taken as a linear combination over the attributes, and the movies with highest similarity will be recommended.

The CF and CBF approach both have their weaknesses and strengths. With knowledge of issues related to one approach that aren’t present in the other, research has been done on how to best combine these two approaches into *hybrid filters*.

Different methods have been tried to combine the two approaches into new techniques, many taken from the field of artificial intelligence. We will use a technique from the field of artificial neural networks called Kohonen’s *self-organizing map* (SOM). The SOM is a mapping from high dimensional data onto e.g. a two dimensional grid, that preserves the neighborhoods of the original data. This allows us to cluster and form neighborhoods of users and items on large two dimensional maps. A user map will be used in combination with CF, i.e. users in the active users neighborhood on the map will be used as recommenders for the active user. An item map will be used in combination with CBF, i.e. movies that

---

<sup>1</sup><http://movielens.umn.edu/>

are similar to the movies in the active user's profile will be recommended to the user. Additionally a heuristic goodness function will be used that based on the recommendations from each map weights how safe and serendipitous the recommendations are.

We believe that a good recommender system should be able to give recommendations that are new, unexpected, trust inspiring and somewhat transparent, so that the users can see the logic behind why a particular item was recommended. To show the logic behind a recommendation we will show the users the maps produced by the SOM algorithm. We believe that if a user can zoom in on the recommenders profiles and the recommended movie's neighborhood, then this will be a sufficient explanation for why the recommendation was made.

## Contributions

The main objective of this thesis is to evaluate the usability of Kohonen's SOM in the construction of a recommender system for movies. The reason for this is two-fold; some previously known problems with content based filtering and collaborative filtering can be solved, and secondly, the visualization ability allows the motivations behind recommendations to be made transparent to the users in a very simply and intuitive manner.

Our contributions are:

1. We will release new datasets<sup>2</sup> that can be used for testing recommender systems based on transaction data, rating data and attribute data.
2. We will give a detailed analysis of the datasets we provide.
3. We will configure the MovieLens dataset so it can be used together with IMDb attribute data.
4. We will release a recommendation technique "framework"<sup>3</sup> in Java that can be used by others to test their similarity and prediction algorithms.
5. We will implement common recommendation techniques and provide detailed evaluations on clearly described comparable evaluation datasets.
6. We will implement our own public experimental recommender system<sup>4</sup> for movies based on the SOM, with focus on visual recommendations and new ways for users to explore recommendations.

## Outline

In Part I, the first chapter "*Historical development and related work*" surveys the historical development that resulted in today's recommender systems by going through and presenting related work. Issues related to recommender systems will be discussed in the context of others research. The chapter ends with a presentation of a few well known recommender systems. In the next chapter "*Implementation and evaluation of common recommendation techniques*" we present and evaluate implementations of common recommendation techniques, as well as a few variations of our own. We provide a thorough statistical analysis of the data we use for evaluation. We provide a detailed explanation of various common evaluation metrics used for evaluating recommender systems, and we discuss how we are going to use them to evaluate the recommendation techniques we have implemented. We end this part with a presentation of the results from the evaluations.

Part II begins with the chapter "*Artificial neural networks*", which serves as an introduction to the field. Commonly used learning algorithms will be discussed as well as different neural networks. The chapter ends with a brief historical survey of the field of artificial neural networks. This is followed by the chapter "*Self-Organizing Maps*" where we introduce Kohonen's SOM algorithm both as a biologically inspired algorithm and as a clustering and projection algorithm. Its visualization capabilities will be discussed as well as some other related issues regarding the SOM. The chapter ends with a presentation

---

<sup>2</sup>Datasets used in this thesis are available for download from <http://rslab.movsom.com/data/>.

<sup>3</sup>Our recommendation technique "framework" which is used for the evaluations in this thesis is available for download from <http://rslab.movsom.com/tools/rslab/>, it includes all of the recommendation techniques described in this thesis.

<sup>4</sup>Our public experimental recommender system MOV SOM can be found at <http://www.movsom.com/>.

of some related work in the area of using the SOM in the recommendation process. In the final chapter of Part II, with the title "*Implementation and evaluation of SOM based recommendation techniques*" we will implement and evaluate a few recommendation techniques based on the SOM.

Some of these techniques from Part II will be used again in Part III which begins with the chapter "*Trust in recommender systems*" in which we will discuss the design and trust issues regarding recommender systems by presenting some results from the literature as well as our own solutions to these issues. This is followed by a chapter on what we refer to as "*Visual recommendations*". Finally we come to the last chapter entitled "*MOVSOM - State of the art*" where we describe the implementation of an experimental recommender system called MOVSOM. MOVSOM relies heavily on visualizing important aspects of the recommendation process and by that conceiving to the user *trust* and encouraging the user to explore the recommendations further. MOVSOM uses the topological ordering property of the SOM as well as its clustering and projection capabilities, which are implemented and evaluated in Part II. The benefits of MOVSOM will be discussed and a few similar web-based solutions in other domains will also be examined and presented. Ideas for future work on the MOVSOM will then be presented. In the end we conclude.

# Part I

## Recommender Systems

*“Those people who think they know  
everything are a great annoyance  
to those of us who do.”*  
– Isaac Asimov

This part consists of two chapters. The first is theoretical, in which necessary concepts are explained regarding the field of recommender systems. We begin by presenting the development of the field, from its roots in information retrieval to where it is today, a field of its own, influenced by many other. Historical important events and work done by others are presented. A detailed analysis of various recommendation techniques are given and a brief discussion of problems and challenges regarding some of these techniques are presented. The chapter ends with a presentation of some well known websites and how they use recommender systems on their sites.

In chapter two, recommendation techniques that we have chosen to implement are evaluated and the results discussed. Different measures for evaluating recommender systems are presented and used. All recommendation techniques are presented in clearly written pseudo code.

# Chapter 1

## Historical development and related work

*“I have had my results for a long time: but I do not yet know how I am to arrive at them.”*  
– Karl Friedrich Gauß

In 2003 a study was conducted [Lyman et al., 2003] that tried to estimate how much new information is created each year. The conclusion was that the world produces between 1 and 2 exabytes of unique information each year, roughly about 250 megabytes for each human being. This amount of information affects most of us in our daily life as we constantly find ourselves overwhelmed by it in many different forms. Think of all TV programs there are to choose among. Which news stories are worth paying attention to? Look at all the new email that keeps on arriving to the inbox? Which restaurant to eat lunch at? And what to eat at the restaurant? Which is the right gift? Which is the right movie to rent? What book, cd or movie to buy from Amazon? A commonly used term for these problems is *information overload*, something that will happen to all of us sooner or later.

Since ninety-two percent of all the information is stored on magnetic media, primarily hard disks, the research and development of computerized tools for storing, organizing and searching information is highly needed, and indeed, in the field of *information retrieval* and *information filtering*, this has been an issue of interest for more than half a century. Applications have been developed in both fields that are suited for different data domains. In the field of information retrieval applications such as search engines for the Web and text-retrieval applications that searches large text-document collections have been developed. In the field of information filtering we have spam filters that filters incoming mail and makes sure no unwanted mail arrives in our mailbox. And from both fields we have recommender systems that retrieve and filter information by figuring out what our needs are, and uses this knowledge to recommend the *information* that we will *probably* have most use of.

### 1.1 Information Retrieval

One definition of information retrieval can be found in the book *“Information retrieval”* by Rijsbergen [Van Rijsbergen, 1979]

*“An information retrieval system does not inform (i.e. change the knowledge of) the user on the subject of his inquiry. It merely informs on the existence (or non-existence) and whereabouts of documents relating to his request.”*

In its basic form, information retrieval is concerned with finding text-documents that match some search criteria. In a typical information retrieval system the user types in a few keywords, a search query, that best describe the wanted information, the information retrieval system then finds this information by matching the keywords against the contents of information stored as textual data. Examples of information retrieval systems are web search engines like Google [Brin and Page, 1998] that matches search queries against the textual contents of web documents, and returns the documents that best

match the keywords, usually sorted according to how much they overlap with the search query and by their link structure. Text and document retrieval applications are by far the most popular use of information retrieval systems. One drawback with the query style for retrieving information is that the users have to type in a new question every time they want to know if new information is available for a subject of interest.

Pioneering work on this matter was done in the late 50's by H.P. Luhn, who was the first to automate the process of information retrieval. The purpose of his work was to make it easier for readers of technical literature to quickly and accurately identify the topic of a paper. This was done by first making the documents machine-readable, using punch-cards, and then using statistical software to calculate word frequencies in order to find significant words and sentences that could be used for abstract creation [Luhn, 1958a]. In his paper "*A Business Intelligence System*" [Luhn, 1958b], he also describes a system that manages *user profiles* that consists of the users interests coded as keywords. The system retrieves and stores information that matches the user profile by comparing the user profile against a database of documents. This approach is similar to what later on will be referred to as *information filtering*.

Beginning in the 1960's a lot of research was done in the field of information retrieval. One of the major researchers in this field was Gerald Salton, who was the lead researcher of a project called the SMART information retrieval system [Salton, 1971]. Important concepts like the *vector space model*, *term-weighting* and *relevance feedback* was introduced and developed as a part of this research. The system itself was an experimental text-retrieval application that allowed researchers to experiment and evaluate new approaches in information retrieval. The applications that were developed during these years were proven to perform well on relatively small text collections (on the order of thousands of documents) but whether or not they would scale to larger text collections wasn't clear until after the *Text REtrieval Conference* (TREC)<sup>1</sup> in 1992. The purpose of TREC was to support research within the information retrieval community by providing the infrastructure necessary for large-scale evaluation of text retrieval methodologies. TREC resulted in large text collections being made available for researchers. TREC is co-sponsored by the National Institute of Standards and Technology and U.S. Department of Defence and is now an yearly event. With the distribution of a gigabyte of text with hundreds of queries, the performance of the retrieval systems could now be judged by their actual performance. New and improved applications could, thanks to this, be developed that were suitable for large text collections. For example, in the early days of the Web, searching the Web was done by using algorithms that had been developed for searching large text collections [Singhal, 2001].

The field of information retrieval is still even after nearly fifty years of research a very active and important research field due to the increasing amount of information being generated - not in the least due to the introduction of the World Wide Web in the beginning of the 1990's. The research area has gone from its primary goal of indexing text and searching for useful documents in text collections to include research on modelling, document classification and categorization, systems architecture, user interfaces, data visualization, filtering, language analysis, etc. Some modern applications of information retrieval are search-engines, bibliographic systems and digital libraries. See for example the book "*Modern Information Retrieval*" [Baeza-Yates and Ribeiro-Neto, 1999] for an overview of the research in information retrieval with a focus on the algorithms and techniques used in information retrieval systems.

## 1.2 Information Filtering

Today's recommender systems have their origin in the field of information filtering, a term coined in 1982 by Peter J. Denning in his ACM president's letter titled "*Electronic junk*" [Denning, 1982]. In the paper Denning writes that:

*"The visibility of personal computers, individual workstations, and local area networks has focused most of the attention on generating information – the process of producing documents and disseminating them. It is now time to focus more attention on receiving information—the process of controlling and filtering information that reaches the persons who must use it."*

Dennings describes one approach for how to deal with the problem of filtering information being received, called content filters. He describes *content filters* as a process by which for example the contents of emails

---

<sup>1</sup><http://trec.nist.gov/>

are scanned in order to decide what to do with them; discard them or forward them. He ends the letter with a open request that something must be done about the growing problem of how to handle the quantity of information that we produce.

The 1970's was a big decade when it came to databases, which in combination with the falling prices of storage space led to an increasing amount of digital information being available in the 80's. There was also an increasing availability of computers as well as an increasing usage of computer networks to disseminate information. It was probably the beginning of these trends that Dennings had observed and which led to his request for more effective filtering techniques. As a result, there was an increased activity in the research of filtering systems during the 80's and several papers were published on the subject, the paper by Malone et al. [Malone et al., 1987] published in 1987 being one of the more influential ones. Their paper discusses three paradigms for information filtering called *cognitive*, *economic* and *social*. Their definition of cognitive filtering is equivalent to the content filter defined by Dennings and is now commonly referred to as content based filtering. Economic filtering is based on the fact that a user often has to make a cost-versus-value decision when it comes to processing information, e.g. "is it worth my time to read this document or email". Social filtering is defined as a process where users annotate documents they have read and where the annotations are then used to represent the documents. Users decide, based on the annotations, if the documents are worth reading. They thought that, if practical, this approach could be used on items *regardless* of whether or not the contents of the item can be represented in a way suitable for content filtering.

In November 1991, Bellcore hosted a Workshop on High Performance Information Filtering. The workshop brought together researchers with interests in the creation of large-scaled personalized information delivery systems and covered various aspects of this area and its relation to information retrieval. The workshop resulted in a special issue on information filtering in *Communications of the ACM* [Cohen, 1992]. One of the articles in the special issue was written by Belkin and Croft [Belkin and Croft, 1992], in it they discuss the difference between information filtering and information retrieval. They come to the conclusion that both techniques are very similar on an abstract level and has the same goal, i.e. to provide relevant information to the user, but define some differences that are significant for information systems; filter systems are suitable for *long-termed user-profiles* on dynamical data whereas retrieval systems are suitable for *short-termed user profiles* on static data. The traditional way of retrieving information is to ask a query (short-termed) that is matched against available information. A system can however also build up a user profile (long-termed) which is used in connection with every query, this means that personalization can be done, each answer is adjusted to the users preferences as well as the actual query. The other articles in the special issue give examples for applications of information filtering and extensions to information retrieval.

## 1.3 Collaborative Filtering

*"Remember to always be yourself. Unless you suck."* – Joss Whedon

One article [Goldberg et al., 1992] in the special issue on information filtering in *Communications of the ACM* [Cohen, 1992] took a different approach to information filtering. They implemented their own version of social filtering in a system called Tapestry. Tapestry allowed a small community of users to add annotations such as ratings and text comments to documents they had read. Users could then create queries like "receive all documents that Giles likes". The system relied on the users all knowing each other, so that users could judge from a personal basis or by reputation if the reviewer of a document was trustworthy or not. Two major drawbacks with the system was that it did not scale well to larger user groups (as everybody had to know each other), and the filtering process had to be done manually by the users (the formulation of specific queries). They chose to call their approach *collaborative filtering*, which also became the name for a new direction in the research area of information filtering.

The new research direction resulted in further research into collaborative filtering, and the development of new collaborative filtering systems, where *GroupLens* [Resnick et al., 1994] for Usenet News, *Ringo* [Shardanand and Maes, 1995] for music and *video@bellacore* [Hill et al., 1995] for movies are among the more notable ones. The main contributions of these new systems were to automate the filtering process of Tapestry and make it more scaleable. By letting users be represented by their ratings on items, the systems could automatically find other users that had a similar type of rating behavior



as the active user. Users with similar rating behavior could then become recommenders for the active user on items they had rated and which the active user hadn't rated. This approach was based on the heuristic that *users that have agreed in the past probably will agree in the future*. As users were no longer required to all know each other the former scalability issue of Tapestry was solved. In other words, the users that were found similar to you would take on the same role as the trusted well-known reviewer in the Tapestry system, with the difference that now you didn't know who the reviewers were, the trust was instead put on the system to find them.

Motivated by the research done into collaborative filtering since the 1991 workshop where Tapestry was presented, a new workshop was held in 1996 by Berkley, this time the topic was collaborative filtering. The workshop resulted in a special March issue of 1997's *Communications of the ACM* [Crawford, 1997], covering the new research into collaborative filtering. It was this year that collaborative filtering got its real major breakthrough in the research community at large and the term *recommender system* got coined.

While the *automated collaborative filtering systems* had solved the scalability issue of Tapestry, now allowing a theoretically unlimited number of users that didn't need to know each other, a new scalability issue had manifested. The increasing number of users required an increasing amount of memory and computation time. Breese suggested in [Breese et al., 1998] a model based approach to CF, and therefore decided to divide CF techniques into two classes, *memory based collaborative filtering* and *model based collaborative filtering*.

*Memory based collaborative filtering* techniques were to correspond to the earlier automated CF techniques that required that the entire database of user's ratings on items was kept in memory during the recommendation process. Since the entire rating database is kept in memory, new ratings can immediately be taken into account as they become available. Recommendations for the active user can be produced using variations of nearest neighbor algorithms, memory based CF is for this reason sometimes referred to as *neighborhood based CF*. These algorithms utilize the whole rating database, to find neighbors to the active user, by calculating similarity between the active user and all the other users in the rating database based on how they have rated items. Users that are found to be similar to the active user are then used as recommenders in the recommendation process. The major drawback with memory based CF techniques is as already noted that they tend to scale very poorly, larger rating databases require more memory and more calculations which slow down the recommendation process.

*Model based collaborative filtering* techniques, Breese suggested, will on the other hand only use the rating database to construct a model (hence the name) that can be used to make recommendations. The construction of such a model is primarily based on the rating database, however once the model is created the rating database is no longer needed. It is thus not necessary to keep the huge rating database in memory, which typically should improve the performance and scalability issues. But since it is just a model, it may (depending on the model) be necessary to reconstruct it as more ratings become available in order to accurately model the relations of users and items in the rating database. The need to reconstruct the model can be a significant drawback with this approach.

Breese proposes in [Breese et al., 1998] a *probabilistic* perspective to model based CF, where calculating the predicted value  $p_{u,i}$  for an item  $i$  for the active user  $u$  can be viewed as calculating the active user's expected rating  $r_{u,i}$  on the active item, given what we know about the active user. The information available about the active user is the set  $u_r$  of all ratings given by active user. Assuming ratings to be in the interval  $[R_{min}, R_{max}]$  the following formula could then be used to calculate the predicted value:

$$p_{u,i} = E(r_{u,i}) = \sum_{v=R_{min}}^{R_{max}} [v \times P(r_{u,i} = v | u_r)] \quad (1.1)$$

Where  $P$  is the *conditional probability* that the active user will give the active item a particular rating. To estimate the probability  $P$ , Breese proposes two alternative probabilistic models: *Bayesian Networks*, where each item is represented as a node in a network and the states of the nodes correspond to the possible ratings for each item; and *Bayesian clustering* where the idea is that there are certain groups or types of users capturing a common set of preferences and tastes, as such user clusters are formed, and the probability that the active user belongs to each of them is estimated, using the estimated class membership probabilities predictions can be calculated.

Several other probabilistic models have been proposed in the literature, some more advanced than others, such as the work of [Shani et al., 2002] where they see the recommendation process as a *sequential decision process* and proposes the use of *Markov decision chains* to create a model. However, they don't report any better accuracy than the models proposed by Breese. Many other non-probabilistic model based CF techniques have been suggested, such as the machine learning techniques found in [Billsus and Pazzani, 1998], where artificial neural networks are used together with feature extraction techniques like *singular value decomposition* to build prediction models. They claim a slightly better result than the correlation based techniques proposed earlier by e.g. GroupLens [Resnick et al., 1994].

In 1999 Herlocker and others [Herlocker et al., 1999] summarizes what can be referred to as the state of the art of CF. The automated CF problem space is formulated as *predicting how well a user will like an item that the user has not yet rated given the ratings for a community of users*. An alternative problem space definition is also given where users and items are represented as a matrix where the cells consist of ratings given by the users on the items, the collaborative filtering recommendation problem can then be formulated as *predicting the missing ratings in the user item rating matrix* [Figure 1.1]. The most prevalent algorithms used by collaborative filtering techniques are considered to be neighborhood based (which is still true). A framework for performing collaborative filtering using neighborhood based methods is presented and discussed in detail. An important and still widely used extension to neighborhood based collaborative filtering methods called *significance weighting* is introduced, where the similarity between users is down weighted according to number of co-rated items.

An alternative approach to collaborative filtering is proposed in [Sarwar et al., 2001] where the relationship between items, instead of users, is used. Sarwar et. al developed a new technique that they chose to call *item based collaborative filtering*, in contrast to the techniques described by [Herlocker et al., 1999] which they refer to as *user based collaborative filtering*. Recommendations are based on what the active user thinks of items similar to the active item, instead of what the neighbors of the active user thinks of the active item [Figure 1.2]. The similarity between items is essentially based on the heuristic that *a person's opinion on two items that are similar doesn't vary much*.

Many may consider [Sarwar et al., 2001] to be the first time an item based CF technique is described in a public paper, however as early as 1995 a similar approach was described in [Shardanand and Maes, 1995] in form of what they called the *artist artist algorithm*, where they used the similarities between artists to make recommendations. The artist similarities were based on user ratings. However, they reported worse results compared to their other user based algorithm. No references is made to the scalability problem so this was probably just a test to see how well the approach stood up to their user based CF technique

A few years later in the paper [Linden et al., 2003] a description is given of the CF technique used by Amazon<sup>2</sup>, who also holds a patent for the technique they use. A brief and not very detailed description of an item based CF technique that reminds a lot of the one described in [Sarwar et al., 2001] is given.

The technique presented in [Sarwar et al., 2001] has been further extended in [Deshpande and Karypis, 2004] for *top n item recommendations*. They propose an alternative algorithm called *item set to item CF* where the difference lies in that the item similarity is computed between a single item and a *set* of items instead of between two single items.

The empirical results for item based CF shown in [Sarwar et al., 2001] show that a model that only uses the 25 most similar items for each item will produce recommendations that are nearly 96% correct compared to a full model that uses all item similarities. Both [Sarwar et al., 2001] and [Deshpande and Karypis, 2004] show comparable results against user based CF, but with a much faster computation time and no scalability problems in relation to the size of the user item rating matrix, allowing for better scalability to systems with millions of users and items.

### 1.3.1 Recommendation techniques

Two prevalent *techniques* based on the CF recommendation *approach* that have emerged over the years are *user based CF* as described in [Resnick et al., 1994], [Shardanand and Maes, 1995], [Herlocker et al., 1999] and *item based CF* as described in [Shardanand and Maes, 1995], [Sarwar et al., 2001], [Linden et al., 2003], [Deshpande and Karypis, 2004]. While both techniques rely on the CF approach where a large group of users ratings are used as a basis to make recommendations, the techniques implement the CF approach

---

<sup>2</sup><http://www.amazon.com/>

	The Thing	Land of the Dead	Magnolia	Kill Bill
Giles	5	4	1	?
Buffy	5	5	-	5
Spike	5	1	-	1
Snyder	3	3	4	-

Figure 1.1: User item rating matrix where items are movies and cells thus consists of user ratings on movies. Shadowed cells shows which ratings are used when comparing Giles’s row of ratings to Buffy’s row of ratings. Only movies rated by both users can be considered.

in two very distinct ways. The names of the techniques vary greatly among papers (*user based*, *user to user*, *neighborhood based*, sometimes more specific such as *artist to artist* etc.), in this paper we have chosen to use the names user based CF and item based CF based on the historical development of CF and the distinctions necessary to make and will use them throughout this paper.

### User based Collaborative Filtering

User based CF is a memory based technique for making recommendations where the main task is to find users that are similar to an active user. The users similar to the active user, usually referred to as the active user’s *neighbors*, are used as recommenders for the active user in the recommendation process.

In [Herlocker et al., 1999] a framework is presented for performing user based CF (referred to as neighborhood based CF in the paper). The framework presented by Herlocker assumes users have been represented using their ratings on items, and instead focuses on the neighborhood formation and the prediction computation. Herlocker’s framework, with additional explanations is as follows:

1. *Weigh all users with respect to their similarity with the active user.* Define a similarity function  $similarity(u, n)$  that measures the similarity between the active user  $u$  and a neighbor  $n$  (since user’s are represented using their ratings profiles consisting of ratings on items, similarity is computed based on the users rating profiles). The more similar a neighbor is to the active user, the more weight is put on his influence on the final recommendation. One should observe that the introduction of a similarity measure as a weight value is an heuristic artifact that is used only for the purpose of differentiating between levels of user similarity and at the same time to simplify the rating estimation. Any similarity function can be used as long as the calculation is normalized (i.e. similarities lie in a expected interval, which typically is  $[0, 1]$  or  $[-1, 1]$ ).
2. *Select a subset of users to use as recommenders.* The subset of users (the relevant *neighbors* of the active user) can be selected in many different ways. Simple ways for selecting neighbors is to choose those that have a similarity above a fixed *threshold value*, or to choose the  $k$  most similar users according to calculated similarity or according to number of commonly rated items. While theoretically all users could be used given that similarities have been calculated, selection of a subset is often necessary practically. For example, similarities *may* be so unreliable that using a subset of the most trusted similarities is highly necessary to get good recommendations.
3. *Compute predictions using a weighted combination of selected neighbors’ ratings.* The predicted value  $p_{u,i}$  for the active item  $i$  that the active user  $u$  has not yet rated is computed as an aggregation of the rating of each neighbor  $n$  for the same item  $i$ , i.e.  $p_{u,i} = aggr(r_{n,i})$ .

The user based CF recommendation technique and the proposed more detailed framework thus leaves open the selection of similarity measure, the method for forming neighborhoods and the definition of the  $aggr(r_{n,i})$  function for making predictions. Much research has been done on possible ways of implementing and varying these parts of the user based CF technique.

In general the problem solved by the user based CF technique can be defined as predicting the missing values in a (typically very large and very sparse) user item rating matrix [Herlocker et al., 1999], where each row represents a user and each column represents an item. For example the user item rating matrix given in [Figure 1.1] shows that Giles hasn’t seen the movie Kill Bill.

Whom of the other users should Giles trust to make the recommendation? By studying how well Giles’s row of ratings in the matrix matches Buffy’s, Spike’s and Snyder’s row respectively, it can be

	The Thing	Land of the Dead	Magnolia	Kill Bill
Giles	5	4	1	?
Buffy	5	5	-	5
Spike	5	1	-	1
Snyder	3	3	4	-

Figure 1.2: User item rating matrix where items are movies and cells thus consists of user ratings on movies. Shadowed cells shows which ratings are used when comparing the movie Land of the Dead's column of ratings to the movie Kill Bill's column of ratings, only ratings by users that have rated both movies are considered.

determined who should be trusted, and even to which degree they can be trusted. Trust is in this case simply based on the heuristic that *users that have agreed in the past probably will agree in the future*, as such Buffy, who's row of ratings is most similar to Giles's row of ratings, should be trusted the most. Spike is difficult to trust as he agrees and disagrees sometimes, and Snyder can't be trusted at all, which doesn't matter however as Snyder hasn't seen Kill Bill and thus can't predict on it for Giles.

### Item based Collaborative Filtering

Item based collaborative filtering is a model based technique for making recommendations where the main task is to find items that are similar to an active item. The populations ratings on items is used to determine item similarity. Recommendations are formed by considering the active users ratings on items similar to the active item, usually referred to as the active items neighbors (to be clear, the neighbors in item based CF are items, not users). It is assumed that items have been represented using the users' ratings on the items. The recommendation technique then consists of the following three major steps:

1. *Weigh all items with respect to their similarity to the active item.* Define a similarity function  $similarity(i, j)$  that measures the similarity between the active item  $i$  and another item  $j$  (since item's are represented using rating profiles consisting of users ratings on the item in question, the similarity is computed based on the items rating profiles).
2. *Form the active item's neighborhood of similar items.* The neighborhood can either be such that it consists of the most similar items out of all items, independent of whether the active user has rated them or not, or it can be the set of the most similar items out of those the active user has rated. Typically only a subset of the  $k$  most similar items are selected.
3. Compute a prediction on the active item for the active user using a prediction algorithm that is based on *how the active user has rated items in the active item's neighborhood*.

The main motivation behind this approach is to overcome some of the problems with scalability and computational performance that are symptomatic for user based CF.

Similar to user based CF the item based CF technique can be defined as predicting the missing values in a user item rating matrix. In [Sarwar et al., 2001] it is proposed that the relations between columns in a user item rating matrix should be considered. Thus, while the problem space is still that of predicting the missing values in a user item rating matrix as defined in [Herlocker et al., 1999], relations between items instead of users is now considered. For example the user item rating matrix given in [Figure 1.2] shows that Giles hasn't seen Kill Bill.

When predicting the missing rating on Kill Bill for Giles in this case, the prediction will be based on what Giles thinks of movies that are similar to Kill Bill. The similarity between movies is found by matching the column vector of Kill Bill against the column vector of the other three movies. The similarity between items is essentially based on the heuristic that *a person's opinion on items that are similar doesn't vary much*. As such, since ratings on Kill Bill and Land of the Dead varies the least, among users that have rated both movies, they can be considered similar (and I think we can all agree that *both* movies have a lot of *death*, serendipity!). Since Giles thought Land of the Dead was a 4 that might thus be a suitable prediction on Kill Bill for Giles.

	The Thing	Grudge	Andrey Rublyvov	Fargo	Magnolia
Giles	5	5	1	2	1
Spike	1	2	1	2	2

Figure 1.3: User item rating matrix consisting of two users. Note that when Giles rates something highly (5), Spike gives the same item a low rating (1 or 2). But when Spike gives an item a low rating (1 or 2) there’s no pattern for how Giles will rate the same item. The users have a low linear correlation, as such their influence on each other will be low despite the fact Giles might be a good negative predictor for Spike.

### 1.3.2 Similarity measures

Prevalent throughout the research into collaborative filtering has been the study of similarity measures to use to determine user (and item) similarities. The two most popular types of similarity measures are *correlation based* and *cosine based*. In [Resnick et al., 1994] they suggested the use of the *Pearson correlation coefficient* as a correlation based similarity measure. [Breese et al., 1998] compared recommendation techniques using the Pearson correlation coefficient and the cosine angle between vectors as similarity measures, the recommendation technique using the Pearson correlation coefficient was found to perform better. The Ringo system [Shardanand and Maes, 1995] made use of the *mean squared difference* and a *constrained version of the Pearson correlation* to measure the similarity between two users. Their results show that the *constrained Pearson performs better* compared to the Pearson correlation coefficient and the mean squared difference.

In [Billsus and Pazzani, 1998] some important issues regarding (*linear*) correlation-based approaches are discussed, which we refer to as the *reliability*, *generalization* and *transitivity* issues.

**Reliability** The correlation between two users is calculated only over items both have rated. Since users can choose among thousands of items to rate, it is likely that the overlap between most users will be small. This has the effect that the use of the correlation coefficient between two users as a similarity measure is in many cases unreliable. For example, correlations based on three items has as much influence in the final prediction as a correlation based on thirty items. It is thus important to somehow take into account the number of items involved in the correlation, a popular solution to this is the usage of significance weighting [Herlocker et al., 1999].

**Generalization** The correlation based methods typically uses a generalized global model of similarity between users, rather than separate specific models for classes of ratings. For example, referring to [Figure 1.3] where a rating scale of 1 to 5 is used, we see that the two users agree on two movies and disagree on the remaining movies.

The correlation between the two users is near zero, so they would have little influence on each other in the recommendation process. It would though be conceivable to interpret *positive* ratings of Giles as perfect *negatives* for Spike. This fact, and many similar such cases, is however completely ignored with the correlation approach as it uses a model that is too generalized.

**Transitivity** Since two users can only be regarded as similar if they have items that overlap, this has the effect that users with no overlap are regarded as dissimilar in the correlation approach. However, just because two users have no overlap, doesn’t necessarily mean that they are not like-minded. Can such like-mindedness still be detected? Consider the following example; user *A* correlates with user *B*; user *B* correlates with user *C*; since user *A* has no items in common with user *B* the two do not correlate. But one thing they do have in common is their similarity with user *B*, i.e.  $A \rightarrow B \leftarrow C$ , and this information could be used to establish a similarity between users *A* and *C*.

Different ways of improving and extending primarily the correlation based similarity measures have been developed, some of the more widely known and used are the ones proposed by Breese [Breese et al., 1998] and Herlocker [Herlocker, 2000]. Breese suggests three different extensions, *default voting*, *inverse user frequency* and *case amplification*. Herlocker suggests the use of *significance weighting*.

**Default voting** Since the reliability of the correlation between two users increases with the number of items that are used for calculating the correlation, few co-rated items means that the reliability of

the similarity is questionable. Instead of only using co-rated items, Breese suggests that the union over all items that two users have rated should be used. For each item in the *union* where one of the two users has a missing rating, a default rating is inserted. Breese suggests that the default value should be a neutral rating or a somewhat negative preference for the unrated item. It is important to realize that the basic idea of default voting *only* applies to the union of items rated by both users in question. However, they also suggest that a default value can be added to some additional items that neither one of the two users have rated but that they might have in common.

**Inverse user frequency** The extension is based on the idea that universally liked items are not as useful in capturing similarity as less common ones. Inverse user frequency for an item is calculated by taking the logarithm of the ratio between the total number of users and how many that have rated the item. This is analogous to the inverse document frequency typically used in information retrieval to reduce influence of common words when calculating document vector similarity. The ratings for each item is then multiplied by their calculated inverse user frequency when calculating similarity between users's rating vectors (this would not be suitable for correlation based approaches).

**Case amplification** Refers to a transform applied to the similarity. Positive similarities will be emphasized while negative will be punished. Breese uses a formula where positive similarities are left unmodified, and negative similarities are modified to lie closer to zero.

**Significance weighting** A method that devalues similarities that are based on too few co-rated items. It is based on the observation made by Herlocker that users that have few items in common but are highly correlated, usually turns out to be terrible predictors for each other. His experiments show that a minimum of 50 co-rated items is typically required for producing more accurate predictions, imposing a higher value on the minimum number of co-rated items doesn't have any significant effect on the accuracy. If two users have 50 or more items in common, their similarity is left unmodified, but if they have less than 50 co-rated items, the similarity is multiplied with the number of co-rated items divide by 50.

In particular significance weighting is also a useful extension for cosine based similarity measures. In [Sarwar et al., 2001] the *adjusted cosine similarity measure* and significance weighting is used to determine item similarities based on user ratings, they report better results when using the adjusted cosine than when using the person correlation coefficient as a similarity measure.

### Pearson's correlation coefficient

[Resnick et al., 1994] suggested (by use of an example) the usage of the Pearson correlation coefficient as a similarity measure between users item ratings.

By studying how users ratings are distributed in a scatter plot it is determined whether a positive or negative linear correlation exists between the two users ratings [Figure 1.4]. As with any correlation based similarity measure only co-rated items can be considered (unless an extension such as default voting is also used).

The correlation between two users  $u$  and  $n$  is calculated using the two user's respective row in the user item rating matrix (as previously described), but ignoring columns for items both users haven't rated. Since it is practical to not have to deal with missing values in the matrix, we will adopt a variation of the user item rating matrix model (in the next chapter a formal definition of this model will be given) which associates a *set* of ratings with each user. Let;  $r_{u,i}$  denote the *rating* on item  $i$  by user  $u$ ;  $u_r$  denote the set of ratings given by user  $u$ . The similarity between users  $u$  and  $n$  can then be defined by their Pearson correlation as follows:

$$similarity(u, n) = \frac{\sum_{i \in I} (r_{u,i} - \bar{u}_r) \times \sum_{i \in I} (r_{n,i} - \bar{n}_r)}{\sqrt{\sum_{i \in I} (r_{u,i} - \bar{u}_r)^2 \times \sum_{i \in I} (r_{n,i} - \bar{n}_r)^2}} \quad (1.2)$$

where  $I$  is the set of co-rated items, i.e.  $I = \{i | r_{u,i} \in u_r \wedge r_{n,i} \in n_r\}$ .

Note that the mean ratings  $\bar{u}_r$  and  $\bar{n}_r$  for each user should be calculated over the set of co-rated items since the correlation formula only deals with co-rated items. Variations of the formula exist, particularly one form that does not require the calculation of user means is often used. The similarities defined

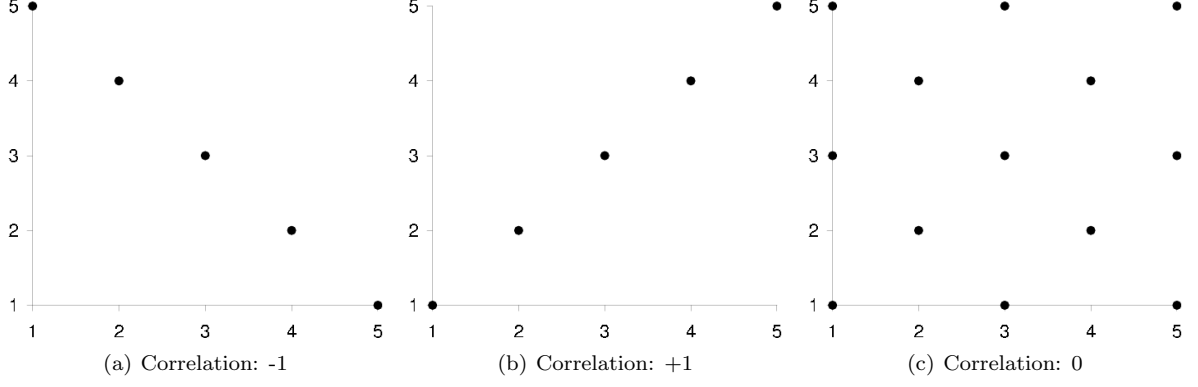


Figure 1.4: Scatter plots showing two users  $u$  (x-axis) and  $n$  (y-axis) correlation when assigning ratings on a discrete rating scale 1 – 5 to items. A  $-1$  correlation can be interpreted as whenever user  $u$  rates something high user  $n$  rates it low, or whenever  $u$  rates something low  $n$  rates it high. A  $+1$  correlation can be interpreted as whenever user  $u$  rates something high user  $n$  rates it high, or whenever  $u$  rates something low  $n$  rates it low. A  $0$  correlation simply means that whenever  $u$  rates something low,  $n$  might rate it low too or just as likely rate it high, no linear relationship exists in the scatter plot.

in this manner will lie in the interval  $[-1, 1]$  where similarities near  $-1$  mean negative correlation and similarities near  $+1$  mean positive correlation. Values around zero indicate that no *linear* correlation existed between the two users, this doesn't necessarily mean the two users have dissimilar taste, however in lack of further analysis we must assume the users are dissimilar.

While the application of the Pearson correlation coefficient has been in terms of determining similarity between two users, a similar reasoning applies to finding the similarity between two items. The correlation between two items is calculated using the two items's respective column in the user item rating matrix, but ignoring rows for users that haven't rated both items. To denote the set of ratings given to an item  $i$  the notation  $i_r$  is used. The similarity between items  $i$  and  $j$  can be defined by their Pearson correlation as follows:

$$similarity(i, j) = \frac{\sum_{u \in U} (r_{u,i} - \bar{i}_r) \times \sum_{u \in U} (r_{u,j} - \bar{j}_r)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{i}_r)^2 \times \sum_{u \in U} (r_{u,j} - \bar{j}_r)^2}} \quad (1.3)$$

where  $U$  is the set of co-rated items, i.e.  $U = \{u | r_{u,i} \in i_r \wedge r_{u,j} \in j_r\}$ .

As can be seen the notation is the same with users substituted for items.

### Cosine angle

In the user rating matrix terminology we can view each user's row in the matrix as a rating vector, and use the cosine angle as a similarity measure between two users rating vectors. The cosine angle is measured in an  $N$ -dimensional space where  $N$  is the number of co-rated items between the two users.

Using the previously adopted terminology the similarity between users  $u$  and  $n$  can be defined by the cosine angle between their ratings vectors as follows:

$$similarity(u, n) = \frac{\vec{u} \cdot \vec{n}}{\|\vec{u}\| \times \|\vec{n}\|} = \frac{\sum_{i \in I} r_{u,i} \times r_{n,i}}{\sqrt{\sum_{i \in I} r_{u,i}^2 \times \sum_{i \in I} r_{n,i}^2}} \quad (1.4)$$

The similarities defined in this manner will lie in the interval  $[0, 1]$ , where zero means no similarity and one strong similarity.

### Adjusted Cosine

When finding the similarity between two *items* based on users ratings on the items, [Sarwar et al., 2001] found that using an adjusted version of the cosine angle when calculating similarities produced better

results (the resulting predictions relying on the calculated similarities were better in terms of some evaluation metric) than using the cosine angle and the person correlation coefficient in the similarity calculation.

The similarity calculation based on adjusted cosine unlike basic cosine takes into consideration that users use different *personal* rating scales. The similarity between items  $i$  and  $n$  is now defined as:

$$\text{similarity}(i, j) = \frac{\sum_{u \in U} (r_{u,i} - \bar{u}_r) \times (r_{u,j} - \bar{u}_r)}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{u}_r)^2 \times \sum_{u \in U} (r_{u,j} - \bar{u}_r)^2}} \quad (1.5)$$

### Mean squared difference

Used most notably in the Ringo system. Let  $|I|$  denote the number of co-rated items for user  $u$  and  $n$ . The similarity between users  $u$  and  $n$  is defined as:

$$\text{similarity}(u, n) = \frac{1}{|I|} \sum (r_{u,i} - r_{n,i})^2 \quad (1.6)$$

### 1.3.3 Recommendations

Equally prevalent in CF research as similarity measures, is research into how to use the similarities to make actual recommendations. Two distinct types of recommendations are considered to exist, *predictions* and *ranking lists* (top  $N$  lists). A prediction is a value indicating to a user some kind of preference for an item, while a ranking list consists items the user will like the most sorted in descending order of preference, typically only the top  $N$  items in the ranking list are shown.

#### Prediction algorithms

A *prediction* is a numerical value  $p_{u,i}$  that express the preference (interest, "how much something will be liked") of item  $i$  for a user  $u$ . The preference is usually encoded as a numeric value within some discrete or continuous scale, i.e.  $[1, 5]$  where 1 expresses dislike and 5 a strong liking. Several different prediction algorithms have been proposed in the literature. The most commonly used is [Equation 1.9], commonly known as *weighted deviation from mean*, which was first proposed in the GroupLens System [Resnick et al., 1994].

Prediction algorithms rely on a set of neighbors  $N$  that consists of users that for simplicity in this discussion will be assumed to have rated the active item  $i$ . Three common ways of computing predictions are then:

$$p_{u,i} = \frac{1}{|N|} \sum_{n \in N} r_{n,i} \quad (1.7)$$

$$p_{u,i} = k \times \sum_{n \in N} w_{u,n} \times r_{n,i} \quad (1.8)$$

$$p_{u,i} = \bar{u}_r + k \times \sum_{n \in N} w_{u,n} \times r_{n,i} \quad (1.9)$$

The constant  $k$ , sometimes called the "normalizing constant" is usually taken to be:

$$k = \frac{1}{\sum_{n \in N} |w_{u,n}|} \quad (1.10)$$

The normalizing constant doesn't cause the predictions to lie within the original rating scale the predictors rated items on, however it will make them lie "close enough". Predictions can be put into the original rating scale by clamping values (e.g. clamp all prediction above 5 to 5). In a sense the predictions are relative each other, a high predicted value indicates a stronger preference than a lower predicted value, however since predictions are often viewed "on their own" it does become necessary to clamp the predictions to a scale the user can understand.

Note the usage of  $w_{u,n}$  where  $w$  stands for weight. The first prediction algorithm uses no such weight, it simply calculates the whole population's average rating for the active item  $i$ . The weight is introduced



in the second prediction algorithm to discern between levels of user similarity, this weight is the similarity between the active user and each neighbor contributing to the prediction, which in this case means all users in the population that have rated the item. (Weights are always taken as some form of similarity between users, the term weight stems from the recommender systems history in information retrieval and filtering and will not be used beyond this chapter, instead the more obvious term similarity will be used.)

The first prediction algorithm [Equation 1.7] expresses the simplest case, the predicted value is just the average value for that item given by all users (because of this the name *Item mean* is sometimes used to refer to this prediction algorithm). This algorithm is often used as a baseline predictor when new prediction algorithms are tested.

The most common aggregation formula used is [Equation 1.8] (which can be referred to as the *Weighted average rating* prediction algorithm), it weighs all user with respect to its similarity to the active user. The purpose of the weighting is to ensure that like-minded users for the active user will have more to say in the final prediction, this algorithm was used in for example the Ringo system [Shardanand and Maes, 1995].

However, it has been found that users use different rating scales when rating items, e.g. some users will be very restrictive about giving items top ratings, while other users may be more generous or have widely different definitions of what a top rating means and and rate many items they like with top ratings (perhaps two users have the opinions "great movie, I give it a two" and "terrible movie, I give it a two" about the same movie, which isn't unusual as the authors can testify). This means that the rating distribution between users will be skewed. Letting a user that rates many movies with the top rating predict for a user that is very restrictive about top grades may have a negative effect on prediction quality. In order to neutralize the negative effect of the skewed rating scales, [Equation 1.9] was introduced. The very basic idea is to start out with the active users mean rating, and then look at how the active user's neighbors deviate from their mean rating (because of this the prediction algorithm is often referred to as *Weighted deviation from mean*). If the neighbors do not deviate from their mean rating, then neither should the active user. This prediction algorithm was used first in the GroupLens System [Resnick et al., 1994].

Common prediction algorithms used by item based CF have similarities with the previously presented prediction algorithms, however with some important differences. The neighborhood set  $N$  consists of items similar to the active item  $i$ . For discussion it will be assumed that  $N$  consists of all items similar to the active item.

A algorithm in the item based case that is similar to [Equation 1.7] would simply take the active users average rating on all items he has rated. However as that makes little sense, the items' degree of similarity is also accounted for.

A common algorithm for making predictions in item based CF is [Equation 1.11] (often referred to as *Weighted sum*) (with e.g.  $k = \frac{1}{|N|}$ ).

$$p_{u,i} = k \sum_{j \in N} w_{i,j} \times r_{u,j} \quad (1.11)$$

## Ranking algorithms

A ranking list, or more commonly a *top N* list since the ranking list typically consists of the  $N$  items (out of all available in the dataset) that the active user will like the most sorted in descending order of preference. Top  $N$  lists can be created both by simply sorting predictions, or by using algorithms specifically designed for top  $N$  list generation. Any recommendation technique making predictions can also sort the predictions to generate top  $N$  lists. Thus the following methods for generating top  $N$  lists are usually created when the underlying data is not ratings, rather transaction data, e.g. information on what items a user has purchased. However, there is nothing to prevent using rating, however it will be necessary for some parts of the algorithm to reduce the rating data to unary data, this means losing information which might lead to worse results had predictions been sorted.

Two ways of producing top  $N$  ranking lists, when the underlying data is transaction data, are given in [Karypis, 2001] and are outlined below.

**Frequent item recommendations** This method creates a top  $N$  ranking list by first creating a set of items consisting of all items neighbors of the active user has *purchased*. Items that the active user has purchased are excluded from the set. The frequency of each item in the set is counted, i.e. it is

counted how many of the neighbors purchased each item. The items are then sorted in descending order of frequency. The  $N$  most frequent items are then presented to the active user as a top  $N$  list.

**Similar item recommendations** This method creates a top  $N$  list by first creating a candidate set of the  $k$  most similar items for each of the items in the active users profile. The set consists only of distinct items and items already purchased by the active user are removed. For each item in the candidate set, its *similarity* to each of the items in the active users profile is computed and summed up. The  $N$  items in the candidate set are then sorted with respect to their summed similarity to the active users profile. The  $N$  items with the highest summed similarity are then presented to the active user as a top  $N$  list.

In [Deshpande and Karypis, 2004] an interesting variation on their *similar item recommendations* method is proposed. Instead of only finding the  $k$  most similar items for each item in the active users profile, the  $k$  most similar items to all possible sets of items in the active user's profile are found. For example, suppose the active user has three items  $i_1$ ,  $i_2$  and  $i_3$  in this profile, then the candidate set should include the  $k$  most similar items for not only items  $i_1$ ,  $i_2$  and  $i_3$ , but also for the sets  $\{i_1, i_2\}$ ,  $\{i_1, i_3\}$ ,  $\{i_2, i_3\}$  and  $\{i_1, i_2, i_3\}$ . This requires a similarity measure that is able to determine similarity between one item and a set of items. They claim that this approach will discover items that can be of potential interest to the active user that otherwise wouldn't be found, i.e. the  $k$  most similar items to e.g. the set  $\{i_1, i_2\}$  might include items that are not in either of the individual items list of  $k$  most similar items. Techniques implementing this method of generating top  $N$  list require very large models of the  $k$  most similar items to be build, to reduce model size only item sets that occur frequently among users can be used. The notation *frequent item sets* is used by the data mining community in connection with association rules as defined in [Agrawal et al., 1993].

## 1.4 Content Based Filtering

Information filtering and *content based filtering* refers to the same paradigm, filtering items by their content. However information filtering is most often used in connection with text documents, while content based filtering allows for a wider definition of content. Pure content based filtering, as defined in [Balabanovic and Shoham, 1997], is when recommendations are made for a user based only on a profile built up by analyzing the content of items that the user has rated or otherwise shown interest in. We will be using the term content based filtering since we will not constrain our self to text documents only, that is, we will use any content that after some preprocessing can be used in the filtering process. From the definition we can identify two important steps in content based filtering, the need for both a technique to represent the items and for a technique to create the user profiles.

Many of the techniques that have been developed and used in text retrieval and text filtering have been adopted by the content based approach [Oard, 1997]. One of the more popular techniques is the vector space model and especially the *tf-idf* weighting scheme for representing the content of the text documents. Popular approaches for creating user profiles are techniques found in the machine learning field. One such technique is the use of a classification algorithm that can learn a user profile that it can use to distinguish between items that are liked and disliked by the user.

So not surprisingly, recommender systems that use the content based approach are often found in domains where the content has some similarity to plain text documents. For example *Syskill & Webert* [Pazzani et al., 1996] is a content based recommender system for webpages. Users rate webpages on a three point scale, the system can then recommend new WebPages or create a search query for the user to be used in a search engine. Users of the system are presented with an index page that links to several manually created topic webpages that contains hundreds of links to pages within that topic. The system creates user profiles for each topic rather than user profiles for all topics combined. Based on the user profiles, some of the links on the topic pages will be associated with a symbol (e.g. thumbs up, thumbs down, smileys, etc.) that symbolizes how much the user will like the page that is linked to.

In the Syskill & Webert recommender system each webpage is represented as a boolean feature vector consisting of the  $k$  most informative words for the topic it belongs to. The boolean values for each feature indicates if a word is present or absent in the web page. They chose  $k$  to be 128. Words to use as features

in the feature vectors differ for each topic, but are the same for all webpage's belonging to the same topic. The words within each topic are chosen based on how much expected information gain the presence or absence of the word will give when trying to classify a webpage for a user. A learning algorithm is used to classify unrated webpage's for users, the learning algorithm analyzes the content of all the pages a user has rated and by this process learns the user's profile. Five different types of learning algorithms were tested by Pazzani et. al, Bayesian classifiers, nearest neighbor algorithms, decision trees, neural nets and tf-idf as described in the vector space model. The algorithms show different results over different topics, but the overall best performance is given by the Bayesian classifier. All algorithms show results that almost always beats the baseline algorithm, which is just guessing what pages a user would like.

Content based filtering systems based on *item attributes* differs in some aspects from keyword based systems like Syskill & Webert. Extracting product content is a straight forward process, the attributes of the product is usually well structured, such as for movies where the content consists of attributes like the movie's director, lead actress/actresses, genre etc. However, the challenge is to select only the most important attributes. The key issue here is: *what defines an attribute as important?* Since importance differs from product to product, no standardized ways, like for example the tf-idf weighting scheme used with text documents in the vector space model, exists for product attributes (that we are aware of). Choosing the wrong attributes can have a great impact on the performance of the recommendation technique, so the choice of attributes is usually done manually either by some domain experts or by empirical tests.

In [Melville et al., 2002] they represent the content of a movie as a set of "slots" containing attributes of the following type: movie title, director, cast, genre, plot summary, plot keywords, user comments, external reviews, newsgroup reviews and awards. A movie will in this case be represented as a vector of *bags of words*. A naive Bayesian classifier is then used to create user profiles. They report almost equal performance between their implementation of a user based collaborative filtering technique and the content based filtering technique.

A way of making movie recommendations with a minimum of knowledge about the users is described in [Fleischman and Hovy, 2003]. Instead of representing and learning user profiles from the users preferences about movies, the recommender system only uses one movie as a seed movie to base its recommendations on. For example, suppose the active user has a favorite movie and wants to find out if there exists any similar movies. By using the favorite movie as a seed movie, the recommender system can make some recommendations that are based on the similarity with the seed movie. They propose two different techniques for this and the only difference between them is how the content of the movies is represented. In the first technique, they only use the plot summaries of the movies and represent the movies with a binary vector, where 1 and 0 represents the presence and absence of a word respectively. The size of the vector is the same as the total size of the used vocabulary. Two movies are found similar by calculating their cosine similarity as described in the vector space model. The second technique also uses the cosine similarity for finding similar movies, but the representation of the movies is different. In this case, each movie is represented by a vector that contains the movies cosine similarities to different movie genres. Each genre is represented as a vector of keywords that are weighted by how indicative they are of the particular genre. The keywords were taken from the Internet Movie database (IMDb)<sup>3</sup>. Thus, the first technique recommends movies based on how well two movie's plot summaries match, and the second one recommends movies from their shared similarity and dissimilarity to particular movie genres. Their results shows that no conclusions can be made whatsoever about the performance of their techniques, for example, they compared the top five recommendations from their techniques against the output from IMDb on one particular movie and none of them recommended the same movies.

## Vector Space Model

The vector space model is described in [Salton and McGill, 1983] as a vector matching operation for retrieving documents from a large text collection. The user formulates a search query consisting of words that best describes his information need and this query is then matched against all documents in the text collection. The matching operation uses a similarity formula to determine which documents best matches the search query and then retrieves those documents for the user.

In this model, a document (and a search query) is represented by a vector of weighted keywords (terms) extracted from the documents [Salton and Buckley, 1988]. The model is also popularly called

---

<sup>3</sup><http://www.imdb.com/>

	$t_1$	$t_2$	$\dots$	$t_j$
$d_1$	$w_{11}$	$w_{12}$	$\dots$	$w_{1j}$
$d_2$	$w_{21}$	$w_{22}$	$\dots$	$w_{2j}$
$\vdots$	$\vdots$	$\vdots$	$\dots$	$\vdots$
$d_i$	$w_{i1}$	$w_{i2}$	$\dots$	$w_{ij}$

Figure 1.5: Term document matrix. The rows represent documents and the columns terms that appear in the document collection, but not necessarily in all documents. The cells contains weights indicating the importance of each term in each document.

the *bag of words* representation since the words are stored in a unordered structure that does not retain any term relation. For example, terms appearing together in a document can have some special meaning, but that will be lost by this kind of representation. The weights represent the importance of the keywords in the document as well as in the entire collection of documents. Since each document vector is represented in a vector space that is spanned by the number of terms, this model can be viewed as a term document matrix [Figure 1.5], where rows are documents and the columns are all (or some of) the terms that appear in the document collection.

If we use  $w_{i,j}$  to denote the weight for term  $j$  in document  $i$ , the document vector can be expressed as:

$$d_i = \langle w_{i1}, w_{i2}, \dots, w_{ij} \rangle \quad (1.12)$$

This model has been used for a long time as a basis for successful algorithms for ranking, filtering and clustering documents, see for example [Baeza-Yates and Ribeiro-Neto, 1999].

Three factors that must be considered when calculating the weights are *term frequency*, *inverse document frequency* and *normalization*. After all the weights have been calculated, similarity between two documents  $d_1$  and  $d_2$  can be calculated using for example the cosine angle:

$$\text{similarity}(d_1, d_2) = \frac{\sum_j w_{d_1,j} \times w_{d_2,j}}{\sqrt{\sum_j w_{d_1,j}^2 \times \sum_j w_{d_2,j}^2}} \quad (1.13)$$

The *term frequency* is just a count of how many times a term occurs in a document. If a term occurs many times in a document, then it is considered important for that particular document. The term frequency  $f$  for term  $j$  in document  $i$ , i.e. the number of occurrences of term  $j$  in document  $i$ , is denoted as  $f_{i,j}$ . This is sometimes referred to as the terms *local frequency*.

The *inverse document frequency* (IDF) identifies terms that occurs only in a few documents. Such terms are considered to be of more importance than terms that are prevalent in the whole document collection. Terms occurring in the whole document collection make it hard to distinguish between documents and if used causes all documents to be retrieved which naturally affects the search precision. The IDF factor is usually referred to as the terms *global frequency*. The IDF factor for a term is inversely proportional to the number of documents that the term occurs in, since when the document frequency for a term increases, its importance decreases. The frequency  $f$  for term  $j$  in document  $d$  is denoted as  $df_j$  and the inverse document frequency for term  $k$  in document  $d$  is denoted as  $idf_j$ . The inverse document frequency  $idf_j$  is inversely proportional to the document frequency  $df_j$ , i.e.:

$$idf_j : \frac{1}{df_j} \quad (1.14)$$

and varies with the document frequency  $df_j$ . The  $idf_j$  for a term  $j$  is usually taken as the logarithm of the ratio between the total number of documents  $D$  and the document frequency, i.e.:

$$\frac{D}{df_j} \quad (1.15)$$

The logarithm (which can be taken to any convenient base) is just there for smoothing out large weight values. Thus we get:

$$idf_j = \log \left( \frac{D}{df_j} \right) \quad (1.16)$$

The term frequency  $f_{i,j}$  and the inverse document frequency  $idf_j$  can be combined as  $f_{i,j} \times idf_{i,j}$  to obtain a weight that accounts both for terms occurrences within a document and within the whole document collection. However, it is necessary to perform some normalization in order to avoid problems related to keyword spamming and bias towards longer document. Two types of normalization that is suggested is *normalized term frequencies* and *normalized document lengths*.

Since higher term frequencies  $f_{i,j}$  results in a larger weight value than for terms with a lesser frequency models are vulnerable to keyword spamming, i.e. a technique where terms are intentionally repeated in a document for the purpose of generating larger weight values and by that way improving the chances of the document being retrieved. One way to avoid this is to calculate the *normalized term frequency*. The normalized term frequency of term  $j$  in document  $i$  is denoted  $tf_{i,j}$  and is usually calculated by dividing the frequency of the term  $j$  with the maximum term frequency in document  $i$  denoted  $\max(f_i)$ , i.e.:

$$tf_{i,j} = \frac{f_{i,j}}{\max(f_j)} \quad (1.17)$$

*Normalized document lengths* are used since longer documents usually use the same term more often, resulting in larger term frequency factors, and using more different terms as opposite to shorter documents. This means that longer documents will have some advantages over shorter ones when it comes to calculating the similarity between documents. To compensate for this *cosine normalization* can be used as a way of imposing penalty on longer documents, it is computed by dividing every document vector by its Euclidian length. The Euclidian length of a document  $i$  is defined as:

$$dl_i = \sqrt{w_1^2 + w_2^2 + \dots + w_j^2} \quad (1.18)$$

Terms that should receive the highest weight are those that have high term frequency and low over-all document collection frequencies. Salton [Salton and Buckley, 1988] refers to this property as *term discrimination* since this suggest that the best terms are those that can be used to distinguish certain documents from the remainder of the collection. This is achieved by multiplying the normalized term frequency with its inverse document frequency  $tf_{i,j} \times idf_{i,j}$ , and then normalizing it to achieve the final weight for the term  $j$  in document  $i$ :

$$w_{i,j} = \frac{tf_{i,j} \times idf_{i,j}}{dl_i} \quad (1.19)$$

## 1.5 Hybrid Filters

Both collaborative filtering and content based filtering have their strengths and weaknesses, hybrid filters are an attempt to combine the strengths of both techniques in order to minimize their respective weaknesses. There are many examples of recommender systems based on hybrid filters.

**FAB** A content based collaborative filtering system for webpage's [Balabanovic and Shoham, 1997]. In FAB webpage's are represented by the words that receives the highest tf-idf weight and the user profiles are represented by an tf-idf vector that is the average of the webpage's that are highly rated by the user. The user can then receive recommendations from similar users using the collaborative approach or by direct comparison between a webpage and a user's profile. Their results show that the FAB system makes better recommendations compared to recommendations that are randomly selected, human selected "cool sites of the day" or pages best matching an average of all user profiles in the system.

**Syskill & Webert** Pazzini [Pazzani, 1999] extended the content based technique that was used in his Syskill & Webert [Pazzani et al., 1996] recommender system with collaborative filtering techniques. He refers to this new technique as *collaboration via content*. Webpage's are represented by the 128 most informative keywords for the class they belong to. A content based approach different from the one presented in [Pazzani et al., 1996] is used both on its own and in combination with a

collaborative filtering technique. In the content based approach a user profile is learned by first applying an algorithm that assigns weight values to each informative word in a webpage a user has rated. Weights are initially set to 1, the weight values for each webpage are then summed up and if the sum is above a threshold value and the user liked the webpage, the weights are doubled, otherwise they are divided by two. Training stops after 10 iterations over all pages or when each rated webpage is correctly classified (i.e. weights for words on all positively rated pages sum over the threshold, and weights for words on all negatively rated pages have a sum under the threshold). Webpage's are then recommended by applying the user's weight values on the words that occurs in each webpage and summing up the total weight for each webpage, the webpage's with the highest sum are recommended to the user. In the collaboration via content technique the content based user profiles, consisting of the user's word weights, are used instead of user profiles consisting of ratings, in the same manner as in standard collaborative filtering. Their experimental results show that when users have very few webpage's in common, collaboration via content has a better precision than standard collaborative filtering. But as soon as users get more and more webpage's in common, collaborative filtering performs almost equally well. Precision was measured based on the top three webpage's that was recommended to a user.

**P-TANGO** [Claypool et al., 1999] is a recommender system for news articles from an online newspaper that uses both the content based approach and the collaborative approach in the recommendation process. It's not a hybrid system in the sense that the two approaches depend on each other, instead they are implemented as two separate modules that are independent of each other. The recommendations from the two approaches are instead weighted and combined to form a final recommendation. One advantages over hybrid systems where both approaches are tightly coupled and deeply dependent on each other, is that the system can choose which approach should have more confidence during different circumstances. For example, when few people have accessed a news article, the content based recommendation is weighted higher but as soon as more and more people is showing interest in the article, the recommendations based on the collaborative approach take over. Also, improvements of the system is easier, since whenever someone comes up with a new and improved way of making either collaborative or content based recommendations, these are easily incorporated into the system. The user profiles in the content based approach consists of keywords that are matched against item profiles that consists of extracted words from news articles. The keywords in the user profiles are either taken from articles the users have rated highly or are keywords the users have marked as important. Their experimental results show that the combined recommendations are a bit more accurate than either of the separate ones.

**Rule based using Ripper** In [Basu et al., 1998] they treated the task of recommending movies to users as a classification problem. They saw the recommendation process as the problem of learning a function  $f(<user, movie>)?\{liked, disliked\}$  that takes as input a user and a movie and produces an output that tells whether the user will like or dislike the movie. For this task, they chose to use an inductive learning system called Ripper. Ripper is able to learn rules from set-valued attributes, in this case by taking as input a tuple  $<user, movie>$  consisting of two set-valued attributes user and movie. They created tuples consisting of what they call collaborative features (user's ratings on movies) and content features (information available on movies, they used over 30 different attributes). Using for example collaborative features the  $\langle user, movie \rangle$  tuple would contain for the user attribute which movies the user likes, and for the movie attribute which users that likes the movie (e.g.  $<\{movie_1, movie_2\}, \{user_1, user_2, user_3\}>$  where  $movie_1$  and  $movie_2$  are two movies liked by the user in question, and  $user_1$ ,  $user_2$  and  $user_3$  are three other users that like the movie in question). They also used what they call hybrid features, e.g. a movie was represented using users that like a particular genre. Each such  $<user, movie>$  tuple is then labeled by whether or not the *movie* was liked by the user. Given such tuples the Ripper can be trained to create rules for every user, which can be used in the recommendation process. Example of an interpretation of a rule for a user that like action movies can be "*if action is present among movies that the user likes then predict user likes movie*". They report that their approach performs reasonably well compared to the collaborative approach on the same data. However, this is only achieved with the combination of a great deal of manual work in selecting and refining the attributes and at a low level of recall.

**CF with content based filterbots** Sarwar et. al uses in [Sarwar et al., 1998] what they refer to as filterbots, software agents whose only purpose is to give content based ratings on new items that arrives into the system. Since the filterbots are used as recommenders for those users that correlate with them, new items will be available for recommendation much faster then if they had to wait for real users to come along and rate them. They tested their filterbots in the GroupLens system that recommends articles from a Usenet news server. One of the filterbots they implemented was a SpellCheckerBot, whose sole purpose was to run a spell checking algorithm on every new article that arrived to the Usenet news server. The final rating on the article was then decided based on the number of misspelled words. Other filterbots that they tried based their ratings on the length of the article and number of quotes in the articles. They report that a simple filterbot like the spell checker can improve both the coverage (number of items available for recommendation among all items) and the accuracy of the given recommendations. Their results however show that filterbots are sensitive to which domain they operate over, e.g. some did well in newsgroups about humor but some did not etc.

**Content-boosted CF** In [Melville et al., 2002] they implement two hybrid systems, the first one combined the outcome from a pure content based filtering technique with the outcome from a user based collaborative filtering technique by taking the average of the ratings that are generated. Their other system is called content-boosted collaborative filtering (CBCF). This system first creates pseudo user-ratings vectors for every user by letting a content based filter (as previously described) predict ratings for every unrated item. These pseudo user-ratings vectors are then put together to give a dense pseudo-user ratings matrix. Collaborative filtering is then performed by the use of this matrix. Similarity between two users are calculated based on the pseudo ratings matrix, in the prediction process two additional weights are added; the calculated accuracy of the pseudo user-ratings vector for the active user and a self-weight that can give extra importance to the active users pseudo self, since he is used as a neighbor. Their results shows that the CBCF performs best, followed by pure CF, combined CF and CB and last pure CB. Although, the difference between the CBCF and the pure CB is very small.

**Combining attribute and rating similarity** Two different recommendation techniques for movies is presented in [Jin and Mobasher, 2003] that both use a combination of item similarity in their implementation of a item based collaborative filtering technique. Similarity between two movies is first calculated based on ratings given to them by users and then based on their content (they refer to content similarity as semantic similarity). These two similarities are then combined into one single similarity score. Rating similarity between two movies is calculated using adjusted cosine. Content similarity is calculated as a linear combination of the similarity between the movies attribute. The content of a movie includes the following attributes; title, year, director, genre, cast, MPAA rating and plot summary. After preprocessing of the content data, the movie is represented as a vector of attributes, where each attribute is represented in some suitable way, e.g. as a bag-of-words in the case of plot summary. Content similarity between two movies, denoted  $ConSim(i,j)$  where  $i$  and  $j$  are two movies, are calculated using their terminology as  $Consim(i,j) = a_1 \times GenreSim(i,j) + a_2 \times CastSim(i,j) + a_3 \times DirectorSim(i,j) + \dots$  where  $a_1, a_2, a_3, \dots$  are predefined parameters that are used as weights. Each attribute similarity is calculated with some suitable method. The final similarity score between two items  $i$  and  $j$  is then calculated as  $TotalSim(i,j) = \alpha \times ConSim(i,j) + (1-\alpha) \times Ratesim(i,j)$  where  $RateSim(i,j)$  denotes the rating similarity between two movies and  $\alpha$  is a predefined parameter used as a weight. Note that with  $\alpha = 0$  this becomes a pure item based collaborative filtering technique. The combined similarity score is used as a weight in the weighted sum formula for producing predictions for the active user (as previously described). This is also how their first technique works. Their second technique, differs in that they first uses the combined similarity score to predict ratings for unrated items to decrease the sparsity in the rating matrix. For each unrated item they find all its similar items that are above some predefined similarity threshold and uses the weighted sum of these similar items ratings as the estimated rating for the unrated item. Unlike [Melville et al., 2002] they don't create a fully dense matrix, they only choose to rate some of the unrated items. This new matrix is then used as in their first technique. Their result shows that there is no significant difference in using a combination of rating and content similarity from using only rating or semantic similarity. On the other hand, this means that content similarity can be used in cases when there are to few ratings

for calculation of rating similarity. (Note: they do claim that their techniques improve accuracy of the predictions, but that claim comes from observations of the third and fourth (!) decimal in their calculations of the mean absolute error. You just cant take such a claim seriously, the mean absolute error is explained in detail in the next chapter as well as a discussion given on why such differences in the thrid and foruth decmial does not matter to the end user.)

Unfortunately, a direct comparison of the results between theses techniques is nor feasible since they all uses different datasets, different sizes of the datasets, different evaluation protocols etc. The need of a standardization for evaluation of recommendation techniques is a major issue and will be discussed further in the next chapter.

## 1.6 Recommender Systems

The environment for a information retrieval system is static in the sense that the available information changes slowly but user needs change frequently. In contrast, information filtering systems are dynamic as the information changes very often (typically there are "streams of information" that need filtering), but user needs are a bit more static [Belkin and Croft, 1992]. It seems therefore natural to build a system that support both of these approaches for handling massive amounts of information, and one such type of systems are recommender systems.

The environment for a recommender system is usually of a highly dynamic nature, both from the information viewpoint and the user viewpoint. Users and items are added and removed often and user's preferences and needs change over time. Additionally the way users interact with the system itself is of a dynamic nature, e.g. searching, browsing, commenting, rating, etc. The domain that a recommender system operates over is usually very high dimensional and sparse, i.e. items can be of magnitudes thousands to millions of which only a very few are known by each individual user.

One of the first articles that uses the term *recommender system* is written by Resnick and Varian [Resnick and Varian, 1997], the term was there used to define a system that uses collaborative filtering to make recommendations to users:

*"In a typical recommender system people provide recommendations as inputs, which the system then aggregates and directs to appropriate recipients"*

The term recommender systems has since then become synonymous with any system that produces recommendations, not only systems based on collaborative filtering. [Schafer et al., 2001] gives a taxonomy from a user perspective for different types of recommender systems used by e-commerce sites, [Burke, 2002] identifies and classifies different hybrid recommender system.

However, as early as 1990 the term "recommender" occurs in the paper "An Algebra for Recommendations: Using Reader Data as a Basis for Measuring Document Proximity" [Karlgrén, 1990] by Karlgrén at Swedish Institute of Computer Science(SICS) where he used the term recommender when he tried to define what he referred to as an "recommender algebra" based on a collaborative approach and later, his paper "Newsgroup Clustering Based On User Behavior - A Recommendation Algebra" [Karlgrén, 1994] was used as a reference in [Resnick et al., 1994], which is one of the earliest recommender systems based on the collaborative approach .

### 1.6.1 Recommender systems in e-commerce

In every domain of items, there are some items that directly appeal to a broad audience, or by some marketing strategy make it to the top charts. This has the effect that niche items or items without financial backup for marketing have trouble reaching out to the broader public. This creates a market where popularity and strong economic interests dictate what the broad audience should like and buy. The Web and in particular recommender systems have changed this, at least to some degree. Recommender systems in e-commerce are not only a way of providing better customer services by giving personalized suggestions and customized ways of searching and browsing the product database, but it also has the effect of increasing sales on obscure items, items that for some reason didn't find a broad audience when they was released. Items without large financial backup and without an immediate broad appeal thus no longer automatically remain unnoticed by the larger public, and as such get a chance to reach their



public, even though it may still be niche items. A theory that describes this is called the long tail, where the term tail refers to obscure items that for some reason can be found in context with more popular items, called the heads. This theory was formulated and presented in an article with the same name in the Wired magazine [Anderson, 2004]. The effect was observed when a fifteen year old book about mountain climbing suddenly reach the top of New York Times bestseller list and stayed there for 14 weeks. What had happened was that Amazon.com's book recommender system had noticed a buying pattern among its customers and suggested that people who liked a newly published book about the same subject would also like this book. People followed up on the recommendation, and the fifteen year old book started to sell like never before. The combination of infinite shelf space with real-time information about buying trends and public opinion resulted in a rising demand for an obscure book. This is something that traditional retailers can't offer and that's why recommender systems have a bright future in the e-commerce business, because this is one of the greatest strengths with recommender systems, the ability to recommend interesting items that most users otherwise might not have found out about.

Several of the early on developed recommender systems found their way into the commercial market. GroupLens founded NetPerception Inc. in 1996, The people behind Ringo founded Agent Inc. in 1995, later changed the name to Firefly and was bought by Microsoft in 1998. One of the biggest e-commerce sites on the WWW, Amazon.com, started to use recommender systems as a part of their site in 1998 [Linden et al., 2003]. Nowadays, almost every e-commerce site use some kind of recommender system, from simple ones that only recommend items according to statistics to complex ones like Amazon.com that uses several different approaches and recommendation techniques.

### 1.6.2 Trust in recommender systems

The success of e-commerce recommender systems depend a lot of trust being achieved from both the user side as well as from the seller's side. A user must feel confident that the recommendations are trustworthy and not just another trick from the seller's side to increase sales without regards to the customers interests, and the seller must trust that the information gathered from the users truly reflects their opinion on the items. In the paper [Lam and Riedl, 2004] this issue is investigated in the movie domain, where the observation had been made that the average rating for new movies was higher than the average rating for the same movie after a significant amount of people had rated it. The author hypothesized that this could be the effect of a so-called shilling attack, i.e. an item is initially given an unfair amount of high ratings (by e.g. the seller) in the hopes that more people will notice it. However, no clear evidence could be found that supported this hypothesis, one explanation they gave was that new movies tend to be initially highly rated because the people who first know about it are users who already like it and are a big fans of it. But one can argue that this is a possibility that must been taken seriously and thought off in the design of a recommender system.

Similarly, rumors had been circulating that a certain recommender systems had been manipulated to always include recommendations for items that otherwise collected dust on the shelves. If a recommender system is caught performing such types of false and greedy recommendations, it will certainly loose all its achieved trust among its users and when the trust is gone among the users, it's gone for a long time.

The focus on understanding and trusting recommender system is something that has gained a lot of research interest the last couple of years, the GroupLens team for example hosted a workshop in conjunction with the 2005 year's Intelligent User Interfaces Conference , where trust was one of the main topics. Some researchers have tried to implement implicit trust into a recommender system on the algorithmic level, hoping this would result in that the explicit trust for the recommendations would increase as a direct result of more accurate recommendations, one such approach is described in the paper Trust in Recommender Systems [O'Donovan and Smyth, 2005] where they uses the collaborative approach for recommendations. To implement trust, the systems keeps track of scores of how well every user is able to recommend items to other users, for every recommendation that match the active users real opinion of an item, the score goes up for the user and for every time there is a mismatch, the score goes down for the user. The idea is that this should reflect real life trust, i.e. if you know someone that is good on recommending movies to you, then you have more trust in this person's recommendations compared to others. They show some significant improvements compared to the user based collaborative filtering approach as described in GroupLens: an open architecture for collaborative filtering of netnews [Resnick et al., 1994] but the main drawback is the calculation of the trust score. The only way of

calculating it, is by letting users recommend items that other user already have an opinion about and then checking who was successful and who was not, this must be done all the time and would probably have an impact on the performance of the system.

In the paper Trust-aware collaborative filtering for recommender systems [Massa and Avesani, 2004] they propose that recommender systems should be extended with what they refer to as trust-awareness: it should be possible to base recommendations only on ratings given by users that the active user trust or are trusted by another trusted user. Trust is collected by letting users rate other users, i.e. if you in some way have interacted with another user, you can give this user a rating based on your personal experience with him. For example, suppose a user have written a review of a movie and that you, based on this review, decides to go watch it. If you experience with the movie was good, then you give the user a good rating. This can then be used in the recommendation process as an extra weight. Trust values can also be propagated through the whole community of users, i.e. if user A trust B and B trust C, then it is possible to infer something about how much A could trust C. Their results shows that by incorporating trust as weights in the recommendation process, the accuracy of the recommendations are increased, this is especially noticeable for users with very few ratings for which recommendations normally are of poor quality.

Other approaches to gain trust has been explored as well, in the paper Beyond Algorithms: An HCI Perspective on Recommender Systems [Swearingen and Sinha, 2001] they let user try out and judge different recommender systems in an attempt to find out which underlying factors beside the algorithm, makes a user willing to accept and explore the given recommendations. One conclusion is that an effective recommender system inspires trust in the system. The trust issue will be discussed further in Part III.

### 1.6.3 Recommendation approaches

Recommender systems takes different approaches in their effort to make recommendations for users. A recommendation approach can be seen as defining *what motivates the recommendations* made by a recommendation system. We distinguish between recommendation approaches and *recommendation techniques*, a technique is simply a very specific outline or implementation of an approach. Techniques based on the same approach will share many common features, especially in what motivates the recommendations. From our research on recommender systems, we have concluded that the following approaches represent the five most commonly used approaches.

#### Manual approach

This is the oldest approach, a critic reviews e.g. a movie and gives his judgement about how good or bad the movie was. Based on this the user can then decide if the movie is worth watching. Works best if the user knows the critic's preferences and thus is able to decide how much he can trust the reviewer. This is kind of a reversed personalization, the recommendations are non-personalized but the recommender himself is personalized.

#### Statistical approach

This approach bases its recommendations on statistical measurements such as the average rating for a item or market basket analysis ("*users who bought this movie also bought these movies*"), and popularity based recommendations such as top selling lists, top rating lists, or top box-office lists for movies etc. The recommendations are non-personalized since they are the same for every one. Due to its simplicity and efficiency this is a very popular method [Schafer et al., 2001].

#### Content based filtering approach

The content based approach tries to recommend items similar to the ones a user is known to like by analyzing the content of the items [Balabanovic and Shoham, 1997]. By letting the user explicitly express his preference about items or by just assuming that items the user has bought or in some other way showed some interest in is liked by the user, the recommender system tries to match the content of those items with items that are new to the user. A user viewing a product page may receive a list of products with similar content (e.g. movies in the same genre and/or by the same director). Personal recommendations can be achieved if user profiles are maintained and updated by the recommender

Hybridization method	Description
Weighted	By using two or more approaches, each approach's recommendations can be weighted against each other on how much they should affect the final recommendation for an item.
Switching	The recommender system switches between different approaches depending on the situation.
Mixed	Recommendations from different approaches are presented at the same time to the user.
Feature combination	User and item data is combined into a single profile.
Cascade	One approach refines the recommendations given from another approach.
Feature augmentation	The output from one approach is used as input feature to another.
Meta-level	The model learned by one approach, is used as input to another approach.

Table 1.1: Hybridization methods.

system. Since recommendations are based on the attributes of an item, the user can get some insight into why a particular item was recommended to him by looking at the recommended item's attributes and compare them to the items in his own profile. Thus, both trust in recommendations and transparency of recommendations can be achieved with the content based approach. A broad perspective to content is taken here which includes pure facts (e.g. actors in a movie) as well as text documents (e.g. a review or description of a movie), keywords ("tags") etc.

### Collaborative filtering approach

With the collaborative approach, the recommender systems tries to recommend items to the active user with the help of other users. Unlike in the statistical approach there is a collaboration between the active user and the larger population, recommendations are thus always personalized for the active user. A classical technique based on this approach is to build a user profile from the ratings the active user has given to some items and match it against all other users to find out their similarity to the active user. This similarity is then used to weight users so the ones with high similarity, i.e. users with a similar taste as the active user, will have more influence in the recommendation process. The most similar users are often referred to as neighbors and the classical technique is sometimes referred to as a neighborhood based technique. Recommendations are given to a user on items the user hasn't yet rated by other user that have rated them. The system will either return predicted values for the unrated items or a ranked list of items which it thinks the user will like. The classical technique is based on the heuristic that users that agreed in the past will probably agree in the future [Resnick et al., 1994]. The user profiles are maintained and updated by the recommender system which allows for personalized recommendations. This approach can also be described as a way of automating the "word of mouth" in a community [Shardanand and Maes, 1995].

### Hybrid approach

Since each approach has its own weaknesses and strengths, combinations of the various approaches have been made in order to use the strengths in one approach to compensate for the weaknesses in the other approach. Burke [Burke, 2002] identifies five different approaches, collaborative, content-based, demographic, utility-based and knowledge-based. Since the demographic approach deals with finding users with a similar demographic profile and to use their opinions about items to influence the recommendations for the active user, we like to think about this as a collaborative approach, similarly for the last two, utility-based and knowledge-based, we use a broader definition of content and how it can be used and therefore classify these two approaches as content-based approaches. Burke also identifies seven different methods that have been used to create hybrid-filters between the five different approaches [Table 1.1], where combinations between the collaborative approach and some of the other approaches using some of these methods are the most common ones.

### 1.6.4 Problems and challenges of the recommendation approaches

The content-based and collaborative approach have their own weaknesses and strengths, a natural way of solving this is to create hybrid filters. The manual and statistical approaches don't suffer from the problems that these approaches have because they provide non-personalized recommendations. We start by identifying some of the issues with collaborative filtering that have been addressed in the literature and then we go on with the content-based approach.

#### The collaborative filtering approach - weaknesses and strengths

**First rater problem** A new user can't receive any recommendations before the system has learned what preferences the user has. One common way of solving this is to ask the user to rate a fix number of items before making any recommendations to the user. This approach is used in the MovieLens system where every new user to the system must rate at least twenty movies before the user can start to use the system and receive recommendations for movies.

**New item problem** A new item can't be recommended until enough people have rated it in a collaborative filtering system. Sarwar et. al solves this in [Sarwar et al., 1998] by the use of filterbots that are programmed software agents. These filterbots rate new items when they arrives to the system by analyzing the content of the items and comparing it to their own rating profile, i.e. in a movie domain some filterbots are specialized in a particular genre and will therefore rate movies that belongs to that genre highly.

**Sparsity** Sparsity deals with the fact that most users only have observed few items or that most of the items only have been observed by a few users. This has consequences when it comes to the coverage, i.e. the recommender system's ability to recommend all items and the accuracy of the given recommendations. With sparse data users will most likely only overlap on a few common items, resulting in recommendations based on questionable data. Several methods have been proposed to deal with this problem. One approach is to add more data into the similarity calculation. This can be done as in [Melville et al., 2002] where they extend the user profiles with content-based ratings. A different approach is to reduce the dimensionality of the rating matrix by the use of singular value decomposition as in [Sarwar et al., 2000b] or Principal Component Analysis as in [Goldberg et al., 2001]. Although, some of these methods manage to produce slightly better recommendations there are some trade-offs, valuable information can get lost by the reduction of data and the adding of extra content to the user profiles means that more information must be gathered, which is not always an easy task.

**Scalability** The main drawback with the collaborative approach is that the computations are very expensive because of the high dimensional spaces of users and items. The memory based techniques requires that the whole user-item database is kept in memory, however when the database of users and items grows this approach fails to scale up. Different solutions to this has been proposed, like the ones in the paper by Breese [Breese et al., 1998] where model-based techniques are used as described earlier. In [Conner and Herlocker., 1999] they experiment with different kinds of clustering algorithms in an attempt to increase scalability and accuracy of predictions. The clustering algorithm is used to make smaller partitions of the item space where each partition contains items that have been found to be similar according to how users have rated them. After the partitions have been found, collaborative filtering is applied to each of the partitions, independent of each other. They report a mixed result for the accuracy of the predictions but they suggest that clustering algorithms do increase the scalability of the collaborative approach. Another technique is proposed in [Sarwar et al., 2001] where they apply singular value decomposition to reduce the dimensionality of the user-item matrix. Although these techniques report improvements on the performance they usually reduce the quality of the recommendations, i.e. *"What you gain in performance, you loose in quality"*. Models also need to be rebuilt to reflect the changes in the user profiles, this is done off-line but it usually takes some time and during that phase, user profiles are not up to date. Some simpler techniques like discarding popular and unpopular items reduces the dimensionality but at the cost that some items will never be recommended. The trade-off between quality and performance is something that must be considered and adjusted to the domain that the recommender system operates over.

**Grey sheep problem** Users with unusual or divergent taste can have trouble in finding users with similar taste to serve as recommenders. It can also be the case that the kind of items these users like have too few ratings to even be considered in the recommendation process. The use of content-based predictions as in [Melville et al., 2002] is one proposed solution to this.

Some of the advantages with the collaborative approach is that it doesn't require any representation of the items, only the rating data, making it suitable for domains where item descriptions are hard to make. Another issue is that it can offer a quality measure of items, i.e. items that are highly liked by a majority can serve as an indication of the quality of the item.

Based on these observations, the collaborative approach is best suited for users that can be matched with many other users and in an environment where the density of user ratings are relatively high and the item domain is small and static.

### The content based filtering approach - weaknesses and strengths

**Content Representation** - Content-based approaches are limited by the attributes that are associated with the items. Obviously, if the item lacks descriptive attributes, it can't be used in a content-based approach and for items like graphical images and audio, it is very troublesome to find descriptive attributes. Another issue is that the content must be added by someone, which can be a costly and tiresome process, and in some cases not even practical [Shardanand and Maes, 1995]. However, the main problem is to find those attributes that contains the most descriptive and significant information about the item, which is often very domain dependent.

**Limited Content Analysis** If two different items are represented with the same attributes, this approach can't distinguish between them. For example, in text-analysis a document is usually represented with its most important keywords, this approach can't distinguish between a badly and a well written article, if they use the same terms.

**Serendipity problem** Since the system only recommends items that matches the user profile, the user will only receive recommendations that are similar to the items the user has already encountered. This leaves little room for new experiences or serendipity and can lead to the user getting tired of the recommendations since the recommendation system seems to only be able to recommend items that the user would have discovered anyhow.

**New user problem** The system can't recommend any items before it knows the user's preferences. In the content based approach, the active user's preferences is the only factor that has influence on the recommendations, fewer preferences means less accurate recommendations.

Advantages with the content-based approach is that a *new item* can be recommended without anyone else having given their opinion on it, thus it doesn't suffer from the new item problem as described in the collaborative approach. *Sparsity* and *scalability* is not an issue, since no matching between users is made, only between users and items.

One problem that both approaches shares is the new user problem, the fact that without enough user preferences, the recommendations will suffer in accuracy. Although the content-based approach has the advantage that it actually can recommend items based on less information about the user than in the collaborative approach. In the content based approach, even if the active user has only rated one item, recommendations of items similar to that particular item can be made, but in the collaborative approach, if two users have one item in common, it gives no indication whatsoever about whether or not they have the same taste and needs.

## 1.7 Case studies of recommender systems in the movie domain

One domain that is well suited for recommender systems is the movie domain. Many movies are just waiting to find their audience but due to the enormous amount of movies, both old and new ones, it can be troublesome for the average movie viewer to find the movies that appeals to his taste. And for the so called movie "cinéasts", discovering those new and old gems beyond ones imagination can be equally challenging. Movie sites on the Web offers a solution to this in many various ways, for example, the



Figure 1.6: The MovieLens welcome screen after you have become a member.

visitor can browse around the site for information on old and new movies, he can search for information regarding movies using criteria's such as who stars in a particular movie, who directed the movie, etc., and he can receive recommendations on movies based on his personal taste in movies. We will be looking at four different websites that operate in the movie domain to see what kind of features they have incorporated into their sites as a way of making it easier for a user to find the information that meets his needs regarding movies. The main focus will be on their recommendation technique; *how they work* and with a subjective opinion by the authors on *how well they work*. Other features that we have found interesting will also be studied and reviewed. We will also compare the recommendation from each one of recommendation systems on the basis of one movie that we like, namely the movie "The Thing" by John Carpenter

### 1.7.1 MovieLens

*MovieLens* is a movie recommendation website that is part of the GroupLens research project at the University of Minnesota. They built MovieLens as a platform for researchers to experiment on in areas related to recommender systems such as recommendation algorithms, recommendation interface design, online community integration etc. By using MovieLens and (optionally) participating in various studies the users contribute with important feedback that helps the researchers to developed and improve MovieLens. However, for most people, MovieLens is just a fun experience where you can receive predictions for movies you haven't yet watched.

#### Recommendations

MovieLens gives *predictions* on movies as well as *non-personalized recommendations*. MovieLens also provides a Top-N list of all movies that the system can make predictions for sorted in descending order of prediction.

#### Predictions

Before you can get any predictions, you must become a member and rate 20 movies [Figure 1.6], that the system randomly chooses, on a scale from 1 to 5.

The ratings for the 20 movies are used by MovieLens to predict ratings [Figure 1.7] on movies that you haven't yet rated. As a member you are encouraged to rate more movies in order to fine tune the accuracy of the predictions that are given to you.

Non personalized recommendations Recently, MovieLens added a new recommendation service called "QuickPick: Our Movie Gift Recommender" which is (more or less) a non personalized recommendation service. You don't need to be a member of MovieLens to use it, just enter the name of one movie that you like and MovieLens will use that movie to find other movies that people have expressed similar

(hide) Predictions for you ↕	Your Ratings	Movie Information	Wish List
★★★★	Not seen	<a href="#">Where the Truth Lies (2005)</a> DVD info imdb add tag Drama	<input type="checkbox"/>
★★★★	Not seen	<a href="#">Blood of Heroes, The (1988)</a> DVD info imdb add tag Action, Sci-Fi	<input type="checkbox"/>
★★★★	Not seen	<a href="#">Decade Under the Influence, A (2003)</a> DVD VHS info imdb add tag Documentary	<input type="checkbox"/>
★★★★	Not seen	<a href="#">Killer, The (Die xue shuang xiong) (1989)</a> DVD info imdb add tag Action, Crime, Drama, Thriller - Cantonese, Japanese	<input checked="" type="checkbox"/>
★★★★	Hide this	<a href="#">Enter the Dragon (1973)</a> DVD VHS info imdb add tag Action, Crime, Drama	<input type="checkbox"/>
★★★★	0.5 stars	<a href="#">Highway 61 (1991)</a> VHS info imdb add tag Comedy	<input type="checkbox"/>
★★★★	1.0 stars	<a href="#">Sword of Doom, The (Dal-bosatsu tōge) (1966)</a> DVD VHS info imdb add tag Action, Drama - Japanese	<input type="checkbox"/>
★★★★	1.5 stars	<a href="#">Professional, The (Le Professionnel) (1981)</a> info imdb add tag Action, Drama, Thriller - French	<input type="checkbox"/>
★★★★	2.0 stars	<a href="#">Cypher (2002)</a> info imdb add tag Action, Fantasy, Sci-Fi, Thriller	<input type="checkbox"/>
★★★★	2.5 stars	<a href="#">Rory O'Shea Was Here (a.k.a. Inside I'm Dancing) (2004)</a> info imdb Drama	<input type="checkbox"/>
★★★★	3.0 stars		
★★★★	3.5 stars		
★★★★	4.0 stars		
★★★★	4.5 stars		
★★★★	5.0 stars		

Figure 1.7: The MovieLens interface. Recommendations are in red stars. You can rate movies you have seen by using the menus.

### Our Recommendations:

- For a Few Dollars More (Per qualche dollaro in più) (1965)
- Ring (Ringu) (1998)
- Hard-Boiled (Lashou shentan) (1992)
- Killer, The (Die xue shuang xiong) (1989)
- Alien (1979)
- Aliens (1986)
- City of God (Cidade de Deus) (2002)
- Evil Dead II (Dead By Dawn) (1987)
- Battle Royale (Batoru Rowaiaru) (2000)
- Evil Dead, The (1981)
- Once Upon a Time in the West (C'era una volta il West) (1968)
- Outlaw Josey Wales, The (1976)

Figure 1.8: Recommendations for the movie "The Thing" given by MovieLens QuickPick recommendation service.

opinions about. It's however also possible to enter the name of multiple movies you like and fine tune the recommendations by making them more personalized. This recommendation service is based on the same technique as that used for making predictions within the MovieLens system. [Figure 1.8] shows the recommendations we got for the movie "The Thing".

## How it works

With reference to a discussion with one of the developers of MovieLens, we will now give brief description of the recommendation technique MovieLens uses and some issues related to it. MovieLens is based on the collaborative filtering approach, before 2003 the recommendation technique they used was a neighborhood based CF technique, i.e. a user based CF technique. That technique used the

Pearson correlation coefficient to determine similarity between users and the weighted deviation from mean prediction formula. Different tweaks (extensions) to the technique was also used, such as only using a relatively small number of the most similar users as recommenders for the active user. Some of the researchers in the GroupLens team founded Net Perception Inc. and brought the source code for the recommendation technique MovieLens was relying on with them, since this was a commercial company the source code became closed sourced. Doing research on a closed sourced recommendation technique makes it hard to maintain and perform new experiments, this was the main reason for a shift to a new recommendation technique. The new recommendation technique that is currently in use by MovieLens is an item based CF technique. Instead of computing similarity between users, the technique computes similarity between pairs of movies (still only relying on user ratings on movies). The movie similarity is then used as a weight in the prediction process. Thus when the active user wants a prediction for a movie (the active movie), the technique will find the 20 movies that the active user has rated that are most similar to the active movie. Note that it will almost always be possible to determine similarity between the active movie and the movies the active user has rated since MovieLens precomputes and stores all item similarities (as opposed to only storing the k highest similarities for each item), however in some cases item similarity can simply not be determine due to missing data. The ratings for each one of these 20 movies are then weighted against their similarity with the active movie and summed up. The average of this weighted sum of ratings is rounded to the nearest half decimal and presented to the user as a prediction on the active movie. Exceptions to the prediction process occurs when the active user's 20 most similar movies to the active movie aren't that similar, then the active movie's average rating (by the whole population) is instead used as a prediction. Additionally when the active movie has been rated by too few users, no prediction at all will be given.

### **Does it work?**

The way the recommendation technique is implemented makes it very conservative. Users with low average ratings will rarely, if ever, get any 5 star predictions. Another reason it's conservative is that in the prediction process, it will almost always find 20 movies to compute predictions with and unless the user has rated these movies very high (or low), the predicted rating tends to be near the user's average rating. Remember that it is the 20 most similar movies out of those that the active user has rated that are used, this means that their similarity isn't necessarily very high, that is a side effect of storing and using all item similarities as opposed to only storing for each item its highest similarities. However, it does ensure that predictions will almost always be possible. Another interesting issue is the difference in accuracy between the predictions made by the new item based CF technique and the old user based CF technique, says one of the developers of MovieLens. The item based CF technique (as implemented) tends to generally be accurate but often, as mentioned, makes predictions near a user's average rating. The user based CF technique (as it was implemented) could be much more speculative, though occasionally wrong. He believes that this can have something to do with the fact that it typically used a total of 50 other users, that had "just one or two ratings [sic]" in common with the active user. Maybe this is the tradeoff that has to be considered for every recommendation technique, should the recommendations be safe and nearly boring or should there be more serendipitous in the recommendations with the risk of giving recommendations that are very wrong in some cases?

### **Browsing**

MovieLens is not a place for browsing around and looking for information about movies. First of all, you need to become a member before you can even take part of the information that is available, and secondly, MovieLens is designed to predict ratings for movies, not to serve as an information source about movies. They do provide some brief information about movies such as title, genre, release year, main cast and director. But for further information they provide a direct link to another movie website (IMDb).

### **Searching**

They support both simple and advance searches, i.e. search by genre, title, main cast, language etc. The search results can be sorted by prediction, title, number of ratings to name a few. You can also



use short-cuts, which are pre-defined searches and you can create your own shortcuts. Examples of pre-defined shortcuts are "Top picks for you" and "Your ratings".

### Other features

MovieLens has started to implement features that encourage people to more actively participate in the website, such as *discussion forums*, allowing users to *edit movie information* and letting users *tag* movies. Tags are words or phrases that are added to the movies by the users. Movies with tags can be organized, searched, classified etc. by their tags. Tags are however currently not used in the recommendation process, it is simply an additional source of movie information. Another nice feature is called *buddies*, you ask other MovieLens members to become your movie buddy and can then view both their predictions for movies and a combine prediction score that reflects the average prediction for the movie among you and your buddies.

## 1.7.2 Internet Movie Database

The *Internet Movie Database* (IMDb), started out as a Usenet newsgroup<sup>4</sup> back in the early 1990's and is now a part of the Amazon group. It is considered to be the largest movie database on the Web, covering nearly 500 000 movie and TV titles. With over 30 million visitors each month and with nearly 9 million registered users, it can be considered to be one of the most popular sites on the Web.

### Browsing

The front page of the website invites a user to start browsing around in a multitude of ways, there are shortcuts with names like top 250 lists, news articles, movies now playing in the theatres, award winners, movie quotes, movie of the day, etc. They even have a shortcut with the name "Browse" that takes you directly to the directory of all IMDb's features. So, finding movies by browsing is easy, just a few clicks and you will have found a couple of movies that might be of interest.

### Searching

So, what about its searching capabilities, how easy is it to find a movie that your favorite director or actor and actress has been involved with? Easy, the front page allows searching the whole database on all attributes or based on a specific attributes like title, plot summary, keyword, etc.

### Recommendations

IMDb has what they call a *recommendation center*, which to them is just one of many features they have incorporated into their site, so no particular effort is made to promote it to the users. Other popular features are for example their message boards, user comments, user votes for movies, goofs, trivia, plot summaries etc. For a complete list and a detailed explanation of all the features offered by IMDb, see IMDb's A-Z Index<sup>5</sup>.

Nevertheless, the user can receive recommendations for movies either by going to the recommendation center and type in the name of a movie or in connection with a movies information page, follow a link named "recommendations", this will give the user recommendations based on the assumption that he likes the movie in question. Since the recommendations are the same for every user, they are non-personalized. [Figure 1.9] shows the recommendations we got for the movie "The Thing" when using the recommendation center.

### How it works

How their recommendation technique works is a bit unclear, the only explanation they give is that *"It isn't feasible to handpick Recommendations for every film. That's why we came up with a complex formula to suggest titles that fit along with the selected film and, most importantly, let our trusted user base steer those selections. The formula uses factors such as user votes, genre, title, keywords, and, most importantly,*

---

<sup>4</sup>[rec.arts.movies](http://rec.arts.movies)

<sup>5</sup><http://www.imdb.com/a2z/>

Suggested by the database	Look up in IMDb	Showtimes (US only)	Available @Amazon	User Rating
<a href="#">Dawn of the Dead</a> (2004)	IMDb		DVD	7.3
<a href="#">Day of the Dead</a> (1985)	IMDb		DVD	6.6
<a href="#">Alien: Resurrection</a> (1997)	IMDb		DVD	6.0
<a href="#">Doom</a> (2005)	IMDb		DVD	5.2
<a href="#">The Lord of the Rings: The Return of the King</a> (2003)	IMDb		DVD	8.9
<a href="#">War of the Worlds</a> (2005)	IMDb		DVD	6.7
<a href="#">Sin City</a> (2005)	IMDb		DVD	8.5
<a href="#">Dawn of the Dead</a> (1978)	IMDb		DVD	7.8
<a href="#">Terminator 3: Rise of the Machines</a> (2003)	IMDb		DVD	6.9
<a href="#">Freddy Vs. Jason</a> (2003)	IMDb		DVD	6.0

The monster inside you	Look up in IMDb	Showtimes (US only)	Available @Amazon	User Rating
<a href="#">Alien</a> (1979)	IMDb		DVD	8.4
<a href="#">An American Werewolf in London</a> (1981)	IMDb		DVD	7.3

IMDb users recommend	Look up in IMDb	Showtimes (US only)	Available @Amazon	User Rating
<a href="#">The X Files</a> (1998)	IMDb		DVD	6.7

Figure 1.9: Recommendations for the movie "The Thing" given by IMDb's recommendation center.

*user recommendations themselves to generate an automatic response.*" Since the recommendations are the same for every user, they are non-personalized. Probably some content-based technique that matches the features of the movie against another movie is used, possibly combined with users ratings.

## Does it work?

If someone would suggest these movies to me I would think he had a pretty good taste in movies, but if someone asked me for recommendation on movies similar to the Thing, then most of these movies wouldn't make the list.

## Other features

In the IMDb database most of the movies have one or more (plot) keywords associated with them, these keywords are manually added descriptive keywords of a movie, for example the movie "The Thing (1982)" has keywords such as "Paranoia", "Alien", "Flame Thrower" and "Ice" associated with it. The movie keyword analyzer (MoKA)<sup>6</sup> is a relative new feature added by the folks at IMDb which allows filtering movies by their keywords, say for example that you are in the mood for some gore, just enter the keyword "Gore" in the IMDb keyword search as the starting keyword for your exploration. A list with keywords matching the search will be shown with the word Gore displayed at the top and the information that there's 2082 titles somehow related to the keyword Gore! Partial matches of the word gore like "Gore-film" and "Al-gore" as well as approximate matches like "Hero-gone-bad" will be shown. Each matching keyword on the list will take you to the complete list of titles that somehow is related to the chosen keyword. We choose to follow the link for the keyword Gore which after all was what we were interested in. And Voila! A list of all the 1019 titles that feature this exact keyword is presented for us together with a related keywords map [Figure 1.10].

<sup>6</sup><http://www.imdb.com/moka/>



Figure 1.10: Part of IMDb's MoKA keyword map for the keyword "Gore" together with the 25 (out of 1019) highest rated movies featuring the keyword.

The related keywords map consists of all the different keywords that are associated with at least one of the 1019 Gore movies. Together with the keyword is a number that indicates how many of the Gore movies that are associated with the each keyword, by clicking on one of the keywords, we can narrow down the list of movies. We choose to click on the keyword "*Blood*" since we want some blood in our Gore movie as well. This action reduces the list to 590 movies that all feature both the keyword Gore and the keyword Blood. A smaller related keywords map showing all keywords that are featured in at least one of the remaining 590 movies is shown [Figure 1.11]. Narrowing the search down further to include also the keyword "*Exploding head*" results in 48 movies [Figure 1.12]. As there is still too many movies to choose from, we narrow it down by selecting the word "*Zombie*" and end up with 16 movies [Figure 1.13].

Finally we choose the word "*Disembodied Hand*" [Figure 1.14], leaving us with three movies, of which the one with the highest user rating is entitled "*Evil Dead II*", sounds like an interesting movie, lets watch it! As [Figure 1.14] shows the remaining three movies have a lot of keywords associated with them, however further narrowing seems pointless.

This way of letting the user select keywords and also see how the resulting list narrows down to just a few movie titles is fun and inspiring, you wonder what kind of movies will be left that corresponds to this list of keywords. The keyword map itself is nothing special, its just a rectangular map with the keywords in alphabetic order with words that appear in many titles in more bolder style than keywords that appears in fewer movies. But its more fun to pick keywords this way than from a vertical list or by figuring them out by your own.

### 1.7.3 All Movie Guide

*All Movie Guide* ("Allmovie")<sup>7</sup> is a searchable database of movie related information that covers over 260.000 movies. Just like IMDb, the site has a number of different features for finding movies. You can

<sup>7</sup><http://www.allmovie.com/>



Figure 1.11: Part of IMDb's MoKA keyword map for the two keywords "Gore" and "Blood" together with the 25 (out of 590) highest rated movies featuring the keywords.

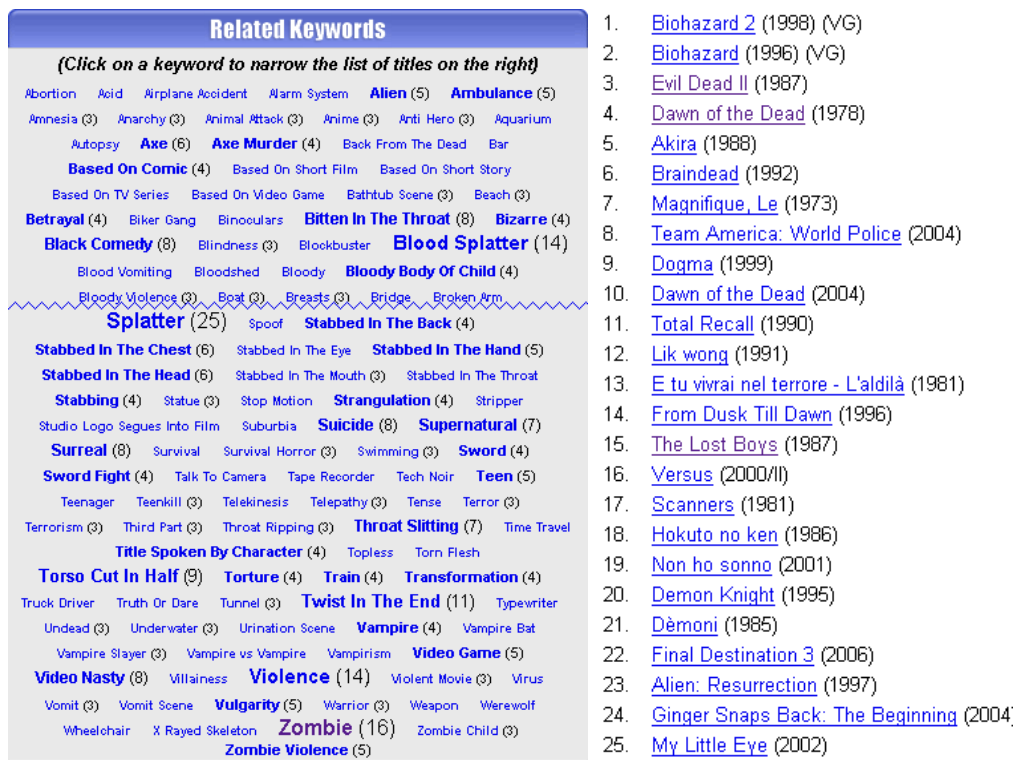


Figure 1.12: Part of IMDb's MoKA keyword map for the three keywords "Gore", "Blood" and "Exploding head" together with the 25 (out of 48) highest rated movies featuring the keywords.



1. [Biohazard 2](#) (1998) (VG)
2. [Biohazard](#) (1996) (VG)
3. [Evil Dead II](#) (1987)
4. [Dawn of the Dead](#) (1978)
5. [Braindead](#) (1992)
6. [Dawn of the Dead](#) (2004)
7. [E tu vivrai nel terrore - L'aldilà](#) (1981)
8. [From Dusk Till Dawn](#) (1996)
9. [Versus](#) (2000/II)
10. [Waxwork](#) (1988)
11. [Doom](#) (2005)
12. [Incubo sulla città contaminata](#) (1980)
13. [Más allá del terror](#) (1980)
14. [Virus](#) (1980/I)
15. [Zombie '90: Extreme Pestilence](#) (1991) (V)
16. [House of the Dead](#) (2003)

Figure 1.13: Part of IMDb's MoKA keyword map for the four keywords "Gore", "Blood", "Exploding head" and "Zombie" together with all 16 movies featuring the keywords.



1. [Evil Dead II](#) (1987)
2. [Versus](#) (2000/II)
3. [Waxwork](#) (1988)

Figure 1.14: Part of IMDb's MoKA keyword map for the five keywords "Gore", "Blood", "Exploding head", "Zombie" and "Disembodied Hand" together with all 3 movies featuring the keywords.

for example search for movies with two people in common or you can search for people with two movies in common.

## Browsing

On Allmovie's front page you are presented with selections of movies and related information to start out with under headings like trailers, new DVD releases, essays about movie related subjects, movies now showing in theaters. There is also have a feature called *Quick Browse* that lets you browse movies by genre, country and time period. A bit modest compared to IMDb's front page, but it will get you started.

## Searching

A search option is available at the top of each page but, strangely, there is no advanced search option anywhere. Two funny search options entitled *film finder* and *people finder* are however available. The film finder lets you type in the name of two people and see if they have a movie in common. The people finder lets you type in the name of two movie titles to see which people they have in common.

Another feature is glossary, which is a reference source for a wide variety of movie and film-related terms written by experts in the field with various search capabilities. The glossary includes an explanation for all kinds of movie related terms which can be of help in the search process.

## Recommendations

The user can't ask for recommendations, instead, the webpage for a particular movie contains not only the movie's attributes (which there is often an abundance of!) but in many cases also three different lists of movie recommendations. They have choose to categorize their recommendations as; similar movies, movies with the same personnel and other related movies. The title of the movie is presented with its release year and director. Since the recommendations are the same for every user, they are non-personalized. [Figure 1.15] shows the recommendations we got for the movie "The Thing".

## How it works

As opposed to IMDb, these recommendations are manually selected by experts working for Allmovie, at least according to the answer we got from them when we asked them how the recommendations were made.

## Does it work?

One can tell by looking at the recommended titles that there has been some effort behind these recommendations, they aren't just picked because they are in the same genre or something trivial, it's more because they share the same mood or have a similar plot, something not quite graspable but definitively similar. Unfortunately, you can't choose any movie you want as seed movie for the recommendations, as in the IMDb case, since not all movies have a recommendation list on their information page.

### 1.7.4 Amazon

*"If I have 3 million customers on the Web, I should have 3 million stores on the Web."*  
– Jeff Bezos, CEO of Amazon.com

*Amazon* is an American e-commerce company that started out in 1995 as an online bookstore. Due to its growing popularity it didn't take long before Amazon also started selling other products such as CDs, DVDs, computer software, gourmet food, musical instruments, toys, etc. As a customer visiting Amazon you are faced with so many options and products to choose among, that without some guidance you will probably real soon get lost. To overcome this problem, Amazon has come up with a variety of solutions for enhancing the customer's shopping experience. The solutions can be divided in the following three categories:

**Site features** Recommendations of various kinds shown while browsing the webstore for products.

#### SIMILAR MOVIES

[Endangered Species](#) (1982, *Alan Rudolph*)  
[Hiruko the Goblin](#) (1990, *Shinya Tsukamoto*)  
[Night Beast](#) (1983, *Donald M. Dohler*)  
[Body Snatchers](#) (1993, *Abel Ferrara*)  
[Boa](#) (2002, *Phillip J. Roth*)  
[Alien](#) (1979, *Ridley Scott*)  
[The Fly](#) (1986, *David Cronenberg*)  
[The X-Files: Ice](#) (1993, *David Nutter*)  
[Shadowzone](#) (1990, *J.S. Cardone*)

#### MOVIES WITH THE SAME PERSONNEL

[Halloween](#) (1978, *John Carpenter*)  
[Dark Star](#) (1974, *John Carpenter*)  
[The Fog](#) (1979, *John Carpenter*)  
[Prince of Darkness](#) (1987, *John Carpenter*)  
[The Terminal Man](#) (1974, *Mike Hodges*)  
[Escape from New York](#) (1981, *John Carpenter*)  
[They Live](#) (1988, *John Carpenter*)  
[The Mean Season](#) (1985, *Phillip Borsos*)

#### OTHER RELATED MOVIES

is a version of: [The Thing](#) (1951, *Christian Nyby*)  
is featured in: [Terror in the Aisles](#) (1984, *Andrew Kuehn*)  
is related to: [Alien](#)<sup>3</sup> (1992, *David Fincher*)  
[Elvis](#) (1979, *John Carpenter*)  
[John Carpenter's Ghosts of Mars](#) (2001, *John Carpenter*)  
[John Carpenter: Fear is Just the Beginning... The Man and His Movies](#) (2002, *Garry Grant*)

Figure 1.15: Recommendations given for the movie "The Thing" by All Movie Guide.

**Your community** Provides, among other things, personalized recommendations and different ways for customer to interact with each other.

**Notifications & E-mail subscribing** An e-mail service that sends recommendations for new items or other interesting news that the customer has choose to be informed about.

Our focus will be on what kind of site features Amazon.com has incorporated into their movie section, i.e. the DVD section. Some community features will also be discussed, such as tagging, rating and reviews.

### Browsing

Amazon's DVD section is filled with different browsing options. You can browse by genre, top sellers, new & future releases, blockbusters, offers, etc. The trick is to lure the visitor into browsing and by that process convert the visitor into a buyer. Amazon does this with the help of different site features where recommendations play the major role.

### Searching

Amazon offers both a simple search and an advanced search with the usual list of search options for movies, such as title, actor, director, genre, etc. One interesting search option allows searching for special features on the DVD, like alternating endings, deleted scenes, etc.



Figure 1.16: All the features that are enable from "Your Store"

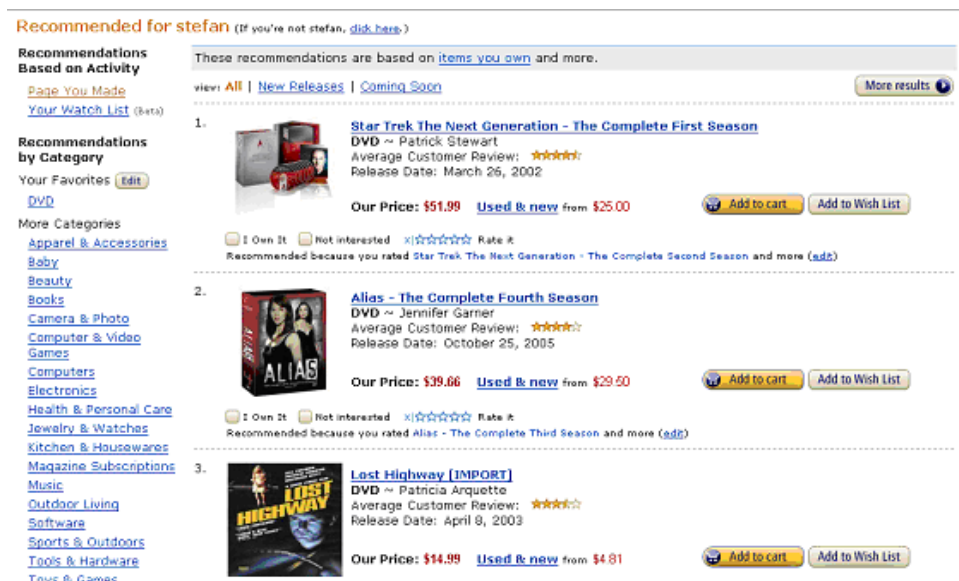


Figure 1.17: Amazon provides a list of personalized recommendations to members in their Your Store feature.

Amazon also has their own search engine called A9<sup>8</sup>, that allows customers to search the internet. Interesting about A9 is that it uses the customers account information to provide personalized search results as well as the users search history.

## Recommendations

To get the most out of the recommendations, Amazon suggests that every user become a member of Amazon's community by setting up an account. In order to buy something from Amazon, the user must create an account, so every *customer* is by default also a member of the Amazon community. After a user has become a member, Amazon activates a feature called *Your Store* [Figure 1.16]. This feature gives the user access to a range of features only available for members. Among those features is the ability for a member to improve his recommendations by giving explicit feedback about his preferences.

Every time the customer visits Amazon, he can log into his store and receive recommendations that are based on his own taste [Figure 1.17].

Amazon also keeps track of all the items that a user has viewed and searched for, and the type of product categories that have been visited, during the current visit. This kind of implicit information

<sup>8</sup><http://www.a9.com/>



[Recommended For You](#) > [Page You Made](#)

**YOUR HISTORY**  
Here are the items in your recent history ([click here](#) to sign in.)  
**Items in your history**  
☒ [The Thing \(Collector's Edition\)](#)  
DVD ~ Adrienne Barbeau  
**Your Recent Searches**  
☒ [the thing](#) in DVD  
[Update](#) [Clear all](#)  
**To turn off the Page You Made and your recent history, [opt out here](#).**  
**Need Help?**  
Visit our [help](#) area to learn more.

**Customers who bought the items in your history also bought:**  

---

- **[The Fog \(Special Edition\)](#)**  
DVD ~ Tom Atkins  
Average Customer Review: ★★★★★  
Release Date: August 27, 2002  
**Our Price: \$12.99**

---

- **[They Live](#)**  
DVD ~ Roddy Piper  
Average Customer Review: ★★★★★  
Release Date: September 23, 2003  
**Our Price: \$10.99**

---

- **[Escape from New York \(Special Edition\)](#)**  
DVD ~ Tom Atkins  
Average Customer Review: ★★★★★  
Release Date: December 16, 2003  
**Our Price: \$26.99**

---

- **[The Thing from Another World](#)**  
DVD ~ Margaret Sheridan  
Average Customer Review: ★★★★★  
Release Date: August 5, 2003  
**Our Price: \$14.99**

Figure 1.18: Recommendations for the movie *The Thing* given by Amazon on their Page you made feature, which bases recommendations primarily on products viewed during the current visit to Amazon. The top four recommendations out of totally twenty are shown in the figure.

**Better Together**  
Buy this DVD with [The Fog \(Special Edition\)](#) DVD ~ Tom Atkins today!  
 +   
**Total List Price: ~~\$29.92~~**  
**Buy Together Today: \$23.98**  
[Buy both now!](#)

**Customers who bought this DVD also bought**  
[The Fog \(Special Edition\)](#) DVD ~ Tom Atkins  
[They Live](#) DVD ~ Roddy Piper  
[Escape from New York \(Special Edition\)](#) DVD ~ Tom Atkins  
[The Thing from Another World](#) DVD ~ Margaret Sheridan  
[Explore Similar Items:](#) in [DVD](#)

Figure 1.19: Recommendations for the movie *The Thing* shown at the product page for the movie.

is then used by a feature called *The page You made* and *Your recent history* [Figure 1.18] to create recommendations based on the interests shown during the current visit.

Amazon also supports non-personalized recommendations. For example when a customer is viewing the product page for an item, in this case for the movie *The Thing*, recommendations of the form "*Customers who bought/viewed this item also bought/viewed these items*" or "*Better together*" (Amazon.com suggest an item that they think the customer should buy together with the one currently being viewed) are shown [Figure 1.19].

A customer at Amazon has his own shopping basket, when the customer adds something to the shopping basket or just views its contents, a page with recommendations is shown [Figure 1.20]. These recommendations are based on the current contents of the shopping basket and on the item (if any) just added to the shopping basket. Recommendations are of the type "*Customers who bought/shopped/viewed the items in the shopping basket also bought/shopped...*" This feature provides a nice browsing experience, where users can always easily find a interesting path to continue browsing on after finding a product interesting enough to add to the shopping basket.

Customers who bought The Thing [1982] also bought:

 <p><a href="#">The Wicker Man - Special Edition Director's Cut (2 disc set) [1973]</a></p> <p>Our Price: <b>£19.99</b></p> <p>Used and new from <b>£4.79</b></p> <p><a href="#">Add to Basket</a></p> <p><a href="#">Explore similar items</a></p>	 <p><a href="#">Escape From New York [1981]</a></p> <p>Our Price: <b>£5.97</b></p> <p>Used and new from <b>£5.96</b></p> <p><a href="#">Add to Basket</a></p>	 <p><a href="#">They Live [1989]</a></p> <p>Our Price: <b>£4.97</b></p> <p>Used and new from <b>£4.01</b></p> <p><a href="#">Add to Basket</a></p>
--	--	---



Customers who shopped for this item also shopped for these items:

 <p><a href="#">Prince Of Darkness [1988]</a></p> <p>Our Price: <b>£6.97</b></p> <p>Used and new from <b>£4.50</b></p> <p><a href="#">Add to Basket</a></p> <p><a href="#">Explore similar items</a></p>	 <p><a href="#">Halloween (25th Anniversary Edition)</a></p> <p>Our Price: <b>£5.97</b></p> <p>Used and new from <b>£3.44</b></p> <p><a href="#">Add to Basket</a></p>	 <p><a href="#">The Fog [1979]</a></p> <p>Our Price: <b>£7.97</b></p> <p>Used and new from <b>£4.50</b></p> <p><a href="#">Add to Basket</a></p>
---	---	---

Customers who bought the items in your Shopping Basket also bought:

 <p><a href="#">Bad for Good</a></p> <p>Our Price: <b>£7.99</b></p> <p>Used and new from <b>£3.74</b></p> <p><a href="#">Add to Basket</a></p>	 <p><a href="#">Welcome to the Neighbourhood</a></p> <p>Our Price: <b>£6.99</b></p> <p>Used and new from <b>£1.90</b></p> <p><a href="#">Add to Basket</a></p>	 <p><a href="#">Bad Attitude</a></p> <p>Our Price: <b>£6.99</b></p> <p>Used and new from <b>£4.22</b></p> <p><a href="#">Add to Basket</a></p>
---	---	---

Recently Viewed Items

 <p><a href="#">The Eye - Sarah Lucas</a></p> <p>Our Price: <b>£11.99</b></p> <p><a href="#">Add to Basket</a></p> <p><a href="#">See more in the Page You Made</a></p>	 <p><a href="#">Satoshi Tomiie - Renaissance Presents 3d</a></p> <p>Our Price: <b>£12.99</b></p> <p>Used and new from <b>£10.65</b></p> <p><a href="#">Add to Basket</a></p>
--	---

Top Sellers

 <p><a href="#">The Phantom Of The Opera</a></p> <p>Our Price: <b>£8.97</b></p> <p>Used and new from <b>£4.60</b></p> <p><a href="#">Add to Basket</a></p>	 <p><a href="#">24: Series 4</a></p> <p>Our Price: <b>£16.97</b></p> <p>Used and new from <b>£14.75</b></p> <p><a href="#">Add to Basket</a></p>	 <p><a href="#">Battlestar Galactica - The Mini Series [2004]</a></p> <p>Our Price: <b>£5.97</b></p> <p>Used and new from <b>£5.30</b></p> <p><a href="#">Add to Basket</a></p>
---	---	--

Figure 1.20: Recommendations shown when a customer views the contents of the shopping basket, in this case the shopping basket contains only the movie The Thing. The recommendations are based on the items currently in the shopping basket.

## How it works

According to the paper written by Linden [Linden et al., 03], Amazon.com's recommendations are based on what Linden calls *item-to-item collaborative filtering*. It is clear that the recommendation engine uses the purchase history of customers, since this is mentioned both in the paper and several times at the Amazon.com website. However, they also mention that customers ratings on items and items that the customer has specified he owns are used in the recommendation process with the possibility to exclude specific items from the recommendation process. So they have probably implemented some tweaked version of the recommendation technique described in the paper. Also what is interesting is that Amazon supports short termed user profiles containing e.g. browsed products as opposite to long termed user profiles consisting of e.g. the user's purchase history. The short termed profiles of a customer can be found in the feature *The page you made*, where the customers recent activity on the web-site can be found together with recommendations based on those activities. The support of both short termed user profiles and long termed user profiles is something that are considered to be highly important, and is referred to respectively as *ephemeral recommendations* and *persistent recommendations* in [Schafer et al., 01]. By giving ephemeral recommendations, the recommender system hopes to catch the customers immediate need and change in taste for, in this case, movies.

## Does it work?

In the authors opinion, it works well, but it is quite obvious that Amazon goes for safe recommendations, probably because the customer is more willing to buy something he is familiar with than going for something completely new and risky. Amazon makes sure to always motivate their recommendations to some degree, usually by simple but clear formulations such as "*Customers who bought item X also bought...*" but in certain cases also by showing which exact purchased products resulted in a specific recommendation. The system is easier to trust and, at a user level, to understand thanks to the usages of these motivations. By supporting different kinds of recommendations, Amazon makes sure that they can always give recommendations that to some degree are based on the customers preferences.

## Other features

When the customer visits a product page, he is not only faced with recommendations and facts about the product, he can also read what the Amazon user community thinks about the product. Some of the features that appears on the product site for a DVD in Amazons DVD-section are:

**Customer reviews and ratings** Every customer that is a member of the Amazon.com community can write reviews and rate items. This allows for other customer to see what previous customer thought about the item.

**Listmania!** Shows lists created by customers. The feature in general allows customers to create lists about just everything. The lists shown on a product page have some kind of connection to it and serves as a kind of related products list.

**So you'd like to...** Allows customers to help each other find items and information that may be relevant in connection with the product. For example, for the movie *The Thing* there is a guide named "*So you'd like to... get scared*" out of your wits, this guide contains information about other movies that the writer of the guide thinks would interest a person that likes *The Thing* in the sense that the person also likes to get scared.

**Customer tagged this item with** Tags are word or phrases that are given to items, in this case movies. The main purpose of showing the tags for a movie is in the hope that they contain some meaningful information that are of interest to the customer, and to provide another way of browsing movies as it is also possible to click on the tags to see what other movies have been tagged with the same tag.

**Customers that tagged this item** Shows names of people that have tagged the movie, by clicking on their names the customer is taken to the person's *Amazon homepage*. The Amazon homepage contains more information about the person who tagged the movie, such as other movies they have tagged, which might provide yet another way of browsing for movies.

### 1.7.5 Recommendations for the movie *The Thing*

Since the recommendation systems that we have reviewed are based in some cases on different techniques and operate on different background data but in the same domain, we thought it would be interesting to see if there is any big difference among the given recommendations. To test this in a *slightly* subjective way we gave each recommendation system one movie to base their recommendations on, the movie we used was *The Thing* directed by John Carpenter in 1982. [Table 1.2] present all recommendations that are given for the movie *The Thing* by the different recommendation systems.

We can make the following observations based on [Table 1.2]:

1. MovieLens shows 12 recommended movies when using their QuickPick service.
2. Amazon shows 20 recommendations when asked for recommendations on the form of "*Customers who bought this DVD also bought...*". Note that Amazon as mentioned provides many more different types of recommendations ("*Customers who bought/shopped/browsed/tagged*" etc.), we picked the most common and what we consider prominent type of recommendation for this comparison.
3. IMDb shows 13 movies in their "*Recommendation center*" although the last recommendations are not based on their recommendation algorithm, instead those are recommended using the motivation "*The monster inside you*" (the 11th and 12th recommendation) and "*IMDb users recommend*" (the 13th recommendation).
4. Allmovie groups three categories of recommendations, "*Similar movies*", "*Movies with the same personnel*" and "*Related in other ways*", together into one listing. In this comparison this was the longest list of recommendations.

The plot synopsis for the movie *The Thing* on the IMDb reads as follows "*An American scientific expedition to the frozen wastes of the Antarctic is interrupted by a group of seemingly mad Norwegians pursuing and shooting a dog. The helicopter pursuing the dog crashes leaving no explanation for the chase. During the night, the dog mutates and attacks other dogs in the cage and members of the team that investigate. The team soon realizes that an alien life-form with the ability to take over other bodies is on the loose and they don't know who may already have been taken over.*"

A closer look at Allmovie's list of "Similar movies" reveals that the common theme among these movies seems to be the unknown desire for flesh - particular human flesh - either by consuming it, remodeling it, or by just taking over the whole body. For example "The Thing" has one scene where a human head turns into a spider-like creature, and the movie "Hiruko the Goblin" is about demons that decapitate humans and use their heads on spider-like creatures to fight humanity. The scene where the dog mutates can be thought of as similar to the plot in the movie "Endangered Species" where the town sheriff starts to investigate the rash of cattle mutilations - is it caused by extraterrestrials or is it the government that is behind it?

Amazon recommends eight movies directed by Carpenter and most of the other movies belongs to the Horror and Alien genre. One movie that stands out from the rest is "The Thing from another planet", another adoption of John W. Campbell's short story "Who Goes There?", the short story that "The Thing" is also based on. Another recommendation that stands out from those given is "Sin City", which can perhaps be seen as a slightly serendipitous recommendation.

The recommendations from IMDb seem to be either Alien or Gore movies with the exception of the movies "The Lord of the Rings" and "Sin City". Notable is that none of the recommended movies are directed by Carpenter.

MovieLens offer the most divergence among the movies that are recommended, no obvious common theme can be detected and the movies belongs to a broad category of genres like westerns, horror, Hong Kong action, drama, gore and alien movies. Also here we notice that none of the recommended movies are directed by Carpenter.

The recommendations given by Amazon and IMDb seem to be very genre dependent, i.e. since "The Thing" by mainstream media is usually categorized as a Alien(Science fiction)/Gore/Horror-movie, the recommended movies will also be from these genres. Amazon's dependency seems to also include directors, since it recommends a lot of movies from the same director. All the recommended movies are also very known movies in their respective genre and some are known even for the broader movie public. We would categorize the recommendations from Amazon and IMDb as very safe ones, a user who receives

MovieLens <i>QuickPick: Our Movie Gift Recommender</i>	Amazon.com <i>Customers who bought this DVD also bought</i>	IMDb <i>Recommendation Center</i>	Allmovie.com <i>Combined Similar movies Movies with same person- nel Other related movies</i>
For a Few Dollars More (1965)	The Fog (1980)	Dawn of the Dead (2004)	Endangered Species (1982)
Ringu (1998)	They Live (1988)	Day of the Dead (1985)	Hiruko the Goblin (1990)
Hard-Boiled (1992)	Escape from New York (1981)	Alien: Resurrection (1997)	Night Beast (1983)
The Killer (1989)	The Thing from Another World (1951)	Doom (2005)	Body Snatchers (1993)
Alien (1979)	Prince Of Darkness (1987)	The Lord of the Rings: The Return of the King (2003)	Boa (2002)
Aliens (1986)	An American Werewolf in London (1981)	War of the Worlds (2005)	Alien (1979)
City of God (2002)	In the Mouth of Madness (1995)	Sin City (2005)	The Fly (1986)
Evil Dead II (1987)	Land of the Dead (2005)	Dawn of the Dead (1978)	The X-Files: Ice (1993)
Battle Royale (2000)	Big Trouble in Little China (1986)	Terminator 3: Rise of the Machines (2003)	Shadowzone (1990)
The Evil Dead (1981)	The Shining (1980)	Freddy Vs. Jason	Halloween (1978)
Once Upon a time in the west (1968)	Assault on Precinct 13 (1976)	Alien (1979)	Dark Star (1974)
The Outlaw Josey Wales (1976)	Halloween (1978)	An American Werewolf in London (1981)	The Fog (1979)
	Alien (1979)	The X Files (1998)	Prince of Darkness (1987)
	Sin City (2005)		The Terminal Man (1974)
	Invasion of the Body Snatchers (1956)		Escape from New York (1981)
	AVP - Alien Vs. Predator (2004)		They Live (1988)
	Predator (1987)		The Mean Season (1985)
	Poltergeist (1982)		The Thing from Another World (1951)
	The Exorcist (1973)		Terror in the Aisles (1984)
	Escape From L.A. (1996)		Alien3 (1992)
			Elvis (1979)
			Ghosts of Mars (2001)
			John Carpenter: Fear is Just the Beginning... The Man and His Movies(2002)

Table 1.2: Recommendations for the movie The Thing (1982) by John Carpenter given by four different web-sites.

this kind of recommendations would probably not be disappointed, which is of great importance for a web-store like Amazon, since a disappointed customer could lead to loss of money. However, showing the customer only movies he already knows about, it is highly likely someone purchasing a movie by John Carpenter is familiar with his other movies, might lead to the customer ignoring the recommendations completely in the long run. It's a difficult balance.

When it comes to a web-site like IMDb, it wouldn't hurt with a bit more serendipity among the recommendations, since the risk of making a bad recommendation is less expensive than for a web-site that makes a living out of selling movies. The overall judgment is that for a person who is unfamiliar with the genres that the movie "The Thing" belongs to, IMDb provides some decent recommendations, but for someone familiar with movies from these genres, chances are very high that he has already watched all the recommended movies.

Allmovie had the most interesting recommendations, their viewpoint is that movies with similar themes are good recommendations. The serendipity factor is high since the movies can be classified as non-mainstream movies, compared to movies like "Alien" and "Dawn of the dead", that are movies that are well-known in their respective genre. If you want to be introduced to new genres and movies you otherwise wouldn't come across, then Allmovies is a good place. The major drawback with their recommendation technique is the fact that it is made manually, which has the consequence that recommendations are not given for all movies and can at times become too subjective.

MovieLens was the web-site that gave the most divergent recommendations, ranging from a wide variety of genres and mostly with no obvious common theme between the movies. All the recommendations, with one exception "City of God", were movies the authors had previously seen and enjoyed. The recommendations have certain elements of similarity. Looking at the recommendation Alien, we find a recommendation given by all the other recommender systems too. A safe recommendation, that can be seen to have both elements of science fiction and elements of atmosphere in common with the seed movie. However, most of the recommendations are not so obvious recommendations. For example "The Killer" and "Hard Boiled" is suggested, two Hong Kong movies where more bullets are fired each minute than there are seconds in an hour[sic]. Based on the authors own movie taste those two movies would actually be good serendipitous recommendations to someone who likes The Thing. Since MovieLens makes recommendations based on the collaborative "people who liked this movie also liked these movies" approach, indeed the basic motivation behind the recommendations is precisely such subject opinions, however not among only the authors, but among a larger group of people. The authors considers it a good sign for a recommender system to be able to make such leaps between apparently divergent genres and themes, that still retain a level of relevance and interest to the receiver of the recommendations.

While browsing a web store such as Amazon might one might expect recommendations to constantly retain a clear relevance and possibility for easy motivations, as such making serendipitous recommendations like MovieLens might be difficult. However, nothing prevents a web store from including such recommendations, in the hopes of steering a customer into new areas of interest.

*The authors would like to emphasize that this has by no means been a comparison on how good or effective respective recommendation technique is compared to each other, since such comparisons would require that testing them all on the same dataset and using a large evaluation dataset. Indeed, only a minor part of some of the systems was even touched upon. This is only a subjective comparison of the respective systems recommendations given one single seed movie and based on the authors experience and knowledge in movies.*

### 1.7.6 Conclusions

Movie databases like IMDb and Allmovie are databases that are loaded with facts about and around movies, thus much effort has been put into their searching and browsing features. When it comes to recommendations, Allmovie goes for quality rather than quantity compared to IMDb. The authors have found that the manual recommendations given by Allmovie is often of more value than those given by IMDb. On the other hand is IMDb loaded with more movies and information and rich with features for filtering this information, which is often more useful. Amazon, the only strictly commercial web-site in this review, tries to please its customers rather than surprise them and risk negative feedback. Nevertheless, by setting up an account and telling Amazon who you are and what your interests are, your shopping experience will be enhanced. The authors have been particularly pleased with as an experiment shopping for 80's slashers at Amazon and relying only on the recommendations trail given when first

adding the movie Text Cheerleader Massacre to the shopping basket. MovieLens, an ongoing research project, does take risks and elaborates with their recommendations as well as their web-site since that is the nature of their research. The authors have found that the non-personalized recommendation service gives very good recommendations, but the authors experience of the systems ability to make *personalized* predictions for movies hasn't been as positive. Most of the predictions were near the authors average rating (including movies the authors weren't very fond of) and predictions that consists of extreme ratings such as 1 star or five star were very rare, similar as to what was commented on by one of the developers.

## 1.8 Patents

During the years, several patents for recommender systems have been granted by the U.S patent and trademark office, some notable ones is Cendant Publishing's patent "*System and method for providing recommendation of goods or services based on recorded purchasing history*" in September 4, 1997 [Stack, 1997] and Amazon.com's patent "*System and methods for collaborative recommendations*" filed in March 17, 1998 [Jacobi and Benson, 1998]. There has been, and still is, a lot of debate around these and other similar patents. Companies are suing each other for using what they claim to be there inventions, as an example Cendant sued Amazon for infringed on their "370 patent" for providing people with recommendations of goods or services to purchase based on a database of previous purchasing histories of other customers. The suit was filed Oct. 29 2004 [Kawamoto, 2004].

## 1.9 Where are we today?

From originally having being a research question about "*How to do it?*" the question has after the ACM SIGIR'99 workshop in recommender system become a question of "*How can we do it better?*" and "*How do we know it's better?*".

Herlocker, one of the co-founders of GroupLens, addressed these issues in his dissertation [Herlocker, 2000] on recommender systems based on the collaborative approach. Herlocker tested over 1800 different collaborative filtering techniques and also goes through the different kinds of metrics that have been used to evaluate recommender systems. Based on the results, he gives a framework for designing recommender systems based on the collaborative filtering approach and what metrics are to be used to evaluate the systems. He also tackles the problem of explaining the recommendations for a user by conducting an on-line research where different recommendation interfaces are shown to a group of users.

In 2001, the SIGIR 2001 workshop for Recommender systems was held as a follow up to the SIGIR'99 workshop. This time the workshop was hosted by Herlocker and the theme was now "*algorithms, applications, and interfaces*". As an attempt to get a clearer picture of the ongoing work in the field of recommender systems, all individuals or groups that are involved in the research, development or production of recommender systems are encourage to submit a one page summary of their work to the workshop. These will then be aggregated and published in a paper entitled "*Who's who in recommender systems*".

The shift from a pure algorithmic to a more user oriented focus on recommender systems research is something that seems to be gaining more and more interest in the research community today. In a speech given at Yahoo in 2004 Herlocker points out two issues missing in today's recommender system research, a uniform standard for evaluating recommender systems and a focus on user interfaces. John Riedl, another former member of the GroupLens team, wrote the year after in a foreword, published in connection with that year's ACM special issue on recommender systems [Riedl and Dourish, 2005], that new algorithms only offers marginal improvements and that focus should now be on providing new ways for users of recommender systems to search, browse and better understand recommendations. These new capabilities will be realized through new interfaces for recommender systems.

## Chapter 2

# Implementation and evaluation of common recommendation techniques

*“Everybody got mixed feelings. About the function and form. Everybody got to deviate from the norm.”* – RUSH

Recommender systems help users find interesting items among a large set of items within a specific domain by using knowledge about user’s preferences in the domain. We will be working in the movie domain, applying common recommendation techniques to large sets of movie data. Although what we describe is aimed at the movie domain, much of what we describe also applies to other domains.

The architecture of a recommender system can be divided into three distinct, but tightly coupled, modules:

1. Recommendation profiles
2. Recommendation techniques
3. Recommendation interfaces

We will in this part focus on implementing recommender techniques with the intention of measuring their accuracy in terms of accurate predictions and accurate Top-N ranking lists, thus focus will lie only on the recommendation profiles and recommendation techniques. The recommendation techniques are the algorithms that produces predictions or Top-N ranking lists.

By prediction we will mean a value signifying how much a user will like a specific item, the prediction is usually, but not necessarily, on the same scale as the underlying rating data used to make the prediction. And by Top-N ranking list is meant a list of the most relevant items for a user sorted in decreasing order of relevance.

We will be using two common approaches for the recommender techniques that we will implement and evaluate.

A recommendation approach solves the problem of what will motivate a recommendation. Two common approaches are collaborative filtering and content based filtering. The CF approach relies on the opinions of a large group of users, whereas the CBF approach relies on pure facts.

A number of common recommendation techniques, some of which have already been introduced in Part I, build on the CBF and CF approach will be discussed, implemented and evaluated. In particular evaluation will be made based on three datasets, one is a modified version of the well known MovieLens rating dataset commonly used in scientific papers of this nature, the other two datasets, a rating and a transaction dataset, are collected from a Swedish e-commerce site selling DVD movies.

Evaluation will be performed using common accuracy and relevance metrics, a thorough explanation of the evaluation procedure and evaluation data will be given.



## 2.1 Recommendation profiles

Fundamental entities of any recommender system are *user profiles* and *item profiles*. The profiles are a necessary part of the recommendation process. It is important to consider both the contents of the profiles as well as the internal representation of the profiles. The profiles must contain enough information for the system to be able to make accurate recommendations based on them. The shape and amount of information differs from which approach is taken. We divide the process of creating profiles into two steps; *information analysis* focuses on which information is needed and how to collect it; *profile analysis* focuses on how to represent the profiles such that they can be used by the system in the recommendation process. The goal of this module is to create profiles that capture the taste and needs of the users and that are most descriptive of items.

## 2.2 Information analysis

There are two ways for the system to collect knowledge about a user's item preferences. Users can either *explicitly* enter their preferences into the system, by rating items or by filling out e.g. a form in which preferences can be described. Or the system can study how the users interact with the system and *implicitly* acquire knowledge about user preferences, for example the user's purchase history can be used as a implicit indication of preferred items, or site access logs can be used to determine which product pages are viewed more often by the user [Schafer et al., 2001].

Some form of *feedback* from the user is also desirable as the system can then improve future recommendations.

Information that represents users preferences can either be ratings for items or content based, e.g. movie genres and directors. Items can be represented by ratings given by users or by its content, e.g. attribute such as genre and director. Items can also be represented using keywords extracted from textual descriptions of the items, if such are available.

Once required data has been identified and collected, the actual profile representation needs to be considered.

## 2.3 Profile representation

It is necessary to represent the profiles in such a way that they can be used efficiently by the recommendation techniques to producing recommendations. The representation of the profiles is usually referred to as a model. Two classical representations is the user-item matrix model, commonly used by recommender systems implementing the CF approach, and the vector space model, commonly used in systems implementing the CBF approach. Montaner presents in [Montaner et al., 2003] several different representations for profiles that have been proposed and used in different approaches. We will however be using simple set models for all profiles. User profiles will be represented as *rating models* and *transaction models*, and item profiles will be represented as *rating models*, *transaction models* and *attribute models*.

### 2.3.1 User rating models

A user rating model consists of all items rated by the user. That is, if  $u$  is a user, then  $u_r$  is the (for simplicity assumed) non-empty set of items the user has rated and  $r_{ui}$  is the rating given to item  $i$  by the user. The number of items rated by the user is denoted using  $|u_r|$ .

### 2.3.2 User transaction models

A user transaction model consists of all items the user has purchased. That is, if  $u$  is a user then  $u_t$  is the (for simplicity assumed) non-empty set of items the user has purchased. The number of items purchased by the user is denoted using  $|u_t|$ .

### 2.3.3 Item rating models

A item rating model consists of all users that have rated the item. That is, if  $i$  is a item, then  $i_r$  is the (for simplicity assumed) non-empty set of users that have rated the item and  $r_{ui}$  is the rating given by user  $u$  to item  $i$ . The number of users that have rated the item is denoted using  $|i_r|$ .

### 2.3.4 Item transaction models

A item transaction model consists of all users that have purchased the item. That is, if  $i$  is a item then it is the (for simplicity assumed) non-empty set of users that have purchased the item. The number of users that have purchased the item is denoted using  $|i_t|$ .

### 2.3.5 Item attribute models

A item attribute model consists of all attribute of one type that a item has. E.g. movies have many different types of attributes, such as genres, directors, actors, release year, keywords etc. A item's genre attribute model thus consists of all the movies genres. That is, if  $i$  is an item, then  $i_{genre}$  is the non-empty set of all genres associated with the item. The number of genres that the item has is denoted using  $|i_{genre}|$ . The same notation applies to any attribute type.

## 2.4 Recommendation techniques

Recommendation techniques will be described using simple pseudo code that outlines how available profile models are used to generate recommendations.

For all the described techniques the common assumption is made that all the rating data uses the same rating scale  $[R_{min}, R_{max}]$ . It is always assumed that there is a set  $U = \{u_1, u_2, \dots\}$  and  $I = \{i_1, i_2, \dots\}$  consisting of all users and items respectively for which necessary profile models are available. In some cases it will also be assumed that there is a set  $T = \{t_1, t_2, \dots\}$  of attribute types. The profile models previously presented will be used.

The techniques that generate predictions outline how to predict ratings on all available items for all available users on items the users have not rated, this gives a better idea of the complexity of the algorithm than simply showing how to predict for the active user on the active item. The techniques do not outline nor discuss how to introduce new users or items into the system.

The techniques in addition to assuming the presence of required profile models also assume that necessary constants have been specified.

### 2.4.1 Baseline Statistical Filtering (BASELINESF)

Simple statistical recommendation technique for making predictions. The idea is to make use directly of the active user's rating model or the active item's rating model. Alternatively the whole populations opinion is used, however without forming any nearest neighborhoods or similar (collaborative filtering without neighborhoods). While rating data is assumed, it is not necessary for e.g. the random prediction algorithm, the ratings are then simply ignored, however the basic idea is that this baseline technique should serve as a statistical based baseline to compare other techniques against.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a item rating model  $i_r$ .

```

baselinesf(U,I)
1   $P \leftarrow \emptyset$ 
2  Foreach  $u \in U$ 
3       $P_u \leftarrow \emptyset$ 
4      Foreach  $i \in I$ 
5          If  $r_{ui} \notin u_r$  Then
6               $p \leftarrow \text{predict}(u, i)$ 
7               $\text{next}(P_u) \leftarrow (i, p)$ 
8  Return  $P$ 

```

On line 7 the notation  $\text{next}(n)$  is introduced, this is a standard notation meaning that a new element is added to the next free index in the list N. It is necessary that N is a list as we wish to be able to sort the list and as the prediction algorithm will need to be able to iterate over it in sorted order.

### User mean prediction algorithm

Uses the active user's rating mean as a prediction on all items for the active user.

```

predict(u,i)
1   $p \leftarrow \overline{u_r}$ 
2  Return  $p$ 

```

### Item mean prediction algorithm

Uses the active item's rating mean as a prediction for all users.

```

predict(u,i)
1   $p \leftarrow \overline{i_r}$ 
2  Return  $p$ 

```

### Random prediction algorithm

Makes a prediction by generating a uniformly random number in the rating interval. Multiple requests for a prediction on the same item for the same user may thus result in different predictions.

```

predict(u,i)
1   $p \leftarrow \text{random}(R_{min}, R_{max})$ 
2  Return  $p$ 

```

### Population deviation from mean prediction algorithm

Considers the active user's mean rating and the entire populations deviation from their mean rating on the active item when making a baseline prediction.

```

predict(u,i)
1   $p \leftarrow \overline{u_r} + \sum_{r_{ni} \in i_r} \frac{(r_{ni} - \overline{u_r})}{|i_r|}$ 
2  If  $p < R_{min}$  Then  $p \leftarrow R_{min}$ 
3  If  $p > R_{max}$  Then  $p \leftarrow R_{max}$ 
4  Return  $p$ 

```

### 2.4.2 User based Collaborative Filtering (UCF)

Recommendation technique based on the collaborative filtering approach. Predictions are based on a user neighborhood sorted in descending order of similarity with the active user. User similarities are calculated based on user rating models.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a item rating model  $i_r$ .

```

 $ucf(U, I)$ 
1   $P \leftarrow \emptyset$ 
2  Foreach  $u \in U$ 
3       $N \leftarrow \emptyset$ 
4      Foreach  $n \in U$ 
5          If  $n \neq u$  Then
6               $s \leftarrow \text{similarity}(u_r, n_r)$ 
7              If  $s \neq \text{Failed}$  Then
8                   $\text{next}(N) \leftarrow (n, s)$ 
9           $\text{sort}(N)$  in descending order of similarity
10      $P_u \leftarrow \emptyset$ 
11     Foreach  $i \in I$ 
12         If  $r_{ui} \notin u_r$  Then
13              $p \leftarrow \text{predict}(u, i, N)$ 
14              $\text{next}(P_u) \leftarrow (i, p)$ 
15 Return  $P$ 

```

The neighborhood  $N$  created on lines 2-8 is the heart of the technique. The technique when creating the set of all neighbors for the active user  $u$  will only ignore neighbors for which no similarity can be calculated by the similarity algorithm. The idea is that it is up to a prediction algorithm to select a suitable subset of neighbors to use when predicting, but since the nearest neighborhood of users can be selected in many different ways the decision of how to do this is left upon a prediction algorithm.

In the evaluation we used this technique together with a similarity algorithm using the common Pearson correlation coefficient as a similarity measure and a prediction algorithm using the common weighted deviation from mean prediction function.

#### Pearson similarity algorithm

The Pearson correlation coefficient is used to determine similarity between two user (or item) rating models  $u_r$  and  $n_r$ .

The constants  $OT$  (Overlap Threshold or "Corated" Threshold),  $ST$  (Significance Weighting Threshold) and  $MNS/MPS$  (Minimum Negative/Positive Similarity Threshold) specify different threshold required for the similarity calculation to succeed.  $OT$ ,  $MNS$  and  $MPS$  are not commonly used, but are shown here to demonstrate possible extensions to the similarity algorithm.  $ST$  is however commonly used and is recommended to be set between 20 to 50 [Herlocker, 00].

Similarity returned is a continuous value in the range  $[-1, +1]$  representing the similarity between two rating models.  $-1$  means perfect dissimilarity.  $0$  means no linear correlation existed (note that if correlation can't be calculated or thresholds aren't met, *Failed* is returned, not  $0$ ), it is still possible the two rating models are dissimilar or similar.  $+1$  means perfect similarity.

*similarity*( $u_r, n_r$ )

1 *Overlap*  $\leftarrow \{i | r_{ui} \in u_r \wedge r_{ni} \in n_r\}$

2 **If**  $|Overlap| < 2$  **Then Return Failed**

3 **If**  $|Overlap| < OT$  **Then Return Failed**

4

$$s \leftarrow \frac{\sum_{i \in Overlap} (r_{ui} \times r_{ni}) - \frac{\sum_{i \in Overlap} r_{ui} \times \sum_{i \in Overlap} r_{ni}}{|Overlap|}}{\sqrt{\left(\sum_{i \in Overlap} r_{ui}^2 - \frac{(\sum_{i \in Overlap} r_{ui})^2}{|Overlap|}\right) \times \left(\sum_{i \in Overlap} r_{ni}^2 - \frac{(\sum_{i \in Overlap} r_{ni})^2}{|Overlap|}\right)}}$$

5 **If**  $s = NaN$  **Then Return Failed**

6 **If**  $|Overlap| < ST$  **Then**  $s \leftarrow s \times \frac{|Overlap|}{s}$

7 **If**  $MNS < s$  **AND**  $s < MPS$  **Then Return Failed**

8 **Return**  $s$

The formula for calculating the Pearson correlation coefficient on line 4 uses no rating means and as such is suitable to use in any implementation of the formula as no separate mean calculation step for co-rated items is needed, the whole correlation coefficient can be calculated in one iteration over the set of co-rated items for the two users.

On line 5 there is a check whether the Pearson correlation coefficient is NaN, this check is needed as the denominator will sometimes be zero (occurs in extreme cases, such as when both users have rated 100 items, all with 5's).

### Weighted deviation from mean prediction algorithm

Neighbors weighted deviation from their mean rating is used when making a prediction. The constants  $NS_{min}$  (Minimum Neighborhood Size) and  $NS_{max}$  (Maximum Neighborhood Size) specify different thresholds required for the prediction to succeed. Note that the entire neighborhood is searched through for neighbors that have rated the active item, the algorithm does not simply consider e.g. the first  $NS_{max}$  neighbors when creating the neighborhood of predictors, this means that if the neighborhood is large enough then  $NS_{max}$  neighbors can usually be found, however they are most likely not always the neighbors with the highest similarity with the active user.

*predict*( $u, i, N$ )

1 **If**  $|N| < NS_{min}$  **Then Return Failed**

2  $NN \leftarrow \emptyset$

3 **ForEach**  $(n, s) \in N$

4     **If**  $r_{ni} \in n_r$

5          $next(NN) \leftarrow (n, s)$

6     **If**  $|NN| = NS_{max}$  **Then Break**

7 **If**  $|NN| = 0$  **Then Return Failed**

8 **If**  $|NN| < NS_{min}$  **Then Return Failed**

9  $p \leftarrow \bar{u}_r + \frac{\sum_{(n,s) \in NN} (r_{ni} - \bar{n}_r) \times s}{\sum_{(n,s) \in NN} |s|}$

10 **If**  $p = NaN$  **Then Return Failed**

11 **If**  $p < R_{min}$  **Then**  $p \leftarrow R_{min}$

12 **If**  $p > R_{max}$  **Then**  $p \leftarrow R_{max}$

13 **Return**  $p$

### 2.4.3 Random Neighbors User based Collaborative Filtering (RNUCF)

Recommendation technique based on the collaborative filtering approach. Predictions are based on a user neighborhood sorted in random order. No user similarities are calculated, instead all neighbors are

assigned similarity 1.0 with the active user. The technique was only implemented in order to evaluate how the prediction algorithms used for the UCF technique work on a randomly sorted neighborhood. While we could have simply implemented prediction algorithms that creates a random nearest neighborhood, we chose to simply implement a second technique for the purpose. Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a item rating model  $i_r$ .

```

rnucf( $U, I$ )
1   $P \leftarrow \emptyset$ 
2  Foreach  $u \in U$ 
3       $N \leftarrow \emptyset$ 
4      Foreach  $n \in U$ 
5          If  $n \neq u$  Then
6               $next(N) \leftarrow (n, 1.0)$ 
7           $permute(N)$  randomly
8       $P_u \leftarrow \emptyset$ 
9      Foreach  $i \in I$ 
10         If  $r_{ui} \notin u_r$  Then
11              $p \leftarrow predict(u, i, N)$ 
12              $next(P_u) \leftarrow (i, p)$ 
13  Return  $P$ 

```

#### 2.4.4 Trust User based Collaborative Filtering (TRUSTUCF)

Recommendation technique based on the collaborative filtering approach. Predictions are based on a user neighborhood sorted in descending order of trust. Trust in a neighbor is a combination of the neighbor's similarity with the active user and the neighbors trusted ability to predict for the active item. User similarities are calculated based on user rating models. Users that have rated many of the items that are most similar to the active item are trusted to be able to predict reliably for it. Item similarities are calculated based on item rating models.

The constant  $MS$  (Model Size) specifies the size of the item neighborhoods to keep in memory. When trust is calculated for a neighbor, the trust will be based on how many of the  $MS$  most similar items to the active item the neighbor has rated.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a item rating model  $i_r$ .

```

trustucf( $U, I$ )
1   $M \leftarrow \text{item-models}(I)$ 
2   $P \leftarrow \emptyset$ 
3  Foreach  $u \in U$ 
4       $N \leftarrow \text{user-neighborhood}(u, U)$ 
5      Foreach  $i \in I$ 
6          If  $r_{ui} \neq u_r$  Then
7               $TN \leftarrow \text{trust-neighborhood}(i, N, M)$ 
8               $p \leftarrow \text{predict}(u, i, TN)$ 
9               $\text{next}(P_u) \leftarrow (i, p)$ 
10 Return  $P$ 

item-models( $I$ )
11  $M \leftarrow \emptyset$ 
12 Foreach  $i \in I$ 
13      $N \leftarrow \emptyset$ 
14     Foreach  $j \in I$ 
15         If  $i \neq j$  Then
16              $s \leftarrow \text{similarity}(i_r, j_r)$ 
17             If  $s \neq \text{Failed}$  Then
18                  $\text{next}(N) \leftarrow (j, s)$ 
19      $\text{sort}(N)$  in descending order of similarity
20      $M_i \leftarrow \text{range}(N, 1, MS)$ 
21 Return  $M$ 

user-neighborhood( $u, U$ )
22  $N \leftarrow \emptyset$ 
23 Foreach  $n \in U$ 
24     If  $n \neq u$  Then
25          $s \leftarrow \text{similarity}(u_r, n_r)$ 
26         If  $s \neq \text{Failed}$  Then
27              $\text{next}(N) \leftarrow (n, s)$ 
28 Return  $N$ 

trust-neighborhood( $i, N, M$ )
29  $TN \leftarrow \emptyset$ 
30 Foreach  $(n, s) \in N$ 
31      $t \leftarrow 0$ 
32     Foreach  $(j, -) \in M_i$ 
33         If  $r_{nj} \in n_r$  Then
34              $t++$ 
35      $t \leftarrow \frac{t}{MS}$ 
36      $s \leftarrow s \times t$ 
37      $\text{next}(TN) \leftarrow (n, s)$ 
38  $\text{sort}(TN)$  in descending order of similarity
39 Return  $TN$ 

```

As can be seen on line 1 a model step is introduced in which item similarities for all items are precalculated, the  $MS$  most similar items for each item are saved. This step is necessary as it would be too costly to recalculate them each time they are needed. Note that on line 20 the  $\text{range}(L, S, E)$  method is introduced, this method simply extracts the elements on index  $S$  to  $E$  in the list  $L$ , additionally is

$E < S$  the whole list is returned.

On line 4 a regular user neighborhood  $N$  is calculated as in e.g. the UCF technique, for clarity it has been placed in a separate method. However unlike the UCF technique, the neighborhood will not be passed directly to the prediction algorithm. Instead, once an active item has been selected the neighborhood will on line 7 be passed to a method that modifies the neighbor similarities based on how much each neighbor is trusted to be able to predict for the active item. A neighbor that e.g. has not rated any item similar to the active is not trusted to be able to predict for the item at all and will have the similarity down weighted to zero.

#### 2.4.5 Common Items Prioritizing User based Collaborative Filtering (CIPUCF)

Recommendation technique based on the collaborative filtering approach. Predictions are based on a user neighborhood sorted in descending order of common items with the active user. User similarities are still calculated with the active user, however the neighborhood is unlike the UCF technique not sorted based on the similarities. User similarities are calculated based on user rating models.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a item rating model  $i_r$ .

```

cipucf( $U, I$ )
1   $P \leftarrow \emptyset$ 
2  ForEach  $u \in U$ 
3       $N \leftarrow \emptyset$ 
4      ForEach  $n \in U$ 
5          If  $n \neq u$  Then
6               $s \leftarrow \text{similarity}(u_r, n_r)$ 
7              If  $s \neq \text{Failed}$  Then
8                   $\text{next}(N) \leftarrow (n, s)$ 
9       $\text{sort}(N)$  in descending order of common items
10      $P_u \leftarrow \emptyset$ 
11     ForEach  $i \in I$ 
12         If  $r_{ui} \notin u_r$  Then
13              $p \leftarrow \text{predict}(u, i, N)$ 
14              $\text{next}(P_u) \leftarrow (i, p)$ 
15 Return  $P$ 

```

Line 9 will keep the previously calculated similarity between each neighbor and the active user, but the members of list  $N$  will be sorted according to the number of items they have in common with the active user (the step is thus quite simplified).

#### 2.4.6 Item based Collaborative Filtering (ICF)

Recommendation technique based on the collaborative filtering approach. Predictions are based on a item neighborhood sorted in descending order of similarity with the active item. Item similarities are calculated based on item rating models.

The constant  $MS$  (Model Size) specifies the size of the item neighborhoods to keep in memory. The basic assumption is that item similarities, unlike user similarities in the UCF technique, are not likely to change as often/much as user similarities, hence a model can be build of item similarities.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a item rating model  $i_r$ .



```

 $icf(U, I)$ 
1   $M \leftarrow \emptyset$ 
2  Foreach  $i \in I$ 
3       $N \leftarrow \emptyset$ 
4      Foreach  $j \in I$ 
5          If  $i \neq j$  Then
6               $s \leftarrow \text{similarity}(i_r, j_r)$ 
7              If  $s \neq \text{Failed}$  Then
8                   $\text{next}(N) \leftarrow (j, s)$ 
9           $\text{sort}(N)$  in descending order of similarity
10      $M_i \leftarrow \text{range}(N, 1, MS)$ 
11   $P \leftarrow \emptyset$ 
12  Foreach  $u \in U$ 
13       $P_u \leftarrow \emptyset$ 
14      Foreach  $i \in I$ 
15          If  $r_{ui} \notin u_r$  Then
16               $p \leftarrow \text{predict}(u, i, M_i)$ 
17               $\text{next}(P_u) \leftarrow (i, p)$ 
18  Return  $P$ 

```

Keep in mind that the model size  $MS$  of the item similarity models  $M_i$  should be kept relatively small, If we assume that the majority of users have rated very few items, suppose around 20. If a excessively large item model size is then used, the prediction algorithm might (assuming it scans the entire item neighborhood for items the active user has rated) end up basing all predictions on the active users ratings on the same 20 items, simply because those 20 items are included in every single item's neighborhood. While those 20 items will have varying similarity with the active item, it is still important to keep this fact in mind, especially if the prediction algorithm ignores the item similarities (e.g. the Average Rating prediction algorithm).

### Adjusted cosine similarity algorithm

The cosine angle is used as a similarity measure between two item rating models  $i_r$  and  $j_r$ , the cosine formula is applied to users deviations from their rating means instead of to their actual ratings. While this algorithm is expressed in terms of item rating models it works for user rating models as well, however [Sarwar et al., 01] considers the adjusted cosine formula more suitable for item rating models and suggests using Pearson for user rating models.

The constants  $OT$  (Overlap Threshold),  $ST$  (Significance Weighting Threshold) and  $MS$  (Minimum Similarity Threshold) specify different threshold required for the similarity calculation to succeed.  $OT$  and  $MS$  are not commonly used, but are shown here to demonstrate possible extensions to the similarity algorithm.  $ST$  is however just as applicable here as for the Pearson similarity algorithm.

Similarity returned is a continuous value in the range  $[-1, +1]$  representing the similarity between the two item rating models. Interpretation of similarities is similar to that of the Pearson similarity algorithm,  $-1$  means perfect dissimilarity,  $+1$  perfect similarity and 0 that nothing certain could be said about either case.

*similarity*( $i_r, j_r$ )

```

1  If  $|i_r| = 0$  OR  $|j_r| = 0$  Then Return Failed
2   $s \leftarrow \frac{\sum_{r_{ui} \in i_r} (r_{ui} - \bar{u}_r) + \sum_{r_{uj} \in j_r} (r_{uj} - \bar{u}_r)}{\sqrt{\sum_{r_{ui} \in i_r} (r_{ui} - \bar{u}_r)^2 \times \sum_{r_{uj} \in j_r} (r_{uj} - \bar{u}_r)^2}}$ 
3  If  $s = NaN$  Then Return Failed
4   $Overlap \leftarrow \{i | r_{ui} \in u_r \wedge r_{ni} \in n_r\}$ 
5  If  $|Overlap| < ST$  Then  $s \leftarrow s \times \frac{|Overlap|}{ST}$ 
6  If  $s < MS$  Then Return Failed
7  Return  $s$ 

```

### Average rating prediction algorithm

Calculates the active user's average rating on items that are similar to the active item. The average is used as a prediction. Unlike the baseline prediction algorithm that uses the active user's mean rating, this prediction algorithm uses a subset of the items the user has rated. The subsets consists of the items the active user has rated that are most similar to the active item.

The constants  $NS_{min}$  (Minimum Neighborhood Size) and  $NS_{max}$  (Maximum Neighborhood Size) specify different thresholds required for the prediction calculation to succeed.

Predictions are in the range  $[R_{min}, R_{max}]$ .

*predict*( $u, i, N$ )

```

1  If  $|N| < NS_{min}$  Then Return Failed
2   $NN \leftarrow \emptyset$ 
3  Foreach  $(j, s) \in N$ 
4      If  $r_{uj} \in u_r$  Then
5           $next(NN) \leftarrow (j, s)$ 
6      If  $|NN| = NS_{max}$  Then Break
7  If  $|NN| = 0$  OR  $|NN| < NS_{min}$  Then Return Failed
8   $p \leftarrow \frac{\sum_{(j,s) \in NN} r_{uj}}{|NN|}$ 
9  Return  $p$ 

```

### Weighted sum prediction algorithm

Uses the active user's ratings on items that are most similar to the active item when making a prediction. The active user's ratings on items are weighted using the items similarity with the active item.

The constants  $NS_{min}$  (Minimum Neighborhood Size) and  $NS_{max}$  (Maximum Neighborhood Size) specify different thresholds required for the prediction calculation to succeed.

Predictions are in the range  $[R_{min}, R_{max}]$ .

*predict*( $u, i, N$ )

```

1  If  $|N| < NS_{min}$  Then Return Failed
2   $NN \leftarrow \emptyset$ 
3  Foreach  $(j, s) \in N$ 
4      If  $r_{uj} \notin u_r$  Then  $next(NN) \leftarrow (j, s)$ 
5      If  $|NN| = NS_{max}$  Then Break
6  If  $|NN| = 0$  OR  $|NN| < NS_{min}$  Then Return Failed
7   $p \leftarrow \frac{\sum_{(j,s) \in NN} r_{uj} \times s}{\sum_{(j,s) \in NN} |s|}$ 
8  If  $p = NaN$  Then Return Failed
9  If  $p < R_{min}$  Then  $p \leftarrow R_{min}$ 
10 Return  $p$ 

```

### 2.4.7 Random Neighbors Item based Collaborative Filtering (RNICF)

Recommendation technique based on the content based filtering approach. Predictions are based on a item neighborhood for the active item sorted in random order. No item similarities are calculated.

The constant  $MS$  (Model Size) specifies the size of the item neighborhoods to keep in memory.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a item rating model  $i_r$ .

```

 $rnidf(U, I)$ 
1   $M \leftarrow \emptyset$ 
2  Foreach  $i \in I$ 
3       $N \leftarrow \emptyset$ 
4      Foreach  $j \in I$ 
5          If  $i \neq j$  Then
6               $next(N) \leftarrow (j, 1.0)$ 
7           $permute(N)$  randomly
8           $M_i \leftarrow range(N, 1, MS)$ 
9   $P \leftarrow \emptyset$ 
10 Foreach  $u \in U$ 
11      $P_u \leftarrow \emptyset$ 
12     Foreach  $i \in I$ 
13         If  $r_{ui} \notin u_r$  Then
14              $p \leftarrow predict(u, i, M_i)$ 
15              $next(P_u) \leftarrow (i, p)$ 
16 Return  $P$ 

```

### 2.4.8 Personalized Content Based Filtering (PCBF)

Recommendation technique based on the content based approach. Predictions are based on a item neighborhood sorted in descending order of similarity with the active item. Item similarities are calculated based on item attributes. An item has one or more attribute type models, the total item similarity is a normalized weighted linear sum over all attribute models.

The ICF and PCBF techniques are similar, essentially only differing in how item similarities are calculated, however the fact the PCBF technique doesn't require any ratings on all items it predicts for (it is enough that the active user has rated some of the items similar to the active item) means a possibly increased coverage as long as attribute models are available for all items.

The constant  $MS$  (Model Size) specifies the size of the item neighborhoods to keep in memory.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a item rating model  $i_r$ . Assume that each item has for each attribute type  $t \in T$  a attribute type model  $i_t$ . Assume that for each attribute type  $t$  there is a weight  $w_t$  and a similarity measure  $sim_t$ .

```

 $pcbf(U, I, T)$ 
1   $w_{tot} \leftarrow \sum_{t \in T} w_t$ 
2  Foreach  $t \in T$ 
3       $w_t \leftarrow \frac{w_t}{w_{tot}}$ 
4   $M \leftarrow \emptyset$ 
5  Foreach  $i \in I$ 
6       $N \leftarrow \emptyset$ 
7      Foreach  $j \in I$ 
8          If  $i \neq j$  Then
9               $s \leftarrow 0$ 
10             Foreach  $t \in T$ 
11                  $s \leftarrow s + sim_t(i_t, j_t) \times w_t$ 
12             If  $s \neq Failed$  Then
13                  $next(N) \leftarrow (j, s)$ 
14              $sort(N)$  in descending order of similarity
15              $M_i \leftarrow range(N, 1, MS)$ 
16  $P \leftarrow \emptyset$ 
17 Foreach  $u \in U$ 
18      $P_u \leftarrow \emptyset$ 
19     Foreach  $i \in I$ 
20         If  $r_{ui} \notin u_r$  Then
21              $p \leftarrow predict(u, i, M_i)$ 
22              $next(P_u) \leftarrow (i, p)$ 
23 Return  $P$ 

```

#### 2.4.9 Common attribute similarity algorithm

Defines similarity between two item attribute models as the ratio of shared attributes, against the total number of distinct attributes, between the two item attribute models.

The constants *OT* (Overlap Threshold), *ST* (Significance Weighting Threshold) and *MS* (Minimum Similarity Threshold) specify different threshold required for the similarity calculation to succeed. *OT* and *MS* are not commonly used, but are shown here to demonstrate possible extensions to the similarity algorithm.

Similarity returned is a continuous value in the range  $[0, 1]$ . 0 means no similarity and 1 means perfect similarity.

```

 $similarity(i_a, j_a)$ 
1   $Overlap \leftarrow i_a \cap j_a$ 
2  If  $|Overlap| < OT$  Then Return Failed
3   $Distinct \leftarrow i_a \cup j_a$ 
4  If  $|Distinct| = 0$  Then Return Failed
5   $s \leftarrow \frac{|Overlap|}{|Distinct|}$ 
6  If  $|Overlap| < ST$  Then  $s \leftarrow s \times \frac{|Overlap|}{ST}$ 
7  If  $s < MS$  Then Return Failed
8  Return  $s$ 

```

#### Cosine similarity algorithm

The cosine angle is used as a similarity measure between two item attribute models. Since attribute models are used the original cosine formula can be simplified significantly (as shown on line 4).

The constants  $OT$  (Overlap Threshold),  $ST$  (Significance Weighting Threshold) and  $MS$  (Minimum Similarity Threshold) specify different threshold required for the similarity calculation to succeed.  $OT$  and  $MS$  are not commonly used, but are shown here to demonstrate possible extensions to the similarity algorithm.

Similarity returned is a continuous value in the range  $[0, 1]$ . 0 means no similarity and 1 means perfect similarity.

```

similarity( $i_a, j_a$ )
1   $Overlap \leftarrow i_a \cap j_a$ 
2  If  $|Overlap| < 2$  Then Return Failed
3  If  $|Overlap| < OT$  Then Return Failed
4   $s \leftarrow \frac{Overlap}{\sqrt{|i_a| \times |j_a|}}$ 
5  If  $s = NaN$  Then Return Failed
6  If  $|Overlap| < ST$  Then  $s \leftarrow s \times \frac{|Overlap|}{ST}$ 
7  If  $s < MS$  Then Return Failed
8  Return  $s$ 

```

#### Same decade similarity algorithm

Assumes two item attribute models contain as first and only element a integer representing a year, the two attribute models are considered similar if they belong to the same decade. Since this is the movie domain, such a similarity measure makes sense, movies belonging to different decades can very roughly be seen to often have different characteristics.

Similarity returned is 1 if the two item attribute models belong to the same decade (in the same year) or 0 if they're from different decades.

```

similarity( $i_a, j_a$ )
1  If  $|i_a| \neq 1$  OR  $|j_a| \neq 1$  Then Return Failed
2  If  $same-decade(first-element(i_a), first-element(j_a))$  Then Return 1.0
3  Return 0.0

```

#### 2.4.10 Top-N User based Collaborative Filtering (TOPNUCF)

Recommendation technique makes predictions for users using the UCF recommendation technique and creates Top-N ranking lists by sorting the predictions for each user in descending order of prediction and recommending the Top-N items with highest predictions.

#### 2.4.11 Top-N Item based Collaborative Filtering (TOPNICF)

Recommendation technique makes predictions for users using the ICF recommendation technique and creates Top-N ranking lists by sorting the predictions for each user in descending order of prediction and recommending the Top-N items with highest predictions.

#### 2.4.12 Top-N Personalized Content Based Filtering (TOPNPCBF)

Recommendation technique makes predictions for users using the PCBF recommendation technique and creates Top-N ranking lists by sorting the predictions for each user in descending order of prediction and recommending the Top-N items with highest predictions.

#### 2.4.13 Top-N Content Based Filtering (TOPNCBF)

Recommendation technique makes Top-N ranking list recommendations based on item similarities. Item similarities are calculated based on attribute models.

The constant  $MS$  (Model Size) specifies the size of the item neighborhoods to keep in memory. The constant  $TN$  specifies the size of the Top-N ranking lists to recommend.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a transaction model it. Assume that each item has for each attribute type  $t \in T$  a attribute type model it. Assume that for each attribute type  $t$  there is a weight  $w_t$  and a similarity measure  $sim_t$ .

```

topncbf( $U, I, T$ )
1   $w_{tot} \leftarrow \sum_{t \in T} w_t$ 
2  Foreach  $t \in T$ 
3       $w_t \leftarrow \frac{w_t}{w_{tot}}$ 
4   $M \leftarrow \emptyset$ 
5  Foreach  $i \in I$ 
6       $N \leftarrow \emptyset$ 
7      Foreach  $j \in I$ 
8          If  $i \neq j$  Then
9               $s \leftarrow 0$ 
10             Foreach  $t \in T$ 
11                  $s \leftarrow s + sim_t(i_t, j_t) \times w_t$ 
12             If  $s \neq Failed$  Then
13                  $next(N) \leftarrow (j, s)$ 
14              $sort(N)$  in descending order of similarity
15              $M_i \leftarrow range(N, 1, MS)$ 
16   $R \leftarrow \emptyset$ 
17  Foreach  $u \in U$ 
18       $R \leftarrow ranking-list(u, M)$ 
19       $R_u \leftarrow range(R, 1, TN)$ 
20  Return  $R$ 

```

#### Similar items ranking list algorithm

Ranking list is generated by considering items with all items purchased by the active user.

```

ranking-list( $u, M$ )
1   $R \leftarrow \emptyset$ 
2  Foreach  $i \in u_t$ 
3      Foreach  $(i, s) \in M_i$ 
4          If  $R[i] = NULL$  Then
5               $R[i] = s$ 
6          Else
7               $R[i] += s$ 
8   $sort(R)$  in descending order of similarity
9  Return  $R$ 

```

#### 2.4.14 Transaction based Top-N User based Collaborative Filtering (TBTOP-NUCF)

Recommendation technique makes Top-N ranking list recommendations based on a user neighborhood sorted in descending order of similarity with the active user. User similarities are calculated based on user transaction models.

The constant  $TN$  specifies the size of the Top-N ranking lists to recommend.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a transaction model it. Assume that each item has for each attribute type  $t \in T$  a attribute type model it. Assume that for each attribute type  $t$  there is a weight  $w_t$  and a similarity measure  $sim_t$ .

```

tbtownucf( $U, I, T$ )
1   $R \leftarrow \emptyset$ 
2  Foreach  $u \in U$ 
3       $N \leftarrow \emptyset$ 
4      Foreach  $n \in U$ 
5          If  $u \neq n$  Then
6               $s \leftarrow similarity(u_t, n_t)$ 
7              If  $s \neq Failed$  Then
8                   $next(N) \leftarrow (n, s)$ 
9           $sort(N)$  in descending order of similarity
10      $R \leftarrow ranking-list(u, N)$ 
11      $R_u \leftarrow range(R, 1, TN)$ 
12 Return  $R$ 

```

#### Cosine similarity algorithm (Simplified transaction case)

The cosine angle is used as a similarity measure between two transaction models. The similarity algorithm is the same as for attribute models, except it works with transaction models.

#### Frequent items ranking list algorithm

Ranking list is generated by considering item frequencies among neighbors of the active user.

The constant  $NS$  (Neighborhood Size) specifies the (exact) number of neighbors of the active user to consider.

```

ranking-list( $u, N$ )
1  If  $|N| < NS$  Then Return Failed
2   $NN \leftarrow range(N, 1, NS)$ 
3   $R \leftarrow \emptyset$ 
4  Foreach  $n \in NN$ 
5      Foreach  $i \in n_t$ 
6          If  $R[i] = NULL$  Then  $R[i] = 1$ 
7          Else  $R[i]++$ 
8   $sort(R)$  in descending order of similarity
9  Return  $R$ 

```

#### 2.4.15 Transaction based Top-N Item based Collaborative Filtering (TBTOP-NICF)

Recommendation technique makes Top-N ranking list recommendations based on a item neighborhood sorted in descending order of similarity with the active item. Item similarities are calculated based on item transaction models.

The constant  $MS$  (Model Size) specifies the size of the item neighborhoods to keep in memory. The constant  $TN$  specifies the size of the Top-N ranking lists to recommend.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a transaction model it. Assume that each item has for each attribute type  $t \in T$  a attribute type model it. Assume that for each attribute type  $t$  there is a weight  $w_t$  and a similarity measure  $sim_t$ .

```

tbtopnicf( $U, I, T$ )
1   $M \leftarrow \emptyset$ 
2  Foreach  $i \in I$ 
3       $N \leftarrow \emptyset$ 
4      Foreach  $j \in I$ 
5          If  $i \neq j$  Then
6               $s \leftarrow \text{similarity}(i_t, j_t)$ 
7              If  $s \neq \text{Failed}$  Then
8                   $\text{next}(N) \leftarrow (j, s)$ 
9           $\text{sort}(N)$  in descending order of similarity
10      $M_i \leftarrow \text{range}(N, 1, MS)$ 
11   $R \leftarrow \emptyset$ 
12  Foreach  $u \in U$ 
13       $R \leftarrow \text{ranking-list}(u, N)$ 
14       $R_u \leftarrow \text{range}(R, 1, TN)$ 
15  Return  $R$ 

```

## 2.5 Evaluation of recommender systems

*“Much theorizing has gone into optimum cell size. I think that history shows that a cell of three is best - more than three can’t agree on when to have dinner”* – Professor Bernardo de La Paz

Recommender systems work with datasets to produce recommendations. Recommender systems that produce personalized recommendations always work with user datasets, such a dataset in current research consists of either user ratings or user “transactions” (information about which items a user has considered relevant). Evaluations are made of such recommender systems using all or parts of available user data as evaluation data, which is split into training and test data, and any of an assortment of suitable evaluation metrics. An evaluation metric for a recommender system is currently focused either on *accuracy* of predictions, or *relevance* of Top-N ranking lists made by the recommender system [Herlocker, 2000]. In either case two simplifying assumptions are made, upon which all evaluations of the recommender system are based:

1. Given any training and test data, if the recommender system is able to accurately predict, based on the training data, ratings for items in the test data, then we assume the recommender system will also predict correctly for new unknown data.
2. Given any training and test dataset, if the Top-N list generated by the recommender system contains items from the test data, then the Top-N list is better than a Top-N list not containing any items from the test data.

Both of these assumptions are flawed, if the recommender system predicts badly for the test data it may still predict perfectly for future data. And if a Top-N list contains no relevant items, it may still be the case all items are relevant.

The user data available is not complete, and typically can not be complete. Hence no evaluation metric can give a true measure of how accurate the recommender system is on all future data. The evaluation metrics however give a true measure of how accurate the recommender system is on a specific evaluation dataset. Whether or not this accuracy holds for any future data is not guaranteed, and is not measured either.

In order to reliably evaluate an recommender system on a given evaluation dataset, a sensible and motivated choice of forming training and test datasets is highly necessary as well as a reliable evaluation protocol.

If evaluation results are to be compared against other evaluation results, it is necessary to make sure the evaluation datasets and evaluation protocols used for both evaluations are similar (preferably



identical), otherwise comparisons and claims of better or worse results should not be made or at the very least be made very carefully.

The evaluation metrics used for evaluating recommender systems are based on well known statistical measures from data mining and information filtering and retrieval. However while it helps to claim the soundness of the measures by referring to the field they come from, it is even more helpful to put them in their new situation and therein analyze what they are measuring, and whether it is meaningful in this new situation.

Evaluation of recommender systems is difficult and requires a great deal of thought before performed. In this thesis, only commonly evaluation metrics are outlined.

For further discussion and more evaluation metrics (such as ROC curves) as well as an comparison of individual evaluation metrics, please refer to [Herlocker, 2000].

### 2.5.1 Evaluation data

Given a user dataset, assuming for discussion it consists of ratings (the same reasoning apply to transaction user datasets), there are at least four factors that can be taken into account when forming the evaluation dataset:

1. Number of users
2. Number of items
3. Number of ratings per user
4. Number of ratings per item

The evaluation data needs to contain enough users and enough items for the evaluation to be meaningful. Too few users or too few items and the evaluation becomes biased towards the size of the data set. Requiring a minimum number of ratings per user is typically sensible, since any recommender system is unlikely to be able to predict reliably for a user before it has some minimum number of ratings for the user. Depending on the recommender system it might also make sense to require a minimum number of ratings per item in the evaluation data. Including such users or items, with too few ratings, in the evaluation data means that the evaluation result can never be perfect. As such the requirement, of a user to have rated at least enough items and a item to have at least enough ratings for a recommendation to be at all possible, might often be a motivated and sensible requirement of the evaluation data. The consensus being that it is not interesting to evaluate a recommender system using data it is expected to fail on, this can be stated separately.

### 2.5.2 Evaluation protocols

Once the evaluation data has been prepared, a evaluation protocol must be selected. Three common evaluation protocols are:

1. Hold out: A percentage of the evaluation data is used as training data while the remainder is left out and used as test data.
2. All but 1: One element of the evaluation data is used as test data, maximizing the amount of data used for training data.
3. K-fold cross validation: The evaluation data is split K times into training and test data, such that all data is used exactly once as test data.

Of these K-fold cross validation intuitively *seems* most sensible, depending on the evaluation metrics hold out or all but 1 may be most suited for each of the k cross validations. Variations of k-fold cross validation with all but 1 may also be sensible, e.g. where k times different ratings are used for test data, but not necessarily all ratings are used for test data.

### 2.5.3 Splitting evaluation data

The split of evaluation data into training and test data is commonly done in either of the following ways:

1. Split over users: A percentage or fixed number of each user's ratings are used as test data and the remainder as training data.
2. Split over items: A percentage or fixed number of each item's ratings are used as test data and the remainder as training data.
3. Split over ratings: A percentage or fixed number of the ratings (independent of who or what has been rated) is used as test data, and the remainder as training data.

Care should be taken whether it is sensible to allow test data to contain items that no user in the training data has rated, or users that do not occur in the training data. The first two approaches ensures respectively either that a user or a item occurs in both training and test data. The third approach allows for users and items to end up in the test data that do not occur in the training data, however no bias towards users or items to occur in both training and test data is introduced.

### 2.5.4 Prediction evaluation metrics

The prediction evaluation metrics measure the RS ability to predict correctly known user ratings. The evaluation metrics use known correct ratings from a test data set and compare those against predictions made by a RS based on a training data set.

The predictions depends on the RS. However, since the evaluation requires a evaluation dataset consisting of ratings, the predictions must be in the same range as the ratings in the user dataset. Otherwise it will not be possible to compare the predictions against the known correct ratings in the test dataset.

Note that the ratings and predictions are assumed to be continuous values in the interval specified by  $[R_{min}, R_{max}]$  where  $R_{min}$  and  $R_{max}$  are the lowest and highest rating a item can be assigned by a user. If the actual rating scale is discrete (which it usually is), the predictions can be rounded to closest integer, to get results that more accurately reflects the user experience.

Unless otherwise stated, evaluation metrics are intended for use with continuous predictions within the rating range.

In addition to reporting the metrics described in this section, it is important to also report how many users could be evaluated (successful users) and how many users could not be evaluated (failed user) and also how many of the made predictions succeeded and how many failed [Table 2.1].

$U_{successful}$	Number of users evaluated (which does not include users for which no predictions could be made).
$U_{failed}$	Number of users for which no prediction could be made.
$P_{successful}$	Number of successful predictions (as in a prediction was possible to make) evaluated.
$P_{failed}$	Number of failed predictions (as in no prediction could be made).

Table 2.1: Additional prediction evaluation metrics.

Additionally a confusion matrix (see example explanation in [Figure 2.1]) showing distribution of predictions per known rating, and a relevance matrix (see example explanation in [Figure 2.2]) showing true/false positives and negatives can be generated and reported (and may often provide a more detailed understanding of the quality of a technique, moreso than a single numeric evaluation metric which however usually is much easier to report and compare than these matrices).

	P=1	P=2	P=3	P=4	P=5	$P_{failed}$
R=1	72	308	414	274	43	47
R=2	27	325	1036	504	32	48
R=3	23	357	2687	2713	177	99
R=4	11	155	1964	6053	883	96
R=5	9	71	470	3739	2462	94

Figure 2.1: Confusion matrix. Whenever a successful prediction P was made for an item with a known rating R, it is counted how often a correct rating was predicted and how often a incorrect rating (and what it was) was predicted. For example, 2462 predictions of 5 were made when the known rating on the item was 5, and only 9 predictions made were a 1 when the known rating was 5.

	TP	FP	TN	FN
T=2	23434	939	172	264
T=3	19640	1590	1445	2134
T=4	8962	1424	7568	6855
T=5	109	36	18022	6642

Figure 2.2: Relevance matrix. For different relevance thresholds T it is shown how many True Positives (TP), False Positives (FP), True Negatives (TN) and False negatives (FN) that were made. If for example T=3 it means that a item with a known rating of 3 or higher which is predicted a rating of 3 or higher is a TP, whereas had it been predicted something below 3 it would have been a FN. Similarly a item with a known rating below 3 which is predicted a rating above or equal to 3 is a FP, and had it been predicted a rating below 3 it would have been a TN.

### Coverage

$$Coverage = \frac{P_{successful}}{P_{successful} + P_{failed}} \quad (2.1)$$

Where

$P_{successful}$  is the number of successful predictions (as in a prediction was possible to make) evaluated.

$P_{failed}$  is the number of failed predictions (as in no prediction could be made).

Keep in mind that it is easy to get 100% coverage, simply implement a RS that makes "default" predictions using some baseline technique (average item rating, average user rating or similar) whenever the "main" prediction fails, however accuracy measures such as MAE may then suffer. While a default prediction is a successful prediction, it is usually more interesting to measure coverage that does not include default predictions

### Average User Coverage

$$Coverage_{AU} = \frac{\sum U_{coverage}}{U_{successful}} \quad (2.2)$$

Where

$\sum U_{coverage}$  is the sum over all user's individual coverage, each user's coverage is calculated as the number of successful predictions made for the user divided by the total number of predictions attempted to be made for the user (e.g. the sum of the number of successful predictions and number of failed predictions for the user).

$U_{successful}$  is the number of users evaluated (which does not include users for which no predictions could be made).

### Mean Absolute Error (MAE)

$$MAE = \frac{P_{error}}{P_{successful}} \quad (2.3)$$

Where

$P_{error}$  is the sum of the error for all evaluated predictions, the error in the evaluated predictions is calculated as the absolute value of the difference between the predicted rating PR and the actual rating AR, e.g.  $|AR - PR|$ .

$P_{successful}$  is the number of successful predictions (as in a prediction was possible to make) evaluated.

Keep in mind the coverage, when coverage goes down (e.g. number of failed predictions increase) less successful predictions will be involved in the MAE calculation, significantly skewing results for the whole test set if coverage is not considered.

$MAE_R$  using rounded ratings and predictions can also be calculated to better evaluate the user experience.

### Average User MAE

$$MAE_{AU} = \frac{U_{MAE}}{U_{successful}} \quad (2.4)$$

Where

$\sum U_{MAE}$  is the sum over all user's individual MAE, each user's MAE is calculated as the absolute value of the difference between the predicted rating and the actual rating divided with the number of predictions made for the user.

$U_{successful}$  is the number of users evaluated (which does not include users for which no predictions could be made).

$MAE_{RAU}$  using rounded ratings and predictions can also be calculated to better evaluate the user experience.

### Normalized Mean Absolute Error (NMAE)

$$NMAE = \frac{MAE}{R_{max} - R_{min}} \quad (2.5)$$

The MAE for the predictions set is divided with the maximum possible prediction error in order to get a number between 0 and 1 where 0 means all predictions were correct and 1 means all predictions were as wrong as they could possibly be.

$R_{max} - R_{min}$  gives the maximum possible MAE, as no prediction error can be larger than the distance between the largest  $R_{max}$  and smallest  $R_{min}$  ratings on the rating scale. E.g. suppose actual rating is smallest possible rating 1 and prediction made is largest possible rating 5, then the prediction error is 4 which is also the maximum possible prediction error.

NMAE using rounded ratings and predictions can also be calculated to better evaluate the user experience.

### Average User NMAE

$$NMAE_{AU} = \frac{U_{NMAE}}{U_{successful}} \quad (2.6)$$

Where

$\sum U_{NMAE}$  is the sum over all user's individual NMAE, each user's NMAE is calculated as the user's individual MAE divided by the maximum possible prediction error.

$U_{successful}$  is the number of users evaluated (which does not include users for which no predictions could be made).

$NMAE_{RAU}$  using rounded ratings and predictions can also be calculated to better evaluate the user experience.

### Mean Squared Error (MSE)

$$MSE = \frac{P_{error}^2}{P_{successful}} \quad (2.7)$$

Where

$P_{error}^2$  is the sum of the squared prediction error for all evaluated predictions.

$P_{successful}$  is the number of successful predictions (as in a prediction was possible to make) evaluated.

### Average User MSE

$$MSE = \frac{\sum U_{MSE}}{U_{successful}} \quad (2.8)$$

Where

$\sum U_{MSE}$  is the sum over all user's individual MSE, each user's MSE is calculated as the user's individual MAE squared.

$U_{successful}$  is the number of users evaluated (which does not include users for which no predictions could be made).

MSE using rounded ratings and predictions can also be calculated to better evaluate the user experience.

### Root Mean Squared Error (RMSE)

$$RMSE = \sqrt{MSE} \quad (2.9)$$

The square root of the MSE for the prediction set is taken to get scaled down error measure where higher values are closer together.

$RMSE_{RAU}$  using rounded ratings and predictions can also be calculated to better evaluate the user experience.

### Average User RMSE

$$RMSE_{AU} = \frac{\sum U_{RMSE}}{U_{successful}} \quad (2.10)$$

Where

$\sum U_{RMSE}$  is the sum over all user's individual RMSE, each user's RMSE is calculated as the square root of each user's individual MSE.

$U_{successful}$  is the number of users evaluated (which does not include users for which no predictions could be made).

### Correctness

$$Correctness = \frac{P_{correct}}{P_{successful}} \quad (2.11)$$

Where

$P_{correct}$  is the number of rounded predictions that were correct, e.g. matched the known correct rounded rating.

$P_{successful}$  is the number of successful predictions (as in a prediction was possible to make) evaluated.

### 2.5.5 Ranking evaluation metrics

The ranking evaluation metrics measure the RS ability to assemble Top-N ranking lists with known relevant items.

The evaluation metrics assumes that the test user models in the evaluation data consists of items that the user would like to see returned in a Top-N list. For transaction models it is assumed all items are relevant since they were purchased. For rating models it is assumed that since the user rated an item, the user wouldn't have minded seeing it in a Top-N list. Thus the evaluation metrics measure whether the Top-N list contains e.g. movies the user would've shown interest in, not necessarily whether it contains movies the user will like.

In addition to the basic assumption made for evaluating Top-N ranking lists, it can be said that the presence of one known relevant item *hopefully* gives some credibility to the other items returned in the Top-N lists. Formulated another way, the presence of a relevant item in the Top-N list *hopefully* increases the probability that the other items in the Top-N list, for which the test model contains no data, are also relevant [McLaughlin & Herlocker, 04].

It would also be possible to consider only the items the user has rated highly as relevant and treat those not rated highly as non-rated (they become part of the set of items not rated by the user). This can be accomplished by using a rating threshold and treat items rated equal to or above the rating threshold as relevant. The rating threshold can either be global for all users or possibly be individual for each user (e.g. use the user's mean rating).

Alternatively items below the rating threshold could be considered as known non-relevant instead of being treated as non-rated. By the same assumptions made for the presence of known relevant items, it is possible to then check for the presence of known non-relevant items (this is not the same as unrated items). The presence of known non-relevant items would then indicate a bad Top-N list. Similarly it can also be assumed that the presence of a known non-relevant item in a Top-N list *most likely* increases the likelihood that the other items in the Top-N list, for which the test model contains no data, are also non-relevant.

Evaluations considering only items with ratings above (or as suggested below) a rating threshold, are made by using training/test data sets where the test data only consists of ratings meeting the rating threshold. Note that it makes little sense to include in the test rating data items that are known non-relevant if those items are just ignored during the evaluation. While it would be possible to use a test model consisting of both known relevant and known non-relevant items and at evaluation time check for the presence of both in the Top-N list, the only way to measure the accuracy of such a system using the common evaluation metrics presented in this paper, would still be by measuring independently recall and precision of known relevant and known non-relevant items. That said, it might be interesting to attempt a accuracy measure that simultaneously checks for the presence of known relevant and known non-relevant items. (E.g. what exactly does it mean that recall is 20% for known relevant items and 20

Another alternative to evaluating ranking lists is to within a given Top-N list consider only the rated items and ignore the non rated items during evaluation, this would however lead to underestimates and will not be considered further. (E.g. in a Top-100 list, perhaps only 4 items have been rated by the user, two favourably and two negatively, ignoring non-rated items would give e.g. a precision of  $2/4=50\%$ , while treating non-rated as non-relevant and rated as relevant would give a precision of  $4/100=4\%$ .)

In addition to reporting the metrics described in this section, it is important to also report how many users could be evaluated (successful users) and how many users could not be evaluated (failed user). Additionally number of evaluated users with no relevant items returned and average size of Top-N ranking list can be reported [Table 8].

$U_{successful}$	Number of successful users evaluated (excluding users for which no recommendations could be made, but including users with Top-N lists containing no relevant items).
$U_{failed}$	Number of failed users that could not be evaluated (Top-N list empty, no recommendations at all was possible for the user).
$TN_{successful}$	Number of successful Top-N lists (as in contains at least one relevant item).
$TN_{failed}$	Number of failed Top-N lists (as in contains no relevant items, includes empty Top-N lists).
$TN_a$	Average size of Top-N ranking list. Expected to be N, but may be less if N recommendations is not possible for all users.

Table 2.2: Additional ranking evaluation metrics.

### Coverage

$$Coverage = \frac{TN_{successful}}{TN_{failed}} \quad (2.12)$$

Where

$TN_{successful}$  is the number of successful Top-N lists (as in contains at least one relevant item).

$TN_{failed}$  is the number of failed Top-N lists (as in contains no relevant items, includes empty Top-N lists).

### Recall

$$Recall = \frac{RI_{returned}}{RI} \quad (2.13)$$

Where

$RI_{returned}$  is the total number of relevant items returned over all users.

$RI$  is the total number of known relevant items that could have been returned.

Note that the size of the Top N ranking list might make it impossible to get a perfect recall, e.g. N=10 but user has rated 2000 items making it impossible to recall all relevant items. However recall relates to precision, and precision typically goes down if we increase recall, hence a perfect recall isn't always wanted, but a good balance between recall and precision typically is.

### Average User Recall

$$Recall_{AU} = \frac{\sum U_{recall}}{U_{successful}} \quad (2.14)$$

Where

$\sum U_{recall}$  is the sum over all users of each users individual recall, which is the number of returned relevant items in the user's Top-N list divided by the total number of known relevant items (or n if number of known items exceeds n in order to only evaluate if what could possibly be returned was returned).

$U_{successful}$  is the number of successful users evaluated (excluding users for which no recommendations could be made, but including users with Top-N lists containing no relevant items).

The max number of known relevant items that can be recalled for the user is thus equal to the size of the user's test data set and the size of the Top-N list, do not go by the size of the test data set as it might contain more than n items, in which case a perfect recall will never be achieved and measurement would be skewed towards smaller recall data sets.

## Precision

$$Precision = \frac{RI_{returned}}{I_{returned}} \quad (2.15)$$

Where

$RI_{returned}$  is the total number of relevant returned items over all users.

$I_{returned}$  is the total number of returned items over all users.

Precision tells us how many of the returned items are known to be relevant, however keep in mind that we are not working with complete data, we do not know of all items that are relevant to the user, hence precision is not wanted to be perfect.

## Average User Precision

$$Precision_{AU} = \frac{\sum U_{precision}}{U_{successful}} \quad (2.16)$$

Where

$\sum U_{precision}$  is the sum over all users of each users individual precision, which is the user's recall divided with the number of items returned in the users Top-N list (which is less than or equal to n in order to not skew results towards small test sets that can fit in the Top-N list).

$U_{successful}$  is the number of successful users evaluated (excluding users for which no recommendations could be made, but including users with Top-N lists containing no relevant items).

## (Breeze's / Halflife) Utility

$$Utility = \frac{Utility_{total}}{Utility_{max}} \quad (2.17)$$

Where

$Utility_{total}$  is the sum over all users of each user's individual utility  $R_a$ , where  $R_a$  is defined by Breeze as:

$$R_a = \sum_j \frac{\max(v_{aj} - d, 0)}{2^{\frac{j-1}{a-1}}} \approx \sum_j \frac{1}{2^{\frac{j-1}{a-1}}} \quad (2.18)$$

Where  $j$  is the position (starting at 1) in the Top-N list of a relevant item. The constant  $a$ , called the *halflife* constant, is a guess of which position in the Top-N list there is a 50% chance of user looking at.

The summation is over relevant items in the Top-N list, relevant items in early positions in the Top-N list have a high utility for the user, whereas items further down have a lower utility as the user isn't as likely to view those. Non relevant items serve to push relevant items further down the Top-N list, thus decreasing the utility of the relevant items in the Top-N list. The choice of  $a$  is made heuristically, the idea is that a user isn't likely to look at the 100th item in a top 100 list, however the first 5 are very likely to be viewed, hence 5 might be a suitable halflife constant. Breeze uses 5 [Breese et al., 98].

The term  $\max(v_{aj} - d, 0)$ , with  $v_{aj}$  denoting user  $a$ 's rating (Breeze calls them votes) on item  $j$ , causes items with ratings below what Breeze calls the "default vote"  $d$  to be ignored. Assuming a rating scale of 1-5 the constant  $d$  can be chosen to be the middle of the rating scale, e.g. 3, in which case items a user has rated with 1,2 or 3 would be ignored and not increase the utility if encountered in the Top-N list. However, the idea of a "default vote" is impractical (not all system makes "default votes") as well as not very meaningful. Ignoring the ratings all together makes more sense, since it was initially stated that the evaluation is based on the assumption the user model only contains relevant items. A low rating doesn't indicate a lower relevance by that assumption. Thus it is motivated to replace the term  $\max(v_{aj} - d, 0)$  with 1 to simplify the formula and make it



more understandable and useable for both rating and transaction data. With this new formulation the utility of the item at the "half-life" position is always 0.5. Unless otherwise noted the simplified version of Breeze's utility will be used.

$Utility_{max}$  is the sum over all users of each users individual max utility  $R_m$ , where  $R_m$  is defined as:

$$R_m = \sum_j \frac{1}{2^{\frac{j-1}{a-1}}} \quad (2.19)$$

Where  $j$  goes from 1 to number of known relevant items or the size of the Top-N list. E.g. the maximum utility is achieved when either all known relevant items are at the top of the user's Top-N list, or when the entire Top-N list consists of known relevant items.

#### Average User (Breeze's / Half-life ) Utility

$$Utility_{AU} = \sum U_{utility} \quad (2.20)$$

Where

$\sum U_{utility}$  is the sum over all users of each user's individual utility, which is the user's utility divided with the user's max utility.

$U_{successful}$  is the number of successful users evaluated (excluding users for which no recommendations could be made, but including users with Top-N lists containing no relevant items).

#### F1

$$F1 = \frac{2 \times Recall \times Precision}{Recall + Precision} \quad (2.21)$$

Since recall and precision complement each other, e.g. low recall typically occurs with high precision, and low precision occurs with high recall, both measures tell more about the situation than one alone, the F1 metric thus aims to combine both measures to get a accuracy metric of higher quality. A high value of the F1 metric ensure that both precision and recall was reasonably high.

#### Average User F1

$$F1_{AU} = \frac{\sum U_{utility}}{U_{successful}} \quad (2.22)$$

Where

$\sum U_{F1}$  is the sum over all users of each user's individual F1.

$U_{successful}$  is the number of successful users evaluated (excluding users for which no recommendations could be made, but including users with Top-N lists containing no relevant items).

#### Average First Hit Position

$$AFHP = \frac{\sum FHP_U}{U_{successful}} \quad (2.23)$$

Where

$\sum FHP_U$  is the sum over all users position of their first relevant item (users with no relevant items returned have no first relevant item and are thus ignored).

$U_{successful}$  is the number of successful users evaluated (excluding users for which no recommendations could be made, but including users with Top-N lists containing no relevant items).

## 2.6 Evaluation data

Evaluations will be made based on four datasets. Three rating datasets and one transaction dataset. Each dataset is represent by a set of histograms showing some characteristics of the datasets. Comparison of the histograms reveals common characteristics between the rating datasets, such as a normal distribution of ratings etc.

### 2.6.1 Consistent MovieLens rating dataset (CML)

The 100.000 ratings MovieLens rating dataset was collected by the GroupLens Research Project at the University of Minnesota. The dataset consists of user ratings on movies, ratings are on a 1-5 scale ("half-star" ratings were later introduced into their experimental movie recommender system.)

The original rating dataset contained a few inconsistencies that was removed, in particular 20 pairs of movies with different "id" are really the same movies, meaning users have rated the same movie twice. These inconsistencies were resolved by removing one of the movies and keeping each user's higher rating in the cases a user had rated both movies and differently. Four movies in the set that could not be located in the IMDb were also removed as we needed to be able to retrieve attribute data for all movies. This lead to a reduction of 24 items and 304 ratings, no users needed to be removed. The rating dataset still consists of only users that have rated at least 20 movies (with the exception of one user that now only has 19 ratings). All movies in the dataset we used contain IMDb ids, meaning we could use the publicly available IMDb attribute datasets together with the rating dataset.

Statistics for the rating dataset are shown in [Table 2.3].

Users	943
Items	1658
Ratings	99696
Sparsity (of user-item matrix)	93%
Mean rating (for users)	3.53
Average number of ratings per user	105
Average number of ratings per item	60

Table 2.3: Statistics for the CML rating dataset.

Sparsity is quite high, however this is to be expected for any dataset as users and items increase and is part of the challenge of recommender systems based on rating data. The number of users and items is relatively small, however the dataset is used in many papers and is thus still interesting to include. A large MovieLens rating dataset consisting of 1 million ratings is also offered, however we chose to only work with this smaller dataset. Naturally the hope is that results on this dataset give a good hint of results on larger datasets, whether or not that is the case is hard to know.

If we look at a histogram showing how often the discrete ratings 1-5 are used we see that the spread is roughly normal distributed [Figure 2.3].

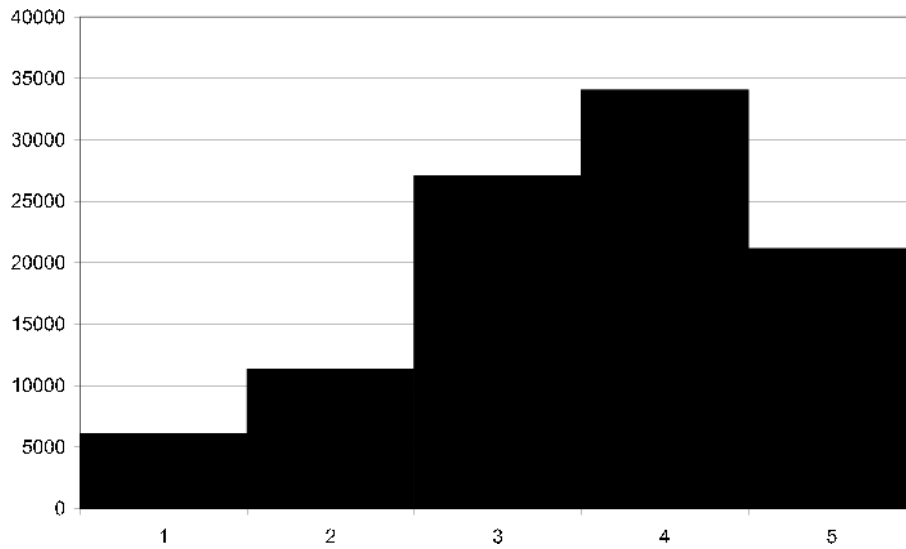


Figure 2.3: Histogram showing the rating frequency in the CML dataset, i.e. how many times each discrete grade 1-5 is assigned an item.

The average number of ratings per user is 105, if we look at a histogram showing how many users have rated varying numbers of items we get a better idea of the rating spread [Figure 2.4].

Similarly we can look closer at the per item rating spread, which is relevant in item based recommendation techniques that will use this rating data [Figure 2.5].

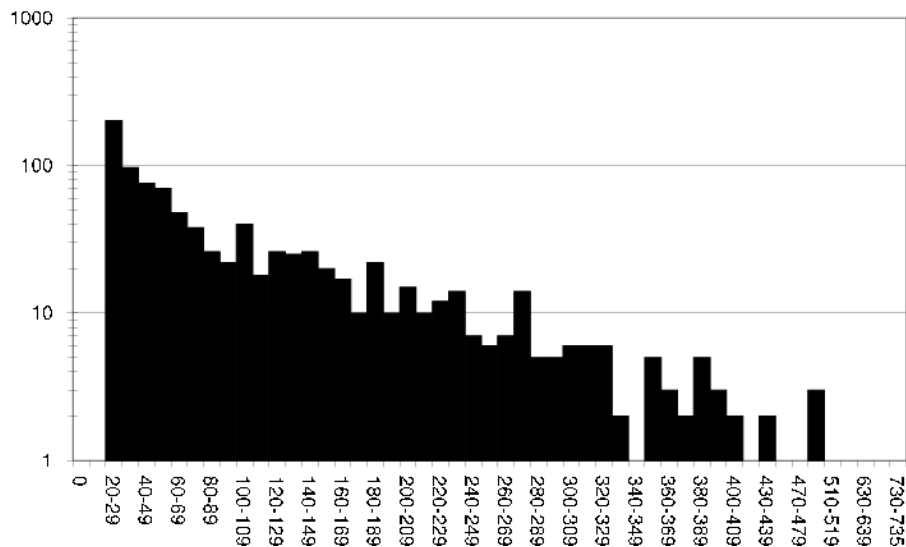


Figure 2.4: Histogram showing how many users have rated varying numbers of items in the CML dataset.

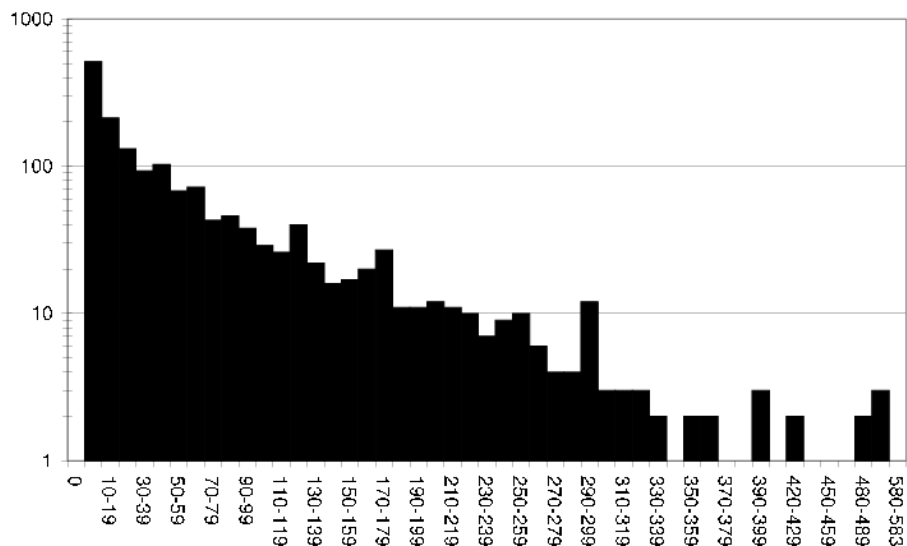


Figure 2.5: Histogram showing how often items have been rated in the CML dataset.

Interesting to study is also how an average user relates to its neighbors. I.e. given any user in the rating dataset, how many neighbors does the user have 100 items in common with [Figure 2.6], and how many neighbors does the user have 100 items in common with that are rated the same [Figure 2.7].

Similarly it is interesting to study how an average item relates to its neighbors. I.e. given any item in the rating dataset, how many neighbors does the item have 100 users in common with [Figure 2.8], and how many neighbors does the item have 100 users in common with that have assigned the item and the neighbor the same rating [Figure 2.9].

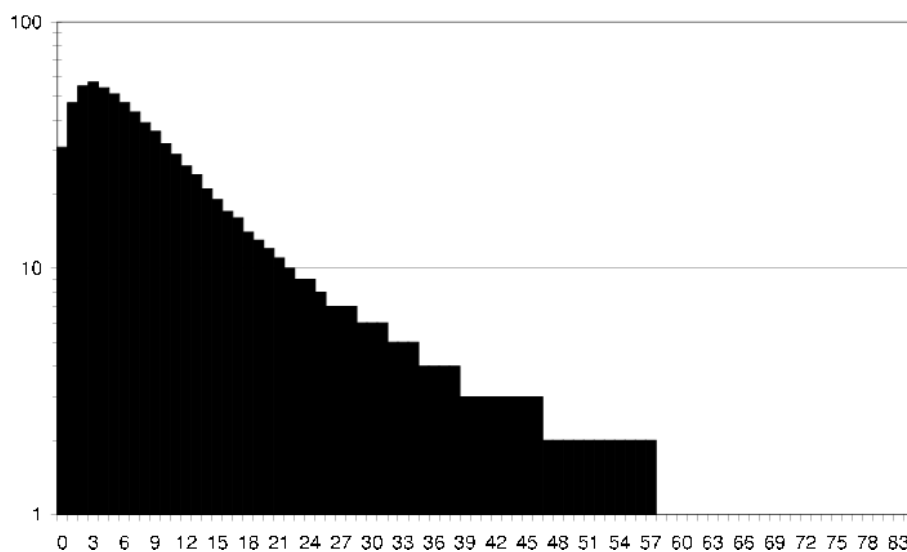


Figure 2.6: How an average user in the CML dataset relates to its neighbors in terms of common items. I.e. how many neighbors does the user have exactly 40 items in common with?

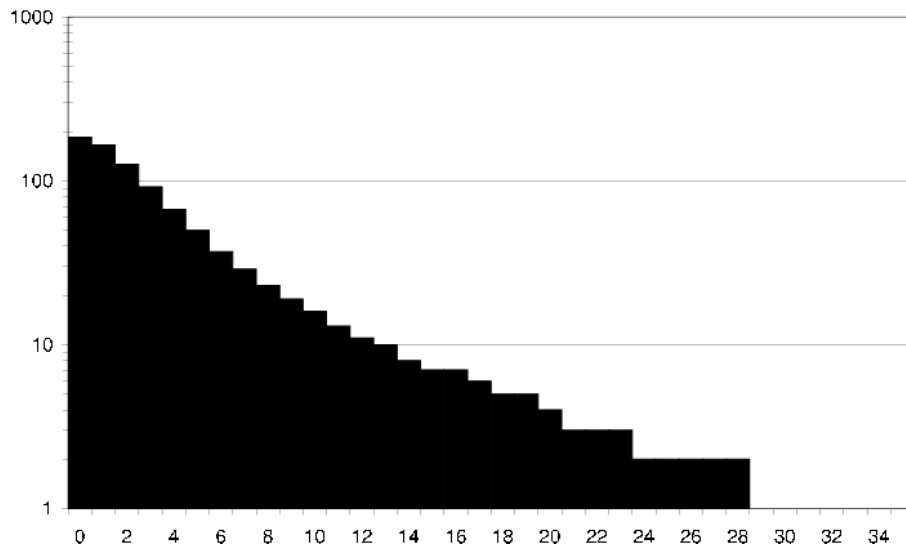


Figure 2.7: How an average user in the CML dataset relates to its neighbors in terms of common ratings. I.e. how many neighbors does the user have exactly 10 items in common with that both the user and the neighbor have rated the same.

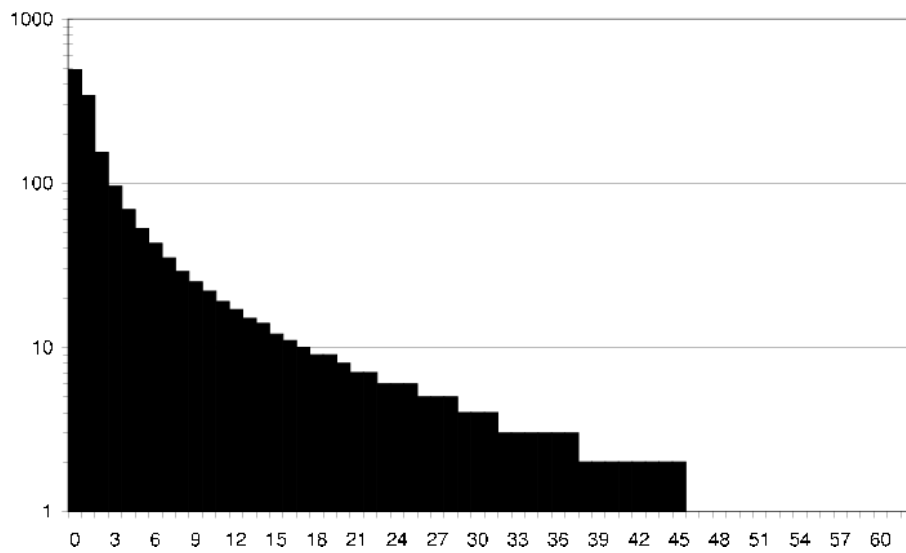


Figure 2.8: How an average item in the CML dataset relates to its neighbors in terms of common users. I.e. how many neighbors does the item have exactly 40 users in common with?

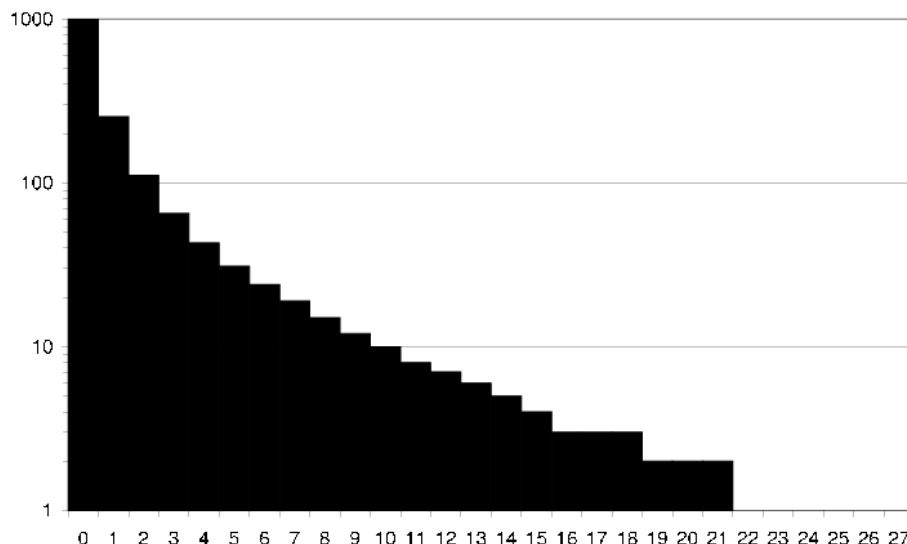


Figure 2.9: How an average item in the CML dataset relates to its neighbors in terms of common ratings. I.e. how many neighbors does the item have exactly 10 users in common with that have given the item and its neighbor the same rating.

## 2.6.2 Discshop rating dataset (DS)

Dataset consists of ratings collected from a Swedish e-commerce site selling DVDs.

Dataset is a preprocessed version of a larger dataset. In particular the original dataset consisted of user ratings on products instead of directly on movies. E.g. ratings were assigned per DVD edition of a movie. When rating a DVD edition the customer was allowed to assign multiple grades, in particular one grade to the DVD edition and one grade to the movie itself. A user may thus rate different DVD editions of the same movie and in the process the user might give the same movie different ratings, this may occur because user changed opinion about the movie or because DVD edition contain significantly different cuts of the movie. The original rating dataset on products does not allow for discerning between different cuts of the same movie, a movie is always the same movie independent of the cut. E.g. Blade Runner and Blade Runner Directors Cut is both identified as the movie Blade Runner, though with two different DVD editions. In order to create this rating dataset of movies, products were mapped to movies by keeping the grade assigned to the movie and ignoring the grade assigned to the DVD edition. In the cases a user had rated multiple DVD editions of the same movie, the highest grade assigned the movie was kept. The reason for not working with the ratings of DVD editions was made in order to get a simpler dataset to work with, a dataset that can easily be used together with attribute datasets for movies.

It was considered practical to ignore users with less than 20 rated movies. Any recommendation technique is considered unlikely to be able to produce reliable recommendations based on too little information about users, in this case less than 20 ratings was considered too little information. By ignoring users with few ratings the size of the rating dataset was also reduced to a more manageable size. Statistics for the rating dataset are shown in [Table 2.4].

Users	2001
Items	6256
Ratings	129829
Sparsity (of user-item matrix)	98%
Mean rating (for users)	3.74
Average number of ratings per user	64
Average number of ratings per item	20

Table 2.4: Statistics for the DS rating dataset.

If we look at a histogram showing the distribution of ratings, we see that it is very similar to the distribution for the CML dataset [Figure 2.10].

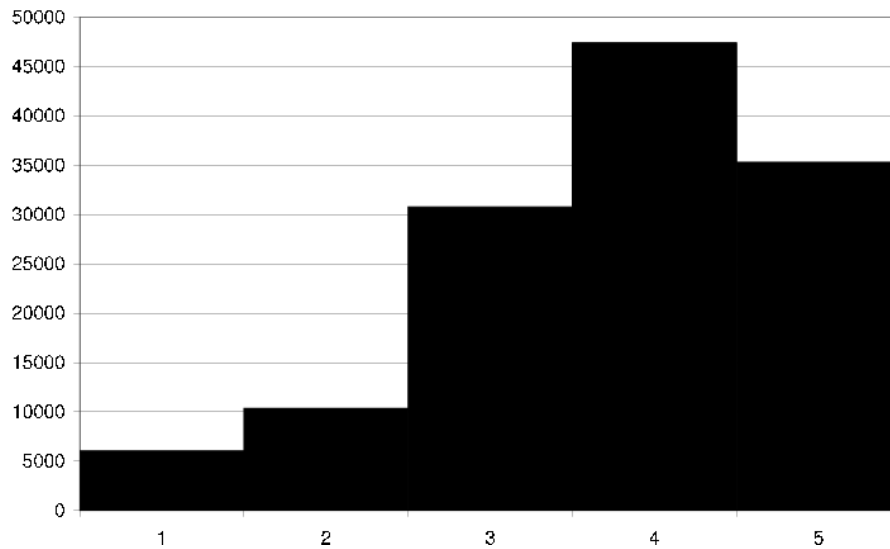
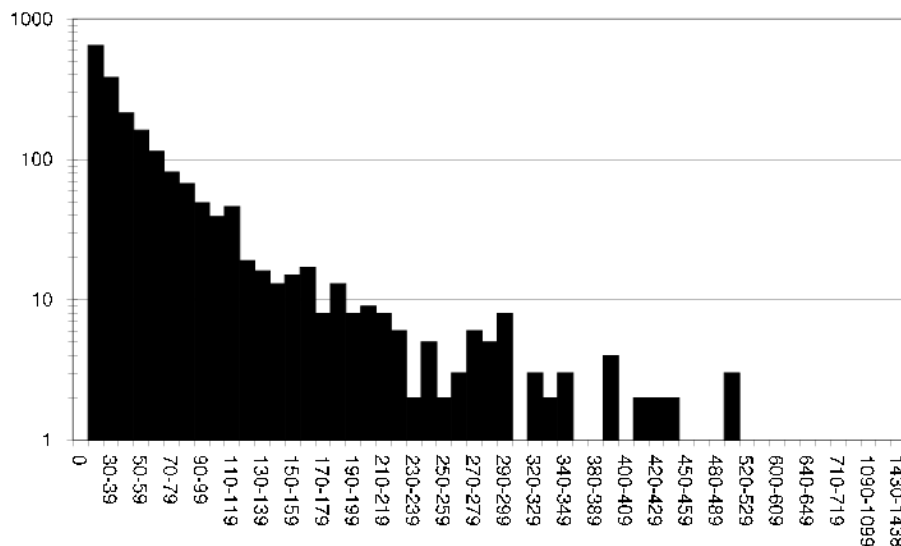


Figure 2.10: Histogram showing the rating frequency in the DS dataset, i.e. how many times each discrete grade 1-5 is assigned to an item.

The average number of ratings per user is 64 in the DS dataset, if we look at a histogram showing how many users have rated varying numbers of items we get a better idea of the rating spread [Figure 2.11].

Similarly we can look closer at the per item rating spread, which is relevant in item based recommendation techniques that will use this rating data [Figure 2.12].



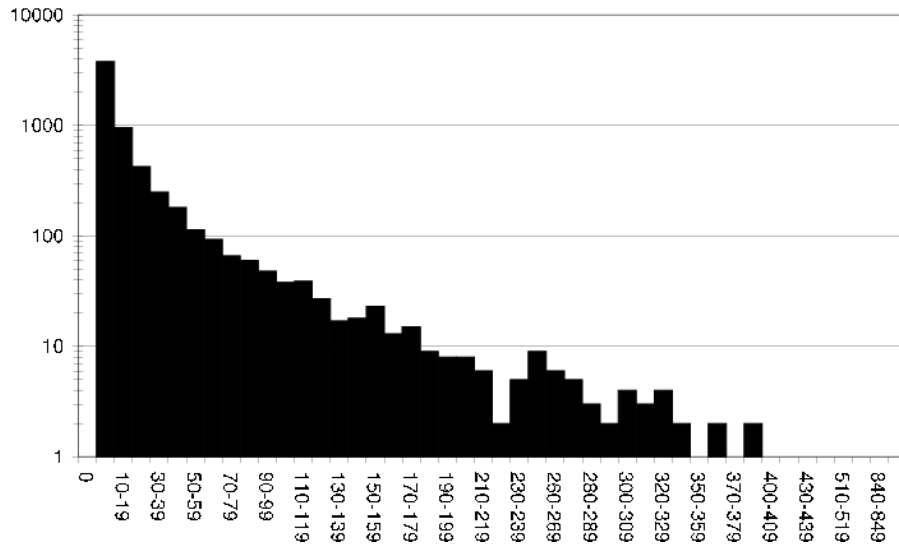


Figure 2.12: Histogram showing how often items have been rated in the DS dataset.

Interesting to study is also how an average user relates to its neighbors. I.e. given any user in the rating dataset, how many neighbors does the user have 100 items in common with [Figure 2.13], and how many neighbors does the user have 100 items in common with that are rated the same [Figure 2.14]. Similarly it is interesting to study how an average item relates to its neighbors. I.e. given any item in the rating dataset, how many neighbors does the item have 100 users in common with [Figure 2.15], and how many neighbors does the item have 100 users in common with that have assigned the item and the neighbor the same rating [Figure 2.16].

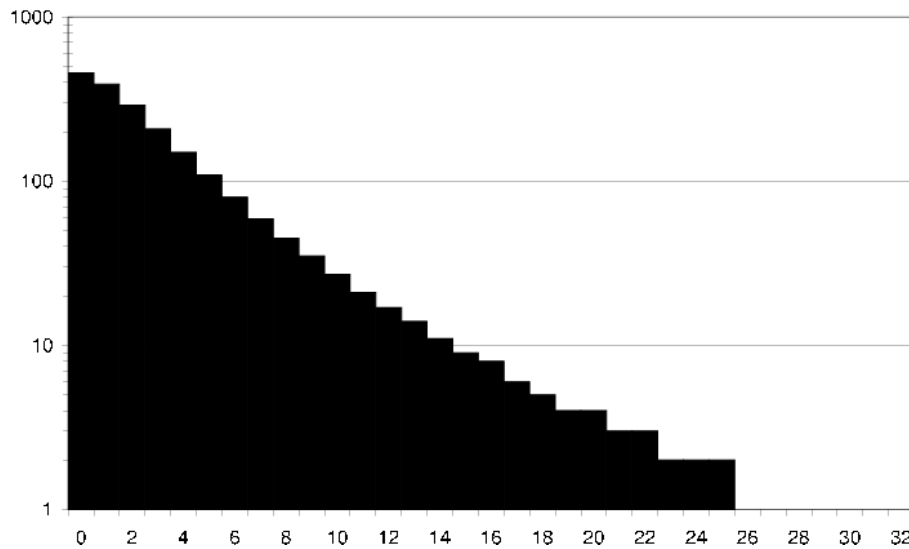


Figure 2.13: How an average user in the DS dataset relates to its neighbors in terms of common items. I.e. how many neighbors does the user have exactly 40 items in common with?



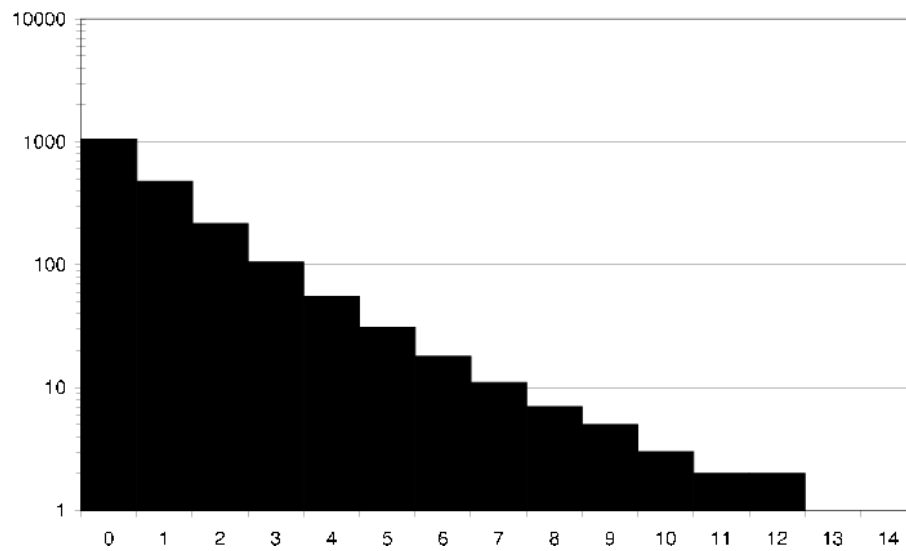


Figure 2.14: How an average user in the DS dataset relates to its neighbors in terms of common ratings. I.e. how many neighbors does the user have exactly 10 items in common with that both the user and the neighbor have rated the same.

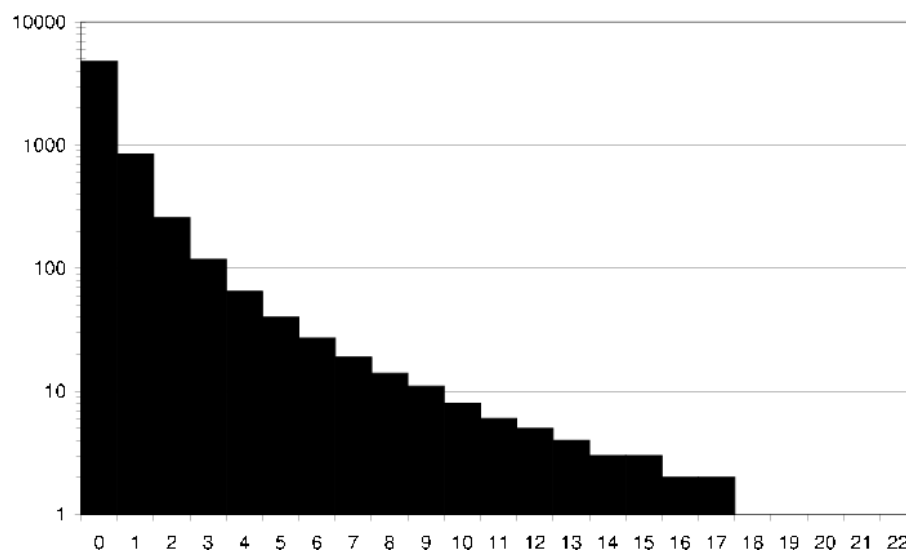


Figure 2.15: How an average item in the DS dataset relates to its neighbors in terms of common users. I.e. how many neighbors does the item have exactly 40 users in common with?

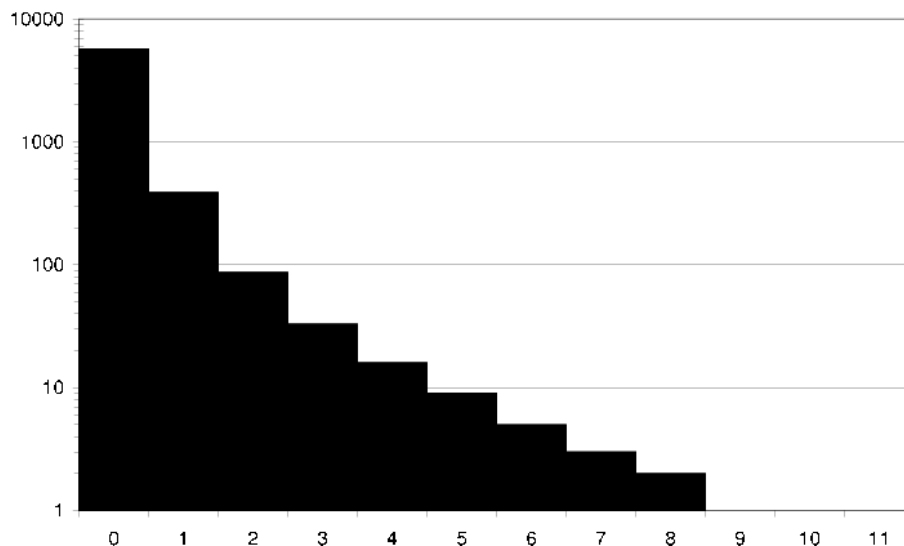


Figure 2.16: How an average item in the DS dataset relates to its neighbors in terms of common ratings. I.e. how many neighbors does the item have exactly 10 users in common with that have given the item and its neighbor the same rating.

### 2.6.3 Discshop rating dataset 2 (DS2)

Dataset is a subset of the DS dataset in which it has been required that in addition to each user having to have rated at least 20 items, each item must also have been rated by 20 users. This avoids situations in which a user has indeed rated 20 items, but that user is the only user that has rated those items. The enforcement is for simplicity not strict, some items will be rated by less than 20 users, however most of the items will have been rated by more than 20 users.

Statistics for the rating dataset are shown in [Table 2.5].

Users	2001
Items	1962
Ratings	110418
Sparsity (of user-item matrix)	97%
Mean rating (for users)	3.79
Average number of ratings per user	55
Average number of ratings per item	56

Table 2.5: Statistics for the DS2 rating dataset.

If we look at a histogram showing how often the discrete ratings 1-5 are used we see that the spread is roughly normal distributed [Figure 2.17] and similar to the DS data's rating spread.

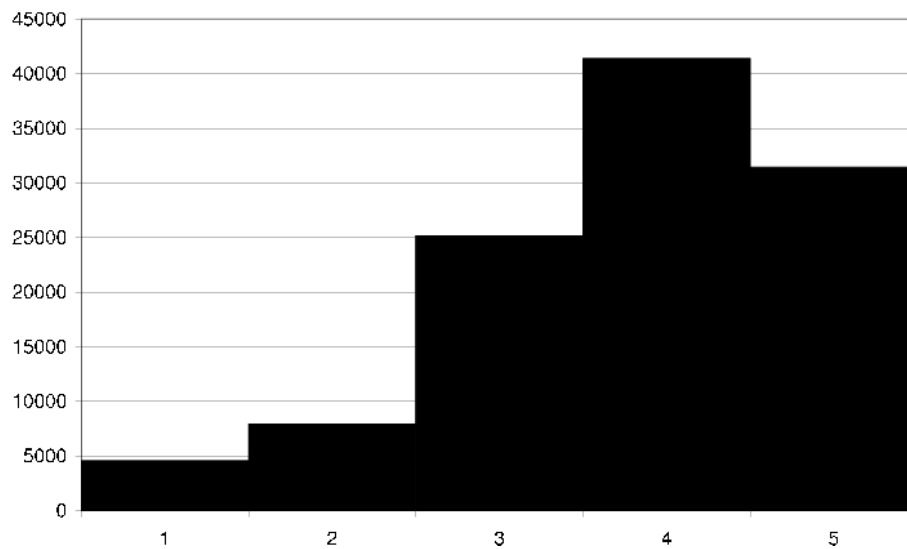


Figure 2.17: Histogram showing the rating frequency in the DS2 dataset, i.e. how many times each discrete grade 1-5 is assigned an item.

The average number of ratings per user is 55, if we look at a histogram showing how many users have rated varying numbers of items we get a better idea of the rating spread [Figure 2.19].

Similarly we can look closer at the per item rating spread, which is relevant in item based recommendation techniques that will use this rating data [Figure 2.18].

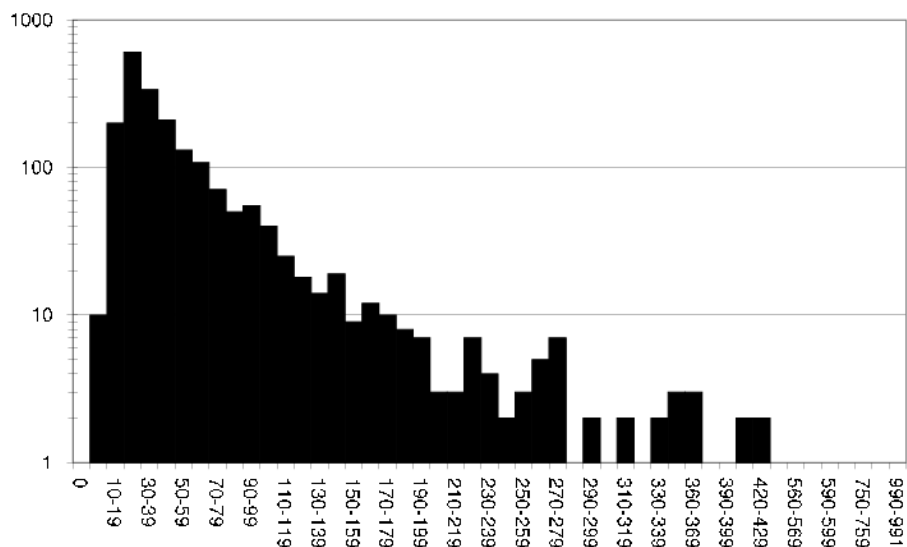


Figure 2.18: Histogram showing how many users have rated varying numbers of items in the DS2 dataset.

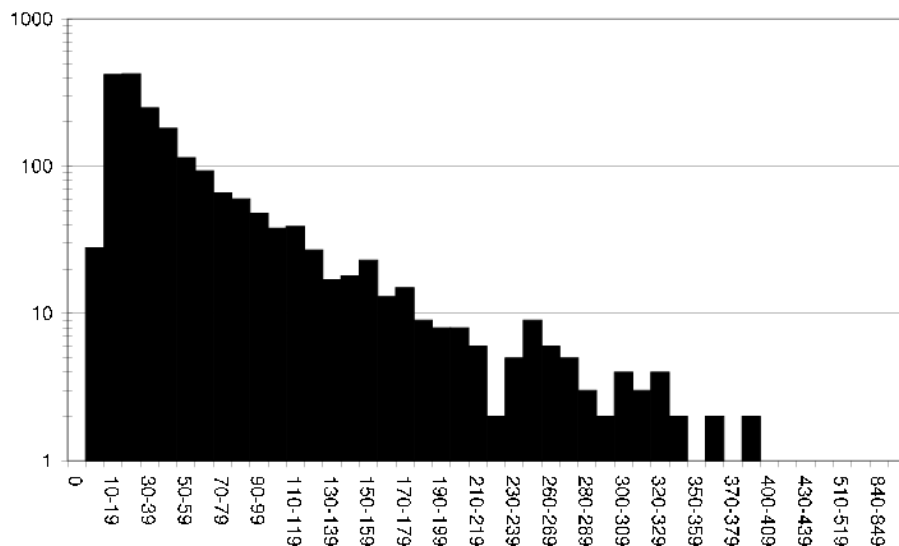


Figure 2.19: Histogram showing how often items have been rated in the DS2 dataset.

Interesting to study is also how an average user relates to its neighbors. I.e. given any user in the rating dataset, how many neighbors does the user have 100 items in common with [Figure 2.20], and how many neighbors does the user have 100 items in common with that are rated the same [Figure 2.21]. Similarly it is interesting to study how an average item relates to its neighbors. I.e. given any item in the rating dataset, how many neighbors does the item have 100 users in common with [Figure 2.22], and how many neighbors does the item have 100 users in common with that have assigned the item and the neighbor the same rating [Figure 2.23].

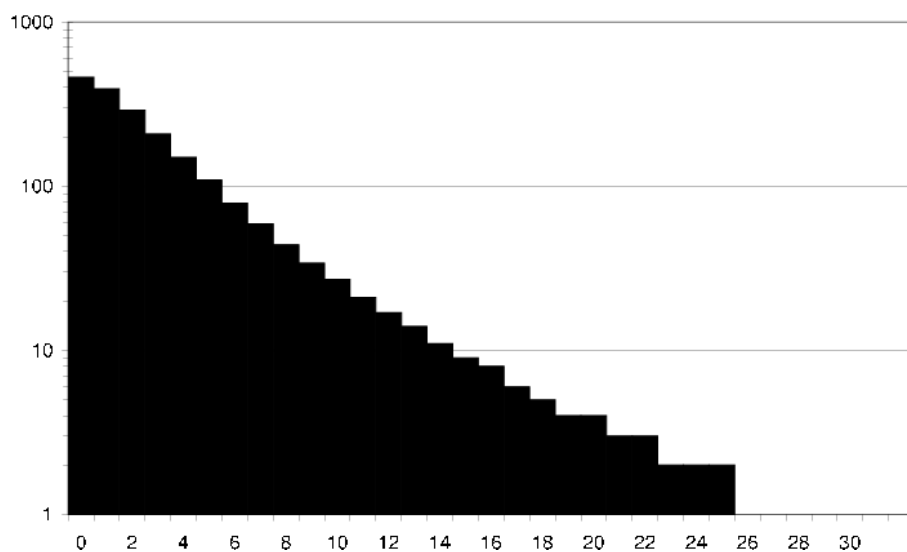


Figure 2.20: How an average user in the DS2 dataset relates to its neighbors in terms of common items. I.e. how many neighbors does the user have exactly 40 items in common with?

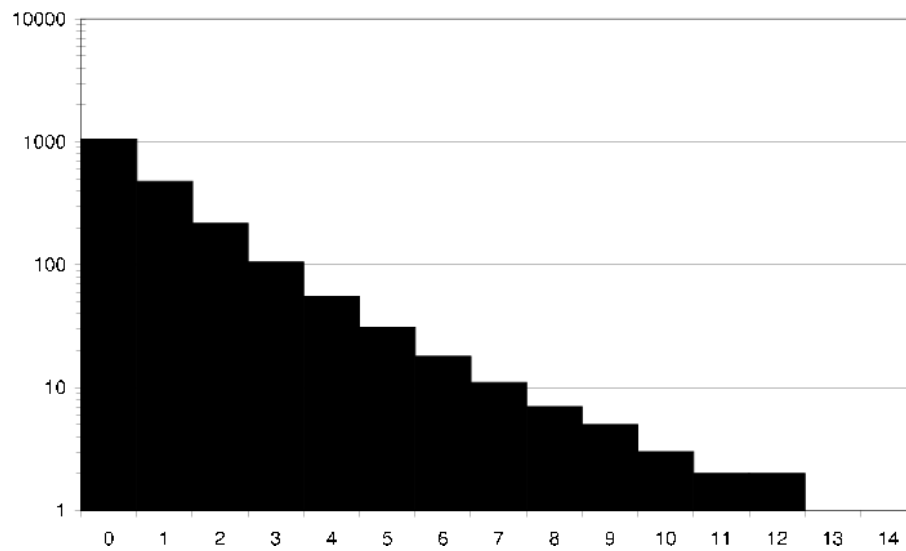


Figure 2.21: How an average user in the DS2 dataset relates to its neighbors in terms of common ratings. I.e. how many neighbors does the user have exactly 10 items in common with that both the user and the neighbor have rated the same.

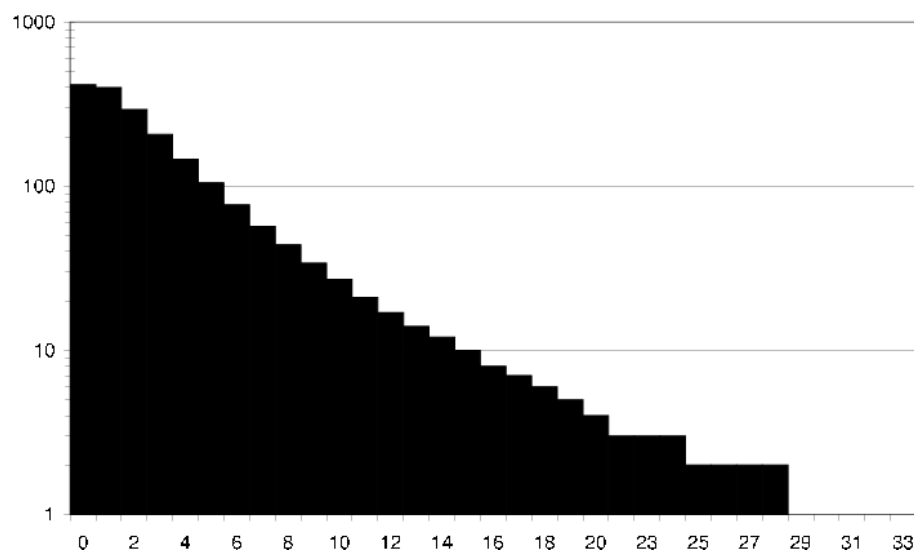


Figure 2.22: How an average item in the DS2 dataset relates to its neighbors in terms of common users. I.e. how many neighbors does the item have exactly 40 users in common with?

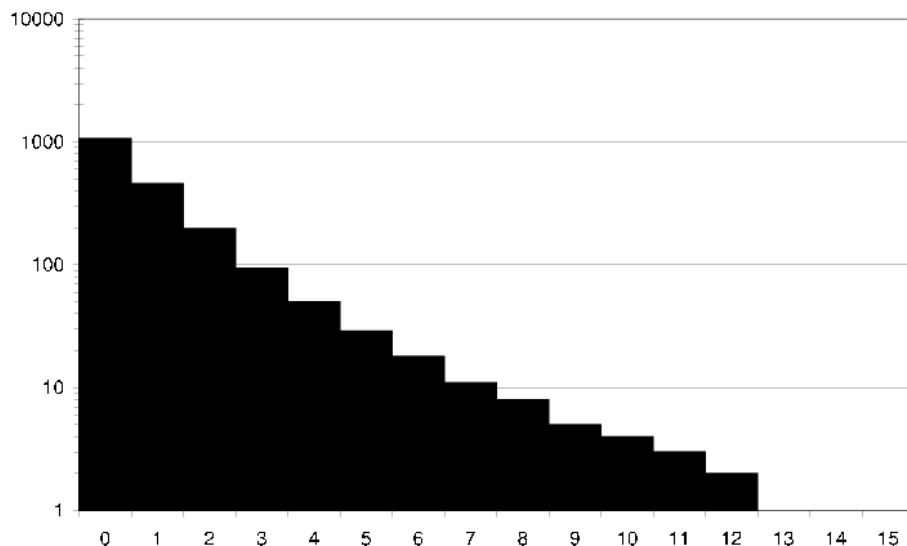


Figure 2.23: How an average item in the DS2 dataset relates to its neighbors in terms of common ratings. I.e. how many neighbors does the item have exactly 10 users in common with that have given the item and its neighbor the same rating.

#### 2.6.4 Discshop transaction dataset (DS-T)

In addition to the rating dataset we also obtain a transaction dataset. Dataset consists of user (customer) transactions. A transaction in this case is defined as a purchase by a user of a single item (movie). Dataset does not aid in market basket analysis as a users complete purchase history has been aggregated. All users in the dataset have purchased at least 20 movies.

Statistics for the transaction dataset are shown in [Table 2.6].

Users	3158
Items	6380
Transactions	145637
Sparsity (of user-item matrix)	99%
Average number of transactions per user	46
Average number of transactions per item	22

Table 2.6: Statistics for the DS-T transaction dataset.

Histograms showing the per user and item purchase distribution are shown in [Figure 37,Figure 38], as well as an average user and average items relation to its neighbors in [Figure 39,Figure 40].

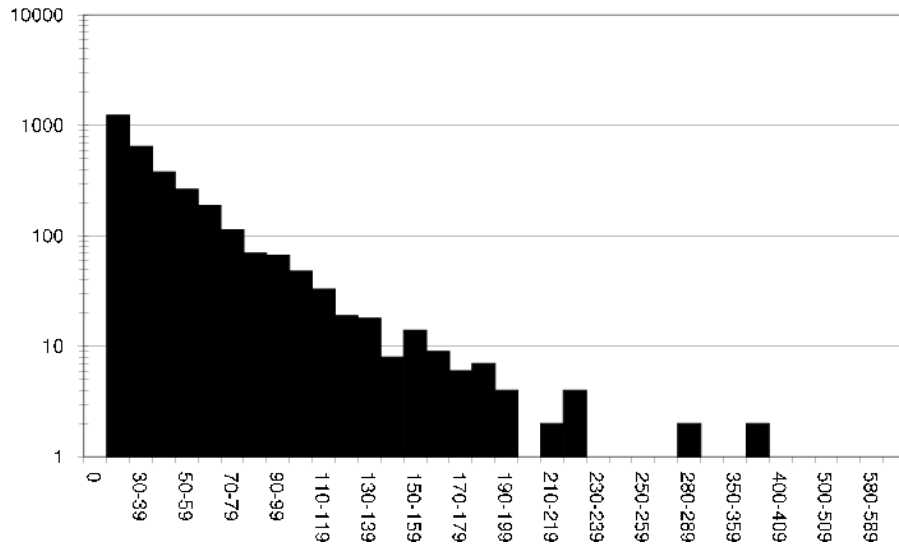


Figure 2.24: Histogram showing the distribution of the size of user transactions in the DS-T dataset.

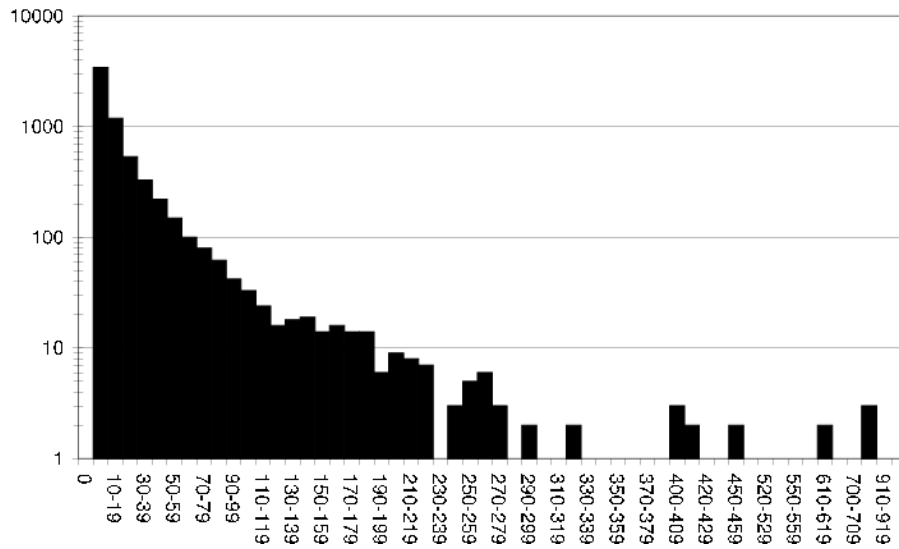


Figure 2.25: Histogram showing the distribution of the size of item transactions in the DS-T dataset.

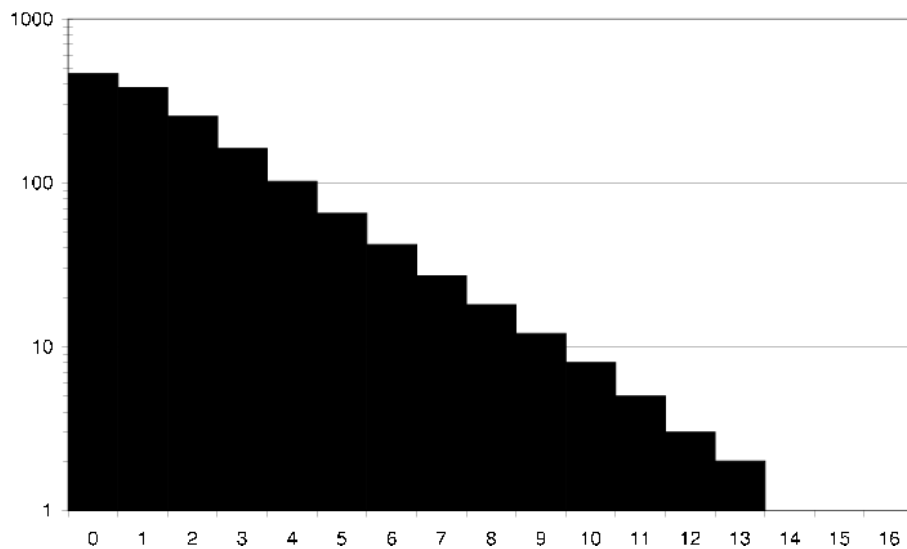


Figure 2.26: How an average user in the DS-T dataset relates to its neighbors in terms of common transactions. I.e. how many neighbors does the user have exactly 40 purchased items in common with?

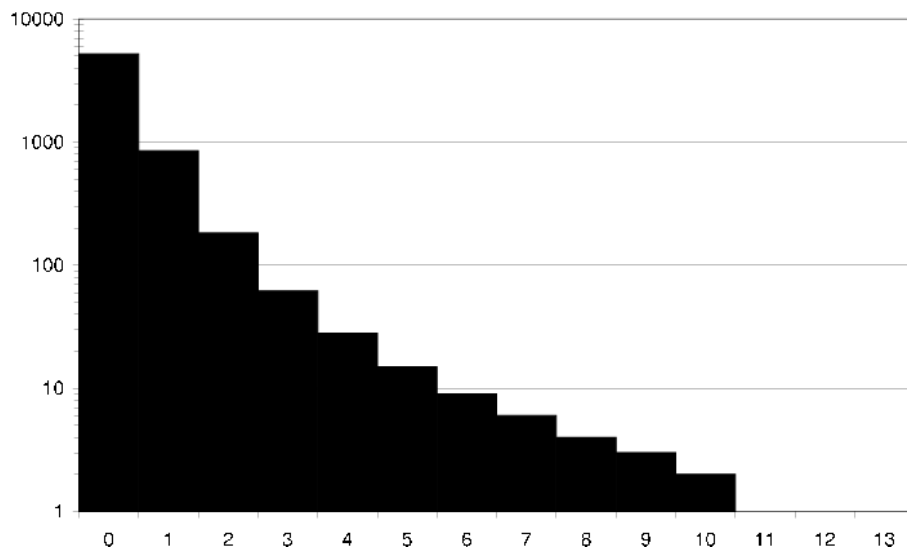


Figure 2.27: How an average item in the DS-T dataset relates to its neighbors in terms of common transactions. I.e. how many neighbors does the item have exactly 40 users (i.e. users that have purchased both the item and the neighbor) in common with?

### 2.6.5 Discshop attribute dataset (DS-A)

In addition to the movie rating- and movie transaction- data we also obtain the information displayed for the movies the customers could purchase, we prepare this per movie attribute information into separate datasets consisting of one type of attribute each. The attribute types we will work with are:

1. Actor
2. Age limit
3. Country
4. Director



5. Genre
6. Title
7. Alternative title
8. Original title
9. Writer
10. Year

### **2.6.6 IMDb attribute dataset (IMDB)**

The movie attribute dataset are obtain from The Internet Movie Database (IMDb). The original attribute datasets use the movie's original title as key, movies with the same title have a identifier specifying the duplication number appended. However, for our purposes it turns out to be more practical to work with IMDb Constant Title Codes ("IMDb-ids"), especially as the CML dataset has been converted to use IMDb-ids. Thus we prepared IMDb attribute datasets that use IMDb-ids instead of the unique IMDb movie titles.

We prepared datasets consisting of one attribute type each, the attribute types we will work with are:

1. Actors
2. Actresses
3. Cinematographers
4. Color info
5. Composers
6. Costume designers
7. Countries
8. Directors
9. Distributors
10. Editors
11. Genres
12. Keywords
13. Language
14. Locations
15. Miscellaneous companies
16. Miscellaneous
17. Producers
18. Production companies
19. Production designers
20. Sound-mix
21. Special effects companies
22. Technical
23. Title
24. Writers

## 2.7 Evaluation results

The recommendation techniques were evaluated for the evaluation datasets using various combinations of evaluation protocol, training/test data formation and evaluation metrics.

For the recommendation techniques we select where necessary an appropriate combination of similarity and prediction algorithm, we use a decent set of settings giving good results, not necessarily the absolute best results, but near enough to be interesting to list.

We will describe briefly for each recommendation technique the setup of the evaluation, such that the evaluation can easily be understood and reproduced.

### 2.7.1 Accuracy of predictions

The result tables use the following abbreviations for the column headings:  $ET$  = Total evaluation time,  $U_s = U_{successful}$ ,  $U_f = U_{failed}$ ,  $P_s = P_{successful}$ ,  $P_f = P_{failed}$ ,  $Cov$  = Coverage,  $Corr$  = Correctness,  $P$  = Precision.

The settings column names the recommendation algorithm instance used of the recommendation technique in question. Typically only the name of the dataset used for evaluation is given (i.e. DS, DS2 or CML). Each table of evaluation results is followed by a description of algorithms and settings used for each recommendation technique.

In all cases a 10-fold evaluation as previously described was performed. Note that the same 10-folded evaluation data of each dataset were used for each evaluation.

#### BASELINESF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS-UM	0	2001	0	12982	0	100.0%	39.7%	0.780	0.745	0.804	0.771	0.195	1.001	1.001
DS-IM	0	2000	0	12861	121	99.1%	41.8%	0.760	0.720	0.778	0.739	0.190	0.965	0.983
DS-PDFM	0	2000	0	12861	121	99.1%	46.0%	0.687	0.649	0.700	0.665	0.172	0.820	0.906
DS-R	0	2001	0	12982	0	100.0%	21.1%	1.428	1.429	1.482	1.483	0.357	3.047	1.746
DS2-UM	0	1991	0	11035	0	100.0%	40.3%	0.766	0.730	0.792	0.760	0.192	0.968	0.984
DS2-IM	0	1991	0	11035	0	100.0%	42.9%	0.736	0.694	0.764	0.723	0.184	0.898	0.948
DS2-PDFM	0	1991	0	11035	0	100.0%	47.2%	0.667	0.627	0.688	0.651	0.167	0.772	0.879
DS2-R	0	1991	0	11035	0	100.0%	21.1%	1.435	1.435	1.488	1.487	0.359	3.078	1.754
CML-UM	0	943	0	9967	0	100.0%	36.2%	0.834	0.802	0.842	0.809	0.209	1.084	1.041
CML-IM	0	943	0	9952	15	99.8%	36.9%	0.815	0.783	0.833	0.801	0.204	1.043	1.021
CML-PDFM	0	943	0	9952	15	99.8%	40.8%	0.754	0.718	0.776	0.742	0.189	0.916	0.957
CML-R	0	943	0	9967	0	100.0%	21.7%	1.386	1.385	1.400	1.399	0.346	2.875	1.695

Table 2.7: Evaluation results for the BASELINESF technique. The abbreviations UM, IM, PDFM and R refer to using respectively the *User Mean*, *Item Mean*, *Population Deviation From Mean* and *Random* prediction algorithms. Thus DS-UM means the Discshop dataset was used and the User Mean prediction algorithm.

As a statistical prediction algorithm the PDFM algorithm seems to be the one that gives the lowest MAE for both datasets. Note that neither of the algorithms are truly CF algorithms, though they rely on the same rating datasets as the CF algorithms we will later try and thus serves as a decent statistical baseline to compare CF algorithms performance against. The results of the random prediction algorithm can be seen to serve as a comparison against any prediction algorithm, as the aim is at least to be better than a random prediction algorithm in any circumstance.

## UCF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS	3	1990	10	12729	253	98.0%	47.2%	0.679	0.640	0.701	0.664	0.170	0.820	0.906
DS2	2	1978	12	11002	33	99.7%	48.2%	0.657	0.618	0.687	0.652	0.164	0.768	0.876
CML	1	943	0	9952	15	99.8%	42.7%	0.729	0.692	0.753	0.718	0.182	0.874	0.935

Table 2.8: Evaluation results for the UCF technique. In all instances the Pearson similarity algorithm (with OT=2, ST=30) and the Weighted Deviation From Mean prediction algorithm (with  $NS_{min}=1$ ,  $NS_{max}=30$ ) was used. No minimum or maximum similarity restrictions were imposed.

Since the UCF technique sorts neighbors in descending order of similarity (highest first) it means that the 30 most similar neighbors to the active user were used as recommenders. We can note that the technique in this case performs better for both datasets than the best performing instance of the baseline technique in terms of MAE.

## RNUCF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS	4	2000	0	12861	121	99.1%	45.8%	0.692	0.653	0.704	0.668	0.173	0.829	0.911
DS2	1	1991	0	11035	0	100.0%	46.7%	0.672	0.633	0.692	0.656	0.168	0.783	0.885
CML	1	943	0	9952	15	99.8%	40.5%	0.762	0.726	0.784	0.749	0.190	0.934	0.967

Table 2.9: Evaluation results for the RNUCF technique. Same settings as for the UCF technique were used.

Exactly the same settings and algorithms as in the UCF Technique evaluation were used. However since the RNUCF technique unlike the UCF technique (to which it otherwise is identical) sorts neighbors randomly it means that between 1 and 30 randomly chosen neighbors will be used as recommenders. Note that only neighbors with which a similarity could be calculated will be considered. So this truly is a random neighbors version of the UCF technique. We can note that the technique in this case only performs marginally worse than the UCF technique for both datasets in terms of MAE. Can this be a consequence of most similarities being observed to be very low, and as such any similarity will do as good as any other?

## TRUSTUCF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS	16	1969	31	10888	2094	83.9%	47.4%	0.676	0.639	0.700	0.666	0.169	0.820	0.906
DS2	11	1963	27	10288	747	93.2%	47.9%	0.664	0.627	0.693	0.659	0.166	0.793	0.890
CML	6	942	0	9583	384	96.1%	42.9%	0.733	0.695	0.759	0.723	0.183	0.892	0.944

Table 2.10: Evaluation results for the TRUSTUCF technique. Same settings as for the UCF technique were used.

Again the same settings and algorithms as in the UCF Technique evaluation were used. However the similarity between the active user and the active user's neighbors is weighted depending on how much the neighbors are trusted to be able to predict for the active item. We can note that the technique in this case performs basically equally well as the UCF technique for both datasets in terms of MAE. Coverage is however lower than for the UCF technique.

### CIPUCF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS	5	1990	10	12729	253	98.0%	46.7%	0.684	0.645	0.711	0.674	0.171	0.826	0.909
DS2	3	1978	12	11002	33	99.7%	47.7%	0.663	0.624	0.697	0.662	0.166	0.774	0.880
CML	1	943	0	9952	15	99.8%	42.7%	0.733	0.696	0.758	0.724	0.183	0.887	0.942

Table 2.11: Evaluation results for the CIPUCF technique. Same settings as for the UCF technique were used.

Again the same settings and algorithms as in the UCF Technique evaluation were used. However neighbors with most items common with the active user were prioritized in hopes of those similarities being most reliable. We can note that the technique in this case only performs marginally worse than the UCF technique for both datasets in terms of MAE.

### ICF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS	10	1998	2	12591	391	97.0%	43.9%	0.727	0.690	0.776	0.745	0.182	0.917	0.958
DS2	2	1990	0	11033	2	100.0%	44.6%	0.707	0.669	0.764	0.731	0.177	0.861	0.92
CML	1	943	0	9936	30	99.7%	41.7%	0.744	0.708	0.786	0.751	0.186	0.902	0.950

Table 2.12: Evaluation results for the ICF technique. In all instances the Adjusted Cosine similarity algorithm (with  $OT=2$ ,  $ST=30$ ) and the Average Rating prediction algorithm (with  $NS_{min}=1$ ,  $NS_{max}=30$ ) was used. No minimum or maximum similarity restrictions were imposed. In all cases a complete item model was created ( $MS=0$ ), only the neighborhood size restricted the number of items involved in predictions (a variation, perhaps, of the more common approach).

Since the ICF technique sorts item neighbors (items similar to the active item) in descending order of similarity (highest first) it means that the 30 most similar items that the active user had rated were used as a basis for the predictions. We can note that the technique in this case for all datasets performs worse than or equal to the best performing instance of the baseline technique in terms of MAE.

### RNICF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS	11	1999	1	12858	124	99.0%	40.0%	0.786	0.750	0.807	0.773	0.196	1.031	1.015
DS2	1	1991	0	11035	0	100.0%	40.2%	0.772	0.735	0.795	0.763	0.193	0.984	0.992
CML	0	943	0	9952	15	99.8%	36.0%	0.843	0.811	0.847	0.815	0.211	1.109	1.053

Table 2.13: Evaluation results for the RNICF technique. Same settings as for the ICF technique were used.

Exactly the same settings and algorithms as in the ICF Technique evaluation were used. However since the RNICF technique unlike the ICF technique (to which it otherwise is identical) sorts item neighbors randomly it means that between 1 and 30 randomly chosen item will be used as neighbors for the active item and make the basis for the predictions. Note that only neighbors with which a similarity could be calculated will be considered. So this truly is a random neighbors version of the ICF technique. We can note that the technique in this case only performs somewhat worse than the ICF technique for both datasets in terms of MAE. The evaluation based on the CML dataset seems to do a bit more than worse though compared to the ICF technique.

## PCBF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS	42	2001	0	12975	7	99.9%	41.3%	0.758	0.722	0.788	0.755	0.190	0.958	0.979
DS2	22	1990	0	11029	6	99.9%	41.9%	0.746	0.709	0.777	0.741	0.186	0.930	0.965
CML	7	943	0	9935	32	99.7%	38.3%	0.790	0.756	0.809	0.777	0.197	0.982	0.99

Table 2.14: Evaluation results for the PCBF technique. For the DS dataset six different attribute types were involved in calculating item similarities. For the CML dataset 21 different attribute types were involved in calculating the item similarities. To calculate per attribute similarities the Attribute Cosine similarity algorithm (with  $OT=1$ ,  $ST=30$ ) and the Common Attributes similarity algorithm (with  $OT=1$ ,  $ST=30$ ). No minimum similarity restrictions were imposed. Predictions were made in all instances using the Weight Sum prediction algorithm (with  $NS_{min}=1$ ,  $NS_{max}=30$ ). In all cases a complete item model was created ( $MS=0$ ), only the neighborhood size restricted the number of items involved in predictions.

For the DS dataset six different attribute types from the DS-A dataset were used: Actor, Age limit, Country, Director, Writer and Genre. The Common Attributes similarity algorithm were used for the Actor, Age limit, Country and Writer attribute types. The Attribute Cosine similarity algorithm were used for the Director and Genre attribute types. For both similarity algorithms (in all cases) at least one common attribute was required and a significance threshold of 30 was used (which might not be a good choice seeing how movies only for the actor attribute type is likely to have more than 30 attributes). Similarities were weighted based on which attribute they were based on as follows; Actor 10, Age limit 1, Country 3, Director 9, Genre 8 and Writer 9 (e.g. the Actor similarity makes up  $10/(1 + 3 + 9 + 8 + 9)$  of the total similarity). That is, actors, director, genre and write were most influential while age limit and country influence the final similarity very little. For the CML dataset all available attribute types except title in the IMDB dataset were used, again the Common Attributes and Attribute Cosine similarity algorithms were used with the same settings as for the DS dataset. Varying weights were used for the different attributes, highest influence was placed on similarities based on the Actor, Actress, Director, Writer and Genre attribute types. The Weighted Sum prediction algorithm was used to generate predictions, basing its predictions on a neighborhood consisting of at least 1 neighbor and at most 30 neighbors. Complete item similarity models were generated, i.e. for each item its similarities with all other items were precalculated and available for consideration by the prediction algorithm.

We can note that the technique in this case only performs somewhat worse than the ICF technique for both datasets in terms of MAE, it performs marginally better than the RNICF technique however.

### 2.7.2 Relevance of Top-N ranking lists

The result tables use the following abbreviations for the column headings: ET = Total evaluation time,  $U_s = U_{successful}$ ,  $U_f = U_{failed}$ ,  $TN_s = TN_{successful}$ ,  $TN_f = TN_{failed}$ , Cov=Coverage, R=Recall, P=Precision, U=Utility,  $R_{AU} = Recall_{AU}$ ,  $P_{AU} = Precision_{AU}$ ,  $U_{AU} = Utility_{AU}$ .

In all cases, unless otherwise noted, a 10-fold evaluation was performed. Note that the same 10-folded datasets were used for each evaluation.

In all cases a top N=10 list was generated for each user. When measuring the utility the half life constant  $a$  was set to 5 in all cases.

### TOPNUCF Technique

Settings	ET	$U_s$	$U_f$	$TN_s$	$TN_f$	$TN_a$	Cov	R	P	F1	U	AFHP	$R_{AU}$	$P_{AU}$	$F1_{AU}$	$U_{AU}$
CML	47	943	0	310	632	10	32.9%	4.6%	4.8%	0.047	0.042	4	4.5%	4.8%	0.039	0.038
DS (4 Fold w/ 75% Test)	8	495	1505	421	1579	6	21.1%	2.6%	40.8%	0.049	0.078	1	3.1%	41.5%	0.057	0.078
DS2 (4 Fold w/ 75% Test)	33	1968	29	572	1426	9	28.6%	2.7%	3.7%	0.031	0.031	4	3.2%	3.8%	0.031	0.030
CML (4 Fold w/ 75% Test)	3	761	181	688	254	9	73.0%	3.6%	36.3%	0.065	0.063	2	4.5%	36.6%	0.075	0.063

Table 2.15: Evaluation results for the TOPNUCF technique. In all instances the Pearson similarity algorithm (with  $OT = 2$ ,  $ST = 30$ ) and the Weighted Deviation From Mean prediction algorithm (with  $NS_{min} = 10$ ,  $NS_{max} = 20$ ) was used. Only the CML dataset was evaluated using a 10-folded evaluation dataset, instead a 4-fold evaluation dataset with 75% test data (note, this is as opposed to a 25% test data) was used to evaluate using many known relevant items.

### TOPNICF Technique

Settings	ET	$U_s$	$U_f$	$TN_s$	$TN_f$	$TN_a$	Cov	R	P	F1	U	AFHP	$R_{AU}$	$P_{AU}$	$F1_{AU}$	$U_{AU}$
DS	15	1272	728	344	1656	8	17.2%	4.2%	4.1%	0.041	0.036	4	4.9%	4.2%	0.039	0.036
DS2	5	1271	719	356	1635	8	17.9%	5.0%	4.2%	0.046	0.039	4	5.5%	4.4%	0.043	0.039
CML	6	856	86	392	550	9	41.6%	6.1%	7.1%	0.066	0.053	4	8.0%	7.2%	0.064	0.055
DS (All-But1)	16	1432	568	60	1940	8	3.0%	4.2%	0.5%	0.008	0.024	4	4.2%	0.6%	0.010	0.024
DS2 (All-But1)	6	1427	563	68	1922	8	3.4%	4.8%	0.5%	0.010	0.027	4	4.8%	0.7%	0.011	0.027
CML (AllBut1)	7	888	54	72	870	9	7.7%	8.2%	0.8%	0.015	0.044	4	8.2%	0.9%	0.015	0.044

Table 2.16: Evaluation results for the TOPNICF technique. In all instances the Adjusted Cosine similarity algorithm (with  $OT = 20$ ,  $ST = 0$ ) and the Weighted Sum prediction algorithm (with  $NS_{min} = 20$ ,  $NS_{max} = 50$ ) was used. In all cases a complete item model was created ( $MS = 0$ ), only the neighborhood size restricted the number of items involved in predictions.

### TOPNPCBF Technique

Settings	ET	$U_s$	$U_f$	$TN_s$	$TN_f$	$TN_a$	Cov	R	P	F1	U	AFHP	$R_{AU}$	$P_{AU}$	$F1_{AU}$	$U_{AU}$
DS	111	2001	0	32	1968	10	1.6%	0.3%	0.2%	0.002	0.003	3	0.3%	0.2%	0.002	0.003
DS2	109	1991	0	28	1962	10	1.4%	0.3%	0.1%	0.002	0.003	4	0.3%	0.1%	0.002	0.003
CML (AllBut1)	59	943	0	172	770	10	18.3%	2.2%	2.3%	0.022	0.025	4	2.2%	2.3%	0.019	0.022

Table 2.17: Evaluation results for the TOPNPCBF technique. Same settings as for the PCBF technique were used, except that now a item model size of 30 ( $MS = 30$ ) was used.

### TOPNCBF Technique

Settings	ET	$U_s$	$U_f$	$TN_s$	$TN_f$	$TN_a$	Cov	R	P	F1	U	AFHP	$R_{AU}$	$P_{AU}$	$F1_{AU}$	$U_{AU}$
DS	49	3158	0	32	3125	10	1.0%	0.2%	0.1%	0.001	0.002	4	0.3%	0.1%	0.001	0.00

Table 2.18: Evaluation results for the TOPNCBF technique. Same settings as for the PCBF technique were used, except that now a item model size of 100 ( $MS = 100$ ) was used.

### TBTOPNUCF Technique

Settings	ET	$U_s$	$U_f$	$TN_s$	$TN_f$	$TN_a$	Cov	R	P	F1	U	AFHP	$R_{AU}$	$P_{AU}$	$F1_{AU}$	$U_{AU}$
DS	5	3158	0	1111	2047	10	35.2%	9.9%	4.6%	0.062	0.068	3	11.1%	4.6%	0.061	0.070
DS (All-But1)	5	3158	0	372	2785	10	11.8%	11.8%	1.2%	0.021	0.077	3	11.8%	1.2%	0.021	0.077

Table 2.19: Evaluation results for the TBTOPNUCF technique. In all instances the Transaction Cosine similarity algorithm (with  $OT = 2$ ,  $ST = 0$ ) and a neighborhood size of 30 ( $NS = 30$ ) was used.

### TBTOPNICF Technique

Settings	ET	$U_s$	$U_f$	$TN_s$	$TN_f$	$TN_a$	Cov	R	P	F1	U	AFHP	$R_{AU}$	$P_{AU}$	$F1_{AU}$	$U_{AU}$
DS	10	3158	0	464	2693	10	14.7%	3.6%	1.7%	0.023	0.023	5	3.8%	1.7%	0.022	0.022
DS (All-But1)	11	3158	0	146	3011	10	4.6%	4.6%	0.5%	0.008	0.023	5	4.6%	0.5%	0.008	0.023

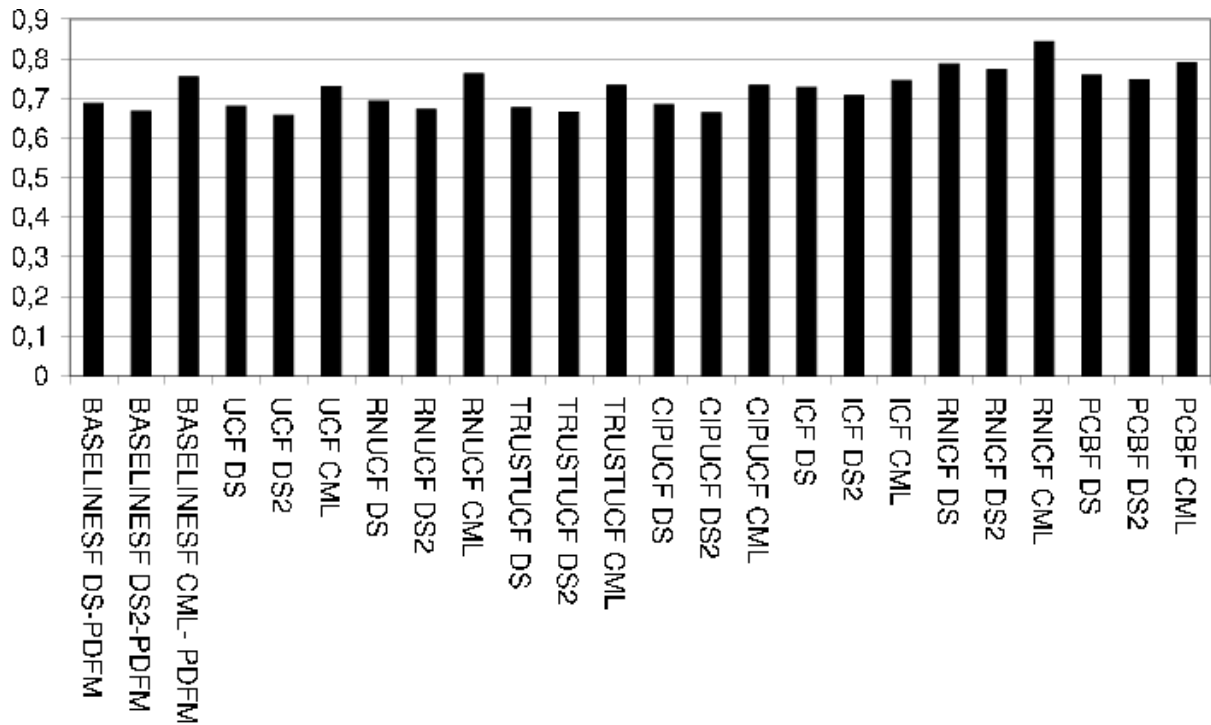
Table 2.20: Evaluation results for the TBTOPNICF technique. For the DS setting the Transaction Cosine similarity algorithm (with  $OT=2$ ,  $ST=10$ ) was used, and for the DS (AllBut1) setting the Transaction Cosine similarity algorithm (with  $OT = 2$ ,  $ST = 0$ ) was used. In all cases a model size of 30 ( $MS = 30$ ) was used.

## 2.8 Summary and conclusions

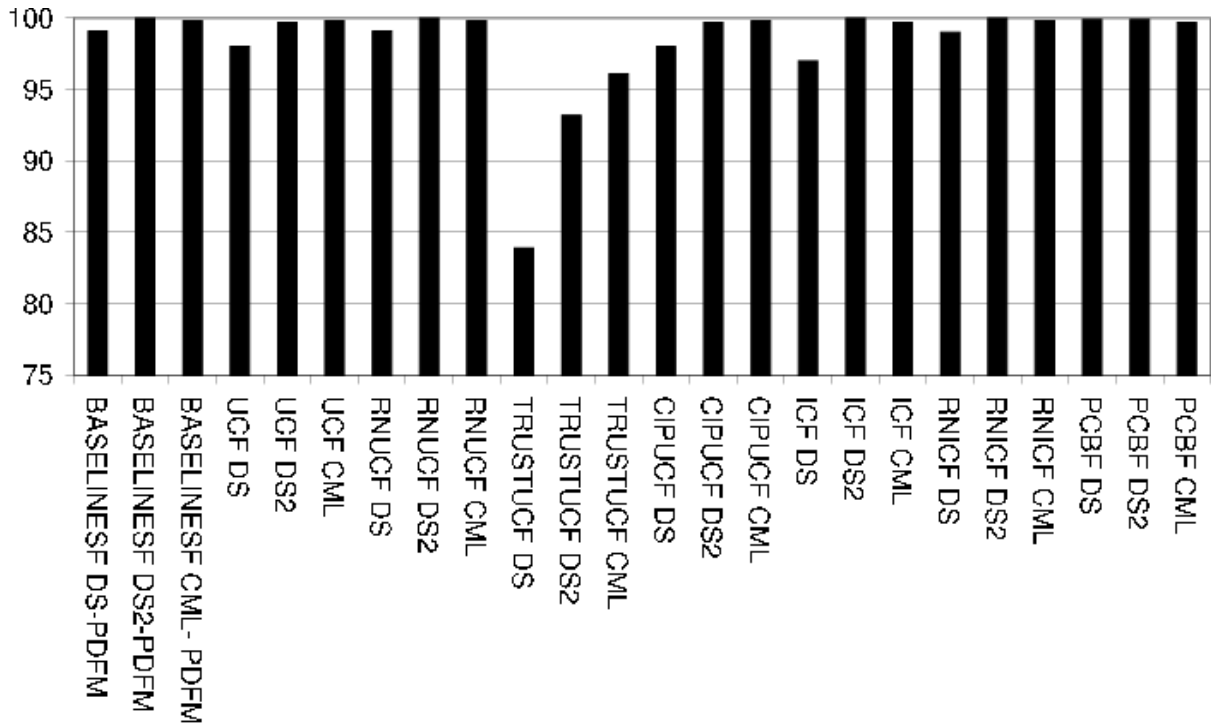
The results of the evaluations are summarized in terms of MAE and Coverage for the prediction evaluations [Figures 2.28(a), 2.28(b)], and in terms of Relevance and Precision for the relevance evaluation of Top-N ranking lists [Figures 2.29(a), 2.29(b)].

The following notable observations can be made on the evaluation results of the prediction accuracy.

1. Evaluation time Evaluations based on the DS dataset take the longest time in all cases. CML fastest.
2. Successful users and failed users In all cases the techniques were able to provide (almost) each user with at least one prediction. For the CML dataset  $U_s$  is max all the time, for the DS and DS2 dataset there's three notable cases where users fail to get any predictions, these are UCF, TRUSTUCF and CIPUCF, with trust being the most notable one with about 30 users.
3. Successful predictions and failed predictions Many failed predictions for TRUSTUCF, considerably worst for DS dataset. For all other techniques number of failed predictions is negligible. In those cases you can note something, it is for the DS dataset.
4. Coverage Near 100% coverage in all cases except for the TRUSTUCF technique for which it is considerably lower than for the rest. Otherwise in the cases we can note anything, it is for the DS dataset the coverage is slightly lower.
5. Correctness Never more than 50% correctness for any technique. For baseline random has worst correctness, and PDFM highest correctness in all cases. For all other techniques, the same pattern appears, with DS2 slightly better than DS and CML.



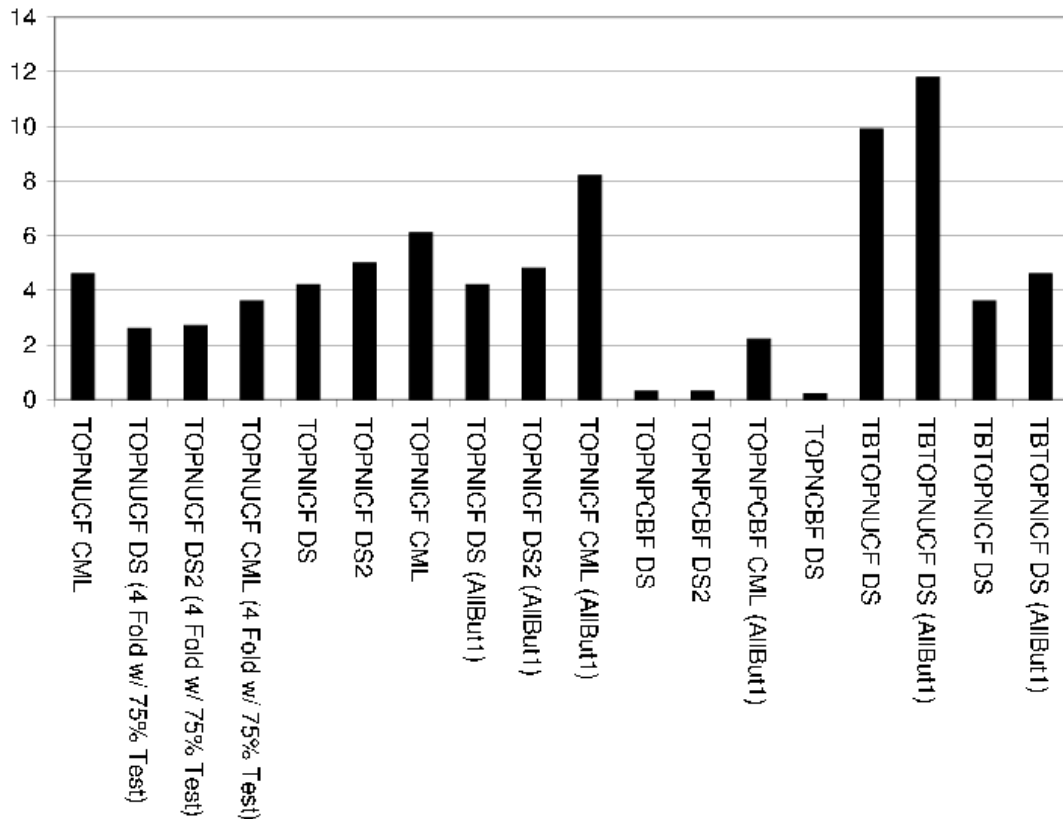
(a) MAE for evaluations of prediction accuracy.



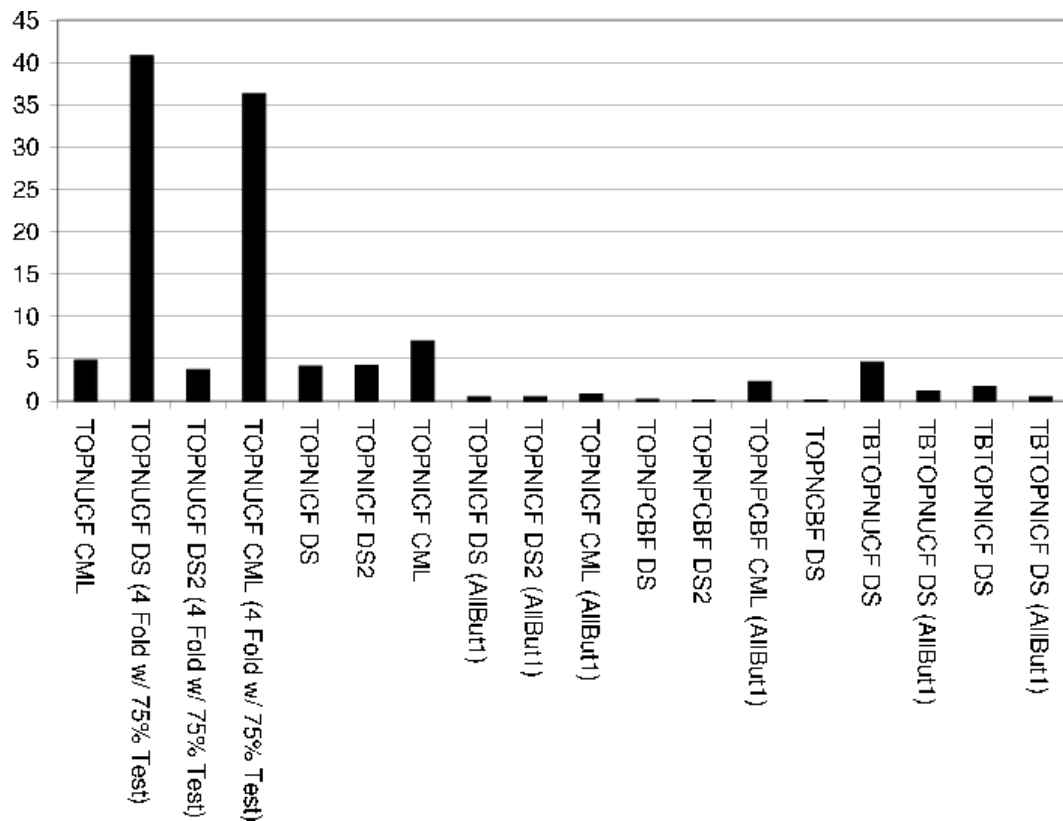
(b) Coverage for evaluations of prediction accuracy.

Figure 2.28: Prediction accuracy in terms of MAE and Coverage for a select set of evaluated recommendation algorithms. Note that MAE can at most be  $|R_{max} - R_{min}|$  (in this case 4), and that Coverage is measured in percent (%).





(a) Recall for evaluations of relevance of Top-N ranking lists.



(b) Precision for evaluations of relevance of Top-N ranking lists.

Figure 2.29: Ranking relevance in terms of Recall and Precision for all evaluated recommendation algorithms. Note that Recall and Precision is measured in percentage (%).

6. Accuracy metrics Notable for the baseline techniques, random has clearly worst MAE, almost twice the MAE of PDFM. We can note that the RNUCF and RNICF have twice as good MAE as the random baseline. If we try out different methods of forming neighborhoods, clearly we should want to outperform the RNUCF and RNICF techniques, and not just the baseline random predictions. For all techniques independent of dataset, we get a MAE of approximately  $0.7 \pm 0.05$ . For all remaining measurements of prediction accuracy (NMAE, MSE, etc.) the same pattern can be detected as for MAE. Meaning the accuracy is always best for the DS2 dataset, followed by DS and CML.

As we can see the choice of dataset has impact on the results, MAE varies (not much, but little) between the datasets, as does the coverage, therefore comparison of algorithm accuracy must be done on exactly the same datasets. All techniques, except the random ones, have more or less the same accuracy. Notable is that the baseline user mean and item mean which does not require much computing time nor memory are not far off from the best performing techniques. But using those techniques, you can not claim to produce personalized predictions. For example, the theoretically quite sound TRUSTUCF technique gives us more reason to state our predictions are personalized, but reduces coverage considerably and increases computation time. The reduction in coverage we believe has to do with the sparsity of the data, the CML dataset is least sparse, the coverage when using i.e. the TRUSTUCF technique is less noticeable (only shows a decrease in coverage of 5%) than for the DS and DS2 dataset. Choice of metric based on our evaluations does not matter much since the same characteristics can be seen independent of the metric. Therefore we agree with [Herlocker, 2000] who argues that a standardized metric for evaluation of accuracy of predictions should be the MAE.

In many papers an increase in the second or third decimal of the MAE seems to be reason for claims for a significant improvement in predictive accuracy of a recommender system. Sometimes statistical tests are used to ensure that this really is a significant improvement. However, such tests do not mean that *any* one user of the system will notice a difference, and only has a meaning when comparing against other evaluation results and your own algorithms (e.g. will a user care whether he is predicted 4 or 4.5?).

The following notable observations can be made on the evaluation results of the ranking list relevance:

1. Successful and failed Users Number of failed users is heavily dependent on the size of the test data, as is to be expected and can be seen when studying e.g. the evaluation results for the TOPNUCF technique for the CML dataset when using a 10-Fold and 4-Fold (with 75% test data) evaluation data set. Since we had the requirement that we wouldn't make any predictions unless the active user had seen at least 20 of the movies similar to the active movie, the number of failed users for the TOPNICF technique was higher for the DS and DS2 dataset than for the CML dataset since the average number of ratings per user is twice as high in the CML dataset. This requirement however increases the confidence of the recommendations, and was deemed a necessary requirement to get any useable results when using the TOPNICF technique. For all other techniques where applicable no such minimum restriction is put on the neighborhood sizes, and the number of failed predictions also goes down for those techniques.
2. Successful and failed Top-N lists For all techniques the number of failed Top-N lists is very high. For the CML dataset the best results are obtained, with around "only" half of the Top-N lists containing no relevant items. Notable is that almost none of the Top-N lists produced by the TOPNCBF technique contain any relevant items.
3. Average size of Top-N ranking list Same observations as for "Successful and failed Users".
4. Coverage Since, as observed, there are few successful Top-N lists the coverage should be low.
5. Recall and precision The effect of the size of the test data when it comes to recall and precision is clearly visible when studying e.g. the evaluation results for the TOPNUCF technique for the CML dataset when using 10-Fold and 4-Fold (with 75% test data) evaluation data sets. We see that with a small test data set with few known relevant movies, recall increases but at the same time precision decreases, as is to be expected. We also see the effect of having more known relevant movies, recall is decreased but precision is increased. Since as previously observed few Top-N lists containing any relevant items are returned, the recall should be low, similarly for precision. If we look at the recall for the TOPNUCF technique for the DS and DS2 dataset we see that the recall

is almost the same, but precision is 40.8% for the DS dataset and only 3.7% for the DS2 dataset. The change of dataset clearly resulted in a decreased precision. If we look at the number of Top-N lists we see that for the DS dataset there's many failed Top-N lists, i.e. lists with no relevant items, however the majority of these failed Top-N lists are also empty, this is not the case for the DS2 dataset where almost all failed Top-N lists are not empty. Since precision is the number of returned relevant items divided by the total number of returned items, the empty Top-N lists accounts for the higher precision.

6. F1 and Utility An interesting observation is that even though F1 and Utility are two different formulas, they produce almost identical results in all cases except when AllBut1 is used. Which is due to the fact that the F1 metric is dependent on the size of the test data, while Utility isn't.
7. AFHP Interesting to note is that in most cases the first known relevant item is on average encountered on the fourth place in the Top-N lists.

Overall, given our results for the ranking evaluation, none of the algorithms produce Top-N lists containing especially many items that had been assumed relevant for the user. In related studies of Top-N recommendations, such as [McLaughlin and Herlocker, 2004], [Sarwar et al., 2000a] and [Karypis, 2001], reported precision (for similar datasets) is overall low and high recall is not reported together with high precision.

The measuring of relevance for ranking lists proves quite difficult, recall and precision is awfully low within the evaluation model applied. For the same recommendation algorithm, higher precision can be achieved when manipulating the form of the evaluation data such that the test dataset is larger, however this decreases recall. Instead the goal is to naturally to alter the algorithm to increase precision, without decreasing recall.

It is not always clear how relevance is defined in different studies, as discussed previously different definitions are possible and must be made clear. In our evaluations rated items were considered relevant and all other items were considered non-relevant. This means that for each user there is a larger number of non-relevant items which will bias the results towards low relevance.

In summary, when evaluating your recommendation algorithms, pay attention to additional accuracy measures when evaluating the prediction accuracy, look at MAE, Coverage and Correctness as well. Also give thought to low measurements, consider e.g. whether a low coverage such as 40% as opposed to a near 100% coverage is really a bad measurement, if for example the MAE goes up with 100% coverage, is it then not better to settle for 40% coverage with low MAE instead? As also suggested elsewhere [Riedl and Dourish, 2005], the focus on evaluation of recommender systems, adjusting algorithms to lower MAE and other measurable quantities, should not be the only focus. Focus should be made on creating useable recommender systems with innovative, experimental and interesting ways of interacting and understanding the recommender system. This does not mean only focusing on the interface, but a focus on from the beginning selecting sensible or interesting approaches to recommendation systems where in the end the recommendation trail is what's presented to the user. As such no incorrect recommendations can be made, eliminating the need for a algorithmic evaluation, only well motivated recommendations will be made that either appeal or do not appeal to the user, but which are never strictly incorrect predictive guesses.



## Part II

# Self-Organization applied to Recommender Systems

*“Progress isn’t made by early risers.  
It’s made by lazy men trying to find  
easier ways to do something.”*

– Robert Heinlein

This part consists of three chapters. The first two are theoretical, giving the necessary background for chapter three, in which we use the knowledge from the previous chapters in our attempt to implement and evaluate recommender systems based on the SOM paradigm.

The purpose of the first chapter is to give the reader a basic understanding in what artificial neural networks are. Motivations behind the development of artificial neural networks are discussed, the basic biological neuron and the artificial neuron are described, network architectures and learning processes are outlined, and a brief historical summary of some of the main events in the field of artificial neural networks is given.

The second chapter deals with self-organizing systems in which Kohonen’s SOM is the main topic. Self-organization as a biological phenomena is discussed and the requirements for self-organization are described. Then, after a short introduction to the SOM paradigm, the incremental SOM algorithm is outlined in detail together with a presentation of how parameters for the algorithm should be selected, how to measure the quality of the SOM visualization and clustering issues in the SOM context. Some important properties regarding the final SOM is also presented and a discussion of the theoretical aspects of the SOM is given. Presentations of related work in the area of using the SOM paradigm in recommender systems are given at the end of the chapter.

Finally, in chapter three, we describes *why* we have chosen to use the SOM paradigm as a part of a recommender system by referring back to Part I and the issues regarding recommender systems that was presented there. We proceed by showing *how* we have implemented different recommender systems based on the SOM paradigm. This is done with pseudo code and descriptive commentaries of the code. The systems are then evaluated in the same manner as in Part I.

## Chapter 3

# Artificial neural networks

*Artificial neural network* is a discipline that draws its inspiration from biological neural networks in the human brain. Artificial neural networks resembles the brain in two respects: the network acquires knowledge from its environment using a *learning process (algorithm)* and stores the learned information in interneuron connection strengths called *synaptic weights*. At the cellular level, the human brain consists primarily of nerve cells called *neurons*. Neurons are *parallel* working units connected to each other in a network style. The human brain contains approximately  $10^{11}$  neurons. Each of these neurons is connected to around  $10^3$  to  $10^4$  other neurons, and therefore the human brain is estimated to have  $10^{14}$  to  $10^{15}$  connections. The neuron is the basic building block of the nervous system and most neurons are to be found in the brain.

### 3.1 The brain

The brain can be seen as a highly *complex, nonlinear, and parallel information-processing system*. It has the capability of organizing neurons so as to perform certain computations (e.g. pattern recognition, perception, and motor control) many times faster than the fastest digital computer. The brain is almost completely enveloped by the cerebral cortex. Although it is only about 2-3 mm thick, its surface area, when spread out, is about 2000  $cm^2$  (roughly three times this paper). Typically, neurons are five to six orders of magnitude slower than silicon logic gates; events in a silicon chip happen in the nanosecond ( $10^{-9}$  s) range, whereas neural events happen in the millisecond ( $10^{-3}$ s) range. However, the brain makes up for the relatively slow rate of operation of a neuron by letting its huge number of neurons operate in parallel. The net result is that the brain is an enormously efficient structure [Haykin, 1994].

The long course of evolution has also given the human brain many other characteristics that are desirable in an artificial neural network. Some of these include:

**Robustness and fault tolerance** Nerve cells can die without any significant loss of performance by the brain.

**Adaptivity and learning** The brain is trained - not programmed - and adapts therefore easily to new environments.

**Fuzziness** Information can be inconsistent and noisy.

**Massive parallelism**

At birth, the distinct areas of the brain are all at place and within each brain area are millions of neurons that are connected to each other by *synapses*. These synapses and the pathways they form make up the neural network in the brain. After birth, the brain development continues and consists of wiring and rewiring the connections (synapses) between neurons. New synapses are created and old ones disappears, all depending on what kind of experience the brain is exposed to. The most dramatically development of the brain takes place in the first two years from birth but it continues well beyond that stage. During this early stage of development, about one million synapses are formed per second.

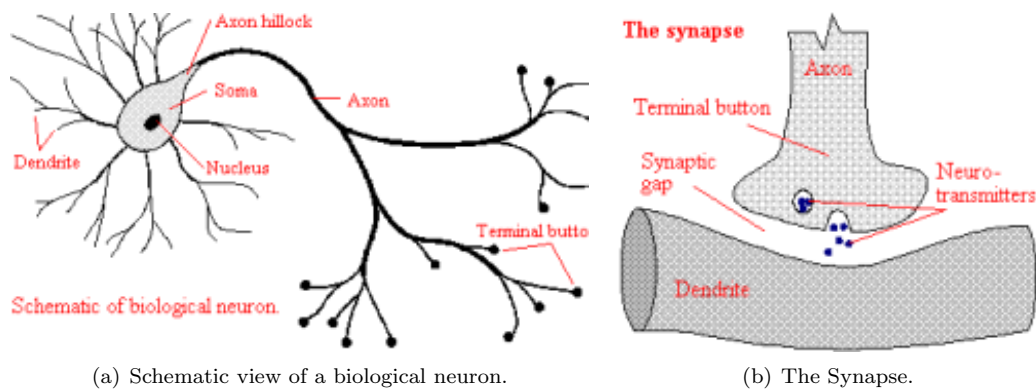


Figure 3.1: Within the brain the biological neurons are connected by way of synapses.

## 3.2 The biological neuron

Neurons can be of many types and shapes, but ultimately they function in a similar way and are connected to each other in a rather complex network stylish way via strands of fibre called *axons*. A neurons axon acts as a transmission line and are connected to another neuron via that neurons *dendrites*, which are fibres that emanate from the cell body (*soma*) of the neuron. The junction that allows transmission between the axons and the dendrites are the synapse. Synapses are elementary structural and functional units that creates the signal connection between two or more neurons; sometimes meaning the connection as whole. [Figure 3.1]<sup>1</sup>

The most common kind of synapse is a chemical synapse: when a neuron is stimulated, it transmit that nerve pulse to another neuron through the axons and causes the release of *neurotransmitters* that travel through the synapse to the other neuron. That is, a *presynaptic* process liberates a transmitter substance that diffuses across the synaptic junction between neurons and then acts on a *postsynaptic* process. Thus a synapse converts a presynaptic electrical signal into a chemical signal and then back into a postsynaptic electrical signal. In traditional descriptions of neural organization, it is assumed that a synapse is a simple connection that can impose *excitation*(positively) or *inhibition*(negatively) on the receptive neuron.

A developing neuron is synonymous with a plastic or adaptive brain. *Plasticity* permits the developing nervous system to adapt to its surrounding environment, i.e. the creation of new synaptic connections between neurons and the modification of existing synapses. It is this type of adaptation that forms the basis for learning. Since plasticity appears to be essential to the functioning of neurons as information processing units in the human brain, the same goes for neural networks made up of artificial neurons. A phrase that sometimes appears in the context of artifical neural networks is the *stability-plasticity dilemma*, which refers to the dilemma of a system having to be adaptive enough to allow for learning of new knowledge (plasticity) while not forgetting previous learned knowledge (stability).

## 3.3 The artificial neuron

Artificial neurons are information-processing units that are only approximations (usually very crude ones) of the biological neuron [Figure 3.2]. Three basic elements of the artificial neuron can be identified[Haykin, 1994]:

- A set of *synapses* or *connection links*, each of which is characterized by a *weight* or *strength* of its own. Specifically, a component  $x_i$  at the input of synapse  $i$  connected to a neuron  $k$  is multiplied by the synaptic weight  $w_{ik}$ . Note that the first subscript of the weight refers to the input end of the synapse and the second to the neuron in question. The weight  $w_{ik}$  is positive if the associated synapse is excitatory; it is negative if the synapse is inhibitory.
- An *summation function* for summing the input signals, weighted by the respective synapses of the neuron and producing a linear output  $S_k$ . Sometimes referred to as a linear combiner.

<sup>1</sup>Figures are taken from the website <http://vv.carleton.ca/~neil/neural/neuron-a.html>

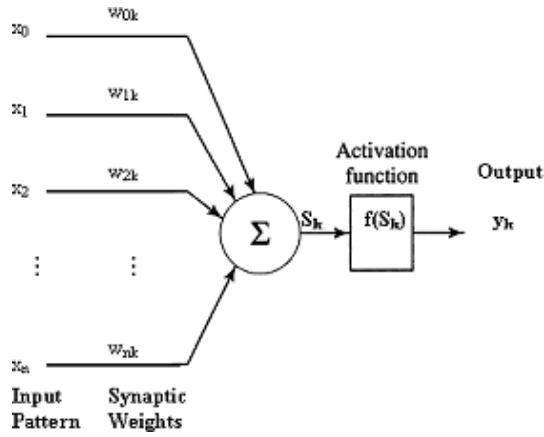


Figure 3.2: Artificial neuron

- An *activation function* for limiting the amplitude of the output of a neuron. In general, activation functions are monotonically increasing functions, where, (excluding the linear function)  $f(-8) = 0$  or  $f(8) = -1$  and  $f(8) = 1$ , i.e. the output of a neuron can be written as the closed unit interval  $[0, 1]$  or alternatively  $[-1, 1]$ .

The model of a neuron also includes an fixed input  $x_0 = 1$  and a weight  $w_{0k} = -\theta_k$  called the threshold. Mathematically, we can describe the artificial neuron as follows:

$$S_k = \sum_{i=1}^n w_{ik}x_i - \theta_k = \sum_{i=0}^n w_{ik}x_i \quad (3.1)$$

$$y_k = f(S_k)$$

where  $x_1, x_2, \dots, x_n$  are the input signals;  $w_{1k}, w_{2k}, \dots, w_{nk}$  are the synaptic weights of neuron  $k$ ;  $S$  is the sum of the input signals and the threshold;  $f(S_k)$  is the activation function and  $y_k$  is the output signal of neuron  $k$ .

The computational process can be described as follows: an artificial neuron receives its inputs  $x_1, x_2, \dots, x_n$  from some external source (the real world), computes a weighted sum  $S$  (usually) of the input signals (including the threshold) using the summation function. This sum  $S$  is then used as input for the activation function  $f(S)$  that generates the final output  $y$ .

Different types of activation functions can be used and three of them are described in [Figure 3.3]. The most commonly used is nonlinear *sigmoid* activation functions such as the *logistic function*. A logistic function assumes a *continuous* range of values form 0 and 1 in contrary to the discrete threshold function. A binary threshold function was used in the first model of an artificial neuron back in 1943, the so-called *McCulloch-Pitts model*. Threshold functions goes by many names, e.g. *step-function*, *heavyside function*, *hard-limiter* etc. Common for all is that they produce one of two scalar output values (usually 1 and  $-1$  or 0 and 1) depending on the value of the threshold. Another type of activation function is the *linear* function or some times called the *identity* function since the activation is just the input. In general if the task is to approximate some function then the output nodes are linear and if the task is classification then sigmoidal output nodes are used.

Mathematically and graphically, we can describe the functions as in [Figure 3.3].



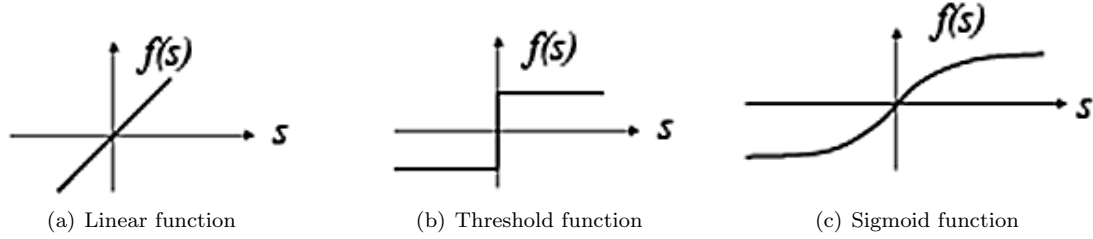


Figure 3.3: (a) The linear function  $f(S) = \alpha f(S)$  produces a linearly modulated output, where  $\alpha$  is a constant (if  $\alpha$  is 1 it becomes the identity function). (b) The threshold function  $f(S)$  in this cases uses a threshold of zero causing the output to be either 1 or -1, i.e.  $f(S) = 1$  if  $S \geq 0$  and  $-1$  if  $S < 0$ . (c) In the sigmoid function  $f(S) = \frac{1}{1+e^{-\lambda S}}$  that is shown (the logistic function)  $\lambda$  controls the steepness of the function, and is usually equal to -1.

A nice property of the sigmoid function is that its derivative [Equation 3.2] is easy to calculate, which is an important feature in network theory.

$$\frac{\delta f(S)}{\delta S} = f(S)(1 - f(S)) \quad (3.2)$$

If every aspect of a biological neuron was to be considered in a computer simulation, then one would need more than all the computing resources that are available to theorist now days [Kohonen, 2001]. Therefore, artificial neurons and hence, artificial neural networks are only approximations to parts of the real brain, and the extent to which a artificial neural network approximate a particular part of the brain usually varies. This means, that if the goal is to understand the principles on which the human brain works, then the biological plausibility of the artificial neural network is of primary concern, otherwise, the artificial neural network models can be seen as practical inventions for new components and techniques inspired by the brain.

### 3.4 Formal definition of artificial neural networks

From a practical point of view, an artificial neural network is a *parallel computational systems*, inspired by the *structure*, *processing methods* and *learning ability* of the human brain, consisting of many simple *processing elements* (artificial neurons) connected together in a specific way (network style) in order to perform a particular task.

One of the most powerful features of artificial neural networks are their ability to *learn* and *generalize* from a set of *training data*. Training data is used only for training the network to perform a certain task. In order to test how good the network is on performing the particular task it has been trained for, a special data set called *test data* is used that contains examples that was not included in the training set. Learning means that the network adapt the strengths/weights of the connections between neurons until enough knowledge about the training data is stored in the network so it can perform the task it been trained for. Generalization means that the network after training can recognize inputs correctly that was not part of the training dataset, hence the importantly of using test data that has not been included in the training data. Note that knowledge is stored in the weights not in the neurons, i.e. the final configuration of the weights represent the networks knowledge. Four basic rules when it comes to knowledge representation in a neural network are given in [Haykin, 1994]:

1. Similar inputs from similar classes of input should usually produce similar representations inside the network.
2. Data categorized as belonging to separate classes should give widely different representations in the network.
3. If a particular feature is important, then there should be a large number of neurons involved in the representation of that data in the network.

4. Prior information and invariance's should be built into the design of a neural network, thereby simplifying the network design by not having to learn them.

From the above discussion, we can adapt Haykins definition of a neural network [Haykin, 1994]:

*"A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:*

1. *Knowledge is acquired by the network through a learning process.*
2. *Interneuron connection strengths known as synaptic weights are used to store the knowledge."*

The procedure used to perform the learning process is called a *learning algorithm*, the function of which is to modify the synaptic weights of the network. The modification of synaptic weights provides the traditional method for the design of neural networks, however, it is also possible for a neural network to modify its own topology, which is motivated by the fact that neurons in the brain can die and that new synaptic connections can grow.

### 3.5 Artificial neural network structures and learning algorithms

[Figure 3.4] provides a functional description of the various elements that constitute the model of an artificial neuron. A more simpler way of representing the structure of an artificial neuron and particular artificial neural networks is with the so-called *architectural graph* [Figure 48] which describes the layout of an artificial neural network. Haykin [Haykin, 1994] also represents neural networks in terms of *signal-flow graphs* in which artificial neurons are *nodes* and *directed links* are connections between nodes. In the terms of a signal-flow graph, the architectural graph is to be considered as a partially complete signal-flow graph.

Two main structures of artificial neural networks, based on the arrangement of nodes and the connection patterns of the layers, are *feedforward* networks and *recurrent* networks. A layered neural network is a network of neurons organized in the form of layers. In the simplest form it consists of one *input layer* of *source nodes* and one *output layer* consisting of one or more neurons. However, multilayered networks are by far the most common architecture of any neural network, in particular the popular *multilayer perceptrons* feedforward network. The difference is the presence of one or more *hidden layers* of neurons. The function of the hidden neurons is to intervene between the external input and the network output, typically, the neurons in each layer of the network have as their inputs the output signals of the preceding layer only. By adding one or more hidden layers, the network is capable of modeling more complex relationships between the input variables and the output variables.

#### 3.5.1 Feedforward networks

The name "feedforward" came from that neuron interconnections are acyclic, i.e. input nodes are connected to output nodes, but not vice versa. [Figure 3.4] actually shows an *Single-Layer Feedforward*

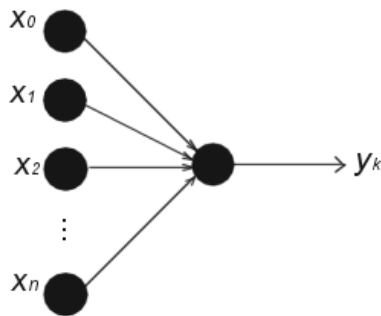


Figure 3.4: The structure of a artificial neuron.

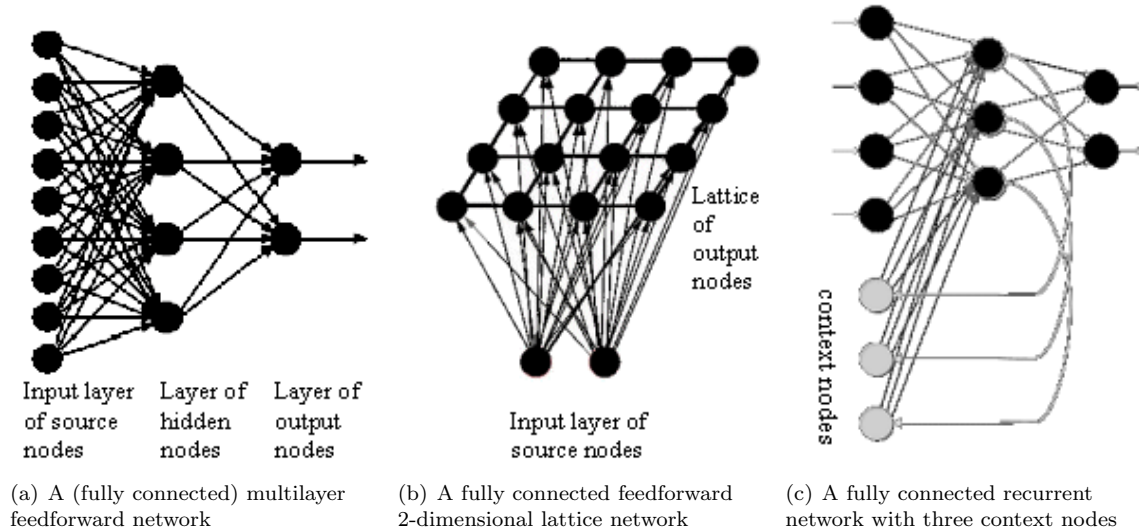


Figure 3.5: Neural networks

network consisting of only one neuron in the output layer. Another class of feedforward networks are *multilayer Feedforward networks*, as shown in [Figure 3.5(a)]. A feedforward network can be fully or partially connected in the sense that every node in each layer is either connected or not connected to every node in the adjacent forward layer. Another type of feedforward network is called a *lattice* network in which the output neurons are arranged in rows and columns. Usually the lattice consists of one or two-dimensional array (although higher dimensions are possible) of neurons with a corresponding set of source nodes. Each source node is connected to every neuron in the lattice [Figure 3.5(b)]. The most known example of this kind of network is Kohonen's SOM.

### 3.5.2 Recurrent networks

Recurrent network has input nodes, hidden neurons, output neurons, just as feedforward networks, what distinguish themselves from feedforward networks is that they have at least one *feedback* loop, i.e. when a neurons output is fed back into the network as input. A special type of feedback loop are called *self-feedback* loops, which refers to a situation in which the output of a neuron is fed back to its own input. Another type of feedback loop introduces a new type of nodes called *context* nodes [Figure 3.5(c)]. These nodes receive connections from the hidden neurons or the output layers of the network and have output connections that travel back to the hidden neurons. Context nodes are required when learning patterns over time (i.e. when the past value of the network influences the present processing). Recurrent networks can therefore be seen as an attempt of incorporate time and memory into a neural network Two examples of recurrent networks that are simple extensions of feedforward networks are *Jordan network* (feedback from output layer to input layer) and *Elman network* (feedback from hidden layer to input layer). Also, a widely known recurrent network is the *Hopfield network* in which all the connections are symmetric.

The structure of an artificial neural network is closely related with the learning algorithm used to train the network, different network architectures require appropriate learning algorithms.

## 3.6 Learning Rules

A learning algorithm refers to a procedure in which *learning rules* are used for adjusting the weights, which formally can be defined in the context of neural networks as follows [Haykin, 1994]:

*"Learning is a process by which the free parameters (the weights) of a neural network are adapted through a continuing process of stimulation by the environment in which the network*

is embedded. The type of learning is determined by the manner in which the parameter changes take place. Which implies the following sequence of events:

1. The neural network is stimulated by an environment.
2. The neural network undergoes changes as a result of this stimulation.
3. The neural network responds in a new way to the environment, because of the changes that have occurred in its internal structure."

Mathematically, this can be describes as follows. Let  $w_{ik}(t)$  denote the value of the synaptic weight  $w_{ik}$  at time  $t$  between two neurons  $i$  and  $k$  or a input node  $i$  and a neuron  $k$ . At time  $t$  an *adjustment*  $\Delta w_{ik}(t)$  is applied to the synaptic weight  $w_{ik}(t)$ , yielding the updated value:

$$w_{ik}(t+1) = w_{ik}(t) + \Delta w_{ik}(t) \quad (3.3)$$

where  $w_{ik}(t)$  and  $w_{ik}(t+1)$  is the *old* and *new* values of the synaptic weight  $w_{ik}$  and  $\Delta w_{ik}$  is the adjustment applied to the old value of  $w_{ik}$  as a result of the stimulation from some environment.

The reason why time is involved in the equation is that the network gradually adapts the synaptic strengths, so  $t$  can be regarded as the number of times the neuron has been updated, or how many training examples that has been run trough.

There is no unique learning algorithm for the design of neural networks, different learning algorithms has advantages of their own. In general, learning algorithms differ from each other in the way in which the adjustment  $\Delta w_{ik}$  to the synaptic weight  $w_{ik}$  is formulated.

Four basic types of learning rules are [Haykin, 1994]: *error-correction learning*, *Hebbian learning*, *competitive learning* and *Boltzmann learning*. Error-correction learning is rooted in optimization theory whereas Hebbian and competitive learning are inspired by neurological considerations. Boltzmann learning is different altogether in that it is based on ideas borrowed from thermodynamics and information theory. A brief explanation of error correction learning, Hebbian learning and competitive learning will be described below. Competitive learning will also be described in more detail in the next chapter on SOMs.

### 3.6.1 Error-correction learning

Typically, the *actual* output  $y_k(t)$  produced by some vector  $x(t)$  applied to the input of the network at time  $t$  in which neuron  $k$  is embedded differs from the *desired* output  $d_k(t)$ . Let the difference between the desired and actual output define an *error signal*  $e_k(t) = d_k(t) - y_k(t)$ .

The error signal  $e_k(t)$  is then used to adjust the values of the synaptic weights so that the output signal approaches the desired response  $d_k(t)$  in a step-by-step manner.

The simplest form of error-correction rule is the *perceptron convergence procedure*, invented by Rosenblatt in the 50s and published 1960. This rule is nonlinear, adjusting the weights makes use of the "quantize error"  $e_k$ , defined to be the difference between the desired output  $d_k$  and the output of the quantizer  $y_k$ , e.g. a binary threshold activation function [Widrow and Lehr, 2003]. The weights are updated using [Equation 3.3] with:

$$\Delta w_{ik} = \begin{cases} 0 & \text{if } y_k = d_k \\ +x_i & \text{if } y_k = 0 \text{ and } d_k = 1 \\ -x_i & \text{if } y_k = 1 \text{ and } d_k = 0 \end{cases} \quad (3.4)$$

That is, if  $e_k = 0$ , no adjusting of the weights takes place. Otherwise, updating the weights is done by adding the input vector  $x$  to the weight vector  $w$  if the error is positive, or by subtracting the input vector  $x$  from the weight vector  $w$  if the error is negative. The input  $x_i$  is also multiplied with a positive *learning-rate parameter*  $\eta$ . For stability, the learning rate should be decreased to zero as iterations progress and this affects the plasticity.

Another type of error correction rule is based on the *gradient descent method*. The purpose of the gradient descent method is to minimize a *cost function* based on the error signal  $e_k(t)$ , such that the actual response of each output neuron in the network approaches the target response for that neuron.

Once a cost function has been chosen, this becomes an optimization problem<sup>2</sup> to which the theory of optimization can be applied.

A criterion commonly used for the cost function is the *instantaneous value* of the *mean-square-error* criterion

$$\mathcal{E}(t) = \frac{1}{2} \sum_k^2 e_k^2(t) \quad (3.5)$$

where summation is over all neurons in the output layer.

The network is then optimized by minimizing the error function  $\mathcal{E}(t)$  with respect to the synaptic weights of the network. The weights are updated using [Equation 3.3] with:

$$\Delta w_{ik}(t) = \eta \left( -\frac{\partial \mathcal{E}}{\partial w_{ik}} \right) \quad (3.6)$$

The learning-rate parameter  $\eta$  is a positive constant which determines the amount of correction. Thus,  $\Delta w_{ik}$  is the adjustment (correction) made to weight  $w_{ik}(t)$  at time  $t$ .

The derivation of

$$\frac{\partial \mathcal{E}}{\partial w_{ik}} \quad (3.7)$$

using the notation from [Figure 3.4], i.e. the derivative of the cost function  $\mathcal{E}$  with respect to the weights is calculated using the chain rule which gives us the following partial derivatives:

$$\frac{\partial \mathcal{E}}{\partial w_{ik}} = \frac{\partial \mathcal{E}}{\partial y_k} \times \frac{\partial y_k}{\partial S_k} \times \frac{\partial S_k}{\partial w_{ik}} \quad (3.8)$$

which are equal to:

$$\begin{aligned} \frac{\partial \mathcal{E}}{\partial y_k} &= 2 \times \frac{1}{2} \times e_k \times -1 = -e_k \\ \frac{\partial y_k}{\partial S_k} &= \frac{\partial f(S_k)}{\partial S_k} = f'(S_k) \\ \frac{\partial S_k}{\partial w_{ik}} &= \frac{\partial \sum_{i=0}^n w_{ik} x_i}{\partial w_{ik}} = x_i \end{aligned} \quad (3.9)$$

Hence

$$\frac{\partial \mathcal{E}}{\partial w_{ik}} = -e_k \times f'(S_k) \times x_i \quad (3.10)$$

Which, in general form is:

$$\Delta w_{ik}(t) = \eta \times e_k(t) \times f'(S_k(t)) \times x_i(t) \quad (3.11)$$

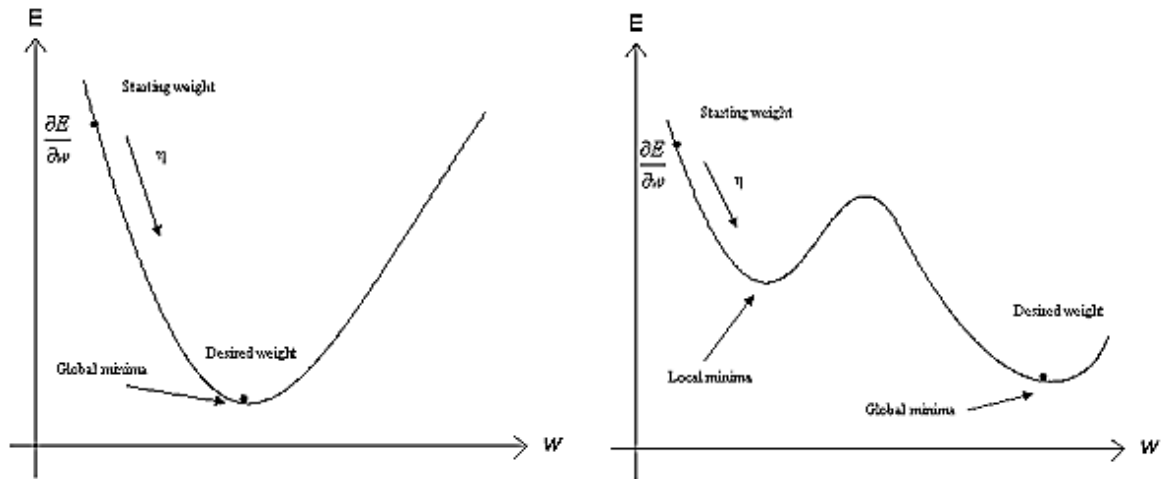
where  $w_{ik}(t)$  is the  $i$ -th element of the weight vector  $w_k(t)$  of the output neuron  $k$  and  $x_i$  is the corresponding  $i$ -th component of the input vector  $x(t)$ .  $f'(S_k(t))$  is the derivative of the activation function.

The first ones to published such a rule was Windrow and Hoff, who in 1960 published the *least mean square* algorithm or *Windrow-Hoffs delta rule* as it also is referred to. This rule is linear, adjusting the weights makes use of the "linear error"  $e_k$ , defined to be the difference between the desired output  $d_k$  and the output of the *linear combiner* [Widrow and Lehr, 2003], hence  $y_k = S_k$ , if we use the notation from [Figure 3.4]. Which gives,  $e - k = d_k - S_k$ , and since  $f(S_k) \equiv S_k$ , its derivative is equal to 1. Using [Equation 3.11], we get that:

$$\Delta w_{ik}(t) = \eta \times e_k(t) \times x_i(t) \quad (3.12)$$

---

<sup>2</sup>In optimization theory, the object of a computational problem is to find the best possible solutions among all feasible solutions which has the minimum (or maximum) value of an objective function. Objective functions can be thought of as cost functions which determines how good a solution is, for instance, the total cost of edges in the travelling salesman problem [Sant, 2005]



(a) The error surface in the linear case for one weight. The gradient of  $E$  in weight space are calculated and the weight are moved along the negative gradient.

(b) The error surface in the nonlinear case for one weight. The gradient of  $E$  in weight space are calculated and the weight are moved along the negative gradient.

Figure 3.6: Error surfaces

The weights are then updated using [Equation 3.3].

This learning rule is also *local* since it uses information directly available to neuron  $k$  through its synaptic connections. The learning rate parameter  $\eta$  is crucial to the performance of error-correction learning in practice as it determines stability, convergence speed and the final accuracy.

A generalization of this rule is called the *generalized delta* learning rule and assumes continuous activation functions which are at least once differentiable, hence the popular choice of the logistic function as the activation function, since it is both continuous and has an easy calculated derivative. We will not go in to detail here, but this generalization makes it possible to train multilayer networks with sigmoidian activation functions in the hidden layer. The rule is nonlinear and adjusting the weights makes use of the "sigmoidian error"  $e_k$ , defined to be the difference between the desired output  $d_k$  and the output of the sigmoidian  $y_k$  (e.g. the logistic function).

The most popular learning algorithm that implements this is the *backpropagation* algorithm. This algorithm consists of two phases:

1. *Feedforward pass*, in which the output error is calculated.
2. *Backward propagation*, in which the error signal is propagated back from the output layer toward the input layer and on its way to the input layer, the weights are adjusted as functions of the backpropagated error, hence the name of the algorithm.

The principle is the same as for the delta rule, we are only applying it to more nodes in several layers and making the chain in the chain rule a bit longer.

To illustrate the gradient decent method, we can plot the error function against the weights. This plot illustrates a multi-dimensional weight space, usually called the *error-surface*. In the linear case, e.g. the least mean square algorithm, the error function becomes quadratic in its parameters and a global minimum solution can easily be found. A plot for the error surface for one weight in the linear case is shown in [Figure 3.6(a)].

However, in most cases the output of an artificial neural network is a nonlinear function of its parameters because of the choice of the activation function, e.g. the sigmoid functions, which is the case in the general delta rule. As a result, it is no longer straightforward to derive at a solution for the weights that is guaranteed to be globally optimal. In the case of the back-propagation algorithm, one way to avoid this is to add a momentum term into the equation. This is easier pictured if we think of the solution as a ball on the error surface, bouncing back and fourth, and whenever it got trapped in a local minima on the error surface, that extra *momentum term* will enforce that ball to bounce out of that minima and

keep it bouncing until it got trapped in a global minima. At least, that's the idea. A plot of a nonlinear error surface is shown in [Figure 3.6(b)] for one weight.

In the case of SOMs, discussed in detail in the next chapter, it has been proven that it is impossible to associate a global decreasing cost function to the SOM in the general case [Erwin et al., 1992].

### 3.6.2 Hebbian Learning

Within the field of artificial neural networks, Hebbian learning is an unsupervised training algorithm in which the synaptic weight  $w_{ik}$  (strength) from neuron  $i$  to neuron  $k$  is increased if both neuron  $i$  and neuron  $k$  are active at the same time. A natural extension of this is to decrease the synaptic strength when the source neuron  $i$  and output neuron  $k$  are not active at the same time. Hence, the adjustment applied at time  $t$  to the synaptic weight  $w_{ik}$  with presynaptic and postsynaptic activities denoted by  $x_i$  and  $y_k$  can be expressed in mathematical terms as:

$$\Delta w_{ik}(t) = F(y_k(t), x_i(t)) \quad (3.13)$$

where  $F$  is a function of both the postsynaptic and presynaptic activities.

As a special case we may use the activity product rule:

$$\Delta w_{ik}(t) = \eta y_k(t) x_i(t) \quad (3.14)$$

where  $\eta$  is the learning rate parameter.

Hence, Hebbian learning can then be expressed as:

$$w_{ik}(t+1) = w_{ik}(t) + \eta y_k(t) x_i(t) \quad (3.15)$$

Of course, from this representation we see that the repeated application of the input signal  $x_i$  leads to an *exponential growth* that finally drives the synaptic weight  $w_{ik}$  into saturation. One way to avoid this is to impose some sort of limit on the growth of synaptic weights. One method for doing this is to introduce a *forgetting factor* into the formula as follows:

$$\Delta w_{ik}(t) = \eta y_k(t) x_i(t) - \alpha y_k(t) w_{ik}(t) \quad (3.16)$$

where  $\alpha$  is a new positive constant and  $w_{ik}(t)$  is the synaptic weight at time  $t$ .

Equivalently, we can write the above expression as the *generalized activity product rule*:

$$\Delta w_{ik}(t) = \alpha y_k(t) [c x_i(t) - w_{ik}(t)] \quad (3.17)$$

where  $c = \eta/\alpha$ . If the weight  $w_{ik}(t)$  increases to the point where

$$c x_i(t) - w_{ik}(t) = 0 \quad (3.18)$$

a *balance point* is reached and the weight update stops.

### 3.6.3 Competitive learning

Competitive learning is a process in which the output neurons of the network compete among themselves to be activated with the result that only *one* output neuron, is on at any time. There are three basic elements to a competitive learning rule [Haykin, 1994]:

1. A set of similar neurons except for some randomly distributed synaptic weights. Therefore the neurons respond differently to input signals.
2. A limit imposed on the strength of each neuron.
3. A competing mechanism for the neurons.

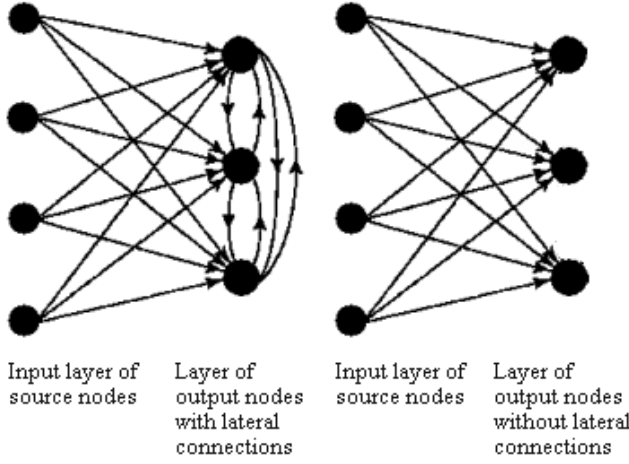


Figure 3.7: Two simple forms of a competitive neural network. Feedforward connections from source nodes to the neurons are excitatory and the connections among the neurons are inhibitory.

The output neurons that wins the competition are called *winner-takes-all neurons* and as a result of the competition, they become specialized to respond to specific *features* in the input data. In the simplest form of competitive learning, the neural network has a single layer of output neurons, each of which is fully connected to the input nodes [Figure 3.7].

The network *may* include lateral connections among the neurons, as shown in [Figure 3.7], and thereby perform lateral inhibition, with each neuron tending to inhibit the neuron to which it is laterally connected. A neuron  $k$  is the winning neuron if its net internal activity level  $S_k$  for a given input vector  $x$  is the largest one among all neurons in the network, where  $S_k$  is the combination of all the forward and feedback inputs for neuron  $k$ . The output signal  $y_k$  of winning neuron  $k$  is set to one and all the others to zero. Each neuron is allowed a *fixed* amount of synaptic weight (typically, all synaptic weights are positive), which is distributed among its input nodes; that is, we have  $\sum_i w_{ik} = 1$  for all  $k$  and where  $w_{ik}$  denotes the synaptic weight between input node  $i$  and neuron  $k$ . A neuron learns by shifting synaptic weights from its inactive input nodes to the active ones, no learning takes place if the neuron don't respond to a particular input vector  $x$ . The change  $\Delta w_{ik}$  applied to synaptic weight  $w_{ik}$  is defined by the standard competitive learning rule:

$$\Delta w_{ik} = \begin{cases} \eta(x_i - w_{ik}) & \text{if neruon } k \text{ wins} \\ 0 & \text{if neruon } k \text{ loses} \end{cases} \quad (3.19)$$

where  $\eta$  is the learning parameter. The effect of this rule is that the synaptic weight vector  $w_k$  of the winning neuron  $k$  is moved toward the input vector  $x$ .

### 3.7 Learning paradigms

Another factor to be considered is the manner in which a neural network relates to its environment during learning, i.e. what kind of information is available to the network. In this context, a *learning paradigm* refers to an *model* of the environment in which the neural network operates. There are three main learning paradigms: *supervised*, *unsupervised* and *reinforcement learning*. Kohonen [Kohonen, 2001] explains the difference in the three learning paradigms as follows.

**Supervised learning** Learning with a teacher, i.e. a learning scheme in which the average expected difference between wanted output for training samples and the true output is decreased, using e.g. the backpropagation algorithm.

**Unsupervised learning** Learning without a priori knowledge about the classification of samples, i.e. learning without a teacher. Often the same as formation of clusters, after which the clusters can be labeled, e.g. Kohonen's SOM algorithm.



**Reinforcement learning** Learning mode in which adaptive changes of the parameters due to reward or punishment depend on the final outcome of a whole sequence of behavior. The results of learning are evaluated by some performance index.

### 3.8 Learning protocols

The ways in which the input vectors are presented to the networks determine the learning protocol in which the network learns its environment by adjusting its parameters. Two main principles are the following ones:

**Incremental Learning** Weights are updated after each presentation of a training vector  $x$ . Also known as pattern learning or stochastic learning if the training vectors are chosen randomly.

**Batch Learning** Weights are updated after all training vectors have been presented once for the network, i.e. the changes of the weights are accumulated until all vectors in the training set have been presented. Also known as epoch learning.

### 3.9 A brief history of artificial neural networks

We will round off this short introduction to artificial neural networks with a brief look at some important milestones within the research field of neural networks<sup>3</sup>.

One early reference concerning the theory about neural networks can be made to Alexander Bain [Wilkes and Wade, 1997]. In his book *"Mind and Body: Theories of their relation"* from 1873, he relates the process of associative memory to the distribution of activity in neural grouping or neural networks in modern terms. He found out that individual nerve cells carrying excitatory links to selected other cells within a grouping, that each nerve cell is summing the degree of stimulation it receives from other parts of the network, and that the same network can generate different output depending on its internal connections. And what even more interesting is that his rule for associations (groupings) precedes that of a later one, better known as Hebb's hypothesis from 1949. With Bains own words as reprinted in [Wilkes and Wade, 1997]:

*"when two impressions concur, or closely succeed one another, the nerve currents find some bridge or place of continuity, better or worse, according to the abundance of nerve matter available for the transition. In the cells or corpuscles where the currents meet and join, there is, in consequence of the meeting, a strengthened connection or diminished obstruction – a preference track for that line over lines where no continuity has been established."*

Bain emerges as the first theorist that goes beyond generalized statements towards providing detailed examples and network drawings. Unfortunately, Bain himself came to doubt about his work and not even one of his own students, David Ferrier who published the classical work *"The Functions of the Brain"* in 1876 mention the details of Bains work. Therefore, much of what Bain discovered is now credited by other researchers.

However, the modern era of neural networks is said to have begun with the pioneering work of McCulloch and Pitts in the 40s. Warren McCulloch (a psychiatrist and neuroanatomist) and Walter Pitts (a mathematician and logician) published a paper 1943 entitled *"A logical calculus of the ideas immanent in nervous activity"* in the Bulletin of Mathematical Biophysics. In this paper, they proposed a mathematical model of a biological neuron based on the *all-or-nothing theory* of biological neurons, i.e. the output from a neuron is constant; either it is maximal or its nothing. What they proposed was a binary threshold unit as a computational model for an artificial neuron which corresponded to their view of the nervous system as a network of finite interconnection of logical devices. The McCulloch-Pitts neuron works on binary signals and in discrete time; they receive one or more input signals and produce an output signal in a similar way as described earlier in figure 2. The activation function they used was a binary threshold function or step function, with a threshold at 0. The central conclusion of their paper

---

<sup>3</sup>The historical notes here are largely (but not exclusively) based on the following two sources: chapter 1 of the book by Haykin [Haykin, 1994] and the introduction part in [Arbib, 2003]. If the authors find it necessary, specific references will be made.

was that any finite logical expression can be realized by a network built of McCulloch-Pitts neurons. This conclusion was important in the sense that it showed that networks of rather simple elements connected together can be computationally very powerful. However, in their model, the weights are constant, there is no learning. The McCulloch-Pitts paper had an great influential, and it was not long before the next generation of researchers added learning and adaptation.

Donald O. Hebb, a Canadian neurophysiologist, presented in his book "*The Organization of Behaviour*", from 1949 a theory of behaviour based on the physiology of the nervous system. The most important concept to emerge from this work was his formal statement (known as Hebb's postulate) of how learning could occur. Learning was based on the modification of synaptic connections between neurons. Specially, as restated in [Haykin, 1994],

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

The principle underlying this statement have become known as Hebbian learning.

Hebb was proposing not only that, when two neurons where active together the connection between the neurons is strengthened, but also that this activity is one of the *fundamental operations necessary for learning and memory*. This meant that the McCulloch-Pitts neuron had to be altered to at least allow for this new biological proposal.

One of the first to exploit this idea, to make a learning rule for artificial neurons, was Frank Rosenblatt in the late 50s. He used the McCulloch-Pitts neuron and the findings of Hebb in the developing of the *perceptron*. The perceptron was originally designed as a linear pattern<sup>4</sup> classifier, i.e, used for the classification of a special type of patterns, which are *linearly separable*. This means that the input vectors can be seen as points in a  $N$ -dimensional input space and what the perceptron does is to put a *discriminant* in this space. This discriminant defines a hyperplane in the  $N$ -dimensional space.

In two dimension this discriminant is a line and can be found where the linear output  $S_k$  equals zero:

$$S_k = x_1w_1 + x_2w_2 + w_0 = 0 \quad (3.20)$$

therefore

$$x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2} \quad (3.21)$$

[Figure 3.8] graphs this linear relation, which comprises a separating line having slope and intercept given by:

$$\begin{aligned} \text{slope} &= \frac{w_1}{w_2} \\ \text{intercept} &= \frac{\theta}{w_2} \end{aligned} \quad (3.22)$$

Rosenblatt's learning algorithm, referred (as previously described) to as the *perceptron convergence procedure*, was used to train the perceptron. Although the core idea of the perceptron was the incorporation of learning into the McCulloch-Pitts neuron, the thing that really caused a lot of enthusiasm in the world of computer science was that Rosenblatt presented this learning algorithm together with a theorem, called the *perceptron convergence theorem*. This theorem says that this learning algorithm will converge to an optimal discriminant in a number of finite steps, *if such a discriminant exists*. This theorem created a big hype regarding perceptrons and caused a lot of activity in the 50s and the 60s, people thought that now we can compute anything, solve any problem, we don't have to program anymore and

---

<sup>4</sup>Pattern recognition is one of the most common applications of a neural network. Pattern recognition is a combination of feature extraction and classification. Feature extraction results in feature vecors that are used as input vectors that are to be classified. Essentially, one may say that any application for neural networks can been transformed to a pattern recognition task even if the actual application wasn't pattern recognition, since the input to the network can be called patterns in any case

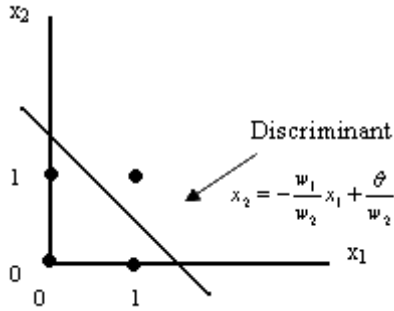
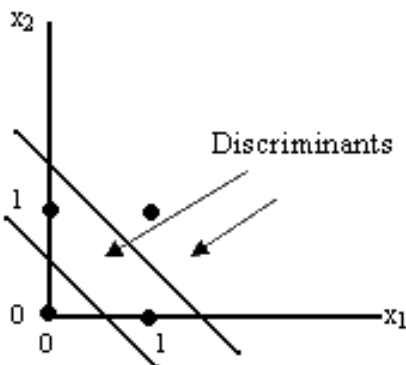


Figure 3.8: A 2-dimensional input space. Four points  $(0,0)$ ,  $(1,0)$ ,  $(0,1)$  and  $(1,1)$  in a two dimensional input space are spanned by  $x_1$  and  $x_2$ . Training of a perceptron with two inputs  $x_1$  and  $x_2$  would then be to adjust the weights  $w_1$  and  $w_2$  such that the weights (and the threshold  $-\theta$  with  $x_0 = 1$ ) defines the slope and position of a line that separates the points into two classes. In this case class one consists of the points  $(0,0)$ ,  $(1,0)$ ,  $(0,1)$  and class two of the single point  $(1,1)$ . In this particular case, the perceptron has been trained to implement the boolean function AND, i.e. when its fed with the points from class one, it will output 0 (false) and the point from class two will yield a 1 (true). It acts just as a logic gate as in McCulloch and Pitts case. The difference is that the perceptron is able of finding suitable weights automatically.

so on. Unfortunately, a lot of people most have missed this last point, because its not always true, which will soon be explained why.

About the same time as Rosenblatt, Bernard Widrow and Marcian 'Ted' Hoff of Stanford developed the Adaline (ADAPtive LInear NEuron) which was very similar to the perceptron. The adaline is an adaptive threshold logic element with a hardlimiter activation function outputting  $+ - 1$ . The principal difference between the perceptron and the adaline lies in the training procedure, adaline uses the least mean square algorithm, as explained earlier, to train the adaline [Widrow and Lehr, 2003].

Unfortunately, both Rosenblatt's and Widrow's networks suffered from the same inherent limitations, which were publicized in the influential book "*Perceptrons: An Introduction to Computational Geometry*" in 1969 by Minsky and Papert. This book contained a detailed analysis of the capabilities and limitations of perceptrons. In their book, Minsky and Papert used the XOR-problem as an example for the demonstration that the perceptron can only do linearly separable problem. Of course, this was nothing new among those who did research on perceptrons and its variants, Rosenblatt himself had done some research regarding this [Carpenter, 1989]. The XOR problem can be considered as a classification problem for the perceptron to solve. That is, the perceptron can be constructed as a XOR logic function that has two inputs  $(x_1, x_2)$  and one output  $y$  which should equal 1 if the input is  $(0,1)$  or  $(1,0)$  and a 0 if the input is  $(1,1)$  or  $(0,0)$ , see the truth table for the exclusive-or function in [Table 3.1]. This input space can be plotted as in [Figure 3.9] and the aim of the perceptron is to find a discriminant that separates this points.



$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0

Table 3.1: The exclusive-or (XOR) function's truth table.

Figure 3.9: A 2-dimensional input space.

As can be seen from [Figure 3.9], no *single* line can separate the points, e.g. the problem is linearly inseparable. A perceptron cannot solve any problem that is linearly inseparable. The solution to the XOR problem in this case is to use more than one perceptron, e.g. *multilayer perceptrons*. In the first layer, two perceptrons are set up to identify small, linearly separable sections of the inputs, e.g. one perceptron implements the OR function and the other implements the AND function and then combining their outputs into another perceptron in the output layer, which produce a final indication of which class the input belongs to. Of course this solution to the XOR problem, that it is possible to construct a network with layers of perceptrons with *predefined* weights, was also widely known. But what Minsky and Papert's stated in their book was that *it is not possible to find a learning algorithm that can find the correct weights automatically*. This problem is also known as the *credit assignment problem*, since it means that the network is unable to determine which of the input weights should be increased and which should not, i.e. the problem of assigning credit to the perceptrons in the network so they can produce a better solution next time.

This hasty conclusion made by Minsky and Papert's had some devastated effects on the research community, lot of people that financed this kind of research read this book, and almost over one night, all funding into this field was stopped. This, combined with the fact that there were no powerful computers at the time to run experiments on, caused many researchers to leave the field. For example, a research team [Haykin, 1994] involved in the developing of a nonlinear learning filter, spent six years building the filter with analog devices.

The years that followed is sometimes referred to as the "ice age" by researchers in the neural network community, because almost no one looked into the fields of neural networks except for researchers in psychology and neuroscience, but from a physics and engineering perspective, the field was almost completely dead [Haykin, 1994]. This period of time lasted between 1969 to mid 80s. Despite this, some important activities and results was made during this period, where some notable ones are the work by James Anderson and Teuvo Kohonen on associative memories in 1972 and the pioneering work on competitive learning and self-organizing maps by von der Malsburg [von der Malsburg, 1973], Stephen Grossberg [Grossberg, 1976] and also Kunihiko Fukushima, who explored related ideas with his biologically inspired Cognitron [Fukushima, 1975] and Neocognitron [Fukushima, 1980] models. It was also during this time period that Kohonen began his work on SOMs that he later published in 1982 [Kohonen, 1982].

The interest in neural networks re-emerged after two important theoretical results were attained in the early eighties. In 1982, the American scientist John Hopfield used the idea of an energy function to formulate a new way of understanding the computation performed by recurrent networks with symmetric synaptic connections. This had the positive effect that physics became interested in the field. The second was the solution to the credit-assignment problem by the rediscovery of the backpropagation algorithm in 1986. By replacing the discontinuous threshold function used by the perceptron with a continuous smooth "sigmoid function" one could use the generalized delta rule, as explained earlier, for the minimization of the output error, i.e. it was now possible to find out how much credit should be assign to the perceptrons so their weights could be updated.

This algorithm was in fact described as early as 1974 by Paul Werbos in his Ph.D. thesis *"Beyond regression: new tools for prediction and analysis in the behavioral sciences"* at Harvard University. Werbos's work should unfortunately remain almost unknown in the scientific community until 1986, when Rumelhart, Hinton, and Williams rediscovered the technique and, within a clear framework, described their work in the article *"Learning Internal Representations by error propagation"*, published in the two-volumed book *"Parallel Distributed Processing: Explorations in the Microstructures of cognition"* by Rumelhart and McClelland, in that same year. This contributed to making the method widely known and this book has since then been a major influence in the use of back-propagation learning.

In the last twenty years, thousands of papers have been written, and neural networks have found many applications such as character recognition, image recognition, credit evaluation, fraud detection, insurance, and stock forecasting. The field is showing new theoretical results and practical applications. The use of neural networks as a tool to be used in appropriated situations and the fact that we know very little about how the brain works makes the future of neural networks to look very promising.

## Chapter 4

# Self-Organizing Maps

*“The SOM paradigm was originally motivated by an attempt to explain some functional structures of the brain. Quite surprisingly, however, the SOM turned out to be very useful in exploratory data analysis and started to live a life of its own: some 6000 scientific articles have been based on it.”* – Teuvo Kohonen, WSOM 2005, 5th Workshop On Self-Organizing Maps.

One of the main features of artificial neural networks is the ability to *adapt* to an environment by *learning* in order to *improve* its performance to carry out whatever task its being trained for. The learning paradigm we will be working with here is the unsupervised learning paradigm and we will use it with competitive learning rules.

Unsupervised learning can also be thought of as self-organized learning [Haykin, 1994], in which the purpose is to discover significant patterns or features in the input data, and to do the discovery without a teacher. To do so, a self-organizing learning algorithm is provided with a set of rules of *local* nature, which enables it to learn to compute an input-output mapping with specific desirable properties (by local it is meant that the effect of changing a neurons synaptic weight is confined to its immediate neighborhood). Different structures of self-organizing system can be used. For example, feedforward networks consisting of an *input layer* and an *output layer* with lateral connections between the neurons in the output layer can be used or a multilayer feedforward network in which the self-organization proceeds on a layer-by-layer basis. Regardless of the structure, the learning process consists of repeatedly modifying the synaptic weights of all the connections in the system in response to some input, usually represented by a data vector, in accordance with the set of prescribed rules, until a final configuration develops. Of course, the key question here is how a useful configuration finally can develop from self-organization. The answer to this can be found in the following observation made by *Alan Turing* in 1952:

*“Global order can arise from local interaction.”*

This observation defines the essence of self-organization in a neural network, i.e, many randomly local interactions between neighboring neurons can coalesce into states of global order and ultimately lead to coherent behaviour.

Haykin [Haykin, 1994] identifies three different Self-Organizing Systems: Self-organizing systems based on *Hebbian learning*, systems based on *competitive learning*, also known as self-organizing (feature) maps and systems rooted in information theory, emphasizing the principle of *maximum information preservation* as a way of attaining self-organization. Only Self-Organizing systems based on competitive learning are considered in this text, i.e. SOMs.

In a SOM, the neurons are placed at the nodes of a lattice that is usually one or two dimensional (higher dimensions are possible but not as common) and in the final configuration of the map, which depends on the competitive learning process, the localized feature-sensitive neurons respond to the input patterns in a orderly fashion, as if a curvilinear coordinate system, reflecting some topological order of events in the input space, were drawn over the neural network. A SOM is therefore characterized by the formation of a *topographic map* of the input data vectors, in which *the spatial locations (i.e. coordinates) of the neurons in the lattice correspond to intrinsic features of the input patterns*, hence the name “*self-organizing (feature) map*”.

The SOM algorithm was developed by Teuvo Kohonen in the early 1980s [Kohonen, 1982] in an attempt to implement a learning principle that would work reliably in practice, effectively creating *globally ordered maps* of various sensory features onto a layered neural network. Kohonen's work in SOMs and related earlier work of David Willshaw and Cristoph von der Malsburg [Willshaw and von der Malsburg, 1976] and Stephen Grossberg [Grossberg, 1976], [Amari, 1980] were all inspired by the pioneering work of von der Malsburg in 1973 [von der Malsburg, 1973]. Kohonen [Kohonen, 1990] credits the above studies for their great theoretical importance concerning self-organizing tendencies in neural systems. However, the results of the ordering of neurons was weak, no "maps" of practical importance could be produced, ordering was either restricted to a one-dimensional case, or confined to small parcelled areas of the network. Kohonen realized that this was due to the fact that the system equations used to form the maps have to involve much stronger, idealized self-organizing effects, and that the organizing effect has to be maximized in every possible way before useful global maps could be created. After some experimenting with different architectures and system equations, a process description was found that seemed to produce globally well-organized maps [Kohonen, 1982]. In its computationally optimized form, this process is now known as the SOM algorithm [Kohonen, 1990]. If not explicitly stated, the main reference source for the rest of this chapter is Kohonen's book entitled "*Self-Organizing Maps*", sometimes also referred to as [Kohonen, 2001].

## 4.1 Brain maps

The nineteenth century saw a major development in the understanding of the brain. At an overall anatomic level, a major achievement was the understanding of localization in the cerebral cortex, e.g. that the cortex of the human brain is organized according to functional areas such as speech control and analysis of sensory signals (visual, auditory, somatosensory, etc) [Figure 4.1(a)]. Around 1900, two major steps were taken in revealing the finer details of the brain. In Spain, pioneering work in brain theory was done by Ramón y Cajál (e.g. 1906), whose anatomical studies of many regions of the brain revealed that the particular structure of each region is to be considered as a network of neurons. In England, the physiological studies of Charles Sherrington (1906) on reflex behaviour provided the basic physiological understanding of synapses, the junction points between the neurons [Arbib, 2003].

More recent experimental studies about the finer-structure of the brain areas has revealed that, for example, some areas are ordered according to some feature dimensions of the sensory signals, like the somatosensory area, where response signals are obtained in the same topographical order on the cortex in which they were received at the sensory organs. These structures are called maps [Figure 4.1(b)]. There are three types of neuronal organizations that can be called "brain maps": sets of feature-sensitive cells (single neurons or groups of neurons that respond to some distinct sensory input), ordered projections between neuronal layers and ordered maps of abstract features (feature maps that reflect the central properties of an organism's experiences and environment). These brain maps can extend over several centimetres such as, for example, the somatotopic map. Based on numerous experimental data and observations, it thus seems as if the internal representations of information in the brain are organized and spatially ordered, i.e. groups of neurons dealing with similar features are also located near each other. Although the main structure of the brain is determined genetically, there are reasons, due to its complexity and dynamical nature, that some other fundamental mechanisms are used in the creation of the brain structure. It was the possibility that the representation of knowledge in a particular category of things in general might assume the form of a feature map that is geometrically organized over the corresponding piece of the brain that made Kohonen to believe that one and the same general functional principle might be responsible for self-organization of widely different representations of information [Kohonen, 2001].

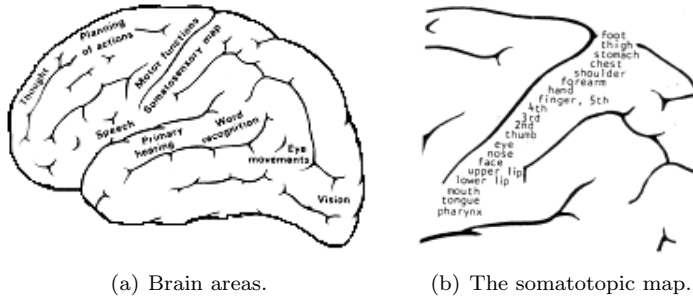


Figure 4.1: Images from [Kohonen, 2001] showing the localization in the cerebral cortex and ordering in the somatosensory area of the brain.

## 4.2 Requirements for self-organization

The self-organizing process can be realized in any set of elements, illustrated schematically in [Figure 4.2], where only a few basic operational conditions are assumed [Kohonen and Hari, 1999]: the broadcasting of the input, selection of the winner, and adaptation of the models in the spatial neighborhood of the winner.

Lets consider an two-dimensional array of elements, e.g. neurons, where each element is represented by a *model*  $m_i$  consisting of a set of numerical values (these values is usually related to some parameters of the neuronal systems such as the synaptic efficacies), and where each model will be modified according to the messages that the elements receives. First of all, assume the presence of some mechanism that makes it possible to compare the ingoing message  $x$  (a set of parallel signal values) with all models  $m_i$ . In brain theory it is customary to refer to "competition" between the elements when they are stimulated by common input, and the element with the most similar model to the ingoing message is activated the most. This element is called the "winner" if it succeeds, while remaining active itself, in suppressing the activity in the neighboring elements by, for example, lateral inhibition. The winner model is denoted by  $m_c$ . The above competitive process can be described as in [Kohonen, 1990]: Competitive learning is an adaptive process, in which the neurons of the neural network are tuned to specific features of input. The response from the network then tend to become localized. The basic principle underlying competitive learning stems from mathematical statistics, namely, cluster analysis. The process of competitive learning can be described as follows:

Assume a sequence of statistical samples of a vector  $x = x(t) \in \mathbb{R}^n$  where  $t$  is time, and a set of variable reference vectors  $m_i(t) : m_i \in \mathbb{R}^n, i = 1, 2, 3, \dots, k$ .

Assume that the  $m_i(0)$  have been initialized in some proper way, random selection may do.

Competitive learning then means that if  $x(t)$  can be simultaneously compared with each  $m_i(t)$  at each successive instant of time then the best-matching  $m_i(t)$  is to be updated to match even more closely the current  $x(t)$ .

If comparison is based on some distance measure  $d(x, m_i)$ , updating must be such that if  $i = c$  is the

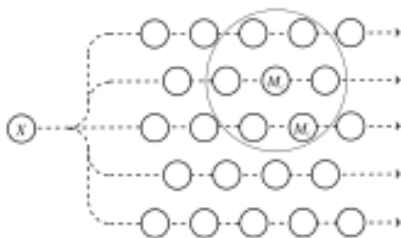


Figure 4.2: [Kohonen and Hari, 1999] A self-organizing set. An input message  $x$  is broadcast to a set of models  $m_i$ , of which  $m_c$  best matches  $x$ . All models that lie in the vicinity of  $m_c$  (larger circle) improve their matching with  $x$ .  $M_c$  differs from one message to another.

index of the best-matching reference vector, then  $d(x, m_c)$  shall be decreased, and all the other reference vectors with  $i \neq c$  left intact.

In the long run different reference vectors becomes sensitized to different domains of the input variable  $x$ . If the probability density function  $p(x)$  of the samples  $x$  is clustered, then the  $m_i$  tend to describe the clusters.

The final requirement for self-organizing is then that all the models in the local vicinity of the winning model(s) also get modified such that they also will resemble the prevailing message more accurately then before. When the models in the neighborhood of the winner start simultaneously to resemble the prevailing message  $x$  more accurately, they also tend to become more similar mutually, i.e. the differences between all the models in the neighborhood of  $m_c$  are smoothed. After the models  $m_i$  have been exposed for different messages at different time steps all over the array, they tend to start acquire values that relate to each other smoothly over the whole array, in the same way as the original messages  $x$  in the input space do. The result is the emergence of a topographically organized map.

### 4.3 The Mexican Hat

The best self-organizing results are obtained if the following two partial processes are implemented in their purest forms:

1. Decoding of the neuron "the winner" that best match with the given input pattern.
2. Adaptive improvement of the match in the neighborhood of the neurons centered around the "winner"

The former operation is signified as the winner takes all function. Traditionally, the winner takes all function has been implemented in neural networks by *lateral-feedback connections*. For an example of a SOM algorithm that implements lateral feedback, see [Sirosh and Miikkulainen, 1993].

If we consider a one-dimensional lattice of neurons as depicted in [Figure 4.2] we see that it contains two types of connections; there are *feedforward connections* from an external excitation input source to the neurons in the lattice (the output layer) with associated weights and there are a set of internal *feedback connections* between the nodes in the lattice, both *self-feedback* and *lateral feedback* ones, also with associated weights. The magnitude and type (excitatory and inhibitory) of these weights are functions of the geometrical distance between the neurons on the lattice. Following biological motivations, a typical function that represents the values of the lateral weights versus the distance between neurons on the lattice, is the *Mexican hat* function. The function is proportional to the second derivative of the gaussian probability density function and is formally written as [Equation 4.1], its shape is depicted in [Figure 4.3].

$$f(x) = \left( \frac{2}{\sqrt{3}} \pi^{-1/4} \right) (1 - x^2) e^{-x^2/2} \quad (4.1)$$

According to the shape of the Mexican hat function, we may distinguish three distinct areas of lateral interactions between neurons [Kohonen, 2001] as depicted in [Figure 4.3].

By means of lateral interactions, positive(excitation) and negative(inhibition), one of the neurons becomes the "winner" with full activity, and by negative feedback it then suppresses the activity of all other cells. For different inputs the "winners" alternate.

Hence, in presence of lateral feedback, the dynamic behavior of the network activity can be described as follows [Haykin, 1994]:

Let  $x_1, x_2, \dots, x_p$  denote the input signals (excitations) applied to the network, where  $p$  is the number of nodes in the input layer [Figure 4.4].

Let  $N_j$  define the region over which lateral interactions are active, usually referred to as the *neighborhood* of neuron  $i$  and let  $c_{ik}$  denote the lateral feedback weights connected to neuron  $i$  inside this neighborhood.

Let  $y_1, y_2, \dots, y_N$  denote the output signals of the network, where  $N$  is the number of neurons in the network.

The output signal (response) of neuron  $i$  is then equal to:



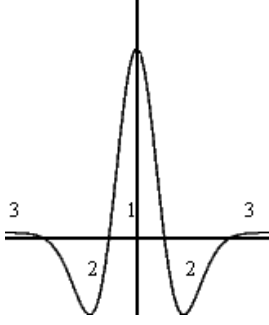


Figure 4.3: One-dimensional "Mexican hat" function. The horizontal axis represents the distance in the lattice between neuron  $i$  and its neighbor neurons. The vertical axis represents the strength of the connections between neuron  $i$  and its surrounding neighborhood. (1) A short-range lateral excitation area. (2) A penumbra of inhibitory action. (3) An area of weaker excitation that surrounds the inhibitory penumbra; this third area is usually ignored.

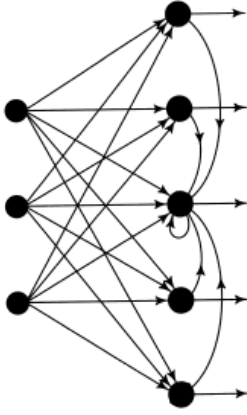


Figure 4.4: One-dimensional lattice of neurons with feedforward connections and lateral feedback connections; the latter connections are shown only for the neuron at the center of the array. First column of neurons make up the input layer, second column of neurons make up the output layer.

$$y_i = f \left( I_i + \sum_{k \in N_i} c_{ik} y_k \right) \quad i = 1, 2, \dots, m \quad (4.2)$$

where  $f$  is a standard sigmoid-type nonlinear function that limits the value of  $y_i$  and ensures that  $y_i = 0$ . The total external control exerted on neuron  $j$  by the weighted effect of the input signals is described by:

$$I_i = \sum_{l=1} p w_{il} x_l \quad (4.3)$$

The solution to the nonlinear equation [Equation 4.2] is found iteratively:

$$y_i(n+1) = f \left( I_i + \beta \sum_{k \in N_i} c_{ik} y_k(n) \right) \quad i = 1, 2, \dots, m \quad (4.4)$$

where  $n$  denotes an iteration step. The parameter  $\beta$  controls the rate of convergence of the iteration process. [Equation 4.4] represents a feedback system. The system includes both positive and negative feedback, corresponding to the excitatory and inhibitory parts of the Mexican hat function, respectively. The limiting action of the nonlinear activation function  $f$  causes the spatial response  $y_j(n)$  to stabilize in a certain fashion, dependent on the value assigned to  $\beta$ . If  $\beta$  is large enough, then in the final

state corresponding to  $n \rightarrow \inf$ , the values of  $y_j$  tend to concentrate inside a spatially bounded cluster, referred as a "activity bubble". The bubble is centred at a point where the initial response  $y_j(0)$  due to the external input  $I_j$  is maximum. The width of the bubble is governed by the ratio of the excitatory to inhibitory lateral connections: the bubble becomes wider for more positive feedback, and sharper for more negative feedback. If the negative feedback is too strong, formation of the activity bubble is prevented. Consequence of bubble formations is the topological ordering among the output neurons.

However, the SOM algorithm proposed by Kohonen [Kohonen, 1990] uses a "shortcut" or a "engineering solution" to the model described above. He suggested that the effect accomplished by lateral feedback, i.e. the bubble formation, *can be enforced directly by defining a neighborhood set  $N_c$  around the winning neuron  $c$* . The lateral connections and its weights are discarded and a topological neighborhood of active neurons that corresponds to the activity bubble is introduced instead. Further, for each input signal, *all the neurons within  $N_c$  are updated, whereas neurons outside  $N_c$  are left intact*. By adjusting the *size* of the neighborhood set  $N_c$ , lateral connections can be emulated, making it larger corresponds to enhanced positive feedback, while enhancement of negative feedback is obtained by making it smaller.

The computational gain with this model is that instead of recursively computing the activity of each neuron according to [Equation 4.4], this model finds the winning neuron and *assumes* that the other neurons are also active if they belong to the neighborhood of the winning neuron. Another problem with the former model, is that the trained map can exhibit several regions of activity instead of just one. This is an immediate consequence of the Mexican hat function for lateral interactions which primarily influence neurons nearby, thus far away regions of activity has little interaction.

## 4.4 The SOM paradigm

With this short introduction we want to emphasize that the SOM and the SOM algorithm refers to two different aspects of the SOM paradigm, something that's not always made clear in the literature. The SOM is the *result* of the *converged* SOM algorithm.

The basic SOM is as a nonlinear, ordered, smooth mapping of high-dimensional data manifolds onto the elements of a regular, low-dimensional array. The mapping is implemented in the following way:

1. Define the set of input variables  $x_j$  as a real vector  $x = [x_1, x_2, \dots, x_n]^T \in \mathbb{R}^n$ .
2. Associate with each element in the SOM array a parametric real vector  $m_i = [w_{i1}, w_{i2}, \dots, w_{in}]^T \in \mathbb{R}^n$  called a *model*.
3. Define a distance measure between  $x$  and  $m_i$  denoted  $d(x, m_i)$ .

The *image* of an input vector  $x$  on the SOM array is then defined as the array element  $m_c$  that best matches with  $x$ , i.e. that has the index  $c = \arg \min_i (d(x, m_i))$ .

The main task is to define the  $m_i$  in such a way that the mapping is *ordered* and *descriptive* of the distribution of  $x$ . The process in which such mappings are formed is defined by the SOM algorithm. This process is then likely to produce asymptotically converged values for the models  $m_i$ , the collection of which will approximate the distribution of the input samples  $x(t)$ , even in an ordered fashion..

*Note:* It is not necessary for the models  $m_i$  to be vectorial variables. It will suffer if the distance measure  $d(x, m_i)$  is defined over all occurring  $x$  items and a sufficiently large set of models  $m_i$ . For example the  $x(t)$  and  $m_i$  may be vectors, strings of symbols, or even more general items.

## 4.5 The incremental SOM algorithm

Kohonen's original SOM algorithm can be seen to define a special recursive regression process, in which only a subset of the models are processed at every step. The SOM algorithm we are about to describe finds a mapping from the input data space  $\mathbb{R}^n$  onto a two-dimensional array of nodes, called a *lattice*, as depicted in [Figure 4.5].

With every node  $i$ , a parametric *model vector*, also called *reference vector*  $m_i = [w_{i1}, w_{i2}, \dots, w_{in}]^T \in \mathbb{R}^n$  is associated. Before recursive processing, the  $m_i$  must be *initialized*. Random values for the components of  $m_i$  may do, but if the initial values of the  $m_i$  are selected with care, the convergence can be

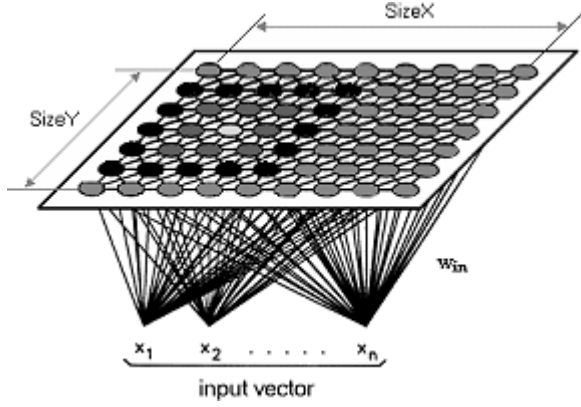


Figure 4.5: A SOM of size  $X \times Y$  in which the activation area of the BMN are depicted.

made to converge much faster. The lattice type of the array can be defined to be *rectangular*, *hexagonal* or even *irregular*. For visual display, hexagonal is effective.

Let  $x = [x_1, x_2, \dots, x_n]^T \in \mathbb{R}^n$  be the input vector that is connected to all neurons in parallel via variable scalar weights  $w_{ij}$  and  $x \in \mathbb{R}^n$  be a stochastic data vector.

The SOM can then be said to be a "nonlinear projection" of the probability density function  $p(x)$  of the high-dimensional input data vector  $x$  onto the two-dimensional array. The central result in self-organization is that if the input patterns have a well-defined probability density function  $p(x)$ , then the weight vectors associated with the models  $m_i$  should try to imitate it [Kohonen, 1990].

#### 4.5.1 Selection of the best matching node ("Winner-takes-all")

Vector  $x$  may be compared with all the  $m_i$  in any metric; in many practical applications, the smallest of the *Euclidian distance*  $\|x - m_i\|$  can be made to define the *best matching node* (BMN), signified by the subscript  $c$ :

$$c = \arg \min_i \|x - m_i\| \text{ which means the same as } \|x - m_c\| = \min_i \|x - m_i\| \quad (4.5)$$

#### 4.5.2 Adaptation (Updating of the weight vectors)

During *learning*, or the process in which the "nonlinear projection" is formed, those nodes that are *topographically close in the array up to a certain geometric distance* will activate each other to learn something from the same input  $x$ , see [Figure 4.2]. This will result in a local *relaxation* or *smoothing* effect on the weight vectors of neurons in this neighborhood, which in continued learning leads to *global ordering*.

This is achieved by defining a *neighborhood set*  $N_c$  around node  $c$  and at each learning step only update those nodes that are within  $N_c$  and keep the rest of the nodes intact. The width or radius defining which nodes are in  $N_c$  (referred to as the "radius of  $N_c$ ") can be time-variable. Experimental results have shown that best global ordering is achieved if  $N_c$  is very wide in the beginning of the learning process and shrinks monotonically with time. Explanations for this is that in the beginning a rough ordering among the  $m_i$  values are made, and after narrowing the  $N_c$ , the spatial resolution of the map is improved.

The following updating process is then used:

$$m_i(t+1) = \begin{cases} m_i + \alpha(t)[x(t) - m_i] & \text{if } i \in N_c(t) \\ m_i & \text{if } i \notin N_c(t) \end{cases} \quad (4.6)$$

where  $t = 0, 1, 2, \dots$  (is the discrete-time coordinate (an integer) and  $\alpha(t)$  is a scalar-valued *learning-rate factor*  $0 < \alpha(t) < 1$  that decreases with time.

However, instead of using the simple neighborhood set where each node in the set is affected equally much we can introduce a scalar "kernel" function  $h_{ci} = h_{ci}(t)$  as a *neighborhood function* defined over the lattice nodes, then the updating process may read as:

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)] \quad (4.7)$$

For convergence it is necessary that  $h_{ci}(t) \rightarrow 0$  when  $t \rightarrow \infty$ . Usually  $h_{ci}(t)$  is a function of the distance between nodes  $c$  and  $i$  in the array, i.e.  $h_{ci}(t) = h(\|r_c - r_i\|, t)$ , where  $r_c$  and  $r_i$  denotes the coordinates of node  $c$  and  $i$  respectively in the array. With increasing  $\|r_c - r_i\|$  we then have that  $h_{ci} \rightarrow 0$ .

Two simple choices for  $h_{ci}(t)$  occur frequently, whereas the simpler of them is the above neighborhood set  $N_c$  defined as a function of time  $N_c = N_c(t)$ , whereby  $h_{ci}(t) = \alpha(t)$  if  $i \in N_c$  and  $h_{ci}(t) = 0$  if  $i \notin N_c$ . Both  $\alpha(t)$  and the radius of  $N_c(t)$  usually decreases monotonically in time.

The other choice is a much smoother neighborhood kernel, known as the *Gaussian function*:

$$h_{ci}(t) = \alpha(t)e^{-\frac{\|r_c - r_i\|^2}{2\sigma^2(t)}} \quad (4.8)$$

where  $\alpha(t)$  is another scalar-valued learning-rate factor and the parameter  $\sigma(t)$  defines the width of the kernel, the latter corresponds to the radius of  $N_c(t)$  above. Both  $\alpha(t)$  and  $\sigma(t)$  are some monotonically decreasing functions of time.

## 4.6 Selection of parameters

The learning process involved in the computation of a feature map is stochastic in nature, which means that the accuracy of the map depends on the number of iterations of the SOM algorithm. We have seen that there are many parameters involved in defining and training a SOM, different values will have different effects on the final SOM. The values of the parameters are usually determined by a process of trial and error. However, the following advices based on our experience and observations made by others can provide a useful guide.

*The Size of SOM* determines the degree of generalization that will be produced by the SOM algorithm - the more nodes, the finer the representation of details, while the fewer nodes, the broader level of generalization. However, the same broad patterns are revealed at each level of generalization. Another issue regarding the size of the SOM, is that more nodes means longer training time. For example, a  $20 \times 20$  SOM takes four times longer to train than a  $10 \times 10$  SOM. *Sammon's mapping* [Sammon Jr., 1969] can be used on the data set to get some hints on the structure of the data set and by that helping in the process of selecting number of nodes in the lattice.

*The learning steps* must be reasonably large, since the learning is a stochastic process. A "rule of thumb" is that for good statistical accuracy, the number of steps must be at least 500 times the number of network nodes.

*The Learning rate parameter*  $\alpha$  should start with a value close to 1 and then during the first 1000 steps it should be decreasing monotonically but kept above 0.1. The exact form of variation of  $\alpha = \alpha(t)$  is not critical, it can be linear, exponential or inversely proportional to  $t$ . For instance,  $\alpha(t) = 0.9(1 - t/1000)$  may be a reasonable choice. It is during the initial phase of the algorithm that the topological ordering of the  $m_i$  occurs. This phase of the learning process is therefore called the *ordering phase*. The remaining (relatively long) iterations of the algorithm are needed for the fine adjustment of the map; this second phase of the learning process is called the *convergence phase*. For good statistical accuracy,  $\alpha(t)$  should be maintained during the convergence phase at a small value (on the order of 0.01 or less) for a fairly long period of time, which is typically thousands of iterations. Neither is it crucial whether the law for  $\alpha(t)$  decreases linearly or exponentially during the convergence phase.

*The Neighborhood function* can be chosen to be the simple neighborhood-set definition of  $h_{ci}(t)$  if the lattice is not very large, e.g. a few hundred nodes. For larger lattices, the Gaussian function may do.

*The size of the Neighborhood* must be chosen wide enough at the start so the map can be ordered globally. If the neighborhood is too small to start with, various kinds of mosaic-like parcellations of the map are seen, between which the ordering direction changes discontinuously. This phenomena can be avoided by starting with a fairly wide neighborhood set  $N_c = N_c(0)$  and letting it shrink with time. The initial radius can even be more than half the diameter of the network. During the first 1000 steps or so,

when the ordering phase takes place, and  $\alpha = \alpha(t)$  is fairly large, the radius of  $N_c$  can shrink linearly to about one unit, during the convergence phase  $N_c$  can still contain the nearest neighbors of node  $c$ .

## 4.7 Incomplete data

The SOM algorithm is very robust, compared to other neural networks. One problem that frequently occurs in practical applications, especially in applying methods of statistics, is caused by missing data. It has been shown that the SOM algorithm is very robust to missing data in [T.Samad and S.A.Harp, 1992], in fact, they claimed that it is better to use incomplete data than to simply discard the dimension from which there are missing data. However, if a *majority* of the data is missing from a dimension, its better to discard that dimension totally from the learning process [Kaski, 1997]. In training of the SOM with incomplete data, the contribution of missing data is assumed to be zero, i.e.  $\|x - m_i\| = 0$ . Note that the reference vector is represented in every dimension and only those dimension that are represented by the input vector causes changes in the reference vector.

## 4.8 Quality measure of the SOM

Although some optimal map always exists for the input data, choosing the right parameters from start is a tricky task. Since different parameters and initializations gives rise to different maps, it is important to know whether the map has properly adapted itself to the training data. This is usually done with an cost function that explicitly defines the optimal solution but since it has been shown that the SOM algorithms not the gradient of any cost function, at least in the general case, other quality measures has to be used. Two commonly used quality measures that can be used to determine the quality of the map and helping in choosing suitable learning parameters and map sizes are the average quantization error and the topographic error.

### 4.8.1 Average quantization error

*Average quantization error* is as a measure of how good the map can fit the input data and the best map is expected to yield the smallest average quantization error between the BMNs  $m_c$  and the input vectors  $x$ . The mean of  $\|x_i - m_c\|$ , defined via inputting the training data once again after learning, is used to calculate the error with the following formula:

$$E_q = \frac{1}{N} \sum_{i=1}^N \|x_i - m_c\| \quad (4.9)$$

where  $N$  is the number of input vectors used to train the map

A SOM with a lower average error is more accurate than a SOM with higher average error.

### 4.8.2 Topographic error

*Topographic error* measures how well the topology is preserved by the map. Unlike the average quantization error, it considers the structure of the map. For each input vector, the distance of the BMN and the second BMN on the map is considered; if the nodes are not neighbors, then the topology is not preserved This error is computed with the following method:

$$E_t = \frac{1}{N} \sum_{k=1}^N N u(x_k) \quad (4.10)$$

where  $N$  is the number of input vectors used to train the map and  $u(x_k)$  is 1 if the first and second BMN of  $x_k$  are not direct neighbors of each other. Otherwise  $u(x_k)$  is 0.

*Note:* In measuring the quality of a SOM consideration must be taking for *both* the average expected error as well as the topographic errors.

## 4.9 Summary of the SOM algorithm

The essence of Kohonen's SOM algorithm is the idea of discarding the Mexican hat lateral interactions for a time depended "topological neighborhood" function as a way of algorithmically emulate the way self-organizing takes place in biological system. The algorithm can be summarized in two steps:

1. Locate the BMN
2. Increase matching at this node and its topological neighbors

The two steps are however usually carried out as follows:

- 1. Initialize network** For each node  $i$ , initialize corresponding reference vector  $m_i$  at random. Set the initial learning rate a close to unity and the radius  $s$  to be at least half the diameter of the lattice.
- 2. Present input** Present the input vector  $x(t)$  at time  $t$  to all nodes in the lattice simultaneously.
- 3. Similarity matching** Find the BMN  $c$  at time  $t$  using the minimum-distance Euclidian criterion:

$$c = \arg \min_i \|x(t) - m_i(t)\|, i = 1, 2, \dots, N \quad (4.11)$$

where  $N$  is the total number of nodes in the lattice.

- 4. Update** Adjust the reference vector  $m_c$  and all reference vectors  $m_i$  within the neighborhood of the winning node  $c$  using:

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)] \text{ where } h_{ci}(t) = \alpha(t)e^{-\frac{\|r_c - r_i\|^2}{2\sigma^2(t)}} \quad (4.12)$$

- 5. Repeat** Decrease learning rate and the width of the neighborhood. Repeat from step 2 by choosing a new input vector  $x(t+1)$  until a predefined stop criteria is reach.

## 4.10 The Dot-Product SOM algorithm

Normalization is not necessary in principle, but sometimes it may be necessary to *normalize the input*  $x$  before it is used in the algorithm. Reasons for this can be to ensure that the resulting reference vectors are kept in the same dynamic range. Another aspect is that it is possible to apply many different metrics in matching, however then the matching and updating laws should be mutually *compatible* with respect to the same metric. For instance, if the dot-product definition of similarity of  $x$  and  $m_i$  is applied, the learning equations should read:

$$x^t(t)m_c(t) = \max_i \{x^T(t)m_i(t)\} \quad (4.13)$$

$$m_i(t+1) = \begin{cases} \frac{m_i(t) + \alpha'(t)x(t)}{\|m_i(t) + \alpha'(t)x(t)\|} & \text{if } i \in N_c(t) \\ m_i(t) & \text{if } i \notin N_c(t) \end{cases}$$

where  $0 < \alpha'(t) < \inf$ , for instance,  $\alpha'(t) = 100/t$ . This process automatically *normalizes the reference vectors* at each step. Normalization usually slows down training process but during matching, the dot-product criterion is very simple and fast. The Euclidian distance and the dot products are the most widely used matching criteria's, although many others have been used.

## 4.11 The batch SOM algorithm

Another type of SOM algorithm is the batch(-map) SOM algorithm [Kohonen, 1993], it is also an iterative algorithm, but instead of using a single data vector at time  $t$ , the whole data set is presented to the map before any adjustments are made – hence the name "batch". The algorithm is outlined below:

**Selection of the BMN** Consider a set of input samples  $X$  and a two-dimensional lattice where with each node  $i$  a reference vector  $m_i$  is associated. The initial values of the  $m_i$  can be selected at random from the domain of input samples. With the use of a distance measure  $d(x, m_i)$ , the node that is most similar to the input  $x \in X$  is found and  $x$  is then copied into a sublist associated with that node.

**Adaptation (Updating of the reference vectors)** After all input samples  $x \in X$  have been distributed among the nodes sublists, the reference vectors can be updated. This is done by finding the "middlemost" samples  $\bar{x}_i$  or  $\bar{x}_i'$  for each node  $i$  and replacing the old value of  $m_i$  with  $\bar{x}_i$  or  $\bar{x}_i'$ .

Start by considering the neighborhood set  $N_i$  around reference vector  $m_i$ . The neighborhood set consists of all nodes within a certain radius from node  $i$ . In the union of all sublists in  $N_i$ , find the "middlemost" sample  $\bar{x}_i$  which have the smallest sum of distances from all the samples  $x(t)$  where  $t \in N_i$ . This "middlemost" sample can be either of two types:

1. Generalized set median: If the sample  $\bar{x}_i$  is part of the input samples  $x \in X$ .
2. Generalized median: If the sample  $\bar{x}_i$  is not part of the input samples  $x \in X$ . Since we are only using a sample of inputs  $x \in X$  it may be possible to find an item  $\bar{x}_i$  that has an even smaller sum of distances from any of the  $x \in N_i$ , this item  $\bar{x}_i'$  is called the generalized median.

The middlemost sample  $\bar{x}_i$  or  $\bar{x}_i'$  that has been found is used to replace the old value of  $m_i$ . This is done for each node  $i$ , by always considering the neighborhood set  $N_i$  around each node  $i$ , i.e. by replacing each old value of  $m_i$  by  $\bar{x}_i$  or  $\bar{x}_i'$  in a simultaneous operation.

The above should then be iterated, i.e. all the samples  $x \in X$  are again distributed into the sublists (which will now most likely change since each  $m_i$  has been updated in the previous iteration) and new medians are computed.

Formally, the above process uses a concept known as *Voronoi tessellation* when forming the neighborhood sets. It is a way of partitioning a set of data vectors into regions, bordered by lines, such that each partition contains a reference vector  $m_i$  that is the "nearest neighbor" to any vector  $x$  within the same partition. These lines together constitute the Voronoi tessellation. All vectors that have a particular reference vector as their closest neighbor are said to constitute a *Voronoi set*  $V$ . That is, the reference vectors  $m_i$  and the search for the BMN defines a tessellation of the input space into a set of Voronoi sets  $V_i$ , defined as:

$$V_i = \{x : \|x - m_i\| < \|x - m_n\| \forall i \neq n\} \text{ where } x \in X \quad (4.14)$$

In each training step, the map units are associated with one such Voronoi set, i.e. the sublist of each reference vector  $m_i$  containing the data vectors  $x$  that have it as BMN constitute a Voronoi set.

If a general neighborhood function  $h_{ci}$  is also used, then the formula for updating the reference vectors becomes:

$$m_i = \frac{\sum_{x \in X} h_{ci}(t)x}{\sum_{x \in X} h_{ci}} \quad (4.15)$$

where  $c$  is the bmn for the current sample  $x$ .

This algorithm is particularly effective if the initial values of the reference vectors are roughly ordered. This algorithm contains no learning rate parameter and therefore it seems to yield more stable asymptotic values for the  $m_i$  compared to the original SOM algorithm [Kohonen, 1993].

## 4.12 The SOM as a clustering and projection algorithm

The SOM is very useful in producing simplified descriptions and summaries of large datasets. It creates a set of topographic ordered reference vectors that represents the data set on a two-dimensional grid. This process can be described in terms of *vector quantization* (VQ) and *vector projection* (VP). The combination of these two methods is called *vector quantization-projection* (VQ-P) methods [Vesanto, 1999], and clearly, the SOM can be categorized as such.

- VQ is used as a way of reducing the original data to a smaller but still representative set of the original data. In the SOM, the reference vectors for the models or nodes in the grid, are representative for a larger set of data vectors in the input space.
- VP is used to project high-dimensional data vectors as points on a low-dimensional, usually 2D, display. In the SOM, this is defined as the location of the data vectors' BMN.

In the SOM process, VQ and VP interacts with each other, in contrast to a serial combination of the two methods, where VQ is performed first (by a clustering algorithm like *K-means*) and then a VP is performed (by a projection method like *Sammons's mapping*).

#### 4.12.1 Vector quantization

Clustering means partitioning of a data set into a set of clusters  $K_i$ ,  $i = 1, 2, \dots, c$ , where each cluster is supposed to contain data that is highly similar. One definition of optimal clustering is a partitioning that minimizes distances within and maximizes distance between clusters where distance is used as a measure of *dissimilarity*, i.e. the magnitude of the distance implies how "unlike" two data items are. The Euclidian distance is a commonly used distance criterion. A survey of different clustering algorithms can be found in [Jain et al., 1999].

Algorithms based on VQ performs an elementary unsupervised clustering and classification of the data vectors in an (input) space [Kohonen, 2001]. Formally the goal of VQ can be stated as; given a set of input data vectors  $x \in \mathbb{R}^n$  find a finite number of reference vectors  $m_i \in \mathbb{R}^n$ ,  $i = 1, 2, \dots, k$  that form a quantized approximation of the distribution of the input data vectors. Once suitable reference vectors have been determined (using a VQ algorithm), VQ (or approximation) of an input vector  $x$  simply means finding the reference vector  $m_c$  closest to the input vector  $x$  (in input space). Finding the closest reference vector is usually done using the Euclidian metric as in  $c = \arg \min_i \|x - m_i\|$  where  $c$  is the index of the closest reference vector  $m_c$ . The optimal formation of reference vectors, when using the Euclidean metric, means finding reference vectors that minimize the average expected square of the *quantization error*, defined as:

$$E = \int \|x - m_c\|^2 p(x) dx \quad (4.16)$$

where  $p(x)$  is a probability density function of  $x$  for the input space and the integration is performed over the entire input space. Since a *closed form*<sup>1</sup> solution for the determination of the  $m_i$  is not available, at least for a general  $p(x)$ , an iterative approximation algorithm is used. See [Kohonen, 2001] for the derivation of such an algorithm and a more rigorous study of VQ. The SOM is therefore very closely related to VQ-based clustering algorithms.

Competitive (or winner-takes-all) neural networks can be used to cluster data. Kohonen has developed two methods for automatic clustering of data, one supervised methods called learning vector quantization and one unsupervised methods called the SOM. Both methods build on the VQ methods originally developed in signal analysis for compressing (signal) data. In learning vector quantization only one or two winning node's reference vectors are updated during each adaptation stage, whereas in the SOM the reference vectors for a neighborhood of nodes in the SOM's lattice are updated simultaneously.

One well known clustering algorithm that performs clustering in a similar way as VQ is the well known *K-means algorithm*. The K-means algorithm is a very popular clustering algorithm, mainly because it is easy to implement and its time complexity is  $O(n)$ , where  $n$  is the number of data vectors. The algorithm is sensitive to the choice of initial reference vectors and may therefore converge to a local minima. The objective of the K-means algorithm is to minimize the distance between the input data vectors and the reference vectors. The algorithm minimizes the sum of squared errors:

$$E = \sum_{i=1}^c \sum_{x \in K_i} \|x - m_i\|^2 \quad (4.17)$$

where  $c$  is the number of clusters and the reference vector  $m_i$  is the mean of all vectors  $x$  in the cluster  $K_i$ .

---

<sup>1</sup>A closed form (expression) is any formula that can be evaluated in a finite number of standard operations.



The K-means algorithm first selects  $K$  data vectors, from the set of data vectors to cluster, to be used as initial reference vectors  $m_i$ . Next  $K$  clusters are formed by assigning each available data vector  $x$  to its closest reference vector  $m_i$ . The mean  $\bar{x}_i$  for each cluster is computed and used as the new value of  $m_i$ . The process is repeated until it is determined that the objective function has reached a minima, i.e. there's a minimal decrease in the sum of squared errors. Note that we have explained the K-means algorithm in SOM terminology, usually the reference vectors is defined in terms of a centroid, which is the mean of a group of points.

The K-means algorithms and the SOM algorithms are very closely related, if the width of the neighborhood function is zero (including only the BMN) then the incremental SOM algorithm acts just as the incremental K-means algorithm. Similarly, the batch version of the K-means algorithm is closely related to the batch SOM algorithm. Although they are very closely related, in K-means the  $K$  reference vectors should be chosen according to the expected numbers of clusters in the data set, whereas in the SOM the number of reference vectors can be chosen to be much larger, irrespective of the number of clusters [Kaski, 1997].

## 4.12.2 Vector projection

The purpose of projection methods is to reduce the dimensionality of high dimensional data vectors. The projection represents data in a low dimensional space, usually two-dimensional, such that certain features of the data is preserved as good as possible [Kohonen, 2001]. Typically, such features are the pair wise distances between data samples or at least their order and subsequently preservation of the shape of the set of data samples. Projection can also be used for the visualization of data if the output space is at most 3 dimensional.

Several projection methods are discussed in [Kaski, 1997], linear projections such as *principal component analysis* (PCA), nonlinear projections such as *multidimensional scaling* (MDS) methods and the closely related *Sammon's mapping*. Kohonen [Kohonen, 2001] also mentions nonlinear PCA and *curvilinear component analysis* (CCA). However, these methods seem to only differ in how the different distances are weighted and how the representations are optimized [Kaski, 1997].

### Linear projection

PCA was first introduced by Pearson in 1901 and further developed by the statistician Hotelling in 1933 and then later by Loève in 1963 and is a widely used projection method. PCA can be used to display the data as a linear projection on a subspace of lower dimension of the original data space that best preserves the variance in the data but since PCA describes the data in terms of a linear subspace, it cannot take into account nonlinear structures, structures consisting of arbitrarily shaped clusters or curved manifolds [Kaski, 1997]. Assuming that the dataset is represented in a proper way as data vectors, PCA is done by the use of matrix-algebra<sup>2</sup>: first the covariance matrix of the input data is calculated and then, secondly, the eigenvectors and eigenvalues are calculated for this matrix. The principal components of the data set is then the eigenvector with the highest eigenvalue. In general, the eigenvectors are ordered by their eigenvalues, from high to low. By leaving out eigenvectors with low eigenvalues, the final dataset will be of lesser dimension than the original one. Of course, some information may be lost, but since the eigenvalues were low, the loss will be minor. It should be noted that there exists a nonlinear version of PCA, called *principal curves*. However, as it turns out, to be used in practical computations, the curves have to be discretized and discretized principal curves are then essentially equivalent to the SOMs [Kohonen, 2001]. Thus, the conception of principal curves is only useful in providing one possible viewpoint to the properties of the SOM algorithm.

### Non-linear projection

In 1969, a projection method called, *Sammon's mapping*, was developed [Sammon Jr., 1969]. Sammon's mapping is closely related to the group of metric based MDS methods in which the main idea is to find a mapping such that the distances between data vectors in the lower dimensional space is as similar as possible to the distance of the corresponding data vectors in the original metric space [Kohonen, 2001].

<sup>2</sup>Maths packages like MATLAB have inbuilt functions for doing such calculations, see the section entitled SOM Toolbox for MATLAB

The basic idea in Sammon's mapping and other projection methods is to represent each high-dimensional data vector as a point in a 2 dimensional space and then arrange these points in such a way that the distances between the points resemble the distances in the original space (defined by some metric) as faithfully as possible. More formally, assume we are given a *distance matrix*  $M$  consisting of pair wise distances between all vectors  $x_i$  and  $x_j$  in the input space  $\mathbb{R}^n$ , where  $n$  is the dimensionality of the input space.  $M$  is then defined by its elements  $d_{ij}$ , where  $d_{ij} = (x_i, x_j)$ . Further, let  $r_i \in \mathbb{R}^2$  be the location or coordinate vector of the data vector  $x_i \in \mathbb{R}^n$ . With  $N$  we denote the distance matrix consisting of the pair wise distances between all the coordinate vectors  $r_i$  and  $r_j$  in the 2 dimensional space  $\mathbb{R}^2$  measured with the Euclidian norm  $\|r_i - r_j\|$ . The goal is then to adjust the  $r_i$  and the  $r_j$  vectors on the  $\mathbb{R}^2$  plane so that the distance matrix  $N$  resembles the distance matrix  $M$  as well as possible, i.e. to minimize a cost function represented as an error function  $E$  by way of a iterative process. The error  $E$  function is defined as (following the notation used in [Kohonen, 2001]):

$$E = \frac{1}{\sum_i \sum_{j>i} d_{ij}} \sum_i \sum_{j>i} \frac{(d_{ij} - \|r_i - r_j\|)^2}{d_{ij}} \quad \text{Sammon's mapping} \quad (4.18)$$

$$E = \frac{1}{\sum_i \sum_{j>i} d_{ij}} \sum_i \sum_{j>i} (d_{ij} - \|r_i - r_j\|)^2 \quad \text{MDS} \quad (4.19)$$

$$E = \frac{1}{\sum_i \sum_{j>i} d_{ij}} \sum_i \sum_{j>i} \frac{(d_{ij} - \|r_i - r_j\|)^2}{e^{\|r_i - r_j\|}} \quad \text{CCA} \quad (4.20)$$

This leads to a configuration of the  $r_i$  points such that the clustering properties of the  $x_i$  data vectors are visually discernible from this configuration. Thus, the inherent structure of the original data can be told from the structure detected in the 2-dimensional visualization. If we compare the above methods with each other, we see that the difference lies in how the distances are weighted. In Sammon's mapping small distances in input space are weighted as more important, in CCA the small distances in output space are weighted as more important and in the case of MDS no weighting is used resulting in larger distances having more impact on the error function. The weighting of distances results in projections that better preserve the *local shape* of the structure of the input space, whereas no weighting as in the MDS case gives a better *global shape* of the structure in the input space.

Kohonen recommends that Sammon's mapping is used for a preliminary analysis of the data set used for SOMs, because it can roughly visualize class distributions, especially the degree of their overlap.

In [Figures 4.6(a), 4.6(b), 4.6(c)] projections based on the same dataset are shown, the dataset is a rating dataset involving 17 movies and 808 users. Comparing [Figure 4.6(a)] with [Figure 4.6(b)] shows that Sammon's mapping spreads the data in a more illustrative way than PCA. In [Figure 4.6(c)] the effect of using a fixed grid size combined with the topographic ordering that is enforced by the SOM algorithm is shown.

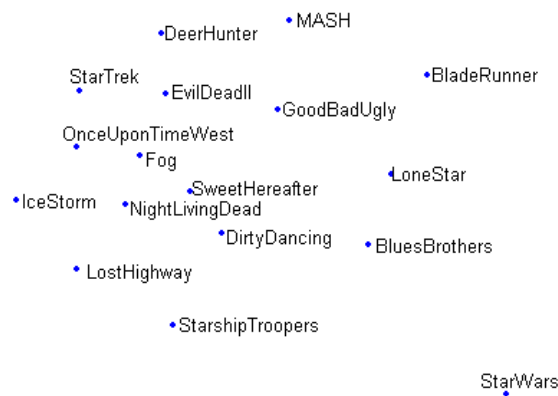
### 4.12.3 Comparison of the SOM to other VQ and VP algorithms

The SOM as discussed by Kohonen in [Kohonen, 2001] combines clustering and visualization operations, however, some essential differences to the methods previously described can be found in [Kohonen, 2001] and [Kaski, 1997]:

1. The SOM represents a open set of multivariate data vectors by a finite set of reference vectors  $m_i$ , like in K-means clustering. This is not the case in the above discussed projection methods where the number of input samples equal the number of outputs.
2. Unlike K-means, the number of reference vectors should not equal the number of expected clusters, instead there numbers of reference vectors should be fairly large. This has to do with the neighborhood function, which makes neighboring nodes on the grid more similar each other, and thus neighboring nodes reflect the properties of the same rather than different clusters. The transition from one cluster to another on the grid is gradual taking place over several nodes. If the goal is to create a few, but quantitative clusters, the SOM has to be clustered. [Vesanto and Alhoniemi, 2000]



(a) Linear projection of the data as points onto the two-dimensional linear subspace obtained with PCA.



(b) Non-linear projection onto a two dimensional space using Sammons's mapping.



(c) Data placed on their BMN in a 10x10 grid using a SOM obtain using the incremental SOM algorithm.

Figure 4.6: Projections using different projection algorithms of the same 808-dimensional dataset consisting of ratings on 17 movies by 808 users.

3. The SOM represents data vectors in an orderly fashion on a (usually) two-dimensional regular grid and the reference vectors are associated with the nodes of the grid. That is, the SOM tries to preserve the topology instead of the distance between data samples, whereas, for example, Sammon's mapping tries to preserve the metric with an emphasis for preserving the local distance between data samples. The grid structure also discretizes the output space, whereas Sammon's mapping has continuous outputs.
4. It is not necessary to recompute the SOM for every new sample, because if the statistics can be assumed stationary, the new sample can be mapped onto the closest old reference vector. In Sammon's mapping, for example, the mapping of all data samples is computed in a simultaneous optimization process, meaning that, with new data samples a new optimal solution exists. Therefore, no explicit mapping function is created since the projection must be recomputed for every new data sample.
5. Sammon's mapping can very easily converge to a local optimal solution, so in practice, the projection must be recomputed several times with different initial configurations. The SOM algorithm is found to be more robust to this if learning is started with a wide neighborhood function that gradually decreases to its final form.

Different methods, like the SOM, K-means and various projection methods, all display properties of the data set in slightly different ways, therefore, a useful approach could be to combine several of them. In the paper [Flexer, 1997], K-means and Sammon's projection is combined into a serial VQ-P method and compared against the SOM. They report that their combined method outperforms the SOM as a clustering and projection tool. However, it seems like their performance measure favors distance preserving algorithms and the SOM is not intended for distance preserving, instead the SOM orders the reference vectors on a lattice structure in such a way that local neighborhood sets in the projection are preserved. In a study done by [Venna and Kaski, 2001] a new concept is defined: the *trustworthiness* of the projection, meaning that if two data samples are close to each other in output space, then they are more likely to be near each other in the input space. The SOM is compared to other methods like the PCA, non-metric MDS and Sammon's mapping and are found to be especially good at this. It is also shown that the SOM is good at preserving the original neighborhood set in the projection. Although PCA and non-metric MDS is found to be better on this. For discrete data, neighborhoods of data vectors is defined as sets consisting of the  $k$  closest data vectors, for relatively small  $k$ .

## 4.13 Visualizing the SOM

Different kinds of methods for visualizing data using the SOM and other VQ-P methods are discussed in the paper [Vesanto, 1999]. Three goals of visualization are defined; to get an idea of the *structure* and the *shape* of the data set, for example, if there are any clusters and outliers. Secondly, *analysis of the reference vectors* in a greater detail, and third, *examining new data* with the map, i.e. the response from the map has to be quantified and visualized.

As for the first goal, while projections give a rough idea of clusters in the data, to actually visualize the clusters on the SOM, techniques based on distance matrices are commonly used. The distance matrix can contain all distances between the nodes' reference vectors in the lattice as well as their immediate neighbors, like in the Unified distance matrix (U-matrix), as illustrated in [Figure 4.7(a)] or just a single value for each node, representing either the minimum, maximum or average distance for the nodes to its neighbors, as illustrated in [Figure 4.7(b)]. A related technique is to use colors, i.e. same color is assigned to nodes that are similar, as illustrated in [Figure 4.7(c)].

The U-matrix can be seen as several component planes stacked on top of each other. Each component plane shows the values of a single vector component in all nodes of the map, hence, together they can be seen as constituting the whole U-matrix. Component planes are commonly used in the analysis of reference vectors since they can give information such as the spread of the values of a component or they can also be used in correlation hunting, i.e. finding component planes with similar patterns [Figure 4.8].

In the examination of new data, the response from the map can be as simple as just pointing out the BMN for that data. Histograms can be used for multiple vectors. Often some form of accuracy is also measured to give some information whether the sample is close to the mapped BMN or very far.

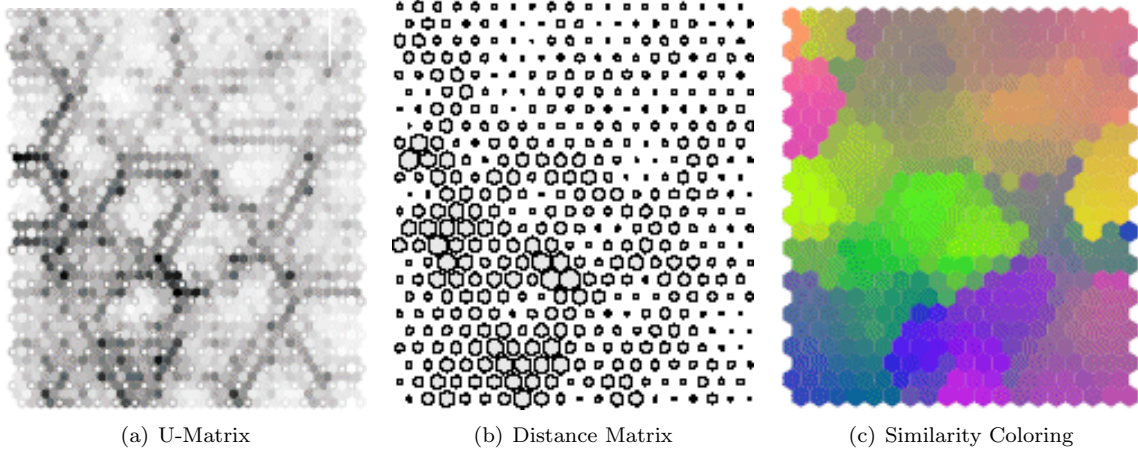


Figure 4.7: The U-matrix (a) is a graphic display frequently used to illustrate the clustering of the reference vectors in the SOM, it represents the map as a regular grid of nodes where the distance between adjacent node's reference vectors is calculated and presented with a coloring scheme. A light coloring between nodes signifies that the reference vectors are close to each other in the input space and A dark coloring between the nodes corresponds to large distances and thus a gap between the reference vectors in the input space. Dark areas can be thought of as clusters separators and light areas as clusters. This is helpful when one tries to find clusters in the input data without having information if there is any. (b) In the Distance Matrix each map node is proportional to the average distance to its neighbors. (c) In the Similarity Coloring map nodes that are similar have the same color.



Figure 4.8: Component planes, useful for correlation hunting.

## 4.14 Properties of the SOM

Once the SOM algorithm has converged, the SOM computed by the algorithm displays important statistical characteristics of the input data. The SOM can be seen as a nonlinear transformation  $\phi$  that maps the spatially *continuous input space*  $X$  onto the *spatially discrete output space*  $L$ , as depicted in [Figure 4.9] [Haykin, 1994]. This transformation, written as  $\phi : X \rightarrow L$  can be viewed as an abstraction of [Equation 4.5], finding of the BMN. The topology of the input and output space are defined respectively by the metric relationship of vectors  $x \in X$  and the arrangement of a set of nodes of a regular two-dimensional grid. If we consider this in a neurobiological context, the input space  $X$  may represent the coordinate set of somatosensory receptors distributed densely over the entire body surface, then output space  $L$  will represent the set of neurons located in the layer of the cortex which the somatosensory receptors are confined.

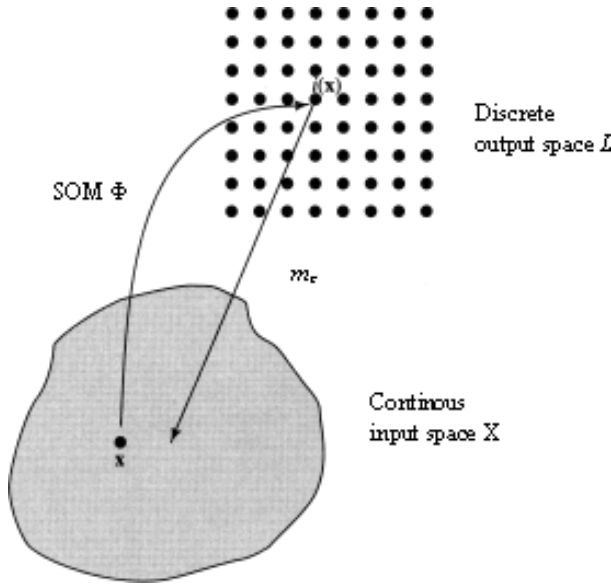


Figure 4.9: [Haykin, 1994] Illustration of a relationship between the SOM  $\phi$  and a reference vector  $m_c$  of a BMN  $c$ . Given an input vector  $x$ , the SOM  $\phi$  identifies a BMN  $c$  in the output space  $L$ . The reference vector  $m_c$  for node  $c$  can then be viewed as a pointer for that node into the input space  $X$ .

#### 4.14.1 Approximation of the input space

The SOM  $\phi$ , represented by the set of reference vectors  $\{m_i | i = 1, 2, \dots, N\}$  in the output space  $L$ , provides an *approximation* to the input space  $X$ .

This property comes from the fact that the SOM performs VQ, as discussed earlier, where the key idea is to use a relatively small number of reference vectors to approximate the original vectors in input space.

#### 4.14.2 Topological ordering

The SOM  $\phi$  computed by the SOM algorithm is topologically locally and globally ordered in the sense that nodes that are close to each other on the grid represents similar data in the input space.

That is, when data samples are mapped to those nodes on the lattice that have the closest reference vectors, nearby nodes will have similar data samples mapped onto them.

This property is a direct consequence of the updating process [Equation 4.7]. It forces the reference vector  $m_i$  of the BMN  $c$  to move toward the input vector  $x$ , and when used with a wide neighborhood kernel in the beginning of the learning process that slowly decreases during training, a global ordering is guaranteed (since it has the effect of moving the reference vectors  $m_i$  of the closest nodes(s)  $i$  along with the BMN  $c$ ).

#### Illustration of the ordering principle implemented by the SOM

The difference between a locally *and* globally ordered SOM  $\phi$  on one hand and a SOM  $\phi$  that is *only* locally ordered can be illustrated if we think of the SOM  $\phi$  as an elastic net that is placed in input space. (Such an illustration however requires a special case, such as when both the input and output space is two dimensional. Keep in mind that the input space is usually of a very high dimension and that ordering is not well defined in higher dimensions, it is however useful to imagine something similar occurring in higher dimensional cases.)

For a two-dimensional input space with the point distribution as shown in [Figure 4.10], four distinct clusters can easily be detected by visual inspection, these are denoted in the figure by  $c_1, c_2, c_3$  and  $c_4$ .

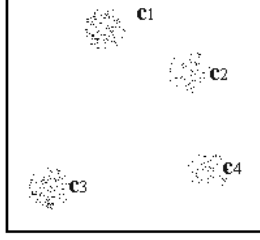


Figure 4.10: Two-dimensional input space consisting of four distinct clusters denoted  $c_1, c_2, c_3$  and  $c_4$ .

The output space consists of a regular two-dimensional grid structure of four nodes, where the corresponding reference vector corresponds to some coordinates in input space  $X$  [Figure 4.11].

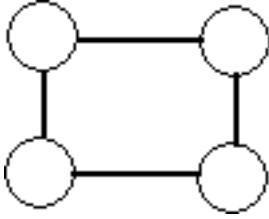


Figure 4.11: Two-dimensional grid structure with four nodes (output space).

The goal of the SOM algorithm is to find suitable coordinates for the reference vectors such that the collection of grid nodes follows (or approximates) the distribution of the data points in input space in such a way that neighboring nodes are more alike than nodes farther apart. Two nodes are said to be similar if they represent similar data points in input space. If this is true, then a grid is said to be globally and locally ordered and is referred to as a topologically ordered SOM  $\phi$ . In order to illustrate this property of the SOM  $\phi$ , we will look at a SOM  $\phi$  that is locally ordered only and one that is both locally and globally ordered.

In the first case, the SOM has been trained with a neighborhood size of zero, i.e. only the BMN is updated for every data point, resulting in the grid depicted in [Figure 4.12(a)]. In the second case, the SOM has been trained with a neighborhood function that monotonically decreasing over time, i.e. more nodes are affected by the same data points in the beginning of the training than in the end of the training of the SOM. The resulting grid is depicted in [Figure 4.12(b)]. Each node is labelled with the cluster in input space that it represents.

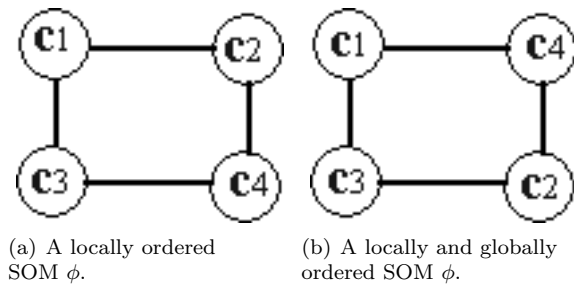


Figure 4.12: SOM with nodes labeled with the cluster in input space they represent.

As can be seen in [Figure 4.12(a)] and [Figure 4.12(b)], the nodes represent different clusters and the effect of this can be detected if the labeled nodes in the figures are placed in input space while keeping their connections between each other. This "stretch" effect of the SOM  $\phi$  illustrates clearly the ordering property of the SOM  $\phi$ , nearby nodes in the grid are also nearby in input space [Figure 4.13(b)]. This is not the case with an SOM  $\phi$  that is only locally ordered [Figure 4.13(a)].

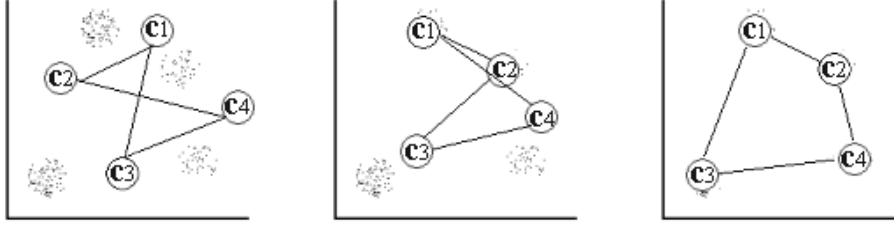
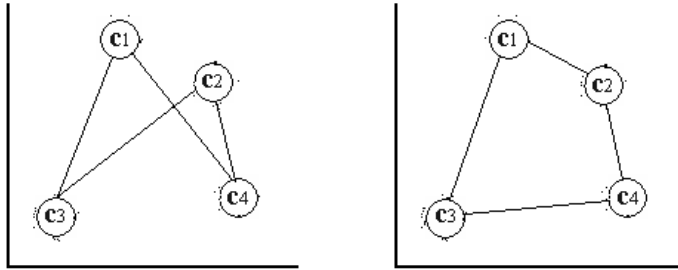


Figure 4.14: A nonlinear regression process of the ordered set of reference vectors. (a) shows the initial state of the nodes in input space. (b) shows how the nodes have moved after a few iteration in their attempts to following the data distribution and at the same arranging the nodes so they have the same ordering as in the grid. (c) shows the converged SOM  $\phi$ , i.e. the nodes are placed in areas with high point density and the location of the nodes respective each other corresponds to the initial ordering of the grid nodes.



(a) A locally ordered SOM  $\phi$  stretched out in input space. (b) A locally and globally ordered SOM  $\phi$  stretched out in input space.

Figure 4.13: SOM arrays stretched out in input space.

To achieve both locally and global order, the neighborhood must be allowed to be wide in the beginning so a rough global ordering can take place among the reference vectors and then by letting it shrink with time, a local structure takes form. So, with topological order is meant a grid structure that is ordered from the beginning (its nodes) and by arranging the associated reference vectors to the nodes so that they *also* become globally and locally ordered in the grid, then we have the SOM  $\phi$ .

The formation of the SOM  $\phi$  can also be thought of as an nonlinear regression of the ordered set of reference vectors into the input space, i.e. the reference vectors in [Figure 4.12(a)] forms a elastic grid with the topology of a two-dimensional grid whose nodes have weights as coordinates in the input space and that follows the distribution of the data samples in input space [Figure 4.14].

The overall aim of the algorithm may thus be stated as follows:

*Approximate the input space  $X$  by reference vectors  $m_i$ , in such a way that the SOM  $\phi$  provides a faithful representation of important features that characterize the input vectors  $x \in X$ .*

#### 4.14.3 Density matching

The SOM  $\phi$  reflects variations in the statistics of the input distribution: the density of reference vectors of an ordered map will reflect the density of the data samples in input space. In areas with high point density, the reference vectors will be close to each other, and in the empty space between areas with high point density, they will be more sparse.

As a general rule, the SOM  $\phi$  computed by the SOM algorithm tends to over represent regions of low input density and under representing regions of high input density. One way of improving the density-matching property of the SOM algorithm is to add heuristics to the algorithm, which force the distribution computed by the algorithm to match the input distribution.



The density match is also different for nodes at the border of the grid compared to center nodes at the grid, since the neighborhood to the border nodes are not symmetric. This effect is usually referred to as the *border effect* and different ways of eliminate these are given in [Kohonen, 2001]. For example, if the batch-map is used, border effects can be eliminated *totally* if after a couple of batch-map iterations the neighborhood set  $N_i$  is replaced by  $\{i\}$ , i.e. having a couple of simple K-means iterations at the end.

## 4.15 Theoretical aspects of the SOM algorithm

A few words has to be mentioned about the theoretical aspect of the SOM algorithm. How can we be sure that the SOM algorithm converges into a ordered stable state? Frankly, we don't, at least in the multi-dimensional case. It is just a well observed phenomena that the SOM algorithm leads to an organized representation of the input space, even if started from an initial complete disorder. Lets quote Kohonen himself regarding this matter:

*"Perhaps the SOM algorithm belongs to the class of "ill posed" problems, but so are many important problems in mathematics. In practice people have applied many methods long before any mathematical theory existed for them and even if none existed at all. Think about walking: theoretically we know that we could not walk at all unless there existed gravity and friction,... People and animals, however, have always walked without knowing this theory."*

It is actually a bit surprisingly that the SOM algorithm has shown such resistant against a complete mathematical study if one consider that the algorithm itself is easy to write down and simulate and that the practical properties of the SOM are clear and easy to observe. And still, no proof for its theoretically properties exists in the general case. These questions are treated by the mathematical expert Marie Cottrell [Cottrell et al., 1998] in a review paper regarding different proofs of convergence for the SOM algorithm under special circumstances, we quote:

*"Even if the properties of the SOM algorithm can be easily reproduced by simulations, and despite all the efforts, the Kohonen algorithm is surprisingly resistant to a complete mathematical study. As far as we know, the only case where a complete analysis has been achieved is the one-dimensional case (the input space has dimension 1) for a linear network (the units are disposed along a one-dimensional array)."*

The study of the SOM algorithm in the one dimensional case is nearly complete, all that's left is to find a convenient decreasing rate to ensure the ordering[Cottrell et al., 1998]. The first proof of both convergence and ordering properties in the one dimensional case was presented by Cottrell and Forth in 1987.

Two obstacles that prevents a thorough study of the convergence of the SOM algorithm in higher dimensions are [Kohonen, 2001]:

1. How to define *ordering* in higher dimensions? In the one-dimensional case, ordering of the SOM can easy be defined, but for higher dimensions, no accepted mathematical definition for the ordered states of multi dimensional SOMs seems to exists.
2. How to define a *objective function* for the SOM in the high-dimensional case? Only in the special case, when the data set is discrete and the neighborhood kernel is fixed, can a potential function be found in the multi-dimensional case. Ritter proved the convergence in this special case with the use of a gradient-descent method. He then later applied this method to the well-known traveling salesman problem, where the probability density function of input is discrete-valued.

Details and references on actual results in the effort of reaching a solid mathematical theory about the SOM algorithm can be found in [Cottrell et al., 1998], as well as in [Kohonen, 2001].

## 4.16 SOM Toolbox for Matlab

SOM Toolbox<sup>3</sup> [Vesanto et al., 2000] is an implementation of the SOM and its visualization in MATLAB<sup>4</sup> (MATrix LABoratory). The Toolbox can be used to initialize, train and visualize the SOM. It also provides methods for pre-processing of data, analysis of data and the properties of the SOM. Two SOM algorithms are implemented, the original SOM algorithm and the batch-map. Although we have implemented our own SOM algorithm, the SOM Toolbox provides functions which we have not implemented, such as PCA and Sammon's mapping for analysis of data and some visualization methods.

## 4.17 SOM based applications

The SOM algorithm was developed in the first place for the *visualization of nonlinear relations of multidimensional data*. However, the properties of the SOM is an interesting tool for two distinct classes of applications:

- Simulators used for the purpose of understanding and modeling of computational maps in the brain.
- Practical applications such as *exploratory data analysis, pattern recognition, speech analysis, robotics, industrial and medical diagnostics, etc.*

A complete guide to the extensive SOM literature and an exhaustive listing of detailed application can be found in [Kohonen, 2001]. The first application area of the SOM was speech recognition (speech-to-text transformation). This application was called a 'neural' phonetic typewriter and was developed by Kohonen himself [Kohonen, 1988]. Another widely known application with Kohonen involved are WEBSOM<sup>5</sup> [Kohonen et al., 2000], which is used for content based organization of large document collections and document retrieval. The user interface of the WEBSOM application will be discussed in part III.

## 4.18 Applications of the SOM in recommender systems

This section contains short descriptions of work related in one way or another to ours, in the context of using the SOM in recommender systems.

**CF using SOM and ART2 networks** In [Graef and Schaefer, 2001] they describe two model based approaches to collaborative filtering using SOMs and ART2 networks. They investigate how well the two models performs regarding *response time, quality of predictions* and *adaptively*. Their approach to use the SOM algorithm is to cluster users into similar nodes on the grid based on their similarities in rating movies. For this, the EachMovie<sup>6</sup> dataset that contains 2811983 ratings on a scale from 1 to 5 for 1628 movies by 72916 users was used. From this, they randomly selected 2000 users with a minimum of 80 ratings. For their experiments they used a SOM of the size 10x10 nodes. To generate a recommendation for a user, his user-profile is compared to each reference vector and the closest reference vectors is chosen for the recommendation process. This reference vector is then compared with the user and recommendations are made for unrated items in the user profile, i.e. the predicted value is the reference vectors value for the particular movie. To measure the quality of the predictions, 30 ratings was randomly excluded from each user and then one, three, five and 30 recommendations was computed and compared to the excluded ratings for each user. Their test results shows that the SOM made correct recommendations in 45% of the

---

<sup>3</sup>Can be downloaded at <http://www.cis.hut.fi/projects/somtoolbox/>

<sup>4</sup>MATLAB is a widely used programming environment for technical computing. The matrix is the main data structure used in MATLAB, so it is a very efficient tool for performing matrix calculations. MATLAB is developed by MathWorks Inc. and can be found at <http://www.mathworks.com/>

<sup>5</sup>WEBSOM can be found at <http://websom.hut.fi/websom/>

<sup>6</sup>Digital Equipment Corporation's Systems Research Center (SRC) originally developed EachMovie in order to gain experience with a collaborative filtering algorithm. EachMovie had been operated as a free public service for 18 months between the years 1995 and 1997. In September 1997 the EachMovie service was terminated. The dataset was then stripped of identifying data and made available so that collaborative filtering researchers could use the data to test their algorithms. The EachMovie Dataset remained available until October 2004 when it was finally retired.

cases for one movie and was correct in 30% of the cases when it had to make 30 recommendations. Compared to their implementation of a collaborative filtering algorithm and the ART2 network, the SOM performed worse. Response time is measured in the cases when the total number of users is 400,1200 and 2000. Not surprisingly, the response time increased with the number of users for the memory based collaborative filtering algorithm, but for both the SOM and the ART2 network, no significant difference can be noted. This has to do with the fact, that the SOM only has to do 100 comparisons, no matter how many users there are, as in contrast to the memory based collaborative filtering algorithm that has to make as many comparisons as there are users. Adaptively means the SOMs ability to adapt to new users without having to re-learn the whole SOM. The SOM clearly outperformed the ART2 network on this, no distinct difference could be detected between the number of correct recommendations made by an updated SOM and a re-learned SOM.

**Hybrid recommender system combining CF with SOM** A hybrid recommender system that combines CF with the SOM is presented in [Lee et al., 2002]. Their approach is to use the SOM algorithm as a cluster algorithm in which the clusters are formed according to the users demographic data (sex and age) and preferences for different movie genres as action, drama, comedy, fiction and thriller. the idea is then to apply collaborative filtering on the cluster to which the active user belongs. They claim that their solution is comparable with the collaborative filtering algorithm introduced by GroupLens [Resnick et al., 1994] based on the computed mean absolute error. However the data set they uses consists of only 174 users and 53 movies and no characteristics is presented for this unknown data set. Also, coverage is not reported at all.

**CBR-CF with SOM as a clustering algorithm** In [Roh et al., 2003] a cluster-indexing case based reasoning (CBR) model for collaborative filtering recommendations are described. Their approach is to use the SOM algorithm as a clustering algorithm and then use the reference vectors as indexes for where similar users (or cases in CBR terminology) can be found in the case-base reasoning process. Clusters are formed according to the users ratings on movies, the well-known MovieLens data set is used, however, they only uses a fraction of the data set, a dense rating matrix with 251 users and 33 movies are used for evaluation. PCA has been used for a pre-step analyzing of the data set and from their experiments, the size of the grid is set to four nodes (clusters). Prediction for the active user is done by finding the active users closest reference vector and then applying collaborative filtering with the users indexed by the same reference vector. Their studies show comparable results compared to collaborative filtering techniques based on the reported normalized mean absolute error.

**Music artist reviews SOM** A content-based approach is taken in [Vembu and Baumann, 2004], very much similar to earlier studies done by the authors of this thesis [Gabrielsson and Gabrielsson, 2004]. Their idea is to use music artist reviews taken from Amazon web site and extract words from the reviews to build artist-profiles. The usual techniques found in text-retrieval is used to form these artist-profiles; vector space model and tf-idf weighting schema. They also hand-picked 324 words that they thought to be of extra importance that was given extra weight in the artist-profiles. The dimension of the artist-profile was reduced to 3313 from original 36708 by excluding words that was present in less than 5% and more than 90% of the artist reviews.

The SOM algorithm was then used to cluster artist-profiles represented as bag-of-words. They implemented a SOM of size 7x7 nodes using the SOM Toolbox for 398 artists. Evaluation was made by comparing their SOM against a web-based music recommendation engine. Each BMN contains artists similar to each other based on artist reviews and the web-based engine recommends artists similar to a specific artist, but from a much larger data set and with a completely different approach. The top 10 list generated by the web-engine for a specific artist was compared to the BMN for the same artist. Observations was made that if they included the artist top 3 or 5 BMN, the overlap between the top 10 list and the artists neighbourhood increased. Which of course is by the virtue of the map's self-organizing property, neighbouring nodes represents similar data.

## Chapter 5

# Implementation and evaluation of SOM based recommendation techniques

### 5.1 SOM implementation

We implemented a basic SOM based on the (Original) Incremental SOM Algorithm as described by Kohonen in [Kohonen, 2001] which has been rigorously introduced and described in the previous chapters.

We make use of a set of  $N$  sparse data vectors, all vectors have the same dimensionality but will not have values for all dimensions. During training the dimensions in which a data vector misses values will be ignored, in the distance calculation carried out when finding the BMN for a data vector missing values are assumed to contribute 0 to the distance and when updating the reference vector only the dimensions in which the data vector has values are updated.

Alternative methods for handling missing values were tested, such as using default values for missing values during the distance and/or updating. Positive results could be obtained with default values, however we are still unsure about the validity of using default values. It seems that sparsity of data and insertion of default values will result in e.g. the distance calculations being dominated by the default values, hence we chose to not use default values for our implementation in the end.

The SOM algorithm will form a projection of high dimensional data manifolds onto a regular array. The elements of the array are as previously mentioned referred to as *nodes*. The regular array has a rectangular lattice structure (nodes are arranged in rectangular patterns) and a sheet shape (nodes on the edges of the array do not "wrap around" to the "other side"). The nodes of the array are referred to using their index  $(i, j)$  where  $(0, 0)$  is always the top left node and  $(rows - 1, columns - 1)$  the bottom rightmost node. A Euclidean distance measure is used to determine distances between pairs of nodes in the array. We will henceforth refer more generally to the regular array as a "map". This terminology is appropriate both as the visualization of data on the grid looks like a map, and the SOM itself is a mapping defined by the regular array and the reference vectors associated with the elements of the array.

With each node a *reference* vector is associated and of the same dimensionality as the vectors in  $V$ . The number of rows and columns in a map  $M$  is accessed using the notation  $M.rows$  and  $M.columns$  respectively. The reference vector for a node in the map is retrieved using the index of the node, e.g.  $M_{0,0}$  specifies the reference vector for the top-left node in the map. A element  $i \in M$  is an index tuple, e.g.  $i = (0, 0)$  for the top left node, which can also be used to specify a reference vector in the map using the notation  $M_i$ , in which case a specific dimension of the reference vector can be specified using the notation  $M_i(d)$ .

We initialize the reference vectors for each node in the array by picking random samples from the training vectors. While this is not an optimal initialization method, it seems to work well enough for us.

The training of a SOM requires specifying the parameters listed in [Table 5.1] the parameters include the size of the map and the settings for the rough order and fine tuning phase of training the SOM.

### Map

<i>rows</i>	Number of rows in the array.
<i>columns</i>	Number of columns in the array.

### Rough ordering phase

<i>Rough<sub>T</sub></i>	Number of training iterations.
<i>Rough<sub>α0</sub></i>	Initial learning rate for training.
<i>Rough<sub>σ0</sub></i>	Initial neighborhood radius for training.
<i>Rough<sub>σT</sub></i>	Final neighborhood radius for training.

### Fine tuning phase

<i>Fine<sub>T</sub></i>	Number of training iterations.
<i>Fine<sub>α0</sub></i>	Initial learning rate for training.
<i>Fine<sub>σ0</sub></i>	Initial neighborhood radius for training.
<i>Fine<sub>σT</sub></i>	Final neighborhood radius for training.

Table 5.1: Parameters required for the training of a SOM using our implementation of the SOM algorithm. Assume that  $V$  is a set containing all vectors.

```

som( $V$ , rows, columns)
1   $M \rightarrow \text{create-map}(\text{rows}, \text{columns})$ 
2  sample-initialization( $M$ ,  $V$ , rows, columns)
3  som-algorithm( $M$ ,  $V$ , RoughT, Roughα0, Roughσ0, RoughσT)
4  som-algorithm( $M$ ,  $V$ , FineT, Fineα0, Fineσ0, FineσT)
5  Return  $M$ 

```

The function *create-map* on line 1 tells us that a map structure as previously described is created and consists of the specified number of *rows* and *columns*.

```

sample-initialization( $M$ ,  $V$ , rows, columns)
1  For row  $\leftarrow 0$  To rows  $- 1$ 
2      For column  $\leftarrow 0$  To columns  $- 1$ 
3           $M_{\text{row}, \text{column}} \leftarrow \text{random-vector}(V)$ 

```

```

random-vector( $V$ )
1  Return  $V(\text{rand}(1, |V|))$ 

```

The *sample-initialization* method simply picks a random vector from the set  $V$  repeatedly using the *random-vector* method as shown. The random selection means that some data vectors might be used for multiple reference vectors as initialization, while this might seem problematic, i.e. how will a BMN be determined if multiple reference vectors are equally "close" to a sample during training, this is however not a problem as we will simply pick the first found BMN as closest and adjust it, no winner takes all situation will (or is likely to) arise as we also adjust the neighboring node's reference vectors, in the end the adjustments will even out.

```

som-algorithm( $M, V, T, \alpha_0, \sigma_0, \sigma_T$ )
1  For  $t \leftarrow 0$  To  $T - 1$ 
2       $x \leftarrow \text{random-vector}(V)$ 
3       $c \leftarrow \text{find-bmn}(M, x)$ 
4       $\alpha \leftarrow \text{inverse-learning-rate}(\alpha_0, T, t)$ 
5       $\sigma \leftarrow \text{linear-neighborhood-radius}(\sigma_0, \sigma_T, T, t)$ 
6      Foreach  $i \in M$ 
7           $\text{activation} \leftarrow \text{gaussian-neighborhood}(M, c, i, \alpha, \sigma)$ 
8          If  $\text{activation} \neq 0$  Then
9              Foreach dimension  $d$  that sample  $x$  has a value for
10                  $M_i(d) \leftarrow M_i(d) + \text{activation} \times (x(d) - M_i(d))$ 

```

We chose to implement a stochastic SOM algorithm. Thus one random data (sample) vector at the time will be selected from the training data and shown to the SOM whereby the currently best matching node (BMN) for the sample is directly found, and the reference vector for the BMN, and the reference vectors for nodes in its neighborhood, are directly adjusted to become more similar to the sample. Since this is an incremental algorithm, *time* is equivalent to the *current iteration number*, starting from 0.

```

find-bmn( $M, x$ )
1   $\text{bm}n \leftarrow \text{unknown}$ 
2   $d \leftarrow \text{inf}$ 
3  For  $\text{row} \leftarrow 0$  To  $\text{rows} - 1$ 
4      For  $\text{column} \leftarrow 0$  To  $\text{columns} - 1$ 
5          If  $\text{euclidean-distance}(x, M(\text{row}, \text{column})) < d$  Then
6               $\text{bm}n \leftarrow (\text{row}, \text{column})$ 
7               $d \leftarrow \text{euclidean-distance}(\text{sample}, M(\text{row}, \text{column}))$ 
8  Return  $\text{bm}n$ 

```

The BMN for a sample data vector is found by computing the Euclidean distance between the sample and each node's reference vector, the node which's reference vector is nearest the sample is chosen as BMN.

```

euclidean-distance( $x, m$ )
1   $d \leftarrow 0$ 
2  Foreach dimension  $d$  that sample  $x$  has a value for
3       $d \leftarrow (x(d) - m(d))^2$ 
4  Return  $d$ 

```

The euclidean distance calculation as already mentioned and motivated ignores dimensions in which the sample  $x$  has no values.

```

inverse-learning-rate( $\alpha_0, T, t$ )
1   $b \leftarrow T/(100 - 1)$ 
2   $a \leftarrow b \times \alpha_0$ 
3  Return  $a/(b + t)$ 

```

We used an *inverse of time* learning rate function which decreases inversely with time towards zero (but does not become zero, which would cause a division by zero error, it approaches roughly a hundredth of the initial learning rate).

*linear-neighborhood-radius*( $T, \sigma_0, \sigma_T, t$ )

1   **Return**  $T + t \times (\sigma_0 - \sigma_T) / (T - 1)$

We used a linear neighborhood radius which decreases linearly from the initial neighborhood radius down to 1.

*gaussian-neighborhood*( $M, c, i, \alpha, \sigma$ )

1    $d \leftarrow \text{node-distance}(c, i)$

2   **Return**  $\alpha \times e^{\frac{d^2}{2 \times \sigma^2}}$

We use a gaussian function to implement the neighborhood function, i.e. a *gaussian neighborhood function*. Neighboring nodes  $i$  within the radius  $\sigma$  at time  $t$  of the BMN  $c$  have a higher activation than nodes outside the radius  $\sigma$ . The learning rate  $\alpha$  decides how much can at most be learned at any time  $t$ , i.e. the maximum activation.

*node-distance*( $c, i$ )

1   **Return**  $\sqrt{(c.\text{row} - i.\text{row})^2 + (c.\text{column} - i.\text{column})^2}$

The distance between the BMN  $c$  and the neighboring node  $i$  is calculated using the *node-distance* function, which uses the row and column indices of the nodes to determine the distance between the node's index vectors on the regular array. Please note that this does not at all involve using the node's reference vectors, it only depends on the structure of the map.)

## 5.2 SOM based recommendation techniques

As in Part I recommendation techniques will be described using simple pseudo code that outlines how available profile models are used to generate recommendations.

For all the described techniques the common assumption is made that all the rating data uses the same rating scale  $[R_{min}, R_{max}]$ . It is always assumed that there is a set  $U = \{u_1, u_2, \dots\}$  and  $I = \{i_1, i_2, \dots\}$  consisting of all users and items respectively for which necessary profile models are available. The profile models previously presented in Part I will again be used.

The techniques that generate predictions outline how to predict ratings on all available items for all available users on items the users have not rated, this gives a better idea of the complexity of the algorithm than simply showing how to predict for the active user on the active item. The techniques do not outline nor discuss how to introduce new users or items into the system.

The techniques in addition to assuming the presence of required profile models also assume that necessary constants have been specified.

### 5.2.1 SOM User based Collaborative Filtering (SOMUCF)

Recommendation technique based on the CF approach. Makes use of the neighborhood preservation of data in the SOM to retrieve similar users for the active user. The active user's neighborhood is simply all other users that have the same BMN as the active user, and users having neighboring nodes as BMN. The constant NR (Neighborhood Radius) specifies the radius within which neighboring nodes of the BMN can be found in the SOM's map. Otherwise the technique is the same as the UCF technique.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a item rating model  $i_r$ . Let  $M$  represent a trained SOM that has been trained on a set of item rating models.

```

somucf( $U, I, M$ )
1  Foreach  $u \leftarrow U$ 
2     $bm_n \leftarrow \text{find-bm}_n(M, u_r)$ 
3     $N \leftarrow \text{neighborhood}(bm_n, M, U, NR)$ 
4     $P_u \leftarrow \emptyset$ 
5    Foreach  $i \in I$ 
6      If  $r_{ui} \notin u_r$  Then
7         $p \leftarrow \text{predict}(u, i, N)$ 
8         $\text{next}(P_u) \leftarrow (i, p)$ 
9  Return  $P$ 

```

```

neighborhood( $c, M, U, NR$ )
1   $NS \leftarrow \emptyset$ 
2  Foreach  $i \in M$ 
3    If  $\text{node-distance}(c, i) \leq NR$  Then
4       $\text{next}(NS) \leftarrow i$ 
5   $N \leftarrow \emptyset$ 
6  Foreach  $u \in U$ 
7     $i \leftarrow \text{find-bm}_n(M, u_r)$ 
8    If  $i \in NS$  Then
9       $d \leftarrow \text{node-distance}(c, i)$ 
10      $\text{next}(N) \leftarrow (u, 1/d)$ 
11 Return  $N$ 

```

The function *neighborhood*(*node*, *M*, *U*, *NR*) determines which of the nodes in *M* lie within the neighborhood radius *NR* of the active user's BMN and returns the set of users in the neighborhood as well as their similarity with the active user.

In order to determine which nodes lie within the neighborhood radius *NR* of the BMN we use the *node-distance* function previously described. An alternative would have been to consider the distance between the node's model vectors. We did attempt this, however we didn't seem to get any better results doing this, probably because as our u-matrices indicate we don't have any clear clustering structure motivating its usage.

Finally the active user's neighbors are considered to be all the users  $u \in U$  in the BMN and its neighboring nodes. Neighbors having the same BMN as the active user are considered to be perfectly similar to the active user. Neighbors having a neighboring node as BMN are assigned a similarity based on their BMN's distance to the active user's BMN, we chose to implement this as a simple  $1/d$  formula, where  $d$  is the distance between the active user's BMN and the neighbor's BMN. Note that the shown neighborhood code is not optimal, only descriptive.

### 5.2.2 SOM Item based Collaborative Filtering (SOMICF)

The recommendation technique is based on the CF approach if the item SOM is based on rating data, however if it is based on attribute data it is based on the CBF approach. The technique is the same in both cases however, the difference lies in how the SOM is created which is beyond the scope of the technique.

Makes use of the neighborhood preservation of data in the SOM to retrieve similar items for the active item. The active item's neighborhood is simply all other items that have the same BMN as the active item, and items having neighboring nodes as BMN. The constant *NR* (Neighborhood Radius) specifies the radius within which neighboring nodes of the BMN can be found in the SOM's map. Otherwise the technique is the same as the ICF technique.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a item rating model  $i_r$ . Let *M* represent a trained SOM that has been trained on a set of user rating models.



```

somicf( $U, I, M$ )
1  Foreach  $i \in I$ 
2     $bm_n \leftarrow \text{find-bm}_n(M, i_r)$ 
3     $N \leftarrow \text{neighborhood}(bm_n, M, I, NR)$ 
4    Foreach  $u \in U$ 
5       $P_u \leftarrow \emptyset$ 
6      Foreach  $i \in I$ 
7        If  $r_{ui} \notin u_r$  Then
8           $p \leftarrow \text{predict}(u, i, N)$ 
9           $\text{next}(P_u) \leftarrow (i, p)$ 
10 Return  $P$ 

```

### 5.2.3 Item SOM User based Collaborative Filtering (ISOMUCF)

The recommendation technique is based on the CF approach if the item SOM is based on rating data, however if the item SOM is based on attribute data it is based on a hybrid filtering approach. The technique is the same in both cases however, the difference lies in how the SOM is created.

User similarities are calculated only over items similar to the active item. Prediction algorithm too is only provided with user rating models containing ratings on items similar to the active item, this means that for example user mean ratings will only be over items similar to the active item. It is conceivable that for certain types of movies the rating behavior of users varies, hence making it motivated to attempt to capture this when predicting. Makes use of the neighborhood preservation of data in the SOM to retrieve similar items for the active item. The active item's neighborhood is simply all other items that have the same BMN as the active item, and items having neighboring nodes as BMN. The constant  $NR$  (Neighborhood Radius) specifies the radius within which neighboring nodes of the BMN can be found in the SOM's map.

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a item rating model  $i_r$ . Let  $M$  represent a trained SOM that has been trained on a set of item rating models.

```

isomucf( $U, I, M$ )
1   $RM \leftarrow \emptyset$ 
2  Foreach  $node \in M$ 
3     $N \leftarrow \text{neighborhood}(bm_n, M, I, NR)$ 
4    Foreach  $(i, s) \in N$ 
5      Foreach  $r_{ui} \in i_r$ 
6         $\text{next}(RM_{node, u}) \leftarrow r_{ui}$ 
7  Foreach  $u \in U$ 
8     $P_u \leftarrow \emptyset$ 
9    Foreach  $i \in I$ 
10     If  $r_{ui} \notin u_r$  Then
11        $BMN \leftarrow \text{find-bm}_n(M, i_r)$ 
12        $N \leftarrow \emptyset$ 
13       Foreach  $n \in U$ 
14          $\text{next}(N) \leftarrow \text{similarity}(RM_{BMN, u}, RM_{BMN, n})$ 
15        $\text{sort}(N)$  in descending order of similarity
16        $p \leftarrow \text{predict}(u, i, N)$ 
17        $\text{next}(P_u) \leftarrow (i, p)$ 
18 Return  $P$ 

```

Line 6 creates the sets of ratings denoted  $RM_{node, u}$ , which are user rating sets consisting only of ratings on items within the neighborhood of the node in question. Thus during prediction we can given an

active item's BMN easily retrieve rating models for users consisting only of ratings on items similar to the active item.

An alternative implementation of this technique is to completely skip the model building step on lines 1-6, which is time consuming and memory intensive. Instead having selected an active user and an active item, the active user's rating model is reduced to only contain ratings on items within the active item's neighborhood. This means that instead item neighborhoods should be modeled, which is less memory requiring. Neighbors models are not modified, instead similarity is computed directly with the active user's reduced rating model, which is in most cases such as with Pearson similarity exactly the same thing as had the neighbors model also been reduced. However, when predicting e.g. using Weighted Deviation From Mean, a difference will arise because we are still working with the neighbors complete rating models, their mean will not be over items similar to the active item but over all their items. We implemented and evaluated both these methods, however results appeared to be almost similar for both approaches, since the technique as it is presented here has a more theoretical sound motivation we choose to only present its results.

#### 5.2.4 Model Predicting SOM User based Collaborative Filtering (MPSO-MUCF)

Recommendation technique based on the CF approach. The value specified for each item by the model vector of the BMN for the active user is used as predictions for all items for the active user.

Assume that each user  $u \in U$  has a user rating model  $u_r$ . Let  $M$  represent a trained SOM that has been trained on a set of user rating models.

```

 $mpsomucf(U, I, M)$ 
1  Foreach  $u \in U$ 
2       $bm_n \leftarrow find\_bm_n(M, u_r)$ 
3       $P_u \leftarrow \emptyset$ 
4      Foreach  $i \in I$ 
5          If  $r_{ui} \notin u_r$  Then
6               $p \leftarrow M_{bm_n}(i)$ 
7               $next(P_u) \leftarrow (i, p)$ 
8  Return  $P$ 

```

#### 5.2.5 SOM Goodness Collaborative Filtering (SOMGOODNESSCF)

Recommendation technique combines the SOMUCF and SOMICF technique and uses the average of the two techniques predictions as a goodness prediction. Note that if the item SOM used by the SOMICF technique is based on attributes and not ratings, then this combined technique will be based on the  $HF$  approach. The SOMUCF based predictions are considered to have a possible element of serendipity, while the SOMICF based predictions to be more safe. By taking into account these two predictions a final goodness prediction can be made, the simplest choice of taking the average of the two prediction was made in this technique. Depending on whether

Assume that each user  $u \in U$  has a user rating model  $u_r$ , and that each item  $i \in I$  has a item rating model  $i_r$ . Let  $MU$  represent a trained SOM that has been trained on a set of user rating models, and let  $MI$  represent a trained SOM that has been trained on a set of item rating models.

```

somgoodnesscf( $U, I, MU, MI$ )
1  Foreach  $u \in U$ 
2     $P_u \leftarrow \emptyset$ 
3    Foreach  $i \in I$ 
4      If  $r_{ui} \notin u_r$  Then
5         $p_{safe} \leftarrow$  SOMUCF based prediction on  $i$  for  $u$ 
6         $p_{serendipity} \leftarrow$  SOMICF based prediction on  $i$  for  $u$ 
7        If  $p_{serendipity} \neq \text{Failed}$  AND  $p_{safe} = \text{Failed}$  Then
8           $p \leftarrow p_{serendipity}$ 
9        Elseif  $p_{serendipity} = \text{Failed}$  AND  $p_{safe} \neq \text{Failed}$  Then
10          $p \leftarrow p_{safe}$ 
11        Elseif  $p_{serendipity} \neq \text{Failed}$  AND  $p_{safe} \neq \text{Failed}$  Then
12          $p \leftarrow (p_{safe} + p_{serendipity})/2$ 
13        Else
14          $p \leftarrow \text{Failed}$ 
15         $next(P_u) \leftarrow (i, p)$ 
16  Return  $P$ 

```

### 5.2.6 Top-N SOM User based Collaborative Filtering (TOPNSOMUCF)

Recommendation technique makes predictions for users using the SOMUCF recommendation technique and creates Top-N ranking lists by sorting the predictions for each user in descending order of prediction and recommending the Top-N items with highest predictions.

### 5.2.7 Top-N SOM Item based Collaborative Filtering (TOPNSOMICF)

Recommendation technique makes predictions for users using the SOMICF recommendation technique and creates Top-N ranking lists by sorting the predictions for each user in descending order of prediction and recommending the Top-N items with highest predictions.

### 5.2.8 Top-N SOM Goodness Collaborative Filtering (TOPNSOMGOODNESSCF)

Recommendation technique makes predictions for users using the SOMGOODNESSCF/CBF recommendation technique and creates Top-N ranking lists by sorting the predictions for each user in descending order of prediction and recommending the Top-N items with highest predictions.

## 5.3 Evaluation data

The rating and attribute data sets presented previously in Part II were used again, this time split over users into a 80% training data set and a 20% test data set. The training data set was used to create a SOM, the user models in the training data set and the item models in the training set were respectively used to create a user SOM and a item SOM. Thus for i.e. the DS dataset the same 80% training set was used to create a item SOM as well as a user SOM, the idea being to use evaluation datasets created similarly to those used in Part I. However we must emphasize that we chose for simplicity (on behalf of reliability of validation results, which we however still consider representative of what can be obtained with the recommendation algorithms in question) to make simple Hold Out evaluations instead of K-Fold Cross Validations.

This time we decided to use the IMDb keyword lists, which provides for each movie a set of descriptive IMDb user specified keywords. The keyword lists were preprocessed to only contain keywords used for at least 5 different movies and at most for 50 different movies, it was also required that each movie had

	Map		Rough ordering				Fine tuning				SOM Evaluation		
<i>SOM</i>	<i>Rows</i>	<i>Columns</i>	<i>T</i>	$\alpha$	$\sigma_0$	$\sigma_T$	<i>T</i>	$\alpha$	$\sigma_0$	$\sigma_T$	<i>Runtime</i>	<i>AQE</i>	<i>TE</i>
IMDBK	50	50	1000	0.9	20	3	100000	0.2	3	1	16min	1.653	0.470
DS2-ITEM	20	20	1000	0.9	20	3	200000	0.2	3	1	13min	3.803	0.043
DS2-USER	20	20	1000	0.9	20	3	200000	0.2	3	1	13min	3.981	0.054
DS-ITEM	20	20	1000	0.9	20	3	200000	0.2	3	1	5min	2.033	0.307
DS-USER	20	20	1000	0.9	20	3	200000	0.2	3	1	16min	4.556	0.047
CML-ITEM	20	20	1000	0.9	20	3	200000	0.2	3	1	16min	3.903	0.075
CML-USER	20	20	1000	0.9	20	3	200000	0.2	3	1	21min	5.792	0.002

Table 5.2: SOMs created based on 80% training datasets, except IMDBK which was created on a preprocessed set of IMDb movie keywords. E.g. DS2-ITEM refers to a 80% training dataset based on the DS2 rating dataset split over users which was used to create an item SOM, i.e. the item rating models in the training dataset were used when training the SOM. The Average Quantization Error (AQE) and the Topographic Error (TE) is also shown for each SOM based on the respective training data. The errors are overall quite low indicating a possibly good SOM.

at least 5 keywords. See [Table 5.2] for further information about the SOM’s generated based on the evaluation datasets.

The DS2 dataset when used to create an item SOM has the advantage of containing few items that have been rated by only a few users, the DS dataset on the other hand contains some items that have been rated by very few users and because of this we’re likely to see some kind of neighborhood formation based only on the fact the item’s share the fact that few users have rated them.

The IMDBK dataset contains only 609 of the items in the CML dataset, the SOM is also very large, which must be taken into account when interpreting the evaluation results (especially the coverage).

## 5.4 Evaluation results

The recommendation techniques were evaluated for the evaluation datasets using various combinations of evaluation protocol and evaluation metrics.

For the recommendation techniques we select where necessary an appropriate combination of similarity and prediction algorithm, we use a decent set of settings giving good results, not necessarily the absolute best results, but near enough to be interesting to list.

We will describe briefly for each recommendation technique the setup of the evaluation, such that the evaluation can easily be understood and reproduced.

### 5.4.1 Accuracy of predictions

The result tables use the following abbreviations for the column headings: ET = Evaluation time (in minutes),  $U_s = U_{successful}$ ,  $U_f = U_{failed}$ ,  $P_s = P_{successful}$ ,  $P_f = P_{failed}$ , Cov = Coverage, Corr = Correctness.

For the settings name the name of the dataset from which prediction evaluation data was generated is always displayed first, i.e. DS, DS2 or CML. Usually the SOM that is needed by recommendation technique being evaluated is based on the same evaluation dataset. However the settings named CML-IMDBK use the IMDb keyword SOM, the prediction evaluation is still based on a evaluation dataset based on the CML dataset. Each table of evaluation results is followed by description of algorithms and settings used for each evaluation.

Where applicable the value of the neighborhood radius NR is indicated in the settings name, e.g. DS-NR3 means a neighborhood radius of 3 (NR=3) was used for that setting.

In all cases a Hold Out evaluation was performed using rating data split over users such that 80% of each users ratings are used for training data and 20% of each users ratings used as test data. Note that the same Hold Out evaluation data was used for each evaluation based on the same rating dataset, similarly the same SOM as previously described was used each time.

### SOMUCF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS-NR0	0	1578	423	9510	15683	37.7%	42.4%	0.769	0.743	0.775	0.749	0.192	1.062	1.030
DS-NR3	0	1997	4	22679	2514	90.0%	44.9%	0.715	0.679	0.739	0.705	0.179	0.897	0.947
DS2-NR0	0	1606	395	8996	12275	42.3%	43.2%	0.771	0.741	0.802	0.776	0.193	1.106	1.051
DS2-NR3	0	1991	10	19949	1322	93.8%	45.0%	0.715	0.678	0.755	0.721	0.179	0.899	0.948
CML-NR0	0	781	162	6860	12711	35.1%	35.7%	0.917	0.895	0.957	0.932	0.229	1.433	1.197
CML-NR3	0	943	0	18323	1248	93.6%	39.1%	0.813	0.780	0.839	0.810	0.203	1.099	1.04

Table 5.3: Evaluation results for the SOMUCF technique. The Weighted Deviation From Mean prediction algorithm (with  $NS_{min} = 0$ ,  $NS_{max} = 0$ ) was used.

### SOMICF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS-NR0	0	1815	186	14829	10364	58.9%	46.2%	0.704	0.685	0.706	0.695	0.176	0.999	0.999
DS-NR4	0	2001	0	24327	866	96.6%	46.8%	0.689	0.648	0.709	0.667	0.172	0.851	0.923
DS2-NR0	0	1293	708	6464	14807	30.4%	46.6%	0.694	0.683	0.668	0.663	0.173	1.037	1.018
DS2-NR4	0	1996	5	20654	617	97.1%	47.1%	0.695	0.656	0.728	0.697	0.174	0.894	0.945
CML-NR0	0	807	136	8850	10721	45.2%	38.9%	0.840	0.835	0.865	0.866	0.210	1.334	1.155
CML-NR4	0	943	0	19283	288	98.5%	41.9%	0.753	0.716	0.791	0.757	0.188	0.948	0.974
CML-IMDBK	1	544	399	2583	16988	13.2%	34.5%	0.954	0.950	0.978	0.979	0.239	1.671	1.292
CML-IMDBK-NR4	1	876	67	10230	9341	52.3%	34.7%	0.912	0.889	0.980	0.964	0.228	1.400	1.183
CML-IMDBK-NR8	1	924	19	12128	7443	62.0%	35.7%	0.862	0.834	0.927	0.906	0.216	1.209	1.100

Table 5.4: Evaluation results for the SOMICF technique. The Weighted Sum prediction algorithm (with  $NS_{min} = 0$ ,  $NS_{max} = 0$ ) was used.

Settings	NR	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS-NR4	4	0	2001	0	24327	866	96.6%	46.0%	0.696	0.657	0.715	0.679	0.174	0.859	0.927
DS2-NR4	4	0	1996	5	20654	617	97.1%	47.0%	0.693	0.657	0.726	0.694	0.173	0.886	0.941
CML-NR4	4	0	943	0	19283	288	98.5%	42.3%	0.748	0.709	0.785	0.749	0.187	0.934	0.966
CML-IMDBK-NR4	4	1	876	67	10230	9341	52.3%	34.7%	0.903	0.885	0.972	0.961	0.226	1.376	1.173
CML-IMDBK-NR8	8	1	924	19	12128	7443	62.0%	36.1%	0.855	0.825	0.915	0.884	0.214	1.188	1.090

Table 5.5: Evaluation results for the SOMICF technique. The Average Rating prediction algorithm (with  $NS_{min} = 0$ ,  $NS_{max} = 0$ ) was used. All provided neighbors are used as predictors, as such the Neighborhood Radius  $NR$  is the parameter that decides how many items similar to the active that a prediction can be based on.

### ISOMUCF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS-NR0	2	975	1026	5540	19653	22.0%	43.9%	0.710	0.677	0.697	0.670	0.177	0.853	0.924
DS-NR5	8	1984	17	22978	2215	91.2%	47.8%	0.673	0.633	0.700	0.662	0.168	0.810	0.900
DS2-NR0	0	347	1654	1012	20259	4.8%	45.0%	0.729	0.682	0.671	0.607	0.182	0.901	0.949
DS2-NR5	5	1934	67	18996	2275	89.3%	47.6%	0.677	0.635	0.715	0.681	0.169	0.822	0.907
CML-NR0	0	530	413	2985	16586	15.3%	37.4%	0.834	0.814	0.836	0.823	0.208	1.145	1.070
CML-NR5	2	943	0	18902	669	96.6%	43.1%	0.728	0.690	0.756	0.724	0.182	0.880	0.938
CML-IMDBK	1	276	667	690	18881	3.5%	36.7%	0.916	0.868	0.914	0.861	0.229	1.335	1.156
CML-IMDBK-NR5	2	772	171	8860	10711	45.3%	39.1%	0.792	0.767	0.841	0.818	0.198	1.039	1.019
CML-IMDBK-NR8	67	875	68	10889	8682	55.6%	41.1%	0.766	0.732	0.822	0.791	0.192	0.978	0.989

Table 5.6: Evaluation results for the ISOMUCF technique. The Pearson similarity algorithm (with  $OT = 2$ ,  $ST = 30$ ) and the Weighted Deviation From Mean prediction algorithm (with  $NS_{min} = 1$ ,  $NS_{max} = 30$ ) was used.

### MPSOMUCF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS	0	2001	0	24946	247	99.0%	41.8%	0.816	0.777	0.847	0.811	0.204	1.206	1.098
DS2	0	2001	0	21271	0	100.0%	43.9%	0.758	0.718	0.803	0.765	0.190	1.029	1.014
CML	0	943	0	19545	26	99.9%	38.3%	0.814	0.784	0.837	0.806	0.203	1.085	1.041

Table 5.7: Evaluation results for the MPSOMUCF technique.

While the MAE is not always that much worse than the other MAEs obtained using the SOM based recommendation techniques it is consistently the highest, however the prediction time is basically instantaneous, and as long as no new items are introduced coverage is near perfect.

## SOMGOODNESSCF Technique

Settings	ET	$U_s$	$U_f$	$P_s$	$P_f$	Cov	Corr	MAE	$MAE_R$	$MAE_{UA}$	$MAE_{RUA}$	NMAE	MSE	RMSE
DS	0	2001	0	24809	384	98.5%	46.8%	0.683	0.640	0.702	0.662	0.171	0.814	0.902
DS2	0	1998	3	21175	96	99.5%	47.1%	0.679	0.639	0.713	0.676	0.170	0.812	0.901
ML	0	943	0	19492	79	99.6%	42.2%	0.750	0.709	0.779	0.739	0.187	0.925	0.962
CML-IMDBK	1	943	0	19197	374	98.1%	38.9%	0.802	0.771	0.823	0.794	0.200	1.050	1.02

Table 5.8: Evaluation results for the SOMGOODNESSCF technique. For the predictions based on the SOMUCF technique; a neighborhood radius of 3 ( $NR=3$ ) is used in all cases; the Weighted Deviation From Mean prediction algorithm (with  $NS_{min} = 0$ ,  $NS_{max} = 0$ ) is used. For the predictions based on the SOMICF technique; a neighborhood radius of 4 ( $NR = 4$ ) is used for the DS, DS2 and CML settings while a neighborhood radius of 5 ( $NR = 5$ ) is used for the CML-IMDBK setting; the Weighted Sum prediction algorithm (with  $NS_{min} = 0$ ,  $NS_{max} = 0$ ) is used.

### 5.4.2 Relevance of Top-N ranking lists

The result tables use the following abbreviations for the column headings: ET = Evaluation time (min),  $U_s = U_{successful}$ ,  $U_f = U_{failed}$ ,  $TN_s = TN_{successful}$ ,  $TN_f = TN_{failed}$ ,  $TN_a$ , Cov = Coverage, R = Recall, P = Precision, U = Utility,  $R_{AU} = Recall_{AU}$ ,  $P_{AU} = Precision_{AU}$ ,  $U_{AU} = Utility_{AU}$ .

In all cases, unless otherwise noted, a 20% Hold Out set was used that contained items assumed to all be relevant for a user (ratings ignored as explained before in Part I), the ability of the technique to return those items in the Top-N list was evaluated. Note that the same 20% Hold Out datasets were used for each evaluation. Keep in mind that

Where applicable the value of the neighborhood radius NR is indicated in the settings name, e.g. DS-NR3 means a neighborhood radius of 3 ( $NR = 3$ ) was used for that setting.

In all cases a top  $N = 10$  list was generated for each user.

## TOPNSOMUCF Technique

Settings	ET	$U_s$	$U_f$	$TN_s$	$TN_f$	$TN_a$	Cov	R	P	F1	U	AFHP	$R_{AU}$	$P_{AU}$	$F1_{AU}$	$U_{AU}$
DS-NR3	23	2001	0	15	1986	10	0.7%	0.1%	0.1%	0.001	0.001	6	0.1%	0.1%	0.001	0.001
DS2-NR3	6	2001	0	42	1959	10	2.1%	0.2%	0.2%	0.002	0.003	5	0.2%	0.2%	0.002	0.003
CML-NR3	0	943	0	79	864	10	8.4%	0.5%	1.0%	0.007	0.008	5	0.7%	1.0%	0.006	0.008

Table 5.9: Evaluation results for the TOPNSOMUCF technique. The Weighted Deviation From Mean prediction algorithm (with  $NS_{min} = 0$ ,  $NS_{max} = 0$ ) was used.

## TOPNSOMICF Technique

Settings	ET	$U_s$	$U_f$	$TN_s$	$TN_f$	$TN_a$	Cov	R	P	F1	U	AFHP	$R_{AU}$	$P_{AU}$	$F1_{AU}$	$U_{AU}$
DS-NR4	50	2001	0	8	1993	10	0.4%	0.0%	0.0%	0.000	0.000	6	0.0%	0.0%	0.000	0.000
DS2-NR4	3	2001	0	102	1899	10	5.1%	0.6%	0.6%	0.006	0.006	5	0.5%	0.6%	0.004	0.006
CML-NR4	0	943	0	156	787	10	16.5%	1.0%	2.1%	0.014	0.016	5	1.0%	2.1%	0.011	0.016
CML-IMDBK-NR4	1	943	0	216	727	10	22.9%	1.4%	2.9%	0.019	0.023	5	2.1%	2.9%	0.019	0.023

Table 5.10: Evaluation results for the TOPNSOMICF technique. The Weighted Sum prediction algorithm (with  $NS_{min} = 0$ ,  $NS_{max} = 0$ ) was used.

Settings	ET	$U_s$	$U_f$	$TN_s$	$TN_f$	$TN_a$	Cov	R	P	F1	U	AFHP	$R_{AU}$	$P_{AU}$	$F1_{AU}$	$U_{AU}$
DS-NR4	50	2001	0	8	1993	10	0.4%	0.0%	0.0%	0.000	0.000	7	0.0%	0.0%	0.000	0.000
DS2-NR4	3	2001	0	83	1918	10	4.1%	0.4%	0.5%	0.005	0.005	5	0.4%	0.5%	0.003	0.004
CML-NR4	0	943	0	154	789	10	16.3%	1.0%	2.1%	0.014	0.017	5	1.0%	2.1%	0.011	0.016
CML-IMDBK-NR4	1	943	0	212	731	10	22.5%	1.4%	2.9%	0.019	0.022	5	2.1%	2.9%	0.020	0.023

Table 5.11: Evaluation results for the TOPNSOMICF technique. The Average Rating prediction algorithm (with  $NS_{min} = 0$ ,  $NS_{max} = 0$ ) was used.

### TOPNSOMGOODNESSCF Technique

Settings	ET	$U_s$	$U_f$	$TN_s$	$TN_f$	$TN_a$	Cov	R	P	F1	U	AFHP	$R_{AU}$	$P_{AU}$	$F1_{AU}$	$U_{AU}$
DS	71	2001	0	5	1996	10	0.2%	0.0%	0.0%	0.000	0.000	5	0.0%	0.0%	0.000	0.000
DS2	9	2001	0	48	1953	10	2.4%	0.2%	0.3%	0.003	0.002	6	0.2%	0.3%	0.002	0.002
CML	1	943	0	45	898	10	4.8%	0.3%	0.6%	0.004	0.004	6	0.3%	0.6%	0.003	0.004
CML-IMDBK	2	943	0	25	918	10	2.7%	0.1%	0.3%	0.002	0.002	6	0.2%	0.3%	0.002	0.002

Table 5.12: Evaluation results for the TOPNSOMGOODNESSCF technique. For the predictions based on the SOMUCF technique; a neighborhood radius of 3 ( $NR = 3$ ) is used in all cases; the Weighted Deviation From Mean prediction algorithm (with  $NS_{min} = 0$ ,  $NS_{max} = 0$ ) is used. For the predictions based on the SOMICF technique; a neighborhood radius of 4 ( $NR=4$ ) is used for the DS, DS2 and CML settings while a neighborhood radius of 5 ( $NR=5$ ) is used for the CML-IMDBK setting; the Weighted Sum prediction algorithm (with  $NS_{min} = 0$ ,  $NS_{max} = 0$ ) was used used.

## 5.5 Conclusions

The SOM based techniques do not perform better [Table 5.13] than the recommendation techniques presented in Part I, however the results do not discourage their use. The ISOMUCF technique and the SOMGOODNESSCF technique performs closest to the recommendation algorithm instances of the classic UCF technique as it was evaluated in Part I (i.e. using Pearson for similarity and Weighted Deviation From Mean for prediction). Notable is that we achieve almost 100% coverage by only using 8% of all nodes. For the ranking list evaluations, similar or even worse results as those in Part I are obtained.

Recommendation techniques can be judged by many different qualities, MAE prediction accuracy is just one quality. Other qualities include their theoretical soundness (meaning something that seems intuitively correct, such as not only looking at how many movies two users have rated in common, but

	<i>Common</i>			<i>SOM based</i>				
DATASET	BASELINESF	UCF	ICF	SOMUCF	SOMICF	ISOMUCF	MPSOMUCF	SOMGOODNESSCF
DS	0.687	0.679	0.727	0.715	0.689	0.673	0.758	0.679
DS2	0.667	0.657	0.707	0.715	0.693	0.677	0.816	0.683
CML	0.754	0.729	0.744	0.813	0.748	0.728	0.814	0.750

Table 5.13: Comparison between the best MAE obtained using a select set of common recommendation techniques (described in detail in Part I) and a select set of SOM based recommendation techniques. While the lowest MAEs have been highlighted, the variations on the third decimal are not to be considered significant, moreso as a Hold Out evaluation was made for the SOM based techniques and a K Fold Cross validation for the Common techniques. All techniques in the table have the same degree of coverage and correctness.



also determining the types of movies they have in common). In Part I the TRUSTUCF recommendation technique is used, which is a more sound recommendation technique than UCF as it tries to determine if a similar user has lots of experience of the type of movie he is to predict for. Another quality to judge a recommendation technique by is to which degree the technique implements transparency. Which quality to emphasize when selecting a technique, depends on the situation it will be used in. Does the situation motivate the additional complexity? In Part III we will use the concept of transparency when we visualize our recommendations.



## Part III

# The recommendation interface problem

*“Seeing, contrary to popular wisdom, isn’t believing.  
It’s where belief stops, because it isn’t needed any more.”*

– Terry Pratchett

The first two chapters in this part deals with the recommendation interface problem in terms of *trust* and *transparency*, and how one way of solving these issues is with well *explained* recommendations. *Interaction design* for recommender systems are briefly discussed and the concept of *visualization of recommendations* are described. Previous studies done in the research community regarding these areas are discussed and presented. Next, follows a chapter presenting the MOV SOM, a highly interactive visual movie recommender system. We will describe the user interface of MOV SOM and its system architecture. Challenges concerning our approach will also be discussed. No empirical evaluation of the interface is made, in the sense of user interviews or tests, instead we highly encourage any reader of this thesis to visit MOV SOM’s web-site<sup>1</sup> and judge for himself. In the end we conclude and present future work within the area of visual recommendations.

---

<sup>1</sup><http://www.movsom.com/>

## Chapter 6

# Trust in recommender systems

The understanding and trusting of recommender systems as well as the design of the user interface has emerged during the last years as one of the main focuses on recommender systems. Previous research has been focused mainly on the algorithms behind the recommender systems and in particular their accuracy as prediction algorithms. Studies [Herlocker et al., 2000], [Sinha and Swearingen, 2002] have shown that the role of transparency in recommender systems improves the users understanding and acceptance of the recommendations. Explanations provide transparency by exposing the reasoning and the data behind a recommendation. The overall effect of transparency is that a user gains trust for the recommender system, which is an important concept in any recommender system. In MOV SOM transparency is achieved on many levels, text and graphical explanations are made for each prediction as well as visualization of user's neighborhoods and movies similar to each other. Another important aspect of the interface is how the user interacts with it. For example, is it designed for the purpose of encouraging users to explore and develop their tastes or for guiding people directly to items they would be interested in buying immediately? What overall features lead to satisfaction with recommendations? Studies regarding these questions and many more have been done by [Swearingen and Sinha, 2002]. MOV SOM supports different user needs, visualization of recommendations in a map like landscape is supposed to encourage the user to further explore new movies and well motivated predictions is there to help the user to choose the right movie according to his taste.

Whenever there exists some form of uncertainty that makes something risky, trust usually becomes an issue. In recommender systems this can happen when recommended items are unfamiliar to the user or when the user can't figure out why he receives the kind of recommendations that he gets. Although the highest risk the users take in a movie recommender system is to spend a couple of dollars on a bad movie, there are domains in which the risk factor would have a crucial impact on the users decision in trusting a recommendation, e.g. recommender systems in health or financial domains. This is also one of the reasons to why recommender systems have trouble making its way into such high-risk content domains [Herlocker et al., 2000]. A user gains trust in a recommender system over time if he has a positive experience of the recommended items. If the experience is negative, uncertainty towards the systems ability of making good recommendations will start to grow.

### 6.1 Transparency of recommendations

One method for maintaining trust and decreasing uncertainty is to implement *transparency* into the system in such a way that the user knows *how* the system comes up with the recommendations and *why* he should trust the recommendations given by the system.

Users like to know why a particular item was recommended to them, because that makes them feel more confident about the recommendations. This was also the main result of a user study conducted in the paper [Sinha and Swearingen, 2002]. They also reported that users find this important even for items they already like, indicating that users are also looking for a *justification* of the reasoning behind the recommendations. How well a user understand the systems reasoning behind a recommendation depends on to what degree the system has implemented transparency, i.e. understanding of the system logic. If a user clearly can see the link between the type of input they give to the system and the corresponding

output, i.e. the recommendations, the system has high transparency [Sinha and Swearingen, 2002]. This also means that a user is able to better refine his recommendations by revising his input to the system, since he can see which parameters that are effecting the recommendations. There are many ways for the system to implement transparency, in [Herlocker et al., 2000] they suggest that *explanations* (e.g. "since you rated x high, we recommend y"), *predicted ratings* (e.g. "based on how you have rated x, we think you would rate y with four stars out of five") and *recommending a few familiar items* (e.g. items or closely related items to which the user have had positive experience with) all are effective ways of increasing the systems transparency from a user perspective. Predictions alone can be considered as a high-risk feature [Swearingen and Sinha, 2001], since a system needs to have a very high degree of accuracy for the user to benefit from it, hence predictions should always be made in the context of explanations to cover for imprecise accuracy. MOV SOM always explain the predictions and if no prediction can be given, explanation for that is given as well. Including familiar items among the recommended items seems to also have a trust-generating effect [Swearingen and Sinha, 2001], as long as they are not directly related to the kind of information that was fed to the system by the user as input to be used in the recommendation process. The previous experience of these items and the impression that the systems knows you personally can be one explanation for this effect. In MOV SOM, recommendations are tagged as familiar, neutral and serendipitous, allowing the user to judge for himself what kind of recommendation he is in the mood for.

## 6.2 Explanation of recommendations

If a user can figure out why he have received a particular recommendation, he can better judge if the recommendation is based on accurate information or not. One paper that deals with this question is [Herlocker et al., 2000], in which different ways of explaining recommendations with the help of graphics and commentaries are investigated. A user study was conducted in which the users were presented with twentyone different explanation interfaces. Each explanation interface contained one recommended untitled movie and each user was told to rate that movie according to the explanation of the recommendation for the movie. To their surprise, it wasn't the explanations that contained the most information that was preferred by the majority of users, instead they preferred more simpler explanations. Examples of these are shown in [Figure 6.1].

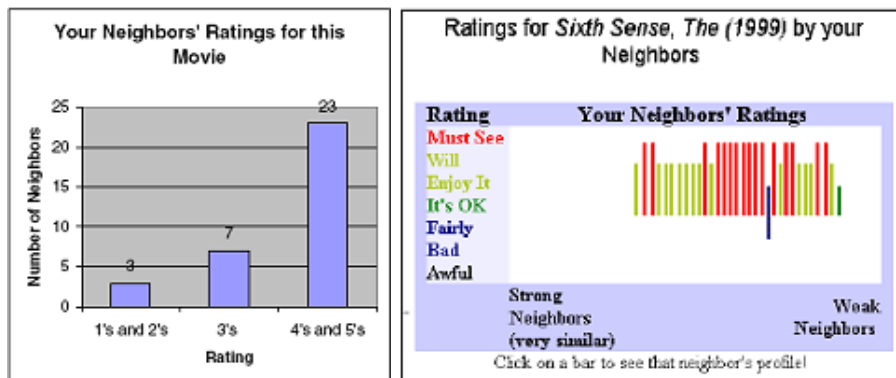


Figure 6.1: Expert users was found to prefer the explanation interface to the right while the majority of the users, i.e. ordinary users, preferred the more simpler one to the left.

Another paper [Bilgic and Mooney, 2005] argues that the approach taken in [Herlocker et al., 2000] only measures how well the explanation *promotes* the recommended item, i.e. convinces the user to adopt the item. They suggest that instead explanation interfaces should not only be selected based on how good they are at promoting items but also on what the users really thinks of the items after they have tried them out, which they refer to as the user *satisfaction* of the recommended item.

Three different explanation interfaces were tested; neighborhood, keyword and influence based. The keyword based explanations consist of the keywords in the user profile that had most influence on why the item was recommended [Figure 6.2]. The neighborhood style explanation uses the opinions of the

users neighborhood in the explanation for a recommended item, as depicted in [Figure 6.3]. The influence style explanation explains the recommendation by telling the user how his interaction with the system has led to this recommendation, it does so by showing a table that explains what had the most impact on the recommended item [Figure 6.4].

Slot	Word	Count	Strength	Explain
DESCRIPTION	HEART	2	94.14	<a href="#">Explain</a>
DESCRIPTION	BEAUTIFUL	1	17.07	<a href="#">Explain</a>
DESCRIPTION	MOTHER	3	11.55	<a href="#">Explain</a>
DESCRIPTION	READ	14	10.63	<a href="#">Explain</a>
DESCRIPTION	STORY	16	9.12	<a href="#">Explain</a>

Title	Author	Rating	Count
<a href="#">Hunchback of Notre Dame</a>	Victor Hugo, Walter J. Cobb,	10	11
<a href="#">Till We Have Faces : A Myth Retold</a>	C. S. Lewis, Fritz Eichenberg,	10	10
<a href="#">The Picture of Dorian Gray</a>	Oscar Wilde, Isabel Murray,	8	5

Figure 6.2: The keyword style explanation. By clicking on explain the second picture appears, revealing where the keyword appears. Rating is the user's rating for that item and count is how many times the keyword appears.

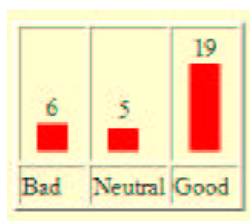


Figure 6.3: The neighborhood style explanation.

BOOK	YOUR RATING Out of 5	INFLUENCE Out of 100
<a href="#">Of Mice and Men</a>	4	54
1984	4	50
<a href="#">Till We Have Faces : A Myth Retold</a>	5	50
<a href="#">Crime and Punishment</a>	4	46
<a href="#">The Gambler</a>	5	11

Figure 6.4: The influence style explanation

In order to compare these method to each other, they designed a user study in which thirtyfour users were told to first rate an item according to the explanation and then try it out and rerate it according to what they really thought about it. The difference between these two ratings was measured and the explanation system with the minimum difference would be considered the best.

Their result from this user study shows that the keyword and influence style explanation was most satisfactory (almost identical) and that the neighbourhood style was least satisfactory. From the promotion perspective of explanations, the neighborhood based style performed best, people clearly overrated items that were explained by how their neighbors thought of them.

These two studies clearly shows that explanations have influences over how a user will judge a recommendation. Users seem to prefer the opinions from their neighbors, which both studies show, but the second study also shows that it is very likely that the users have too much faith in others opinion. Additionally the second study shows that the promotion ability of a recommendation is not good enough, it should also let the user assess the true quality of the recommended item. In MOV SOM, explanations either encourage the user to explore the items further or explain why they should be avoided.

### 6.3 Interactive design of recommender systems

Two important questions that must be considered in designing a highly interactive user interface is *what user needs are satisfied by interacting with the system* and *what specific system feature leads to satisfaction of those needs* [Sinha and Swearingen, 2002]. Interaction with a recommender system usually means that the user provides information about himself as input, the system processes that information and generates an output of recommendations that usually are displayed as a ranking list with some explanations about the reasoning behind these recommendations. Some findings made in the study done by [Sinha and Swearingen, 2002] suggests that users must be allowed to explore the given recommendations, refine recommendations e.g. rerate them or exclude/include new information into his user profile and have access to detailed information about the recommended items. Users also seem to be more willing to provide more information about themselves if it leads to better recommendations. MOV SOM has a clear and well defined goal; it should be fun to search for movies, both new and old ones. The process of rating and seeking out similar movies is taken into a new dimension, instead of the commonly used one-dimensional lists of movies ranked by some parameters, MOV SOM visualizes similar movies on a two-dimensional map, making rating and exploring of movies more intuitive and funnier. Users with a need of quickly locating movies similar to a certain movie are supported by the basic MOV SOM and users that want help in deciding if a certain movie is in his taste are supported by Personalized MOV SOM.

## Chapter 7

# Visual recommendations

Most recommender systems given e.g. a single movie are able to produce a Top N list of recommendations:

1. Movie 1
2. Movie 2
3. Movie 3
4. ...


In which the numbering (if available) indicates the relative "ranking" of the recommendations, the first ranked recommendation is by the system determined as the strongest recommendation, the third ranked recommendation is weaker than the two above it. Recommendations may be ranked by predicted value, similarity or other similar ranking metrics. In [Figures 7.1–7.4] various versions of Top N lists are displayed based on just a single movie, in [Figure 7.5] a Top N list is shown based on an entire rating profile.



Figure 7.1: MovieLens QuickPick feature, displaying top recommendations given the movie "The Thing (1982)" (discussed in Part I).




Customers who bought **The Thing [1982]** also bought:



**An American Werewolf in London : Two Disc 21st Ar**  
 DVD ~ David Naughton  
 Release Date: October 10, 2005  
 Used & new from £3.00

☐ I Own It ☐ Not interested x|☆☆☆☆☆ Rate it


---



**The Fog [1979]**  
 DVD ~ John Houseman  
 Release Date: October 18, 2004  
 Used & new from £3.73

☐ I Own It ☐ Not interested x|☆☆☆☆☆ Rate it

---



**They Live [1989]**  
 DVD ~ John Carpenter  
 Release Date: October 21, 2002  
 Used & new from £4.21

☐ I Own It ☐ Not interested x|☆☆☆☆☆ Rate it

Figure 7.2: Amazon displaying top recommendations on "The page you made" (discussed in Part I) given that I visited the product page for the movie "The Thing (1982)"

## Recommendations for The Thing (1982)

[How do these recommendations work?](#)

Suggested by the database	Look up in IMDb	Showtimes (US only)	Available @Amazon	User Rating
<a href="#">Dawn of the Dead</a> (2004)	IMDb			7.3
<a href="#">Day of the Dead</a> (1985)	IMDb			6.7
<a href="#">Alien: Resurrection</a> (1997)	IMDb			6.0
<a href="#">Beyond Re-Animator</a> (2003)	IMDb			6.0
<a href="#">Doom</a> (2005)	IMDb			5.2
<a href="#">The Lord of the Rings: The Return of the King</a> (2003)	IMDb			8.9
<a href="#">Sin City</a> (2005)	IMDb			8.4
<a href="#">Future Wars</a> (2004)	IMDb			4.2
<a href="#">War of the Worlds</a> (2005)	IMDb			6.8
<a href="#">Dawn of the Dead</a> (1978)	IMDb			7.8
<b>Tip:</b> if you want to see if a movie is showing in a cinema near you, click the film roll. (USA only)				

Figure 7.3: IMDB's recommendation center (discussed in Part I) displaying top recommendations for the movie "The Thing (1982)".

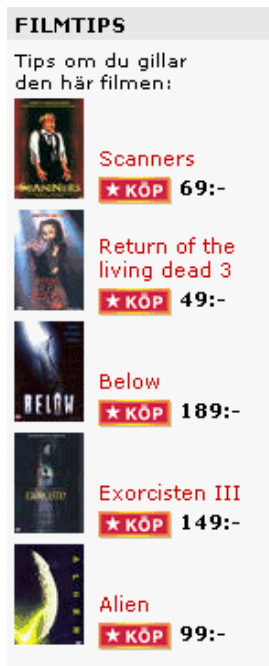


Figure 7.4: Top recommendations displayed on the product page of the movies "The Thing (1982)" of the Swedish e-commerce store Discshop

Predictions for you ↕	Your Ratings	Movie Information
★★★★★	Not seen ▾	<a href="#">Black Christmas (1974)</a> DVD VHS info   imdb   add tag Comedy, Horror, Mystery, Thriller
★★★★★	Not seen ▾	<a href="#">Blood of Heroes, The (1988)</a> DVD info   imdb Action, Sci-Fi [add tag] Popular tags: <a href="#">australia</a> , <a href="#">fighting</a> , <a href="#">delroy lindo</a>
★★★★★	Not seen ▾	<a href="#">Sisters (1973)</a> DVD VHS info   imdb   add tag Horror, Thriller
★★★★★	Not seen ▾	<a href="#">Unsane (Tenebre) (1982)</a> DVD VHS info   imdb Crime, Horror, Mystery, Thriller - <a href="#">Italian</a> , <a href="#">English</a> [add tag] Popular tags: <a href="#">Giallo</a> , <a href="#">Italian</a>
★★★★★	Not seen ▾	<a href="#">Collector, The (1965)</a> DVD VHS info   imdb   add tag Drama, Horror, Thriller
★★★★★	Not seen ▾	<a href="#">Howling, The (1980)</a> DVD VHS info   imdb Horror, Mystery [add tag] Popular tags: <a href="#">werewolf</a>
★★★★★	Not seen ▾	<a href="#">Dawn of the Dead (2004)</a> DVD VHS info   imdb Action, Drama, Horror, Thriller [add tag] Popular tags: <a href="#">mmm... brains...</a> , <a href="#">Zombie Movie</a> , <a href="#">bmovie</a>
★★★★★	Not seen ▾	<a href="#">Dark Water (Honogurai mizu no soko kara) (2002)</a> DVD VHS info   imdb Horror, Mystery, Thriller - <a href="#">Japanese</a>

Figure 7.5: MovieLens "Top picks for you" will displayed a list of movies ranked by predictions made based on a user's entire ratings profile.

Within information retrieval the traditional approach is again to return a ranked list of documents, Google uses a ranking metric termed PageRank [Brin and Page, 1998] to order the retrieved documents in a ranked list [Figure 7.6].



Figure 7.6: Google documents retrieved using the search query "the thing".

In the traditional 1-dimensional case the user usually focus his interest on the top of the list, and most often that is all that the user is provided with. Providing the user with the entire list of ranked items or documents may prove of little use to the user as it may be hard to judge the relative relevance of items on positions in the middle of the list. This does not encourage the user to explore the recommendation list. An alternative is to display a 2 dimensional list, i.e. a map of recommendations, since there no longer will be any "Top", focus can be put on the explorative part.

As previously discussed in Part II, there are different methods for projecting high dimensional manifolds onto a two dimensional display, the usage of a two dimensional display of document similarity was suggested for use in information retrieval as early as 1969 by Sammon [Sammon Jr., 1969]. Sammon suggested an experimental system in which a user would examine the 30 highest ranked documents retrieved by the system, the examination would involve looking at the abstracts of the documents and while doing so the user would indicate which documents he considers relevant. Having done this, a scatter diagram [Figure 7.7] would be shown where each of the 30 documents would be indicated by an *I* or *R* depending upon its relevance. In addition the original query vector would be displayed as a *Q* on the scatter diagram. The user could then examine the relative positions of the documents in the mapping, the user would then select one or more relevant documents to be used to generate new query vector(s). The concept being that a query vector can be moved to highly relevant regions of the document space by interacting at a display console with a geometric representation of the space.

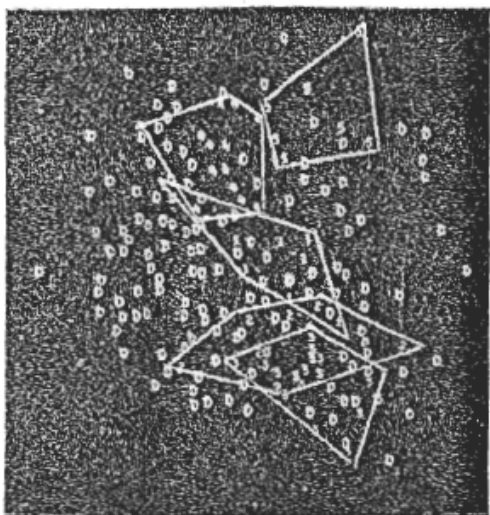


Figure 7.7: A suggested experimental document retrieval system. Given the 30 most relevant documents to a user query, the user can indicate which documents he considers relevant and irrelevant, a sammon mapping (or similar) could then be done of the documents to display their relative location, documents would be labeled as I or R depending on their user decided relevance. The original query vector is also displayed. The user can then make new queries by selecting one or more documents (using e.g. a light pen on a crt display) to base new queries on. The concept being that a query vector can be moved to highly relevant regions of the document space by interacting at a display console with a geometric representation of the space.

In [Silaghi, 2004] it is discussed how the automated categorization of texts into predefined categories can be visualized. In a information retrieval system that retrieves text documents, the similarity between documents can then be visualized by showing how similar the categories the text documents retrieved belong to are. It is additionally stated that visual techniques harness human perceptual capabilities to detect patterns and outliers in visual information. A algorithm that visualizes the *distances* between document categories is presented, a test dataset consisting of 20 different categories (newsgroups) with 1000 documents for each category (posts to each newsgroup) is used. A two dimensional (called bi-dimensional in the paper) display of the categories from the test dataset is displayed in [Figure 7.8].

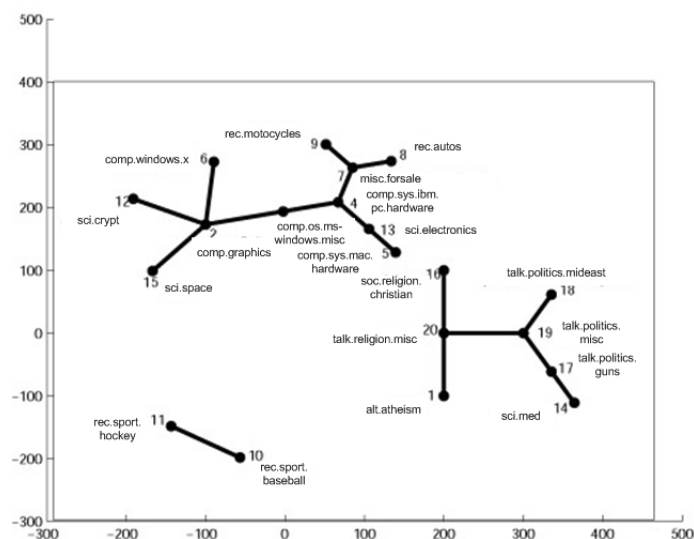


Figure 7.8: Two dimensional visualization of distances between text document categories (newsgroups).

However, it is not necessary for the visualization to preserve distances, only the documents relative similarity needs be preserved. A information retrieval system for newsgroup documents that visualizes the *ordering* of newsgroup documents as well as newsgroups using what is referred to as the WEBSOM method is presented in [Honkela et al., 1996]. The WEBSOM method uses the SOM algorithm to create two dimensional document maps, this means that distances wont be preserved, however visualization of document similarities will be easy to visualize, as will the discovery of related documents be. A presentation of the latest WEBSOM is given in [Kohonen et al., 2000] where a two dimensional map of over 80 million newsgroup documents is created. An interface for the WEBSOM map is further described in [Lagus, 2000]. The interface is build on the idea of zooming in on the map on locations containing information of interest, labels are displayed on the map to give an idea of the information stored at that map location, and exploring nearby locations on the map once a location containing interesting information has been found [Figure 7.9].

Figure 7.9: The WEBSOM method was used to create a two dimensional map of over 80 million newsgroup documents. The map is visualized as a heat map where dark colors indicate few documents, light colors indicate many documents. The labeling indicates the category most representative for the type of documents contained in the location. At the highest level of zoom no documents are visible, only the labeling which can be used as an indication of newsgroup similarity. Clicking on a location on the map zooms in (in this case in three steps) on the node until finally the documents in the node are shown. By moving to nearby nodes further similar documents can be found.

In [Novak et al., 2003] a information retrieval system for articles is presented, a first application context of their work is the Internet platform [netzspannung.org](http://www.netzspannung.org)<sup>1</sup> which aims at establishing a knowledge portal that provides insight into the intersection between digital art, culture and information technology. They provide a SOM based visualization system of articles with an exploratory interface [Figure 7.10]. In their paper they also discuss the concept of personal maps [Figure 7.11], the idea being that expert users use their system to find articles within their expertise and interest and add those to their personal map where they themselves visually indicate the articles relative similarity. By comparing personal maps with other user's, the system can recommend personal maps between users. However the idea of personal maps is not further explained (specifically how to compare the maps is not explained) and is only presented as a future work.

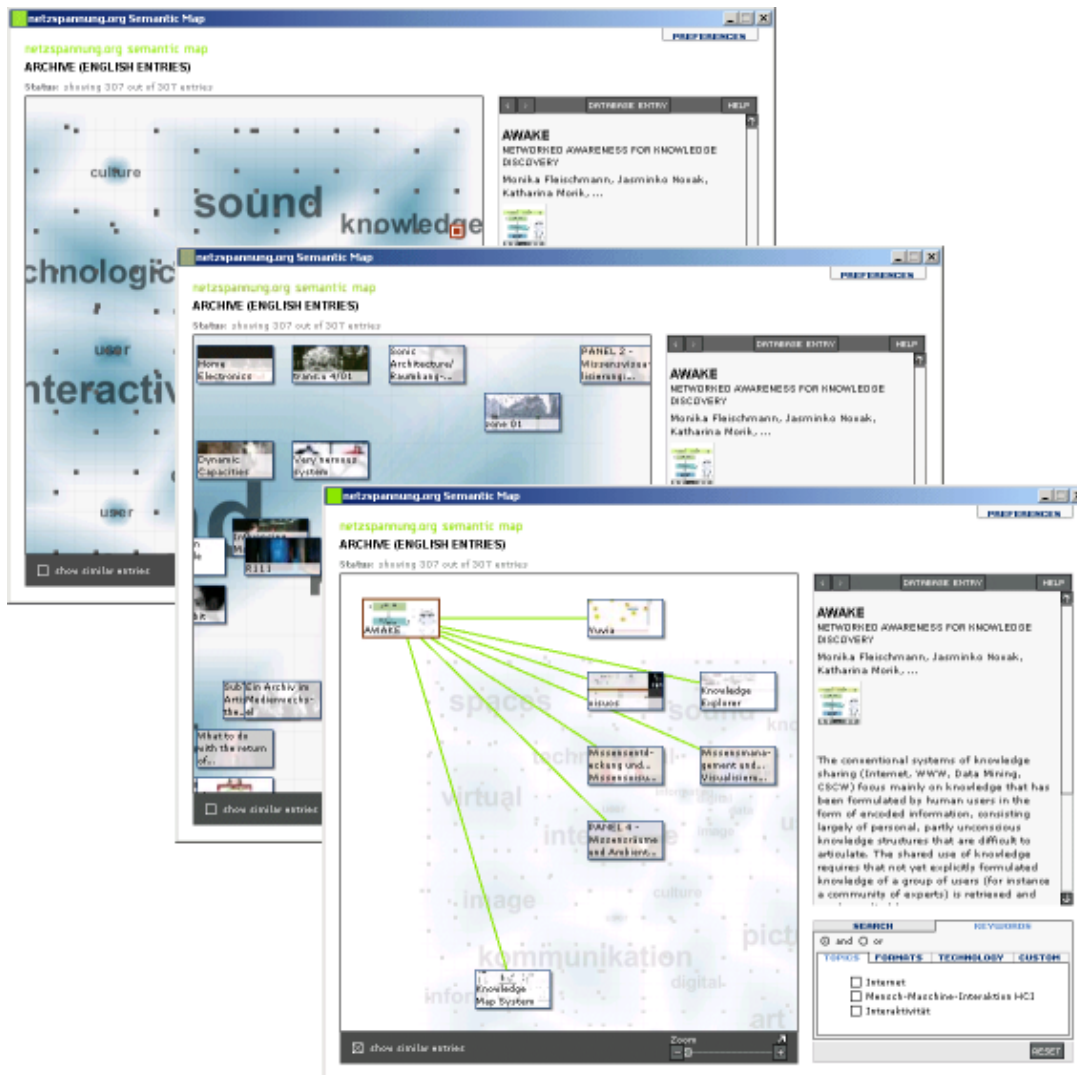


Figure 7.10: Netzspannung interface for exploring similar articles. At the highest zoom only various category labels are visible, however some dots are available that can be clicked to look at a document probably representative for the category. In this case a node containing information on the AWAKE project was selected. By zooming in on the map (which is done by dragging a slide bar) more detailed information is shown on the map about documents in the zoomed in area. By selecting an article and asking for documents similar to it a graph is displayed indicating locations (far or near on the map) containing similar documents. The idea probably being that besides exploring nearby locations on the map for similar documents, the user can jump to seemingly distant locations that still contain documents of some relevance.

<sup>1</sup><http://www.netzspannung.org/>

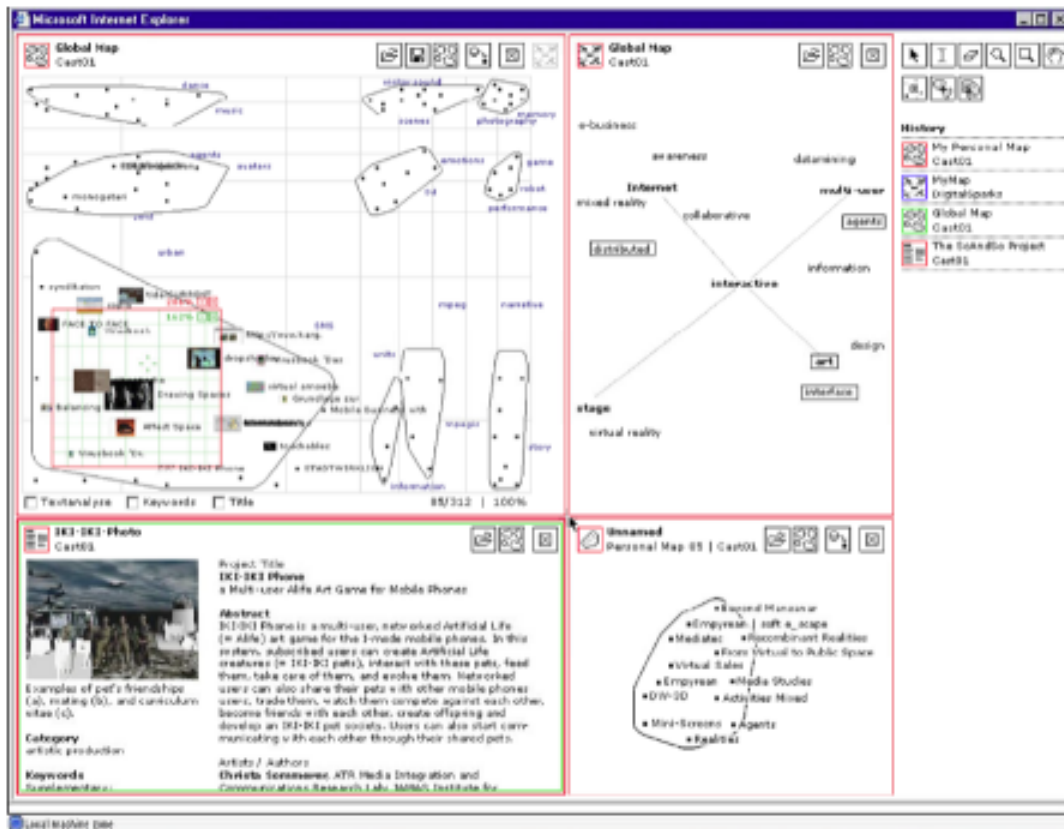
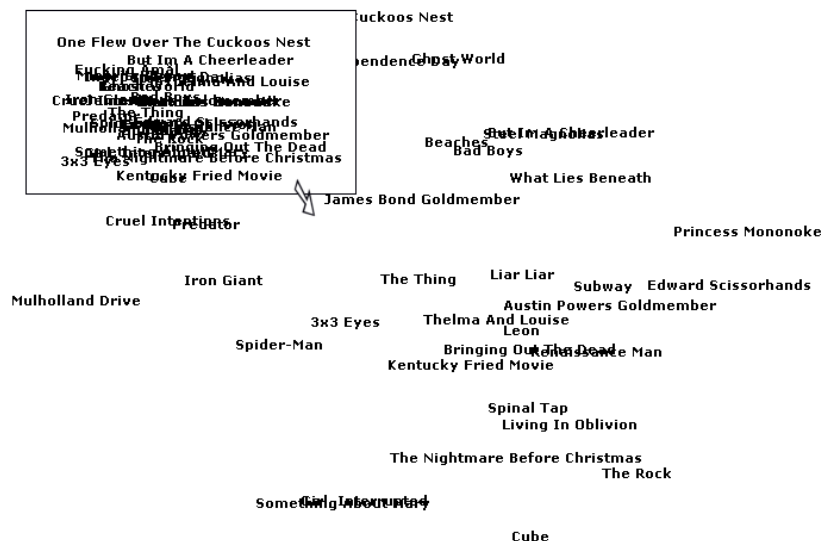


Figure 7.11: What is called the "Knowledge explorer system interface" in [Novak et al., 2003]. The upper left hand window is a "content map" with what is called the "concept map" to its right. The content map displays documents while the concept map attempts to motivate the structure that gave rise to the document display. The lower-right hand window displays a personal map, users are able to drag documents from the document map onto it and arrange them as they deem suitable. And the lower-left hand window displays information about the currently selected document.



In [Pampalk, 2001] the SOM algorithm is applied to music files represented using features of the actual music. Users can then explore the map based visualization "Islands of Music" to find similar music [Figure 7.12].

Figure 7.12: "Islands of music". A map is created using the SOM that displays music and allows for exploration of similar music pieces. Assuming that the user knows neither artist nor title the pieces of music are represented by white balls instead. Moving the mouse over one of these white balls displays the tool tip (yellow popup) with the artist and title of the song.



<sup>2</sup><http://www.gnovies.com/>



Indeed most of the examples presented can be seen as making recommendations, they are after all displaying documents or music similar to some known (selected) document or music. Within a recommender system the traditional Top N recommendation process can be made transparent to the user by displaying recommended items on a map, i.e. showing a two dimensional Top N list [Figure 7.14]. Instead of the traditional one dimensional Top N list where the first entries are easy to grasp as relevant and the remaining items harder to judge relevance for and not easy to explore, the two dimensional Top N list's map display will further encourage the user to explore the recommendations since moving around in the map is intuitive and easily understandable. As the only change that has been made is the display of the Top N list, this is termed a visual recommendation which by the use of a visual display adds value to a traditional Top N list while at the same time encouraging exploration of recommendations. By studying any nearby item in the map, its neighbourhood of similar items is immediately visible adding on an understanding for what kind of recommendation it is.



Figure 7.14: Basic MOV SOM map, looking up the movie "The Thing" shows its position on the MOV-SOM map and its surrounding similar movies. A two dimensional Top-N list, that encourages exploration of recommendations.

## Chapter 8

# MOVSOM - State of the art

*“Always listen to experts. They’ll tell you what can’t be done and why. Then do it.”*  
– Robert Heinlein

MOVSOM is a recommender system for movies that will tell a user how much he will like a certain movie and why he should trust the given recommendation. The system visualizes movie recommendations and encourages users to explore the recommendations. Recommendations consist of similar movies visualized (as a two dimensional Top N list) in the form of a map. The system has two usage modes:

1. Non-profiled based MOVSOM (Basic MOVSOM)
2. Profile based MOVSOM (Personalized MOVSOM)

In the first mode the system visualizes recommendations based on the movie the user searched for, thus requiring no user profile, in the second profile based mode, the user is able to rate and tag movies and during exploration of recommendations also get predictions for movies. In addition the profile based mode introduces a user map, consisting of all MOVSOM users. The user is able to locate himself on the map, as well as see where users that have seen the active item are in relation to himself. As predictions are made, the users making the predictions will also be indicated on the user map.

MOVSOM does not focus on providing users with information about movies, instead the visualized relationships between movies is the focus, however a basic set of identifying information is given about each movie, including title, release year and director. Instead links to the IMDb are always provided, encouraging users that find seemingly interesting movies to visit the IMDb for actual information.

### 8.1 MOVSOM map

MOVSOM uses the SOM algorithm to create a SOM that allows mapping of movies onto a two dimensional map. However the SOM will map multiple movies into the same nodes, i.e. movies end up “stacked on top of each other” in the same node on the map, thus to make it possible to provide visual recommendations and to navigate the map by only looking at the relationship between movies, the map needs to be “stretched” out such that movies in the same node are no longer stacked on top of each other.

The map stretching can be done in several ways, a rough but simple solution is to determine how many movies a node in the map contains on an average (or at a maximum, but that may lead to a unnecessarily sparse stretchmap) and then transform each node into a (quadratic) grid large enough to contain such a number of movies and simply tile up each node’s grid into a larger stretch map [Figure 8.1]. Nodes with more than the average number of movies would need to be treated specially by e.g. listing the overflowing movies separately below the viewed map section (assuming the whole map is not viewed at once). Nodes with less than the average number of movies would have its data placed out in some suitable fashion, e.g. randomly (to cover more of the node’s grid) or focused in the center of the node’s grid (to emphasize the data is equal to each other).

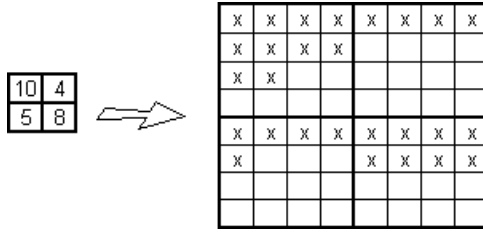


Figure 8.1: Stretching out a 4x4 map with 27 items mapped out into 4 nodes to a 16x16 map where each item lies in its own node.

However, despite that movies ended up in the same node in the SOM, i.e. had the same BMN, they are not going to be exactly the same. A user may pick up on this and consider the randomly selected relative placement of two movies within a node's grid in the stretch map as an indication of a relationship, e.g. since movie A is below movie B, the user may select to navigate down the map for more movies like B. Unless movies are truly equal in a node the movies within a node's grid in the stretchmap should be ordered in some fashion.

A solution would be to create a smaller SOM using only the movies in each node, essentially making a hierarchical SOM and puzzling the hierarcies together into one level. However such a scheme would fail to account for the relationships between movies in the node and movies in neighoring nodes, thus not truly solving the problem.

A optimal solution to map stretching would be to simply arrange the movies within the node's grid such that movies as far as is possible are placed on the edge adjacent to the grid for the neighboring node they are most similar. Thus movies in a node more similar to the node on the "left" should be placed near the left edge of the node. Such a scheme for map stretching is essential to MOV SOM's ability to provide visual recommendations where the users themselves can determine movie similarity based on distances in the movie map.

## 8.2 MOV SOM interface

The MOV SOM interface consists of two basic parts [Figure 8.2], the MOV SOM map and the sidebar.

By searching for movies by their title the section of the map containing the movie will be shown with the searched for movie placed in the middle of the map. Clicking on nearby movies will display information about that movie and recenter the map around the new movie, by this method the map can be navigated.

The sidebar always shows at least the title of the currently selected movie (if any) and provides a link to the IMDb where more attribute information can be obtained for the movie. In the personalized mode the user can always rate and add tags to the movie using the sidebar.

In the basic mode the sidebar [Figure 8.3] displays a U-matrix that shows the global movie map with the active movie indicated. The U-matrix can also be used to jump around in the movie map by simply clicking in it. By showing the U-matrix as a global view of the whole movie map the user can get a clearer sense of direction when searching and moving around in the map.

If the user has a rating profile a U-matrix for the user map is also shown in the sidebar [Figure 8.4(a)] indicating where the active user's position and where users that have rated the active movie are located in the user map. Groups of users are shown as green, yellow or red dots, indicating the groups preference for the active movie (red = don't like it, yellow = neutral, green = likes it). Two predictions are also shown, one based on the active user's average rating on movies similar to the active movie and one based on the active user's neighbors opinion on the active movie. Additionally a goodness recommendation combining and summarizing the shown visual recommendations and predictions into a textual recommendation is provided.

The sidebar also contains tags [Figure 8.4(b)], descriptive keywords users can associate with movies in the moviemap, shown as a ordered tag cloud where similar tags are placed near each other and sized according to have many times the tag has been assigned to the movie by different users. The tags provide additional movie information in place of complete movie attribute data.

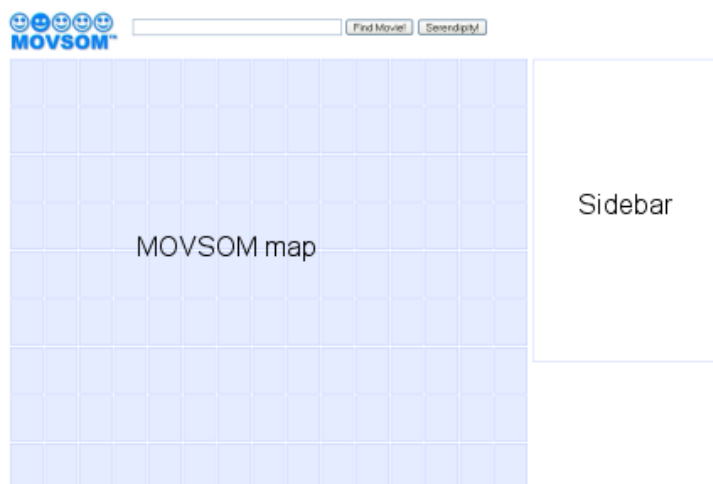


Figure 8.2: MOV SOM interface showing the placement of the map (left) and the sidebar (right). Movies are searched for using the search interface at the top of the interface. A search and a serendipity button is provided, the later showing a movie at random in the map.

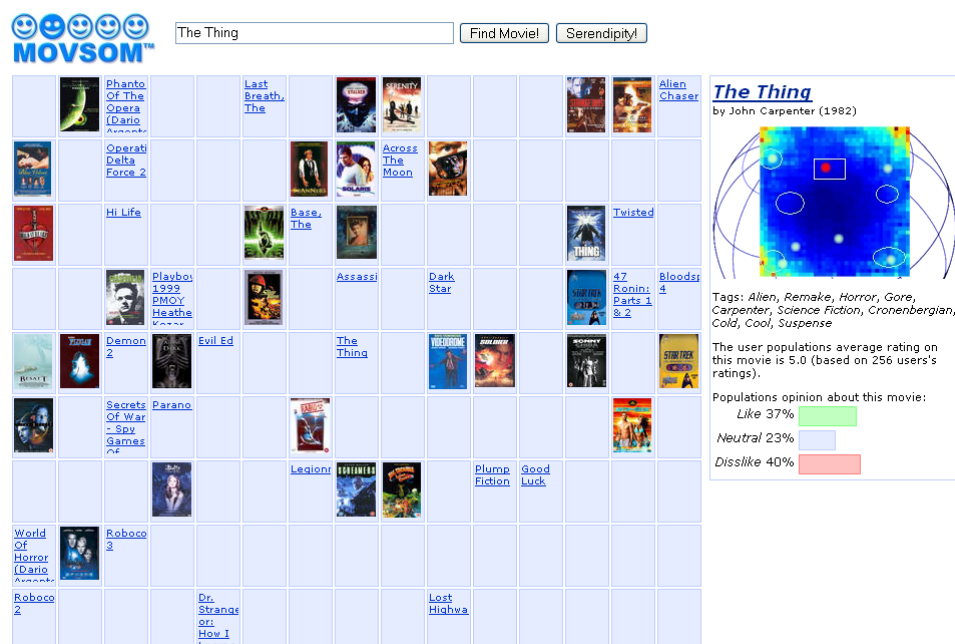
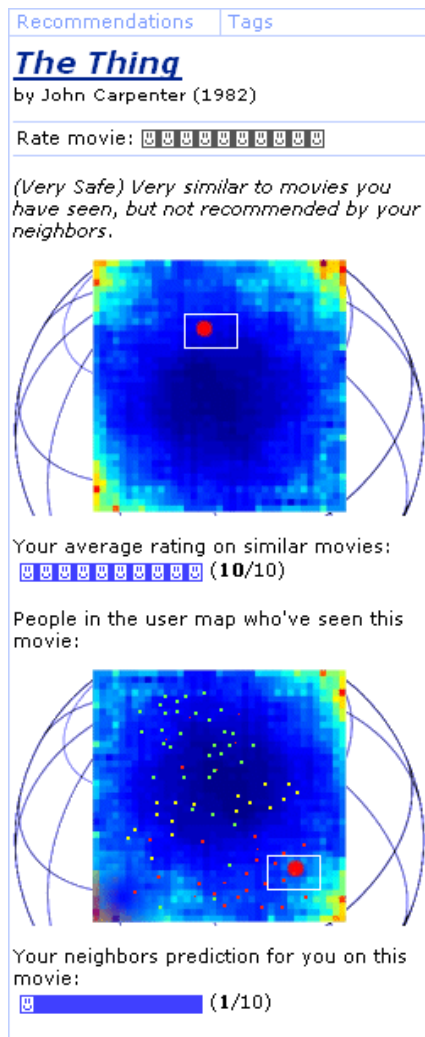
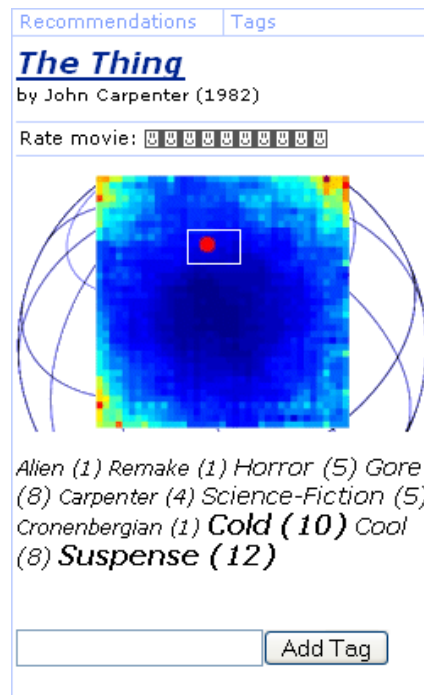


Figure 8.3: MOV SOM in basic mode, the user searched for "The Thing", the part of the MOV SOM map containing the thing at its center is shown. Note that movies are either displayed using a icon representing the movie (typically the DVD cover etc.) or its name in plain text. Since MOV SOM is in the basic mode the sidebar shows only non-personalized information, tags, populations average rating on movie and populations opinion on the movie. The U-matrix indicates where the movie searched for is located.



(a) The active user's average rating on movies similar to the active movie is shown, as well as the active user's closest neighbors prediction on the movie. In addition a textual goodness recommendation is shown at the top. The lower U-matrix in this case is of a user map, where the red dot indicates the active user's location in the map, the green, yellow and red dots indicate where groups of users that like, stand neutral or dislike the active movie are located. The idea being that the user can judge where people who like the current movie are located relative himself.



(b) The users can tag movies. Tags are ordered based on similarity to each other and sized according to number of times tag has been assigned to the movie by different users.

Figure 8.4: Sidebars shown in the personalized mode, in which the active user has a rating profile.

## 8.3 MOV SOM architecture

The MOV SOM architecture will be described as consisting of several separate *modules*. Each module has a specific parameter requirement and output specification, typically each module is connected with one very specific part of the MOV SOM interface, the flow chart [Figure 8.5] gives a rough idea of how these modules will be used in the live application. A user is placed out and a series of arrows give an idea of the flow the user will set in motion when interacting with various modules.

**SOM module** Using the entire database of user ratings on movies a user SOM and a movie SOM is trained using e.g. the SOM implementation described Chapter 5.

**Map stretching module** Stretches the movie map created by the movie SOM such that movies "stacked on top of each other" in the same node are no longer stacked on top of each other, and instead are placed beside each other such that they have both a relationship to the movies in the same node and to the directly neighboring nodes.

**Map navigation module** Locates a movie in the stretched out maps and serves up the movie's immediate neighborhood in the stretched out map. Given the movies stretchmap position the movie's position in the movie U-matrix is marked (note that the U-matrix for the movie SOM is still representative of the cluster structure in the stretch map).

**Search module** Given a search query a movie database is searched for a movie matching the search query, the map navigation module is then automatically called to retrieve the corresponding map section.

**Recommendation module** Provides recommendations for an active user on an active movie, the recommendations are statistical non-personalized recommendations in the basic mode, however if the user has a profile the recommendations are personalized. The statistical recommendations calculated are based on the populations average rating on the active item. In particular the personalized recommendations include calculating the user's average rating on movies similar to the active movie and calculating the user's neighbors average rating on the active movie.

*Average rating prediction* The prediction can be made using a recommendation algorithm based on the SOMICF technique which simply calculates the active user's average weighted rating on similar movies, i.e. uses the Weighted Sum prediction algorithm. In this case similar movies can include all movies currently in the map section the user is viewing, plus a few nearby movies not visible in the current map section. This prediction can only be made when the user has rated a minimum number of movies. However, because of size of the movie map, even if the user has rated 20 - 100 movies, it is overall not likely those movies lie in the visible neighborhood of an movie the user wants a prediction for, thus the user will not always receive a average rating prediction, however it will be clear to the user why he isn't receiving any average rating prediction, he hasn't yet rated any movies similar to the active movie.

*Neighbor prediction* The prediction can be made using a recommendation algorithm based on the SOMUCF technique which uses the Weighted Deviation From Mean prediction algorithm, which calculates the active user's neighbors weighted deviation from their mean rating and adds it to the active users mean rating to get a prediction on the active movie. The neighborhood radius can be set dynamically to accommodate a sufficiently large part of the map. This prediction can only be made when the user has rated a minimum number of movies. Since neighbors of the active user can have seen movies not within the neighborhood of any movie the active user has seen, the neighbors may be able to produce predictions with a much larger coverage than for the average rating predictions.

*Goodness recommendation* The recommendation indicates the level of safety and serendipity of the movie. The recommendation can be based on the SOMGOODNESSCF technique, but give a textual recommendation instead of a weighted prediction based on the SOMUCF and SOMICF technique.

**Rating module** Adds the active user's rating on the active movie to the rating database  $R$ . If the number of ratings for the active user or the active movie exceeds a minimum ratings threshold, e.g. 5 or 20, then the active user or the active movie's position in the map will be recalculated.

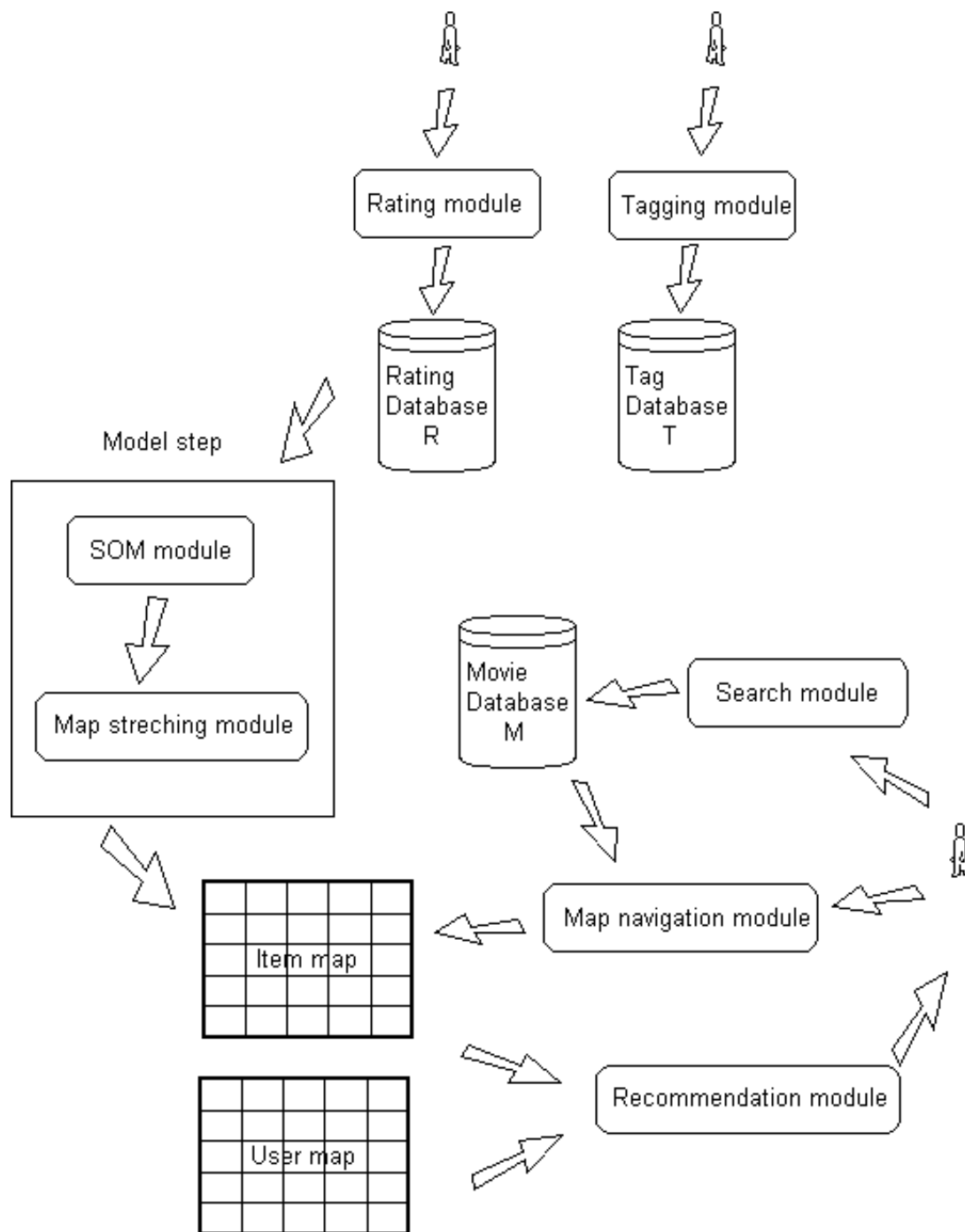


Figure 8.5: MOVSOM architecture flow chart. A user either requests a movie from the movie database, rates a movie or tags a movie, each action setting up a sequens of interactions between various modules. The model step is made when enough new ratings have been added to the rating database.

**Tagging module** Adds the active user's tag on the active movie to the tag database  $T$ .

## 8.4 Challenges

While the SOM introduces a model step that greatly reduces the constraints of the memory based recommendation part of the MOVESOM recommender system, some notable challenges are however introduced related to the introduction of new movies and new users.

Let the original set of movies and users used to create the user and movie SOM be denoted by  $M$  and  $U$  respectively.

### 8.4.1 Introduction of new movies and new users

If a new movie  $m \notin M$  is introduced to the system, since it has no ratings it is impossible to place it out on the movie map. As soon as a suitable number of users  $u \in U$  has rated the movie it can be placed out by comparing the movie's rating vector with all the reference vectors to find a closest reference vector, the map stretching module can then give the new movie a location in the stretched out map. (Variant of the SOM algorithm such as hierarchical SOM's and growing SOM's can also be used to accommodate for new users and movies.)

But a user  $u \in U$  ratings on the the new movie will not affect the user's position in the user map. Thus nomatter how many new movies are introduced into the system, the user's position in the user map will never change when rating the new movies.

If a new user  $u \notin U$  is introduced to the system, since the user has not rated any movies it is impossible to place the user in the user map. Hence the user can not initially receive predictions by any neighbors while browsing the movie map. The user can not receive any average rating prediction either as the user has not rated any movies. However, as soon as the user has rated a suitable number of movies  $m \in M$  the user can be placed out in the user map by comparing the user's rating vector with all the reference vectors to find a closest reference vector, the map stretching module can then give the new user a location in the stretched out user map. The user will then be able to receive predictions by his neighbors. But if the user only rates movies  $m \notin M$  the user will never be possible to place out in the user map, and since the user hasn't rated any movies in  $M$ , the user wont be able to receive any predictions of either kind until the SOM's are retrained with the new users and movies.

The challenge with making predictions for new movies and new users can be overcome to a limit by adding new movies and new users to the movie map and user map reference vectors correspondingly. Ratings on new movies  $m \notin M$  and by new users  $u \notin U$  will then affect movie and user positions in both maps just likes movies  $m \in M$  and users  $u \in U$  will. By requiring new users to initially rate a random sample of 5 - 20 movies  $m \in M$  the system will be able to place out the new user, and having done so the system will be able to update the corresponding reference vector and reference vectors of neighboring nodes to resemble the new user rating vector. It is not as straightforward to solve the problem of new movies, but a similar approach where users are asked to rate new movies placed in a "New movies queue" could be used.

### 8.4.2 Retraining the SOM

As the number of users and movies grows, retraining the SOM to accommodate for new users and movies will become necessary, however retraining will grow more and more timeconsuming as the dimensions of the user and movie rating vectors will grow. To this end some form of dimensionality reduction techniques must be applied. This however introduces a new challenge when introducing new users and movies as reference vectors are now on reduced dimensionality form. However at such a point the SOM and corresponding map would be made very large, reducing the frequent need for updating reference vectors. However, using variants of the SOM algorithm the dimensionality problem can possibly be solved more elegantly, such as creating lots of smaller maps and combining them into a large map (then retraining only has to be done on the smaller maps).



# Conclusions

In this thesis, we have described the architecture and implementation of a movie recommender system, called the MOV SOM. The system incorporates the SOM, developed by Teuvo Kohonen [Kohonen, 2001] as well as known recommendation techniques. Early versions of MOV SOM was developed in two projects, in connection with the courses *"artificial neural networks"* and *"information retrieval on the internet"* taught at the university of Uppsala during the year 2004. This thesis has since the fall of 2004 been an ongoing project that finally resulted in this master thesis. In this thesis we have shown that MOV SOM is comparable to traditional techniques used by recommender systems when it comes to accuracy of the given recommendations. MOV SOM offers a highly interactive user interface, and implements transparency that gains trust among its users, as defined by recent studies. A new way of presenting recommendations is utilized by MOV SOM. We claim that this visualization of recommendation is more intuitive than the traditional way of presenting recommendations as one-dimensional lists, in the sense of visualized similarity among the items. Our study of different prediction algorithms has convinced us that the accuracy of present algorithms are as good as they can be, in the sense that a few decimals in either direction of the mean absolute error doesn't have any impact on the overall acceptance by the user of the recommendation. The use of the SOM algorithm as a part of a recommender system is promising, its topologically preserving capabilities has been shown to be a very suitable feature to use in a recommender system.

*"O Spot, the complex levels of behaviour you display connote a fairly well-developed cognitive array."* – Data

## Future work

An important issue for future work is to research and explain how MOV SOM incorporates what is commonly referred to as Web 2.0. How can for example MOV SOM be used in the new Web 2.0 phenomena commonly referred to as *collaborative tagging*. Using MOV SOM tags can be visualized in new and interesting ways as opposed to being used as simple alphabetical lists of tags associated with a item.

Navigating the semantic web, another often heralded feature of Web 2.0, have been attempted using tags and hierarchical categories, it is our intention in the near future to provide a browser plug-in interface (for which Firefox seems to be a good choice) that incorporates visual navigation of both the collaborative and the semantic *movie* web.

Another interesting direction for future work is how recommender systems can play an important part of online communities, i.e. social collaborative filtering.



Figure 8.6: Five by Five! The Firefox browser plugin for *VISUALLY* navigating the collaborative and semantic movie web. Soon, in a future near you!

Further research regarding the interface of MOV SOM is also planned, features such as personalized maps, where each user creates his own map consisting of movies, ordered and labeled by the user himself. This maps can then be compared to other users maps and also be used by MOV SOM to base recommendations on

Algorithmically we would like to investigate different SOM techniques like *hierarchical* SOMs and *growing* SOMs to see if they can contribute to making the interface even more interesting without losing any of its current capabilities. We are also interested in researching the possibilities of using a spherical SOM to increase the intuitivity of the maps.

Practically we will investigate the implications of the usage of very large datasets. Finally we are also interested in studying the possibilities of using the MOV SOM interface for navigating between sites on the web.

*“It is, in an unprecedented sense, your movie. So, if it sucks, it’s your fault.”* – Joss Whedon

# References

- [Agrawal et al., 1993] Agrawal, R., Imielinski, T., and Swami, A. N. (1993). Mining Association Rules between Sets of Items in Large Databases. In Buneman, P. and Jajodia, S., editors, *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C.
- [Amari, 1980] Amari, S.-I. (1980). Topographic Organization of Nerve Fields. *Bulletin of Mathematical Biology*, 42:339–364.
- [Anderson, 2004] Anderson, C. (2004). The Long Tail. *Wired*, 12(10).
- [Arbib, 2003] Arbib, M. A. (2003). Part I: Background: The Elements of Brain Theory and Neural Networks. In Arbib, M. A., editor, *The Handbook of Brain Theory and Neural Networks*, pages 3–23. The MIT Press, 2 edition.
- [Baeza-Yates and Ribeiro-Neto, 1999] Baeza-Yates, R. A. and Ribeiro-Neto, B. (1999). *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
- [Balabanovic and Shoham, 1997] Balabanovic, M. and Shoham, Y. (1997). Fab: Content-Based, Collaborative Recommendation. *Communications of the ACM*, 40(3):66–72.
- [Basu et al., 1998] Basu, C., Hirsh, H., and Cohen, W. W. (1998). Recommendation as Classification: Using Social and Content-Based Information in Recommendation. In *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, pages 714–720, Menlo Park, CA, USA. American Association for Artificial Intelligence.
- [Belkin and Croft, 1992] Belkin, N. J. and Croft, W. B. (1992). Information Filtering and Information Retrieval: Two Sides of the Same Coin? *Communications of the ACM*, 35(12):29–38.
- [Bilgic and Mooney, 2005] Bilgic, M. and Mooney, R. J. (2005). Explaining Recommendations: Satisfaction vs. Promotion. In *Proceedings of Beyond Personalization 2005, the Workshop on the Next Stage of Recommender Systems Research(IUI2005), San Diego, California*.
- [Billsus and Pazzani, 1998] Billsus, D. and Pazzani, M. J. (1998). Learning Collaborative Information Filters. In *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 46–54. Morgan Kaufmann, San Francisco, CA.
- [Breese et al., 1998] Breese, J. S., Heckerman, D., and Kadie, C. (1998). Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In *Proceedings of the 14th Annual Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 43–52. Morgan Kaufmann.
- [Brin and Page, 1998] Brin, S. and Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117.
- [Burke, 2002] Burke, R. (2002). Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370.
- [Carpenter, 1989] Carpenter, G. (1989). Neural Network Models for Pattern Recognition and Associative Memory. *Neural Networks*, 2(4):243–257.

- [Claypool et al., 1999] Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., and Sartin, M. (1999). Combining Content-Based and Collaborative Filters in an Online Newspaper. In *Proceedings of ACM SIGIR Workshop on Recommender Systems*.
- [Cohen, 1992] Cohen, J., editor (1992). *Communications of the ACM. Special issue on information filtering.*, volume 35, New York, NY, USA. ACM Press.
- [Conner and Herlocker., 1999] Conner, M. O. and Herlocker., J. (1999). Clustering Items for Collaborative Filtering. In *Proceedings of the ACM SIGIR Workshop on Recommender Systems*.
- [Cottrell et al., 1998] Cottrell, M., Fort, J.-C., and Pagès, G. (1998). Theoretical aspects of the SOM algorithm. *Neurocomputing*, 21(1–3):119–138.
- [Crawford, 1997] Crawford, D., editor (1997). *Communications of the ACM*, volume 40, New York, NY, USA. ACM Press.
- [Denning, 1982] Denning, P. J. (1982). ACM President’s Letter: Electronic Junk. *Communications of the ACM*, 25(3):163–165.
- [Deshpande and Karypis, 2004] Deshpande, M. and Karypis, G. (2004). Item-Based Top-N Recommendation Algorithms. *ACM Transactions on Information Systems*, 22(1):143–177.
- [Erwin et al., 1992] Erwin, E., Obermayer, K., and Schulten, K. J. (1992). Self-Organizing Maps: Ordering, Convergence Properties and Energy Functions. *Biological Cybernetics*, 67:47–55.
- [Fleischman and Hovy, 2003] Fleischman, M. and Hovy, E. (2003). Recommendations Without User Preferences: A Natural Language Processing Approach. In *Proceedings of the 7th International Conference on Intelligent User Interfaces (IUI)*. Miami Beach, FL.
- [Flexer, 1997] Flexer, A. (1997). Limitations of Self-Organizing Maps for Vector Quantization and Multidimensional Scaling. In Mozer, M. C., Jordan, M. I., and Petsche, T., editors, *Advances in Neural Information Processing Systems 9. Proceedings of the 1996 Conference*, pages 445–51. MIT Press, London, UK.
- [Fukushima, 1975] Fukushima, K. (1975). Cognitron: A Self-Organizing Multilayered Neural Network. *Biological Cybernetics*, 20:121–136.
- [Fukushima, 1980] Fukushima, K. (1980). Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position. *Biological Cybernetics*, 36:193–202.
- [Gabrielsson and Gabrielsson, 2004] Gabrielsson, S. and Gabrielsson, S. (2004). MOV SOM-II - Analysis and Visualization of Movieplot Clusters. Uppsala University, Department of Information Technology.
- [Goldberg et al., 1992] Goldberg, D., Nichols, D., Oki, B. M., and Terry, D. (1992). Using Collaborative Filtering to Weave an Information Tapestry. *Communications of the ACM*, 35(12):61–70.
- [Goldberg et al., 2001] Goldberg, K., Roeder, T., Gupta, D., and Perkins, C. (2001). Eigentaste: A Constant Time Collaborative Filtering Algorithm. *Information Retrieval*, 4(2):133–151.
- [Graef and Schaefer, 2001] Graef, G. and Schaefer, C. (2001). Application of ART2 Networks and Self-Organizing Maps to Collaborative Filtering. In *Revised Papers from the international Workshops OHS-7, SC-3, and AH-3 on Hypermedia: Openness, Structural Awareness, and Adaptivity*, pages 296–309.
- [Grossberg, 1976] Grossberg, S. (1976). Adaptive Pattern Classification and Universal Recoding, I: Parallel Development and Coding of Neural Feature detectors. *Biological Cybernetics*, 23:121–134.
- [Haykin, 1994] Haykin, S. (1994). *Neural Networks : A Comprehensive Foundation*. Macmillan, New York.
- [Herlocker, 2000] Herlocker, J. L. (2000). *Understanding and Improving Automated Collaborative Filtering Systems*. PhD thesis, University of Minnesota. Adviser-Joseph A. Konstan.

- [Herlocker et al., 1999] Herlocker, J. L., Konstan, J. A., Borchers, A., and Riedl, J. (1999). An Algorithmic Framework for Performing Collaborative Filtering. In *SIGIR '99: Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 230–237, New York, NY, USA. ACM Press.
- [Herlocker et al., 2000] Herlocker, J. L., Konstan, J. A., and Riedl, J. (2000). Explaining Collaborative Filtering Recommendations. In *Computer Supported Cooperative Work*, pages 241–250.
- [Hill et al., 1995] Hill, W., Stead, L., Rosenstein, M., and Furnas, G. (1995). Recommending and Evaluating Choices in a Virtual Community of Use. In *CHI '95: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 194–201, New York, NY, USA. ACM Press/Addison-Wesley Publishing Co.
- [Honkela et al., 1996] Honkela, T., Kaski, S., Lagus, K., and Kohonen, T. (1996). Newsgroup Exploration with WEBSOM Method and Browsing Interface. Technical Report A32, Helsinki University of Technology, Laboratory of Computer and Information Science, Espoo, Finland.
- [Jacobi and Benson, 1998] Jacobi, J. and Benson, E. (1998). System and Methods for Collaborative Recommendations. US Patent No. 6,064,980.
- [Jain et al., 1999] Jain, A., Murty, M., and Flynn, P. (1999). Data Clustering: A Review. *ACM Computing Surveys*, 31(3):264–323.
- [Jin and Mobasher, 2003] Jin, X. and Mobasher, B. (2003). Using Semantic Similarity to Enhance Item-Based Collaborative Filtering. In *Proceedings of The 2nd IASTED International Conference on Information and Knowledge Sharing*, Scottsdale, Arizona.
- [Karlgrén, 1990] Karlgrén, J. (1990). An Algebra for Recommendations: Using Reader Data as a Basis for Measuring Document Proximity. Technical Report 179, Swedish Institute of Computer Science (SICS), Stockholm University, Stockholm, Sweden.
- [Karlgrén, 1994] Karlgrén, J. (1994). Newsgroup Clustering Based On User Behavior — A Recommendation Algebra. Technical Report T94:04, Swedish Institute of Computer Science (SICS), Stockholm University, Stockholm, Sweden.
- [Karypis, 2001] Karypis, G. (2001). Evaluation of Item-Based Top-N Recommendation Algorithms. In *CIKM '01: Proceedings of the tenth international conference on Information and knowledge management*, pages 247–254, New York, NY, USA. ACM Press.
- [Kaski, 1997] Kaski, S. (1997). *Data Exploration Using Self-Organizing Maps*. PhD thesis, Helsinki University of Technology, Finland. Acta Polytechnica Scandinavica, Mathematics, Computing and Management in Engineering Series No. 82.
- [Kawamoto, 2004] Kawamoto, D. (2004). Cendant Publishing has filed a lawsuit against Amazon.com, alleging patent infringement. *CNet.com*.
- [Kohonen, 1982] Kohonen, T. (1982). Self-Organizing Formation of Topologically Correct Feature Maps. *Biological Cybernetics*, 43:59–69.
- [Kohonen, 1988] Kohonen, T. (1988). The "Neural" Phonetic Typewriter. *IEEE Computer*, 21(3):11–22.
- [Kohonen, 1990] Kohonen, T. (1990). The Self-Organizing Map. In *Proceedings of the IEEE*, volume 78, pages 1464–1480.
- [Kohonen, 1993] Kohonen, T. (1993). Things You Haven't Heard About the Self-Organizing Map. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 3, pages 1147–1156.
- [Kohonen, 2001] Kohonen, T. (2001). *Self-Organizing Maps*. Springer, Berlin, third edition.
- [Kohonen and Hari, 1999] Kohonen, T. and Hari, R. (1999). Where the Abstract Feature Maps of the Brain Might Come From. *Trends in Neurosciences*, 22(3):135–139.

- [Kohonen et al., 2000] Kohonen, T., Kaski, S., Lagus, K., Salojärvi, J., Honkela, J., Paatero, V., and Saarela, A. (2000). Self-Organization of a Massive Document Collection. *IEEE Transactions on Neural Networks*, 11:574–585.
- [Lagus, 2000] Lagus, K. (2000). *Text Mining with the WEBSOM*. PhD thesis, Helsinki University of Technology, Finland. Acta Polytechnica Scandinavica, Mathematics and Computing Series no. 110.
- [Lam and Riedl, 2004] Lam, S. K. and Riedl, J. (2004). Shilling Recommender Systems for Fun and Profit. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 393–402, New York, NY, USA. ACM Press.
- [Lee et al., 2002] Lee, M., Choi, P., and Woo, Y. (2002). A Hybrid Recommender System Combining Collaborative Filtering with Neural Network. In *AH '02: Proceedings of the Second International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems*, pages 531–534, London, UK. Springer-Verlag.
- [Linden et al., 2003] Linden, G., Smith, B., and York, J. (2003). Amazon.com Recommendations: Item-To-Item Collaborative Filtering. *Internet Computing, IEEE*, 7(1):76–80.
- [Luhn, 1958a] Luhn, H. P. (1958a). The Automatic Creation of Literature Abstracts. *IBM Journal of Research Development*, 2(2):159–165.
- [Luhn, 1958b] Luhn, H. P. (1958b). A Business Intelligence System. *IBM Journal of Research Development*, 2(4):314–319.
- [Lyman et al., 2003] Lyman, P., Varian, H. R., Charles, P., Good, N., Jordan, L. L., and Swearingen, K. (2003). How Much Information? 2003. Technical report, UC Berkeley’s School of Information Management and Systems. <http://www2.sims.berkeley.edu/research/projects/how-much-info-2003>.
- [Malone et al., 1987] Malone, T. W., Grant, K. R., Turbak, F. A., Brobst, S. A., and Cohen, M. D. (1987). Intelligent Information-Sharing Systems. *Communications of the ACM*, 30(5):390–402.
- [Massa and Avesani, 2004] Massa, P. and Avesani, P. (2004). Trust-Aware Collaborative Filtering for Recommender Systems. In *Proceedings of Federated International Conference On The Move to Meaningful Internet: CoopIS, DOA, ODBASE*, pages 492–508.
- [McLaughlin and Herlocker, 2004] McLaughlin, M. R. and Herlocker, J. L. (2004). A Collaborative Filtering Algorithm and Evaluation Metric that Accurately Model the User Experience. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 329–336, New York, NY, USA. ACM Press.
- [Melville et al., 2002] Melville, P., Mooney, R. J., and Nagarajan, R. (2002). Content-boosted collaborative filtering for improved recommendations. In *Proceedings of the 18th National Conference on Artificial Intelligence*, pages 187–192, Menlo Park, CA, USA. American Association for Artificial Intelligence.
- [Montaner et al., 2003] Montaner, M., López, B., and De La Rosa, J. L. (2003). A Taxonomy of Recommender Agents on the Internet. *Artificial Intelligence Review*, 19(4):285–330.
- [Novak et al., 2003] Novak, J., Wurst, M., Fleischmann, M., and Strauss, W. (2003). Discovering, Visualizing, and Sharing Knowledge through Personalized Learning Knowledge Maps. In *Agent Mediated Knowledge Management, International Symposium AMKM*, pages 213–228.
- [Oard, 1997] Oard, D. W. (1997). The State of the Art in Text Filtering. *User Modeling and User-Adapted Interaction*, 7(3):141–178.
- [O’Donovan and Smyth, 2005] O’Donovan, J. and Smyth, B. (2005). Trust in Recommender Systems. In *IUI '05: Proceedings of the 10th international conference on Intelligent user interfaces*, pages 167–174, New York, NY, USA. ACM Press.
- [Pampalk, 2001] Pampalk, E. (2001). Islands of Music - Analysis, Organization, and Visualization of Music Archives. Master’s thesis, Vienna University of Technology, Austria.

- [Pazzani, 1999] Pazzani, M. J. (1999). A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligence Review*, 13(5-6):393–408.
- [Pazzani et al., 1996] Pazzani, M. J., Muramatsu, J., and Billsus, D. (1996). Syskill & Webert: Identifying Interesting Web Sites. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 54–61.
- [Resnick et al., 1994] Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., and Riedl, J. (1994). Grouplens: an Open Architecture for Collaborative Filtering of Netnews. In *CSCW '94: Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186, New York, NY, USA. ACM Press.
- [Resnick and Varian, 1997] Resnick, P. and Varian, H. (1997). Recommender Systems. *Communications of the ACM*, 40(3):56–58.
- [Riedl and Dourish, 2005] Riedl, J. and Dourish, P. (2005). Introduction to the special section on recommender systems. *ACM Transactions on Computer-Human Interaction*, 12(3):371–373.
- [Roh et al., 2003] Roh, T. H., Oh, K. J., and Han, I. (2003). A Cluster-Indexing CBR Model for Collaborative Filtering Recommendation. In *Proceedings of the 7th Pacific Asia Conference on information Systems*, pages 150–167.
- [Salton, 1971] Salton, G. (1971). *The SMART Retrieval System: Experiments in Automatic Document Processing*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.
- [Salton and Buckley, 1988] Salton, G. and Buckley, C. (1988). Term-Weighting Approaches in Automatic Text Retrieval. *Information Processing and Management*, 24(5):513–523.
- [Salton and McGill, 1983] Salton, G. and McGill, M. (1983). *Introduction to Modern Information Retrieval*. McGraw-Hill Education.
- [Sammon Jr., 1969] Sammon Jr., J. W. (1969). A Nonlinear Mapping for Data Structure Analysis. *IEEE Transactions on Computers*, C-18(5):401–409.
- [Sant, 2005] Sant, P. M. (2005). optimization problem. in *Dictionary of Algorithms and Data Structures* [online], Paul E. Black, ed., U.S. National Institute of Standards and Technology. (accessed 17 August 2006) Available from: <http://www.nist.gov/dads/HTML/optimization.html>.
- [Sarwar et al., 2000a] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2000a). Analysis of Recommendation Algorithms for E-Commerce. In *EC '00: Proceedings of the 2nd ACM conference on Electronic commerce*, pages 158–167, New York, NY, USA. ACM Press.
- [Sarwar et al., 2000b] Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2000b). Application of Dimensionality Reduction in Recommender Systems—A Case Study. In *Proceedings of the WebKDD 2000 Workshop at the ACM-SIGKDD Conference on Knowledge Discovery in Databases (KDD'00)*.
- [Sarwar et al., 2001] Sarwar, B. M., Karypis, G., Konstan, J. A., and Reidl, J. (2001). Item-Based Collaborative Filtering Recommendation Algorithms. In *Proceedings of the 10th international WWW Conference*, pages 285–295.
- [Sarwar et al., 1998] Sarwar, B. M., Konstan, J. A., Borchers, A., Herlocker, J. L., Miller, B. N., and Riedl, J. (1998). Using Filtering Agents to Improve Prediction Quality in the Grouplens Research Collaborative Filtering System. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, pages 345–354.
- [Schafer et al., 2001] Schafer, J. B., Konstan, J. A., and Riedl, J. (2001). E-Commerce Recommendation Applications. *Data Mining and Knowledge Discovery*, 5(1/2):115–153.
- [Shani et al., 2002] Shani, G., Brafman, R., and Heckerman, D. (2002). An MDP-Based Recommender System. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 453–460, San Francisco, CA. Morgan Kaufmann.

- [Shardanand and Maes, 1995] Shardanand, U. and Maes, P. (1995). Social Information Filtering: Algorithms for Automating “Word of Mouth”. In *Proceedings of ACM CHI’95 Conference on Human Factors in Computing Systems*, volume 1, pages 210–217.
- [Silaghi, 2004] Silaghi, G. C. (2004). Bi-Dimensional Visualization of Target Categories for a Text Categorization Task. In *IEEE Conference on Advances in Intelligent Systems: Theory and Applications, AISTA 2004*.
- [Singhal, 2001] Singhal, A. (2001). Modern Information Retrieval: A Brief Overview. *IEEE Data Engineering Bulletin*, 24(4):35–43.
- [Sinha and Swearingen, 2002] Sinha, R. and Swearingen, K. (2002). The Role of Transparency in Recommender Systems. In *CHI ’02: CHI ’02 extended abstracts on Human factors in computing systems*, pages 830–831, New York, NY, USA. ACM Press.
- [Sirosh and Miikkulainen, 1993] Sirosh, J. and Miikkulainen, R. (1993). How Lateral Interaction Develops in a Self-Organizing Feature Map. In *Proceedings of ICNN’93 International Conference on Neural Networks*, volume III, pages 1360–1365, Piscataway, NJ. IEEE Service Center.
- [Stack, 1997] Stack, C. (1997). System and Method for Providing Recommendation of Goods or Services Based on Recorded Purchasing History. US Patent No. 6,782,370.
- [Swearingen and Sinha, 2001] Swearingen, K. and Sinha, R. (2001). Beyond Algorithms: An HCI Perspective on Recommender Systems. *ACM SIGIR 2001 Workshop on Recommender Systems*.
- [Swearingen and Sinha, 2002] Swearingen, K. and Sinha, R. (2002). Interaction Design for Recommender Systems. *Designing Interactive Systems*.
- [T.Samad and S.A.Harp, 1992] T.Samad and S.A.Harp (1992). Self-Organization with Partial Data. *Network: Computation in Neural Systems*, 3:205–212.
- [Van Rijsbergen, 1979] Van Rijsbergen, C. J. (1979). *Information Retrieval, 2nd edition*. Department of Computer Science, University of Glasgow, 2 edition.
- [Vembu and Baumann, 2004] Vembu, S. and Baumann, S. (2004). A Self-Organizing Map Based Knowledge Discovery for Music Recommendation Systems. In *Proceedings of the 2nd International Symposium on Computer Music Modeling and Retrieval*, pages 119–129.
- [Venna and Kaski, 2001] Venna, J. and Kaski, S. (2001). Neighborhood Preservation in Nonlinear Projection Methods: An Experimental Study. In *ICANN ’01: Proceedings of the International Conference on Artificial Neural Networks*, pages 485–491, London, UK. Springer-Verlag.
- [Vesanto, 1999] Vesanto, J. (1999). SOM-Based Data Visualization Methods. *Intelligent Data Analysis*, 3(2):111–126.
- [Vesanto and Alhoniemi, 2000] Vesanto, J. and Alhoniemi, E. (2000). Clustering of the Self-Organizing Map. *IEEE Transactions on Neural Networks*, 11(3):586–600.
- [Vesanto et al., 2000] Vesanto, J., Himberg, J., Alhoniemi, E., and Parhankangas, J. (2000). SOM Toolbox for Matlab. Technical Report A57, Helsinki University of Technology.
- [von der Malsburg, 1973] von der Malsburg, C. (1973). Self-Organization of Orientation-Sensitive Cells in the Striate Cortex. *Kybernetik*, 15:85–100.
- [Widrow and Lehr, 2003] Widrow, B. and Lehr, M. A. (2003). Perceptrons, Adalines, and Backpropagation. In Arbib, M., editor, *The handbook of brain theory and neural networks*, pages 871–877. MIT Press, Cambridge, MA, USA, 2 edition.
- [Wilkes and Wade, 1997] Wilkes, A. and Wade, N. (1997). Bain on Neural Networks. *Brain and Cognition*, 33(3):295–305.
- [Willshaw and von der Malsburg, 1976] Willshaw, D. J. and von der Malsburg, C. (1976). How Patterned Neural Connections can be set up by Self-Organization. *Proceedings of the Royal Society of London Series B*, 194:431–445.