Algorithms and Representations for Personalised Information Access

Rickard Cöster

A Dissertation submitted to Stockholm University
in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

2005

Stockholm University /
Royal Institute of Technology
Department of Computer and System Sciences

Stockholm, Sweden

ISBN Nr 91-7155-030-5
ISSN 1101-8526
ISRN SU-KTH/DSV/R–05/7–SE
DSV Report series No. 05-007

Swedish Institute of Computer Science

Userware Laboratory

Kista, Sweden

ISSN 1101-1335
ISRN SICS-D–36–SE
SICS Dissertation Series 36

Doctoral Dissertation
Department of Computer and System Sciences
Stockholm University / Royal Institute of Technology

ISBN Nr 91-7155-030-5
This thesis was typeset by the author in the Charter and Euler fonts using LaTeX
Printed by Akademitryck AB, Edsbruk, Sweden, 2005.

Abstract

Personalised information access systems use historical feedback data, such as implicit and explicit ratings for textual documents and other items, to better locate the right or relevant information for individual users.

Three topics in personalised information access are addressed: learning from relevance feedback and document categorisation by the use of concept-based text representations, the need for scalable and accurate algorithms for collaborative filtering, and the integration of textual and collaborative information access.

Two concept-based representations are investigated that both map a sparse high-dimensional term space to a dense concept space. For learning from relevance feedback, it is found that the representation combined with the proposed learning algorithm can improve the results of novel queries, when queries are more elaborate than a few terms. For document categorisation, the representation is found useful as a complement to a traditional word-based one.

For collaborative filtering, two algorithms are proposed: the first for the case where there are a large number of users and items, and the second for use in a mobile device. It is demonstrated that memory-based collaborative filtering can be more efficiently implemented using inverted files, with equal or better accuracy, and that there is little reason to use the traditional in-memory vector approach when the data is sparse. An empirical evaluation of the algorithm for collaborative filtering on mobile devices show that it can generate accurate predictions at a high speed using a small amount of resources.

For integration, a system architecture is proposed where various combinations of content-based and collaborative filtering can be implemented. The architecture is general in the sense that it provides an abstract representation of documents and user profiles, and provides a mechanism for incorporating new retrieval and filtering algorithms at any time.

In conclusion this thesis demonstrates that information access systems can be personalised using scalable and accurate algorithms and representations for the increased benefit of the user.

Acknowledgements

During the course of these studies, I have been fortunate to work with inspiring, generous and fascinating people.

I thank my supervisor Lars Asker for giving me the opportunity to embark on this journey and for supporting me and believing in me and my work.

Many thanks go to my friend and colleague Martin Svensson – endless support, bright ideas, and a great deal of fun!

My brother Joakim has inspired and helped me for as long as I can remember, and was the person who inspired me to do research at all. I am glad that you have been here for me all the time. Thank you for everything!

A big thank you goes to Jussi Karlgren and Magnus Sahlgren for the thought-bending discussions on information retrieval, but most of all for all the fun.

I am very grateful to Kia Höök and Martin Svensson who convinced me that SICS is a great place to do research – and they were absolutely right!

Many thanks to Björn Gambäck, Magnus Boman, Jussi Karlgren, Martin Svensson, Henrik Boström and Kia Höök for giving me comments on various draft versions, and for the good suggestions for improvements. A special thank you goes to Björn for both very detailed comments and suggestions for how I could improve the structure of the thesis. I thank Åsa Rudström and Fredrik Olsson for comments on various parts of this work in early stages.

A special thank you to Kia and Åsa for a great time in the MobiTip project – it has been very inspiring and great fun!

Many thanks also to all members of the previous HUMLE lab for a great time!

Finally, I thank my friends and family, for the things that are more important than anything else: Jenny, Jocke, Martin, David, Thomas, Raymond, Petra and Mathias.

Thank you all!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is about computer algorithms that help people find, in a vast information space, the right or relevant information. More specifically, the algorithms considered here are such that they take advantage of user preferences and actions regarding information, so that the algorithms can learn from, and present better and more relevant information to the user.

Several different aspects of information retrieval algorithms that learn from experience are investigated. In the first part of this thesis it is proposed how document retrieval and document categorisation can be improved by learning methods that represent documents and queries in a conceptual or semantic feature space. This representation is motivated by the difficulty of measuring conceptual similarities between objects with traditional word based representations, and the need for a compact representation for the learning algorithms. One of the proposed algorithms automatically improves on queries based on previous feedback from the users. Different learning algorithms for the two tasks document retrieval and document categorisation are then experimentally evaluated using real data captured from several sources, and it is demonstrated that the new algorithms and representations improve on their tasks.

In the second part, new algorithms for collaborative filtering are proposed and evaluated. Collaborative filtering is a technique that helps us find information that is not necessarily represented as text: the technique instead relies on users' ratings and opinions to find the right or relevant information. The proposed algorithms are motivated by the need for methods that are scalable and have the ability to learn in an incremental fashion. An extensive evaluation of the algorithms is provided, again using real data, and it is shown that they produce good results, in terms of generating predictions at a high speed, with low error rates.

In the third part, a system architecture is described that enables possible combinations of information retrieval and filtering using both textual and collabora-

tive information.

This introductory chapter next gives a brief history of the growth of information. An overview of the techniques that exist for organising and searching information is given in Section 1.1.2. The research goals put forth in the thesis are discussed in Section 1.3.

1.1 The Information Age

In a study conducted in 2003 (Lyman et al., 2003), it was estimated that the amount of new information (stored on paper, film, magnetic, and optical media) increased by about 30% each year during the years 1999–2002.

The number of new unique book titles produced during 2003 was about $950,000$, the number of newspaper publications $25,276$ (2001), the number of scholarly periodicals was $37,609$ publications (2001) and the number of archivable, original office documents was estimated to about $1075 * 10^7$ pages.

The estimated amount of digital information on the surface web (web pages not dynamically generated from databases) in January 2003 was about 170 TB, or $170 * 10^{12}$ bytes. At that point, the surface web was about seventeen times larger than the entire print collections of the U.S. Library of Congress. The estimate of the deep web at that time was that it contained $91,850$ TB of information.

This research group also looked at communication flows, and the amount of communicated new information through the four primary communication channels: radio, television, telephone calls, and the Internet. The amount of electronic flow of new information through the Internet was estimated to be approximately $532,897$ TB, whereas the same figure for telephone calls was a striking $17,300,000$ TB.

To manage these huge amounts of information, we need methods that help us store, organise and search it. In a very general sense, this is the motivation for all research in information retrieval and filtering, and thus also the motivation for the techniques developed here.

From a computer science perspective, algorithms for information retrieval and filtering have been a well established part of the research field for about half a century, but the general need for techniques for organising and searching for information is far much older.

1.1.1 A Brief History

Perhaps the most well known historical example of a huge repository of information is the Library of Alexandria. The library was created around 300 B.C., and contained a great number of papyrus scrolls. The exact number of scrolls is

not known: some sources claim the library contained at its peak around 700.000 scrolls (Witten & Frank, 2000), whereas other sources give much smaller figures. The library contained works written by the greatest thinkers of the ancient world: Socrates, Plato and many others. The major part of the library is believed to have been destroyed around 30–50 B.C., and the last remaining books to have been destroyed a few hundred years later.

The mechanised printing press – invented 1041 in China, modified and introduced in the West around 1450 – was a great revolution for storing and accessing information. The press paved way for many libraries and for the accessibility of books to the general public.

The first scientific journals appeared in France and England around 1665, and such journals are since then the main forum for communicating new scientific knowledge.

Creative thinkers soon realised that since information will continue to grow, one of the great challenges is to create tools that help people reach information in an instant. At the end of World War II, Vannevar Bush, the director of the Office of Scientific Research and Development in the U. S., wrote a highly influential article (Bush, 1945) on this very subject. He advocated that scientists should now focus on the immense task of storing and making accessible human produced information. In his vision, there would be tools to reach any information in an instant.

Shortly after the war, the first general-purpose programmable digital computer was constructed. In the decades that followed, the technologies advanced up to the era of the modern computer. The invention of the microprocessor started the development of low cost general purpose computers, and at about the same time the first local area network was also invented, which allowed computers to communicate with each other.

Today, computers come cheap and the computer is an essential tool for both work and play. The interconnection of computers through the Internet has made it possible for people to meet and exchange information and thoughts across borders of time, geography, and, in some aspects, language and culture. The amount of information is constantly growing, produced rapidly, and often stored in digital form. Publicly available information on the Internet grows at a tremendous rate, and web search engines provide us with a way of sifting through this vast information space.

The devices we now use to view and manage digital information are no longer confined to stationary computers or laptop portables. Mobile phones, personal digital assistants and wearable computing devices have larger memory and more processor power than stationary computers had a decade ago. The computer networks are also moving out in the open: cables and wires are complemented with wireless radio networks capable of transmitting data at an ever increasing rate.

Portable devices with wireless communication can reach tremendous amount of information, and in this network of devices and information flows, the need to find the right or relevant information is ever more accentuated.

Going back to the library of Alexandria: it was recently announced that the world's currently largest web search engine, Google[1], is working on a project to digitise and make searchable library material from some of the world's most important and complete libraries. The vision of the entire world's information within reach is slowly becoming less science fiction and more of a reality.

1.1.2 Information Retrieval and Collaborative Filtering

Information Retrieval (IR) is the science of creating systems that assist users in finding the information they need (see for example the introductory books Rijsbergen (1979); Baeza-Yates and Ribeiro-Neto (1999)).

A typical example of an IR system is a web search engine, such as Google or MSN Search[2]. Users can type a query that describes what they are interested in finding, and the system scans its database to find the pages it believes best match the query. The results are often presented as a list of pages, ranked according to how well each page matched the query.

Matching a query to a document is done by matching the textual contents of the query with the textual contents of the documents: if they are similar then the document is ranked high for the query. For this reason, methods that match elements (queries, documents, etc.) on the basis of their content such as the text will be called *content-based* methods.

Text searching and content-based methods in general, is an effective way of finding information. It is easy to type a few keywords, or construct a more elaborate query, and in many cases this is sufficient for finding the desired information.

Still, finding the right information may not always be easy, or even possible, by textual search. Consider for example that you wish to find food recipes that are liked by people who like the same food recipes that you like. This is a fundamentally different information need: we are looking for recipes (information) but do not particularly aim to use the content (the ingredients, the author, etc.) to locate the recipes. Instead, we wish to find recipes based on collaborative information: our own recipe preferences, and the preferences of others. This preference matching process can be made regardless of whether the users are similar or not in other ways: they may live thousands of kilometres apart, be of different age and have different background, but still prefer the same food.

The technique outlined in the example can be generalised to other domains

[1] `www.google.com`
[2] `search.msn.com`

as well: it has been already been applied to movies, books, travel destinations, music, and on-line products in general. In fact, it should be possible to apply this to every domain where items are judged differently by different people. This way of retrieving information is called *Collaborative Filtering* (Goldberg, Nichols, Oki, & Terry, 1992; Breese, Heckerman, & Kadie, 1998), or a *Recommender system* (Karlgren, 1990; Resnick & Varian, 1997), since information is retrieved on the basis of recommendations from others.

For our purposes, the term Recommender system is too broad since it encompasses virtually any technique that produces a recommendation. Collaborative Filtering is a more able term since this clearly defines that predictions are based on collaborative techniques.

These two different methods for finding information (content-based and collaborative) are explored, improved and combined in this thesis. A suitable designation for both content-based and collaborative retrieval and filtering is *Information Access*.

1.2 Motivation

The primary inspirational source for this work is that people continuously share and locate new information through the help of others. People are parts of groups and networks consisting of people that share similar interests, help each other search for information, etc. There is much to be gained by assisting each other in finding the right or relevant information, even if we assist each other anonymously.

The work presented in this thesis is part of a general effort to improve the performance of information retrieval and filtering systems by adding a layer of collaborative information on top of, or perhaps beside, the layer of information content. The layer of collaborative information contains the user's actions on the information, such as the user's queries, feedback information, implicit and explicit ratings, paths taken when browsing, bookmarks, etc. By using this collaborative layer, it is hypothesised that the general performance of information retrieval and filtering systems can be improved.

Taking advantage of collaborative information is one of many paths to tread: it is easy to list others that are of no less significance:

- Natural language techniques, for improved understanding of the queries and documents (Strzalkowski, 1999).

- Cross-language retrieval, where users can pose a query in one language and retrieve results (possibly translated) from another language (Savoy, 2003).

- User interface design, to guide users in formulating queries and browsing results, and viewing information from other perspectives (Hearst, 1999).
- Fundamentally new models of computation, such as quantum computing, where we might find a new and perhaps more convenient language for describing the processes and objects in information retrieval, such as that outlined by (Rijsbergen, 2004).

1.3 Research Questions and Goals

There is an abundance of techniques for information access. The main focus of this thesis is to investigate ways in which user feedback, exemplified by relevance information and user ratings, can be used as means for improving information access for the user. This will be referred to as *personalised information access*. The thesis contains an exploration of a few closely related topics in personalised information access.

The first topic is how feedback from users can improve results in information retrieval in a long-term fashion. In line with this, the impact of combining concept-based text representations with traditional word-based ones is also explored. The second topic is how Collaborative Filtering can be performed in an environment where users and items are plentiful: for this purpose scalable retrieval methods from Information Retrieval are used. In line with the issue of scalability, an incremental collaborative filtering algorithm suited for mobile devices is proposed. The third topic is how to combine efforts in order to find the right or relevant information. A general architecture for an information access system that encompasses the above techniques is developed and described. More specifically, the following questions are investigated:

1. **Long-term learning from feedback in information retrieval**. It is well known that expanding a query with relevance feedback information can enhance the effectiveness of the query. Usually, the user feedback information is used only for the current query, and therefore lost when the user embarks on a new query. The first question is how a system can learn from this user feedback in a long-term fashion. In pursuit of an answer to this question, a concept-based representation of documents and queries using Latent Semantic Indexing (LSI) is used, to capture query–document similarity in a broader sense than that accomplished by mere word-level matching.
2. **Concept-based representation of text documents**. A simple and common representation of text documents is to put the words in an unordered

structure, a so-called Bag-of-Words, but this has the disadvantage of not identifying relationships between terms, such as synonyms, or words that are related by the context in which they appear. There have been quite a few attempts to remedy this problem: one such method, Random Indexing, is investigated, for the purpose of categorising text documents. The question here is whether this representation can capture properties of texts that can be combined with the traditional representation for increased performance.

3. **Inverted files for Collaborative Filtering**. Text-based retrieval and collaborative filtering share a number of characteristics, and can also be used as complementary methods for finding and discovering information. The past 50 years of research in text-based retrieval has produced a number of techniques that might be worthwhile to transfer to the relatively new domain of collaborative filtering. In particular, text-based retrieval systems have an efficient representation and storage model for documents and terms using inverted files. The question is thus whether this model is suitable for collaborative filtering as well.

4. **Incremental Collaborative Filtering**. Portable devices such as digital assistants and mobile phones are becoming more integrated in our everyday activities. In contrast to a stationary computer, which may very well be constantly connected to other computers or the Internet, a portable device is not always this well connected. If it would be possible to move the centralised computational model of collaborative filtering to a partly decentralised model, it would remedy the problem of not always being connected. Another related problem is that mobile devices are not as powerful as stationary ones: battery time is generally limited and the amount of memory smaller. To meet these challenges an incremental algorithm is proposed, where predictions can be updated directly on the device.

5. **Integration**. The fundamental purpose of all algorithms and representations presented and evaluated in this thesis, is to improve on, in some manner, the task of finding the right and relevant information. There are two broad categories of methods that help us find information: those that are based on the content or textual appearance of the information, and those that are based on collaboratively filtering information on the basis of ratings. An architecture and implementation of a system that is suitable for both methods, as well as combinations, is presented.

1.4 Contributions

The main research contributions have previously been published in five papers: four conference papers and one technical report. Still, the material in the thesis differ somewhat from the papers in a number of ways: the most notable is that new and more complete evaluations have been performed for some of the algorithms.

The material for Chapter 3, on learning from relevance feedback in the Latent Semantic Indexing model, has been published in (Cöster & Asker, 2000), with Cöster as the primary author and investigator.

Chapter 4 has previously been published in (Sahlgren & Cöster, 2004), with Sahlgren as the primary author. Sahlgren is the primary investigator of the Random Indexing approach for representing text, while my involvement primarily concerns the machine learning and evaluation parts.

The material for Chapters 5 and 6, on Collaborative Filtering, are published in (Cöster & Svensson, 2002) and (Cöster & Svensson, 2005), with Cöster as the primary author and investigator.

Chapter 7 is published as a technical report in (Cöster, 2002a) and also in (Cöster, 2002b).

1.5 Outline

In the next chapter necessary background is given for the algorithms and representations used and refined throughout the thesis, as well as material on performance evaluation. Chapters 3–7 contain the research contributions in the order discussed in the previous sections.

The thesis concludes with reflections upon the presented work, the conclusions that can be drawn from the experiments, highlights of the properties of the algorithms and representations, as well as a discussion of future work.

Chapter 2

Foundations

In this chapter, necessary background reading for the rest of the thesis is given. The material here can perhaps be skipped at a first reading by the informed or impatient reader; there will be references to relevant parts of this chapter throughout the thesis. Nonetheless, the material here also serves as motivation for the algorithms and representations developed and used, as properties of current ones are reviewed and commented.

The first section is about Information Retrieval, and provides background for chapters 3, 4 and 7. Chapters 5, 6 and also Chapter 7 will refer to material in Section 2.4 about Collaborative Filtering. Throughout the thesis machine learning techniques are used: the relevant material for machine learning is covered in Section 2.5. The last section concerns experimental evaluation, and provides some details about the data sets used in the experiments.

2.1 Information Retrieval

Written text is the primary way that human knowledge is stored, and next to speech, the primary way it is transmitted. In the context of information retrieval systems, the word 'information' is often replaced by 'document' (Rijsbergen, 1979). This is not surprising, since textual carriers such as documents, are one of the most important information sources that exist.

The material in this section provides n overview of the some of the dominant models for text retrieval, how they are implemented and somewhat about their usage. The focus is on the Vector Space and Latent Semantic Indexing models, since they form an important background reading for Chapters 3 and 4.

2.1.1 Introduction

A text retrieval system is composed of a database of text documents, and a mechanism for querying this database. From a user's point of view, such a query should retrieve documents that contain information needed by the user.

A user communicates with the retrieval system by posing a query, expressed in a query language. A simple query language lets the user specify a query as a text string: the string is parsed, and the system tries to find documents that fulfil the information need expressed in the query.

The result of a user query is often displayed as a ranked list of documents, where the underlying *retrieval model* determines the ranking. The three dominant models for full-text retrieval are the Vector Space, Latent Semantic Indexing and Probabilistic models. The models define representations for terms, documents, queries, and provide a definition of the retrieval function.

The first step for a text retrieval system is to *index* the text to be searched: to extract meaningful terms and phrases contained in the documents, and place these *index terms* in an (alphabetically ordered) index. A number of pre-processing steps are usually applied to the text before extracting index terms:

- Lexical analysis to treat digits, hyphens and other special characters.
- Removal of words with low discriminating value for retrieval, so called stop words.
- Morphological analysis by for example stemming, for reducing variations of words by normalising them all to a root stem.

After being indexed, the text is ready for retrieval. When the user poses a query, the text processing steps are applied to the query, and the index is consulted to locate the documents that contain the user's query terms.

Text retrieval is concerned with retrieving documents in many different formats and varying structure: web pages, news articles, books, legal documents, email, etc. Furthermore, there are several thousands languages in the world: still, most text retrieval systems are specialised for some major European and East-Asian languages, such as English and Chinese.

Regardless of the format of the text, the fundamental components of any IR system are index terms, documents, queries and weights. A *term* is an individual word or a phrase. Terms are extracted from either the body of a text or a surrogate text (such as the abstract), which will be called a *document*. A *weight* is a value reflecting the importance of a term in a document, or a query. A user's statement of her information need is called a *query*. A query is usually represented by a set of terms, but may also contain operators of various kinds, such as Boolean and adjacency operators.

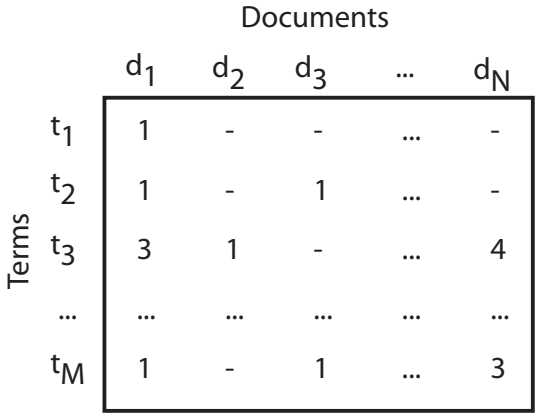

Figure 2.1: Example of a term-document matrix. A non-zero cell $[i, j]$ means that the term i is contained in document j.

2.1.2 The Vector Space Model

In the Vector Space Model (Salton & McGill, 1983), VSM, documents and queries are represented as weighted vectors in a term space. Each term in the collection of documents thus corresponds to a dimension in the vector space. In this space, a document is represented by a sparse vector that has non-zero values for dimensions that correspond to terms contained in the document. A query is represented in the same manner: a vector that contains non-zero values for terms contained in the query. The query vector is matched against each document vector in the collection, using some vector similarity measure. For each document, the matching process returns a score that reflects the similarity between the query and the document. The result is a set of documents, ranked in decreasing order of similarity scores.

This kind of representation is also called 'Bag-of-Words', since the terms are taken from the documents and placed in an unordered structure that does not retain term relations (such as nearby terms).

Figure 2.1 is an example of such a sparse matrix of terms and documents: a non-zero cell $[i, j]$ in this term-document matrix means that the term i is contained in document j. In the Figure, the term weight is set to the frequency of the term within the document; the number of times it occur.

Document Term Weights

In VSM and in other retrieval models as well, terms are weighted according to two observations: the frequency of the term in the document, and the frequency of the term in the collection of documents as a whole.

The first observation is to note that if a term occurs often in a document, it is probably more important to the document than a term that occurs only once. This observation is not without pitfalls: if the term is very common in the collection of all documents, it may be useless to distinguish two documents from each other.

An approximation of the distribution of terms in text documents is Zipf's law (Zipf, 1949). It captures the observation that a few terms occur very frequently, a medium number of terms occur with medium frequency and many terms occur very infrequently. Terms that occur very frequent are seldom useful, and therefore often removed from the document collection, whereas terms that occurs with medium frequency often are the most useful ones.

An effective approach is therefore to combine the local and global frequency statistics: this family of weighting methods are called *tf-idf* weighting. The weight $w_{i,j}$ for term i in document j may for example be defined as

$$w_{i,j} = \mathrm{tf}_{i,j} \log\left(\frac{N}{\mathrm{df}_i}\right). \tag{2.1}$$

In Formula 2.1, the term frequency $\mathrm{tf}_{i,j}$ is the number of occurrences of term i in document j, and the document frequency df_i is the number of documents in which term i occur. The total number of documents in the collection is N. The tf-idf weighting scheme is thus a product of the term frequency (tf) and inverse document frequency (idf).

Indexing and Retrieval

As discussed, indexing a set of documents means to extract terms and phrases from the text, and place these in a data structure that enables efficient retrieval of documents on the basis of a query. An observation that is crucial for the design of this structure is that queries tend to be short in comparison to the total number of terms in the collection. The access to the index is therefore via the terms, which are stored in a dictionary for fast lookup.

Each term in the dictionary points to a posting list (also known as an inverted list), containing a list of document number and term frequency pairs. If phrase and adjacency queries are supported, the term's document positions are also included in the list. As an example of the index and inverted list, Figure 2.2

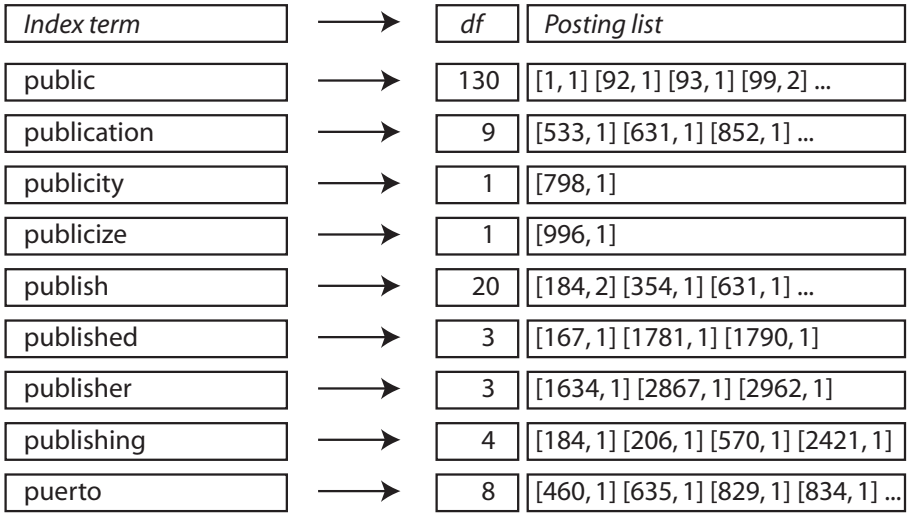

Figure 2.2: Subset of the term index and corresponding inverted lists for a news article collection. Each index term points to the document frequency and inverted list for that term. The inverted list contains document number and term frequency pairs

displays a subset of the index and the posting lists for a collection of news articles.

For retrieving documents, the *cosine* of the angle between the query and document vectors is often used as a measure of similarity. From linear algebra we know that for two vectors $\mathbf{x}, \mathbf{y} \in R^n$

$$\cos(\mathbf{x}, \mathbf{y}) \quad = \quad \frac{\langle \mathbf{x} \cdot \mathbf{y} \rangle}{|\mathbf{x}||\mathbf{y}|} \tag{2.2}$$

where $\langle \mathbf{x} \cdot \mathbf{y} \rangle$ is the inner product between two vectors

$$\langle \mathbf{x} \cdot \mathbf{y} \rangle = \sum_{i=1}^{n} \mathbf{x}_i \mathbf{y}_i \tag{2.3}$$

and $|\mathbf{x}| = \sqrt{\langle \mathbf{x} \cdot \mathbf{x} \rangle}$ is the Euclidean vector length.

The cosine of the angle yields a number between -1 and 1, where a value close to 1 (-1) means that the vectors point in approximately the same (opposite) direction.

Since the index is accessed through the index terms, the cosine function must be expressed in a form suitable for *inverted* retrieval: there is no direct access to the document vectors, only to the index of terms and their posting lists.

The inverted search algorithm (Witten, Moffat, & Bell, 1999) scans the index for each term in the user's query. For each term, its posting list is fetched, and for each document and term frequency pair in the list a score is accumulated in an in-memory data structure. When all terms have been processed, the accumulated scores are then normalised according to the document lengths, and the accumulators will now contain the cosine values. The accumulator array is then sorted so that the top k documents with highest cosine values are presented as the result of the query.

We will return to the problem of how to express a vector similarity function for inverted retrieval in Chapter 5. In that case, the similarities are calculated between user rating vectors.

2.1.3 Latent Semantic Indexing

In VSM, the term space is very sparse and requires a document to contain exact terms from the query if it is to be retrieved. This problem can be alleviated by several well-known techniques: by representing words by their stems, by providing fuzzy query matching, etc.

A more general problem in text retrieval is that terms are commonly treated as entities with little connection to the language they are written in. In natural language, some words are ambiguous and have different meanings in different contexts, while some words that are different have the same meaning, i.e., are synonymous.

The Latent Semantic Indexing (LSI) (Deerwester, Dumais, Furnas, Landauer, & Harshman, 1990) model attempts to overcome these vocabulary problems by looking for terms that co-occur throughout the document collection. Terms that co-occur are collapsed to a single dimension, resulting in a compressed vector space. Each dimension represents a collection of terms that are associated in some sense.

It is difficult to explicitly state the properties of the words that are associated using this method of observing co-occurrences. The model is fundamentally dependent on the content of the documents, and the parameters of the dimensionality reduction method. Nevertheless, the method does capture words with similar meaning and usage patterns.

The dimensionality reduction is carried out by the matrix decomposition technique Singular Value Decomposition (SVD). SVD is applied to term-document matrix (see Figure 2.1) formed by treating documents as column vectors, terms as row vectors and placing term frequencies in the matrix cells. This matrix is very sparse: the number of non-zero cells is typically very small compared to the total number of cells for general document collections.

The performance of Latent Semantic Indexing as a text retrieval system is evaluated in (Dumais, 1995). As a general purpose model for IR, LSI produces results comparable to keyword matching methods such as VSM. However, LSI has other qualities such as its dense representation of documents and queries, and that documents and queries are not matched strictly at the word-level. We take advantage of these two properties in Chapter 3, where a VSM representation of documents and queries would be inadequate.

The Singular Value Decomposition

The underlying mathematical operation in LSI is Singular Value Decomposition (Berry, Dumais, & Brien, 1995). SVD is a way of approximating a rectangular matrix in the least-squares sense. The result is a much lower-dimensional space in which all relationships in the original matrix can be approximated using the dot product or cosine measure. The matrix subject for SVD in LSI is the term-document matrix (see Figure 2.1) and each dimension in the new space can be thought of as representing common meaning components of many different words and documents (Deerwester et al., 1990). Each term, document, and query is represented as a vector in the low-dimensional space.

SVD decomposes the original matrix into three smaller matrices. The matrix sizes are determined by the rank of the original matrix, i.e., the number of linearly independent row or column vectors.

Two matrices contain the left and right singular vectors of the original matrix. The third contains the non-negative eigenvalues of the matrix formed by multiplying the original matrix with its transpose. The singular vectors and the eigenvalues form the compressed representation of the original matrix. An approximation of the original matrix may be obtained by multiplying the smaller matrices, and this approximation is known to be optimal in the least-square sense.

Given an $(m \times n)$ rectangular matrix X of rank[1] r, where without loss of generality $m \geq n$, the *Singular Value Decomposition* (SVD) of X is defined as

$$X = TSD^T \tag{2.4}$$

The matrices T and D have *orthonormal* columns, meaning that the vectors are of unit length and orthogonal to each other. This means that $T^TT = D^TD = I$. Furthermore, the matrix S is a *diagonal matrix* since it has only values on the diagonal. The values (ϕ_i) of S are the nonnegative square roots of the d eigenvalues of XX^T, such that

$$\phi_1 \geq \phi_2 \ldots \geq \phi_r > \phi_{r+1} = \ldots = \phi_d = 0 \tag{2.5}$$

[1]The *rank* of a matrix is the number of linearly independent columns or rows.

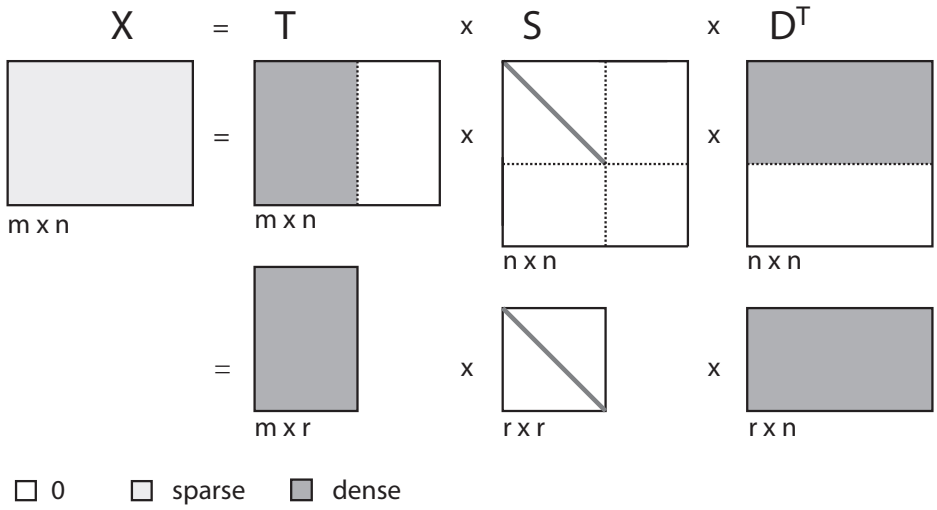

Figure 2.3: Singular Value Decomposition. The original $m \times n$ sparse matrix X is decomposed into the smaller matrices T, S and D. The X matrix can be further reduced by only keeping the $k < r$ largest values in the S matrix, resulting in the Truncated Singular Value Decomposition.

The elements of the diagonal matrix S are called *singular values* (factors) whereas the vectors of the matrices T and D are called the left and right *singular vectors*, respectively. The elements of S can be rearranged in decreasing order of magnitude, since the SVD is unique up to certain row, column and sign permutations. Figure 2.3 displays the Singular Value Decomposition.

Truncated Singular Value Decomposition

The sizes of the matrices T, S and D depend on the rank of the original matrix X. It is, however, possible to approximate the matrix X by a set of smaller matrices T_k, S_k and D_k. If the singular values in S are ordered by size, the first k may be kept and the others set to zero, resulting in the *truncated* matrix X_k, such that

$$X = TSD^T \approx X_k = T_k S_k D_k^T \tag{2.6}$$

Thus, the dimension of the vectors in T and D equals k, the number of factors. It can be shown that the matrix X_k is the best approximation of X in the least-square sense (Berry, Dumais, & Letsche, 1995).

LSI uses this approximation of X for its information retrieval model, since the truncated SVD has a number of advantages over the full SVD. Algorithms for calculating the truncated SVD of large sparse matrices are substantially faster

than the algorithms that compute the full SVD (Berry, Dumais, & Brien, 1995). Furthermore, experiments indicate that retrieval performance is best when only the 100–300 largest singular values are kept (Dumais, 1994).

Term weighting

In the Vector Space Model, a common way to obtain the term weights in the original matrix X is by using Formula 2.1. The same weighting scheme may be used for LSI, although the method used is often Log Entropy (Dumais, 1992):

$$w_{i,j} = \log(tf_{i,j}) \times \left(\sum_{j} \frac{p_{i,j} \log p_{i,j}}{\log N} \right) \tag{2.7}$$

where $p_{i,j} = \frac{tf_{i,j}}{gf_i}$ and gf_i is the total number of times the term i occurs in the whole collection.

More mathematical details of SVD can be found in a number of papers, e.g. (Berry, Dumais, & Brien, 1995; Berry, Dumais, & Letsche, 1995).

Indexing and Retrieval

Latent Semantic Indexing compresses the original vector space to a few hundred dimensions, a space that is no longer sparse. This has the effect that inverted files have less of an advantage in this model. Since the resulting space is dense, it is difficult to improve upon a linear scan of the vectors in order to find the top document matches for a query vector.

2.1.4 Other Retrieval Models

The vector space models VSM and LSI are perhaps the most popular of IR models, but not the only ones. The focus of the thesis is on vector space models, but there are other models that have been very influential for the progress of IR.

The Probabilistic Model (Sparck-Jones, Walker, & Robertson, 2000) is also a standard model for document retrieval. The basic model is based on two parameters: the probability that a document is relevant to a query, and the probability that it is not. These *a priori* probabilities are estimated from the text collection, using term frequency information. Several variants of a probabilistic model have been proposed, and the interested reader should consult the above reference for further reading.

In the *Boolean* model a query is stated as a set of terms connected with Boolean operators. The system evaluates the Boolean expression and returns

the document set which matches the expression. This makes it difficult to rank the output with respect to the query, since a document can only match the expression or not. One possibility however is to use quorum-level search (Salton & McGill, 1983), where the result list is grouped in levels of specificity.

Various techniques to incorporate real-valued weights in the Boolean model have been proposed. Many of these stem from the discipline of fuzzy logic, some from techniques used in other IR models. In (Lee, 1994), various properties of each technique was analysed. The most effective technique was found to be the p-norm model (Salton, 1984) which is a hybrid of the vector space model and a generalised formula for Boolean operators.

2.2 Information Filtering

Information retrieval, as we have discussed, is concerned with retrieving information to a user, on the basis of a user query. Information filtering can be seen as the other side of the coin: it is concerned with building a long-term profile of a user's information need, and sorting out relevant new incoming information to the user (Belkin & Croft, 1992).

Information filtering is used in a slightly different situation than that of information retrieval: the document set is dynamic in that new documents arrive all the time, and the query is expressed as a user's long-term information need. In short, the filtering process works by inspecting incoming information and checking whether it should be presented to the user or not.

A document filtering system need not be very different from a retrieval system, at least not at the technical level. Documents and queries can be represented as in the Vector Space, Latent Semantic Indexing or Probabilistic models, and the method for matching queries with documents can be essentially the same. A user's interest can be described by a set of content features such as words or phrases appearing in documents found relevant by the user.

A document filter should learn how to correctly label, for a given user, unseen information as relevant or not. In machine learning, this corresponds well to the classical framework of supervised learning. The input to the learning algorithm is a representation for those documents the user has viewed and an indication for each document if it was found relevant or not.

If viewed in this form, document filtering can be seen as text categorisation, discussed in the next section.

2.3 Text Categorisation

Text categorisation (or classification) is the task of assigning one or more categories or classes to a text document. Text documents may be web pages, news articles etc. Categories can be topical, for example business, sports, science, but can also be an indication of whether the document is relevant or not to a user.

For document representation, we have already encountered the bag-of-words representation. This is a simple, and often efficient, representation for text categorisation. Standard text pre-processing by stemming and by removing stop words has similar positive effects as in document retrieval.

Feature selection is used to remove non-informative terms for the task at hand and thus further reduce the dimensionality of the problem. Other representations have also been used: n-grams and higher-level orthogonal features such as latent semantic dimensions.

An extensive study of different classifier's performance on a news article categorisation task is reported in (Yang & Liu, 1999). Five methods were evaluated: Support Vector Machines, Nearest Neighbour, Linear Least Squares Fit, Neural Networks and Naive Bayes. The five methods were found to perform comparably well for tasks where each category contained more than a few training examples. When the number of examples per category was small, the Support Vector Machines, Nearest Neighbour and Linear Least Square Fit methods performed significantly better than the other two.

2.4 Collaborative Filtering

Collaborative filtering (Goldberg et al., 1992; Shardanand & Maes, 1995) (CF) is a way of automating "word-of-mouth", the process by which people give and take advice from each other. In everyday life we get and give advice or follow trails and this sometimes helps us to find good books, nice restaurants etc. In order to automate this process, it was suggested that user opinions, for example ratings, could be used as a basis for information filtering.

To illustrate the idea of collaborative filtering, consider the following: If two people tend to like the same movies, it is probably the case that one of them would like a movie that the other has just recently seen and liked. An important aspect of collaborative filtering is that it can be used in domains where the item's content is not easily parsed, or even when items carry no content at all.

The filter can then be used in either of two ways: users can explicitly ask what the system believes they would think about a certain item, or ask the system to return a ranked list of items the system thinks they would like.

Collaborative Filtering allows us to tackle three drawbacks found in traditional filtering methods:

- Items must be parsable; it is for example difficult to search for the content of music or movies unless these objects are explicitly tagged with a description of the content (Shardanand & Maes, 1995).
- It is difficult to achieve serendipity: content-based systems do not typically allow for discovery of information not sought for.
- Content-based filtering disregards qualitative aspects such as genre (Karlgren, 1999).

Combined with regular content-based techniques, collaborative filtering offers a powerful way of finding and filtering information.

The fundamental idea in collaborative filtering is thus to utilise the subjective user opinions about information, instead of focusing on the content or structure. The algorithms operate on data composed by user ratings for a set of items. This data is often sparse; not all users rate all items. Ratings may be *explicit*, in the users express an opinion on an item, or *implicit* in the sense that the rating is taken from the user's actions on items.

A collaborative filtering system will not work well unless there is sufficient amount of data to make predictions from: after all, predictions are based on finding regularities and similarities in the data. When there is no or little data available, the system is faced with the *cold-start* problem (Maltz & Ehrlich, 1995). If there is little or no rating data available, then the system must be filled with data from another source before any reasonable predictions can be made.

One approach is to construct artificial users or prediction rules based on background knowledge of the domain (Sarwar et al., 1998). Background knowledge can for instance be relationships between items or users based on the item content. In the movie domain, we could use the movie's genre, director, actors, etc. to construct some general rules of how people rate movies. This method typically creates stereotypes and relationships between users and items found in the majority of a population.

An early reference to collaborative filtering or recommender systems is the work of Karlgren (1990). In his work, a rating vector describes a user's interest for a set of items, such as books, and these rating vectors are matched to produce a prediction. The term collaborative filtering was coined in the 1992 paper by Goldberg and colleagues (Goldberg et al., 1992), in which the authors describe a system where a collaborative filter was used to filter out irrelevant e-mail. Another early system is Ringo, a recommender system for music albums and artists, where user similarities are calculated by matching user rating vectors (Shardanand & Maes, 1995).

Two different strategies for making predictions based on collaborative data have emerged: item-based and user-based. Item-based strategies use similarities between items to make predictions whereas user-based calculate similarities between users. Item-based methods are explored in (Sarwar, Karypis, Konstan, & Reidl, 2001) and have been successfully applied to commercial systems, most notably Amazon (Linden, Smith, & York, 2003).

Another distinction can be made between memory-based and model-based algorithms. This division of memory- versus model-based is followed in the next two sections, where an overview of some of the algorithms described in the literature is given.

2.4.1 Memory-based Collaborative Filtering

Memory-based CF uses a nearest-neighbour approach, similar to the Nearest Neighbour algorithm (2.5.2), or the Vector Space Model (2.1.2) for text retrieval.

Herlocker et al. (1999) discuss three central steps in memory-based algorithms: a) weighting neighbours, b) selecting a suitable subset of neighbours and c) producing predictions from the neighbour set. To alleviate the sparsity problem, they propose *significance weighting* to penalise user weights that are based on a small number of overlapping weights, and to increase the similarity between users when the overlap is larger.

The neighbourhood may be selected by taking all users as potential neighbours to the *active user*, or to select a subset using a top-k method, or by thresholding the similarity value. In general, the top-k method is preferred since it makes it easier to find a neighbourhood covering a high number of items or documents.

The weighted neighbourhood is then used to predict how the active user would rate items that the user has not already rated. One method is to calculate the weighted average of the neighbour's ratings, where the weight is the similarity value between the active user and the neighbour. Memory-based predictions have been used in systems such as GroupLens (Konstan et al., 1997) and Ringo (Shardanand & Maes, 1995).

The formulation of memory-based algorithm collaborative filtering that will be used is defined in (Breese et al., 1998). The prediction is based on a linear combination of other users' ratings for the items, weighted by their similarity with the active user. The prediction $p_{a,j}$ for user a on item j is

$$p_{a,j} = \bar{v}_a + \kappa_j \sum_i w(a,i)(v_{i,j} - \bar{v}_i) \tag{2.8}$$

where the index i runs over all users that have at least one rating for item j.

Each user i has a vector v_i of ratings, whose rating for item j is denoted $v_{i,j}$. The value $\bar{v}_i$ is the mean value of the ratings made by user i.

The function $w(a, i)$ should measure the similarity between two users a and i. The more similar, the more influence on the predictions for user a, and vice versa. In collaborative filtering, this weight is often taken to be the correlation coefficient between the two users' rating vectors, or a variant thereof.

There is also a normalising factor, κ_j, selected so that the absolute values of the weights $w(a, i)$ sum to unity. The sum of the absolute values of the weights is $\sum_i \|w(a, i)\|$ so κ_j is taken to be

$$\kappa_j = \frac{1}{\sum_i \|w(a, i)\|} \tag{2.9}$$

The basic similarity function for collaborative filtering is *Pearson correlation*. It measures the quality of a least squares fit to a set of data points:

$$w(a, i) = \frac{\sum_j (v_{a,j} - \bar{v}_a)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_j (v_{a,j} - \bar{v}_a)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}} \tag{2.10}$$

In collaborative filtering the rating vectors are generally sparse, meaning that each user only rates a very small subset of all available items. In effect, the actual number of items in the intersection of two users' rating vectors can be small, and this problem led to other formulations of the weight $w(a, i)$.

One extension is to calculate the correlation in the union of the voting vectors, by replacing missing ratings with a default value. This partially alleviates the problem of sparsity, since the union of two user's rating vectors is in practical cases a much larger set than the intersection.

Another extension is influenced by document weighting in Information Retrieval (Salton & McGill, 1983). Each item is given a weight that decrease as the number of ratings for the item increases. This *inverse user frequency* is based on the reasonable assumption that items that are rated by few users are better indicators of user similarity than items that all or very many users have rated.

As with the VSM model of information retrieval, the available data can be represented by a sparse matrix: in this case a matrix of users and items (see Figure 2.4 for an example of a user-item matrix).

Memory-based methods differ in the choice of similarity measure and in the way they combine the neighbours to form a prediction. The first systems used measures the Pearson correlation for determining the similarity between user profiles, as in e.g. (Resnick, Iacovou, Suchak, Bergstrom, & Riedl, 1994).

The advantage of the memory-based scheme over other methods is that its structure is dynamic and immediately reflects changes in the user database.

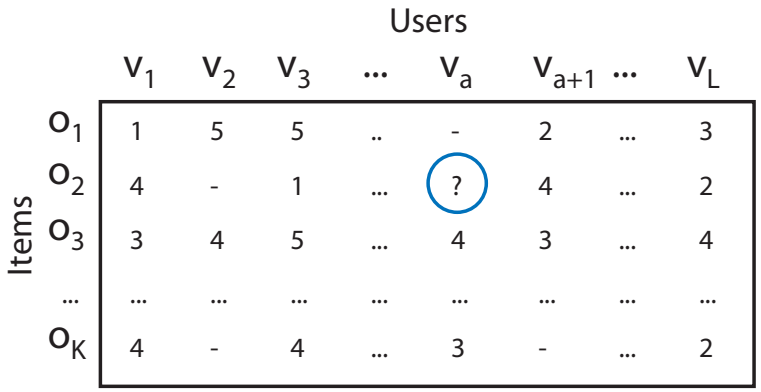

Figure 2.4: Example of a user-item matrix of ratings. A non-zero value at position $[j, i]$ denotes the rating for item j by user i. The circle marks an unrated item: to predict what user v_a will rate for item o_2 is an instance of collaborative filtering.

Every new rating added to the user database will be included in the neighbourhood search. The reason for this is that similarities between users are calculated in memory when needed. This property is also the potential drawback of the method. When user profiles are matched against each other every time a prediction is needed, the process can be extremely slow. This would not only take time, but also require a large amount of memory. In some cases this has been solved by only keeping parts of the user database in memory and sample users from this subset, or to precompute the similarity matrix.

2.4.2 Model-based Collaborative Filtering

A Model-based collaborative filtering algorithm builds one or more models from the user data that are used to make predictions. Several algorithms have been proposed within this framework. Bayesian networks, for instance, have been found to be about as effective as the memory-based correlation methods outlined in the previous section. Such a network has one node for each item, and each node has as many states as there are rating values. Each node has one or more parents, which represents the conditional probabilities for the node's possible rating values. The learning phase consists of searching for a network structure where each node's parents are the best predictors for that item's ratings. When filtering information, the user's profile is exposed to the network, and the user's rating for an item is predicted according to the conditional probabilities for the corresponding node.

Latent Semantic Indexing has also been successfully applied to collaborative filtering (Billsus & Pazzani, 1998; Sarwar, Karypis, Konstan, & Riedl, 2000). The algorithm is the same as in text retrieval, but the matrix cells now contain user ratings. As in text retrieval, there are both advantages and disadvantages to using Latent Semantic Indexing. In some cases, it increases prediction accuracy compared to base line algorithms such as Pearson correlation. However, it has not yet been compared to state of the art algorithms such as Bayesian networks or extended memory-based algorithms. One problem, also found in text retrieval, is that it is difficult to interpret the meaning of the latent semantic dimensions. This is not necessarily an important issue; the collaborative filter may well be used as a black box. For some applications though, it may be necessary to explain why and how the system has come up with a certain prediction.

One of the major benefits of Bayesian networks and similar machine learning algorithms is that once the model has been built, it can generate predictions at a high speed and with a small amount of resources. The potential drawback is the static structure of many models; once the model has been built it is difficult to update it without rebuilding it. In dynamic domains where users and ratings are constantly changing the model could soon become inaccurate.

2.4.3 Hybrid Methods

Information retrieval and filtering systems may use several different sources of data. Collaborative information, such as users' ratings for items, says something about the user's preferences regarding the objects. Content information such as object descriptors, keywords, phrases, etc. says something about the properties of those objects. It is obvious that both types of information should be used when making predictions about objects, if they are both available.

Basu, Hirsh and Cohen (1998) combine content and collaborative attributes for the task of recommending movies. The collaborative attributes are set-based features such as "Users who liked movie X". Content attributes were extracted from the Internet Movie Database[2] and included the movie's actors, directors, genres, reviews, etc. A set of hybrid features was also constructed, such as "Users who like the genre Drama", which helped increase the recommendation accuracy.

Mooney and Roy (2000) use a content-based approach for the task of recommending books. Each user rates a set of books, and each book and the corresponding rating forms a training example for a classifier that is built specifically for each user. A book is recommended to a user if the user's classifier predicts a high rating for the book. A book may contain several different content sources such as the title, author, synopsis, reviews etc. For managing these different

[2] www.imdb.com

sources, a multinomial text model (A. McCallum & Nigam, 1998) is used for learning a Naïve Bayesian classifier. Such a multinomial model can handle vectors of bags of words instead of just bags of words.

In (Melville, Mooney, & Nagarajan, 2002) the content-based book recommendation process is extended to include collaborative attributes. Each user rating vector is augmented with predictions from the content-based prediction, so that each user's rating vector is a mixture of real ratings and predicted ratings. By a set of experiments it was found that the approach of augmenting the user rating vector with content-based predictions yielded better results than pure content-based or collaborative approaches.

A probabilistic framework for combining content and collaborative attributes is presented in (Popescul, Ungar, Pennock, & Lawrence, 2001). The assumption is that users choose "topics" of interest, while topics are the generators of documents and descriptors. However, these topics are unknown, or latent, variables. To estimate these variables, a *latent class model* is trained on observations of users selecting documents containing descriptors. The number of latent variables is set prior to learning the model.

Baudish (1999) describes a possible integration of content-based and collaborative attributes by joining (as opposed to merging) the attributes into a single table. The attributes are joined by using the two relations (descriptor-matches-object) and (user-likes-object) so that the rows and columns of the table correspond to users, objects and descriptors. Each cell defines a particular type of combined function. For example, the cell (user, descriptor) is interpreted as a function which transforms users into descriptors, i.e., generates keywords from a profile. Content-based filtering is defined by the cell (descriptor, object) and collaborative filtering by the cell (user, object) or (object, object). Through the operations "likes" and "matches" this framework allows for all standard content-based and collaborative types of queries, but also more elaborate ones such as "Give me all objects that users that are similar to user X like".

2.5 Machine Learning

Machine learning is the general study of algorithms that automatically improve their behaviour on the basis of experience (see (Mitchell, 1997) for a good starting point on the subject). Machine learning algorithms are used for many tasks, for example predicting the protein structure from a protein sequence, recognising handwritten optically scanned digits, categorising text documents into subject categories, driving unmanned vehicles on a public highway, quantifying the species-specificity in genomic signatures, and many more.

A machine learning algorithm forms a hypothesis (or model) by observing a

set of *examples*. An example is a representation of some entity: it could be a vector space representation of a document, a protein sequence, a bitmap representation of an optically scanned digit, etc. In its most general form, machine learning is concerned with finding the function that generated the examples. The solution, or hypothesis, is an approximation of this function.

It is important to distinguish between examples that the program has already seen (in the learning phase), and examples that the program is asked to, for example, predict a value or class for (in the operational phase). The former are called training examples, and the latter test examples.

There are three basic types of machine learning programs: supervised, unsupervised and reinforcement learning. Only supervised learning algorithms are considered in this text.

2.5.1 Supervised Learning

Supervised learning deals with the problem of predicting the *class* or value of an example, on the basis of a set of *training examples* where the class for each is known *a priori*. The class denotes some property of the example, and the machine learning algorithm should correctly distinguish examples that belong to one class versus examples of another class. Recall the discussion on text categorisation in Section 2.3; this can be modelled as a supervised learning problem, where the examples are text documents and the classes are document categories.

The basic task in supervised learning is binary *classification*: to predict whether an example belongs to one class or the other: for example, to predict whether a text document is relevant or not for a user. Other tasks may involve multiple classes, e.g. to categorise news articles into a set of predefined topics such as sports, politics, entertainment etc. A binary classifier can be extended to handle multiple classes by creating several binary classifiers that are trained with two classes at a time from the training set (or trained to make a binary decision regarding membership to each of the classes).

Supervised learning also encompasses *regression*, the prediction of real values. Regression is a well-studied field in statistics, and many of the statistical algorithms may of course be directly applied. There are specialised algorithms for regression, such as regression trees and Support Vector Regression. A shortcut to regression is to split each output variable into a number of discrete classes. Effectively, this turns a regression problem to a multi-class classification problem. Conversely, binary classification problems can easily be turned into regression problems.

Supervised Learning - Terminology

A supervised learning algorithm is given a training set of l labelled examples $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_l, \mathbf{y}_l)$ where $\mathbf{x}_i \in R^n$, if the examples are expressed as vectors, and $\mathbf{y}_i$ are the associated labels (classes or output variables). The labels can be binary $\mathbf{y}_i \in \{0, 1\}$, discrete $\mathbf{y}_i \in N$, or real-valued $\mathbf{y}_i \in R$. We will if not stated otherwise assume that $\mathbf{y}$ is not a vector but a single variable y. The label y_i is sometimes written $f(\mathbf{x}_i)$, as a function of the example.

The task for a supervised learning algorithm is to find a good approximation to the function $f(\mathbf{x})$ that generated the training examples. The solution, or hypothesis, model, is a new function $\widehat{f}(\mathbf{x})$. The function $\widehat{f}(\mathbf{x})$ is learned from the *training set* of examples that contain the (class) labels. The function is evaluated using another set of examples, the *test set*, where labels are hidden from the algorithm.

2.5.2 The Nearest Neighbour Algorithm

The Nearest Neighbour algorithm (KNN) is an instance-based learning method. The general idea is to simply store (remember) the training examples, and when classifying a test example, find the "nearest" training example(s). To classify the new example, the class can for instance be taken to be the majority class of the "nearest" training examples.

In this way, the nearest neighbour algorithm generalises beyond the training examples only whenever it is asked to classify a new example (Mitchell, 1997). Nearest Neighbour is a "lazy" method, since no explicit model is built during training.

The central question is then how to find the "nearest" training example to a new example $\mathbf{x}$. A common choice of distance metric is the Euclidean distance

$$d(\mathbf{x}, \mathbf{z}) = \left[\sum_{j=1}^{n} (\mathbf{x}_j - \mathbf{z}_j)^2 \right]^{1/2} \tag{2.11}$$

which measures the distance between two vectors $\mathbf{x}$ and $\mathbf{z}$. The Euclidean distance function is very sensitive to features whose values are in different ranges, why it is always important to normalise the vectors.

For classification, a suitable method is to take the majority vote among the k nearest neighbours. For regression with a single output variable, the mean value of f from the k nearest neighbours may be used.

2.5.3 Artificial Neural Networks

Artificial Neural Networks (ANN) is a family of machine learning algorithms that operate on a network of neurons, or nodes.

The first neural network algorithm, the Perceptron, was developed in the 1940s, and is due to Rosenblatt (1958). The Perceptron is the simplest form of network. It consists of a single input layer of nodes, one output node, and weights connecting each node in the input layer to the output node, see Figure 2.5.

An example is fed into the network in the input layer, where the input nodes take the values of the example attributes. The input layer is of the same dimension as the examples. A Perceptron's model of the training data is the set of weights that connect the input layer to the output node; the value of the output node is a function of the input nodes and the weights. The Perceptron is traditionally used as a binary classifier: the value of the output node is transforming to either $+1$ or -1 depending on whether the output value is above or below zero.

The Perceptron classification rule can be stated as

$$\begin{aligned} \hat{f}(\mathbf{x}) &= \text{sign}(\langle \mathbf{w} \cdot \mathbf{x} \rangle + b) \\ &= \begin{cases} +1, & \text{if } \langle \mathbf{w} \cdot \mathbf{x} \rangle + b \geq 0 \\ -1, & \text{otherwise} \end{cases} \end{aligned} \tag{2.12}$$

where $\langle \mathbf{x} \cdot \mathbf{z} \rangle$ is the inner product between two vectors

$$\langle \mathbf{x} \cdot \mathbf{z} \rangle = \sum_{j=1}^{n} \mathbf{x}_j \mathbf{z}_j \tag{2.13}$$

Geometrically, the Perceptron's hypothesis $\hat{f}(\mathbf{x})$ is represented by a straight line (*hyperplane* in higher dimensions). The examples on one side of the hyperplane are classified as $+1$, the others as -1, see Figure 2.5. The weight vector $\mathbf{w}$ is a normal vector to the plane, and determines a direction perpendicular to the plane. The *bias* b determines the distance to the hyperplane from the origin.

To train a Perceptron is to find a weight vector $\mathbf{w}$ and a bias b that separate the examples. The algorithm is simple and can be formulated as follows. First, set the vector $\mathbf{w} \leftarrow \mathbf{0}$ and the bias $b \leftarrow 0$. Each training example $\mathbf{x}_i$ is then in turn classified according to the current $\mathbf{w}$ and b. If the classification is incorrect, $\mathbf{w}$ and b are updated to reflect the new example. If the classification is correct, there is nothing to do for that example. The algorithm terminates when all examples are correctly classified by the same $\mathbf{w}$ and b.

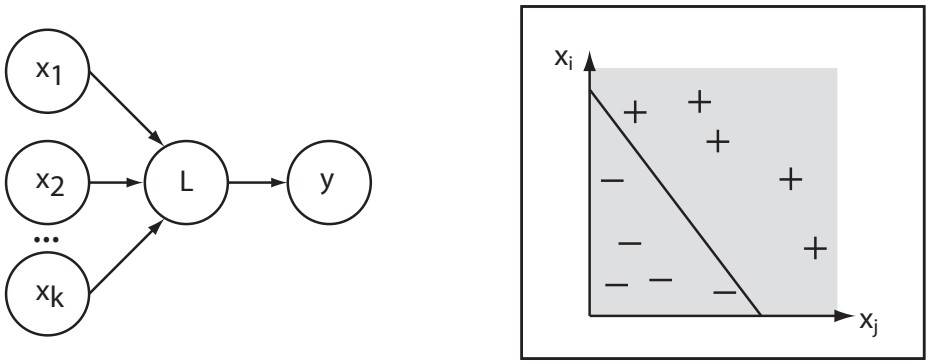

Figure 2.5: Perceptron (left) and an example of the linear decision surface (right) learned by a Perceptron. The input layer to the left in the network architecture takes the values from an input vector $\mathbf{x}$; the input is combined with the input and the weights (by the inner product, marked by arrows), converted by the step function L, to produce the output y.

The crucial step is to select the Δ values which should update $\mathbf{w}$ and b so that eventually all examples are classified correctly. It can be shown that by setting $\Delta\mathbf{w}_j = y_i\mathbf{x}_{i,j}$ and $\Delta b = y_i$, the algorithm will converge after a finite number of iterations, if the examples are *linearly separable*. That a set of examples are linearly separable means that there exists a hyperplane that correctly classifies the examples. After all, if the examples are not linearly separable, then it is impossible to find a hyperplane (in input space) which correctly classifies the examples.

The definition of linearly separable examples will be used when discussing Support Vector Machines, why the formal definition is stated here. A set of examples $S = \{\mathbf{x}_i, y_i\}_{i=1}^{l}$ are linearly separable if there exists a $\mathbf{w} \in R^n$ and a $b \in R$ such that

$$\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \geq +1 \quad \text{for } y_i = +1 \tag{2.14}$$

$$\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \leq -1 \quad \text{for } y_i = -1 \tag{2.15}$$

The points $\mathbf{z}$ that fulfil $\langle \mathbf{w} \cdot \mathbf{z} \rangle + b = 0$ lie on the separating hyperplane.

Gradient Descent

The Perceptron assumes that the examples are linearly separable; what to do when the examples are not linearly separable but we still wish to use a hyperplane for representing the hypothesis ? One choice is to let the hyperplane misclassify some examples, but choose this hyperplane while keeping the number of errors as small as possible.

To define the error, inspect the output from the Perceptron:

$$o_i = \langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b \tag{2.16}$$

The error can e.g. be defined as the sum of the quadratic errors on an entire training set S

$$E[\mathbf{w}] = \frac{1}{2} \sum_{i=1}^{l} (y_i - o_i)^2 \tag{2.17}$$

To find the $\mathbf{w}$ that minimises the error function $E[\mathbf{w}]$, first note that $E[\mathbf{w}]$ is a function of the weights $\mathbf{w}$. By taking the vector derivate (the *gradient*) of E, we can use this vector to find the direction, at a certain point, where the function decrease the most.

The gradient $\nabla E[\mathbf{w}]$ is a vector that consists of the first-order partial derivatives of $E[\mathbf{w}]$:

$$\nabla E[\mathbf{w}] = (\frac{\partial E}{\partial \mathbf{w}_1}, \frac{\partial E}{\partial \mathbf{w}_2}, \ldots, \frac{\partial E}{\partial \mathbf{w}_n}) \tag{2.18}$$

The direction of the gradient, at a certain point, is the direction where the function $E[\mathbf{w}]$ increases the most. The algorithm for updating a Perceptron network using the gradient is called *gradient descent*, since the algorithm updates the weights $\mathbf{w}$ in the opposite direction of the gradient, which is the direction where the function $E[\mathbf{w}]$ decreases the most.

There are basically two types of gradient descent: batchwise or incremental. Batchwise gradient descent uses the error function 2.17, that is, it calculates the gradient on the error of the entire training set. Incremental gradient descent, on the other hand, incrementally updates the weights by the error for each training examples.

The batchwise gradient descent rule is derived by observing that

$$\begin{aligned} \Delta \mathbf{w} &= -\eta \nabla E[\mathbf{w}] \\ \Delta \mathbf{w}_i &= -\eta \frac{\partial E}{\partial \mathbf{w}_i} \end{aligned} \tag{2.19}$$

The parameter η is called the *learning rate*, and determines how large steps the update will take on the error surface. For the incremental rule, we have to update weight i for example p, by $\mathbf{w}_i \leftarrow \mathbf{w}_i + \Delta_i \mathbf{w}_i$. The error function is

$$E[\mathbf{w}] = \sum_{i=1}^{l} E_i \tag{2.20}$$

and for one example p this is

$$E_i = \frac{1}{2}(o_i - y_i)^2 \tag{2.21}$$

Through partial derivation

$$\begin{aligned} \Delta_i \mathbf{w}_i &= -\eta - (o_i - y_i)\mathbf{x}_i \\ &= \eta(o_i - y_i)\mathbf{x}_i \end{aligned} \tag{2.22}$$

The value b can be updated by $\Delta_i b = \eta(o_i - y_i)$.

Backpropagation

The Perceptron is very limited, since it can only express linear functions (hyperplanes). To enable a neural network to express non-linear hypotheses, one introduces *hidden layers*, so that the network consists of several layers of nodes. Figure 2.6 displays such a network. The input layer to the left in the network architecture takes the values from an input vector $\mathbf{x}$; the input is combined with the input and the weights (by the inner product, marked by arrows), converted by the step function L, to produce the output y.

There are several choices of algorithms to train a multi-layered feed-forward network. The first method that gained popularity was Back Propagation, which uses gradient descent for weight updating.

In short, the algorithm is as follows; first, the network weights are initialised to small, random values. The network output is then calculated for a training example. The output is compared to the desired output, and an error is calculated. The error is propagated backwards in the network, and the weights updated accordingly.

Since the Backpropagation method calculates the gradient, it will involve calculating the derivative of the function that converts a node's input. This function is called *activation function*, and in the case of the Perceptron there was only one such function, namely the sign function. Since it is required to calculate the gradient of the activation function, it must be differentiable.

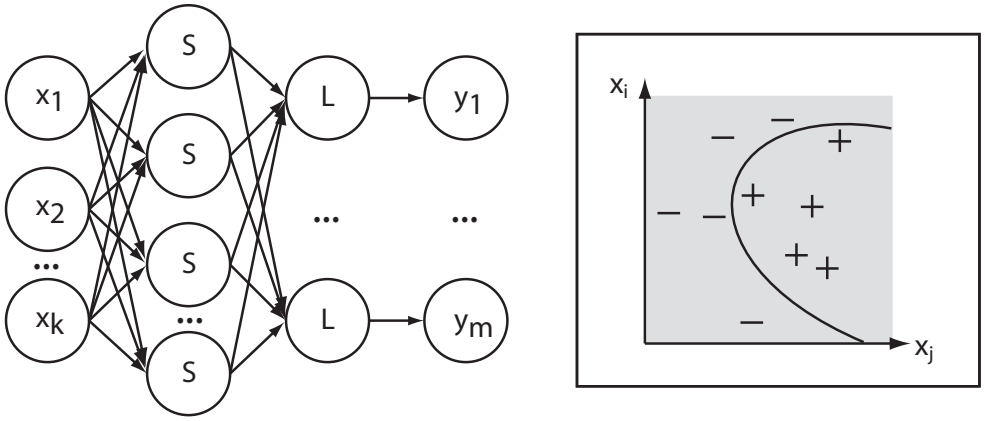

Figure 2.6: Feed-forward network (left) and an example of a decision surface (right) of a network that has learned a non-linear function. The input layer to the left takes the values from an input vector **x** which is combined with the weights by a transfer function. The output from the hidden layer (nodes marked with S for sigmoid) is again combined with the next set of weights and (in classification) converted by the step function L to the output **y**.

A common first choice of a suitable differentiable activation function is the sigmoid or "squashing" function

$$g(z) = \frac{1}{1+e^{-z}} \tag{2.23}$$

This function "squeezes" its input value z to a real number in the range $(0,1]$, and has a very simple derivative

$$g'(z) = g(z)(1-g(z)) \tag{2.24}$$

The derivation of the general Backpropagation rule can be found in (Bishop, 1995). The gradient descent method may (theoretically) get stuck in local minimum of the error function. There are, however, ways of overcoming this problem, such as adding *momentum* to the weight update. This adds a small part of the previous update to the current update, and is a common method to use to help alleviate the problem of local minimum.

Another problem when using the Backpropagation method is to select the number of hidden layers and the number of nodes in each such layer. A theoretical analysis of the expressiveness of a feed-forward network reveals that

two hidden layers are theoretically sufficient to enable the network to learn any differentiable function. Since the hidden layers will add to the memory of the network, it is desirable to have as few hidden nodes as possible, with as low error as possible.

2.5.4 Support Vector Machines

Support Vector Machines are a family of machine learning algorithms for tasks such as classification and regression (Vapnik, 1995)(Cristianini & Shawe-Taylor, 2000).

In its simplest form, when examples are linearly separable and belong to one of two classes, the algorithm finds the hyperplane that correctly classifies the examples, but also has the maximum distance to the examples. This hyperplane is called a maximum margin hyperplane, since the margin between the classes is maximised. As discussed in Section 2.5.3, the Perceptron also finds a separating hyperplane, but that hyperplane is not necessarily a maximum margin hyperplane.

Maximising the margin is supported by a theory that gives some evidence that this implies good generalisation (Vapnik, 1995). In the following, some of the properties of SVMs are discussed.

Going back to the ANN gradient descent method, it was stated that the basic learning principle was to (roughly) minimise the empirical error (or *risk*), in order to select the network weights. However, even if the network is trained until the sum of the quadratic error on a test set is 0, this does not mean that the network will have zero error on an independent test set. The true error (or Expected Risk) is not only dependent on the Empirical Risk, but also on, e.g., the number of nodes in the hidden layer. More generally, there are other properties than the empirical risk that affects how close an algorithm is to the true error.

VC Dimension The *VC dimension* is one formulation of a property of a learning algorithm that affects its generalisation ability. Let R be the "true" error or *expected risk*. This is the smallest error a specific classifier would do if given the (true) distribution of the function we want to approximate. The problem is that the distribution is unknown.

Let R_{emp} be the *empirical error*, the error that is obtained from a sample of the distribution. Then, the expected risk R can be *bounded* by the empirical risk R_{emp} and a value that is dependent on the number of training examples and the "capacity" of the classifier. The "capacity" is a measure of the expressiveness of the classifier's possible hypotheses.

Let h be the "capacity" and l the number of training examples, then with probability at least $1-\eta$ the following result holds (Vapnik, 1995)

$$R \leq R_{emp} + \sqrt{\left(\frac{h(\log(\frac{2l}{h})+1) - \log(\frac{\eta}{4})}{l}\right)} \tag{2.25}$$

The right hand side will decrease as the number of examples l increase, and decrease as the capacity h decrease. The parameter h is called the *VC dimension* of a set of functions (the classifier's possible hypotheses).

The VC-dimension of a set of binary classification functions F is the maximum number of examples that can be completely "shattered" by functions from the function set F.

To understand the VC dimension, assume there are l examples with class labels $\{-1,+1\}$. The examples can be labelled in 2^l different ways. A set of functions F completely shatter the examples if, for each labelling, there exists a correct classification of the examples using functions from F.

As an example, assume that F is the class of straight lines in R^2. The maximum number of examples that can be completely shattered is three. Three examples can be labelled in $2^3 = 8$ different ways; it is possible to find three examples that can be completely shattered but impossible to find four.

According to this VC theory, an algorithm should try to minimise both the empirical risk and the VC-dimension, and also increase the number of training examples. The VC-dimension must therefore be controlled (directly or indirectly) by the algorithm. SVMs implicitly minimise the VC dimension by maximising the margin.

Linearly Separable Problems

When examples are linearly separable, the Support Vector Machine algorithm finds the maximum margin hyperplane that separates the examples. The problem of finding this hyperplane is expressed as a mathematical optimisation problem:

$$\begin{aligned} \min \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{w.r.t} \quad & y_i(\langle \mathbf{w} \cdot \mathbf{x}_i \rangle + b) - 1 \geq 0, \forall i \end{aligned} \tag{2.26}$$

which states that the procedure should find the hyperplane with maximum margin under the conditions that all examples are correctly classified by the hyperplane.

The learning algorithm is expressed as a quadratic programme, a mathematical optimisation problem of a quadratic function with linear constraints. The

objective function is to minimise the norm of the hyperplane weight vector, to find the hyperplane with maximum distance to the examples. The constraints, or conditions, are that the hyperplane correctly classifies each training example.

Solving the optimisation problem leads to

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \mathbf{x}_i \tag{2.27}$$

which shows that the weight vector $\mathbf{w}$ (the separating hyperplane) can be written as the sum of all the example vectors $\mathbf{x}_i$ weighted by the introduced α_i-values and the class labels y_i. The goal of the optimisation procedure is to find the unknown variables α_i. The SVM classification function is, after finding these variables, written as

$$\begin{aligned} \hat{f}(\mathbf{z}) &= \text{sign}(\langle \mathbf{w} \cdot \mathbf{z} \rangle + b) \\ &= \text{sign}(\sum_{i=1}^{l} \alpha_i y_i \langle \mathbf{x}_i \cdot \mathbf{z} \rangle + b) \end{aligned} \tag{2.28}$$

The values α_i are either $= 0$ or > 0. For the examples i where $\alpha_i = 0$ it is clear that the example will not contribute to the separating hyperplane. For those examples i where $\alpha_i > 0$ then the example contributes to, or supports the hyperplane and are thus called Support Vectors.

The learning problem is convex, since the norm of a vector is a convex function and the constraints are linear and therefore convex. This has the effect that the optimisation procedure is guaranteed to find the function's global minimum, since any local minimum of a convex function is always global.

Thus, the algorithm finds the hyperplane that has maximum margin between the two classes. The weight vector of this hyperplane can be expressed as a linear combination of the training examples. In this linear combination, only a subset of the examples has non-zero coefficients, the Support Vectors. If only the Support Vectors are kept and the machine re-trained, the same separating hyperplane will be found.

Linear Separable Problems with Errors

When the examples are not entirely linearly separable, slack variables are introduced in the optimisation constraints to allow for misclassifications. The objective function is also extended to include the sum of the slack variables, C, controlling the number of misclassifications. The parameter C gives a trade-off between the margin and the sum of the errors.

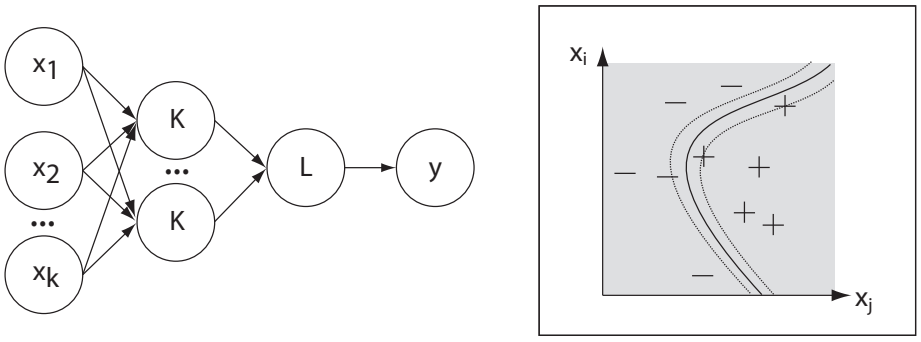

Figure 2.7: Support Vector Machine (left) and the decision surface (right) of a maximum margin hyperplane in feature space mapped back to input space

Non-Linear Decision Functions

Support Vector Machines are also capable of representing non-linear hypotheses, by mapping the examples from input (example) space to another, high-dimensional, feature space. The algorithm is then trained to find a separating hyperplane in feature space. The feature space could be, for example, the space of all possible pair-wise combinations of the input dimensions. A direct mapping from input space to feature space can be extremely costly or even impossible.

A special class of functions can implicitly map examples to a feature space, by calculating the inner product of the examples in feature space but only using the examples from input space. Such functions are called Kernel functions, and can be used in any learning algorithm involving inner products between examples.

In the Support Vector Machine optimisation phase, examples are only compared by the inner products, so by replacing the inner product in input space with a Kernel function, the Support Vector Machine finds the maximum margin hyperplane in the feature space mapped by the Kernel.

Several Kernel functions have been used in Support Vector machines, some of which coincide with other learning algorithms. The simplest is the linear kernel, which does not perform any mapping but instead calculates the inner product of two vectors in input space. An example of a more elaborate kernel is the hyperbolic tangent (tanh) kernel function. The architecture of such a SVM is identical to a feed-forward neural network with the Support Vectors as nodes in the hidden layer, and tanh as the activation function (Vapnik, 1995).

The following three general-purpose kernels (and the linear kernel) are often used as a first start when trying to find a good kernel function for use with the SVM:

Polynomials of degree d	$K(\mathbf{x}, \mathbf{z}) = \langle \mathbf{x} \cdot \mathbf{z} \rangle^d$
Radial Basis functions	$K(\mathbf{x}, \mathbf{z}) = \exp(-\gamma \|\mathbf{x} - \mathbf{z}\|^2)$
ANN with tanh as Activation Function	$K(\mathbf{x}, \mathbf{z}) = \tanh(\gamma \cdot \langle \mathbf{x} \cdot \mathbf{z} \rangle + \theta)$

2.5.5 Other Methods

The ANN, KNN and SVM algorithms all operate on data that is best described as mathematical vectors, or at least that the similarity between examples of the data is expressed as a vector similarity measure such as the inner product or vector distance. However, not all machine learning algorithms rely on the fact that examples are expressed as vectors. In fact, the SVM algorithm need not be used explicitly with vectors, if only a suitable kernel function can be devised that measures the similarity between two examples in the desired feature space.

For reference we discuss Bayesian Learning here, which is motivated by results from Bayesian statistics.

A Bayesian learning algorithm makes an estimate of which hypothesis is the most probable, assuming that the hypothesis is dependent on an underlying probability distribution. This distribution and the available data is used for estimating which hypothesis is most probable.

The most probable hypothesis is learned from data, where each example either increases or decreases the probability that a hypothesis is correct. Bayesian learning has a principled way of including prior knowledge, in the form of prior probabilities for each possible hypothesis. Classification is performed by combining the predictions of the different hypotheses, weighted by their probabilities.

Bayes Theorem

The main theorem in Bayesian learning is Bayes Theorem, which relates the posterior probability of a hypothesis with the observed probabilities.

Finding the best hypothesis in H, given data D, can be expressed as the problem of finding the most probable H, given training data D plus prior knowledge of the probabilities of various hypotheses in H. The following notation will be used; $P(h)$ is the probability that a hypothesis $h \in H$ holds, $P(D)$ is probability of observing data D (regardless of H), $P(D|h)$ is probability of observing data D, given that hypothesis h holds, and $P(h|D)$ is what we want to find: the probability that a hypothesis h holds, given training data D.

Bayes theorem relates $P(h|D)$ to $P(D|h)$, $P(D)$ and $P(h)$ by

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \tag{2.29}$$

Note that $P(h|D)$ increases with $P(h)$ and $P(D|h)$. Also, $P(h|D)$ decreases when $P(D)$ increases, since the more probable the data is regardless of the hypothesis, the less evidence it gives to support the hypothesis. Again, the probability $P(h|D)$ is called the posterior probability. The most probable (Maximum A Posteriori) hypothesis is

$$\begin{aligned} h_{MAP} &= \text{argmax}_{h\in H} P(h|D) \\ &= \text{argmax}_{h\in H} \frac{P(D|h)P(h)}{P(D)} \\ &= \text{argmax}_{h\in H} P(D|h)P(h) \end{aligned} \tag{2.30}$$

Now we know how to find the most probable hypothesis, but how can we find the most probable classification? This is accomplished by combining the classifications of all hypotheses, weighted according to their posterior probabilities. Let v_j be a particular classification from all classes V, then the probability that v_j is the correct classification is

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i)P(h_i|D) \tag{2.31}$$

The best classification is the one with highest probability. In practice, this method that we have outlined is difficult to use since it requires calculating the posterior probability of each hypothesis.

Another, more practical, method for estimating the most probable classification is Naïve Bayes, which can be used when the data is expressed by a conjunction of attribute values $(a_1, a_2, \ldots, a_n)$. Let a class be denoted v_j, from the set V of classes. The most probable classification is

$$v_{MAP} = \text{argmax}_{v_j \in V} P(v_j|a_1, a_2, \ldots, a_n) \tag{2.32}$$

Using Bayes Theorem the classification can be rewritten as

$$\begin{aligned} v_{MAP} &= \text{argmax}_{v_j \in V} \frac{P(a_1, a_2, \ldots, a_n|v_j)P(v_j)}{P(a_1, a_2, \ldots, a_n)} \\ &= \text{argmax}_{v_j \in V} P(a_1, a_2, \ldots, a_n|v_j)P(v_j) \end{aligned} \tag{2.33}$$

The question is now how to estimate $P(a_1, a_2, \ldots, a_n|v_j)$. By assuming that attributes are independent, then the probability of observing the conjunction of the attributes is the product of the individual attribute probabilities

$$v_{NB} = \text{argmax}_{v_j \in V} P(v_j) \prod_i P(a_i|v_j) \tag{2.34}$$

2.6 Experimental Evaluation and Data sets

In the fields of information retrieval, collaborative filtering and machine learning, a new algorithm or representation is often *evaluated* by an experimental procedure on real or synthetic data, in order to get an understanding of how well the algorithm performs. The typical evaluation metrics concern the algorithm's error (such as the misclassification rate of a classifier), the time taken to train a model and the time taken to make a prediction or calculate a ranked list of documents.

In general, it is also sometimes important to analytically determine the *complexity* of a new algorithm (algorithm running time and memory usage), as a mathematical function of the input.

The algorithms and representations presented in this thesis are primarily evaluated using experiments, and, when appropriate, by a more formal analysis. The experimental procedures, evaluation metrics and data sets that will be used throughout the thesis are discussed in the next pages.

2.6.1 Information Retrieval

To evaluate the performance of an information retrieval system, we must define what we mean by *performance*, and then find the appropriate measure. In 1966, Cleverdon (Cleverdon, Mills, & Keen, 1966) listed six different measurable factors that determine the performance of an IR system, that still hold today:

1. The *coverage*; the extent to which the system includes relevant matter.
2. The *time lag*; the average time between a query and an answer.
3. The *form of presentation* of the output.
4. The user's *effort* involved in obtaining answers to queries.
5. The *recall*; the proportion of relevant documents actually retrieved.
6. The *precision*; the proportion of retrieved documents that is relevant.

A retrieval algorithm, which is the central part of an IR system, is most often evaluated by precision and recall. This evaluation does not necessarily involve users, but is often performed on a collection of documents where queries and relevance judgements are known beforehand. The coverage is fixed, since the collection is fixed, which makes it easy to compare the performance of different algorithms be comparing the precision and recall values.

Precision and recall are defined using the notion of the *relevance* of a document to a given query. This notion is of central importance to IR. Nevertheless,

this notion is subjective, and different persons may disagree on which documents are relevant to a given query. Furthermore, it is not defined whether relevance should be interpreted as a binary decision, or if there are degrees of relevance.

Much can be said about the simplification of using relevance scores, such as binary judgements, as the notion of relevance of a document to a query. The first problem is that relevance is subjective; not all users agree on which items or documents are relevant to a given query. Moreover, human language does not make the matter less complicated; it is easy to form an ambiguous query that may have two different meanings in two different contexts. Another problem is that relevance does not need to be static; we continuously learn from the information that we retrieve – what was relevant today may not be relevant tomorrow.

These complications are however not assumed to be strong enough to invalidate large scale experimental evaluation of information retrieval systems using precision and recall. For this reason, it is custom to evaluate the performance, or effectiveness, of a retrieval algorithm using precision and recall.

Precision and recall have simple definitions, when we define relevance as a binary decision. Assume that the user poses a query, and that relevance judgements are available for all documents. Let A be the set of relevant documents to the query, and let B be the set of documents retrieved by the query. The precision $\mathcal{P}$ and recall $\mathcal{R}$ are then defined as (Rijsbergen, 1979):

$$\mathcal{P} = \frac{|\, A \cap B \,|}{|\, B \,|} \tag{2.35}$$

$$\mathcal{R} = \frac{|\, A \cap B \,|}{|\, A \,|} \tag{2.36}$$

For a retrieval system which ranks the search results, precision and recall is calculated for each document in the ranked result list. For the document with rank k, the sets A and B are deduced from the k top ranked documents. In this way it is possible to see how precision changes at different levels of recall, since for the $k+1$st highest ranked document, its recall must be equal to or higher than the recall for the kth highest ranked document.

When all precision-recall points are calculated for a query, it is common to interpolate precision on some standard levels of recall. The technique we employ is an interpolated eleven-point precision-recall curve, where precision is interpolated at each recall level from 0.0 to 1.0 with step size 0.1. The interpolated precision at some recall level is the maximum precision at the current and all higher levels. The results are then presented either in a table or in a graph. When there are many queries, it is common to present the result as the average (mean) of the queries. This evaluation is used in large-scale evaluation forums

such as the Text Retrieval Conference (TREC), and Cross-Language Evaluation Forum (CLEF).

For the text retrieval experiments in Chapter 3, we use several standard test collections. A test collection contains a number of documents, a number of queries and a set of relevance judgements that lists the relevance of documents to a query. Using this data, we can evaluate the effectiveness of a retrieval algorithm by plotting, e.g., the interpolated 11pt precision-recall curve.

Since IR has been an active research field for a long time, a number of test collections have been developed. In the early days, the collections were small and specific; now, much due to the TREC initiative, the collections are fairly large and can contain many different topics.

2.6.2 Text Categorisation

Text Categorisation, as discussed in Section 2.3, is the task of assigning one or more categories to a document, based on the textual contents of the documents. In the same spirit as when evaluating a retrieval system, a document can be said to be relevant, or belong, to a category or not. If a document can belong to more than one category, we calculate the average over the set of categories.

The standard terminology (see, for example, Witten and Frank (2000)), for the possible outcomes of a binary prediction is as follows: the prediction is one of true positive (TP), true negative (TN), false positive (FP) or false negative (FN). Positive means that the document was classified as belonging to the category, negative that it was not. True means that the classification was correct, false that it was not. From these four outcomes, we can re-formulate the standard IR evaluation metrics precision and recall as follows:

$$\mathcal{P} = TP/(TP + FP) \tag{2.37}$$

$$\mathcal{R} = TP/(TP + FN) \tag{2.38}$$

Precision $\mathcal{P}$ and recall $\mathcal{R}$ can be combined in a single measure (Rijsbergen, 1979), called the $\mathcal{F}$ measure:

$$\mathcal{F}_\beta = \frac{(\beta^2 + 1)\mathcal{P}\mathcal{R}}{(\beta^2)\mathcal{P} + \mathcal{R}} \tag{2.39}$$

If we let $\beta = 1$, we give equal weight to precision and recall, and obtain the $\mathcal{F}_1$ measure

$$\mathcal{F}_1 = \frac{2\mathcal{P}\mathcal{R}}{\mathcal{P} + \mathcal{R}} \tag{2.40}$$

In order to combine the $\mathcal{F}_1$ score across categories, there are essentially two possible approaches; micro- and macro-averaging. Micro-averaging means that we sum the TP, TN, FP and FN over all categories and then compute the $\mathcal{F}_1$ score. In macro-averaging, the $\mathcal{F}_1$ score is computed for each category, and then the average of these scores is taken.

The $\mathcal{F}_1$ measure is not the only possible evaluation metric for text categorisation; the so-called Break-Even Point (BEP) (Yang & Liu, 1999) has also been used in several experiments. The BEP is the point where precision equals recall, and can be calculated if the categorisation algorithm produces a ranked list of predictions. The $\mathcal{F}_1$ measure is used for the categorisation experiments in Chapter 4, since it is a relatively standard method, and also it does not require the classifier to rank its output.

2.6.3 Collaborative Filtering

There are several different evaluation metrics that have been proposed for collaborative filtering, and which one to choose depends on the user task. An extensive study of evaluation methods is presented in (Herlocker, Konstan, Terveen, & Riedl, 2004), together with guidelines concerning which metrics to use depending on the user's task.

For single-prediction tasks, where the user is asking for, or is given a single prediction, perhaps together with the description of an item, a good choice of metric is to calculate the deviation of the prediction from the true user rating. This is the type we will consider in this thesis; metrics that have no parameters, making the results easy to interpret and understand.

As with any retrieval or filtering system, the success of a collaborative filtering system depends partly on the accuracy of the predictions, but this is not the only factor; the factors listed in the previous section should very well be applicable to collaborative filtering as well.

Mean Absolute Error

When predicting the rating for a single item, it is natural to measure the difference between the predicted rating and the actual rating. By assuming that the error is equally small or large regardless of whether the prediction is pessimistic (the predicted rating is lower than the actual one) or optimistic (predicted higher than the actual), one may use the absolute difference or error. The absolute difference is then averaged over all hold out items j for each user, and then averaged over all users. The Mean Absolute Error, MAE, for a set of L users is

$$\text{MAE} = \frac{1}{L}\sum_{a=1}^{L}|p_{a,j} - r_{a,j}| \tag{2.41}$$

where $p_{a,j}$ is the predicted rating for item j for user a, and $r_{a,j}$ is the user's observed rating.

Root Mean Squared Error

Another predictive accuracy metric is the Root Mean Squared Error, RMS, which calculates the squared difference between a prediction and the actual rating. This function penalises large errors, since the error increases by the square of the difference instead of, as in MAE, the linear difference between prediction and rating. RMS is calculated for a set of L users as

$$\text{RMS} = \left[\frac{1}{L}\sum_{a=1}^{L}(p_{a,j} - r_{a,j})^2\right]^{1/2} \tag{2.42}$$

The CF community does not nearly have as many or as diverse data sets as the information retrieval or machine learning communities. In the experiments in Chapters 5 and 6, we have used two different publicly available data sets with explicit ratings in the movie domain; EachMovie and MovieLens. These have been extensively used in the literature. For implicit ratings, we know of one public data set (MSWeb) with implicit ratings of web page groups, based on web site browsing. Other data sets have also been used in the literature; some which have unfortunately not yet been publicly released and some that are too small for the experiments in this thesis.

The reason that there are so few data sets, is perhaps first of all due to it being an expensive task to build up a service, maintain a community, collect enough data (Herlocker et al., 2004), and difficult to compete with commercial interests. The research field is also relatively young; the first influential paper was published in 1994.

2.6.4 Cross Validation

A very important issue in the evaluation of machine learning, collaborative filtering and information retrieval algorithms is to separate data that is used for *training* the algorithm and the data that is used for *testing* or evaluating the algorithm. If we would use the same data for training and testing, we would get an overly optimistic estimate of the performance of the algorithm on new data.

	CISI	CACM	CRAN	REUTERS
Number of documents	1460	3204	1398	21,578
Number of queries	112	64	225	N/A
Number of categories	N/A	N/A	N/A	90
Avg. relevant docs/query	41.0	15.3	8.2	N/A

Table 2.1: Data sets for the Text Retrieval and Categorisation experiments

To get a fairly unbiased estimate of the error it is common to evaluate the algorithm several times on a given data set, using different divisions (splits) of the data into training and test sets. A method that is widely used and often recommended is *10-fold cross validation*; the data set is first randomly shuffled and then divided into 10 approximately equal-sized parts. Each part, corresponding to about 10% of the original data, is selected once as test set, while the remaining nine parts are used for training. Every example is thus part of a test set exactly once, and the algorithm is repeated 10 times over approximately 90% of the data each time.

Another, related, strategy, is *leave-one-out cross validation*, which is the extreme case of n-fold cross validation. If there are n examples, the algorithm is trained with $n-1$ training examples and one test example, and this procedure is repeated for all n examples.

For some research data sets, the data is already split into a training set and a test set, so that algorithms can be compared on this particular division of the data. One problem with this approach is that researchers may spend much time creating algorithms that are very well suited for a particular data set and split, but has little general merit.

2.6.5 Data sets

We now turn to the data sets that will be used in the experiments. For text retrieval and text categorisation, we will encounter a total of four data sets, whose statistics are summarised in Table 2.1. The CISI, CACM and CRAN data sets were taken from the Glasgow IR group[3]. CACM contains titles and abstracts from Communications of the ACM, CISI contains information science documents, CRAN contains a collection of documents on aerodynamics. The Reuters-21578 data set[4] contains a collection of 21578 categorised newswire stories collected in 1987, and is a widely used data set for document categorisation.

[3] `ir.dcs.gla.ac.uk`

[4] `www.daviddlewis.com/resources/testcollections/reuters21578/`

	EachMovie	MovieLens
Number of users	61265	6040
Number of items	1623	3706
Number of ratings	2811718	1000209
Rating scale	[0 . . . 5]	[1 . . . 5]
Global mean rating	3.037	3.582

Table 2.2: Data sets for the Collaborative Filtering experiments. Users with only one rating are not included.

For collaborative filtering, we selected the two large movie data sets, which differ somewhat in their characteristics. In Table 2.1, we list the relevant characteristics of these data sets. The MovieLens data set[5] is maintained by the GroupLens research group. The EachMovie data set was earlier maintained by HP/Compaq but as of October 2004, HP retired the EachMovie dataset and it is no longer available for download.

[5] `www.grouplens.org`

Chapter 3

Learning from Relevance Feedback

In several information retrieval systems there is a possibility for user feedback. Many machine learning methods have been proposed that learn from the feedback information in a long-term fashion. In this chapter, we present an approach that builds on user feedback across multiple queries in order to improve the retrieval quality of novel queries. This allows users of an IR system to retrieve relevant documents at a reduced effort.

Two algorithms for long-term learning across multiple queries in the scope of LSI have been implemented in order to test these ideas. The algorithms are based on the Nearest Neighbour Algorithm and Backpropagation Artificial Neural Networks introduced in Section 2.5.2 and 2.5.3. Training examples are query vectors, and by using LSI, these examples are reduced to a fixed and manageable size.

In order to evaluate the methods, we performed a set of experiments where we compared the performance of LSI and the proposed methods. The results demonstrate that the methods can automatically improve on the performance of LSI by using the feedback information from past queries.

3.1 Problem Background

The performance of LSI and other similar retrieval systems can usually be further improved by applying a technique known as relevance feedback (Harman, 1992; Dumais, 1991). This semi-automatic technique requires a user to explicitly evaluate the relevance of retrieved documents to supply as feedback to the system. Relevance feedback may be seen as a short-term learning mechanism, where the valuable feedback information is lost when the user starts a new query.

Several methods have been proposed to make IR systems learn from relevance feedback (Chen, 1995; Crestani, 1993; Vogt, Cottrell, Belew, & Bartell, 1997).

One problem in this domain is imposed by the high dimensionality of the feature space; the feature space grows linearly with the number of terms. The approach we use here is to reduce the dimensionality of the feature space by the SVD.

The algorithms are based on the hypothesis that queries which are similar in the LSI representation have similar result sets. From this hypothesis, we develop two learning algorithms, based on techniques from nearest neighbour searching and Backpropagation neural networks.

3.1.1 Relevance Feedback

The relevance feedback process may be viewed as an iterative dialogue between the user and the system, initiated by the user when posing a query. The query is evaluated and the system responds with a ranked list of documents which, according to the system's ordering, best match the query. If the user is not satisfied, the user may mark one or several documents as relevant and ask the system to refine the search. Again, the system produces a ranked list of documents and this iteration continues until the user stops interacting with the system, either because the user was content or gave up.

A common way to implement relevance feedback is to construct a query vector which is closer to the relevant, and further from the non-relevant documents as specified by the user. This can be seen as trying to construct an *optimal query*, i.e., one which separates the relevant documents from the non-relevant ones. There are several ways of implementing relevance feedback, for example, (Robertson & Sparck-Jones, 1976; Rocchio, 1971; Salton & Buckley, 1990). We focus on the technique most suited for our work.

One effective method to implement relevance feedback in LSI is to heuristically approximate the optimal query vector. This is carried out by replacing the query vector with the centroid of the vectors representing the relevant documents (Dumais, 1991). This method is defined by formula 3.1. In the formula, the approximation of the optimal query (hereafter called *improved*) is denoted q' whereas the set R' contains the vectors d_i for the documents determined relevant by the user.

$$q' = \sum_{d_i \in R'} \frac{d_i}{|R'|} \tag{3.1}$$

3.1.2 LSI for Query Representation

By using LSI for representing the queries and the documents, we hope to be able to compare queries in a way that makes more sense than if we would have used

the VSM model. In the LSI model, the queries need not share exact terms, rather share terms that are related in the sense of the SVD.

Recall that an optimal query is one which separate relevant documents from non-relevant ones. Raghavan and Sever (1995) investigated how to reuse past optimal queries in VSM. The purpose of their work was not to create an automatic learning system, but rather to speed up the execution of novel queries, by reusing past queries. For the purpose of reusing past optimal queries, they investigate similarity measures between queries, and demonstrate that comparing VSM query vectors may be misleading. Their suggestion was to base the comparison on the result lists instead.

That query vector similarity measures in VSM may be misleading suggests that the problem is the sparseness and high dimensionality of the vector space. Our hypothesis here is that queries which are similar in the LSI representation have similar result sets.

3.1.3 Previous Work

The idea of using relevance feedback for training IR systems to perform better is not new and the few articles we describe here is by no means all work in this area. Instead, we present the articles most relevant to our work.

Optimising system parameters

Our aim is to have the system remember, and ideally generalise from, the relevance feedback judgements. This can be viewed as trying to optimise the system parameters which govern the search. System parameters may be term weights, similarity functions, retrieval thresholds etc. Strategies to optimise these parameters may be divided in *implicit*, *exhaustive* and *heuristic* methods (Bartell, Cottrell, & Belew, 1998).

The implicit methods try to improve the system's ability to rank documents but do not have as primary goal to optimise the system parameters. The exhaustive methods, on the other hand, search the entire space of possible values of system parameters and then choose the best setting. This is generally an impractical method, since the number of parameters and their possible values are usually large. Finally, the heuristic methods perform a heuristic search of a subset of the possible parameters and values. These methods are not guaranteed to find the optimal parameters but are usually efficient and applicable to a large class of system parameters.

The methods we propose can be classified as implicit. That is, we do not try to optimise the actual system parameters. Rather, we generalise from a set

of previously posed queries, if they are similar enough to a new query. This is discussed in Section 3.2.1 and 3.2.2.

Neural networks

Crestani (1993) developed a neural network relevance feedback mechanism, in scope of the probabilistic model. A neural network simulator based on the Back-propagation algorithm was used for three different types of learning. The network was constructed so that the network input and output were binary vectors, where the input vectors corresponded to query terms and the output vectors corresponded to document terms.

The first learning type, Total Learning(TL), aimed at teaching the system application domain knowledge, where an example consisted of a query and one of its relevant documents. For queries with a small number of relevant documents, the results were good but gradually got worse as the number of relevant documents increased.

Horizontal Learning (HL) aimed at resolving the problems encountered in TL. Each example consisted of a query and the centroid of the query's relevant documents. This led to better results than those obtained by TL, but was still lower than that of by using the IR system.

Vertical Learning (VL), similar to TL, used only a portion of the relevant documents. This improved the results, and performed better than the IR system.

Syo, Lang and Deo (1996) incorporated LSI in a competition-based neural network. The purpose of the network was to capture associations between terms and documents, and the associations were then used to retrieve documents on the basis of a query. The network was originally trained using thesaurus associations, but in their experiments they also used LSI to capture the associations between terms and documents. They compared the LSI approach to the thesaurus approach, and found that the LSI associations improved the overall retrieval effectiveness of their model.

Gradient descent

A general approach to improve the effectiveness of IR systems was described by Bartell in his Ph.D. thesis (Bartell, 1994). Bartell developed a criterion, called J, which measures how well an IR system's ranking corresponds to the user preference. The J formula involves a set of IR system parameters, θ, such as similarity function parameters, retrieval thresholds, term weighting functions etc. If the first-order derivative of θ is available then it is possible to differentiate J with respect to θ. This means that for example gradient descent can be used to find the set of parameters which maximises J. In (Bartell et al., 1998), this technique

is used to optimise the cosine similarity measure in VSM. The optimised measure equalled or outperformed standard VSM measures. This is an example of a *heuristic* method to optimise system parameters.

Another experiment with the J criterion is described in (Vogt et al., 1997) where gradient descent was used to optimise the system's term weights, i.e., the term-document matrix. In order to make the matrix more manageable, LSI was used to compress the representation. Experiments verified that, in the *ad hoc* task, this technique improved performance on the training set, while not degrading performance on the test set.

Statistical methods

In (Hull, 1994), the LSI model was for the routing tasks. The first task in routing is to use relevance feedback information from one text collection to a new, unseen collection of text. The second task is to find the remaining relevant documents in a collection, given a query and a set of relevant documents. LSI was used to construct local factors for each query, based on the set of relevant documents. Discriminant analysis was then used to rank documents according to their probabilities of relevance.

3.2 Learning Algorithms

We propose two algorithms for long-term learning from relevance feedback in LSI, which both take as input an LSI query vector and produce a new, optimised query vector for retrieval. We base our algorithms on regression, as opposed to classification. That is, we do not assign each input to one of a number of discrete classes. The input is instead mapped to a new query vector in the LSI space.

LSI improves the initial query by replacing it with the centroid of the vectors representing the relevant documents. At this point, the system has gathered much information: the initial query in text format, the initial query vector, the list of relevant documents, the improved query vector, the text in each of the relevant documents etc. The learning algorithm makes use of only the initial query vector q and the improved query vector q'. Each training example is thus on the form $\{q, q'\}$.

The two algorithms that we propose will be called Nearest Neighbour Regressor (NNR) and Backpropagation Regressor (BPR). The first is based on the KNN algorithm 2.5.2 and the other on a back propagation neural network 2.5.3. The learning problem is modelled in a similar way in both algorithms, so that it is feasible to compare the experimental results. We use the term *optimise* for

the process of generalizing from previously posed queries to improve the performance of a new, unseen query.

3.2.1 Nearest Neighbour Regressor

NNR uses a k-nearest neighbour method which is slightly similar to locally weighted regression. There are however some differences, the first is that the 'nearness' is measured by the cosine of the angle. The second is the way we weigh the contribution of the nearest vectors.

During the training phase, the examples are simply stored. Each example is a tuple (q, q') where q is the initial query vector and q' the improved query vector.

When the user poses a new query, NNR performs a nearest neighbour search with the new query vector against all stored initial query vectors. If there is one or more initial query vectors which are sufficiently 'near' the new query, we have the hit case. Otherwise, we have the miss case and the new query is used as is. In this second case, no optimisation is performed.

When the hit case occurs, NNR calculates the optimised query in three steps. The vectors of the k pairs of initial and improved query vectors are first normalised to unit length, as is the new query vector. The centroid of the initial and the centroid of the improved query vectors are then calculated. To obtain the optimised query vector the first centroid is subtracted from the new query vector and the second centroid is added to it.

More formally, let k be the number of neighbours to the new query. Denote the new query vector v, the initial query vectors q_i and the improved query vectors q_i'. The optimised query vector v' is calculated by:

$$v' = \frac{v}{|v|} - \frac{1}{k}\sum_{i=1}^{k}\frac{q_i}{|q_i|} + \frac{1}{k}\sum_{i=1}^{k}\frac{q_i'}{|q_i'|} \tag{3.2}$$

Note that formula 3.2 has a desirable property when $k = 1$: if the new query is identical to a previous initial query, it will be replaced by its relevance feedback query. When the new query is similar to one or more previous initial query vectors, a combination of the previous initial and improved query vectors is added to it. This combination favours the direction of the improved vectors if the new query vector is close to the nearest neighbours. The resulting vector points more in the direction of the improved queries, and the amount of the change in direction is determined by the difference between the new query and its nearest neighbours.

The vectors are normalised to unit length because it is the angular position which is of interest. If they are not normalised, some vectors may swamp the contribution of others. Furthermore, NNR will not try to optimise every query,

the hit case will only occur if there are any vectors in the training set which are sufficiently similar to the new query. A threshold value is used to determine if two vectors are sufficiently similar, a way to determine if there is enough information in the training set to generalise from.

3.2.2 Backpropagation Regressor

The BPR algorithm is based on a combination of the Backpropagation algorithm and NNR. When training the network, each example is composed by the initial query vector as the input, and the difference between the improved query vector and the initial query vector as the target. Using the terminology from the previous section, the network input vector is q and the network target vector is $q' - q$. Thus, we train the network to learn the difference between an improved query and a new query, given the new query.

If there is no information in the network regarding a new query, it is not reasonable to demand generalisation from the network. To alleviate this problem, we set up a criterion which must be obeyed by a new query if it is to be run through the network. The criterion is that there should be at least one example in the training data which is sufficiently similar to the new instance, using cosine as similarity measure.

When given a new query, an exhaustive nearest neighbour search is performed in the training set. If there is at least one query vector which is similar to the new query, it is run through the network, otherwise the query vector is used as is.

After the new query is run through the network, it is optimised in two steps. First, both the new query and the network output are normalised to unit length. Secondly, the new query is added to the network output to form the optimised query.

Using vector notation, let the new query vector be v and the network output o. The optimised query v' is then constructed by

$$v' = \frac{v}{|v|} + \frac{o}{|o|} \tag{3.3}$$

The input vectors are normalised in the range $(-1.0, 1.0)$ whereas the target vectors are normalised in the range $(0.1, 0.9)$. The input vector normalisation is necessary to improved training time, since the vector values are typically rather small. The target vector normalisation is necessary since we use a bounded activation function (the sigmoid function).

Collection	Targets	Top cosine
CISI	5	0.77
CACM	16	0.98
CRAN	55	0.97

Table 3.1: Collection analysis

3.3 Experimental Evaluation

In order to investigate the effectiveness of the two methods, we performed a series of experiments. The learning algorithms were implemented in C++ and we used an implementation of LSI provided by Telcordia (formerly Bellcore).

3.3.1 Setup

The CACM, CISI and CRAN collections were used in these experiments. Each collection consists of a set of documents, a set of queries and a list describing which documents are considered relevant for each query. Table 2.1 provides a summary of the collection statistics.

After some initial testing, it became clear that the algorithms did not improve the retrieval for the CACM and CISI collections. In fact, in only very few cases were the algorithms able to optimise queries from those collections. This was due to the very small amount of queries with sufficiently high cosine similarity measure. Table 3.1 displays the results of this analysis. The second column shows the number of possible targets for optimisation, the third shows the top cosine similarity measure found. Consequently, we decided not to use the CACM and CISI collections, since the training sets turned out to be too small. The CRAN collection however, turned out to be more useful for an experimental evaluation of our algorithms.

For the CRAN collection, the query set was divided into two randomly selected subsets of equal size. The first subset was used for training and the second for testing. This process was repeated seven times.

Since the purpose of the algorithms is to optimise new queries on the basis of a set of stored queries, only the test data is subject to evaluation. Since we use a threshold value to determine which queries should be subject for optimisation, only these particular queries are included in the evaluation. In Table 3.2 we list the number of optimised queries for each threshold value. The third column in the table lists the average number of queries subject for optimisation. The last column lists the percentage of the optimised queries in the test data set.

Collection	Threshold	Optimised	Percentage
CRAN	0.7	15.4	13.8
CRAN	0.8	8.8	7.9

Table 3.2: Optimisation statistics

At first we used all relevant documents to construct the approximation of the optimal queries, as described in (Dumais, 1991). However, we noticed that this did not yield the best performance. The results improved further when we used the top (1-5) relevant documents from the result set of the approximation of the optimal query. The improved queries were constructed using this method.

The NNR algorithm was evaluated using two different threshold values: 0.7 and 0.8. After some experiments, we found that values below 0.7 caused the algorithm to make incorrect judgements with respect to the similarity between queries while values higher than 0.8 caused very few queries to be optimised.

For each threshold value, we set the maximum number of neighbours first to 1 and then to 3, making a total of four experiments. It should however be noted that not all queries had 3 neighbours above the threshold.

The BPR algorithm was evaluated using the same threshold values (0.7 and 0.8) as for NNR. We performed some initial tests to find a reasonable set of parameter values for the network. First of all, we saw that even though we did not have many examples, we needed more nodes in the hidden layers than in the input or output layers. This should both be due to the fact that the function we wish to approximate is fairly complex and that the network output should be an approximation of real values.

When we used more than one hidden layer the convergence of the network was very slow due to the fact that the network needed many epochs to recover from local minimum. We found that one hidden layer of 200 nodes proved fairly good, both in terms of running speed and results. For the CRAN collection LSI compressed the term, document and query vectors to 113 dimensions, why we used 113 nodes in the input and output layers of the network.

We experimented with different values on learning rate and momentum constants, and noticed that values closer to 0.0 than to 1.0 were to prefer. We set both the learning rate and the momentum to 0.2, which made the network slowly converge to a small error on the test set.

Another problem we faced during this experiment was that the stop condition for the neural network was not easy to determine. Therefore, we decided to study the behaviour of two different networks. The networks were trained for 10000 epochs each, and we listed the network error at each epoch. Figure 3.1 plots

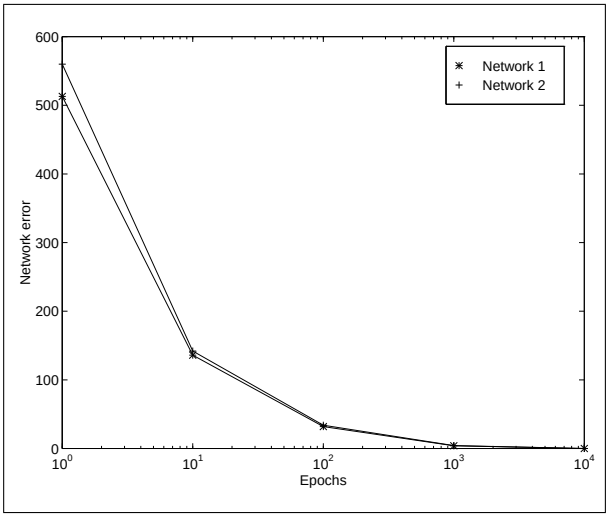

Figure 3.1: Network error versus number of epochs

the network error versus the number of epochs (on a logarithmic scale). Both networks follow the same pattern; a high initial error which quickly gets smaller, and approximately the same error after 1000 epochs. In the figure, the error after 1000 epochs is approximately 4.0, whereas the error at epoch 10000 is close to 0.0001. We saw that the difference in average precision between using the network after 1000 and after 10000 epochs was approximately ± 0.02. In light of this, we decided to evaluate the networks after 1000 epochs.

3.3.2 Results

We present the main results as precision-recall graphs in Figures 3.2 and 3.3. The graphs were constructed by calculating the eleven-point precision-recall curve described in Section 2.6.1.

In Figure 3.2 we used a threshold value of 0.7, in Figure 3.3 we used 0.8. NNR(1) and NNR(3) refer to the Nearest Neighbour Regressor algorithm using 1 and max 3 neighbours, whereas BPR refer to the Backpropagation Regressor method. The results obtained by using no optimisation at all (LSI) are also included. The four curves (LSI, BPR, NNR(1) and NNR(3)) are the results of using completely *automatic* methods, and they show the performance of the initial user query.

For comparison, we have also included the maximal precision that can be obtained by the relevance feedback mechanism (RF) described in Section 3.1.1. Recall that this performance is based on perfect information about all 1398 documents returned by a query. In reality however, a human user will be able to give

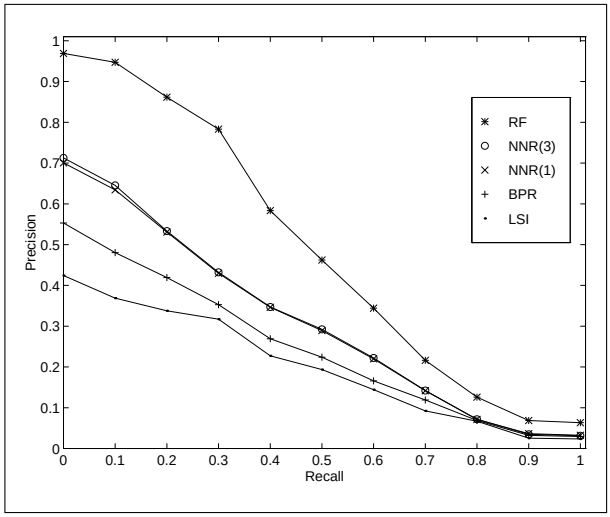

Figure 3.2: Precision/Recall using threshold 0.7

an accurate relevance score of only 5 or 10 of the top ranked documents. Hence, performance using relevance feedback for an ordinary user, would be more comparable with that of NNR or BPR (depending on the number of documents that the user has the patience to rate). The NNR and BPR methods on the other hand, improve on the original query automatically, based on their knowledge about the relevance of (similar) past queries.

From the figures, it is clear that the other methods outperform the LSI search technique. Furthermore, the nearest neighbour method also performs better than the Backpropagation method.

The difference between using 1 neighbour and max 3 neighbours is very small. We noticed that 3 neighbours were only used twice (1.8% of all optimised queries) in the test runs with threshold 0.8. 2 neighbours were only used 11 times (10.1%). The test data was therefore not very much clustered, in terms of the cosine measure.

The figures show only a small difference between using 0.7 and 0.8 as threshold value, in terms of effectiveness. But using 0.7 caused twice as many queries to be optimised, without considerably degrading the results.

It is interesting to see that the nearest neighbour method performs much better than the Backpropagation method. This can be due to a number of reasons, perhaps the most likely that we were incapable of finding the best network topology. There is also an error source involved in the network algorithm; the test data vectors are normalised according to the minimum and maximum vector values in the training set.

In terms of effectiveness, the other methods performed better than the origi-

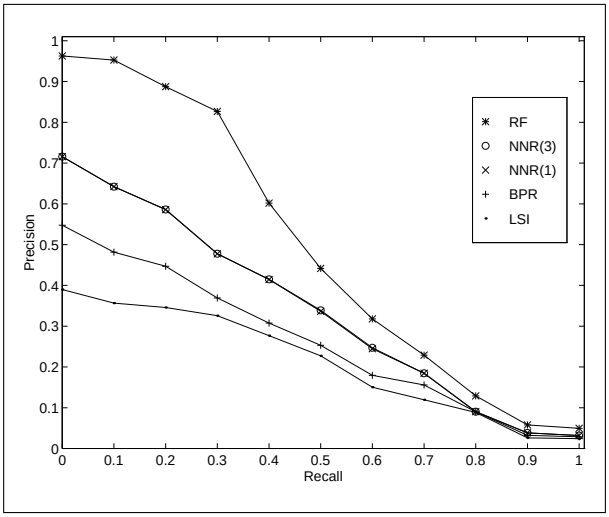

Figure 3.3: Precision/Recall using threshold 0.8

nal LSI query. However, the number of optimised queries was rather small. The knowledge used from previous queries is also very specific (or vertical) since most of the time one neighbour was used for generalisation. The results also indicate that the less complex solution (nearest neighbour) is capable of better results than the complex one (neural network).

3.4 Discussion

We have presented results regarding two learning algorithms for improving previously unseen user queries in LSI.

The proposed methods are based on two different learning schemes from the area of machine learning. The first method, Nearest Neighbour Regressor, uses a nearest neighbour search of the training set to find queries similar to the initial query. If found, the initial query is then optimised by changing its direction to be more similar to that of the improved queries.

The second method, Backpropagation Regressor, implements a back propagation neural network and is trained to learn the difference between an improved query and its initial query. When a new query is found similar to at least one of the queries in the training set, it is optimised by adding the network output to itself.

In order to evaluate the methods, we retrieved three standard IR test collections. We found that there were only a few queries in the first two collections with high similarity values, according to the similarity measure we chose. We

evaluated the methods on the third test collection, using a standard measure of effectiveness.

The experiments revealed an array of results. First, we found that the methods improved new queries considerably. About 15% of the queries in the test set were improved. Secondly, we found that the nearest neighbour method outperformed the Backpropagation neural network. This, we believe, was mainly because it was difficult to find the optimal parameters for the network. We also found that using the 1-5 top-ranked documents from the result list of the improved query during training generally increased the effectiveness of both methods. It thus seems like clustered documents are preferred as training examples by our algorithms.

We found that the long-term relevance feedback learning mechanism was well suited for a particular type of queries. In the CRAN collection, where the algorithm improved the search results, the queries are long, and there are many relevant documents for each query. In every other collection we tried (besides the three datasets discussed here), the queries were short and there were few relevant documents per query.

In general, we can expect that the methods are best suited in applications where the queries are as elaborate (long) as they are in the CRAN collection. For shorter queries, such as web queries, the situation is most probably reversed, since the probability of two randomly selected queries to be similar should be higher when the queries are short. Our conclusion is therefore that for all practical purposes we should, in the LSI model, use the KNN method to improve future queries, on the basis of past queries and their relevance judgements, if the queries are more elaborate than a few query terms.

The data collections were not particularly suited for this problem; the algorithm is designed to learn from past similar queries, but in the data sets there were not many similar queries to learn from. The collections are not explicitly constructed for experiments involving query similarity, they are constructed for experiments in *ad hoc* retrieval. We considered the possibility of creating a new collection that instead had several similar queries, but this required resources we did not have.

In conclusion, we have shown that relevance feedback in Latent Semantic Indexing can be improved by using information implicitly represented in feedback from previous queries. Most techniques for relevance feedback today use information that is collected from the user at the time of posing new queries to an information retrieval system. This is often time-consuming and requires an unnecessarily big effort from the user. It also means that collected information is lost at the end of a query session. This method instead uses knowledge about the user's preferences as they have appeared in earlier sessions, to improve on the current search session.

3.4.1 Future Work

An important issue for future work is to explore the usefulness of the methods in the scope of much larger text collections. The critical factor would be whether the SVD can be computed for the collection or not, and if the vector dimensions then are small enough.

The dimension of a query vector in LSI is equal to the number of singular values one chooses in the SVD operation. Tests with LSI at the TREC conference (Dumais, 1995) have demonstrated that even with collections of hundreds of thousands of documents, only a few hundred singular values are required for good performance. Thus, it is reasonable to believe that for collections with more documents there will not be a tremendous increase in the number of dimensions.

Algorithms for computing the truncated SVD are getting faster all the time. However, the complexity of the current algorithms is fairly high. On the other hand there is no need to compute the truncated SVD of the entire term-document matrix. A carefully selected sample of the document set can be used to construct the LSI space. The rest of the document set can then be *folded-in* to the space (Berry, Dumais, & Brien, 1995), in a manner similar to how the query vector is represented.

Another interesting direction for future work is to investigate how the proposed methods can be modified to work for a group of users with similar frames of reference and interests. In such a scenario the common feedback from all users in the group will be available to optimise new queries. This is a scenario where individual efforts (in terms of giving feedback to the system) will be minimised while the benefits (in terms of being presented less irrelevant documents) will be maximised.

Chapter 4

Random Indexing for Text Categorisation

In this chapter we investigate the use of concept-based representations for text categorisation. The concept-based text representations are created using Random Indexing (Kanerva, Kristofersson, & Holst, 2000; Karlgren & Sahlgren, 2001), for the purpose of investigating whether this representation holds any valuable properties for text categorisation tasks. We use the Support Vector Machine classifier, described in Section 2.5.4, for learning the categories.

The results of the experiment show a small difference between the traditional bag of words representations and the concept based ones, but more specifically we identify certain categories for which we can use concept-based representations as a supplement or complementary representation. To further analyse this, we calculate the optimal performance, using the $\mathcal{F}_1$ measure from Section 2.6.2, obtained by the classifier when we select the overall best combination of representation for the different categories. This combination contains both categories learned with the bag-of-words representation as well as the concept-based ones.

This chapter has previously been published in (Sahlgren & Cöster, 2004), with Sahlgren as the primary author. Sahlgren is the primary investigator of the Random Indexing approach to text representation, and the cooperation here concerns the use of Random Indexing for text categorisation.

4.1 Problem Background

For text categorisation (TC), the papers by Joachims (1998) and Dumais et. al. (1998) demonstrated that Support Vector Machines applied to a Bag-of-Words (BoW) representation of documents is a very effective approach.

Other representations have been used for the TC task; n-grams and phrases

(Lewis, 1992; Dumais et al., 1998), the use of synonym clusters and latent dimension (Baker & McCallum, 1998; Cai & Hofmann, 2003). However, none of the more elaborate representations seem to significantly outperform the standard BoW approach (Sebastiani, 2002). In addition to this, they are typically more expensive to compute.

What interests us here is the difference between using standard BoW and more elaborate, concept-based representations. Since text categorisation is normally cast as a problem concerning the *content* of the text (Dumais et al., 1998), one might assume that looking beyond the mere surface word forms should be beneficial for the text representations. We believe that, even though BoW representations are superior in most text categorisation tasks, concept-based schemes *do* provide important information, and that they can be used as a supplement to the BoW representations. Our goal is therefore to investigate whether there are specific categories in a standard text categorisation collection for which using concept-based representations is more appropriate, and if combinations of word-based and concept-based representations can be used to improve the categorisation performance.

In order to do this, we introduce a new method for producing concept-based representations for natural language data. The method is efficient, fast and scalable, and requires no external resources. We use the method to create concept-based representations for a standard text categorisation problem, and we use the representations as input to a Support Vector Machine classifier. The categorisation results are compared to those reached using standard BoW representations, and we also demonstrate how the performance of the Support Vector Machine can be improved by combining the representations.

4.2 Bag-of-Concepts

The standard BoW representations are usually refined before they are used as input to a classification algorithm. One refinement method is to use *feature selection*, which means that words are removed from the representations based on statistical measures, such as document frequency, information gain, χ^2, or mutual information (Yang & Pedersen, 1997). Another refinement method is to use *feature extraction*, which means that "artificial" features are created from the original ones, either by using clustering methods, such as distributional clustering (Baker & McCallum, 1998), or by using methods such as the SVD.

As discussed earlier, feature extraction methods also handle problems with synonymy, by grouping together words with similar meaning, or by restructuring the data (i.e. the number of features) according to a small number of informative dimensions, so that similar words get similar representations. Since these

methods do not represent texts merely as collections of the words they contain, but rather as collections of the *concepts* they contain – whether these be synonym sets or latent dimensions – a more fitting label for these representations would be *Bag-of-Concepts* (BoC).

4.3 Random Indexing

One serious problem with BoC approaches is that they tend to be computationally expensive. This is true at least for methods that use factor analytic techniques. Other BoC approaches that use resources such as WordNet[1] have limited portability, and are normally not easily adaptable to other domains or to other languages.

To overcome these problems, we use an alternative approach for producing BoC representations. The approach is based on *Random Indexing* (Kanerva et al., 2000; Karlgren & Sahlgren, 2001), which is a vector space methodology for producing *context vectors* for terms based on cooccurrence data. A context vector is a vector that represents the contexts in which a term occurs. Random Indexing uses these context vectors to produce a Bag-of-Concept representation, for example for a document, by summing the context vectors for every term that occurs in the document.

To construct the context vectors, Random Indexing does the following. Each context (for example each document, each paragraph, each clause, or each window of terms) is first assigned a unique and randomly generated representation in the form of an *index vector*. The index vector is a high-dimensional, sparse representation of the context. The dimensionality, k, is typically in the order of thousands. The vector is sparse: it contains only a small number of non-zero elements, which are either -1 or $+1$. Thus, the values of the index vectors are ternary: they are either -1, 0 or $+1$.

Each term is then given a high-dimensional vector of the same dimensionality as the index vectors. Random Indexing proceeds by scanning the text sequentially from the beginning, and whenever a term occurs in a context (for example in a document) then the index vector for that context is added (by vector addition) to the term's high-dimensional vector. These term vectors are called *context vectors* since they are effectively the sum of the terms' contexts.

Random Indexing thus builds a context matrix G of order $w \times k$, where $k \ll c$. Each row G_i is the k-dimensional context vector for term i. The context vectors are, as described, accumulated by adding together the k-dimensional index vectors that have been assigned to each context in the data – whether

[1] `wordnet.princeton.edu`

document, paragraph, clause, window, or neighbouring terms.

Note that the same procedure will produce a standard cooccurrence matrix F of order $w \times c$ if we use unary index vectors of the same dimensionality c as the number of contexts. These unary index vectors would have a single 1 marking the place of the context in a list of all contexts – the nth element of the index vector for the nth context would be 1. Mathematically, the unary vectors are orthogonal, whereas the random index vectors are only *nearly* orthogonal. However, since there are more nearly orthogonal than truly orthogonal directions in a high-dimensional space, choosing random directions gets us sufficiently close to orthogonality to provide an approximation of the unary vectors (Hecht-Nielsen, 1994).

The distribution of the $+1$ and -1 values is the key to constructing the index vectors so that they are *nearly* mutually orthogonal. The near orthogonality of random directions in a high-dimensional vector space is the core part of a family of dimension reduction techniques such as Random Mapping(Kaski, 1998), Random Projections(Bingham & Mannila, 2001), and Random Indexing. These are all motivated by the Johnson-Lindenstrauss Lemma (Johnson & Lindenstrauss, 1984), which states that if we project points into a randomly selected subspace of sufficiently high dimensionality, the distances between the points are approximately preserved. Thus, if we collect the random index vectors into a random matrix R of order $c \times k$, whose row R_i is the k-dimensional index vector for context i, the following relation holds:

$$G_{w \times k} = F_{w \times c} R_{c \times k}$$

That is, the Random Indexing context matrix G contains the same information as we get by multiplying the standard cooccurrence matrix F with the random matrix R, where RR^T approximates the identity matrix.

4.3.1 Bag-of-Context vectors

The context vectors produced by Random Indexing can be used to generate BoC representations. This is done by, for every text, summing the (weighted) context vectors of the terms that occur in the particular text. Note that summing vectors result in *tf*-weighting, since a term's vector is added to the text's vector as many times as the term occurs in the text. The same procedure generates standard BoW representations if we use unary index vectors of the same dimensionality as the number of terms in the data instead of context vectors, and weight the summation of the unary index vectors with the idf-values of the terms.

4.3.2 Advantages of Random Indexing

One advantage of using Random Indexing is that it is an *incremental* method, which means that we do not have to sample *all* the data before we can start using the context vectors — Random Indexing can provide intermediary results even after just a few vector additions. Other vector space models need to analyse the entire data before the context vectors are operational.

Another advantage is that Random Indexing avoids the "huge matrix step", since the dimensionality k of the vectors is much smaller than, and not directly dependent on, the number of contexts c in the data. Other vector space models, including those that use dimension reduction techniques such as singular value decomposition, depend on building the $w \times c$ cooccurrence matrix F.

This "huge matrix step" is perhaps the most serious deficiency of other models, since their complexity becomes dependent on the number of contexts c in the data, which typically is a very large number. Even methods that are mathematically equivalent to Random Indexing, such as random projection (Papadimitriou, Raghavan, Tamaki, & Vempala, 1998) and random mapping (Kaski, 1998), are not incremental, and require the initial $w \times c$ cooccurrence matrix.

Since dimension reduction is built into Random Indexing, we achieve a significant gain in processing time and memory consumption, compared to other models. Furthermore, the approach is scalable, since adding new contexts to the data set does not increase the dimensionality of the context vectors.

4.4 Experiment Setup

In the following subsections, we describe the setup for the text categorisation experiments.

4.4.1 Data

We use the Reuters-21578 test collection, which consists of 21,578 news wire documents that have been manually assigned to different categories. In these experiments, we use the "ModApte" split, which divides the collection into 9,603 training documents and 3,299 test documents, assigned to 90 topic categories. After lemmatization, stopword filtering based on document frequency, and frequency thresholding that excluded words with frequency < 3, the training data contains 8,887 unique terms.

4.4.2 Representations

The standard BoW representations for this setup of Reuters-21578 are 8,887-dimensional and very sparse.

To produce BoC representations, a k-dimensional random index vector is assigned to each training document. Context vectors for the terms are then produced by adding the index vectors of a document to the context vector for a given term every time the term occur in that document.

We initially also used word-based contexts, where index vectors were assigned to each unique word, and context vectors were produced by adding the random index vectors of the surrounding words to the context vector of a given word every time the word occurred in the training data. However, the word-based BoC representations consistently produced inferior results compared to the document-based ones, so we decided not to pursue the experiments with word-based BoC representations for these experiments. Our initial guess is that the heavy preprocessing by removing stop words and using frequency thresholding may have been the reason for the inferior results.

The context vectors are then used to generate BoC representations for the texts by summing the context vectors of the words in each text, resulting in k-dimensional dense BoC vectors.

4.4.3 Support Vector Machines

For learning the categories, we use the Support Vector Machine (SVM) algorithm for binary classification, see Section 2.5.4.

In our experiments, we use three standard kernel functions – the basic linear kernel, the polynomial kernel, and the radial basis kernel:[2]

- Linear: $K(\mathbf{x}_i, \mathbf{z}) = \mathbf{x}_i \cdot \mathbf{z}$
- Polynomial: $K(\mathbf{x}_i, \mathbf{z}) = (\mathbf{x}_i \cdot \mathbf{z})^d$
- Radial Basis: $K(\mathbf{x}_i, \mathbf{z}) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{z}\|^2)$

For all experiments, we selected $d = 3$ for the polynomial kernel and $\gamma = 1.0$ for the radial basis kernel. These parameters were selected as default values and are not optimised.

[2]We used a SVM implementation called *SVM*^{light} available at: `svmlight.joachims.org`

	tf	*idf*	*tf*×*idf*
BoW	82.52	80.13	82.77
BoC 500-dim	79.97	80.18	81.25
BoC 1,000-dim	80.31	80.87	81.93
BoC 1,500-dim	80.41	80.81	81.79
BoC 2,000-dim	80.54	80.85	82.04
BoC 2,500-dim	80.64	**81.19**	82.18
BoC 3,000-dim	80.67	81.15	82.11
BoC 4,000-dim	80.60	81.07	82.24
BoC 5,000-dim	**80.78**	81.09	**82.29**
BoC 6,000-dim	**80.78**	81.08	82.12

Table 4.1: Micro-averaged $\mathcal{F}_1$ score for *tf*, *idf* and *tf*×*idf* using BoW and BoC representations.

4.5 Experiments and Results

In these experiments, we use a one-against-all learning method, which means that we train one classifier for each category (and representation). For evaluation, we used the standard $\mathcal{F}_1$ measure, as defined by Formula 2.40.

There are a number of parameters that need to be optimised in this kind of experiment, including the weighting scheme, the kernel function, and the dimensionality of the BoC vectors. For ease of exposition, we report the results of each parameter set separately. Since we do not experiment with feature selection in this investigation, our results will be somewhat lower than other published results that use SVM with optimised feature selection. Our main focus is to compare results produced with BoW and BoC representations, and not to produce a top score for the Reuters-21578 collection.

4.5.1 Weighting Scheme

Using appropriate word weighting functions is known to improve the performance of text categorisation (Yang & Pedersen, 1997). In order to investigate the impact of using different word weighting schemes for concept-based representations, we compare the performance of the SVM using the following three weighting schemes: *tf*, *idf*, and *tf*×*idf*.

The results are summarised in Table 4.5.1. The BoW run uses the linear kernel, while the BoC runs use the polynomial kernel. The numbers in boldface are the best BoC runs for *tf*, *idf*, and *tf*×*idf*, respectively.

As expected, the best results for both BoW and BoC representations were produced using *tf*×*idf*. For the BoW vectors, *tf* consistently produced better results than *idf*, and it was even better than *tf*×*idf* using the polynomial and radial basis kernels. For the BoC vectors, the only consistent difference between *tf* and *idf* is found using the polynomial kernel, where *idf* outperforms *tf*.[3] It is also interesting to note that for *idf* weighting, all BoC runs outperform BoW.

4.5.2 Parameterising RI

In theory, the quality of the context vectors produced with the Random Indexing process should increase with their dimensionality. Kaski (1998) shows that the higher the dimensionality of the vectors, the closer the matrix RR^T will approximate the identity matrix, and Bingham and Mannila (2001) observe that the mean squared difference between RR^T and the identity matrix is about $\frac{1}{k}$, where k is the dimensionality of the vectors. In order to evaluate the effects of dimensionality in this application, we compare the performance of the SVM with BoC representations using 9 different dimensionalities of the vectors. The index vectors consist of 4 to 60 non-zero elements ($\approx$ 1% non-zeros), depending on their dimensionality. The results for all three kernels using *tf*×*idf*-weighting are displayed in Figure 4.1.

Figure 4.1 demonstrates that the quality of the concept-based representations increase with their dimensionality as expected, but that the increase levels out when the dimensionality becomes sufficiently large; there is hardly any difference in performance when the dimensionality of the vectors exceeds 2,500. There is even a slight tendency that the performance decreases when the dimensionality exceeds 5,000 dimensions; the best result is produced using 5,000-dimensional vectors with 50 non-zero elements in the index vectors.

There is a decrease in performance when the dimensionality of the vectors drops below 2,000. Still, the difference in $\mathcal{F}_1$ score between using 500 and 5,000 dimensions with the polynomial kernel and *tf*×*idf* is only 1.04, which indicates that Random Indexing is very robust in comparison to, e.g., singular value decomposition, where choosing appropriate dimensionality is critical.

4.5.3 Parameterising SVM

Regarding the different kernel functions, Figure 4.1 clearly shows that the polynomial kernel produces consistently better results for the BoC vectors than the other kernels, and that the linear kernel consistently produces better results than

[3] For the linear and radial basis kernels, the tendency was that *tf* in most cases was better than *idf*.

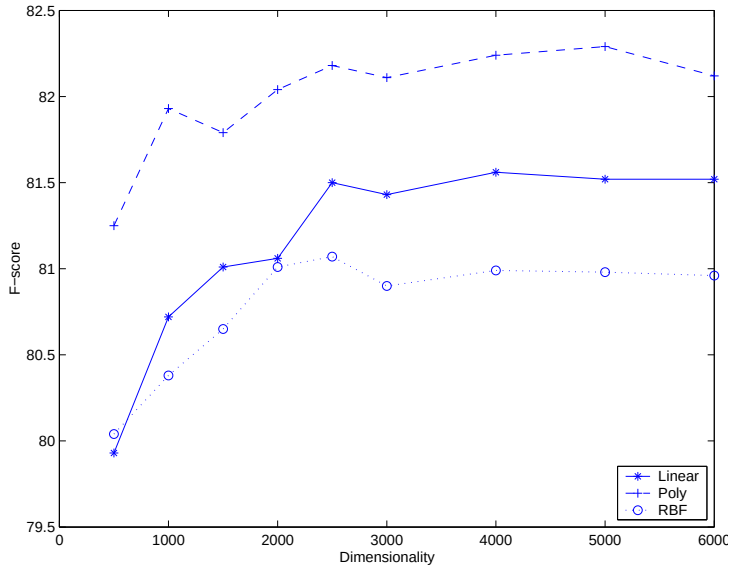

Figure 4.1: Micro-averaged $\mathcal{F}_1$ score for three kernels using 9 dimensionalities of the BoC vectors.

the radial basis kernel. This could be a demonstration of the difficulties of parameter selection, especially for the γ parameter in the radial basis kernel. To further improve the results, we can find better values of γ for the radial basis kernel and of d for the polynomial kernel by explicit parameter search.

4.6 Comparing BoW and BoC

If we compare the best BoW run (using the linear kernel and $tf \times idf$-weighting) and the best BoC run (using 5,000-dimensional vectors with the polynomial kernel and $tf \times idf$-weighting), we can see that the BoW representations barely outperform BoC: 82.77% versus 82.29%. However, if we only look at the results for the ten largest categories in the Reuters-21578 collection, the situation is reversed and the BoC representations outperform BoW. The $\mathcal{F}_1$ measure for the best BoC vectors for the ten largest categories is 88.74% compared to 88.09% for the best BoW vectors. This suggests that BoC representations are more appropriate for large-size categories.

The best BoC representations outperform the best BoW representations in 16 categories, and are equal in 6. Of the 16 categories where the best BoC outperform the best BoW, 9 are better only in recall, 5 are better in both recall and precision, while only 2 are better only in precision.

It is always the same set of 22 categories where the BoC representations score better than, or equal to, BoW.[4] These include the two largest categories in Reuters-21578, *"earn"* and *"acq"*, consisting of 2,877 and 1,650 documents, respectively. For these two categories, BoC representations outperform BoW with 95.57% versus 95.36%, and 91.07% versus 90.16%, respectively. The smallest of the "BoC categories" is *"fuel"*, which consists of 13 documents, and for which BoC outperforms BoW representations with 33.33% versus 30.77%. The largest performance difference for the "BoC categories" is for category *"bop"*, where BoC reaches 66.67%, while BoW only reaches 54.17%. We also note that it is the same set of categories that is problematic for both types of representations; where BoW score 0.0%, so does BoC.

4.7 Combining Representations

The above comparison suggests that we can improve the performance of the SVM by combining the two types of representation. The best $\mathcal{F}_1$ score can be achieved by selecting the quadruple $(\mathsf{TP}, \mathsf{FP}, \mathsf{TN}, \mathsf{FN})$ for each individual category from either BoW or BoC so that it maximises the overall score. There are 2^{90} such combinations, but by expressing the $\mathcal{F}_1$ function in its equivalent form $\mathcal{F}_1 = (2 * \mathsf{TP})/(2 * \mathsf{TP} + \mathsf{FP} + \mathsf{FN})$, we can determine that for our two top runs there are only 17 categories such that we need to perform an exhaustive search to find the best combination. For instance, if for one category both runs have the same TP but one of the runs have higher FP and FN, the other run is selected for that category and we do not include that category in the exhaustive search.

Combining the best BoW and BoC runs increases the results from 82.77% (the best BoW run) to **83.91**%. For the top ten categories, this increases the score from 88.74% (the best BoC run) to **88.99**%. Even though the difference is admittedly small, the increase in performance when combining representations is not negligible, and is consistent with the findings of previous research (Cai & Hofmann, 2003).

4.8 Discussion

We have introduced a new method for producing concept-based (BoC) text representations, and we have compared the performance of an SVM classifier on the Reuters-21578 collection using both traditional word-based (BoW), and concept-based representations. The results show that BoC representations outperform

[4]The "BoC categories" are: *veg-oil, heat, gold, soybean, housing, jobs, nat-gas, cocoa, wheat, rapeseed, livestock, ship, fuel, trade, sugar, cpi, bop, lei, acq, crude, earn, money-fx*.

BoW when only counting the ten largest categories, and that a combination of BoW and BoC representations improve the performance of the SVM over all categories.

We conclude that concept-based representations constitute a viable supplement to word-based ones, and that there are categories in the Reuters-21578 collection that benefit from using concept-based representations.

4.8.1 Future Work

The idea of extracting higher-level concepts from text is intellectually appealing, although to date the basic word-based representations often have better accuracy, are easier to implement and can be represented more efficiently. Research in concept-based representations has much to prove before it can replace this current paradigm. Random Indexing is one exciting new form of text representation that we believe should be developed further. The first issue that needs more research is perhaps a deeper investigation (mathematical or linguistic) of the extracted concepts in order to better understand how to tune the parameters of the method.

Chapter 5

Inverted Files for Collaborative Filtering

In this chapter we explore the possibility of using an inverted file structure for collaborative filtering. The hypothesis is that this structure allows for faster calculation of predictions and also that pruning strategies from information retrieval can be used to further speed up the filtering process and perhaps even improve the quality of the predictions.

There are two reasons for this. Firstly, matching user profiles in a collaborative filtering system can be very expensive. Secondly, if it is possible to use a structure that allows us to access all user preferences stored directly from disk it is possible to maintain a much larger set of users and items.

In order to evaluate this, a series of experiments were performed on several large datasets. The evaluation compare inverted file search with in-memory vector search, both in terms of prediction accuracy and neighbourhood formation time.

We have already reviewed Collaborative Filtering in Section 2.4. In Section 5.1 we express four correlation algorithms in a form suitable for inverted retrieval. The file organisation for implementing the inverted files is briefly described in Section 5.2. Sections 5.3 and 5.4 describe the experimental setup and the main results. The main parts of this chapter have previously been published in (Cöster & Svensson, 2002), although we have performed new experiments with new datasets and new parameters; the largest difference is that we now have performed a 10-fold cross validation for evaluating the algorithms.

5.1 Inverted search

The reason for using inverted files in information retrieval is the observation that a user query usually contains a small percentage of the total number of terms in the document collection. Recall from Section 2.1 how the index terms are used as entry points into the documents. In collaborative filtering, a user's ratings (or profile) can be seen as a document, which in itself can be used as a query. However, there are some differences between a document and a profile. If the user, as in a movie recommender system, explicitly expresses the ratings, the user's profile will certainly grow over time and perhaps contain several hundreds or even thousands of ratings. In the case of implicit voting, the profile may grow even faster when, for example, each web page visit or news article read render a rating that is stored in the profile.

Since the user profiles are expected to grow over time, it may seem unlikely that inverted files are suited for collaborative filtering. We argue quite the opposite, for two main reasons: the first being that inverted file search is well researched and handles scalability very well. The second is that the inverted file structure makes it easy to employ heuristic stop conditions, so that it is not necessary to investigate all inverted lists in a neighbourhood search and thus further speed up execution.

When the user data is in inverted form, we have a data structure that enables us to quickly look up the users and their ratings for a particular item. Analogous to information retrieval, the items can be thought of as terms, the users as documents and the ratings as the frequency of a term in a particular document.

The basic algorithm for inverted file search scans one inverted list at a time (Witten et al., 1999). During this scan, accumulators are stored in main memory for holding partial sums of scores or weights. In our case, the partial sums will be the partial correlations or similarities between the active user and all other users. When all inverted lists are processed, the accumulators contain the complete sums. Usually there are one or more arrays of static weights, such as normalising factors, which are combined with the accumulators to produce the final score for a user. The final score is calculated for all non-zero accumulators, and users with top scores are returned.

One of the most commonly used algorithms for calculating neighbours in memory-based predictions is Pearson correlation. Various extensions have been made to the basic correlation (Breese et al., 1998). We will demonstrate, in sections 5.1.1 through 5.1.4, how these algorithms may be expressed in a form suitable for inverted retrieval.

5.1.1 Pearson Correlation

The Pearson correlation (see also Formula 2.10) between two users a and i is defined as

$$w(a,i) = \frac{\sum_j (v_{a,j} - \bar{v}_a)(v_{i,j} - \bar{v}_i)}{\sqrt{\sum_j (v_{a,j} - \bar{v}_a)^2 \sum_j (v_{i,j} - \bar{v}_i)^2}} \tag{5.1}$$

where v_a is the profile for user a, $v_{a,j}$ is that user's rating for item j and $\bar{v}_a$ is the mean value of the ratings for user a. The index j runs over the intersecting items in the two profiles. We will use a set-like notation for listing the indexes; in this case $j \in \{v_a \cap v_i\}$. Furthermore, let $|v_a \cap v_i|$ be the number of ratings in the intersection between users a and i.

To apply inverted search to correlation, it is necessary to keep three different accumulators in memory: one for the sum in the numerator, and two for the sums in the denominator. Call these variables SAI, SAA and SII respectively. In addition, the mean values for each user must be easily accessible, and these are stored in the static array $MEANS$. Let N denote the number of users in the database. Algorithm 1 will then compute the top K neighbours to user a. The sorting at step 15 can be performed by the use of min-heaps (Witten et al., 1999).

5.1.2 Inverse User Frequency

An extension to the basic Pearson correlation algorithm is inverse user frequency (Breese et al., 1998), which assigns lower weights to items that have a larger number of ratings. The assumption is that items that are viewed by a large number of users are not as useful in capturing the similarity between two users as items that have been viewed by fewer users. Inverse user frequency is similar to the inverse document frequency weighting (Salton & McGill, 1983) used in text retrieval.

The inverse user frequency for item j can be defined as $f_j = \log(N/n_j)$ where n_j is the number of users that have rated for item j. This weight is calculated for each item in the data set.

For the algorithm, we use the same notation as in (Breese et al., 1998). The inverted search algorithm requires six accumulators for the different sums $\sum_j f_j$, $\sum_j f_j v_{a,j} v_{i,j}$, $\sum_j f_j v_{a,j}$, $\sum_j f_j v_{i,j}$, $\sum_j f_j v_{a,j}^2$ and $\sum_j f_j v_{i,j}^2$. Furthermore, the inverse frequencies should be kept in a static array IUF, which at index j holds the value $\log(N/n_j)$. The inner loop of the search algorithm is then similar to that of Algorithm 1.

Algorithm 1 Inverted correlation neighbourhood search

Input is the user profile v_a, max number of neighbours K and array of means MEANS. Output is array of nearest neighbours.

```
1: Allocate accumulators SAI, SAA and SII of size N
2: Allocate array TOP of size K for nearest neighbours
3: for all j ∈ v_a do
4:    locate inverted list L for item j
5:    for each user u and rating v in L do
6:       SAI[u] = SAI[u] + (v_{a,j} − v̄_a) * (v − MEANS[u])
7:       SAA[u] = SAA[u] + (v_{a,j} − v̄_a)^2
8:       SII[u] = SII[u] + (v − MEANS[u])^2
9:    end for
10: end for
11: for all u ∈ SAI, such that SAI[u] ≠ 0 do
12:    corr = SAI[u]/√(SAA[u] * SII[u])
13:    if corr > TOP[K − 1] then
14:       add (u, corr) to TOP
15:       restore TOP to sorted order
16:    end if
17: end for
18: return TOP
```

5.1.3 Default Voting

Default voting is computed between two users a and i in the union of their ratings. If both users have not rated item j, a default rating d is used for the missing rating. The default rating is selected globally for the whole data set, as a neutral or slightly negative rating. Another parameter k estimates the number of items two users have not seen or rated but would agree on.

Let n be the number of items that both user a and i have rated. Default voting is then

$$w(a,i) = \frac{A_1 - B_1}{\sqrt{U_1 V_1}} \tag{5.2}$$

where

$$\begin{aligned} A_1 &= (n+k)(\sum_j v_{a,j} v_{i,j} + kd^2) \\ B_1 &= (\sum_j v_{a,j} + kd)(\sum_j v_{i,j} + kd) \end{aligned}$$

$$
\begin{aligned}
U_1 &= (n+k)(\sum_j v_{a,j}^2 + kd^2) - (\sum_j v_{a,j} + kd)^2 \\
V_1 &= (n+k)(\sum_j v_{i,j}^2 + kd^2) - (\sum_j v_{i,j} + kd)^2
\end{aligned}
$$

Recall that the inverted search finds the intersection of the ratings of the active user and all other users. This leads to a slight complication when inverting default voting, since it is calculated in the union of the user's ratings, i.e. now $j \in \{v_a \cup v_i\}$. To remedy this, we need to express the sum over union of items in terms of the sum of the intersecting items. The sum in A_1 can be written as

$$
\begin{aligned}
\sum_j v_{a,j} v_{i,j} &= \sum_m v_{a,m} v_{i,m} + \\
&\quad + d(\sum_s v_{a,s} - \sum_m v_{a,m}) + \\
&\quad + d(\sum_t v_{i,t} - \sum_m v_{i,m})
\end{aligned}
\tag{5.3}
$$

where $j \in \{v_a \cup v_i\}$, $s \in \{v_a\}$, $t \in \{v_i\}$ and $m \in \{v_a \cap v_i\}$.

The idea is then to accumulate the sum over m during the search and have the sum over t in a static array. The sum over s (for the active user a) may be stored in the static array or calculated during the search.

The four remaining sums for formula 5.2 ($\sum_j v_{a,j}$, $\sum_j v_{i,j}$, $\sum_j v_{a,j}^2$ and $\sum_j v_{i,j}^2$) can be calculated using a similar argument. Consider for example the sum $\sum_j v_{a,j}$, which is equal to $\sum_s v_{a,s} + d(|v_i| - |v_a \cap v_i|)$. This may be calculated if we know $\sum_s v_{a,s}$ and $|v_i|$, and calculate $|v_a \cap v_i|$ during the search. This sum and the three remaining ones are symmetrical.

Altogether, three static arrays are needed to look up the values of $\sum_t v_{i,t}$, $\sum_t v_{i,t}^2$ and $|v_i|$. Four accumulators are needed, three for the partial sums over m ($\sum_m v_{a,m} v_{i,m}$, $\sum_m v_{a,m}$, $\sum_m v_{i,m}$) and one for the number of intersecting items $|v_a \cap v_i|$. Again, the sums over s may be stored in the static arrays or calculated during the search. We chose the latter for our experiments.

5.1.4 Default Voting and Inverse User Frequency

The idea of weighting items according to their inverse user frequency can also be applied to default voting. If we use the index variables j, s, t and m as in the previous section, then default voting with inverse user frequency can be expressed as

$$
w(a,i) = \frac{A_2 - B_2}{\sqrt{U_2 V_2}} \tag{5.4}
$$

where

$$
\begin{aligned}
A_2 &= (n+k)(\sum_j f_j v_{a,j} v_{i,j} + kd^2) \\
B_2 &= (\sum_j f_j v_{a,j} + kd)(\sum_j f_j v_{i,j} + kd) \\
U_2 &= \sum_j f_j((n+k)(\sum_j f_j v_{a,j}^2 + kd^2) \\
&\quad -(\sum_j f_j v_{a,j} + kd)^2) \\
V_2 &= \sum_j f_j((n+k)(\sum_j f_j v_{i,j}^2 + kd^2) \\
&\quad -(\sum_j f_j v_{i,j} + kd)^2)
\end{aligned}
$$

The sum in A_2 can be rewritten as

$$
\begin{aligned}
\sum_j f_j v_{a,j} v_{i,j} &= \sum_m f_m v_{a,m} v_{i,m} + \\
&\quad +d(\sum_s f_s v_{a,s} - \sum_m f_m v_{a,m}) + \\
&\quad +d(\sum_t f_t v_{i,t} - \sum_m f_m v_{i,m})
\end{aligned}
\tag{5.5}
$$

The remaining sums to complete formula 5.4 are symmetrical to

$$
\sum_j f_j v_{a,j} = \sum_s f_s v_{a,s} + d(\sum_t f_t - \sum_m f_m) \tag{5.6}
$$

Again, we require the sums over t to be kept in static arrays and accumulate the sums over m. The required arrays should contain at index i the values of $\sum_t f_t$, $\sum_t f_t v_{i,t}$, $\sum_t f_t v_{i,t}^2$ and $|v_i|$. Five accumulators are needed, where four are for the sums over m ($\sum_m f_m$, $\sum_m f_m v_{a,m} v_{i,m}$, $\sum_m f_m v_{a,m}$, $\sum_m f_m v_{i,m}$). The number of intersecting items $|v_a \cap v_i|$ is also required, to calculate $n = |v_a \cup v_i|$.

5.1.5 Early Termination Heuristics

As discussed earlier, user profiles can grow quite large, especially in the case of implicit ratings. If there are ways that would allow us to only use parts of a user profile for prediction, it would further speed up execution time.

Several heuristic stop conditions for text retrieval have been devised (Witten et al., 1999). Some of these take advantage of the properties of the similarity function. For example, if the document weights are normalised to unit length, then it is possible to bound the accumulator weights of the cosine function (Buckley & Lewit, 1985). Other heuristics place stop conditions on the number of traversed inverted lists or, as in our case, stop conditions on the number of users that are retrieved during the scanning of the inverted lists.

Two strategies for this are well known: Quit and Continue. In both methods, the query is sorted by decreasing weight and lists are then processed in this order until the stop condition is met. When the stop condition is reached, only those accumulators that are non-zero are considered. In the Quit strategy, no more lists are processed after this point and the accumulators are used as is. The other strategy is to continue processing lists but only to update those accumulators that are already non-zero. Thus, Continue will always calculate the total weight for users that have reached into non-zero accumulators, whereas Quit may not.

We have implemented both Quit and Continue for the four algorithms described in the previous sections. The items were weighted according to inverse user frequency.

5.2 File Organisation

The Prefix B^+-tree (Folk, Zoellick, & Riccardi, 1998; Bayer & Unterauer, 1977) was used for storing the index, since it is widely used and a particularly good data structure for storing and maintaining an index on external memory. The tree is divided into two separate files: one to store the index as minimal key separators (the index file) and one file to store the sorted key/value pairs (the value file). The key consists of an item id with the value containing either an inverted list or a pointer to a posting file that contains large inverted lists (lists larger than 256 bytes in size). The posting file was divided into 512 byte blocks and for each inverted list a pointer list was kept for tracking which blocks that list occupied.

5.2.1 Compression

For indexing, each data set was first converted to an internal vector format and then indexed. On the reference machine described in Section 5.3.3, the indexing process took less than a minute for each dataset; about 21 seconds to index 90% of the EachMovie data and 7 seconds for the MovieLens data.

To construct the index and the inverted lists, we used sort-based inversion (Witten et al., 1999). The lists of run-lengths of user ids together with the rating

were then compressed for each item, and stored in the posting file.

Four different compression algorithms were tried: Golomb, Elias delta, Elias gamma and variable byte coding. All these four methods are well described in the information retrieval literature (Williams & Zobel, 1999; Witten et al., 1999), so they will not be re-stated here. For the Golomb, Elias delta and Elias gamma codings the rating was stored as a binary number of three bits. For variable byte coding, one byte was used.

We compressed the inverted lists for the first 60000 users in the EachMovie data that had rated at least two items, a total of 2805901 ratings. The average number of bytes per pointer (user id, rating pair) was close to one on Golomb, Delta and Gamma coding (1.04, 1.10 and 1.11). Using variable byte coding the data was compressed to 2.05 bytes per pointer. The block file utilisation was 93.7%, 93.5%, 93.2% and 96.2% respectively.

When running the experiments described in Section 5.3, we selected the simplest compression method, variable byte coding, since we wanted to see whether it was possible to use less complicated compression algorithms and still gain in performance by using inverted search.

5.2.2 Updating the Inverted Files

A collaborative filtering system will not serve its purpose if it is difficult to update it with new ratings and users. In a memory-based model this is one of the greatest advantages over model-based approaches. How will the inverted file structure handle updates? The problem that needs to be solved is how to update the inverted lists.

There are two ways of handling updates, either incrementally or in batch. If the index needs to be updated whenever new ratings arrive, the inverted lists should be represented without the need of decompression in order to insert or delete ratings. This approach has the drawback that the list postings will take up a larger amount of space. For batch updating changes would first be stored in a secondary index and then, at some point in time, merged with the primary index.

The static arrays of weights must also be updated, and in most cases the weight depends only on local properties of the profile. It is more problematic to update the static arrays in default voting with inverse user frequency, since those values depend on global properties. If a frequency is changed for an item, all weights must be updated to reflect this change. The simplest solution is to search the inverted list for that item, and for each user in the list subtract the old value from its weight and add the new value.

However, since the indexing process is so fast, one should consider to instead

completely re-index the data when there have been many updates.

5.3 Experiments

To evaluate the algorithms, we conducted a set of experiments. We measure the elapsed time to perform neighbourhood search and predictive accuracy.

We compare the inverted algorithms against in-memory vector searching. The user profiles are implemented as linked list, i.e. sparse vectors. This is not the fastest in-memory structure for performing intersection among user profiles: a hash table structure can be even faster. This will be discussed in relation to the results.

Each object in the profile contains the item id and the rating, and each user profile is sorted on ascending item id. The different correlation algorithms were implemented as list intersection (sections 5.1.1 and 5.1.2) or list union (sections 5.1.3 and 5.1.4). The mean value was pre-computed for each profile.

5.3.1 Data Sets and Experimental Setting

The datasets used in these experiments are EachMovie and MovieLens, which are described in more detail in Section 2.6.5.

For these datasets, the experimental setting was as follows:

- Users that had rated less than 2 items were removed.
- The algorithms were evaluated using the *AllBut1* protocol, meaning that for each user, a single rating was held out that should be predicted on the basis of all the other ratings in the profile
- A 10-fold cross validation was performed, as described in Section 2.6.4.
- For each experiment, we ran 4 different implementations on the same data: inverted search, inverted search using Quit, inverted search using Continue and in-memory vector search. Together with the four different algorithms, this makes 16 algorithms evaluated in each experiment.
- We limited the neighbourhood size to no more than 50, which has been found to be of reasonable size for movie data(Herlocker et al., 1999).
- For Quit and Continue we set the threshold at 10000 users for the EachMovie data, and 500 for the MovieLens data.

- For EachMovie, the ratings are integers in the range $[0 \ldots 5]$; for MovieLens $[1 \ldots 5]$.

Any experimental evaluation will face the problem of not covering all parameter settings, we could for instance have varied the threshold parameters to Quit and Continue, and the size of the neighbourhood. In these particular experiments we focus on the relative differences in performance and speed between the various algorithms, not to find the best parameter setting so as to find the lowest error.

5.3.2 Metrics

The metrics we use are time taken to compute a neighbourhood, and the predictive accuracy. For neighbourhood computations, we measure mean neighbourhood formation time: how long on average the algorithm takes to form a neighbourhood.

For predictive accuracy, we use mean absolute error (MAE) and root mean squared error (RMS) for measuring single predictions. MAE calculates the absolute difference between the prediction and the actual rating, averaged over all predictions. The two metrics are defined by formulas 2.41 and 2.42.

5.3.3 Machine Specifications

The experiments in this chapter were performed on a computer with a 3.0 GHz Intel Pentium 4 processor, 1 GB RAM and a 7200 RPM Seagate Barracuda hard drive and running Windows XP SP2. The software was implemented in Java and run with JDK 1.4.2.

5.4 Results

In this section we describe the main results of applying inverted file search to collaborative filtering. We measure neighbourhood formation time in Section 5.4.1 and predictive accuracy in Section 5.4.2.

In the tables, the rows labelled Inv and Vec give the results for the algorithm run on inverted search and in-memory vector search. In terms of accuracy, these two should produce the same result; any small difference is due to rounding errors in various steps of the algorithm. The Quit and Cont rows are the results of applying the early termination heuristics. The correlation algorithms are labelled CORR (Pearson Correlation), CIUF (Correlation with Inverse User Frequency), DEF (Default voting) and DIUF (Default voting with IUF).

5.4.1 Neighbourhood Formation Time

The mean time to compute neighbourhoods is displayed in Tables 5.1 and 5.2. The Quit method is the fastest, since only a few inverted lists are examined. Continue is slower than normal inverted search, because of the use of an additional data structure to hold the partial similarities. We implemented the accumulators for Quit and Continue as static arrays.

The time to compute a neighbourhood for the inverted search algorithms is measured by the time taken to complete all steps in the process, including sorting the neighbourhood values to locate the 50 top neighbours. In the current implementation, this is done by allocating a single object for each non-zero value, sorting this array and returning the top 50 neighbours. This can be made more efficient by the use of arrays and min-heaps, but the added extra time is only 5–20 ms. for the EachMovie data, and about 1 ms for MovieLens, and thus not of great importance for the argument here. For the in-memory vector search, the time is simply taken to be the time spent calculating the similarities and adding these values to an array.

The improvement in mean neighbourhood formation time by using inverted search is very encouraging. In general, inverted search is many times faster than in-memory vector search, even when using such a simple compression algorithm as variable byte coding. Furthermore, most data in the inverted structure resides on disk (except for the caveat of the operating system cache mechanisms). By explicitly caching parts of the index and the inverted lists, it is of course possible to further improve the neighbourhood formation time.

As noted earlier, the linked list implementation of the sparse vectors is not the optimal data structure if we are to perform an intersection between two vectors. Instead, one may use a hash table representation and then by direct access fetch the (possibly) intersecting features in the larger of the two profiles by iterating over all features in the smaller of the two. When running this experiment with the EachMovie data, the mean time to calculate neighbourhoods for CORR and CIUF (which are the algorithms that operate in the intersection), the timing results were much improved: 167.01 and 183.81 ms. compared to 600.83 and 615.92 respectively. For union calculations, hash tables are not an equally good alternative since calculating the union requires to iterate over all elements in both vectors. Furhermore, hash tables require more internal memory than lists.

Despite this huge speedup due to a more clever use of the in-memory data structures, this approach is still slower than using the inverted file. The reason for this is that there are elements that the hash table algorithm will inspect that are empty, and thus not part of the intersection, while the inverted file algorithm only retrieves the necessary elements.

	CORR	CIUF	DEF	DIUF
Vec	600.83	615.92	718.61	818.47
Inv	81.44	102.69	83.89	87.11
Quit	62.62	63.37	69.22	64.23
Cont	116.83	135.02	121.13	120.24

Table 5.1: Mean Neighbourhood formation time in ms. for EachMovie data set. Note that the time for CORR and CIUF for Vec can be further improved by the use of hash tables.

	CORR	CIUF	DEF	DIUF
Vec	154.01	159.50	190.71	220.77
Inv	25.57	29.35	25.22	25.63
Quit	14.10	14.54	15.54	14.03
Cont	31.58	34.29	31.02	31.03

Table 5.2: Mean Neighbourhood formation time in ms. for MovieLens data set. Note that the time for CORR and CIUF for Vec can be further improved by the use of hash tables.

5.4.2 Predictive Accuracy

The mean absolute error and root mean squared error for the *AllBut1* protocol is listed in Tables 5.3 and 5.4.

The first interesting result is that Continue is the top performer for all data sets; since the user profiles are sorted in descending weight order, this means that the shorter inverted lists (items that have fewer ratings) have high discriminating power, and that items with a large number of ratings may corrupt the ranking.

The second result is that the Quit method performs very well even in cases where very little information is used for making the prediction. We noted that in many cases only one inverted list was used by Quit. After all, if the size of the first list is greater than the threshold of 1000 or 10000, no more lists will be traversed.

	CORR	CIUF	DEF	DIUF
Vec	1.043	1.053	0.969	1.021
Inv	1.047	1.049	0.969	1.021
Quit	1.056	1.077	0.989	1.018
Cont	1.008	1.043	0.957	1.018

	CORR	CIUF	DEF	DIUF
Vec	1.335	1.361	1.234	1.317
Inv	1.339	1.358	1.234	1.317
Quit	1.353	1.392	1.264	1.315
Cont	1.301	1.354	1.224	1.314

Table 5.3: Mean Absolute Error (left) and Root Mean Squared Error (right) for the EachMovie data set. Lower scores are better.

	CORR	CIUF	DEF	DIUF
Vec	0.752	0.798	0.720	0.771
Inv	0.752	0.798	0.720	0.771
Quit	0.783	0.785	0.739	0.762
Cont	0.724	0.777	0.721	0.760

	CORR	CIUF	DEF	DIUF
Vec	0.956	1.014	0.925	0.970
Inv	0.954	1.014	0.926	0.970
Quit	0.987	1.004	0.941	0.962
Cont	0.937	0.996	0.931	0.960

Table 5.4: Mean Absolute Error (left) and Root Mean Squared Error (right) for the MovieLens data set. Lower scores are better.

Quit does not compute the true correlation (the accumulator sums may not be complete), so essentially the incomplete accumulator sums are combined with the complete pre-calculated values in the static arrays. This behaviour did not seem to harm the algorithm in a large way.

In terms of mean absolute error, we see a difference between the algorithms but also between the data sets. Continue is the top performer, and performs equally well or better than normal inverted search. Quit performs well compared to not using termination heuristics at all. This means that if we are interested in single predictions, we could do well by only taking into account a few items from the profile.

Default voting was found the most effective correlation method, both in terms of mean absolute error and root mean squared error. This is interesting, since the top performer in a previous study (Breese et al., 1998) on the EachMovie dataset was default voting with inverse user frequency. One very simple explanation for this could be that our training set was much larger than the one used in the previous study.

Another, more interesting issue, is why the inverse user frequency did not seem to help in our experiments. In previous studies, the inverse user frequency was used when all users were used as neighbours, while here we select a subset of 50 neighbours. The purpose of the inverse user frequency information is to dampen the effect of items with many ratings, and thus raise the effect of items with few ratings. If all users are taken as a neighbourhood, this effect is positive according to previous results. However, when used in combination with a thresholded subset of the neighbourhood, it seems likely that we should receive as top neighbours users that have rated many items that few others have rated, and from the results this is not the best neighbourhood.

5.5 Discussion

Collaborative Filtering has been successfully used in domains where the information content is not easily parsable, and where traditional information filtering

and retrieval techniques are difficult to apply. It is useful as a complement to traditional content based filtering techniques.

There are however a number of problems with memory-based collaborative filtering that needs to be resolved before it is truly useful. One important problem is that of scalability, i.e. how to manage a large number of users and items. In this chapter, we cast the collaborative filtering problem into an information retrieval problem. Our hypothesis was that by the use of inverted files, we could speed up execution time and perhaps even improve the predictive performance with early termination heuristics.

In order to test this hypothesis, four well-known algorithms for collaborative filtering were selected and expressed in a form suitable for inverted retrieval. The experiments gave interesting and encouraging results:

- The inverted file search algorithms were generally many times faster than in-memory search, even when having an equal amount of pre-calculated values.
- The Continue termination heuristic was the top performing algorithm in terms of predictive accuracy, and about as fast as normal inverted search.
- The Quit termination heuristic was very fast compared to the other methods tested, and yielded good predictions as well.

5.5.1 Future Work

The method of inverted retrieval should be possibly to apply to any sufficiently simple vector similarity function. An important theoretical question is how to define sufficiently simple in this case, that is, what properties a similarity function must have in order to be invertible.

Chapter 6

Incremental Collaborative Filtering

In this chapter we describe how collaborative filtering can be used for mobile devices. When the user is connected to a central repository, the algorithm selects a subset of profiles to store on the device. When the user is not connected to the repository, the predictions can be incrementally updated to reflect new or updated ratings. Experiments on one of the movie data sets show that the method can dramatically reduce the data needed while still performing nearly as well as a centralised approach. The material in this chapter has been published in (Cöster & Svensson, 2005).

6.1 Problem Background

Information filtering and retrieval systems generally operate on central data repositories, such as document indexes and databases of user ratings (Loeb & Terry, 1992). Client applications connect to databases where predictions for relevant items are computed by algorithms that have all data available. Whenever a user submits feedback, it can almost instantly be stored in the central repository and be quickly available to other users.

We consider a different type of application that needs to operate without being constantly connected to a server. Our system, MobiTip (Rudström, Svensson, Cöster, & Höök, 2004), is a mobile recommender system that allows people to create, rate and share information using short-range Bluetooth[1] communication while occasionally synchronising with a central server. In such a scenario it is vital that the filtering algorithms on the mobile device can account for new information without first having to connect to a central repository. This opens up for

[1]Bluetooth enable devices to connect and exchange data within a range of roughly 10 meters (*Specification of the Bluetooth System*, 2003).

a set of interesting issues regarding how to support information filtering when the connection to other devices and servers is limited.

The problem we focus on is to develop an incremental decentralised collaborative filtering algorithm. To date there are only a few attempts that we know of that support collaborative filtering on mobile devices, for example (Miller, Albert, Lam, Konstan, & Riedl, 2003; Tveit, 2001). Standard collaborative filtering algorithms require that many user profiles are directly available when making predictions (Herlocker et al., 1999). Such algorithms use the profiles in the nearest neighbourhood to make predictions or create a model of user preferences from the profiles.

We argue that it is not feasible to keep a database of all users and their profiles on every device, since calculating predictions from a large profile database would make the device unresponsive and soon drain it of battery power. Furthermore, storing the database of all users and their profiles on the device can be highly memory intensive for some domains.

Still, a collaborative filtering algorithm should react to, and change its predictions based on a user's new or updated ratings or from encounters with other devices. It is therefore not enough to produce a single list of predictions or a static model when the user is connected to the central server that does not update the predictions based on new data.

The general idea for the technical solution is to first store a user's profile on the device, together with a ranked list of predictions from a central server, computed the last time the user docked. Whenever new data becomes available, for example when encountering another user running the service, it should be possible to recompute these predictions based on the new data.

The incremental algorithm for mobile devices is presented in Section 6.2, along with three ways of selecting a subset of the profile database on the device. In Section 6.3 we describe the experiments and our main results.

6.2 Collaborative Filtering for Mobile Devices

The collaborative filtering algorithm that we use is that presented in Section 2.4.1, which is based on a linear combination of other user's ratings for the items, weighted by their similarity with the active user. We state again the algorithm here for ease of exposition. The prediction $p_{a,j}$ for user a on item j is

$$p_{a,j} = \bar{v}_a + \kappa_j \sum_i w(a,i)(v_{i,j} - \bar{v}_i) \tag{6.1}$$

The function $w(a,i)$ should measure the similarity between two users a and i. There is a normalising factor, κ_j, selected so that the absolute values of

the weights $w(a,i)$ sum to unity. The absolute values of the weights sum to $\sum_i |w(a,i)|$ so κ_j is taken to be $\kappa_j = \frac{1}{\sum_i |w(a,i)|}$.

The similarity function we use is the basic Pearson Correlation, detailed in Formula 2.10.

6.2.1 Mobile Device Characteristics

The computational power and storage capacity of a mobile device, such as a mobile phone, is limited in many ways when compared to a stationary computer. The amount of internal memory is smaller, the processor operates at a much lower speed and the battery time is relatively short. Furthermore, some devices do not have floating-point arithmetic built into the hardware, which demonstrates the need for algorithms that makes few such computations.

Many mobile devices may be connected to a server (using e.g. WAP or GPRS) but the user often has to pay extra for this data transmission. On the other hand, most new devices have the possibility of connecting to other devices in their physical vicinity, using Bluetooth, Wireless LAN, or infrared communication, without additional charge.

These characteristics influence the design of information filtering algorithms for mobile devices. The algorithm we propose take into account some of these considerations in that it is incrementally updated with relatively small amounts of data by connecting to nearby devices over Bluetooth without the need for constant connection to a central server.

6.2.2 Incremental Algorithm

Our information filtering algorithm produces a ranked list of predictions that can be updated and changed with new or changed user profiles. In brief, it works as follows.

1. When a user is connected to the central repository a subset of user profiles is first selected that will be used for any further calculations. This subset is copied to the users' mobile device.

2. Based on the subset, a ranked list of the predictions for all items is calculated on the server, using formula 6.1. This list is stored on the user's device. When the user is no longer connected to the server (*floating* mode), the service can still recommend items based on the ranked list.

3. When a user gives new ratings, or updates previous ones, the predictions must be recomputed. If there is currently no connection to the server, the

stored predictions should be recomputed based on the previously copied subset of user profiles.

4. When two users meet, their services may exchange user profiles (which could be changed from the last server synchronisation). For each user, the ranked list of predictions should now be recomputed. The past influence of the other user should be removed from all predictions and the new influence should be added to the predictions.

The filtering algorithm 6.1 expresses the prediction as a linear function of the user similarities, so we can subtract and add user similarities as needed.

Consider the basic prediction algorithm 6.1 in a more general form where we set $b_a = \bar{v}_a$ and $g(v_{i,j}, \bar{v}_i) = (v_{i,j} - \bar{v}_i)$. The reason for expressing it in this more general form is e.g. that the mean rating value is not necessarily an optimal bias value (Tian & Cheung, 2003). The algorithm can then be reformulated as

$$p_{a,j} = b_a + \kappa_j \sum_i w(a,i) g(v_{i,j}, \bar{v}_i) \tag{6.2}$$

When user a docks, the user's profile v_a is stored on the server. The server then recalculates the predictions $p_{a,j}$ for all items. Depending on the storage capacity on the user's device a subset of all user profiles known at the time of docking will be used. The ranked list of predictions $p_{a,j}$ is stored on user a's device, together with the normalising values κ_j. The subset of the other user's profiles is also stored.

When the user leaves the docked mode, there is a ranked list of all items directly available on the device. Now the user is in floating mode, and may connect to other devices by Bluetooth until the device is docked again.

In summary, the data that has to be stored on the device is: a) the user's profile v_a and the bias b_a, b) the list of predictions $p_{a,j}$ for every item, c) the κ_j values for each item, and d) a set of profiles. If the algorithm uses global information such as inverse user frequency, this must be stored as well.

6.2.3 Profile Subset Selection

We will focus on selecting this subset to improve a user's own predictions, although there can be other (altruistic) reasons for choosing a particular profile subset to store on the device, for instance to pass along newly fetched profiles to users that seldom docks.

The first obvious method for selecting a subset of profiles is to take the top ranked profiles according to the weight function $w(a,i)$. This is the baseline

subset selection method:

$$s_1(a,i) = w(a,i) \tag{6.3}$$

One way of viewing the problem of selecting a good subset of profiles is to select profiles that have high probability of changing a prediction when the active user updates the profile. A user may update a previous rating or store a new rating. In the first case, the change of prediction is completely dependent on the algorithm $w(a,i)$. For the second case users can be chosen based on the number of ratings in their profile that user a has not yet rated. If we assume that each item has equal probability to be rated by a, then the user with most ratings not rated by a has the highest probability of such a match. So, if a user i has rated many items that user a have not rated, then we can expect the probability that user a rates one of those is high.

Each user i can thus be ranked according to the number of ratings in the intersection of a and i, and the number of ratings for items that user i has rated but a has not.

Let $|v_i|$ be the number of ratings in the profile v_i, and let $|v_i \cap v_a|$ be the number of ratings in the intersection of i and a. If N is the total number of items, then the fraction of remaining items that user i has made but are not in a, is

$$\frac{|v_i| - |v_i \cap v_a|}{N - |v_a|} \tag{6.4}$$

The higher this fraction, the higher the probability that when user a rates a randomly chosen item it will be included in user i's set of ratings. Although user i has many ratings, there may be few of those that are also rated by user a, why we also include the intersecting items.

This can be formulated as

$$s_2(a,i) = \frac{|v_i| - |v_i \cap v_a|}{N - |v_a|} \times \left(\frac{|v_i \cap v_a|}{|v_a|} \right) \tag{6.5}$$

Since the device's storage capacity may be limited, small profiles might be preferred over large ones. The third selection method is therefore a combination of s_1 and s_2. We use s_2 to promote small profiles, i.e. $1 - s_2(a,i)$, and multiply that value with the similarity value s_1, to select small but similar profiles. This method can be written as

$$s_3(a,i) = s_1(a,i)(1 - s_2(a,i)) \tag{6.6}$$

6.2.4 Updating the User's Profile

When users update their profile, the predictions should also be updated. Assume that there are T profiles stored on the device, and that user a has an updated

profile. The goal is to calculate the new prediction $p'_{a,j}$ for each item j.

From 6.2 we get the value of the sum on the right hand side by

$$\frac{p_{a,j} - b_a}{\kappa_j} = \sum_i w(a,i)g(v_{i,j}, \bar{v}_i) \tag{6.7}$$

Let the index t run over all users that are stored. After user a has updated the profile to a', the new sum on the right hand side of 6.2 should now be

$$S = \frac{p_{a,j} - b_a}{\kappa_j} - \sum_t w(a,t)g(v_{t,j}, \bar{v}_t) + \sum_t w(a',t)g(v_{t,j}, \bar{v}_t) \tag{6.8}$$

Note that we need to be able to calculate both $w(a,t)$ and $w(a',t)$ for the active user.

The κ_j value must also be updated, so we subtract the old weights and add the new weights in the same manner

$$\kappa'_j = \frac{1}{1/k_j - \sum_t |w(a,t)| + \sum_t |w(a',t)|} \tag{6.9}$$

The updated prediction can now be written as

$$p'_{a,j} = b'_a + \kappa'_j S \tag{6.10}$$

where b'_a is the user's updated bias value.

6.2.5 Updating Profiles in the Subset

When user a connects to another user, the predictions $p_{a,j}$ should be updated to reflect the new profile that is transmitted to a. There are three cases when the predictions need to be updated when receiving a profile from a user i:

1. If i is known to a but has updated ratings. In this scenario the same argument as when user a had updated ratings holds.

2. If i is not known to a. In this scenario the predictions need to be updated based on a new user t. The new sum on the right hand side of 6.2 should now be

$$S = \frac{p_{a,j} - b_a}{\kappa_j} + w(a,t)g(v_{t,j}, \bar{v}_t) \tag{6.11}$$

The κ_j value is similarly updated by

$$\kappa'_j = \frac{1}{1/k_j + |w(a,t)|} \tag{6.12}$$

and formula 6.10 is again used for the updated prediction value.

3. If i has ratings for items that user a does not have predictions for. The prediction for the new item can be calculated with formula 6.2.

The incremental algorithm only operates on a subset of the user profiles stored on the server. One reason for doing so is that it simplifies handling of new or updated profiles, since it is safe to assume that a user who is not known to the user (Item 2 above) has not previously affected the recommendations. There are, however, situations for which we would like to use all available data whenever possible. Thus, instead of first selecting a subset of user profiles, the algorithm would use all user profiles to calculate the server-based predictions.

If the algorithm uses all user profiles on the server, the update mechanism in item (2) above must be slightly changed. When user a receives a profile that is not stored on the device but was used when calculating user a's predictions on the server, the past influence of that user's profile must first be removed from the predictions on the device. The solution is to time stamp all events in a user's profile. The algorithm can then synchronise the new profile against the last dock and remove all ratings up to that time and then recalculate the predictions as explained above.

6.3 Experiments

In order to evaluate the incremental algorithm, and the three methods for selecting the subset of users to store on the device, we performed a set of experiments with data gathered from a real collaborative filtering application. In these experiments, we evaluate the incremental algorithm described in Section 6.2.4, and compare it with its stationary counterpart.

6.3.1 Data set

The data set we have used is the MovieLens (*ML*) data (see Section 2.6.5), which contains in total 6040 users and approximately one million ratings. The ratings are explicit integer values from 1 to 5, for a set of 3706 movies, and each user has made at least 20 ratings.

We selected this data set since it was the set most similar in spirit to the data that we expect to find in our target application MobiTip. The target application may contain ratings for leisure items such as movies, books and restaurants, and it also uses explicit ratings.

For the actual experiment, we randomly selected 1000 of the 6040 users, to have a more manageable data set to work with. From this set, we performed 10

evaluation runs where for each run we split the 1000 users into 900 train and 100 test users.

6.3.2 Evaluation Method

As explained in Section 6.2, the device contains a subset of the profiles and a list of predictions, taken from the server the last time the user docked. Whenever the user gives a new rating, the predictions are updated according to formula 6.10.

To simulate this docking/floating behaviour, we first randomly shuffle each profile in the test set, and split it into 10 equally sized partitions, or dock points, meaning that each test user must have at least 10 ratings. Each user is then docked at point number $D_b \in [1, 2, \ldots, 9]$, and given a set of predictions and a profile subset from the server (i.e. the training set) based on the ratings made up to this begin dock point.

To evaluate the incremental behaviour of the algorithm, we evaluate one rating at a time, up to the remaining end dock points $D_e \in [D_b + 1, \ldots, 10]$. Each following rating from D_b to D_e is predicted and evaluated, and then incrementally added to the profile using formula 6.10. The error that we will report is thus an incremental, or cumulative, value that is calculated from the first dock point D_b. Therefore, we call this metric the cumulative mean absolute error

As an example, assume that a user has made 100 ratings. The profile is split into 10 partitions of 10 ratings each. For begin dock point $D_b = 1$, the user is given a profile subset based on the first 10 ratings. The remaining 90 ratings are then evaluated one at a time. Each rating is predicted, evaluated and then incrementally added to the profile, and a cumulative error is calculated at end dock points $D_e \in [2, \ldots, 10]$. For begin dock point $D_b = 2$, the user subset is calculated on basis of the first 20 ratings, and the cumulative error is calculated for the 80 remaining ratings at end dock points $D_e \in [3, 4, \ldots, 10]$.

Using this type of evaluation, we can track how the error evolves if the user docks when only 10% ratings are given, and then does not dock until most of the ratings are given.

We compare the incremental algorithm to its stationary counterpart, where we select a new neighbourhood from the training set for each prediction. All algorithms select 100 user profiles either as a subset (for the incremental method) or for neighbourhood formation (for the server method) to make predictions from. Note that the incremental method will select these 100 users only at the begin dock points, while the server methods forms this neighbourhood for every prediction.

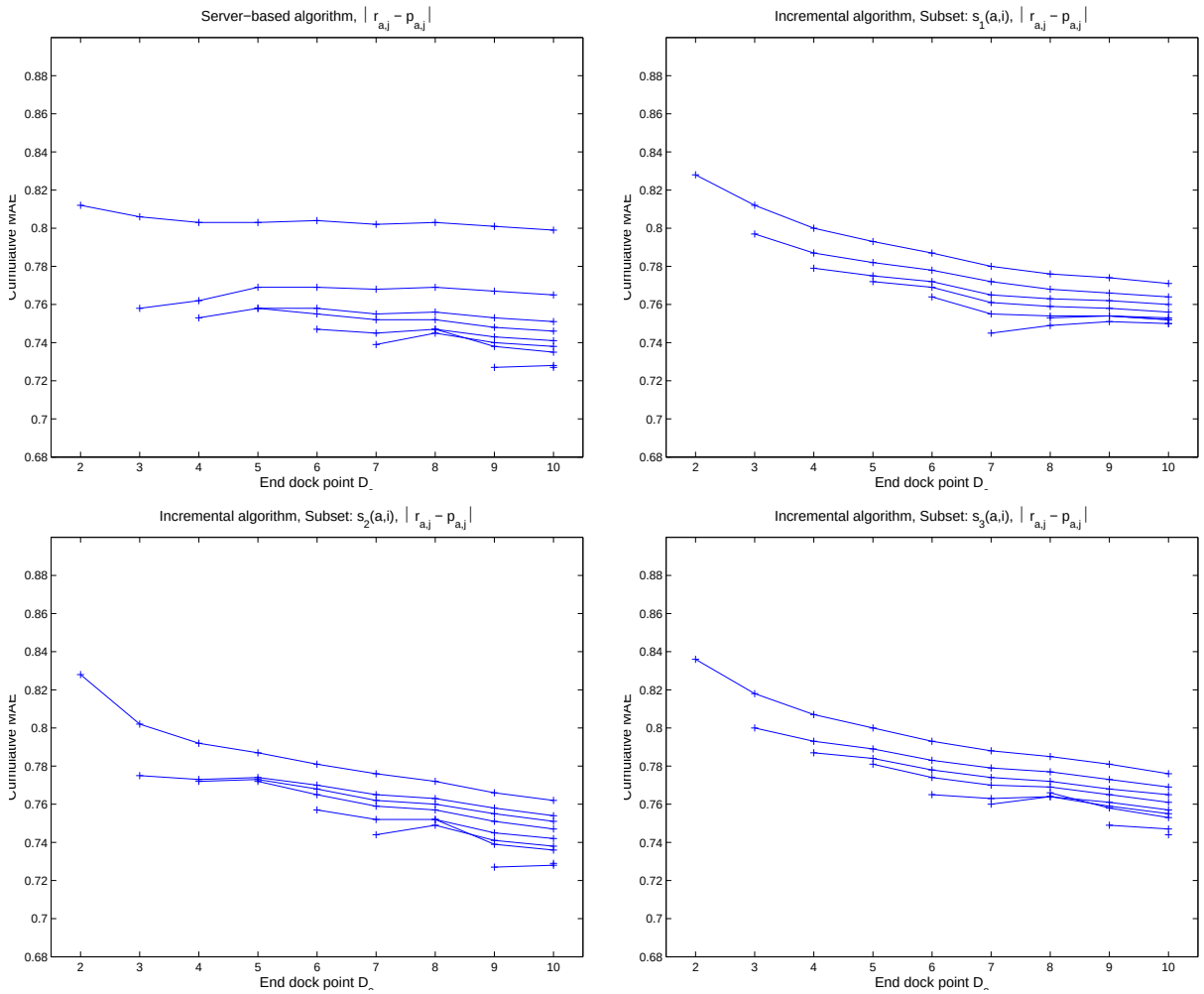

Figure 6.1: Cumulative Mean Absolute Error for Server-based algorithm (top left), Incremental algorithm using s_1 (top right, subset selected by the similarity function $w(a.i)$), s_2 (bottom left, subset selected by promoting large profiles) and s_3(bottom right, subset selected by promoting small profiles weighted by the similarity function $w(a, i)$).

	D_b=1	D_b=2	D_b=3	D_b=4	D_b=5	D_b=6	D_b=7	D_b=8	D_b=9
s_1	21510	17775	14879	12999	11672	10850	10200	9778	9457
s_2	57956	59614	60093	60361	60533	60645	60729	60794	60854
s_3	10177	7452	6649	6193	6082	5980	5988	5996	6041

Table 6.1: Average number of ratings in the subset of 100 profiles selected by methods s_1,s_2 and s_3 on the MovieLens data set. Columns indicate the begin dock point D_b.

6.3.3 Evaluation Metrics

Each individual rating is evaluated using the AllBut1 protocol, where all ratings that are currently in the profile are used for making a prediction for a single holdout item. After an item has been evaluated, it is added to the profile.

For this data set, we use the mean absolute error (MAE) as the measure of effectiveness, see Section 2.6.3. As just discussed we measure, for each user, the error for each prediction made from dock point D_b to D_e. We call this the *cumulative* error, and this is averaged over all test users for each dock endpoint.

For each algorithm we report 9 different curves that begin on different dock points, but all end at dock point 10. On the x-axis is the end dock point, which means that the begin dock point for a curve is the dock point prior to the curve's leftmost point. The error or is displayed on the y-axis. For mean absolute error, lower scores are better.

We also display, for the incremental algorithm, a table that shows the average total number of ratings in the subset of 100 user profiles that is selected from the server at the different begin dock points.

6.3.4 Results

The results are displayed in Figure 6.1 with four graphs that show the result of applying four different algorithms on the ML data. The server-based algorithm is displayed in the upper left corner, and the incremental algorithm using the different subset selection methods are displayed in the remaining three graphs.

The first observation is that the server-based method is more stable in the sense that the cumulative error does not change much when given more ratings in a profile. The incremental methods have a larger error near the begin dock point, but as more and more ratings are added to the profile the error decrease more rapidly.

For example, the longest curve (for dock points $D_b = 1$ and $D_e \in [2, \ldots, 10]$) has an initial error near 0.84 for s_3, but the cumulative error on the last dock point $D_e = 10$ is approximately 0.78 while for the server-based method the initial

error is near 0.81 and the cumulative error is 0.80. For the low begin dock points, the incremental algorithm has a lower cumulative error than the server-based algorithm.

Another notable result is that the server-based method and the subset selection method s_2 have the lowest error scores, for the high begin dock points. The smallest error is slightly above 0.72 for profiles with $D_b = 9, D_e = 10$.

In Table 6.1 we display the average number of ratings in the subset. Selection method s_3 performs especially well if we take into account the small average number of ratings that is included in the subset of the 100 profiles. This method approximately halves the size of the subset compared to s_1 while still keeping the error low. The selection method s_2 selects very large profiles, as expected, and has the lowest error (together with the server based method) for the high begin dock points.

These results indicate that we need only store at most 20KB of data (assuming 2 bytes per rating) for an expected cumulative MAE that is comparable to that achieved by a server-based method where all profiles are stored. This should be compared to the average amount of ratings in the training set on the server that amount to about 150K ratings, or even all data that may contain millions of ratings.

6.4 Discussion

We have described and evaluated a collaborative filtering algorithm that can be used for devices that have limited storage capacity, limited power, and which is not always connected to a central server. We have shown that it is possible to use a powerful server to produce predictions and then later have the client incrementally update those when new data is available, without being connected to the server.

An interesting pattern emerges, if we look at how the collaborative filtering algorithm behaves over time. For the ML data it is clear that the more that is known about a user the better the predictions are, even though we fix the neighbourhood of users that are used for matching. One explanation for this could be that the user's mean rating becomes more and more accurate.

6.4.1 Future Work

So far we have evaluated the incremental algorithm with respect to how a user's own profile changes. The next step is to evaluate what happens when a user encounters other users and updates to the profile subset occur. This is related to profile subset selection method, for which more research is needed.

For the subset selection problem, we formulated a problem that might be of a general interest in collaborative filtering: what is the probability that a user will rate a particular item? That is, the question is not to predict the actual rating, but rather to predict whether the user will rate the item at all. If it would be possible to make such predictions with high accuracy, we could select profiles on the basis of whether the profiles contain ratings for items that the user is expected to rate.

The collaborative filtering algorithm used here is based on a sum of recommendations from other users, weighted by their similarity with the active user. This algorithm is a linear function, and it is this linearity that enabled us to formulate the incremental update algorithm. Another issue for future work might be to investigate which classes of linear prediction algorithms that can be incrementally updated in a way similar to that presented here.

Chapter 7

Predictor

In this chapter we describe a system, *Predictor*, which can be used for both content-based and collaborative information access. The functionality in Predictor is fairly general, and not tied to a specific application domain; it may be used for categorising documents, recommending movies and more.

Information Filtering was briefly discussed in Section 2.2; it was defined as the process of sorting out relevant documents or items on the basis of a long-term query or profile. In the preceding chapters, we have discussed several algorithms and representations that can be classified as parts of an information filtering system:

- In Chapter 3, we discussed a method for improving retrieval queries on the basis of past relevance feedback information, while representing documents and queries as dense vectors in a reduced space.
- In Chapter 4, we discussed a method for representing documents as dense concept vectors for a text categorisation task.
- In Chapter 5, we discussed inverted file search algorithms for nearest neighbour search among sparse user rating vectors in collaborative filtering.
- In Chapter 6, we discussed an incremental learning method for collaborative filtering, particularly applicable to a scenario when a mobile collaborative filtering client is not always connected to a central server.

These representations and algorithms have one major component in common; they are based on a *vector* representation of user ratings and document contents. In the case of dense vectors, such as those used in Chapters 3 and 4, the retrieval and classification algorithm is readily performed by in-memory vector search operations. In the case of sparse vectors, such as the collaborative filtering rating

vectors in Chapters 5 and 6, it was noted that inverted files could be used for fast and accurate retrieval on the server side, and an incremental method for the client side.

Regardless of whether the content-based or collaborative information is represented by sparse or dense vectors, the purpose of Predictor is to enable ratings, documents and prediction algorithms to be represented and used in one system.

The rest of this chapter is organised as follows. In the next section, we discuss what kind of functionality an information filtering system is expected to have. We review related work in Section 7.2. In Section 7.3 we discuss how a general vector-based representation can support both content-based and collaborative filtering, and an overview of Predictor's system architecture and the implementation is presented in Section 7.4.

The material in this chapter was first published as a technical report in (Cöster, 2002a). The system, Predictor, can at the time of writing be downloaded from `http://www.sics.se/humle/socialcomputing/download/`.

7.1 Information Filtering

The basic properties of an information filtering system can be summarised by the following;

- Information is filtered from a stream of information. To filter means to find, or remove, information from this stream.
- The filtering process is based on a description of user's information need.
- The information subject for filtering is unstructured or semi-structured data, such as text documents.

For a more detailed overview of information filtering systems than that given here, the reader should consult (Belkin & Croft, 1992) and (Oard, 1997).

Depending on the application domain, the size and rate of the information stream, which can be both user ratings and documents, can vary. A news filtering service may need to filter huge amounts of news articles to a large number of users, whereas an e-mail filtering software need only categorise one user's incoming e-mail.

In our scenario, we do not explicitly model the information stream; it is assumed that the information arrives at a rate that the system can handle. As such, updating mechanisms are not the main focus, instead the focus is on general properties of information filtering applications that enables us to combine

content-based and collaborative filtering in the same system architecture and be free to combine algorithms as we desire.

When constructing an information filtering system, some essential design decisions must be considered:

- How to capture a user's information need.
- How to represent a user's information need.
- How to represent the information.
- How to filter information based on the user's information need.

Capturing a user's information need is not a trivial task, and a lot of research has been devoted to this issue. In general, the system tries to capture this in a cycle of interactions between the system and the user (Oard, 1997). The interaction can be explicit, as in the form of, e.g., dialogues, or implicit, by observing the user's actions. A technique that captures a user's information need will be called a *feedback function*, since its purpose is to capture the feedback from the user.

The result of the feedback function is stored in a *profile*. A profile is the system's internal representation of a user's information need, and is the basis for filtering information.

The information, whether a news article, e-mail, etc. has to have an internal representation. For the purposes of this chapter, the internal representation of the information will be called a *document*.

The *filtering function* is the matchmaking part of an information filtering system. Based on a user's profile and a document, the filtering function should produce a value that reflects the system's belief in the document being relevant to the user.

In the following, we discuss these design decisions in the context of content-based and collaborative filtering.

7.1.1 Content-based Filtering

Content-based filtering systems filter information on the basis of the characteristics of the information. Thus, it must be possible to describe the information by its content. In general, it could be any unstructured or semi-structured data that can be described by its content. It will be assumed that a document carries textual content, for the sake of discussion, if not stated otherwise.

Document Representation Typically documents are represented by the presence of some more or less characteristic feature in them: terms, phrases, etc. This can be realised by documents represented by a set of binary attributes as in the Boolean or Binary Independence models (see Section 2.1.4) or use statistical properties of features in the text as the basis for representation, such as the tf-idf representation in VSM (see Section 2.1.2). As discussed throughout this thesis, there are a number of attempts to overcome known drawbacks of this still quite simple document model: instead of pure occurrence statistics a vector of higher level derivatives of the occurrence statistics can be used, such as in LSI (Section 2.1.3) or Random Indexing (Section 4.3).

User Profile Representation Depending on the feedback function, the profile can be constructed and represented in various ways. Two techniques are outlined below:

- A set of weighted keywords or concepts that represent the characteristics of the documents that the user is interested in.
- As a list of documents, where for each document there is an indication of whether the document is relevant or not for the user.

In the first case, the feedback function may return a set of (weighted) keywords and phrases specified by the user in an interaction.

In the second case, the feedback function returns relevance scores for a set of documents. These relevance judgements can be used to build a model of the user's interest by using, e.g., supervised machine learning techniques.

Filtering Function If the profile consists of a set of weighted keywords, the profile can be seen as a query. Several techniques from information retrieval can be used to calculate a similarity score between a query and a document. If the score is above a certain threshold, the document is predicted to be relevant for the user.

If the profile consists of a list of positive and negative examples of relevant documents, a machine learning algorithm can be trained to learn the characteristics of relevant documents. The model is then used to predict relevance scores for documents.

Examples of these two methods are depicted in Figure 7.1 (A) and Figure 7.1 (B). In both figures, the dimensions of the space are the different keywords $k_1, \ldots, k_t$ known to the system.

Figure 7.1 (A) gives an example of matching a profile p_a and a document d_q, when represented as weighted keyword vectors. If the cosine of the angle

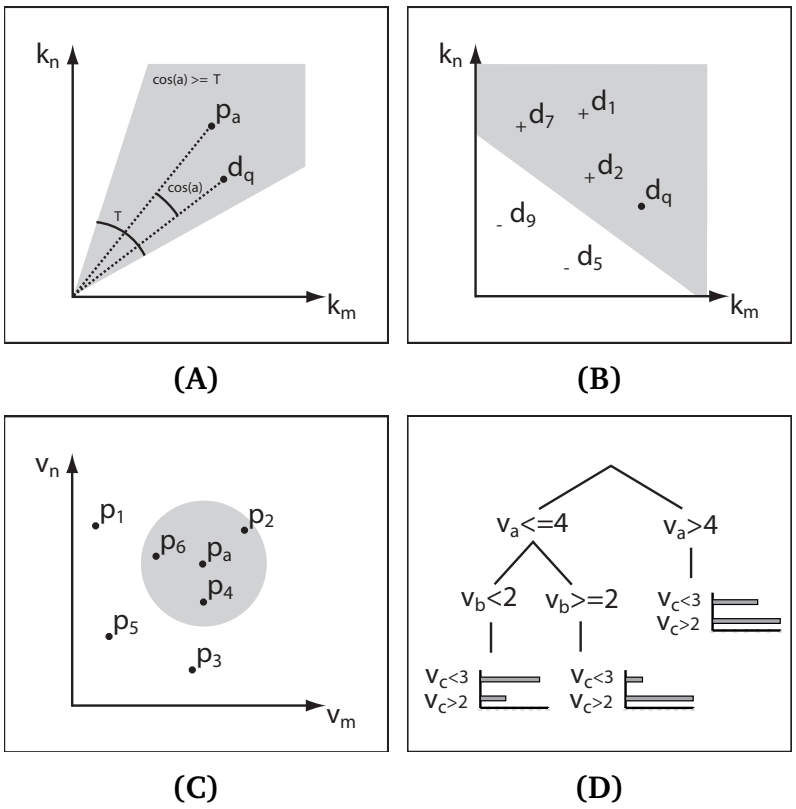

Figure 7.1: Examples of four different algorithms for content-based and collaborative filtering supported by Predictor.

between the profile and the document is above a certain threshold T, the document is considered relevant for the user. The shaded area is the region where the cosine of the angle between a new document and the profile is above the threshold.

Figure 7.1 (B) exemplifies an approach for classifying a new document d_q according to a model that has been built by learning from positive and negative examples of relevant documents. The algorithm constructs a model from a training set of documents (d_1, d_2, d_5, d_7, d_9 in this example). In the figure, the model is a linear separation of the keyword space, and a new document d_q is classified according to which side of the separating line its vector falls. The shaded area is the region that represents that a document would be classified as relevant for a user.

7.1.2 Collaborative Filtering

A collaborative filtering algorithm, as discussed throughout the thesis, predicts the relevance of a document for a user on the basis of user opinions, or ratings. The characteristics of the documents are not taken into account. Thus, there is no need to explicitly represent documents.

User Profile Representation In the case of explicit user feedback, the profile can be represented by a set of document ratings, where a rating is an ordinal number or a real value. In the case of implicit feedback, the profile could contain a value for each action the user has performed on the document. The value could be binary, indicating whether the user has viewed the document or not.

Filtering Function There are two main approaches to collaborative filtering, as discussed in Section 2.4; the memory-based and the model-based approaches. Examples of these two methods are given in 7.1 (C) and 7.1 (D).

In Figure 7.1 (C), each user profile p_i is represented by a vector in a space where each dimension corresponds to a document (these dimensions are named v for vote). The figure exemplifies the process of locating the nearest neighbours to the profile p_a using Euclidean distance. The relevance or rating of a new document is predicted by a combination of the ratings of the nearest neighbours, weighted by their distances from the profile.

A model-based approach is exemplified in Figure 7.1 (D). The figure depicts a decision tree that has been trained to predict a user's rating for document c. If a user's ratings for documents a and b are both below 2, then the decision tree would predict that the user would give document c a rating less than 3. The decision tree in this example is assumed to be a part of a larger model capable of predicting values for any document.

7.2 Related Work

Several systems for information filtering are described in the research literature. In many cases, the systems are specialised for certain filtering task, such as filtering news articles or recommending movies. The representation of documents, profiles, feedback functions and filtering algorithms are hard-wired into the systems, making it difficult to use them for other tasks or changing the internal algorithms used.

Since we describe the development of a general system for both content-based and collaborative filtering, it is interesting to review how some specialised systems work. There are, to the author's best knowledge, no publicly available sys-

tems which take a general approach to content-based and collaborative filtering in the sense described here.

SIFT SIFT (Stanford Information Filtering Tool) filters USENET news and mailing list articles for registered users (Yan & Garcia-Molina, 1999). Users subscribe by specifying one profile for each area of interest, in the form of a Vector Space or Boolean query. Each incoming article is presented as a query to an inverted profile index to find top matching profiles. If a user's profile matches the document, the user is notified via e-mail.

SIFTER SIFTER (Smart Information Filtering Technology for Electronic Resources) is another system for text document filtering (Mostafa, Mukhopadhyay, Palakal, & Lam, 1997). SIFTER takes on an unsupervised learning approach for creating a cluster representation of documents, where each cluster represents a class of interest. Each incoming document is classified according to the class of the most similar cluster.

The user profile is a vector, where the value of an element in the vector represents the system's belief that the user is interested in documents of that class. The profile is updated using a reinforcement learning approach, from user relevance feedback. The system also includes a module for capturing changing user needs.

GroupLens Within the GroupLens (Resnick et al., 1994) research project, several aspects and applications of collaborative filtering have been studied, such as news filtering and movie recommendations. Both memory-based and model-based techniques have been developed within the project. Several extensions, such as filtering information to groups of users, have also been developed. The filtering software is not publicly available.

Tapestry Tapestry (Goldberg et al., 1992) is a system for filtering e-mail that supports both content-based and collaborative filtering, although in a slightly different form than that presented in Section 7.1.

A user specifies a profile in the form of a Boolean search query, which can be tuned on a repository of past e-mail. The query language supports structured keyword searching in the different e-mail fields such as sender, date, subject, etc. The system also enables users to annotate e-mail, where an annotation may contain ratings. Users can specify profile queries that search both e-mails and annotations, thus making it possible to filter e-mail both in terms of content-based and collaborative features.

SELECT SELECT (Procter & al., 1999) is a EU-funded project that investigates how to create an information filtering system for web pages and Usenet News. In the project, a system has been implemented that enables users to submit and view ratings for web pages, and also receive predicted ratings for new web pages. The system's architecture is modular in the sense that information filtering algorithms can be plugged in.

Machine Learning Packages There are a number of publicly available software packages that may be used for content-based classification. These packages have not been developed for information filtering purposes, but may well be used by implementers of such systems. Two examples out of a wide range are: (1) LIBBOW – a package for text classification using Naïve Bayes and extensions (A. K. McCallum, 1996), and (2) WEKA – a general library of machine learning algorithms (Witten & Frank, 2000).

7.3 Representation

In Section 7.1, four algorithms for content-based and collaborative filtering were exemplified. These four are representative for a wide range of filtering algorithms. For instance, the algorithm that calculates the cosine of the angle between a document and a profile could have been used another similarity measure such as the Euclidean distance. The Support Vector Machine classifier could equally well have been a Naïve Bayes classifier. Similar arguments hold for the collaborative filtering methods. The memory-based approach could have measured the similarity by Pearson Correlation. The model-based approach could have been a linear classifier instead of a decision tree.

This observation is the starting point for creating an architecture that supports a variety of content-based and collaborative filtering methods. The issues that need to be discussed when developing a common representation for content-based and collaborative filtering are:

- Document representation.
- Profile representation.
- Feedback function.
- Filtering function.

In the next subsections, each of these four issues is discussed. Along with the discussion, the design decisions made in Predictor are presented and motivated.

7.3.1 Document Representation

In the discussion on document representation in Section 7.1.1, some examples were given. The difference between the first three representations (the Boolean, Vector Space and Latent Semantic Indexing models) lies in the interpretation of the various dimensions of the documents when represented as vectors. In the Vector Space case, each dimension of the document vector is treated as a term, and the value of the vector at that dimension reflects the frequency with which the term occurs throughout the document. In the case of Latent Semantic Indexing, the dimensions are treated as higher-level synthetic concepts, and the value of the vector at a certain dimension specifies how much the document is related to that concept.

There are other ways of representing documents, and any filtering system must choose which ones to support. The approach taken in Predictor is to represent a document as a vector of features, where a feature is a tuple of an ordinal number and a value. The ordinal number is a mapping to some object, determined by the application. The value is an integer or real value that reflects the user's interest or relevance judgements for some object. The interpretation of the actual meaning of these values lies outside the system.

It is easy to see that the document representations discussed above are supported by this design: the VSM, LSI and Random Indexing models all use a vector-based representation, the difference between the models (on a level of representation) lie in the interpretation of the vector dimensions.

7.3.2 User Profile Representation

The user profile should be able to record the user's interest or information need for documents. As discussed in Section 7.1.2, one choice of representation is as a list of relevance judgements for a set of documents. Another choice is as a set of weighted keywords. A more sophisticated model, such as a Support Vector Machine classifier, could also represent the user's information need.

As for documents, the approach taken in Predictor is to represent a user's profile as a vector of features. Furthermore, Predictor also provides a mechanism for connecting a more sophisticated model to a user, in the form of a machine learning program.

7.3.3 Feedback Function

The algorithm for updating a profile according to the user's actions is assumed to lie outside Predictor. Instead, a number of functions for updating profile vec-

tors, such as setting the value for a feature with a certain ordinal number, are provided.

7.3.4 Filtering Function

The discussion at the beginning of this section exemplified four algorithm types that are representative for a number of filtering algorithms.

The approach taken in Predictor is to supply a framework for information filtering, where these four algorithm types are supported. The algorithms operate on the general vector representation of documents and profiles.

To start the discussion on how to support these four types in a single system it is first convenient to distinguish between content-based and collaborative methods.

The content-based methods predict a single value (e.g., relevant or not relevant) on the basis of a description of a document. In Predictor, this type of prediction is called a *vector prediction*, since the description of the document is a vector.

The collaborative methods predict the user's rating for one or more documents, based on a description of the user's profile. Since the user's profile contains explicit or implicit ratings for documents, it means that the method should predict values for unrated features in that vector. For this reason, such a prediction is called a *feature prediction*.

Regardless of whether the algorithm implements a vector or feature prediction, the model can be built in one of two ways:

- Lazy – the model is built when making the prediction.
- Eager – the model is built off-line and stored for future predictions.

For example, the memory-based collaborative filtering algorithm discussed in Section 7.1.2 makes a *lazy feature prediction*, whereas the Support Vector Machine classifier from Section 7.1.1 implements an *eager vector prediction*. Predictor supports all four combinations of these algorithms.

7.3.5 Discussion

The restrictions imposed by this design are that documents and profiles must be represented by feature vectors, and that the filtering algorithms must be of the four types.

There are several other issues that have not been discussed, such as the size and rate of the information stream and how this will be managed by the system. This will be discussed in Section 7.5.

In the next section, some details regarding the architecture and implementation of Predictor are given.

7.4 System Architecture

Predictor is primarily accessed via an application programmer's interface (API), written in Java.

The API is a class that contains all necessary methods for accessing the system. There are methods for creating and deleting users, updating profiles, requesting predictions, etc. The API is initialised with a set of properties, such as where to put internal database and log files.

There are two different types of users: regular users and administrators. Each regular user must register by name and password in order to get access to the system. The administrator has special privileges, which will be described later.

The system manages four basic objects:

- User objects.
- Profile objects.
- Document objects.
- Predictor objects.

An overview of the system architecture is given in Figure 7.2 and Figure 7.3.

7.4.1 User Objects

User objects contain name, password, profile and a data string where threshold values and other information may be stored.

The API provides several functions for user management. Users can be created, deleted, updated and inspected. There are also methods for listing users in various ways. For example, it is possible to list all users that have more than a certain number of features in their profile vector.

7.4.2 Profile Objects

As stated earlier, the user profile is implemented as a vector of features, where a feature is a tuple of an ordinal number and a value. There are various implementations of the vector class, so that vectors may store bytes, integers or floating-point numbers. Furthermore, there are different implementations for the

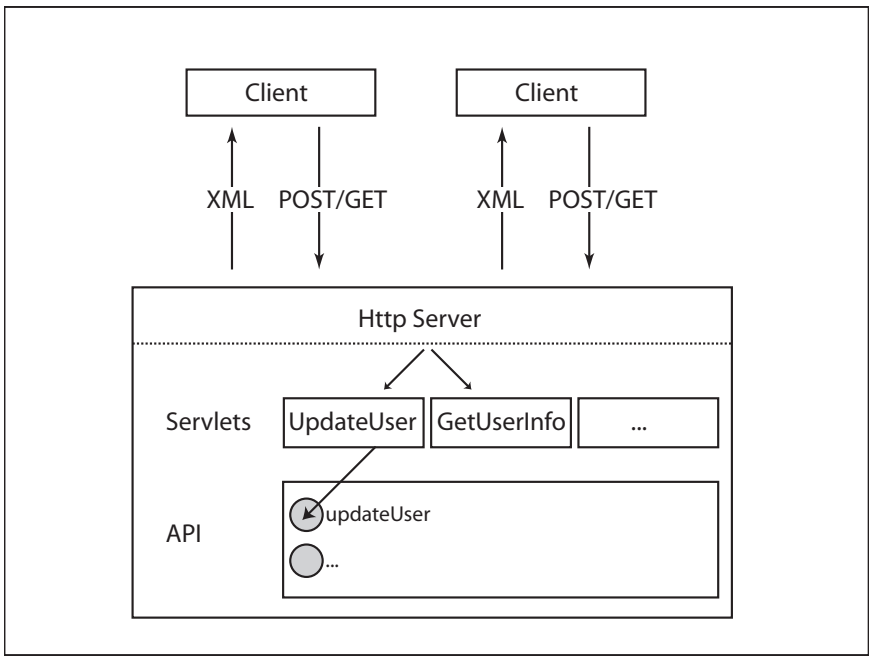

Figure 7.2: Overview of client-server communication

underlying data structure, so that e.g., both sparse and dense vector implementations are provided. When a user is created, the desired vector implementation is chosen.

The API provides a number of methods for updating and inspecting Profile objects.

7.4.3 Document Objects

Document objects contain a name and a document vector. The document vector is implemented using the same feature vector classes that are available for the profile vectors.

The API contains methods for viewing, creating and deleting Document objects. The methods that change or update Document objects are only accessible to the administrator.

7.4.4 Predictor Objects

Information filtering is performed by invoking a method in a Predictor object. Any registered user may ask a Predictor object to predict a value for one or

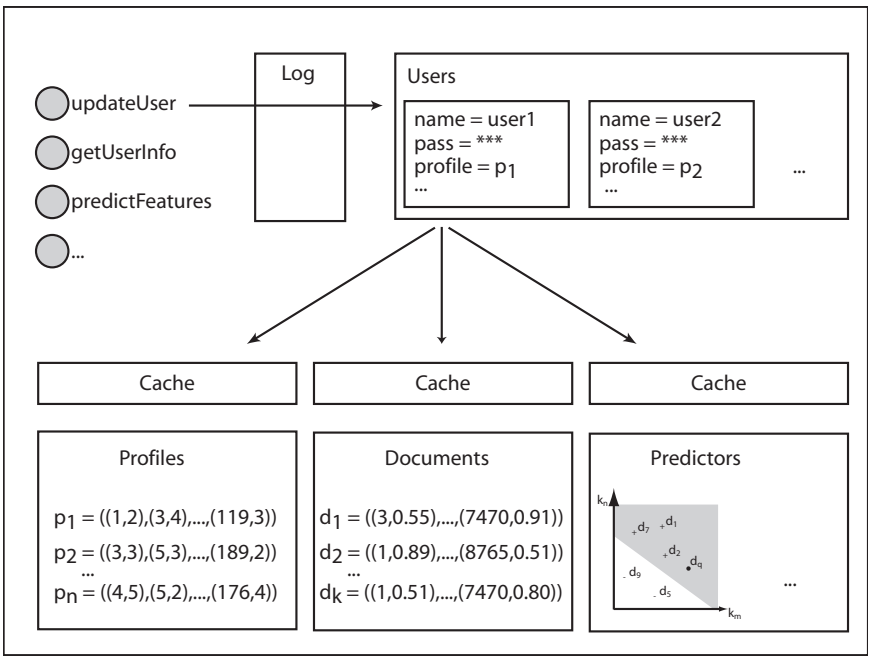

Figure 7.3: Overview of server architecture

more documents. However, users are not allowed to update, remove or alter the internal model of a Predictor object. Instead, it is the responsibility of the administrator to manage these objects.

The API contains methods for adding, deleting and inspecting Predictor objects. Since machine learning is an inherently iterative process, it is convenient to train such a program off-line and, when satisfied with its performance, add it to the system.

There are, as stated earlier, four types of Predictor objects that can be represented in the system. Currently, two objects are: one content-based and one collaborative.

- CollaborativeFilter implements *lazy feature prediction* using several memory-based collaborative filtering algorithms such as Pearson correlation, inverse user frequency and other extensions as described in Section 2.4.1.

- SVMClassifier implements *eager vector prediction* using a Support Vector Machine classifier. The classifier can be used to train models on the basis of a set of positive and negative examples of relevant documents. These models can then be used for making predictions. The classifier is implemented using the third-party library LIBSVM (Chang & Lin, 2001).

7.4.5 Storage Model

User objects are stored in a database structure composed of a disk-based B+Tree index and a variable length record file. The index contains user names, and the complete objects are stored in the record file. Document and Predictor objects are stored in a similar fashion, in separate indexes.

The storage model is implemented using a library for disc-based data structures developed at SICS[1].

All objects (User, Document and Predictor objects) are cached. The administrator sets which objects to cache, and the size of the cache. Currently, the least-recently-used algorithm is used for caching.

Depending on the application domain, different caching schemes should be used. For instance, when used as a pure memory-based collaborative filtering system, it is important to cache the User objects. When used solely as a content-based filtering system, it is more important to cache Document or Predictor objects.

7.4.6 Error Management and Logging

It is important that the implementation of an information filtering system is robust, and that there are ways to recover from a potential hardware or software failure.

The current implementation does not deal with the issue of robustness in an optimal manner, but provides extensive error and exception handling as well as detailed logging capabilities. Since the architecture provides for adding and removing executable code in the form of Predictor objects, there must be a mechanism for managing exceptions thrown by these objects in order to secure the integrity of the system.

For example, every method call to the API is written directly to a log file, with enough detail to reproduce the call. Furthermore, every exception is caught and written to a separate log file.

7.4.7 Client-Server Architecture

The API is the core part of Predictor, but to be more useful the system can also be used in a client-server environment. The architecture is based on servlet communication via Hypertext Transfer Protocol (http). An overview of the client-server architecture is given in Figure 7.2. There is one servlet for each method group in the API, i.e., create user, delete user, predict values, etc. The methods in the

[1]Swedish Institute of Computer Science AB, Kista, Sweden. Contact:martins@sics.se

API that are only accessible by the administrator are not accessible through the servlets.

The communication between client and server is as follows. The client sends a POST/GET request to a servlet, and receives a reply in XML format. Each servlet has the same name and corresponding parameters as the method group in the API. Each servlet response is in a specific XML format, so for each servlet there is also a DTD specification.

A Java client has also been implemented, which takes care of all communication to and from the servlets. In the client, the XML response is parsed and converted to Java objects.

The server may be any servlet server that supports Java Servlet API version 2.3[2]. The system has been tested on the Apache Tomcat server version 1.4[3] and Resin 2.1[4].

7.5 Discussion

In this chapter, we have described the architecture and implementation of a system for information filtering. The system is modular in the sense that information filtering algorithms can be added and removed at any time. Documents and profiles are represented by feature vectors whose contents are determined outside the system. As such, the system is capable of supporting a variety of possible representations, and a wide class of filtering algorithms. Currently, there are two specific algorithms implemented: memory-based collaborative filtering and document categorisation using Support Vector Machines. The system is accessible through an API, or through a set of servlets.

Early versions of the system have already been used in a number of research projects at SICS. So far, it has been used in three different systems: Kalas, RIND and GeoNotes. Kalas (Svensson, Höök, Laaksolahti, & Waern, 2001) is a social navigation system for finding food recipes, which uses collaborative filtering for recommending recipes to users. The RIND configurator (Cöster, Gustavsson, Olsson, & Rudström, 2002) uses a combination of collaborative and content-based filtering to guide users in the process of configuring a personal computer. Lastly, GeoNotes (Persson, Espinoza, Fagerberg, Sandin, & Cöster, 2003) is a mobile system for populating the ”real world” with virtual post-it notes. In GeoNotes, users can filter or view only those notes that are selected from a collaborative filter.

[2] `java.sun.com/products/servlet/download.html`

[3] `jakarta.apache.org/tomcat/index.html`

[4] `www.caucho.com/download/`

7.5.1 Future Work

The issue of how to deal with a large number of documents and ratings at a high input speed have not been discussed here. As for release 0.9 of Predictor, this is not yet added. Including the inverted file search algorithms is scheduled for the next major release, by including a *ranking prediction* class that should produce a ranked list of documents or users, for a given vector.

Predictor has no knowledge of the meaning of the features it manages, which can be a problem when there are different kinds of representations stored in the same instance of the API. The current use of the API is to instantiate one API for each representation. A better solution is to perhaps formalise the interface to the feature vectors, so that implementors can add information about the representation when storing the vectors, and later use this information when selecting, e.g., which filtering algorithm to use for a particular user and feature vector.

Chapter 8

Summary and Final Remarks

This thesis started out by pointing out a few areas in the intersection of information retrieval and collaborative filtering that were of particular importance; long-term learning from relevance feedback, concept-based representations, fast and accurate similarity calculations for collaborative filtering, incremental collaborative filtering for mobile client applications, and integration of the differerent representations and algorithms in a common system architecture.

It is now time to summarise key results from each chapter in turn, and discuss general conclusions that can be drawn from this work.

Learning from Relevance Feedback If was found that the long-term relevance feedback learning mechanism was well suited for a particular type of queries. In the CRAN collection, where the algorithm improved the search results, the queries are long, and there are many relevant documents for each query. In every other collection that was tried, the queries were short and there were fewer relevant documents per query.

Another finding was that the best way to construct the relevance feedback query was not to take the centroid of all relevant documents; the results improved if instead the query was constructed by the centroid of a smaller set of documents. A new strategy for calculating the relevance feedback query was therefore devised; the centroid of all relevant documents was calculated, and the new query was formed by taking the top n documents closest to this centroid.

Two different learning algorithms were used; KNN and ANN. The KNN method, which has no free parameters, produced the best results. This particular regression problem was not well suited for the neural network: it has many output variables, many free parameters, and few training examples. One important property of the methods is that they are thresholded, to make sure that the similarity between queries are sufficiently high before trying to use relevance feedback information from a past query.

Random Indexing Bag-of-Words and Bag-of-Concepts were both used for representing text documents in a text categorisation task. Random Indexing was used for representing the Bag-of-Concepts, which is an iterative method for refining an originally sparse random representation of a term or a document.

The best performance in terms of the $\mathcal{F}_1$ measure was achieved when combining the two representations, which was found by searching for the best combination of the two. The question of how to combine two or more representations in a given situation is still open.

Bag-of-Concept representations using Random Indexing can help the text categorisation task, if combined properly with Bag-of-Words.

Inverted Files We used inverted files for Collaborative Filtering, for the purposes of managing a large amount of users and items in the memory-based framework. First, it was found that several similarity measures could be expressed in a form suited for inverted retrieval. The inverted file search algorithms required less memory resources, was more efficient in terms of speed, and gave results comparable to or better than the in-memory algorithm.

We have also implicitly outlined a general method of inverting a similarity function by the use of precomputed static arrays and rearranging partial sums. For future research, one interesting question is what properties a vector similarity or distance function must have, in order to be inverted.

It can be concluded that inverted files are well suited for Collaborative Filtering problems, and that there is little reason to use the traditional in-memory vector approach for memory-based CF when the data is sparse.

Incremental Collaborative Filtering The incremental algorithm for Collaborative Filtering was primarily designed for client devices where resources are scarce and the time to make computations is limited due to low processor speed and drain of battery power.

The method incrementally updates the predictions based on new data, by the use of the past predictions and the subset of the neighbouring users. From the experiments, we conclude that there was little difference between making predictions from the server and using the incremental method. In the initial phase, when there were very few ratings, the server-based method performed better.

Integration We described the architecture and implementation of a system for information filtering, modular in the sense that prediction algorithms can be added and removed at any time. Documents and profiles are represented by feature vectors whose contents are determined outside the system. As such, the

system is capable of supporting a variety of possible representations, and a wide class of filtering algorithms.

8.1 Future Work

In this thesis, some algorithms and representations for personalised information access have been discussed, and all of these can be improved and further developed as outlined in the previous chapters.

Personalised information access systems tries to learn from user behaviour, in order to present more relevant information than what would be possible if the system had no knowledge of the user's interests or preferences. There is an inherent problem with this approach: people continuously learn and experience more things every day, without being connected to the system that tries to learn from their behaviour. There will perhaps always be a mismatch between what the system believes is relevant to the user and what the user has experienced after the last session with the system.

An important pointer for future work could therefore be to make more transparent the system's model of the user, and make it possible to edit the model. This is not a new thought, but it should perhaps be worthwhile to revisit.

Another aspect that would perhaps be beneficial to the user is an explanation or a description of why a particular item has been recommended or presented at the top of a search result list. If the user is better informed of the reasons for a particular recommendation or search result, the user can better judge the quality of the presented information.

8.2 Final Remarks

How information is represented is important. This is highlighted by the fact that it takes a lot of effort to improve learning algorithms beyond a certain point.

Relevance feedback information as represented by a set of optimised queries in a LSI space can be beneficial for the purpose of improving novel queries. Combining traditional word-based representations with concept-based ones using Random Indexing for categorisation can be beneficial; the difference was however small and more research is needed to be conclusive on this matter.

It has also been demonstrated that it is possible to implement large scale use of collaborative filtering techniques. It was furthermore observed that this methodology should be possible to apply to any inner-product based calculation operating on sparse vectors.

From the work on incremental collaborative filtering for mobile devices, we anticipate that prediction formulas based on a linear combination of similarities can be used in a distributed environment by the use of incremental updates.

In summary this thesis has demonstrated that it is feasible to combine different representations and to implement those using scalable algorithms for personalised information access.

Bibliography

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern information retrieval*. ACM Press / Addison-Wesley.

Baker, D., & McCallum, A. (1998). Distributional clustering of words for text classification. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and Development in Information Retrieval* (pp. 96–103).

Bartell, B. (1994). *Optimizing ranking functions: A connectionist approach to adaptive information retrieval* (Ph.D. Thesis). Department of Computer Science and Engineering, The University of California.

Bartell, B., Cottrell, G., & Belew, R. (1998). Optimizing similarity using multi-query relevance feedback. *Journal of the American Society for Information Science*.

Basu, C., Hirsh, H., & Cohen, W. (1998). Recommendation as classification: Using social and content-based information in recommendation. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence.*

Baudisch, P. (1999). Joining collaborative and content-based filtering. In *Interacting with Recommender systems, Online Proceedings of the CHI '99 workshop.*

Bayer, R., & Unterauer, K. (1977, March). Prefix B-trees. *ACM Transactions on Database Systems*, *2*(1), 11–26.

Belkin, N., & Croft, W. B. (1992). Information filtering and information retrieval: two sides of the same coin? *Communications of the ACM*, *35*(12), 29–38.

Berry, M., Dumais, S., & Brien, G. O. (1995). Using linear algebra for intelligent information retrieval. *SIAM Review*.

Berry, M., Dumais, S., & Letsche, T. (1995). Computational methods for intelligent information access. In *Proceedings of Supercomputing '95.*

Billsus, D., & Pazzani, M. (1998). Learning collaborative information filters. In *Proceedings of the 15th International Conference on Machine Learning* (pp. 46–54). Morgan Kaufmann.

Bingham, E., & Mannila, H. (2001). Random projection in dimensionality reduction: applications to image and text data. In *Knowledge Discovery and Data*

Mining (pp. 245–250).

Bishop, C. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Breese, J., Heckerman, D., & Kadie, C. (1998, July). Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence.* Madison, WI: Morgan Kaufman Publisher.

Buckley, C., & Lewit, A. F. (1985). Optimization of inverted vector searches. In *Proceedings of the 8th annual international ACM SIGIR conference on Research and Development in Information Retrieval.*

Bush, V. (1945). As We May Think. *The Atlantic Monthly*, *176*(1), 101–108.

Cai, L., & Hofmann, T. (2003). Text categorization by boosting automatically extracted concepts. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and Development in Information Retrieval* (pp. 182–189).

Chang, C.-C., & Lin, C.-J. (2001). *LIBSVM: a library for support vector machines.* (Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`)

Chen, H. (1995). Machine learning for information retrieval: Neural networks, symbolic learning, and genetic algorithms. *Journal of the American Society for Information Science*.

Cleverdon, C., Mills, J., & Keen, M. (1966). *Factors determining the performance of indexing systems: ASLIB Cranfield Research Project. volume 1: Design.* Cranfield: ASLIB Cranfield Research Project.

Cöster, R. (2002a). *The architecture and implementation of a system for collaborative and content-based filtering* (Tech. Rep. No. DSV Report Series 2002-032). Department of Computer and System Sciences.

Cöster, R. (2002b, December). *Learning and scalability in personalized information retrieval and filtering* (Ph. Lic. Thesis). Department of Computer and System Sciences, Stockholm University.

Cöster, R., & Asker, L. (2000). A similarity-based approach to relevance learning. In *Proceedings of the 14th European conference on Artificial Intelligence* (pp. 276–280). IOS Press.

Cöster, R., Gustavsson, A., Olsson, T., & Rudström, A. (2002). Enhancing web-based configuration with recommendations and cluster-based help. In *Workshop on Recommendation and Personalisation in ecommerce, at 2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems* (pp. 30–39).

Cöster, R., & Svensson, M. (2002). Inverted file search algorithms for collaborative filtering. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and Development in Information Retrieval* (pp. 246–252). ACM Press.

Cöster, R., & Svensson, M. (2005). Incremental collaborative filtering for mobile devices. In *Proceedings of the 2005 ACM Symposium on Applied Computing*. ACM Press.

Crestani, F. (1993). Learning strategies for an adaptive information retrieval system using neural networks. In *IEEE International Conference on Neural Networks*.

Cristianini, N., & Shawe-Taylor, J. (2000). *An introduction to support vector machines*. Cambridge, UK: Cambridge University Press.

Deerwester, S., Dumais, S., Furnas, G., Landauer, T., & Harshman, R. (1990). Indexing by latent semantic analysis. *Journal of the American Society for Information Science*.

Dumais, S. (1991). Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments and Computers*.

Dumais, S. (1992). LSI meets TREC: A status report. In *Text REtrieval Conference* (p. 137-152).

Dumais, S. (1994). Latent semantic indexing and TREC-2. In *NIST Special Publication 500-215: The Second Text REtrieval Conference (trec-2)*.

Dumais, S. (1995). Latent semantic indexing (LSI): TREC-3 report. In *NIST Special Publication 500-226: Overview of the Third Text REtrieval Conference (trec-3)*.

Dumais, S., Platt, J., Heckerman, D., & Sahami, M. (1998). Inductive learning algorithms and representations for text categorization. In *Proceedings of ACM -CIKM98* (pp. 148–155).

Folk, M. J., Zoellick, B., & Riccardi, G. (1998). *File structures: An object-oriented approach with c++* (3rd ed.). Addison-Wesley.

Goldberg, D., Nichols, D., Oki, B., & Terry, D. (1992). Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, *35*(12), 61–70.

Harman, D. (1992). Relevance feedback revisited. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and Development in Information Retrieval*.

Hearst, M. (1999). User interfaces and visualization. In R. Baeza-Yates & B. Ribeiro-Neto (Eds.), *Modern Information Retrieval*. ACM Press / Addison-Wesley.

Hecht-Nielsen, R. (1994). Context vectors: general purpose approximate meaning representations self-organized from raw data. In J. Zurada, R. Marks II, & C. Robinson (Eds.), *Computational intelligence: Imitating Life* (pp. 43–56). IEEE Press.

Herlocker, J., Konstan, J., Borchers, A., & Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and Development in*

Information Retrieval.

Herlocker, J., Konstan, J., Terveen, L., & Riedl, J. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS).*, *22*(1), 5–53.

Hull, D. (1994). Improving text retrieval for the routing problem using latent semantic indexing. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and Development in Information Retrieval.*

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning.*

Johnson, W., & Lindenstrauss, J. (1984). Extensions of lipshitz mapping into hilbert space. *Contemporary Mathematics*, *26*, 189–206.

Kanerva, P., Kristofersson, J., & Holst, A. (2000). Random indexing of text samples for latent semantic analysis. In *Proceedings of the 22nd Annual Conference of the Cognitive Science Society* (p. 1036). Mahwah, New Jersey: Erlbaum.

Karlgren, J. (1990). *An algebra for recommendations, syslab working paper 179* (Tech. Rep.). Department of Computer and System Sciences, Stockholm University.

Karlgren, J. (1999). Stylistic experiments in information retrieval. In T. Strzalkowski (Ed.), *Natural Language Information Retrieval.* Dordrecht: Kluwer.

Karlgren, J., & Sahlgren, M. (2001). From words to understanding. In Y. Uesaka, P. Kanerva, & H. Asoh (Eds.), *Foundations of Real World Intelligence* (pp. 294–308). CSLI publications.

Kaski, S. (1998). Dimensionality reduction by random mapping: Fast similarity computation for clustering. In *Proceedings of the ijcnn'98, International Joint Conference on Neural Networks* (pp. 413–418). IEEE Service Center.

Konstan, J. A., Miller, B. N., Maltz, D., Herlocker, J. L., Gordon, L. R., & Riedl, J. (1997). Grouplens: Applying collaborative filtering to usenet news. *Communications of the ACM*, *40*(3), 77-87.

Lee, J. (1994). Properties of extended boolean models in information retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and Development in Information Retrieval.*

Lewis, D. (1992). An evaluation of phrasal and clustered representations on a text categorization task. In *Proceedings of the 15th annual international ACM SIGIR conference on Research and Development in Information Retrieval* (pp. 37–50).

Linden, G., Smith, B., & York, J. (2003). Industry report: Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Distributed Systems Online*, *4*(1).

Loeb, S., & Terry, D. (1992). Information filtering. *Communications of the ACM*,

35(12), 26–28.

Lyman, P., Varian, H. R., Charles, P., Good, N., Jordan, L. L., & Swearingen, K. (2003). *How much information? 2003*. http://www.sims.berkeley.edu/research/projects/how-much-info-2003/.

Maltz, D., & Ehrlich, K. (1995). Pointing the way: active collaborative filtering. In *Proceedings of the Conference on Computer-Human Interaction* (pp. 202–209).

McCallum, A., & Nigam, K. (1998). A comparison of event models for naive bayes text classification. In *AAAI-98 Workshop on Learning for Text Categorization*.

McCallum, A. K. (1996). *Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering*. (Software available at `http://www.cs.cmu.edu/~mccallum/bow`)

Melville, P., Mooney, R. J., & Nagarajan, R. (2002, July). Content-boosted collaborative filtering for improved recommendations. In *Eighteenth National Conference on Artificial Intelligence* (pp. 187–192). Edmonton, Canada.

Miller, B. N., Albert, I., Lam, S. K., Konstan, J. A., & Riedl, J. (2003). Movielens unplugged: experiences with an occasionally connected recommender system. In *Proceedings of the 8th international conference on Intelligent user interfaces* (pp. 263–266). ACM Press.

Mitchell, T. (1997). *Machine learning*. McGraw Hill.

Mooney, R. J., & Roy, L. (2000). Content-based book recommending using learning for text categorization. In *Proceedings of DL -00, 5th ACM Conference on Digital Libraries* (pp. 195–204). San Antonio, US: ACM Press, New York, US.

Mostafa, J., Mukhopadhyay, S., Palakal, M., & Lam, W. (1997). A multilevel approach to intelligent information filtering: model, system, and evaluation. *ACM Transactions on Information Systems (TOIS)*, *15*(4), 368–399.

Oard, D. (1997). The state of the art in text filtering. *User Modeling and User-Adapted Interaction*, *7*(3).

Papadimitriou, C. H., Raghavan, P., Tamaki, H., & Vempala, S. (1998). Latent semantic indexing: A probabilistic analysis. In *Proceedings of the 17th ACM Symposium on the Principles of Database Systems* (pp. 159–168). ACM Press.

Persson, P., Espinoza, F., Fagerberg, P., Sandin, A., & Cöster, R. (2003). Geonotes: A location-based information system for public spaces. 151–173.

Popescul, A., Ungar, L., Pennock, D., & Lawrence, S. (2001, August 2–5). Probabilistic models for unified collaborative and content-based recommendation in sparse-data environments. In *17th Conference on Uncertainty in Artificial Intelligence* (pp. 437–444). Seattle, Washington.

Procter, R., & al. et. (1999). Select: Social and collaborative filtering of web documents and news. In A. Kobsa & C. Stephanidis (Eds.), *Proceedings of*

the fifth ercim workshop on "User Interfaces for All " (pp. 23–37).

Raghavan, V., & Sever, H. (1995). On the reuse of past optimal queries. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and Development in Information Retrieval.*

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P., & Riedl, J. (1994). Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the conference on Computer supported cooperative work* (pp. 175–186). ACM Press.

Resnick, P., & Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, *40*(3), 56–58.

Rijsbergen, C. J. van. (1979). *Information retrieval*. Butterworth.

Rijsbergen, C. J. van. (2004). *The geometry of information retrieval*. Cambridge University Press.

Robertson, S., & Sparck-Jones, K. (1976). Relevance weighting of search terms. *Journal of the American Society for Information Science*.

Rocchio, J. (1971). The SMART retrieval system: Experiments in automatic document processing. In (chap. 313–323). Prentice-Hall.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*, 386-408.

Rudström, A., Svensson, M., Cöster, R., & Höök, K. (2004). Mobitip: Using bluetooth as a mediator of social context. In *Ubicomp 2004 Adjunct Proceedings.*

Sahlgren, M., & Cöster, R. (2004). Using bag-of-concepts to improve the performance of support vector machines in text categorization. In *Proceedings of the 20th International Conference on Computational Linguistics.*

Salton, G. (1984). *The use of extended boolean logic in information retrieval* (Tech. Rep.). Cornell University.

Salton, G., & Buckley, C. (1990). Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*.

Salton, G., & McGill, M. (1983). *Introduction to modern information retrieval*. McGraw-Hill.

Sarwar, B., Karypis, G., Konstan, J., & Reidl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *WWW '01: Proceedings of the tenth international conference on World Wide Web* (pp. 285–295). ACM Press.

Sarwar, B., Karypis, G., Konstan, J., & Riedl, J. (2000). Application of dimensionality reduction in recommender system – a case study. In *ACM WebKDD 2000 Web Mining for E-Commerce Workshop.*

Sarwar, B., Konstan, J. A., Borchers, A., Herlocker, J., Miller, B., & Riedl, J. (1998, Nov). Using filtering agents to improve prediction quality in the grouplens research collaborative filtering system. In *Proceedings of the 1998 Conference on Computer Supported Cooperative Work.*

Savoy, J. (2003). Cross-language information retrieval: Experiments based on clef 2000 corpora. *Information Processing Management*, *39*(1), 75–115.

Sebastiani, F. (2002). Machine learning in automated text categorization. *ACM Computing Surveys*, *34*(1), 1–47.

Shardanand, U., & 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* (Vol. 1).

Sparck-Jones, K., Walker, S., & Robertson, S. E. (2000). A probabilistic model of information retrieval: development and comparative experiments - part 2. *Information Processing and Management*, *36*(6), 809-840.

Specification of the Bluetooth system. (2003, November). http://www.bluetooth.org.

Strzalkowski, T. (Ed.). (1999). *Natural language information retrieval*. Dordrecht: Kluwer.

Svensson, M., Höök, K., Laaksolahti, J., & Waern, A. (2001). Social navigation of food recipes. In *Proceedings of the SIGCHI conference on Human factors in computing systems* (pp. 341–348).

Syu, I., Lang, S., & Deo, N. (1996). Incorporating latent semantic indexing into a neural network model for information retrieval. In *International Conference on Information and Knowledge Managements (1996)*.

Tian, L. F., & Cheung, K. (2003, April). Learning user similarity and rating style for collaborative recommendation. In *Proceedings of European Conference on Information Retrieval Research* (p. 135-145). Pisa, Italy.

Tveit, A. (2001). Peer-to-peer based recommendations for mobile commerce. In *Proceedings of the 1st international workshop on Mobile commerce* (pp. 26–29). ACM Press.

Vapnik, V. (1995). *The nature of statistical learning theory*. Springer-Verlag New York, Inc.

Vogt, C., Cottrell, G., Belew, R., & Bartell, B. (1997). *User lenses - achieving 100 % precision on frequently asked questions* (Tech. Rep.). UCSD CSE.

Williams, H. E., & Zobel, J. (1999). Compressing integers for fast file access. *The Computer Journal*, *42*(3), 193–201.

Witten, I. H., & Frank, E. (2000). *Data mining: Practical machine learning tools and techniques with java implementations*. San Francisco: Morgan Kaufmann.

Witten, I. H., Moffat, A., & Bell, T. C. (1999). *Managing gigabytes: Compressing and indexing documents and images* (2nd ed.). Morgan Kaufmann Publishing.

Yan, T., & Garcia-Molina, H. (1999). The SIFT information dissemination system. *ACM Transactions on Database Systems (TODS)*, *24*(4), 529–565.

Yang, Y., & Liu, X. (1999). A re-examination of text categorization methods.

In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and Development in Information Retrieval* (pp. 42–49). ACM Press.

Yang, Y., & Pedersen, J. (1997). A comparative study on feature selection in text categorization. In *Proceedings of the 14th International Conference on Machine Learning* (pp. 412–420).

Zipf, G. K. (1949). *Human behaviour and the principle of least effort: an introduction to human ecology*. Addison-Wesley.