NTNU – Trondheim
Norwegian University of
Science and Technology

Collaborative Filtering in the News Domain with Explicit and Implicit Feedback

Dag Einar Monsen
Patrick Heia Romstad

Master of Science in Computer Science
Submission date: June 2014
Supervisor: Jon Atle Gulla, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Abstract

In online recommender systems, we use computerized algorithms to present articles targeted at the preferences of each individual user. One such technique, called collaborative filtering, works by selecting articles that is preferred by users with preferences similar to those of the target user. There are different ways to determine a users preference. A common practice is to ask for it, for instance by offering the user to provide a rating from one to five. Users are, however, often reluctant to provide such information. From a usability perspective, it is better to look at their behaviour, or implicit feedback, and use that to infer their preferences.

The main goal of this thesis is to determine whether we can use such implicit feedback in order to improve the accuracy a collaborative filtering method. We have implemented and tested a number of standard machine learning algorithms for mapping implicit feedback to explicit ratings. These algorithms are applied to a news data set consisting of user interactions collected while reading and rating news articles. After the mapping techniques have inferred possible new ratings, the next step is to generate recommendations, which is done with a matrix factorization algorithm. To measure the effect of adding implicit feedback to the recommendations we calculate the root mean square error (RMSE).

We have developed a fast and scalable collaborative filtering component and integrated it with a news aggregation service called SmartMedia. The component uses a combination of nearest neighbor search on articles with more than 30% correlation and a time threshold of 20 seconds to infer ratings based on user interactions collected. We show that these methods improve the RMSE by 5.14% compared to using only ratings users have given.

Our work shows that utilizing implicit feedback can improve recommendation accuracy in the news domain. The techniques we have developed require less explicit rating to be given from users in order to create good recommendations. Thus we believe that the next generation of recommender systems will be less dependent on explicit rating while still being able to provide high quality recommendations.

Sammendrag

I nettbaserte anbefalingssystemer kan vi bruke algoritmer for å presentere en samling artikler tilpasset hver individuelle bruker. Én slik teknikk, kalt collaborative filtering, fungerer ved å anbefale artikler foretrukket av brukere med samme preferanser som målbrukeren. Det er ulike måter å avgjøre slike preferanser. En vanlig praksis er å spørre om det, for eksempel ved å tilby brukeren å gi en karakter fra en til fem. Brukere er imidlertidig ofte motvillige til å oppgi slik informasjon. Fra et brukbarhetsperspektiv er det bedre å se på deres interaksjoner, eller implisitte brukerdata, og bruke det til å gi anbefalinger.

Hovedmålet med denne masteroppgaven er å avgjøre om vi kan bruke slik implisitt brukerdata for å forbedre nøyaktigheten til et anbefalingssystem. Vi har implementert og testet ulike maskinlæringsalgoritmer som oversetter implisitt brukerdata til eksplisitte karakterer. Alle algoritmene har blitt testet på et nyhetsdatasett som består av brukerinteraksjoner som har blitt innsamlet mens brukerne har lest og gitt karakterer til nyhetsartiklene. Etter at brukerdata har blitt oversatt til karakterer, er det neste steget å lage anbefalinger ved hjelp av matrisefaktorisering. For å måle effekten av å utnytte brukerdata til å lage anbefalinger, beregner vi rotmiddelkvadratavviket (RMSE).

Vi har utviklet en kjapp og skalerbar anbefalingskomponent, og integrert den med en nyhetsinnsamler kalt SmartMedia. Komponenten bruker en kombinasjon av nærmeste nabosøk på nyhetsartikler med mer enn 30% korrelasjon og en tidsterskel på 20 sekunder for å inferere nye karakterer basert på innsamlet brukerdata. Vi viser at disse metodene forbedrer RMSE med 5.14% sammenliknet med metoder som kun utnytter eksplisitte karakterer.

Arbeidet vårt viser at implisitt brukerdata kan forbedre anbefalingsnøyaktigheten i nyhetsdomenet. Teknikkene vi har utviklet krever færre karakterer fra brukere for å kunne gi gode anbefalinger. Vi tror derfor at neste generasjons anbefalingssystemer vil være mindre avhengige av karakterer for å kunne gi anbefalinger av høy kvalitet.

Preface

This master's thesis is the culmination of our work, carried out during the 10^{th} semester of our Master of Computer Science degree at the Norwegian University of Science and Technology, NTNU.

We wish to thank our supervisor Professor Dr. Jon Atle Gulla and co-supervisor Özlem Özgöbek at the Departement of Computer and Information Science, Norwegian University of Science and Technology for helpful guidance throughout the project.

Trondheim, June 5, 2014 - Dag Einar Monsen and Patrick Heia Romstad

Contents

List of Figures

Part I

Introduction

Chapter 1

Introduction

In this chapter we introduce our thesis. In Section 1.1 we explain our motivation to work with recommender systems in the news domain. In Section 1.2 we discuss what problems we have investigated in our thesis and in Section 1.3 we explain how we approached these problems. Then we present a summary of our findings in Section 1.4 and finally, in Section 1.5 we outline the structure of our thesis.

1.1 Background and Motivation

With the advent of the Internet, the media industry has found itself going through a radical transformation. In the scope of the last two decades, traditional printed newspapers have seen their readership decline as a result of competition by online publishing. Using mobile devices with Internet access, people can read news articles in the moment they are published – from anywhere in the world. This transformation naturally implicates many challenges for newspaper companies. It does, however, also involve many opportunities for those who are able to adapt. A printed newspaper is usually composed of articles selected to satisfy as many readers as possible. One can target a certain market segment, but targeting a single user is not feasible in print. In the electronic medium, on the other hand, a newspaper company can look at the reading patterns of every single user, and make an estimate of which articles they will find to be interesting. That is the domain of news recommender systems. There are many different ways to make such an estimate. A common method, known as collaborative filtering, works by the assumption that a user will prefer articles that are preferred by similar users, where users are assumed to be similar if they have expressed similar preferences to items in the past. Users often express these preferences as a numeric rating from 1-5, or on a different scale. Users are, however, often reluctant to provide such explicit feedback, and we cannot expect users to rate every article they read. In addition, we can look at how users interact with a system. How much time do they spend reading an article? Do they share it with their friends? This kind of information does not require an explicit action from the user, it is merely collected as he or she uses the system. In this master's thesis, we explore how this implicit feedback can be used to improve the accuracy of a recommender.

1.2 Problem Statement

News recommendation involves a set of unique challenges, especially the lack of explicit feedback to many articles. In our work, we explore ways to make use of implicit feedback available to improve the accuracy of a collaborative filtering algorithm, and how to integrate these methods to a real system. Formally, we attempt to answer the following research questions:

R1: How can we use implicit feedback to improve news recommendation accuracy in collaborative filtering?

Specifically, we want to investigate how we can decrease the Root Mean Squared Error (RMSE) of the recommender system using implicit feedback. The decrease in RMSE can then be compared to other research papers that consider implicit feedback as a method to increase recommendation accuracy.

R2: How do different techniques for including implicit feedback compare?

Implicit feedback in this thesis consists of user-article interactions that can be collected without any overhead to the user, while explicit feedback are ratings given explicitly by the user after reading an article. The techniques we compare utilize implicit feedback that only exists in the data set given, and we will therefore not utilize information about the users from other sources. Consequently, the techniques convert implicit feedback to explicit feedback through mapping in such a way that we can run matrix factorization recommender algorithms with the resulting explicit feedback.

R3: How can a collaborative filtering component be integrated with the SmartMedia news aggregator?

As the SmartMedia news aggregator utilizes a variety of filtering methods, we need to implement our component as a standalone application that supports a standard interface. The SmartMedia news aggregator can then control the priority between different filters to give an optimal set of articles.

1.3 Approach

To find solutions for these problems, we need a deep understanding of the problem space and the domain in focus. Based on an extensive knowledge search, we implement a number of standard machine learning algorithms for mapping from implicit to explicit ratings. These are applied to a news data set consisting of user interactions collected while reading and rating news articles.

The evaluations are done with the prospect of integrating a recommendation component to the SmartMedia news aggregator. With this integration we can present novel and interesting articles to users based on their reading habits.

1.4 Results

The evaluations demonstrated that given the right mapping technique, mapping implicit feedback to explicit feedback can improve accuracy of recommendations in the news domain. In addition, the mapping technique that improved accuracy the most, is a combination of using nearest neighbor search with one neighbor and a time threshold of 20 seconds, which had a 5.14% improvement. Our results also show mapping techniques that looks for global patterns in the dataset, as opposed to patterns for each item, is detrimental for recommendation accuracy.

Further, we developed a stand-alone collaborative filtering component that integrates with the SmartMedia application. The component consists of a combination of a mapping technique and a parallel matrix factorization algorithm based on stochastic gradient descent. It is scalable and fast, computing recommendations in less than 150 milliseconds. The components implements a standard REST API, and can easily be replaced by improved components in the future.

1.5 Thesis Structure

Chapter 2 - The SmartMedia Project: A quick overview of the NTNU SmartMedia Project and how we contribute to it.

Chapter 3 - Recommender Systems: An introduction to news recommendations and collaborative filtering.

Chapter 4 - Feedback: A thorough discussion about feedback and the differences between explicit and implicit feedback.

Chapter 5 - Matrix Factorization: Describes matrix factorization and how it can be used as a collaborative filtering component.

Chapter 6 - Tensor Factorization: Describes tensor factorization and how it can be used as a collaborative filtering component.

Chapter 7 - Mapping Techniques: Explanation of different techniques that maps implicit feedback to explicit ratings.

Chapter 8 - Related Work: A survey of related work in recommender systems.

Chapter 9 - Implementation: Implementation details of the recommender and how the recommender is integrated to the SmartMedia application.

Chapter 10 - News Recommendation in the SmartMedia Application: Explains how feedback is represented, preprocessed and used to generate recommendations.

Chapter 11 - Evaluation of Mapping Techniques: Describes the evaluation setup and the results from our evaluations.

Chapter 12 - Integrational Issues: A discussion of the integration to SmartMedia in terms of modifiability and scalability.

Chapter 13 - Discussion and Conclusions: A final discussion including conclusions and suggestions for further work.

Chapter 2

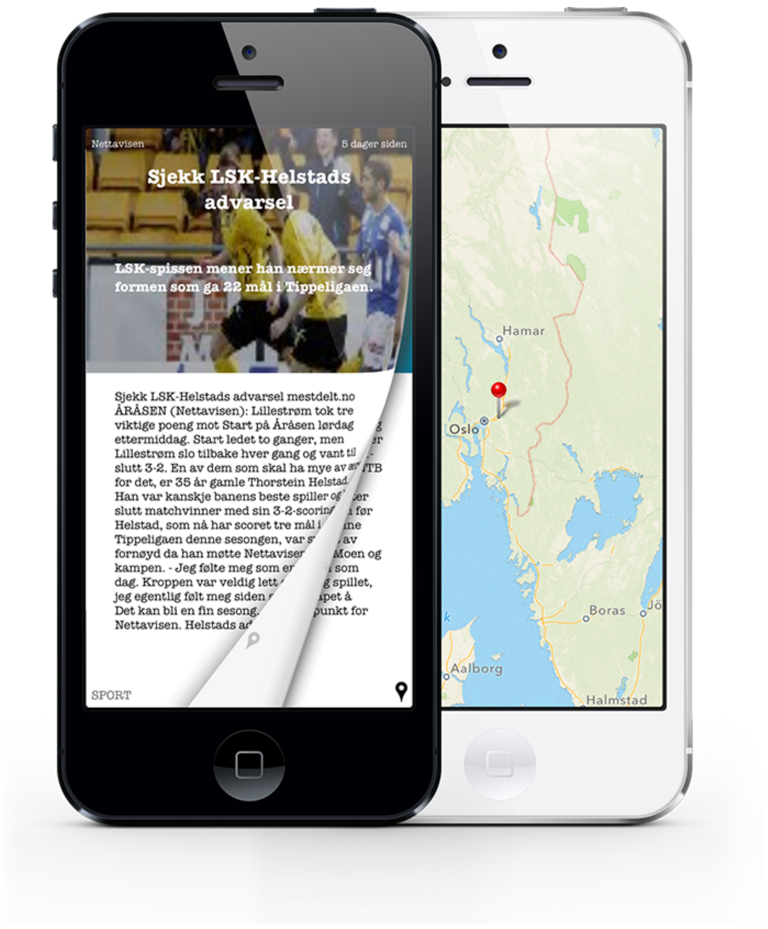
The SmartMedia Project

In this chapter we give a brief overview of the NTNU SmartMedia project. In Section 2.1, we outline the project background, in Section 2.2 we show the architecture of the news aggregator and in Section 2.3 we show how user profiling is done. Finally, in Section 2.4 we outline our contribution to the project.

2.1 Project Background

The NTNU SmartMedia project was established in 2012 in close collaboration with the Scandinavian media industry. The team is led by Prof. Jon Atle Gulla, and is divided into different research areas. Nafiseh Shabib is working on group recommendations. Özlem Özgöbek is working on data collection and semantics. Dag Einar Monsen and Patrick Heia Romstad are working on the collaborative filtering part of the recommender system. Dr. Jon Espen Ingvaldsen is working on content filtering and the underlying news index. Martin Akre Midstund and Marius Krakeli have been working on geospatial filtering. Pål-Christian Salvesen and Lars Smørås Høysæter are working on sentiment analysis of news articles. Kent Robin Haugen has made a client application for the iOS platform, and Amir Ghoreshi and Neberd Salimi are working on a web-based client application.

The main goal of the SmartMedia project is to create a mobile application that efficiently recommends news articles to users and is easy to use. An early version of the iPhone application is presented in Figure 2.1. Technologies involved in making


Nettavisen 5 dager siden

Sjekk LSK-Helstads advarsel

LSK-spissen mener han nærmer seg formen som ga 22 mål i Tippeligaen.

Sjekk LSK-Helstads advarsel mestdelt.no
ÅRÅSEN (Nettavisen): Lillestrøm tok tre
viktige poeng mot Start på Åråsen lørdag g
ettermiddag. Start ledet to ganger, men er
Lillestrøm slo tilbake hver gang og vant til -
slutt 3-2. En av dem som skal ha mye av ær NTB
for det, er 35 år gamle Thorstein Helstad.
Han var kanskje banens beste spiller og ter
slutt matchvinner med sin 3-2-scoring n før
Helstad, som nå har scoret tre mål i ne
Tippeligaen denne sesongen, var s av
fornøyd da han møtte Nettavisen Moen og
kampen. - Jeg følte meg som en som
dag. Kroppen var veldig lett g spillet,
jeg egentlig følt meg siden pet å
Det kan bli en fin sesong. punkt for
Nettavisen. Helstads ad

SPORT

FIGURE 2.1: iPhone application

this application are big data architectures, information retrieval, semantics, text analytics and sentiment analysis as well as various frameworks.

2.2 System Architecture

The news recommender system is structured as a traditional client-server architecture, as shown in Figure 2.2. The server side will aggregate news articles daily from all (89) major online newspapers in Norway by parsing their RSS feeds for all metadata, including the ingress. The aggregator proceeds by scraping the website of each newspaper for the full article content. All of these articles are written in Norwegian, and are provided in a semi-structured format. In some articles, we are able to obtain certain features like topic or category from the RSS feed. We also apply entity recognition to each article. Using entity recognition, we are able to extract locations, names of people and other interesting features which can then be used to support a number of recommendation techniques, including information

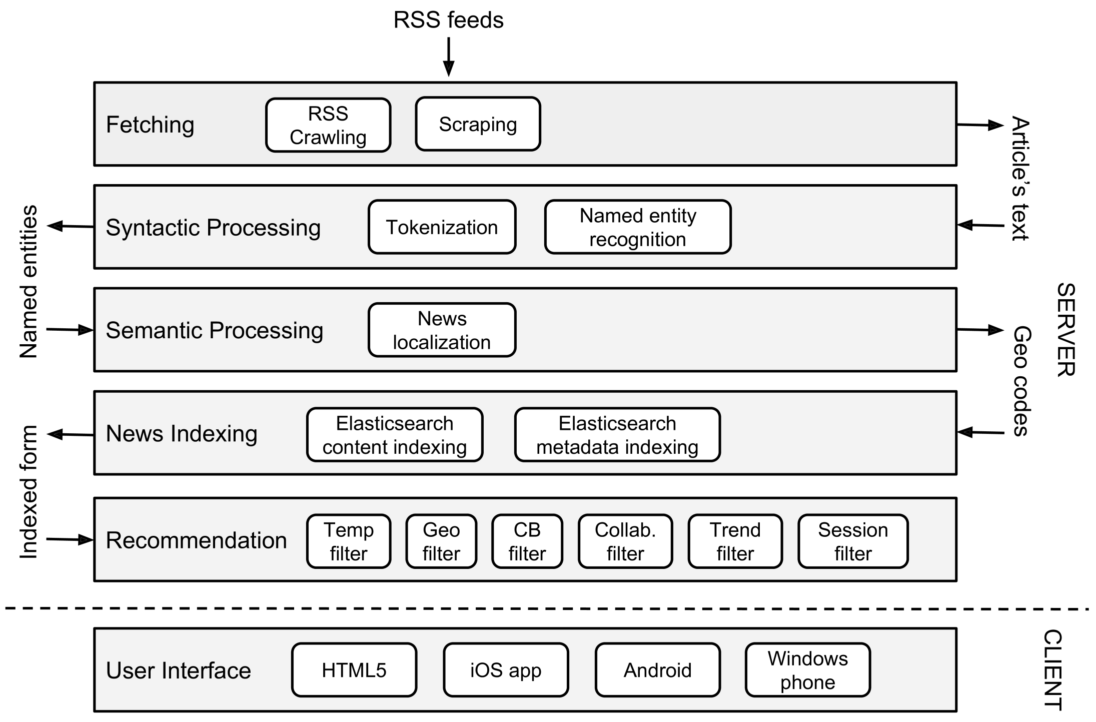


Figure 2.2: Overall architecture of the SmartMedia news application

filtering, geospatial filtering and content filtering. The recommendation process is discussed further in [1] and a diagram of this process is presented in Figure 2.3.

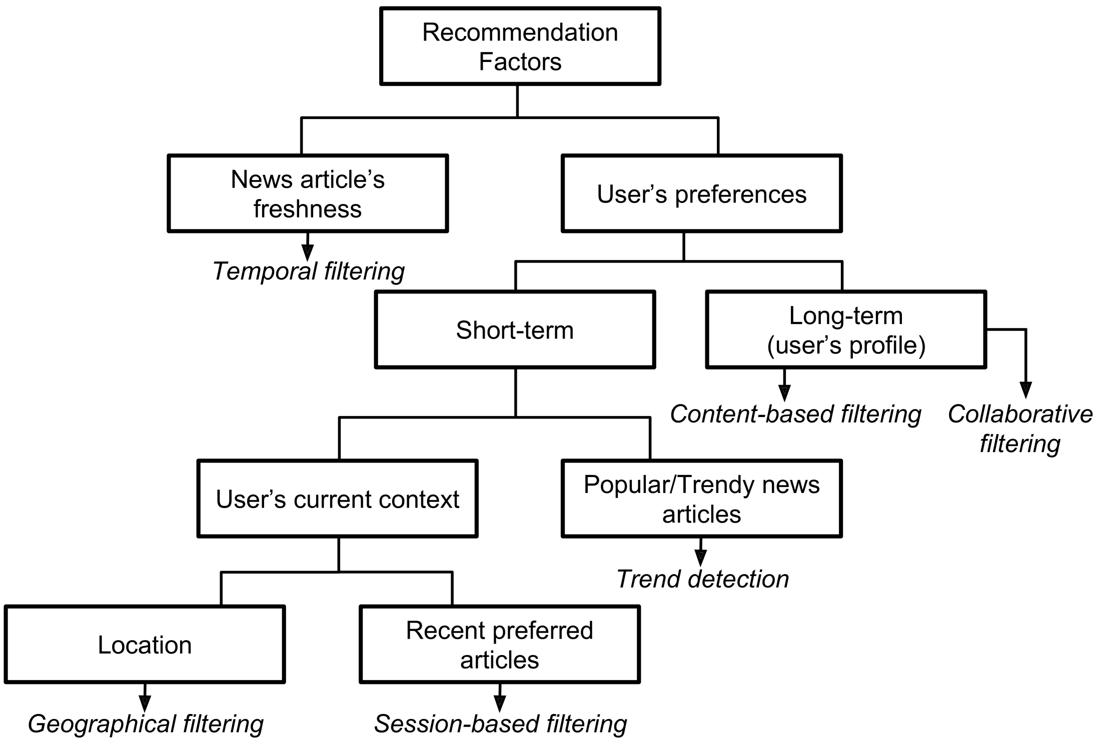


Figure 2.3: Different recommendation factors

The client application is designed using web technology, simplifying a cross-platform deployment. The application is independent of each publisher, and we plan to allow voluntary sign-ups, thus our knowledge about each client is limited to the data we collect. We allow each user to rate an article by a 1-5 star rating. We also take note of certain user interactions, which we elaborate further on in Section 4.2.

2.3 User Profiling

User profiling is important for the SmartMedia application in order to do individual recommendations for each user. A user profile is constructed for a chosen time interval, and if there already exists a user profile for this mobile device, the profiles are combined in such a way that the new profile is weighted more than the old one[2]. This way of constructing user profiles enables the recommender system to (i) do recommendations to new unregistered users since it constructs a user profile for each session, and (ii) give improved recommendations to registered users by combining the constructed user profiles. Therefore, the second (ii) step becomes crucial to build long-term user profiles. Examples of user interactions collected by the SmartMedia applications are shown in Table 2.1.

The user profile consists of two vectors $P =< \vec{C}, \vec{K} >$[2]. $\vec{C}$ is a category vector where each news category is given a weight that indicates how important this category is for the user. $\vec{K}$ is a content vector that weights key phrases and entities that the user might find interesting. Examples of $\vec{C}$ and $\vec{K}$ vectors are shown in Equation 2.1 and 2.2.

$$\vec{C} =< (''NEWS'', 98.0), (''TECH'', 50.0), (''SPORTS'', 15.4), (''STYLE'', 2.5) > \tag{2.1}$$

$$\vec{K} =< (''iPhone'', 20.0), (''Tyson'', 5.0), (''LG'', 3.4), (''Abra'', 0.41), \ldots > \tag{2.2}$$

While the $\vec{C}$ vector is limited to the number of categories, the $\vec{K}$ vector grows as the user reads news articles. Since only the highest weighted terms are relevant for the recommendation stage, the less important terms can be removed to ensure the

Table 2.1: Examples of user interactions used to construct user profiles

User interaction	**Description**
Open article	User opened full text version of article
Time spent article	Time the user spent viewing the article
Time spent preview	Time the user spent viewing the RSS version of article
Shared twitter	User shared article on Twitter
Starred article	User added the article to favorites

vector do not become unnecessary large. More information about SmartMedia's approach to user profiling is further described in [2].

These user profiles are currently not used by the collaborative filtering component, but resembles the user and item vectors created in the matrix factorization stage. The differences between these vectors and the user profiles are that the user and item vectors created by the collaborative filtering component comprises of the latent factors found, and do not have labels that describe them. It is therefore challenging to incorporate the user vectors created by our component with the user profiles created by the SmartMedia application.

2.4 Our Contribution

Our part in the NTNU SmartMedia Project is to implement an efficient and accurate collaborative filtering algorithm. To do so, we intend to incorporate implicit feedback as well as explicit feedback in the recommender engine and analyze the results of this implementation.

Part II

Background

Chapter 3

Recommender Systems

In this chapter we introduce the theory behind recommender systems and their applications. In Section 3.1 we outline the motivation behind recommender systems. In Section 3.2 we describe collaborative filtering and in Section 3.3 we reason about news recommendation and what distinguishes it from other domains.

3.1 Introduction

As the internet has become a primary source of information, finding what one is looking for can be a challenge. When looking for a specific piece of information, a user normally uses a search engine like Google or Bing to retrieve a list of results matching their query. A common use case, however, occurs when we are not looking for something specific, but merely something interesting. A simple way of providing such items is to retrieve the most popular items in a catalog. A more personalized approach, on the other hand, can recommend items based on the preferences of each individual user. Schafer et al.[3] argues that personal recommendations will increase sales by converting browsers into buyers, increasing cross-sell opportunities, and building customer loyalty. Linden et al.[4] also saw increased sales and customer retention at the online retailer Amazon.com by showing a list of similar items below each item. We normally divide recommender systems into two categories. In a content-based approach, users are often encouraged to state their topics of interest. In a news domain, this can for example be sports and politics. Items are then defined with a set of features, where a set of topics would be a feature of a news article. A recommender system would then

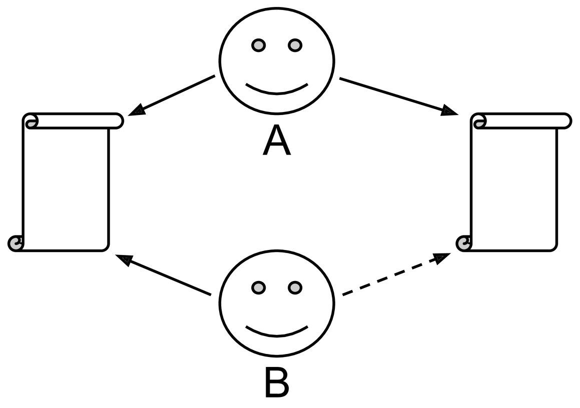


Figure 3.1: Simple example of collaborative filtering

be able to recommend articles by calculating the similarities between the users expressed interests and each item, thus returning the most similar ones.

Another method of providing recommendations is to look for similar users. If a user B has shown to have similar preferences to a user A, one might recommend a book to user B that user A finds interesting, and user B has not read yet, as shown by the dotted line in Figure 3.1. This method of providing recommendations is called collaborative filtering. A more thorough introduction to collaborative filtering is given in the next section.

In our work, we have implemented a collaborative filtering component and integrated it with the SmartMedia application, allowing users to read news selected to their personal preferences.

3.2 Collaborative Filtering

Collaborative filtering recommends items to a user by looking at similar users and recommend items that they have expressed an interest in. The basic form of collaborative filtering takes in a matrix of user-item ratings as input and produces two types of output: (i) a numerical prediction of the degree a user will like or dislike an item and (ii) a list of n recommended items.

Collaborative filtering can be divided in two categories: memory-based and model-based. Given a matrix of user-item ratings, memory-based collaborative filtering uses the matrix on each query to generate new item recommendations to the user. The most common method of memory-based collaborative filtering is neighborhood-based collaborative filtering with item-based or user-based top-n recommendations.

Collaborative filtering methods face certain challenges depending on the particular domain. In Section 3.3 we discuss the challenges unique to the news domain. There are, however, a range of challenges faced by collaborative filtering methods that are common to most domains. Data sparsity: data sets often consist of many more user-item combinations than there are interactions between these. The user-item matrix of the 100M netflix data set contained 17770 movies and 480189 users, resulting in a density of roughly 1.17%[5]. Such sparse data sets means that the algorithms have to make predictions based on very little information. This challenge is related to the cold start problem, which refers to what occurs when new users or items enter the system. These additions do not have any feedback as new users have not rated any items, and new items have not been rated. This is a problem, because the collaborative filtering methods need feedback in order to provide recommendations. Collaborative filtering methods also face a challenge when the scale of the system increases. As new users and items are added to the system, the user-item matrix thus increases. This means that the memory-based methods have to iterate over an increasing number of rows and columns in the user-item matrix. This will eventually lead to performance problems and new recommendations cannot be computed fast enough.

This problem led researchers to look at different methods for computing recommendations. Using machine learning techniques, an intermediate representation of the user-item matrix could be computed, and then used to make recommendations with less computational effort. This intermediate representation is called a model and we call these methods model-based. Model-based methods using matrix factorization has been shown to scale to hundreds of millions of entries in the user-item matrix[6, 7]. Matrix Factorization works by modeling ratings as a sparse matrix indexed by user ID and item ID, and then factorizing this matrix into a product of matrices of much lower dimensions. This smaller representation can then be used to estimate ratings very efficiently. We explore matrix factorization further in Chapter 5.

A challenge with matrix factorization surfaces when we want to include implicit data in the model. By allowing the matrix to contain more than a single feedback value in each cell, we effectively have what is called a tensor of three dimensions, as a matrix is merely what we call a two-dimensional tensor. For clarity, a single dimensional tensor of numbers is simply a vector. Thus, tensor factorization, as opposed to matrix factorization, deals with the factorization of a generalized tensor of N dimensions. Tensor factorization does, unfortunately, also involve a set of challenges. We discuss these challenges further, as well as methods to do tensor factorization for providing recommendations in Chapter 6.

An alternative to tensor factorization works by combining the feedback values prior to building a model. When we have both implicit and explicit feedback available, we can try to find patterns between them. From these patterns, we might be able to estimate the explicit feedback, which means that we no longer have to collect an explicit feedback value from a user – we can rely solely on his or her actions. If we are able to infer explicit feedback from implicit feedback, we might be able to improve the accuracy of a model computed using matrix factorization. This has been the main focus of our thesis, and we discuss these mapping techniques further in Chapter 7.

3.3 News Recommendation

Following the same approach as other recommender systems, the purpose of a news recommender system is to find and present news articles that a user might find interesting. Using the definition presented in [2], this can be formulated as

$$s : U \times A \to V \tag{3.1}$$

where s is the utility function, U the set of users, A the set of news articles and V is an ordered set of values representing the rating or preference of a user for an article. Then the purpose of a recommender system is to recommend an article a' that maximizes the utility function for the user.

$$a' = \arg\max_{a \in A} s(u, a) \tag{3.2}$$

There are many important factors to consider when recommending news articles due to the complexity of the news domain compared to other domains e.g. books, movies and songs. The recommender system must be robust and tackle changes in the news domain. Robustness is needed according to Gulla et al. [2] since news articles are unstructured and require a thorough analysis, and unlimited reach of news lead to changes in terminologies and topics over time.

Factors closer to the part of generating recommendations are location and freshness of the news articles. Location is used to give users recommended articles that are either close to their physical location or news about their hometown. Freshness is just as important — we need to present the latest news to users. The approach we make in SmartMedia are geospatial and temporal filtering, as presented in Figure 2.3.

The key challenge in the domain of news recommendation is item churn[6]. Item churn can be mitigated with memory-based methods, but as mentioned in the previous section, memory-based methods become infeasible when we have millions of users and items. For this reason, we use a model-based approach. However, since news articles are added and deleted at a high frequency, it becomes difficult to keep the model up to date.

Other challenges mentioned in [2] are cold start problems due to new and unread news and the ability to recommend new articles to a user in topics the user have never read (serendipity). Cold start problems in news recommendation is more difficult to handle than in more stable item set, such as movies, since news articles are continuously added. Serendipity is important in order to give the user more varied set of news articles.

Additionally, a problem occurs when the system aggregates news from many different sources. In this case, we might have multiple articles on the same subject. A collaborative filtering system does not necessarily see this similarity, and may thus recommend several articles on the same subject to a user. In this case, we can use a content-based filtering system to discover these similarities and group the articles written on the same subject. This is illustrated in Figure 3.2, where we recommend five articles.

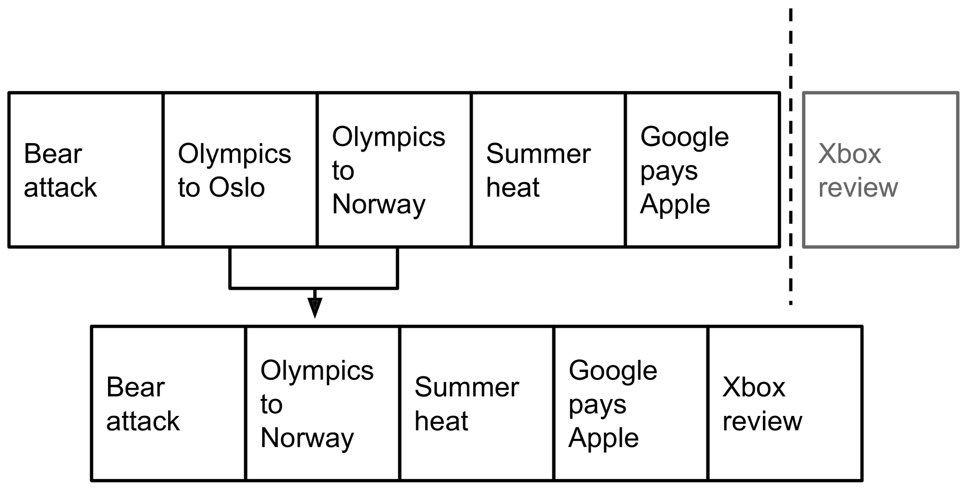


FIGURE 3.2: Example of grouping articles by content-based filtering

Chapter 4

Feedback

Feedback from users is essential for recommender systems, but collecting sufficient user information is a challenge. User information can be collected in two ways: explicit and implicit. In sections 4.1 and 4.2 we outline these two kinds of feedback, respectively.

4.1 Explicit Feedback

Explicit feedback is so named because the user provides it explicitly. This means that it depends on users willingness to provide such feedback. A five star ratings scale is an example of explicit ratings given by users; other examples are listed in Table 4.1. Explicit feedback is generally considered as more reliable than implicit feedback, but it also suffers from noise or inconsistencies[8]. Amatriain et al. provide an in-depth analysis of user rating noise in recommender systems in [9]. Some of their main findings were: (i) extreme ratings are more consistent than mild opinions and (ii) users are more consistent when items with similar ratings are grouped together. Knowing that extreme ratings are more consistent enables us to give higher weight to articles with extreme values when we are mapping implicit feedback to explicit feedback.

Another issue with explicit feedback in news recommendation is, according to Thurman[10], that users are often reluctant to give explicit feedback on news articles that can be used to construct and maintain user profiles. Therefore, a recommender system should not impose any requirements to the user to give explicit feedback, and the final SmartMedia application will only use implicit feedback to generate recommendations. However, explicit feedback will be collected to build the data set used in this master's thesis. How this feedback is collected and used to create user profiles should then be described, encouraging the test users to leave explicit feedback.

Furthermore, even though explicit feedback contains noise, it is generally accepted that it is more reliable than implicit feedback in most situations[9]. It will therefore be used as the true preference for articles in the data set used in our evaluation. This enables us to transform implicit feedback to an explicit rating through various techniques explained in Chapter 7.

TABLE 4.1: Common types of explicit and implicit feedback

Explicit feedback	**Implicit feedback**
Like or dislike buttons	Browsing history
Rating scales (e.g. stars)	Search patterns
Questionnaires	Mouse movements
Reviews	Time user spends on a page
	Keyboard actions
	Click behavior
	User shares an article

4.2 Implicit Feedback

Implicit feedback relies on collecting user information by analyzing user actions or content that users interact with. Examples of user actions are time spent reading an article, click behavior and time spent on moving the mouse/cursor. More examples are listed in Table 4.1. The main advantages of collecting implicit feedback are: (i) users do not have to actively engage to provide useful information to the recommender system and (ii) implicit feedback can be combined with explicit ratings to obtain a more accurate representation of user interests[11].

Implicit feedback has several challenges. Hu et al. [12] list four prime characteristics: (i) No negative feedback: When observing user actions we can infer which items users probably like, but we can not assume users dislike items they have not interacted with. (ii) Implicit feedback is noisy: We can only guess users preferences for items. (iii) Implicit feedback uses confidence, that is, how much confidence do we have about users preference and (iv) implicit feedback recommender systems requires appropriate evaluation methods. With implicit feedback we have to no way of clearly measuring what is a successful prediction, as opposed to explicit ratings where we can measure the success of a prediction with a numeric score like RMSE.

However, if we have enough implicit ratings we can assume that low feedback is negative feedback [13], contradicting the first characteristic of Hu et al. But it requires that the items can be grouped into instances that more feedback means higher preference. One example is TV-shows, where more feedback usually means that the user like the show and watches it every week. However, in the news domain this is not the case as users normally read articles only once, leaving this challenge still viable in the news domain.

Further, as we discussed in Section 4.1, explicit feedback is also noisy. Whereas explicit feedback noise mostly refers to the users preferences changing over time, implicit feedback noise refers to the error the recommender system has when trying to interpret the actions of a user.

In addition, given enough explicit ratings and implicit feedback, an appropriate mapping between explicit and implicit feedback is possible as shown in the news domain by [14, 15] and in the music domain in [13]. The main advantage of mapping instead of interpreting user actions is that we can have a higher confidence in our mapping if the correlation is high. We also have a clear metric for evaluating our results, since we can work on explicit ratings when evaluating the recommender system, but use implicit feedback to improve or add explicit ratings.

Chapter 5

Matrix Factorization

This chapter introduces matrix factorization as a technique to generate recommendations. Section 5.1 gives a motivation for use and a brief intro to matrix factorization. Then, singular value decomposition is described in Section 5.2. Stochastic gradient descent and alternating least squares are described in Section 5.3 and Section 5.4. At the end we discuss the tradeoffs between methods in Section 5.5.

5.1 Introduction

As we mention in Section 3.2, memory-based collaborative filtering systems met challenges with performance as data sets scaled into millions of users and items. This lead researchers and the industry to look at different ways to handle the scalability issue. As ratings could be expressed as a matrix indexed by users and items, matrix factorization could be a solution. By factorizing the user-item matrix into components of matrices in a much smaller dimension, one could approximate the missing values of the original user-item matrix. The low-dimension matrices in the model would then represent so called latent factors where each user and each item would have their own set of latent factors. Hence, matrix factorization is also sometimes referred to as Latent Semantic Indexing (LSI), or Latent Semantic Analysis (LSA). The latent, or hidden factors represent patterns found in the original user-item matrix. For example, in the domain of movie recommendations, the first cell in the latent factor vector of a movie could represent to which degree the movie contains elements of romance. The first cell in the latent factor vector of a user would then represent to which degree that user prefers romantic movies. When computing the estimated rating of that user to that movie, the latent factor vectors are multiplied, and thus if the latent properties of romance of both vectors are high, it directly translates to a high rating. The reason the factors are called latent is that we do not know what each factor represent. In some cases it could be something obscure like the amount of red colored houses appearing in the movie, or something incomprehensible to humans.

Matrix factorization involves a potential challenge when the data set is continuously changing. In situations where we have repeated additions and deletions, the model needs to be rebuilt at certain intervals to correctly reflect the data set. If the intervals are not often enough the quality of the recommendations can decline, and if they are too narrow the build time might become a computational bottleneck in the system, ultimately leading to lower user satisfaction.

Formally, matrix factorization decomposes two or more matrices such that when multiplied they are returned to the original matrix as shown in Equations 5.1 and 5.2, where M is the original matrix and U and V the matrices that will result in M when multiplied.

$$\overset{M}{\begin{bmatrix} 11 & 12 & 5 \\ 22 & 24 & 10 \\ 13 & 11 & 5 \end{bmatrix}} = \overset{U}{\begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 1 \end{bmatrix}} \times \overset{V}{\begin{bmatrix} 3 & 2 & 1 \\ 4 & 5 & 2 \end{bmatrix}} \tag{5.1}$$

$$M_{2,1} = U_{2,1} \times V_{1,1} + U_{2,2} \times V_{1,2} = 2 \times 3 + 4 \times 4 = 22 \tag{5.2}$$

In collaborative filtering, matrix factorization divides the user-item matrix in two matrices: the user matrix and the item matrix. Each user is associated with a vector $p_u \in R^f$ and each item with a vector $q_i \in R^f$, where f denotes the dimensionality of the latent factor space, meaning the number of latent factors. The vectors q_i and p_u represents the corresponding interest the user has to the factors of an item. By calculating the dot product of these vectors, we can approximate the rating given by user u to item i, denoted by $\hat{r}_{ui}$[16].

$$\hat{r}_{ui} = q_i^T p_u \tag{5.3}$$

5.2 Singular Value Decomposition

Singular value decomposition (SVD) is based on a theorem from linear algebra which states that a rectangular matrix M can be broken down into the product of three matrices - an orthogonal matrix U, a diagonal matrix Σ, and the transpose of the orthogonal matrix V. This is presented in Equation 5.4

$$M = U\Sigma V^T \tag{5.4}$$

where $U^TU = I$, $V^TV = I$. The columns of U are orthonormal eigenvectors of MM^T , the columns of V are orthonormal eigenvectors of M^TM, and Σ is a diagonal matrix containing the square roots of eigenvalues from U or V in descending order[17].

Table 5.1: Rating matrix for a SVD recommender

	Amy	Bob	Charlie	Dina
The Matrix	1	3	3	5
E.T.	4	3	5	2
iRobot	1	4	3	5
Hercules	3	5	2	1

The matrix in Table 5.1 consists of four users and four movies, and if we use SVD, it will be decomposed into quadratic 4×4 matrices, U, V^T and Σ as shown in Equation 5.5. Each row in matrices U and V^T represents the latent factors for each movie and user.

$$\begin{array}{cc} U & V^T \\ \begin{bmatrix} -0.4912 & 0.5042 & 0.1050 & 0.7025 \\ -0.5324 & -0.5123 & 0.6658 & -0.1041 \\ -0.5367 & 0.4641 & -0.1787 & -0.6816 \\ -0.4327 & -0.5177 & -0.7168 & 0.1761 \end{bmatrix} & \begin{bmatrix} -0.3500 & -0.6063 & 0.1659 & 0.6945 \\ -0.5798 & -0.1741 & -0.7506 & -0.2648 \\ -0.5193 & -0.1593 & 0.6327 & -0.5519 \\ -0.5213 & 0.7594 & 0.0931 & 0.3780 \end{bmatrix} \end{array}$$

$$\begin{array}{c} \Sigma \\ \begin{bmatrix} 12.7314 & 0 & 0 & 0 \\ 0 & 4.3442 & 0 & 0 \\ 0 & 0 & 2.6462 & 0 \\ 0 & 0 & 0 & 0.1913 \end{bmatrix} \end{array} \tag{5.5}$$

Then, if we want to use the two most important factors, which are those with the largest singular values, we can choose the two first columns in the decomposed matrices from Equation 5.5. The resulting matrices are presented in Equation 5.6.

$$\begin{array}{ccc} U_2 & V_2^T & \Sigma_2 \\ \begin{bmatrix} -0.4912 & 0.5042 \\ -0.5324 & -0.5123 \\ -0.5367 & 0.4641 \\ -0.4327 & -0.5177 \end{bmatrix} & \begin{bmatrix} -0.3500 & -0.6063 \\ -0.5798 & -0.1741 \\ -0.5193 & -0.1593 \\ -0.5213 & 0.7594 \end{bmatrix} & \begin{bmatrix} 12.7314 & 0 \\ 0 & 4.3442 \end{bmatrix} \end{array} \tag{5.6}$$

The matrix U_2 present the movies' and the matrix V_2^T present the users' positions in the latent factor space. Since we chose the two most important factors, the latent factor space is two-dimensional. Figure 5.1 presents how the factor space looks like, where the movies and users are plotted according to their positions given by the decomposed matrices. The latent factor space represents the similarity between the items and users, and when we want to give a recommendation to a new user, lets say Mark, we use his rating vector (the items he has rated) and multiply it with the item matrix and the inverse of the singular value matrix.

$$Mark_{2D} = Mark \times U_2 \times \Sigma^{-1} = [-0.5, -0.25] \tag{5.7}$$

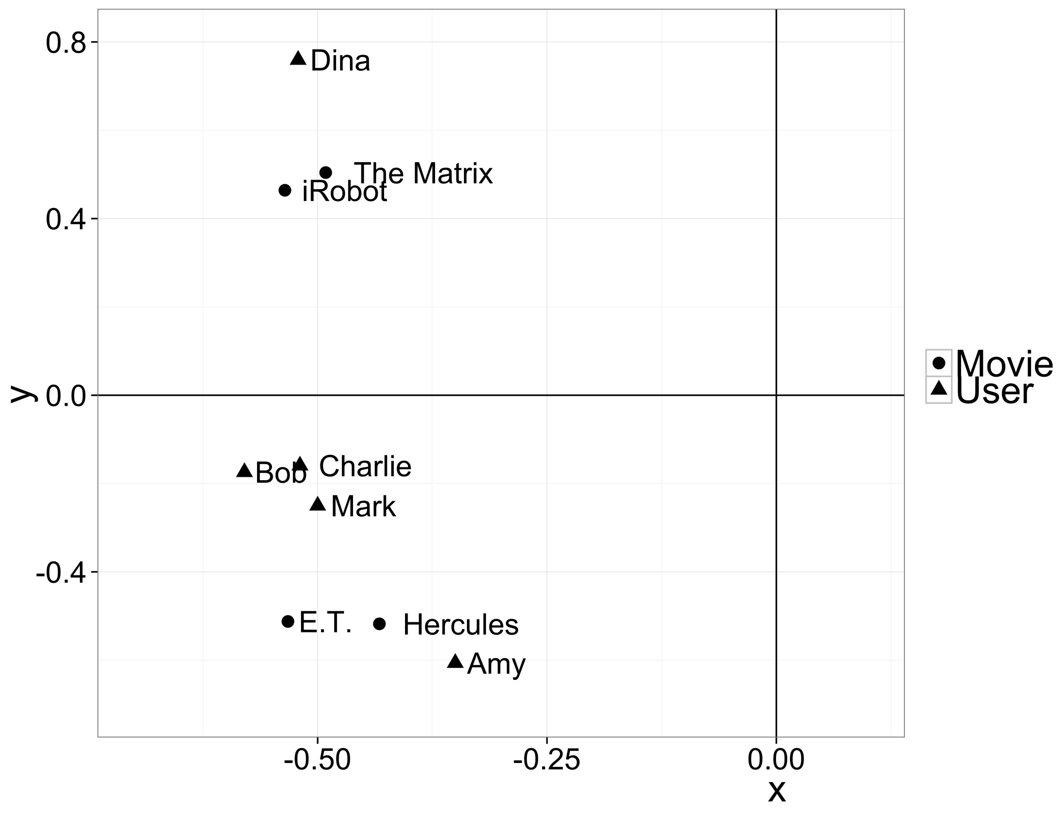


FIGURE 5.1: Users and movies in the latent space

His position in the latent factor space can be used in recommendations. By using his position in Figure 5.1, we could recommend the movies E.T. and Hercules. The position can also be used to find the closest neighbors that can be used in a neighborhood algorithm, which in his case are Bob and Charlie.

In essence, SVD reduces a high dimensional set of points to lower dimensions that exposes the substructure of the original data. Unfortunately for recommender systems, SVD is undefined when knowledge about the matrix is incomplete. Further, using the only known entries carelessly is highly prone to overfitting. In order to avoid overfitting, research suggests to model directly on the observed ratings while avoiding overfitting through an adequate regularized model[18] as shown in Equation 5.8

$$\min_{p*,q*} \sum_{(u,i)\in\kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\|p_u\|^2 + \|q_i\|^2) \tag{5.8}$$

where κ is the set of the (u, i) pairs for which r_{ui} is known and λ is a constant that controls the extent of regularization and is data-dependent. There are many ways to minimize Equation 5.8 and find p_u and q_u. In the next sections we look at stochastic gradient descent and alternating least squares.

5.3 Stochastic Gradient Descent

With stochastic gradient descent (SGD)[1], we minimize the error function in Equation 5.8 by an iterative approach. SGD differs from regular gradient descent (GD) since it does not have to iterate through all training examples in order to converge[19]. In some cases it might be less precise than GD, but with large data sets it is superior in terms of performance. The pseudocode for the SGD algorithm is given in Algorithm 1. We pass two parameters to the SGD algorithm; a vector of latent factors ω, which are the parameters of the error function E representing the difference between the actual rating and the estimated rating. The second parameter is the learning rate α, which tells how big steps we want to take towards the minimum. The learning rate depends on the possible values of a rating. A common value is 0.001. Given these two parameters, SGD calculates an approximate minimum by calculating the gradient, or slope of a single training example, and shuffling the samples in each step. With this, the true gradient of $E(\omega)$ is approximated.

```
Data:
$\omega$ vector of latent factors
$\alpha$ learning rate
Result: $\omega$ converged vector of latent factors
while above minimum threshold do
    randomly shuffle examples in the training set
    for $i = 1, 2, ..., |\omega|$ do
        $\omega_i := \omega_i - \alpha \nabla E_i(\omega_i)$
    end
end
```

Algorithm 1: Pseudo code for SGD

[1]http://en.wikipedia.org/wiki/Stochastic_gradient_descent

5.4 Alternating Least Squares

Alternating least squares (ALS) works by fixing either p_u or q_i in order to make Equation 5.8 quadratic and can therefore be solved optimally. Hence, ALS techniques rotate between fixing the q_u's and p_u's. For example, when all q_u's fixed, the system recomputes the p_u's by solving a least-squares problem, and vica versa[16]. A pseudo code of the algorithm is presented in Algorithm 2.

```
Data: Empty matrices p and q
Result: Matrices p and q
Initialize matrix q by assigning the average rating for the item as the first row,
and small random numbers for the remaining entries;
while $r_{ui} - q_i^T p_u >$ error criterion do
    Fix q;
    Solve p by minimizing the sum of squared errors;
    Fix p;
    Solve q by minimizing the sum of squared errors;
end
```

Algorithm 2: Pseudo code for ALS

5.5 ALS vs. SGD

In generally, ALS is more expensive than SGD because it has to solve a large number of linear least squares problem. However, according to Makari et al. [20] this computational overhead is acceptable when the rank of the factorization is sufficiently small. Further, big advantages of ALS over SGD is that ALS requires less parameters to be tuned, since SGD makes use of a step size sequence, and easy parallelization since either p_u or q_u are fixed.

According to experiments shown in [20], SGD is the preferred method when the step size sequence is chosen intelligently. In addition, SGD is less memory-intensive than ALS since ALS needs to store the data matrix twice. Nonetheless, choosing between ALS and SGD on a data set should be decided by evaluation of both algorithms on the relevant data set. In our framework, we have chosen to use an implementation of SGD since the Mahout implementation of ALS was mainly targeted towards distributed computing, while the Parallel SGD implementation was targeted towards a single multi-core machine[2]. Since we did not have a distributed evaluation setup, the SGD implementation performed much better.

[2]http://tdunning.blogspot.no/2009/01/real-time-decision-making-using-map.html

Chapter 6

Tensor Factorization

This chapter will introduce tensor factorization as a framework for generating recommendations. Section 6.1 explains the motivation behind tensor factorization, and how they work. Then we will describe how we can use tensors to do recommendations in Section 6.2 and at the end in Section 6.3 we will describe the most popular decompositions of tensors.

6.1 Introduction

As we mention in Section 3.2, we cannot directly express implicit feedback in a two-dimensional user-item matrix. When we add several kinds of feedback to each cell, we effectively have a tensor of three dimensions. These kinds of higher order tensors can also be factorized in order to estimate the missing values. By estimating not only explicit feedback, but all kinds of feedback, we might be able to achieve higher accuracy of recommendations.

Tensor factorization is a general form of matrix factorization. Whereas matrix factorization decomposes a matrix in two or more matrices, tensor factorization decomposes higher order matrices to several smaller matrices. A tensor factorization of a cubic matrix will result in at least three matrices, one for each axis.

Tensor factorization enables recommender systems to add a contextual element in a tensor, in [21] Thai-Nghe et al. describe a three-dimensional tensor $\mathbf{Z}$ of size $U \times I \times T$, where the first and second dimension describe the user and item while the last dimension describe the context, which in their case was time. In the following section we describe how to utilize the new dimensions as well as how to decompose the tensor.

6.2 Tensor Factorization in News Recommender Systems

Having more than two dimensions in recommender systems allows us to make use of contextual information. Examples of contextual information are time, seasonality and location. Thai-Nghe et al.[21] had promising results when they incorporated the time dimension in their predictor of student performance. In their predictor they utilize the time dimension to describe the progression students have on algebra in an online tutoring system. Hidashi and Tikk[22] efficiently segmented periodical user behavior in different time bands when they used seasonality as context. A simple example of seasonality is that horror movies are normally seen at night while animation is watched in the afternoon.

Time, seasonality and location are interesting contextual information in the news domain. Time and location are normally incorporated in a news recommender separately besides collaborative filtering, as done in the SmartMedia application. However, as studies shown in [21] and [22], adding time and seasonality as contexts in collaborative filtering improved their recommendations, making it interesting to use these contexts in the news domain as well.

Contextual information can also be feedback as a whole, where both explicit and implicit feedback is stored in the same vector, combining them to *feedback*. Since implicit feedback is dense, that is, as long as a user has read an article, implicit feedback will always be collected. This means that the feedback tensor will consist of much more information than what is common in matrix factorization scenarios. The increase of information will likely contribute to improved accuracy in recommendations. In matrix factorization, we estimate a rating by computing the dot product of the vectors in the factorized matrices. In tensor factorization, on the other hand, we compute the product of matrices, thus ending up with a vector, or a feedback tube. From this feedback tube, we could either use the estimated rating directly, or combine the feedback tube to a single rating by using a weighting scheme. A weighting scheme could be that explicit feedback should be weighted as two and implicit feedback as one, in such a way that feedback that the user give the recommender system more important than what the recommender system collects.

Tensor factorization is a promising framework for computing recommendations, however, its adoption in the recommender systems industry is still limited. We look at some related work in Section 8.1.2. Due to a lack of available recommendation libraries based on tensor factorization, we decided to not pursue it further in this thesis.

6.3 Tensor Decompositions

There are a number of different tensor decompositions, but the two most popular tensor decompositions are Tucker and CANDECOMP/PARAFAC (CP). A comprehensive review of other tensor decompositions can be found in [23].

6.3.1 Tucker Decomposition

The Tucker decomposition is a form of higher-order principal component analysis. It decomposes a tensor into a core tensor multiplied by a matrix along each dimension[23]. Figure 6.1 presents the decomposition in the case of a three-dimensional tensor. The equation for the three-dimensional tensor in Figure 6.1 where $\mathbf{X} \in \mathbb{R}^{I \times J \times K}$ is formulated in Equation 6.1

$$\mathbf{X} \approx \mathbf{G} \times_1 \mathbf{A} \times_2 \mathbf{B} \times_3 \mathbf{C} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} \mathbf{g}_{pqr} \mathbf{a}_p \circ \mathbf{b}_q \circ \mathbf{c}_r \tag{6.1}$$

where $\mathbf{X}$ is the tensor to be decomposed, $\mathbf{G}$ is the core tensor and $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ are respectively the matrices for each dimension.

6.3.2 CANDECOMP/PARAFAC

CANDECOMP/PARAFAC (CP) decomposition factorizes a tensor into a sum of component rank-one tensors[23]. For a three-dimensional tensor $\mathbf{X} \in \mathbb{R}^{I \times J \times K}$, the decomposition results to a sum of vectors for each dimension as shown in Figure 6.2 and is written as

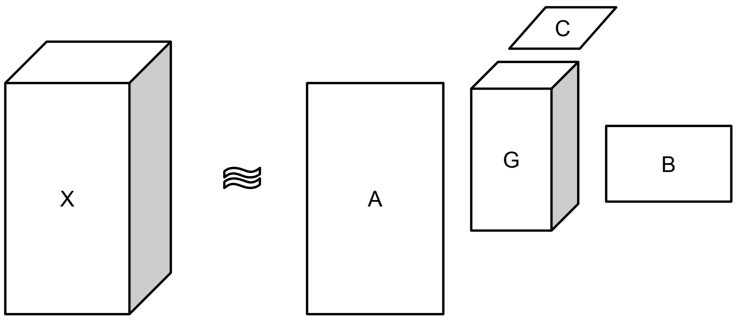


FIGURE 6.1: Tucker decomposition of a three-dimensional tensor

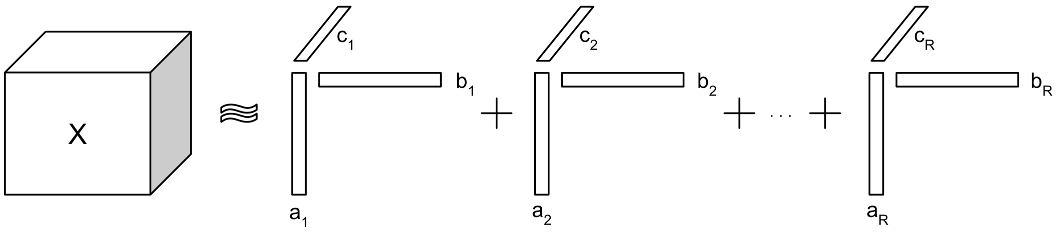


FIGURE 6.2: CANDECOMP/PARAFAC decomposition of a three-dimensional tensor

$$\mathbf{X} \approx \sum_{r=1}^{R} \mathbf{a}_r \circ \mathbf{b}_r \circ \mathbf{c}_r \tag{6.2}$$

where $\circ$ is the outer product and each vector $\mathbf{a}_r \in \mathbb{R}^I$, $\mathbf{b}_r \in \mathbb{R}^J$, and $\mathbf{c}_r \in \mathbb{R}^K$.

Chapter 7

Mapping Techniques

In this chapter we introduce mapping techniques used to generate explicit ratings from implicit feedback from a user. Section 7.1 introduces these mapping techniques and their rationale. In Section 7.2 we introduce the correlation coefficient, which is used to measure the relationship between to variables, and in the following sections we describe different mapping techniques.

7.1 Introduction

In Section 3.2 we introduced the notion of mapping implicit ratings to explicit ratings in order to improve recommendation accuracy. Many of the current recommendation systems depend on explicit ratings from users in order to compute recommendations. Users are, however, often reluctant to provide such information [10]. Instead, we can look at how users interact with a system. This kind of information is called implicit feedback, and we look further into the various kinds of implicit feedback available in Section 4.2. The question is, how can we use this kind of implicit feedback? In Chapter 6 we looked at tensor factorization as a way of utilizing implicit feedback. Using tensor factorization, we need to postprocess the implicit feedback in some way in order to properly order items. A different way to use implicit feedback works by preprocessing the data set prior to building a recommendation model. With this preprocessing step, we can fill in missing explicit values based on the patterns found between instances where users have provided such data, and the corresponding implicit feedback. When we know that the implicit feedback directly corresponds to expressed explicit feedback, we can be confident that the users have preferred items with the respective feedback. Consequently, we can use these relations to estimate how much a user preferred an item based on how he or she interacted with it. In this chapter we look at different methods to achieve this mapping from implicit to explicit feedback.

7.2 Correlation Coefficient

The correlation coefficient measures the strength and the direction of a linear relationship between two variables[1]. The coefficient, p, have values between -1 and 1, whereas -1 represents perfect negative fit and 1 perfect postive fit. When p is positive, the relationship between variables x and y is that when x increases y increases and when p is negative y decreases when x increases. The mathematical formula for computing p is

$$p = \frac{n\sum xy - (\sum x)(\sum y)}{\sqrt{n(\sum x^2) - (\sum x)^2}\sqrt{n(\sum y^2) - (\sum y)^2} \tag{7.1}$$

where n is the number of pairs of data. Examples of different correlations coefficients are presented in Figure 7.1.

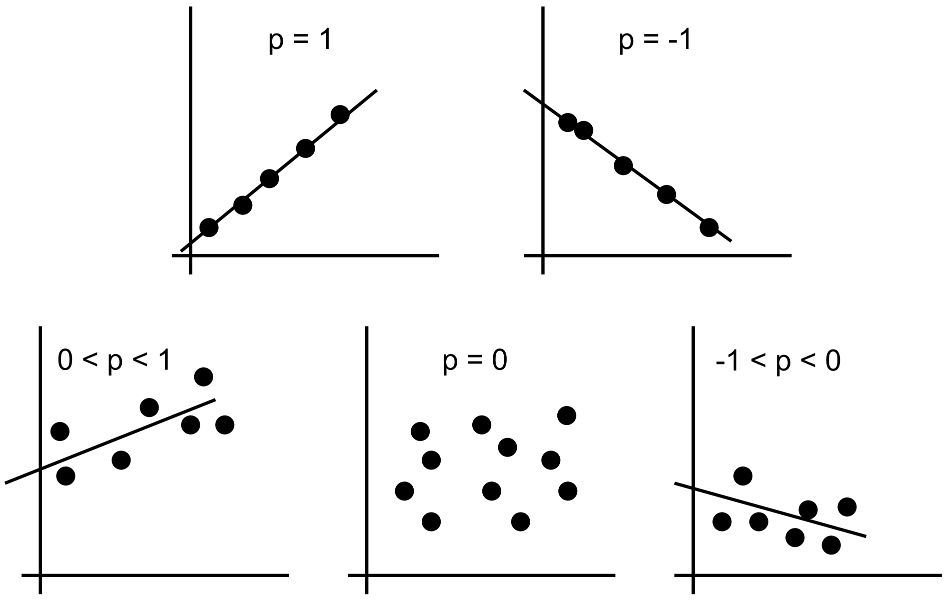


FIGURE 7.1: Examples of different values for the correlation coefficient

[1]http://www.stat.yale.edu/Courses/1997-98/101/correl.htm

The correlation coefficient itself does not map implicit feedback to explicit feedback, but it can be used to find implicit-explicit ratings pairs that correlate and hence can be used to map implicit to explicit ratings through other techniques such as nearest neighbor, rating bins or linear regression.

7.3 Bins

Bins is a simple way of mapping two variables. After an analysis that confirms that a mapping exists (e.g. correlation coefficient above or below threshold), we can create a number of bins. In a recommender system where explicit ratings are given by a value between one and five, we create five bins with different ranges according to the implicit ratings.

If we observe the following explicit-implicit rating pairs: (1 - 20 000), (3 - 45 000), (4 - 50 000) and (5 - 65 000), we clearly see that higher implicit value maps to higher explicit value. In this example, the explicit value is a rating between one and 5, and implicit rating is the time in milliseconds the user has spent reading an article. With equal size of each bin, the equation used to find the size of each bin is:

$$\frac{max - min}{n} \tag{7.2}$$

where max and min is the highest and lowest implicit value accordingly and n the number of different ratings. In our example, the size of each bin is $(70000 - 20000)/5 = 10000$, and the range of each bin will look like the bins presented in Figure 7.2. A new rating without explicit, but with implicit rating of 55 000 will then be given four as explicit rating.

7.4 Linear Regression

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data[2]. The two variables are called independent and dependent variables, where the independent variable in recommender systems

[2]http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm

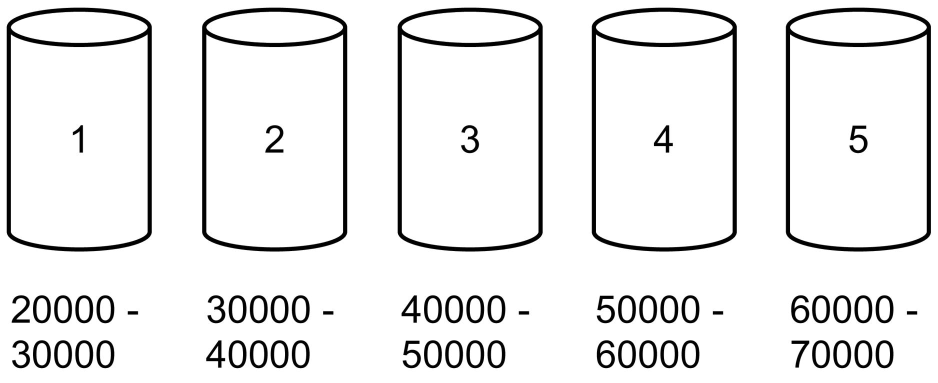


Figure 7.2: Example of bins with equal size

refers to implicit feedback and the dependent variable refers to explicit feedback. A linear regression line has an equation of the form

$$Y = a + bX \tag{7.3}$$

where X is the independent variable, Y the dependent variable, a is the intercept and b the slope of the line.

In the news domain, the dependent variable Y is the rating while the independent variable can be the time a user spends reading the article. When Equation 7.3 has been fitted by observed data the equation can look like Equation 7.4. If we then want to map another article which only has a TimeOnPage as 55 000, we get a rating of $1.5 + 0.00005 \times 55000 = 4.25 \approx 4$.

$$Rating = 1.5 + 0.00005 \cdot TimeOnPage \tag{7.4}$$

When linear regression is used, an analysis that determines if there exists a relationship between the two variables should be done. As discussed in Section 7.2, the correlation coefficient is a numerical measure of association between the variables, and can be used to determine if a relationship exists and ensure that it make sense to use linear regression.

7.4.1 Multiple Linear Regression

Multiple linear regression attempts to model the relationship between two or more independent variables and a dependent variable by fitting a linear equation to observed data[3]. When we use multiple linear regression, the formula is

$$Y = \beta_0 + \beta_1 X_1 + \beta_1 X_1 + ... + \beta_n X_n \tag{7.5}$$

where n is the number of independent variables and β describes how much each variable affects the regression line.

With multiple linear regression, we can expand Equation 7.4 with other implicit feedback such as the time a user moves the mouse when reading an article. With two such independent variables, the equation becomes:

$$\mathit{Rating} = 1.5 + 0.00005 \cdot TimeOnPage + 0.0008 \cdot TimeOnMouse \tag{7.6}$$

Subsequently, an article that a user has spent 55 000 ms reading and 800 ms moving the mouse, will then get a rating of $1.5 + 0.00005 \times 55000 + 0.0008 \times 800 = 4.89 \approx 5$.

7.5 Classifiers

The mapping from implicit to explicit feedback can also be seen as a classification problem, where we want to find or classify an explicit rating based on a known set of mappings or instances. When we want to classify a rating from 1-5, we are able to use not only numeric predictors like regression models, but also classifiers normally working on nominal class values. A lot of research has been done on the task of classification, and a number of standard classification methods exist, where each has its own strengths and weaknesses. In this section we look at a set of classifiers with the feedback mapping in mind.

7.5.1 Naive Bayes Classification

Naive Bayes is a probabilistic classifier named after Bayes rule[4], which assigns a class to an instance based on the highest estimated probability [24]. Given a fixed set of classes, for example rating 1-5, the classifier is trained by calculating the

[3]http://www.stat.yale.edu/Courses/1997-98/101/linmult.htm
[4]http://en.wikipedia.org/wiki/Bayes'_theorem

global probability for each class. The Naive Bayes classifier is so called because it assumes that the features in the data are independent of each other. It has, however, shown to perform well in many different environments[25]. When calculating the likelihood of each class, apply the naive feature assumption to Bayes theorem given in equation 7.7. When assuming independence, we get Equation 7.8, often called the maximum a posteriori probability (MAP) where B denotes an implicit feedback variable such as time spent reading an article, and A denotes a rating. To classify, we then apply the argmax function, yielding the most probable classification.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{7.7}$$

$$P(A|B) \propto P(B) \prod_{1 \leq k \leq n_d} P(t_k|B) \tag{7.8}$$

7.5.2 K-Nearest Neighbor

K-Nearest Neighbor (KNN)[26] is a very simple classification method, where we do not train a model, but simply assign the majority class of the K nearest neighboring instances. It is important to select an appropriate number of neighbors and which distance measure to use. When data sets become large, performance issues can occur. These can be mitigated to a degree by computing the similarity matrix prior to classification. There are many different distance measures that can be used, depending on the data set. Euclidean distance or cosine similarity are examples of common distance functions that can be applied to items with numeric data. Given vectors x_a and x_b, the euclidean distance between these two vectors is given in Equation 7.9.

$$\text{distance}(x_a, x_b) = \sqrt{\sum_{t=1}^{n} (x_{a,t} - x_{b,t})^2} \tag{7.9}$$

7.5.3 Artificial Neural Network

An artificial neural network (ANN)[27] is a computational model devised from the human brain, and is often used to solve machine learning problems, including that of classification. The artificial neural network is a network of neurons that are able to learn appropriate mapping models based on a training set. A neuron is simply a mathematical function, usually modeled after biological neurons. An example of a simple neuron is a perceptron. It is a binary classifier which yields 1 or 0 based on a linear prediction function, as given in Equation 7.10.

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{7.10}$$

When we are dealing with multi-label classification however, we need a different kind of neural network. A network multilayer perceptron (MLP) has been shown to be a very versatile network, and has been applied to many different problems [28, 29, 30]. It is a modification of the standard linear perceptron as it uses three or more layers of neurons with nonlinear activation functions, and is able to distinguish data that is not linearly separable, or separable by a hyper-plane. An example of a MLP with two hidden layers is shown in Figure 7.3.

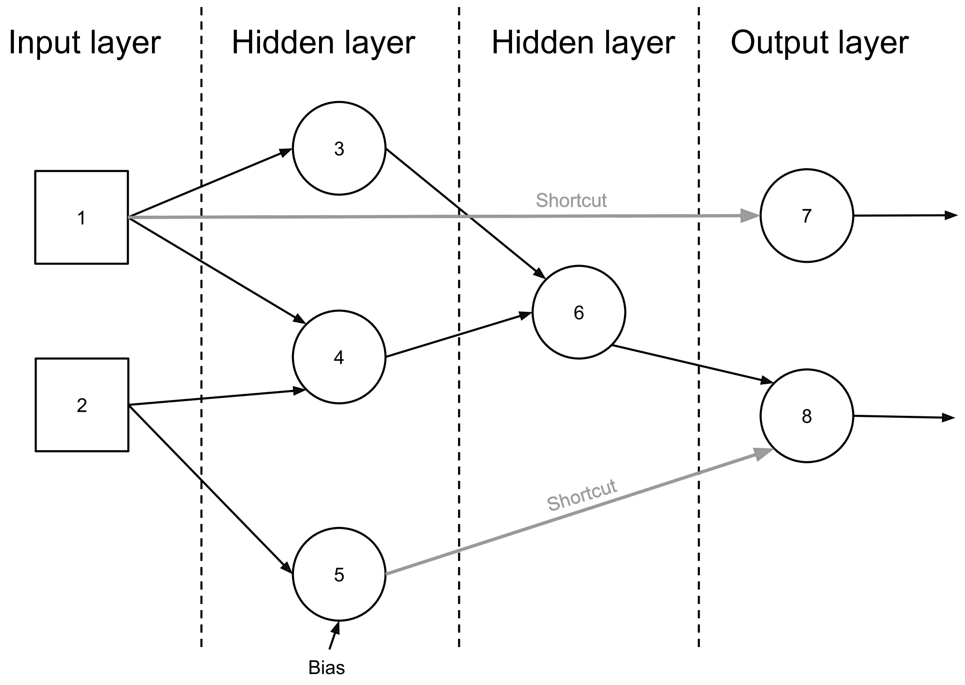


Figure 7.3: A multilayer perceptron with two hidden layers

When using MLP it is beneficial to experiment with different values of learning rate. The learning rate is how much weight it gives for new examples. There is a trade off between performance and accuracy. If we use a low learning rate with a high iteration number the learning process will be more conservative, but if we use a low iteration number and high learning rate, the accuracy might suffer.

7.5.4 Support Vector Machine

A support vector machine (SVM), proposed by Cortes and Vapnik in 1995[31], determines the classification of an instance by finding a high-dimensional hyperplane selected to separate the classes with the largest margin. With margin, we mean the distance between the hyperplane and the closest training data point of any class. The data we want to classify is then projected onto the high-dimensional space, where they can be separated by the selected hyperplane. Thus, a bigger margin means that we have a higher probability of correctly classifying new instances. An example of a binary hyperplane is shown in Figure 7.4. In this example, we see that the margin a is larger than margin b, and thus the hyperplane of *a* is better for this problem.

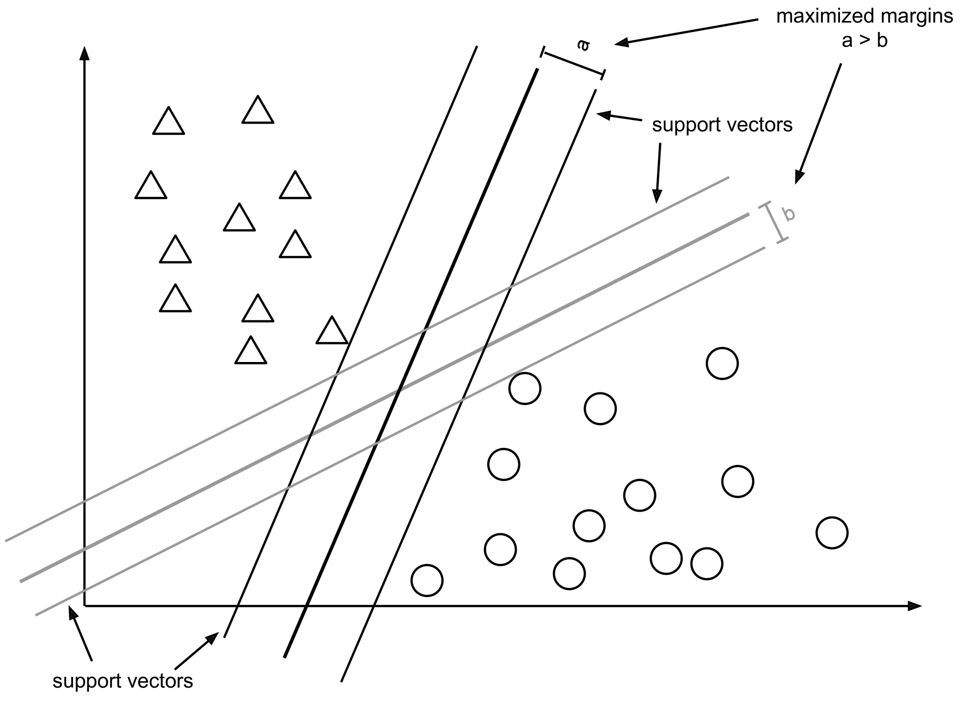


FIGURE 7.4: Example of a SVM hyperplane

7.6 Classification via Clustering

Clustering algorithms group a set of items into subsets or clusters, and the goal of clustering is to create clusters that are coherent internally, but different from each other. After a set of clusters is made, new instances are then evaluated to find which clusters they belong to. Using this scheme, we can classify items via clustering by first training the model and create a fixed number of clusters. Then, when new items are added, we find out which cluster they belong to and add them to that cluster, i.e. if we have one cluster for each rating, we can guess how the user will rate an item based on which cluster it belongs to.

The choice of clustering algorithm decides the properties of the clusters, and will be further elaborated below.

7.6.1 K-Means

K-Means, coined by MacQueen in 1967[32], clusters a set of items based on K clusters. It iterates through two steps: assign all items to nearest cluster and then update the mean centres, until a finish criterion has been met. Different finish criteria are: the movements of means are less than a threshold, a fixed number of iterations or a combination of these.

```
Data: Set of items I and number of clusters K
Result: K cluster means
Initialize means (pick K items random);
while Finish criterion do
    Assign each item to nearest mean;
    Move mean to center of its cluster;
end
```

Algorithm 3: Pseudo code for K-Means algorithm

7.6.2 X-Means

X-Means is an improvement over K-Means, and overcomes the following shortcomings of K-Means: scalability and runtime; the number of clusters must be supplied by the user; and finds local optima based on the number of clusters supplied. In X-Means, the number of clusters is found given a minimal and maximal number of clusters given by the users, explaining why it is called ”X” means. This will in turn give empirical better optimums than using only K clusters. The scalability and runtime performance is also improved. Pseudo code of X-Means is shown in Algorithm 4

```
Data: Set of items I, min number of clusters $K_{min}$ and max number of clusters
      $K_{max}$
Result: K cluster means
while $K > K_{max}$ do
    Improve-Params;
    Improve-Structure;
end
```

Algorithm 4: Pseudo code for X-Means algorithm

where *Improve − Params* consists of running conventional K-means to convergence and *Improve−Structure* finds out if and where new centroids should appear. More details on the algorithm can be found in [33].

7.6.3 Expectation-Maximization

Expectation-Maximization clustering is a generalization of K-Means which assigns a probability distribution to each instance which indicates the probability of it belonging to each of the clusters[34].

```
Data: Set of items I and number of clusters K
Result: K cluster means
while Finish criterion do
    Calculate cluster probabilities;
    Calculate distribution parameters;
end
```

Algorithm 5: Pseudo code for Expectation-Maximization algorithm

7.6.4 Conceptual Clustering

Conceptual clustering takes as input a set of objects descriptions and produces a classification scheme over the observations [35]. As opposed to K-means, X-means and EM, conceptual clustering does not have a preset number of clusters, but finds the appropriate number of clusters by learning from the observations.

In [35], Douglas Fischer presents COBWEB, which is an incremental system for hierarchical conceptual clustering. COBWEB carries out a hill-climbing search through a space of hierarchical classification schemes. Its incremental feature is motivated by real world usage, where knowledge may be rapidly updated with new observations. This observations fits well with the news domain, which is constantly changing and as news change in both content and presentation, COBWEB could incrementally adjust the mapping from user interaction to ratings. However, it is only useful if COBWEB discovers five clusters from the feedback, one for each rating. Following the example in [35] a hierarchical decision tree from Table 7.1 can be represented as Figure 7.5.

Table 7.1: Animal descriptions

Name	**Body cover**	**Heart champer**	**Body temp.**	**Fertillization**
mammal	hair	four	regulated	internal
bird	feathers	four	regulated	internal
reptile	cornified-skin	imperfected-four	unregulated	internal
amphibian	moist-skin	three	unregulated	external
fish	scales	two	unregulated	external

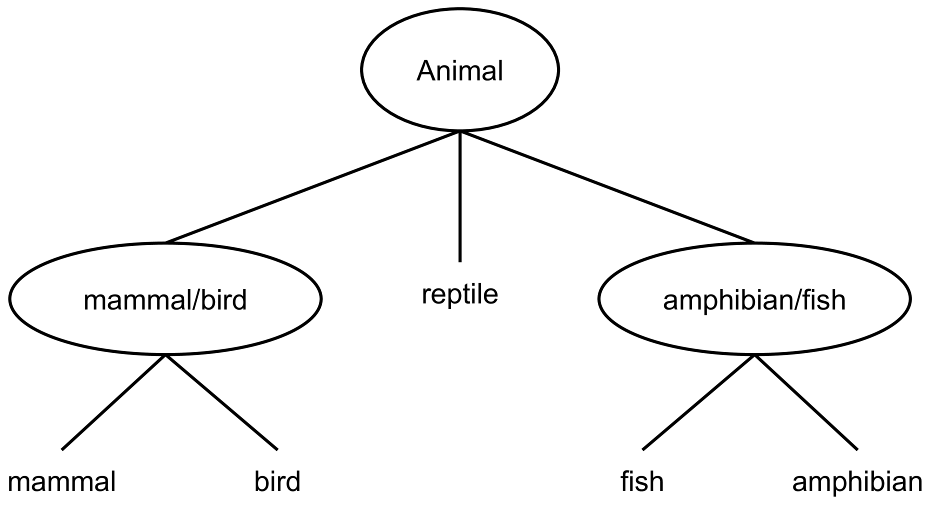


Figure 7.5: A hierarchical clustering over animal descriptions

Chapter 8

Related Work

In this chapter we present earlier work on recommender systems. In Section 8.1 we introduce how matrix factorization became popular during the Netflix Prize competition and describe how implicit feedback can be used in tensor factorization. In Section 8.2 we present how implicit feedback can be used in recommender systems.

8.1 Matrix and Tensor Factorization

8.1.1 Netflix Prize

Netflix is one of the biggest video streaming media companies in the world, and through their services they provide recommendations to their users about which movies or TV-shows they might like. In order to improve their own recommendations, they released a data set consisting of 100 millions ratings from their website and created the Netflix Prize. Netflix Prize was a competition where they challenged the research community to come up with a recommendation algorithm that performed 10% better than their own implementation, Cinewatch, which had a RMSE of 0.9525. It started in October 2, 2006 and ended when the Grand Prize was received by the winning team "BellKor's Pragmatic Chaos" on September 21, 2009. The winning team had then improved the RMSE with 10.06% compared to Cinewatch, decreasing it to 0.8556.

The competition led to much research on collaborative filtering techniques, and it demonstrated that matrix factorization models are superior to classic nearest-neighbor techniques for producing recommendations. The main advantages of matrix factorization compared to nearest-neighbor techniques are that matrix factorization allows the incorporation of additional information such as temporal effects and confidence levels[16]. In addition, the winning team highlights baseline predictors as one of their biggest contributions. Their winning solutions used several baseline predictors combined with a factorized neighborhood model, see [7] for details about their algorithm.

8.1.2 Tensor Factorization

As shown in Chapter 6 tensor factorization can be used in collaborative filtering to add contextual information to the model. Karatzogluo et al. defines this approach as multiverse recommendation, and in [36] they show that their tensor factorization algorithm performs better than normal matrix factorization on data sets with 2 or 5 contextual dimensions, including season and day of the week for a movie data set.

Others include time as a third dimension. In [21] Thai-Nghe et al. predicts student performance on online courses where time is a crucial factor. By including time the model can handle the progress of the students in a better way, which is shown by improvement of recommender compared to matrix factorization in experimental results. Hidasi et al. includes seasonality and sequentiality as contextual information, which improved the performance of the recommender system significantly [22].

8.2 Implicit Feedback in Recommender Systems

8.2.1 Mapping Implicit Feedback to Explicit Ratings With Last.fm Data Set

In their paper [13], Parra and Amatriain analyze the relation between implicit and explicit feedback on a data set collected from Last.fm. The data set was collected by conducting an online user study of users of the last.fm music service. The implicit feedback such as user listening history was obtained by crawling the users last.fm page, while the explicit feedback was obtained by asking the users to rate albums on a one to five star scale.

The main variables analyzed to find the influence between implicit and explicit feedback was: (i) playcount for a user on a given item (Implicit Feedback - IF); (ii) global playcount for all users on a given item (Global Popularity - GP); and (iii) time elapsed since user played a given item (Recentness - R). Of these three variables, they observed that the amount of implicit feedback and recentness were the most influential variables.

Using a regression analysis on the results of their implicit feedback analysis, they incorporated the implicit feedback in four different models, dependent on how many variables they include in the prediction of ratings, from using only implicit feedback to using IF, GP, R and a combination of IF and R. The results were calculated by using RMSE and their improvement over the user average using only explicit feedback was 6.5%.

8.2.2 Recommending With Implicit Feedback Only

Hu et al. presents in [12] a method for doing collaborative filtering when only implicit feedback is available. They have compensated the loss of having explicit ratings with a model that incorporates two new variables: (i) p_{ui}, which indicate the binary preference of user u to item i, that is, if the user u has consumed item i; and (ii) c_{ui} which measure the confidence in observing p_{ui}. With these two variables they are able to better reflect the nature of the data. Matrix factorization is then used to calculate the preference values for items they have not yet consumed, with the cost function:

$$\min_{x_*,y_*} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2) \tag{8.1}$$

The data set used in their analysis was based on data from a digital television service. During a four week period, the collected data was the number of times each user watch a tv program (which is the number of minutes that a given show was watched). Based on their analysis, this model performed better than a neighborhood based version of their model as well as just recommending the most popular programs.

8.2.3 Implicit Feedback Used in Text Retrieval

Morita and Shinoda discovered already in 1994 that there exists a correlation between time spent reading a news article and the explicit rating given by the user. In [14] they conducted a study about information filtering systems, where they looked at three important issues: (i) how to collect user preferences; (ii) how to represent the user profile; and (iii) how to do the filtering.

In their study, they collected user preferences by recording the time spent per article, the length of the article and the readability of the article. The feedback was collected by finding eight volunteers who would rate articles on a scale from A to F, where A is very interesting and F is not interesting. The news articles were read on an internet information service called NetNews.

The main findings in their paper were that there is a strong tendency to spend a long time to read interesting articles, a tendency not to spend a long time on uninteresting articles, very low correlation between length of an article and time spent reading it and that other factors such as readability and number of unread articles have low effect on reading time. More specifically, their result indicated that treating articles as interesting when the user has spent 20 seconds of reading them yields 30% recall and 70% precision.

Using these results, they represent users by storing all articles they have read in a database, and filter new articles by using a sub-string-indexing method. It works by deciding if a new article is interesting or not is done by counting occurrences of all sub-strings of the article in a sub-string database containing sub-strings made by splitting articles determined to be interesting in the past.

If they only stored articles that the users had spent more than 20 seconds reading, the filtering method was further improved, encouraging the use of time spent reading feedback to improve an information filtering system.

8.2.4 Google News

Google news is an online news recommender system initiated by Google Inc., that aggregates news from more than 4 500 news sources worldwide. As an online news recommender, it has many of the same challenges as the SmartMedia application, such as scalability and item churn. Google News solved these challenges by dividing the recommender system into three parts[6]: (i) an offline component which utilizes implicit feedback by clustering users based on their click history, (ii) a set of online servers which updates user and story statistics as well as generating news recommendation on requests and (iii) two types of data tables: a user table storing user click history and clustering information, and a story table storing real time click counts for every story-story and story-cluster pair.

The offline component scales with millions of users and news stories. This is done by modeling it as a distributed computation when clustering the users. The clustering is based on clicks made by users over a short time window and the clustering algorithms used are either MinHash or PLSI.

The data tables are used to store all relevant user and story statistics that are used by the real time servers. To emphasize recent news stories and avoid giving old news stories too high click count, Google News uses time decayed counts[6], giving more weight to recent user clicks.

Live evaluation of their recommender system shows that it performs better than recommending the most popular news articles. This indicates that implicit feedback, and in Googles case user click history, is enough to create good recommendations. They also ran the clustering algorithms on the Movielens and a NewsSmall data set, where they observed that the clustering algorithms performed better than a memory-based item-covisitation algorithm.

Part III

Realization

Chapter 9

Implementation

This chapter is divided into three sections. In Section 9.1 we present the architecture made to support recommendation with implicit data. In Section 9.2 we present the framework used to evaluate hundreds of different recommendation models, and in Section 9.3 we show how our recommendation component was integrated with the SmartMedia news application.

9.1 Recommender

In our work, we have made use of the Apache Mahout Library[1]. Mahout is a machine-learning library implemented in Java. It contains a number of implementations of collaborative filtering methods as well as fast data structures for storing explicit feedback. It also contains methods to calculate a number of accuracy metrics for a recommendation model. Mahout does not, however, support the utilization of implicit feedback, and thus we have extended Mahout with a custom data model. As we discuss in Chapter 6, when additional kinds of feedback are available, the sparse user-item matrix can be generalized into a sparse three-dimensional tensor, often called a cube. In Mahout's data model, each cell in the user-item matrix is addressed in terms of the respective userID and itemID. In a cube of feedback, we also need to address cells by their feedbackID. In order to support fast look-up of both user rows, item columns and tubes of implicit feedback, we utilize an in-memory SQL Database called H2[2]. By adding each cell

[1]https://mahout.apache.org
[2]http://www.h2database.com

to the in-memory database, and setting indices on each coordinate property, we effectively have a fast, sparse, 3-dimensional data structure. The structure of the events table is given in Listing 9.1.

```
CREATE TABLE IF NOT EXISTS events (
    userID BIGINT NOT NULL,
    itemID BIGINT NOT NULL,
    feedbackID BIGINT NOT NULL,
    value REAL NOT NULL,
    PRIMARY KEY (userID,itemID,feedbackID)
);

CREATE INDEX uIndex ON events (userID);
CREATE INDEX iIndex ON events (itemID);
CREATE INDEX fIndex ON events (feedbackID);
```

Listing 9.1: SQL code for events table

The implicit data is then used in a preprocessing step, where we look for items where explicit feedback is not given, and try to infer it from patterns found in the data set. We elaborate on how these patterns are inferred in Chapter 7. After preprocessing, we extract the resulting user-item matrix of explicit feedback. The processed user-item matrix can then be used to build a recommendation model. This is done by factorizing it using Parallel Stochastic Gradient Descent, which is a part of Apache Mahout. After this stage, we can use the model to provide recommendations to a user.

```
Data: implicit and explicit feedback F
Result: recommendation model
if F has been updated then
    load F from DB into memory;
    preprocess(F);
end
model = recommender.buildModel(F);
return model
```

Algorithm 6: Pseudo code for recommender

9.2 Evaluation of Mapping Techniques

To solve our research questions, we needed to evaluate and compare many different mapping algorithms with changing parameters on different data sets. Thus we decided to develop a standardized way of computing these evaluations. We needed to conduct each evaluation on a certain data set using a certain mapping technique and a certain collaborative filtering method. We also needed to test each mapping technique with up to hundreds of different parameter combinations. Thus, we created a Configuration class to contain all these variations. When bootstrapping the evaluation, we create a Configuration object to contain all the parameters, and add it to a list. The list is passed to an evaluator, which then conducts the evaluations using the parameters contained in the Configuration object. Upon completion, the evaluator returns a list of result metrics – one for each evaluation. This list of results can then be serialized to a file and further analyzed. An example code showing the evaluation pipeline is given in Algorithm 7.

```
Data: dataModel, configurations
Result: results
foreach config in configurations do
    Preprocessor = config.get(”preprocessor”);
    newModel = Preprocessor.preprocess(dataModel, config);
    result = Evaluator.evaluate(newModel, config);           // evaluate config
    results.add(result);                                // add to list of results
end
return results
```

Algorithm 7: Pseudo code for evaluation pipeline

9.3 Integration to SmartMedia

An important part of our work was to integrate the collaborative filtering component to the SmartMedia news application in order to recommend news articles to users. The CF component runs independently of the news application, and acts as a service that provides recommendations. Both systems have access to the same database, and will communicate via simple HTTP REST APIs. As shown in Figure 9.1, there are two distinct forms of communication that will happen; (i) A notification about updates to the database, and (ii), a request for recommendations to a client. As an example of (i), Amy reads a news article, which is reported to the news API. The news API then stores an event in the database containing implicit feedback collected as Amy was reading, as well as an explicit rating if she gave one. Then, the CF API is notified about the new information, and it can fetch it from the database and rebuild the recommendation model. In example (ii), Mark requests articles personally recommended to him from the news API. The news API forwards the request to the CF API which returns a set of recommended items. Since all requests to the CF component will be from the news server, we can run it behind a firewall for enhanced security. This also alleviates the need for dealing with user authentication in the CF component. A detailed specification of the CF API is elaborated next.

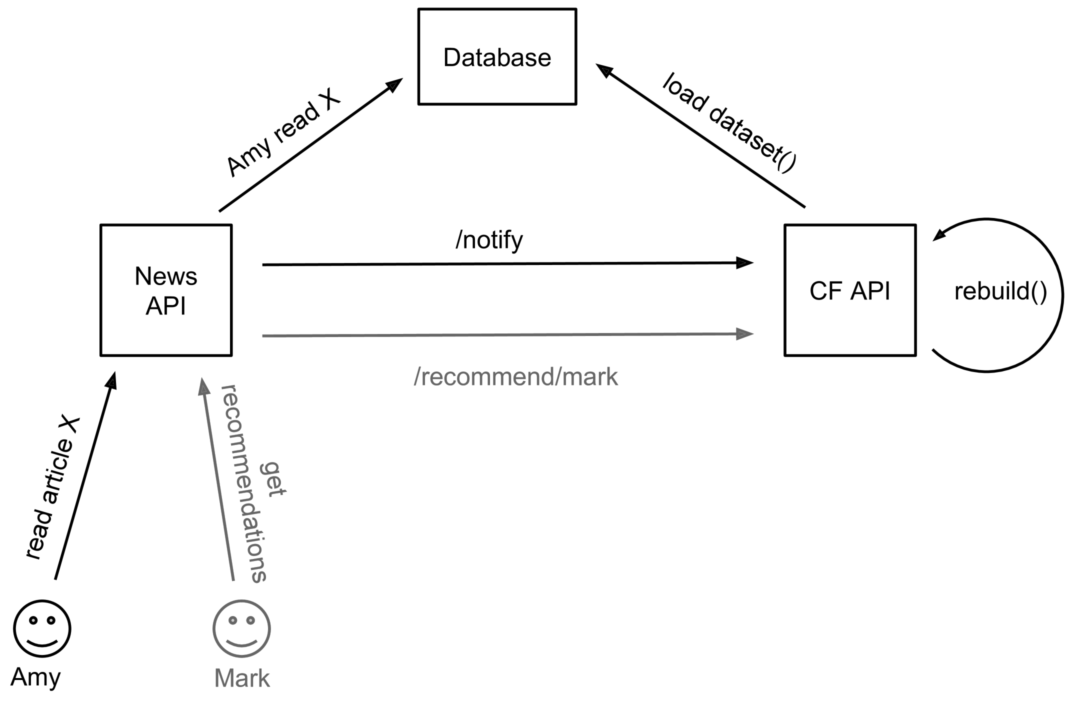


FIGURE 9.1: The SmartMedia recommender integration

1. Notifications

When users read articles, the data set changes. We need to know about these updates so we can rebuild the recommendation model when necessary. The current implementation stores data in a MongoDB[3] database. As of this writing, MongoDB does not support collection subscriptions which means that the CF component either needs to poll the database for updates, or that the SmartMedia Application has to notify the CF component when have been made. Polling can significantly lower the performance of the database, which can affect the performance of the system as a whole. Therefore we opt for the latter option in our implementation.

When the news application makes a change to the database, an HTTP request is made to the CF component at /notify. The result is a JSON object containing the following properties:

[3]http://www.mongodb.com

- **rebuild**: true/false – true if the recommendation model was rebuild after this notification
- **lastrebuild**: the UNIX timestamp of the last rebuild
- **changes**: the number of changes since the last rebuild

Example:

- request: POST https://cf.smartmedia.idi.ntnu.no/notify
- response: {rebuild: true, lastrebuild: 1399985213, updates: 151}

2. Recommendations

When a user wants to read recommended articles on his device, a request is sent to the news server. The news server forwards the request to the CF component at /recommend/{userid}?limit={limit}, where the parameters are described as

- **userid**: the ID of the user to provide recommendations for
- **limit** [optional]: the maximum amount of recommendations to provide (can be lower)

The response is represented as JavaScript Object Notation (JSON), containing an object with a single property ”data”, which is an array of objects with the following properties:

- **itemid**: the item identifier
- **value**: the estimated rating of the respective item

Example:

- request: GET https://cf.smartmedia.idi.ntnu.no/recommend/412?limit=10
- response: {data: [{itemid: 306860, rating: 4.25}, {itemid: 304486, rating: 4.22}, . . .]}

Exceptions

If an error occurs during recommendation or while computing the model, the API yields a proper status code according to the HTTP/1.1 specification[4] and a JSON object is returned with the type of error and an error message.

Example:

- request: POST https://cf.smartmedia.idi.ntnu.no/recommend/asdf
- response: {error: ”NoSuchUserException”, message: ”The user asdf does not exist”}

[4]http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Chapter 10

News Recommendation in the SmartMedia Application

In this chapter we describe the process of creating recommendations in the SmartMedia application. In Section 10.1 we outline how the feedback is represented, in Section 10.2 we show how it is preprocessed and in Section 10.3 we show how to generate recommendations.

10.1 Feedback Representation

The SmartMedia application currently collects both explicit and implicit feedback, although in the future we plan to collect only implicit feedback. The explicit feedback is represented as a five star rating given by the users after they have finished reading an article, while implicit feedback is collected by the application itself and consists of the time a user has spent reading the article in milliseconds.

However, other user interactions can easily be recorded and utilized, such as time a user has spent moving the mouse or number of mouse clicks inside the article, but these are yet to be implemented in the SmartMedia application.

The recommender system uses explicit ratings as the users true interest in the article, and implicit feedback as indication of interest.

Table 10.1: Feedback used in SmartMedia application

Explicit feedback	Implicit feedback
Five star rating	TimeOnPage in milliseconds

10.2 Preprocess the Feedback

Preprocessing the feedback in the SmartMedia application is the main focus of this thesis. The preprocessing ensures that we utilize implicit feedback by mapping it to explicit feedback which the SVD recommender can understand.

Since the feedback consists of ratings and time spent per article, we have chosen to limit the techniques to utilize the implicit feedback to the mapping techniques described in Chapter 7, influenced by the work of Parra and Amatriain [13], and Morita and Shinoda [14]. The mapping techniques are evaluated in the next chapter on a realistic news data set, enabling us to find a good mapping technique to be used in the news domain.

As Figure 11.2 shows, the mapping technique that had the lowest RMSE was a preprocessor that combines closest neighbor search and time spent per article analysis to generate recommendations. Table 11.7 presents the different configurations of this technique, and Table 11.17 presents the respective results. The top configuration got an RMSE of 0.978, which is a 5.14% improvement over using only explicit feedback. Properties of this configuration are presented in Table 10.2, which are the ones that will be used in the SmartMedia collaborative filtering component.

Table 10.2: PreprocessorStat class properties

Prediction method	Time threshold	Correlation limit
Closest Neighbor	25 seconds	0.3

Figure 10.1 presents the flowchart of the preprocessor algorithm. The start and end symbols define the scope of the preprocessor and as long as there exists unrated articles in the data set which is not preprocessed, it loops over the data set. The first condition ensures that it is possible to find a correlation between ratings for each article. If it is possible, and the correlation is above 30% as defined by the correlation limit, the algorithm will generate a pseudo rating for the unrated article by doing a nearest neighbor search with one neighbor. If the article has less than 3 ratings or the ratings has correlation less than 30%, the algorithm will generate a

pseudo rating of four if the time the user has spent reading the article is above 25 seconds. As seen in Table 11.17, the flowchart is robust, and minor variations in the correlation threshold or time threshold does not incur any significant changes to the RMSE, as long as we use a closest neighbor prediction method.

The preprocessing step is done offline at regular intervals to keep the model updated and ensure that users are recommended relevant articles. In the SmartMedia application, the model is updated every 10 minutes if the data set has changed. This frequency is based on results from Chapter 12, which demonstrated that our recommender is able to build the model within five minutes on a synthesized data set with one million feedback entries. We use an extra buffer of five minutes in case of unexpected performance strains. The results are presented in Table 12.3, demonstrating that ten minutes is sufficient to update the model and replace it with the old model.

After preprocessing the feedback, the user-article rating matrix consists of both original ratings and pseudo ratings. Properties of the user-article matrix when the preprocessing is used on the YOW data set is presented in Table 10.3. The preprocessed feedback is then used to create recommendations, which is described in the next section.

Table 10.3: User-article matrix properties

Feedback	**Ratings**	**Density**
Original	6653	5.0%
Preprocessed	7686	5.8%

10.3 Create Recommendations

After the preprocessing step is completed, we have a denser user-article matrix, from which we can construct a recommendation model. This is done by factorizing the matrix using a parallel SVD factorizer using Stochastic Gradient Descent. This factorization yields a model of two matrices of size $u \times l$ and $i \times l$, where u denotes the number of users, i denotes the number of articles and l denotes the number of latent factors selected for this factorization.

As we discuss in Section 5.1, the latent factors represents hidden patterns found in the data. This can for example be the degree of sports in the news article, or the

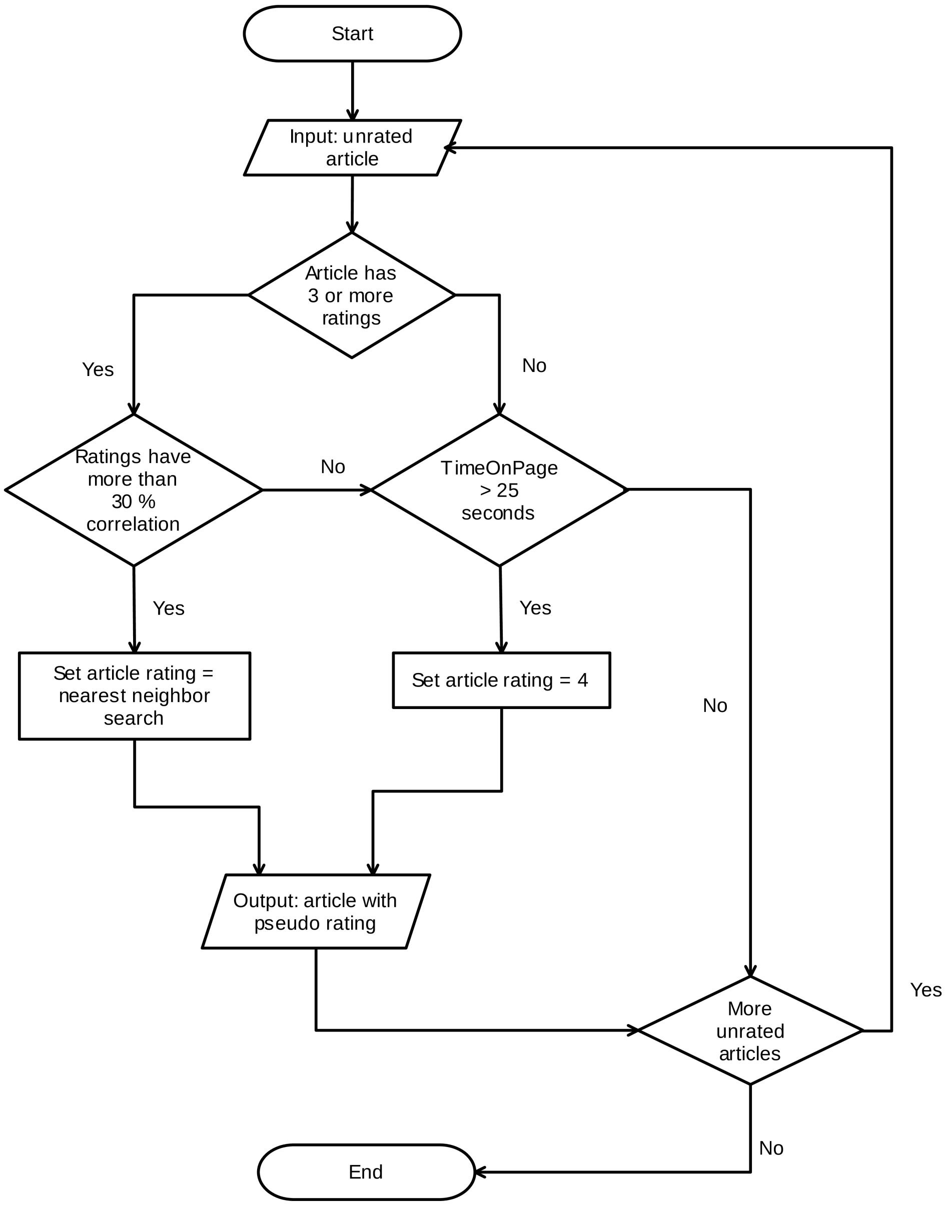

Start
Input: unrated article
Article has 3 or more ratings
Yes
No
Ratings have more than 30 % correlation
No
TimeOnPage > 25 seconds
Yes
Yes
Set article rating = nearest neighbor search
Set article rating = 4
No
Output: article with pseudo rating
More unrated articles
Yes
No
End

FIGURE 10.1: The steps in preprocessor

degree of preference towards sports for a user. When we calculate the product of a user vector and an article vector, we get the estimated preference of a user towards that article. Thus, we can get an ordered list of recommendations by calculating the estimated preference of all articles a user have not already read. The number of latent factors one should choose depends on the size of the data set. We use 10 latent factors for our implementation, but that value is subject to increase as the data set grows bigger.

Part IV

Evaluation and Conclusion

Chapter 11

Evaluation of Mapping Techniques

This chapter outlines our evaluations on a news data set. It starts with introducing the data set in Section 11.1, then the different configurations of the algorithms and the environment are presented in Section 11.2 and Section 11.3. How we conducted the evaluations are explained in Section 11.4. We further present our results in Section 11.5 and at the end of this chapter discuss our findings in Section 11.6.

11.1 News Data Set

The news data set used in the evaluations is called the "YOW User Study Data: Implicit and Explicit Feedback for News Recommendation" and was originally collected by the Principal Investigator in Carnegie Mellon University. After their evaluations, they made it publicly available online[1].

The goal of the study was to collect a data set from a variety of users with explicit feedback, implicit feedback and a wider range of information about news articles and topics. It was conducted during a four week time period, and the currently available data set contains 24 users, 5921 news articles and a total of 10010 rows with explicit and implicit feedback. More details about how the user study was conducted and statistics about it can be found in [37].

[1]http://users.soe.ucsc.edu/~yiz/papers/data/YOWStudy

Tables 11.1 and 11.2 present some of the information collected during the user study. The data set consisted of many missing values, both explicit and implicit feedback. Missing values in explicit feedback was mostly due to the fact that it was optional to give this feedback, but missing values in implicit feedback means that users did not use the keyboard buttons that was monitored, or moved the mouse enough to give feedback.

Since the SmartMedia application only collects ratings as explicit feedback, and currently only time spent reading an article as implicit feedback, we modified the YOW data set to better resemble it. We also included time on mouse, because it could be included in the SmartMedia application in the future.

Table 11.1: Explicit feedback collected in the user study

Explicit Feedback		
Classes	String	Topics the article belonged to
User_like	1-5	The rating
Relevant	1-5	How relevant was the news related to the classes
Novel	1-5	How novel the article was
Authoritative	Boolean	Whether the news was authoritative

Table 11.2: Implicit feedback collected in the user study

Implicit Feedback		
TimeOnMouse	int	ms spent moving the mouse
TimeOnPage	int	ms spent on a page / reading article
EventOnScroll	int	number of clicks on scroll bar
ClickOnWindow	int	number of clicks inside browser window
Keyboard activities	int	Other keyboard activities

11.1.1 Modified Data Set Used in Evaluations

The YOW data set used in our evaluations had four major modifications: (i) first we trimmed away feedback that was not necessary for our algorithms; (ii) next we removed all duplicate entries; (iii) then we removed all -1 ratings that already exists in the data set; (iv) at last we changed the user_like value to -1 on a range of feedback in order to have data to map from implicit feedback to explicit feedback (in our case user_like). Reasons behind the modifications and more details are elaborated next.

As mentioned earlier, the only explicit feedback used in our recommender are ratings, which maps to user_like in the YOW user study. Of the implicit feedback in the YOW user study, only TimeOnPage and TimeOnMouse had enough entries to be useful in our preprocessor. Consequently, the data we used from the YOW user study are presented in Table 11.3. All other feedback was trimmed away.

Table 11.3: Data from YOW user study used in our evaluations

Data set				
User id	Item id	User_like	TimeOnPage	TimeOnMouse
int	int	1-5	int	int

When we inspected the trimmed data set we observed several duplicates, with duplicate we mean that a user has read same article more than once with or without ratings. Since we do not know which feedback that was added as a mistake or which feedback that fits the users true feedback the most, we removed all duplicates. This ensured that the data set we are working with consists only of valid explicit-implicit feedback mappings.

However, the trimmed and non-duplicate data set had only 106 missing user_likes. Missing user_likes are represented as -1, and for these ratings, our preprocessor tries to use the implicit feedback to guess a pseudo rating that can be used instead. To make matters worse, most of these -1 user_likes were missing preferences on items without any other ratings. This means that the baseline estimate, that calculates a RMSE on the data set where ratings are between 1-5 do not know of this item, and will calculate a RMSE on a user-item matrix with less items than the estimate after the preprocessing part. Therefore, the next step was to remove these -1 ratings, and as a consequence a few items from this data set.

The final step was to select ratings where we can change the user_like without removing items from the user-item matrix. Our heuristic for this step is explained in the list below and a small example of the user-item matrix is presented in Tables 11.4 and 11.5.

1. Choose items with several ratings
2. If item has six or more ratings, change user_like to -1 on half of the ratings
3. If item has between two and six ratings, change user_like to -1 on one rating.

Properties of the data set during the different phases of the modifications can be seen in Table 11.6 and Figure 11.1 shows that these modifications do not change the characteristics of the data set.

Table 11.4: Original rating matrix

	User 1	User 2	User 3	User 4	User 5	User 6
Item 1			3			
Item 2	4			2	2	1
Item 3		4		5		
Item 4	3	5	2	1	3	3
Item 5	3				4	5

Table 11.5: Modified rating matrix

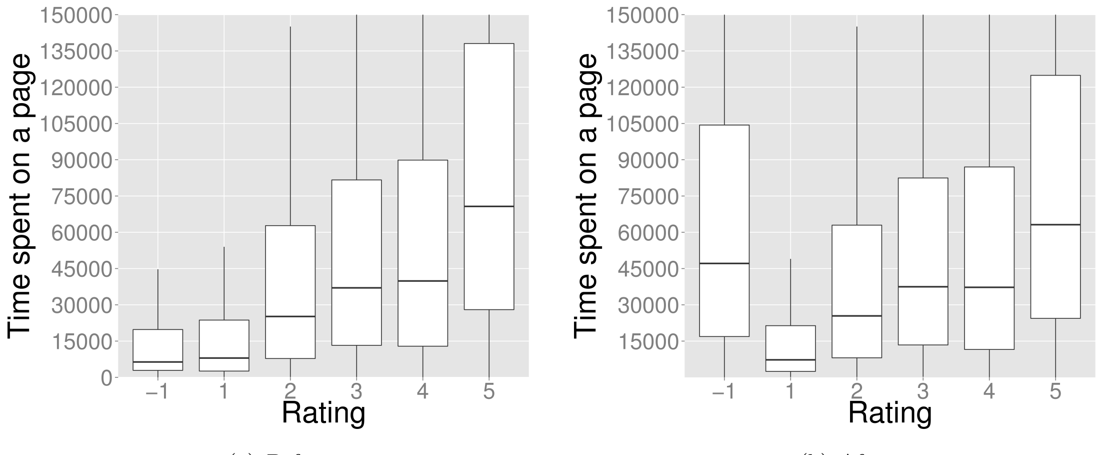
	User 1	User 2	User 3	User 4	User 5	User 6
Item 1			3			
Item 2	4			2	-1	1
Item 3		4		-1		
Item 4	-1	5	-1	-1	3	3
Item 5	3				4	-1

Table 11.6: Data set properties during different stages

Data set	**Rows**	**Users**	**Items**	**User_likes**	**Implicit feedback**	**Density**
Original	10010	24	5921	9829	9996	7.04%
Step 1	9979	24	5907	9824	9979	7.04%
Step 2	8430	24	5615	8312	8418	6.26%
Step 3	8312	24	5518	8312	8312	6.28%
Step 4	8312	24	5518	6653	8312	6.28%

11.2 Algorithms

The recommendations that are evaluated are processed in two steps: (i) the preprocessing step and (ii) the recommendation step. The focus in this thesis is the preprocessing step, where we infer pseudo ratings for news articles without a rating based on the implicit feedback they have. Every preprocessor has the same data set as input, and based on this data set they try to return a more dense data set where the new pseudo ratings contributes to better accuracy. The different properties in each preprocessor are presented in Tables 11.7 to 11.14, and during



(a) Before (b) After

FIGURE 11.1: Time on page versus ratings before and after sampling the YOW data set: (a) before and (b) after

the evaluation we test all possible configurations with these properties. For example, the preprocessor for Time Threshold has two properties with respectively four and two values which means that the total number of configurations for this preprocessor is $4 \times 2 = 8$.

The recommendation step is done by using Mahouts SVD recommender with SGD factorization using 10 latent factors. It uses the preprocessed data set, which only has explicit feedback represented as ratings. Thus, for each configuration, we reload the unprocessed data set, preprocess it with a set of configurations, then pass it to the SVD Recommender, where we do 5000 evaluations of the root mean squared error of the recommendations given by the recommender. From this, we can see which preprocessor and which parameters that yields the best end result.

TABLE 11.7: Configurations of Correlation and Time Threshold

TimeOnPage threshold	**Correlation limit**	**Prediction method**
15 000	{0.0, 0.1, 0.2 ...0.9}	Linear regression
20 000		Closest neighbor
25 000		Equal bins
30 000		

11.3 Environment

The evaluations were conducted on three Linux nodes, each running a 2.67 GHz Intel Xeon CPU with 6 cores and 6GB of RAM.

Table 11.8: Configurations of Correlation

Correlation limit	**Prediction method**
{0.0, 0.1, 0.2 ... 0.9}	Linear regression
	Closest neighbor
	Equal bins

Table 11.9: Configurations of Time Threshold

TimeOnPage threshold	**Rating**
15 000	4
20 000	5
25 000	
30 000	

Table 11.10: Configurations of Clustering

Clusterer	**Cluster data set**	**Distance function**
K-Means	Time on page	Euclidean
X-Means	Time on page and mouse	Manhattan
Density K-Means	Page times mouse	Chebyshev
EM		
Cobweb		
Farthest First		

Table 11.11: Configurations of K-Nearest Neighbor

Error minimization	**Confidence threshold**	**K**
MSE	{0.3, 0.4, 0.5 ... 0.8}	{1, 3, 5 ... 19}
MAE		

Table 11.12: Configurations of Multiple Linear Regression

Number of independent variables	1	2	3

Table 11.13: Configurations of Support Vector Machine

Kernel	**C**	**Gamma**
RBFKernel	2^n, $-15 \leq n \leq 15$	2^t, $-6 \leq t \leq 6$

Table 11.14: Configurations of Artificial Neural Network

Learning rate	{0.001, 0.003, 0.005 ... 0.3000}
Epochs	{100, 200, 300 ... 1000}

11.4 Testing Methodology

10-fold cross validation with hold-out set, that is, the pseudo ratings we infer during the preprocessing are only in the training set. The RMSE is calculated by doing 5000 iterations of each configuration and then returning the mean value.

11.5 Results

11.5.1 Baseline Estimate

The baseline estimate is calculated using only explicit feedback, and it dictates if the configurations using implicit feedback are valuable or not. The RMSE on the baseline estimate was **1.031**, and consequently, configurations with lower RMSE imply that using implicit feedback improves the recommendations and vice versa.

Figure 11.2 presents the best result from each configuration, revealing that mapping implicit feedback with a simple mapping method combined with a time threshold of how long users spend reading articles has the lowest RMSE and is the preferred technique. The figure also show that the data set may not be suitable for either multiple linear regression or clustering mapping techniques, where we believe the main reason is the lack of doing a quality assurance of the generated pseudo ratings. Without the quality assurance, both good and bad pseudo ratings are used in the training phase, which affects the quality of the model, resulting in high RMSE.

11.5.2 Time Threshold

Using the time a user spends reading each article yielded good results. Table 11.15 presents the results, where TimeOnPage threshold as 25 seconds and rating as 4 received the lowest RMSE of 1.008, which represents a 2.23% improvement over the baseline estimate. These results fit well with the results Morita and Shinoda found in their analysis. In [14] they observed that if they treated articles users spent more than 20 seconds reading as interesting, they could use this feedback to get precision and recall values of 70% and 30% of the retrieved articles.

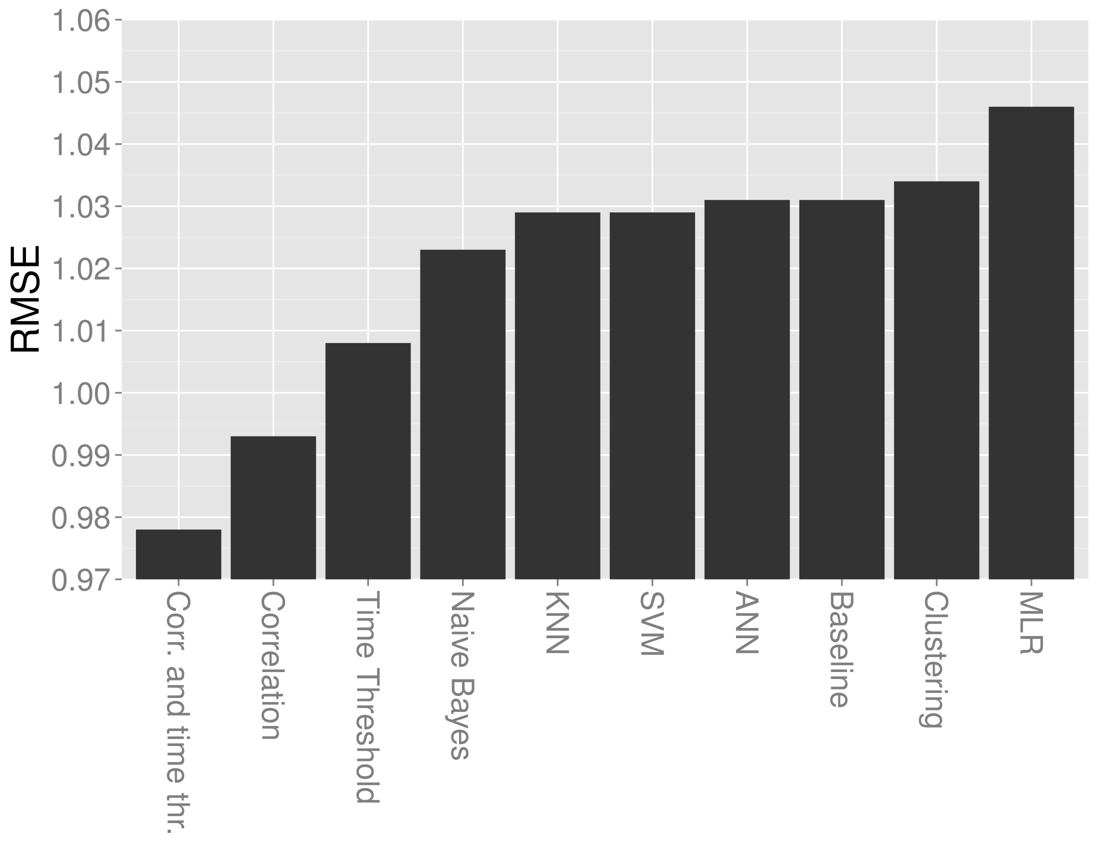


FIGURE 11.2: RMSE of best performing techniques compared to baseline

We further observe that rating an article based on the time user has spent reading it with four, gives continuously better results than giving it five. Looking at Figures 11.3 and 11.4, this is to be expected since rating four has the highest frequency in the data set. Combined with a time threshold the probability of an unrated article having rating four is higher than other ratings. Similar behavior can be seen in the news article data set collected by Claypool et al. [15], indicating that these characteristics describe the news domain in general.

The time spent reading articles is very news domain specific, but we believe that the reason it works so well is because time spent reading and ratings are linearly correlated and that rating four has the highest frequency. Other studies with linearly correlated data sets show the same results [13, 14], which means that utilizing the behavior in the data set, in this case the linear correlation between ratings given by the users and the time spent reading the news articles, has proven to be beneficial to recommend items using implicit feedback.

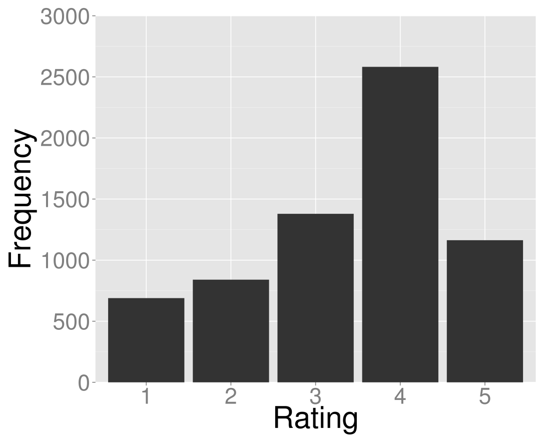


FIGURE 11.3: Rating frequency in data set

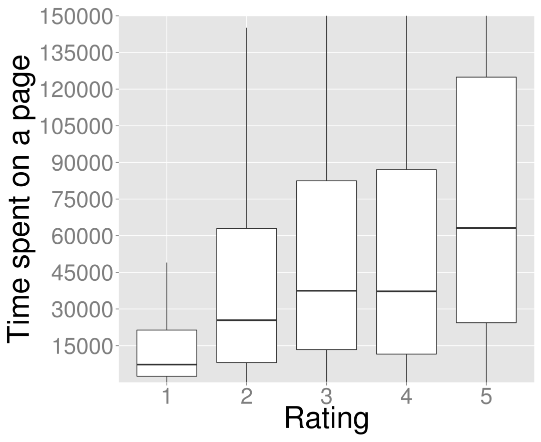


FIGURE 11.4: Time on page versus ratings

TABLE 11.15: Results when using time spent reading

#	RMSE	Min time on page	Rating	Pseudo ratings
1	1.008	25 000	4	788
2	1.011	15 000	4	937
3	1.012	30 000	4	707
4	1.012	20 000	4	869
5	1.015	30 000	5	707
6	1.016	25 000	5	788
7	1.022	20 000	5	869
8	1.025	15 000	5	937

11.5.3 Correlation

The results using a correlation limit and simple prediction methods are presented in Table 11.16 and reveals that closest neighbor is the preferred prediction method, as the top seven configurations use this method. We included the best RMSE for linear regression and equal bins as well in order to compare them. The best configuration had a correlation limit of 0 and prediction method as closest neighbor and received a RMSE of 0.993. This represents a 3.69% improvement over the baseline estimate.

We observe that closest neighbor and linear regression is not dependent on having a correlation between the ratings and the implicit feedback to generate good pseudo ratings. However, it is important to be aware that the prediction method is only used on unrated articles, which at least three other users have rated (this is necessary in order to find a correlation). The data set overall is linearly correlated, which means that most of the articles with at least three ratings have a linear correlation between the rating and the time spent per page. This is shown in Table 11.16 where there are only 72 more pseudo rating when the correlation limit is 0.4 instead of 0.0. Therefore, we believe that for other data sets that are not as correlated as ours, the correlation limit becomes more important.

In contrast, equal bins does not have this behavior. This is to be expected, as equal bins need good correlation examples in order to create reasonable bins. This can be seen by observing that the best configuration with equal bins has a correlation limit of 0.9 and the worst has a correlation limit of 0.0. However, we expected this method to perform better than it does with high correlation. One possible reason for the bad performance is that the bins should have varying size, so with equal size it consistently misses the right bin, resulting in bad performance.

We expected linear regression to perform better, as Parra and Amatriain used multiple linear regression (with both single and multiple implicit feedback) with great results in their analysis of their Last.fm data set[13]. However, in their implementation they used regression on all unrated songs, not only those with a correlation above a threshold. Therefore, to get a better comparison between our results and theirs based on multiple linear regression, we did the same and the results are presented in Section 11.5.5.

TABLE 11.16: Correlation results

#	RMSE	Corr. limit	Pred. method	Pseudo ratings
1	0.993	0.0	Closest Neighbor	585
2	0.994	0.2	Closest Neighbor	579
3	0.994	0.3	Closest Neighbor	577
4	0.995	0.1	Closest Neighbor	580
5	0.999	0.4	Closest Neighbor	513
.	.	.	.	.
8	1.008	0.1	Linear Regression	580
.	.	.	.	.
21	1.047	0.9	Equal Bins	244
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
30	1.078	0.0	Equal Bins	585

11.5.4 Combined Time Threshold and Correlation

Combining both correlated prediction methods and time threshold had the lowest RMSE values of all the mapping techniques as shown in Figure 11.2. The rating that is generated when the time threshold used is four, which is motivated by our results in Table 11.15. Table 11.17 presents the results from the top five performing configurations. We also included the best RMSE when the prediction method was linear regression and equal bins as well, in order to compare them with the top five configurations. We observe that the lowest RMSE is 0.978, which represents a 5.14% improvement over the baseline estimate.

Again, we observe that closest neighbor is superior to the other prediction methods and 25 seconds is the preferred time threshold. It is interesting that the correlation limit is 0.3, which indicates that articles with less correlation is better off with an analysis of how long the user spent reading the article.

We believe the main reason that this technique performs better than the involved techniques individually is that it is able to generate more high quality pseudo ratings. With this technique, we are able to generate 1033 pseudo ratings, while the best time threshold configuration generates 788 ratings and the best correlation configuration generates 585 pseudo ratings. Consequently, we conclude that as long as the pseudo ratings are of high quality they should be used to generate recommendations.

TABLE 11.17: Combined time threshold and correlation results

#	RMSE	Time thres.	Corr. limit	Pred. method	Pseudo ratings
1	0.978	25 000	0.3	Closest Neighbor	1033
2	0.979	20 000	0.0	Closest Neighbor	1088
3	0.979	25 000	0.1	Closest Neighbor	1036
4	0.979	25 000	0.2	Closest Neighbor	1035
5	0.980	15 000	0.0	Closest Neighbor	1128
.	.	.	.	.	.
26	0.991	25 000	0.0	Linear Regression	1041
.	.	.	.	.	.
41	1.027	15 000	0.8	Equal Bins	862
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
108	1.054	30 000	0.0	Equal Bins	993

11.5.5 Multiple Linear Regression

Table 11.18 presents the results of using multiple linear regression on the data set with one, two or three independent variables: (i) TimeOnPage - TOP, (ii) TimeOnMouse - TOM, and (iii) TimeOnPageTimesTimeOnMouse - TOPTTOM. The beta constants are calculated using all news articles with both explicit and implicit feedback, and pseudo ratings are generated for all unrated news articles. The results indicate that this technique does not generate enough high quality pseudo ratings to be beneficial, as the best models perform worse than the baseline estimate.

This technique was motivated by the results of Parra and Amatriain in [13]. However, we do not get any improvement over the baseline estimate as opposed to Parra and Amatriain and we believe the cause is two-fold. First, since we do not remove outliers or extreme values in the training data, that is, ratings with both explicit and implicit feedback, these values negatively affects the model and as a consequence affects the quality of all the pseudo ratings. Examples of outliers in the news domain are users that starts reading an article, then becomes preoccupied before they finish the article and then rate it. Similar outliers in the Last.fm data set do not exist and consequently their model becomes better than ours. Second, as discovered in the previous section, only high quality pseudo ratings improve the recommendations, but in this technique all unrated articles are given a pseudo rating independent of the quality of the pseudo rating.

Regardless of the resulting RMSE, we think it is interesting to note that including other implicit feedback variables than time spent on page in the same model increases the performance. This indicates that including other implicit feedback variables in the SmartMedia application is beneficial as long as some correlation between the explicit feedback and the other implicit feedback exist. By doing a quality check of the pseudo ratings before using them, we believe that such a technique could offer good recommendations.

Table 11.18: Multiple linear regression results

#	**RMSE**	**Model**
1	1.046	$r_{ui} = \beta_0 + \beta_1 \cdot TOP_{ui} + \beta_2 \cdot TOM_{ui}$
2	1.046	$r_{ui} = \beta_0 + \beta_1 \cdot TOP_{ui} + \beta_2 \cdot TOM_{ui} + \beta_3 \cdot TOPTTOM_{ui}$
3	1.048	$r_{ui} = \beta_0 + \beta_1 \cdot TOP_{ui}$

11.5.6 Clustering

Table 11.19 presents the results where clustering is used to map different ratings to clusters, and then classify new unrated news articles to the closest cluster. The expectation-maximization configuration with the data set with time on page, time on mouse and time on page multiplied with time on mouse received the lowest RMSE of 1.033. However, this is worse than the baseline estimate and questions the benefits of using clustering as a mapping method. These results indicate that implicit feedback from the news domain, represented by our data set, is not fit for clustering.

The results from the COBWEB configurations support this hypothesis, as COBWEB did not create different clusters for each rating. COBWEB found only 1 cluster, which mapped to rating 4. With one cluster in this preprocessor, all COBWEB configurations rates all unrated articles as 4. As a consequence, COBWEB configurations results are the same if we just gave all unrated articles rating 4 without doing any other kind of preprocessing.

Nonetheless, given the performance of EM we still believe that clustering techniques have potential. Clustering classifies all missing ratings, that is, 1659 pseudo ratings and with only 0.3% worse performance, we believe that with further enhancements to the clustering configurations e.g. only classify instances that have a certain probability value (as done with Naive Bayes and KNN) combined with a better training model could improve the clustering results.

Table 11.19: Clustering results

#	RMSE	Clusterer	Cluster data set	Dist. Func.
1	1.034	EM	Page times mouse	-
2	1.040	X-Means	Page times mouse	Manhattan
3	1.040	COBWEB	Time on page and mouse	-
4	1.040	COBWEB	Page times mouse	-
5	1.040	EM	Time on page	-
6	1.041	COBWEB	Time on page	-
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
30	1.061	Density K-Means	Time on page	Euclidean

11.5.7 Naive Bayes

TABLE 11.20: Naive Bayes results

#	RMSE	Threshold	PseudoRatings
1	1.023	40%	726
2	1.033	70%	76
3	1.034	60%	99
4	1.035	50%	174
5	1.040	10%	1659
6	1.040	20%	1659
7	1.041	30%	1659

Table 11.20 presents the results given by mapping implicit feedback to explicit feedback using a Naive Bayes classifier. As seen in the table, when using a confidence threshold of 40%, the classifier is able to improve the RMSE with about 0.88%, which is not very significant. At this threshold level, the classifier is able to infer 726 ratings. However, the RMSE is not improved to any significance because the quality of the ratings is not good enough. This is also evident at lower thresholds, as the inferred ratings increase the RMSE. At higher thresholds, the quality of the ratings is better, but since the classifier is not able to infer more than 200 ratings, the effect they have on the RMSE is too small, and thus the RMSE lies between 1.033-1.035, very close to the baseline RMSE.

Parra and Amatriain had an improvement of over 6% over baseline estimate in [13] using a regression method, and wanted to explore Bayesian Models in their future work. With this evaluation we see that Naive Bayes is not a mapper feasible to use with this data set. However, since we only classify based on a single parameter, the naive assumption of Naive Bayes is effectively insignificant in this evaluation. Thus, it is likely that we would see different results if we had more kinds of implicit feedback available.

11.5.8 K-Nearest Neighbor

In Table 11.21 we show the results yielded by preprocessing using KNN classification. As we can see, the improvement upon the baseline evaluation is marginal, with an RMSE of the top three configurations of 1.029. Using KNN, we are able to set a threshold on the confidence value for each classification. Thus, we can

Table 11.21: Instance-based k-nearest neighbor classifier results

#	RMSE	Pseudoratings	Threshold	K	Minimization
1	1.029	980	40%	15	MinimizeMeanSquaredError
2	1.029	24	80%	7	MinimizeMeanAbsoluteError
3	1.029	980	40%	15	MinimizeMeanAbsoluteError
4	1.030	34	80%	5	MinimizeMeanSquaredError
5	1.030	1130	40%	17	MinimizeMeanSquaredError
.	.	.	.	.	.
.	.	.	.	.	.
.	.	.	.	.	.
120	1.048	1659	30%	1	MinimizeMeanAbsoluteError

exclude the samples where the classifier has a low confidence in the classification. We see that the top result has a threshold of 40%, yielding 980 inferred ratings for an RMSE of 1.029. However, we get an equal RMSE using a threshold of 80% yielding only 24 ratings. It is safe to assume then, that a significant amount of the ratings inferred with a 40% threshold are of a low quality, therefore trifling the potential RMSE increase caused by the higher-quality ratings. At the same time, we see that the threshold can be lower with a higher value of K. This is because we are dealing with a noisy data set. Thus, with higher values of K, the outliers are ”smoothed out”, and become less influential. Another observation is that the minimization measure does not affect the result in any way.

11.5.9 Artificial Neural Network

Table 11.22: Artificial neural network results

#	RMSE	Learning rate	Epochs
1	1.031	0.041	310
2	1.031	0.041	510
3	1.032	0.021	310
4	1.032	0.081	910
5	1.032	0.021	410
6	1.032	0.021	210
.	.	.	.
.	.	.	.
.	.	.	.
150	1.052	0.001	10

Table 11.22 outlines the RMSE given by ANN classification. We can see that the classifier is not able to improve the RMSE, yielding a top RMSE of 1.031. Since the classifier is non-probabilistic, we have no way of telling the confidence of a classification. Thus, we infer all pseudo ratings, whether or not we are confident that they are in fact correct. This will likely lead to many low-quality ratings, and consequently the overall RMSE is the same as the baseline estimate.

If we look at the parameters, learning rate and epochs, it is unsurprising that a learning rate of 0.001 with 10 epochs yields a poor result. This comes from the fact that with only 10 epochs, the algorithm will not have reached convergence before stopping. There is often a balance between learning rate and epochs in terms of accuracy and performance. With few epochs, the algorithm is faster, but it also requires us to increase the learning rate in order to converge within the set number of epochs. Respectively, if we have a low learning rate, we need to many epochs in order to reach convergence, thus increasing the computational cost. In our evaluations, we did a grid search of 10 to 1000 epochs with increments of 100, and learning rate from 0.001 to 0.3 with increments of 0.02.

Due to a lack of time and resources, we were not able to do a full grid search of ANN, and we believe that the classifier should be investigated further as a preprocessing algorithm.

11.5.10 Support Vector Machine

Table 11.23: Artificial neural network results

#	RMSE	C	Kernel gamma
1	1.029	128	32
2	1.031	0.00781	16
3	1.031	0.25	8
4	1.031	0.5	1
5	1.031	2.0	64
.	.	.	.
.	.	.	.
.	.	.	.
272	1.239	512	0.06250

Table 11.23 outlines the results seen when using a support vector machine (SVM) based on regression. As we can see, the SVM method does not improve the result

by any significant amount, with lowest RMSE of 1.029, just a 0.002 improvement over baseline. The table also shows that there are no apparent patterns in the parameters. A complexity constant C of 0.00781 performs almost equal to a C of 128, while the worst performance was seen with a C of 512. We believe that noisy data is the main reason for the poor performance of the SVM. As we outline in Section 7.5.4, the SVM constructs a hyperplane with the goal to separate the different instances. If these instances are very noisy, the SVM will not be able to construct a good hyperplane, resulting in poor results.

11.6 Discussion

Our main findings from the evaluation of the recommender are:

1. Combining neighborhood search and time threshold gave best results
2. Simple mapping techniques performed better than more complex ones
3. High quality pseudo ratings and the number of pseudo ratings are the main reasons for high performance

The evaluation demonstrated that combining a neighborhood search with one neighbor (if it was possible to find some correlation between the ratings) and a time threshold for articles without correlation or enough ratings produced the best results. The latter method is very news domain specific and the correlation between time spent reading news articles and ratings is well studied in [14, 15]. However, similar methods for other domains should perform well as long as an analysis of the domain similar to those in the news domain is conducted. What is more interesting is the good results from using a neighborhood search with only one neighbor. As Section 11.5.3 shows, this technique does not even need high correlation to perform well. We suspect the reason is that users tend to rate the same articles similar, explaining that one neighbor is enough and on average leads to high quality pseudo ratings.

After all the configurations of the recommender algorithm had been tested, we observe that simple mapping techniques represented by the Time and Correlation configurations performed better than the more complex techniques such as clustering, artificial neural network and other classifier algorithms. We can think of several reasons for these results. First, the complex techniques need more fine-tuning than we did in order to perform at the same level as the best configurations. Some of this tuning is mentioned above, such as a probability check for each classified news article when clustering or SVM are used, inspired by Naive Bayes and K-nearest neighbor results. The probability check then ensures that only high quality pseudo ratings are generated. Other fine-tuning is shown by the importance of selecting the parameter combinations that performs the best. Furthermore, complex techniques may have higher restrictions concerning the data set characteristics and consequently performs better at data sets suited for each mapping technique. Therefore, simple mapping techniques do not have these constraints and can perform on a much wider data set format, such as the news domain. Another reason is that the complex configurations creates a model based on the data set as a whole, while the simple techniques creates a model locally for each produced pseudo rating. When the model is created based on the whole data set, it is more prune to noise and when the model ”smooths” the data set, the pseudo ratings tends to get the same rating.

We also discovered that to get a substantial decrease in RMSE, the preprocessor had to generate many high quality pseudo ratings. We believe that high quality pseudo ratings are necessary since the SVD matrix factorization algorithm that is used to generate recommendations is good to begin with. As a consequence, the pseudo ratings we infer must be better than the guesses from the recommender algorithm. Figure 11.5 shows how the RMSE changes when the classify threshold changes and Table 11.20 shows the number of pseudo ratings for each threshold. As the figure and table indicates the optimal threshold is 0.4, which is when the quality of pseudo ratings and the number pseudo ratings generated are balanced optimally.

Of all the techniques discussed, we believe that the using time a user spends reading an article is the only mapping technique limited to the news domain. The other techniques adept well to other domains since they do analysis of the data set when they create a model, e.g., clustering clusters the data set before it classifies new unrated news articles, multiple linear regression finds the beta constants by using the data set and other classifier techniques do similar steps.

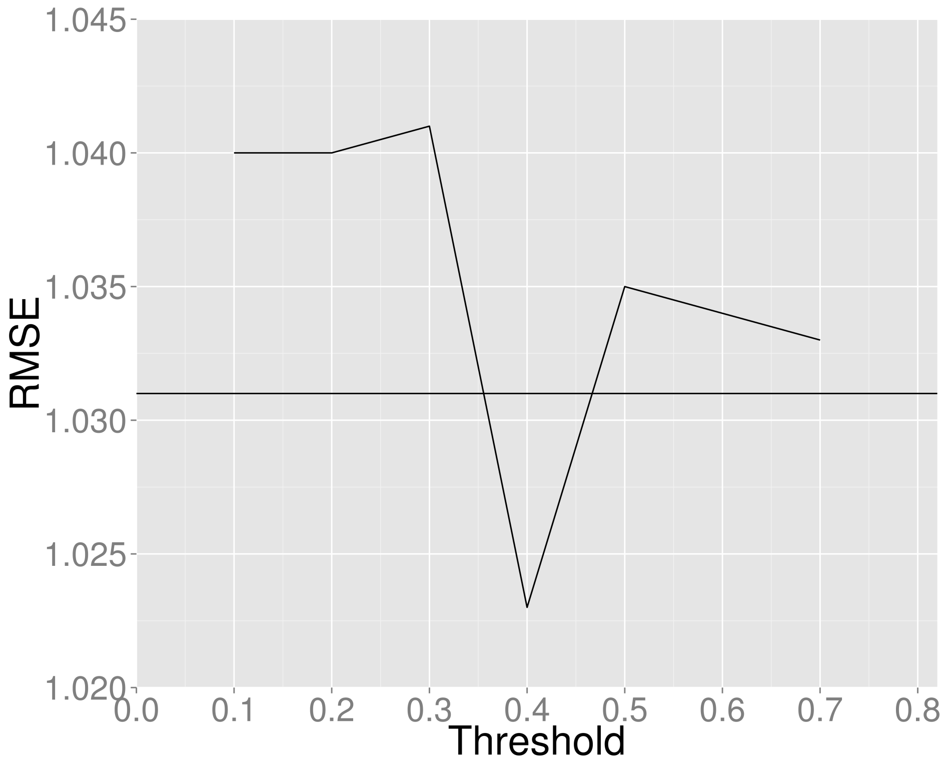


FIGURE 11.5: Naive Bayes configuration plotted at different classify threshold with baseline estimate border

Chapter 12

Integrational Issues

In Section 9.3 we described how we integrated our collaborative filtering recommendation component to the SmartMedia news application. In this chapter we discuss and evaluate the integration. In Section 12.1 we the modifiability of the integration and in Section 12.2 we evaluate its scalabilty.

12.1 Modifiability

SmartMedia is an ongoing research project where the collaborative filtering component is one of several filtering methods used. These various filters can then be weighted by a scheme chosen by the user. Because we need to integrate a number of standalone filters, each filter is required to implement a specific interface as defined by the SmartMedia application. This interface is further elaborated in Section 9.3. When designing the evaluation pipeline, and constructing the API, we faced similar challenges. We wanted to be able to be able to modify the mapping technique as well as the related parameters with ease. To solve this problem, we designed a preprocessor interface, which is implemented by each preprocessor or mapping technique. When constructing a preprocessor, we pass a configuration object to it, in which we store all relevant parameters for that specific preprocessor. A pseudocode for this process is given in Algorithm 7.

12.2 Scalability

An important measure of a recommender system is the computational performance. In the collaborative filtering component there are mainly two relevant measures. (i) "How fast can we provide recommendations?" and (ii) "How fast can we train the recommendation model?" With the integration to the SmartMedia news application, we had the opportunity to measure this performance in a real system. To answer question (i) we measured the round trip time delay seen when requesting recommendations from a client. For (ii) we measure the time taken to build the recommendation model, and how much overhead is incurred with preprocessing the feedback. From these results we can make a suggestion of how often the model should be rebuilt. We want to keep it as up to date as possible, without causing unnecessary performance strain on the database and the overall system performance. These two evaluations are presented next.

12.2.1 Recommendations

To measure how fast we can compute recommendations, we view it from a client's perspective. How fast is a request for recommendations answered by the real system? To answer these questions we simulated requests for a random selection of 18 anonymous users in the system. Specifically, for each of these users, we sent 10 request for recommendations to /recommend/{userid} using cURL[1]. The results of these measurements are given in Table 12.1, where we show the mean of each measurement. The manual for cURL[2] describes how we should interpret the timings, which we have converted from seconds to milliseconds. Given an initialization time init, the DNS lookup time is the time since init to read the IP address from cache. The connect time, is the time since init to an established TCP connection and app connect is the time since init to a successful TLS handshake. Pre-transfer is the time since init to the start of transferring the request data to the server and start-transfer is the time since init to the first byte was about to be transferred. If we look at the total, we see that the mean delay of these requests is 111.9 ms, where the app connect delay counts for close to 90%. While TLS is not needed for the CF API when it is behind the news API proxy, we chose to enable it for the CF API in order to get a more realistic result.

[1]http://curl.haxx.se
[2]http://curl.haxx.se/docs/manpage.html#-w

Table 12.1: Mean recommendation times

DNS lookup	Connect	App connect	Pre-transfer	Start-transfer	**Total**
1.2 ms	4.7 ms	102.1 ms	102.2 ms	111.7 ms	**111.9 ms**

An illustration of the recommendation process is given in Figure 9.1, where the user Mark is requesting recommendations. However, since parts of the news API was not implemented in time, we sent the requests directly to the CF API. This should be considered when inspecting the results. We believe that the extra delay caused by proxying requests through the news API would be negligible, but that should be confirmed in later experiments.

12.2.2 Build Time

We measure the build time as the duration taken between the initial build instruction to we have a model ready to provide recommendations. This procedure consists of three steps. First, we need to load the data set from the database or file storage. Then we need to preprocess the data set and infer potential ratings, and finally we build the recommendation model using matrix factorization. A problem we faced with this evaluation was that the SmartMedia project was still in a development phase when we did our evaluations, meaning that there were very few users using it. This, in turn meant that it contained very little feedback, and the build time of such a small data set would not have been realistic. For this reason, we decided to synthesize a larger data set with similar properties as the YOW data set we used for our evaluations in. The YOW data set was introduced in 11.1. The synthesization was done using an online service[3], and the properties of this data set are shown in Table 12.2. It contains close to 1 million rows, with the density of explicit feedback being 2.7%.

Table 12.2: Properties of synthesized data set

Rows	**Users**	**Items**	**User_likes**	**Implicit feedback**	**Density**
979343	6000	4000	649414	979343	2.7%

After synthesizing this data set, we evaluated the build time on it. The duration of each step is shown in Table 12.3. From the table we can see that the total build

[3]http://www.mockaroo.com

time is less than 5 minutes. The scalability of the matrix factorization algorithm has been proven in earlier work[38, 39], and takes only 2 seconds to complete. The interesting parts here are the data set loading and the preprocessing. The preprocessing time takes 1 minute and 17 seconds, while the loading of the data set takes 3 minutes and 25 seconds, counting for $\frac{2}{3}$ of the total duration. This means that the preprocessing algorithm is scalable, and the bottleneck lies in parsing the data set. In the current implementation, the full data set is loaded on each build. In future work, however, we can implement streaming updates, and only update the parts changed in the data set. This will likely limit the long load time to the initial bootstrap phase, which only occurs once. It is also relevant to mention that the matrix factorization algorithm is based on Apache Mahout, and uses the highly optimized data structures bundled with the library, while the data structure used for the preprocessor is made to support implicit data. It is likely that this data structure can be optimized in order to lower the preprocessing time even further. Given a build time of less than five minutes, we can propose a maintenance scheme where we rebuild the model every 10 minutes if there have been changes to the data set. The 5-minute buffer is proposed as an extra precaution in cause of unforeseen performance strain on the database. Since the data set is continuously changing in the news, the duration of the build process should be monitored.

Table 12.3: Build time of synthesized data set

Load data set	Preprocess	Factorize explicit matrix	**Total**
3 min, 25 sec	1 min, 17 sec	0 min, 2 sec	**4 min, 44 sec**

Chapter 13

Discussion and Conclusions

This chapter contains the final remarks in our thesis. In Section 13.1 we discuss the dependency on explicit feedback and how it can be phased out. Section 13.2 contains the conclusions drawn from our work. In Section 13.3 we outline a few proposals for further work. Finally, Section 13.4 contains a few final remarks on our work.

13.1 Phasing Out Explicit Feedback

SmartMedia's goal is to eventually rely on implicit feedback to do recommendations. The main advantage of this approach is improved usability, since the application collects implicit feedback automatically and less information needs to be shown in the graphical user interface. However, as we have discussed in this thesis, there are challenges with this approach. The biggest challenge is how we classify which news articles that are interesting, and which are not, based only on information that is available for the application.

Hu et al. [12] introduce confidence and preference variables in their model to describe implicit feedback for items. Their model is adapted to the domain of TV shows where the preference was 1 if the user had seen the TV show or 0 if not. Then they used confidence as a parameter to describe how much a user liked the TV show. A similar model could be adapted to the news domain. The preference would be the same, 1 if a user has read the article or 0 if not. However, the confidence must be modified to reflect the news domain, and a parameter in the confidence equation could be the time spent reading an article, which has shown to be a good mapping technique. In addition, an evaluation methodology have to be created that can evaluate and find interesting news article to be recommended. The evaluation methodology can be tuned as the application is used in order to further improve the recommendations.

Das et al.[6] utilize even less information to create recommendations on the Google News web site. Their recommender system only records user click history on news articles, and if we adapt Hu et al.'s approach to describe the numeric score of a story, the preference and confidence variables are as follows: preference is 1 if the user has read the article and 0 otherwise and confidence is calculated by the fractional membership of the different clusters times the number of times other users has read the article.

To our knowledge, there have not been a study that compares recommender systems that uses both explicit and implicit feedback and recommender systems that only utilizes implicit feedback. However, Hu et al. [12] and Das et al. [6] compared their recommender against recommending the most popular TV shows/news articles, as people tends to like the same TV shows/news articles. According to their study, using implicit feedback gave better results than the naive approach described, which indicates that even though explicit feedback is not available, it is useful to include implicit feedback.

13.2 Conclusion

In this thesis, we have looked at how we can incorporate implicit feedback in a news recommender system. The evaluations have been on a data set with both explicit feedback and implicit feedback, enabling us to compare different ways of using implicit feedback to improve the recommendations. The resulting recommender system has then been integrated to the SmartMedia news application as the collaborative filtering component.

The first research question was how implicit feedback could improve news recommendation. The majority of the current recommender systems focus on explicit feedback, because it is simple to interpret the users preference. However, we can also collect user behavior and determine whether it can improve the recommendation accuracy. In our evaluations, we observed that mapping implicit feedback to explicit feedback (represented as ratings on a scale between one and five) could improve news recommendation given the right mapping method. The 5.14% improvement in accuracy on our data set is comparable to the 6.5% improvement Parra and Amatriain[13] got when including implicit feedback for music recommendation.

The second research question was how different techniques for utilizing implicit feedback compared. Implicit feedback contains significant noise, making it difficult to find consistent patterns. We compared a set of different mapping techniques, and discovered that those utilizing the linear correlation between ratings and time spent reading each article received the lowest (best) RMSE scores. The preprocessor that used a nearest neighbor search with one neighbor to infer news ratings combined with a time threshold of 20 seconds for time spent reading each article yielded a 5.14% improvement over recommendations generated without implicit feedback.

The third research question was how we could integrate our collaborative filtering component to SmartMedia's recommender system. The news domain is very active, meaning that the data set is continuously changing as new articles are added and old ones removed. This imposes strict scalability requirements recommender algorithms, as they have to rebuild frequently and provide recommendations with minimum delay. We have developed a collaborative filtering component that is stand-alone with an intuitive API that is easily integrated to the SmartMedia application. Further, our component is scalable both in model build time and recommendation time; experimental results demonstrated that building the model with approximately 1 000 000 rows of feedback is done within five minutes and the time from a user requests recommendations and the recommendations is presented is less than 150 milliseconds, as presented in Section 12.2.1.

Further, we believe that our collaborative filtering component will perform well on the news domain in general. Our evaluations have been conducted on a data set collected on regular computers in 2005. We have since seen a plethora of different devices for consuming such content. We believe however, that the users will interact similarly with news articles independently of which device they use, and that the techniques we have proposed will need minimal change in configuration on a more recent data set.

The techniques we have implemented are news domain specific, such as the linearly correlation between user ratings and time spent reading each article. On other domains such as music or movies, our component might need to undergo modifications since these domains may have other characteristics to exploit. Yet, in general, we believe that the use of implicit feedback can improve the accuracy of recommender systems, given the right model.

13.3 Further Work

To continue this research we list a few suggestions for further work:

- To evaluate the SmartMedia application, the recommender system should be tested on a data set collected from SmartMedia. This data set contains Norwegian users and Norwegian news articles, and will collect the actual implicit feedback that will be used in the SmartMedia application. Such a data set is being collected as this thesis is written, and we think that evaluating this recommender on that data set will give valuable feedback to further improve the SmartMedia recommendations.
- In the current collaborative filtering component, we utilize time spent reading an article and time spent moving the mouse while reading an article. In the future, we believe that utilizing other types of implicit feedback such as share with a friend and adding article to favorites could further improve the recommendations, as indicated by the results of multiple linear regression and clustering where more feedback resulted in lower RMSE.
- As we outline in Section 4.2, there are many different kinds of implicit feedback that can be used to improve recommendation accuracy. Tensor factorization is a promising method for latent semantic analysis, which allows us to easily include all kinds of implicit feedback in the model. We are curious whether tensor factorization can perform better than the preprocessing step we have introduced in this thesis, and hope to see more work on this in the future.
- The current version of the collaborative filtering component can undergo further performance optimizations, such as streaming updates from the database,

faster data structures for feedback representation and streaming mapping techniques. This will allow it to scale much better without the need for additional hardware. It can also lead to better recommendations, as the model will always represent the current state as the system is used.

13.4 End Notes

An interesting project has come to an end. The insights obtained in the domain of recommender systems leaves us with a better understanding of their importance. An effective recommender system will improve user satisfaction and hopefully increase revenue. We believe that the collaborative filtering component we have integrated with SmartMedia will allow it to present interesting news articles to users.

Furthermore, the project has not been without challenges. Delays with the SmartMedia data collection triggered a late start of our evaluations and the scope of the master's thesis have been a challenge in terms of personal expectations and limits. With discipline and thorough planning, however, we have been able to answer our research questions and we now feel well prepared to enter the professional life.

Appendix A

All Evaluation Results

This appendix presents all the results we obtained in our evaluations. The following is a list of abbreviations found in the headers:

- **RMSE** root mean square error
- **IVs** independent variables
- **PR** pseudoRatings
- **mTOP** minTimeOnPage
- **corrLim** correlationLimit

A.1 Multiple Linear Regression Raw Results

RMSE	PR	IVs
1.046	1656	2
1.046	1656	3
1.048	1656	1

A.2 Time Threshold

RMSE	PR	mTOP	rating
1.008	788	25000	4

RMSE	PR	mTOP	rating
1.011	937	15000	4
1.012	707	30000	4
1.012	869	20000	4
1.015	707	30000	5
1.016	788	25000	5
1.022	869	20000	5
1.025	937	15000	5

A.3 Correlation and Time Threshold

RMSE	PR	mTOP	corrlim	predictionMethod
0.978	1033	25000	0.3	ClosestNeighbor
0.979	1035	25000	0.2	ClosestNeighbor
0.979	1036	25000	0.1	ClosestNeighbor
0.979	1088	20000	0.0	ClosestNeighbor
0.980	1041	25000	0.0	ClosestNeighbor
0.980	1080	20000	0.3	ClosestNeighbor
0.980	1082	20000	0.2	ClosestNeighbor
0.980	1083	20000	0.1	ClosestNeighbor
0.980	1128	15000	0.0	ClosestNeighbor
0.981	1122	15000	0.2	ClosestNeighbor
0.981	1123	15000	0.1	ClosestNeighbor
0.981	987	30000	0.2	ClosestNeighbor
0.982	1120	15000	0.3	ClosestNeighbor
0.982	985	30000	0.3	ClosestNeighbor
0.982	988	30000	0.1	ClosestNeighbor
0.982	993	30000	0.0	ClosestNeighbor
0.983	969	25000	0.4	ClosestNeighbor
0.985	1016	20000	0.4	ClosestNeighbor
0.985	1056	15000	0.4	ClosestNeighbor
0.987	918	25000	0.5	ClosestNeighbor
0.987	965	20000	0.5	ClosestNeighbor
0.988	921	30000	0.4	ClosestNeighbor
0.990	1005	15000	0.5	ClosestNeighbor
0.990	870	30000	0.5	ClosestNeighbor

RMSE	PR	mTOP	corrlim	predictionMethod
0.990	874	25000	0.6	ClosestNeighbor
0.991	1035	25000	0.2	LinearRegression
0.991	1036	25000	0.1	LinearRegression
0.991	1041	25000	0.0	LinearRegression
0.991	921	20000	0.6	ClosestNeighbor
0.992	1033	25000	0.3	LinearRegression
0.992	1082	20000	0.2	LinearRegression
0.992	1088	20000	0.0	LinearRegression
0.992	826	30000	0.6	ClosestNeighbor
0.992	961	15000	0.6	ClosestNeighbor
0.993	1083	20000	0.1	LinearRegression
0.993	1128	15000	0.0	LinearRegression
0.994	1080	20000	0.3	LinearRegression
0.994	1123	15000	0.1	LinearRegression
0.994	838	25000	0.7	ClosestNeighbor
0.994	993	30000	0.0	LinearRegression
0.995	1122	15000	0.2	LinearRegression
0.995	885	20000	0.7	ClosestNeighbor
0.995	985	30000	0.3	LinearRegression
0.995	987	30000	0.2	LinearRegression
0.996	1120	15000	0.3	LinearRegression
0.996	925	15000	0.7	ClosestNeighbor
0.996	988	30000	0.1	LinearRegression
0.997	1016	20000	0.4	LinearRegression
0.997	1056	15000	0.4	LinearRegression
0.997	775	25000	0.8	ClosestNeighbor
0.997	790	30000	0.7	ClosestNeighbor
0.997	822	20000	0.8	ClosestNeighbor
0.997	969	25000	0.4	LinearRegression
0.998	918	25000	0.5	LinearRegression
0.999	727	30000	0.8	ClosestNeighbor
0.999	862	15000	0.8	ClosestNeighbor
0.999	874	25000	0.6	LinearRegression
0.999	965	20000	0.5	LinearRegression
1.000	1005	15000	0.5	LinearRegression

RMSE	PR	mTOP	corrlim	predictionMethod
1.000	921	30000	0.4	LinearRegression
1.001	921	20000	0.6	LinearRegression
1.002	838	25000	0.7	LinearRegression
1.002	961	15000	0.6	LinearRegression
1.003	775	25000	0.8	LinearRegression
1.003	822	20000	0.8	LinearRegression
1.003	870	30000	0.5	LinearRegression
1.003	885	20000	0.7	LinearRegression
1.004	826	30000	0.6	LinearRegression
1.004	925	15000	0.7	LinearRegression
1.005	727	30000	0.8	LinearRegression
1.005	862	15000	0.8	LinearRegression
1.006	790	30000	0.7	LinearRegression
1.027	775	25000	0.8	EqualBins
1.027	822	20000	0.8	EqualBins
1.027	862	15000	0.8	EqualBins
1.029	885	20000	0.7	EqualBins
1.030	874	25000	0.6	EqualBins
1.030	921	20000	0.6	EqualBins
1.031	838	25000	0.7	EqualBins
1.031	961	15000	0.6	EqualBins
1.032	727	30000	0.8	EqualBins
1.032	925	15000	0.7	EqualBins
1.033	790	30000	0.7	EqualBins
1.034	826	30000	0.6	EqualBins
1.037	965	20000	0.5	EqualBins
1.038	1005	15000	0.5	EqualBins
1.039	918	25000	0.5	EqualBins
1.042	1016	20000	0.4	EqualBins
1.043	1056	15000	0.4	EqualBins
1.043	870	30000	0.5	EqualBins
1.044	969	25000	0.4	EqualBins
1.047	1120	15000	0.3	EqualBins
1.047	1123	15000	0.1	EqualBins
1.048	1041	25000	0.0	EqualBins

RMSE	PR	mTOP	corrlim	predictionMethod
1.048	1080	20000	0.3	EqualBins
1.048	1082	20000	0.2	EqualBins
1.048	1083	20000	0.1	EqualBins
1.048	1088	20000	0.0	EqualBins
1.048	1122	15000	0.2	EqualBins
1.048	921	30000	0.4	EqualBins
1.049	1033	25000	0.3	EqualBins
1.049	1036	25000	0.1	EqualBins
1.049	1128	15000	0.0	EqualBins
1.050	1035	25000	0.2	EqualBins
1.051	987	30000	0.2	EqualBins
1.052	985	30000	0.3	EqualBins
1.053	988	30000	0.1	EqualBins
1.054	993	30000	0.0	EqualBins

A.4 Naive Bayes

RMSE	PR	threshold
1.023	726	0.4
1.033	76	0.7
1.034	99	0.6
1.035	174	0.5
1.040	1659	0.1
1.040	1659	0.2
1.041	1659	0.3

A.5 Correlation

RMSE	PR	corrlim	predictionMethod
0.993	585	0.0	ClosestNeighbor
0.994	577	0.3	ClosestNeighbor
0.994	579	0.2	ClosestNeighbor
0.995	580	0.1	ClosestNeighbor
0.999	513	0.4	ClosestNeighbor

RMSE	PR	corrlim	predictionMethod
1.004	462	0.5	ClosestNeighbor
1.007	418	0.6	ClosestNeighbor
1.008	580	0.1	LinearRegression
1.009	585	0.0	LinearRegression
1.010	577	0.3	LinearRegression
1.010	579	0.2	LinearRegression
1.012	382	0.7	ClosestNeighbor
1.013	319	0.8	ClosestNeighbor
1.015	513	0.4	LinearRegression
1.017	244	0.9	ClosestNeighbor
1.017	462	0.5	LinearRegression
1.020	319	0.8	LinearRegression
1.020	418	0.6	LinearRegression
1.022	382	0.7	LinearRegression
1.025	244	0.9	LinearRegression
1.047	244	0.9	EqualBins
1.051	319	0.8	EqualBins
1.056	418	0.6	EqualBins
1.057	382	0.7	EqualBins
1.065	462	0.5	EqualBins
1.069	513	0.4	EqualBins
1.075	580	0.1	EqualBins
1.076	577	0.3	EqualBins
1.076	579	0.2	EqualBins
1.078	585	0.0	EqualBins

A.6 Cluster

RMSE	PR	clusterer	clusterDataset	distFunc
1.034	1659	EM	PageTimesMouse	None
1.040	1659	XMeans	PageTimesMouse	Manhattan
1.040	1659	Cobweb	TimeOnPageAndMouse	None
1.040	1659	EM	TimeOnPage	None
1.040	1659	Cobweb	PageTimesMouse	None
1.041	1659	Cobweb	TimeOnPage	None

RMSE	PR	clusterer	clusterDataset	distFunc
1.042	1659	FarthestFirst	TimeOnPageAndMouse	None
1.042	1659	SimpleKMeans	PageTimesMouse	Euclidean
1.042	1659	FarthestFirst	PageTimesMouse	None
1.042	1659	DensityBased	PageTimesMouse	Euclidean
1.043	1659	EM	TimeOnPageAndMouse	None
1.045	1659	XMeans	PageTimesMouse	Euclidean
1.045	1659	XMeans	PageTimesMouse	Chebyshev
1.045	1659	XMeans	TimeOnPageAndMouse	Manhattan
1.046	1659	XMeans	TimeOnPageAndMouse	Euclidean
1.046	1659	XMeans	TimeOnPageAndMouse	Chebyshev
1.049	1659	SimpleKMeans	PageTimesMouse	Manhattan
1.049	1659	SimpleKMeans	TimeOnPageAndMouse	Euclidean
1.049	1659	DensityBased	TimeOnPageAndMouse	Euclidean
1.051	1659	FarthestFirst	TimeOnPage	None
1.054	1659	DensityBased	TimeOnPage	Manhattan
1.054	1659	SimpleKMeans	TimeOnPageAndMouse	Manhattan
1.055	1659	DensityBased	PageTimesMouse	Manhattan
1.055	1659	SimpleKMeans	TimeOnPage	Manhattan
1.055	1659	DensityBased	TimeOnPageAndMouse	Manhattan
1.059	1659	XMeans	TimeOnPage	Euclidean
1.059	1659	XMeans	TimeOnPage	Manhattan
1.060	1659	SimpleKMeans	TimeOnPage	Euclidean
1.060	1659	XMeans	TimeOnPage	Chebyshev
1.061	1659	DensityBased	TimeOnPage	Euclidean

A.7 K-Nearest Neighbor

RMSE	PR	threshold	K	minimization
1.029	24	0.8	7	MinimizeMeanAbsoluteError
1.029	980	0.4	15	MinimizeMeanAbsoluteError
1.029	980	0.4	15	MinimizeMeanSquaredError
1.030	1	0.8	15	MinimizeMeanAbsoluteError
1.030	1030	0.4	19	MinimizeMeanSquaredError
1.030	1130	0.4	17	MinimizeMeanSquaredError
1.030	34	0.8	5	MinimizeMeanAbsoluteError

RMSE	PR	threshold	K	minimization
1.030	34	0.8	5	MinimizeMeanSquaredError
1.030	62	0.6	19	MinimizeMeanSquaredError
1.031	0	0.8	17	MinimizeMeanSquaredError
1.031	1	0.8	13	MinimizeMeanAbsoluteError
1.031	1	0.8	13	MinimizeMeanSquaredError
1.031	1011	0.4	11	MinimizeMeanAbsoluteError
1.031	1030	0.4	19	MinimizeMeanAbsoluteError
1.031	1174	0.4	5	MinimizeMeanSquaredError
1.031	156	0.8	3	MinimizeMeanAbsoluteError
1.031	156	0.8	3	MinimizeMeanSquaredError
1.031	21	0.7	17	MinimizeMeanSquaredError
1.031	24	0.8	7	MinimizeMeanSquaredError
1.031	4	0.7	19	MinimizeMeanSquaredError
1.031	62	0.6	19	MinimizeMeanAbsoluteError
1.032	0	0.8	19	MinimizeMeanAbsoluteError
1.032	0	0.8	19	MinimizeMeanSquaredError
1.032	1	0.8	15	MinimizeMeanSquaredError
1.032	1130	0.4	17	MinimizeMeanAbsoluteError
1.032	1174	0.4	5	MinimizeMeanAbsoluteError
1.032	159	0.7	3	MinimizeMeanAbsoluteError
1.032	21	0.7	17	MinimizeMeanAbsoluteError
1.032	4	0.7	19	MinimizeMeanAbsoluteError
1.032	587	0.5	9	MinimizeMeanSquaredError
1.032	62	0.7	11	MinimizeMeanAbsoluteError
1.032	889	0.4	13	MinimizeMeanAbsoluteError
1.032	889	0.4	13	MinimizeMeanSquaredError
1.033	0	0.8	17	MinimizeMeanAbsoluteError
1.033	1011	0.4	11	MinimizeMeanSquaredError
1.033	1492	0.3	17	MinimizeMeanSquaredError
1.033	15	0.7	13	MinimizeMeanAbsoluteError
1.033	15	0.7	13	MinimizeMeanSquaredError
1.033	159	0.7	3	MinimizeMeanSquaredError
1.033	1616	0.3	19	MinimizeMeanAbsoluteError
1.033	19	0.7	15	MinimizeMeanAbsoluteError
1.033	19	0.7	15	MinimizeMeanSquaredError

RMSE	PR	threshold	K	minimization
1.033	37	0.7	9	MinimizeMeanAbsoluteError
1.033	5	0.8	11	MinimizeMeanAbsoluteError
1.033	66	0.6	17	MinimizeMeanAbsoluteError
1.034	1492	0.3	17	MinimizeMeanAbsoluteError
1.034	1579	0.3	15	MinimizeMeanAbsoluteError
1.034	1579	0.3	15	MinimizeMeanSquaredError
1.034	1616	0.3	19	MinimizeMeanSquaredError
1.034	1645	0.3	13	MinimizeMeanSquaredError
1.034	195	0.7	7	MinimizeMeanAbsoluteError
1.034	195	0.7	7	MinimizeMeanSquaredError
1.034	198	0.6	7	MinimizeMeanSquaredError
1.034	37	0.7	9	MinimizeMeanSquaredError
1.034	587	0.5	9	MinimizeMeanAbsoluteError
1.034	62	0.7	11	MinimizeMeanSquaredError
1.034	66	0.6	17	MinimizeMeanSquaredError
1.034	678	0.5	7	MinimizeMeanAbsoluteError
1.034	678	0.5	7	MinimizeMeanSquaredError
1.034	7	0.8	9	MinimizeMeanAbsoluteError
1.034	7	0.8	9	MinimizeMeanSquaredError
1.035	1541	0.3	11	MinimizeMeanAbsoluteError
1.035	1541	0.3	11	MinimizeMeanSquaredError
1.035	1645	0.3	13	MinimizeMeanAbsoluteError
1.035	198	0.6	7	MinimizeMeanAbsoluteError
1.035	440	0.5	15	MinimizeMeanSquaredError
1.035	5	0.8	11	MinimizeMeanSquaredError
1.035	799	0.5	5	MinimizeMeanAbsoluteError
1.035	799	0.5	5	MinimizeMeanSquaredError
1.036	1216	0.4	9	MinimizeMeanAbsoluteError
1.036	1216	0.4	9	MinimizeMeanSquaredError
1.036	167	0.7	5	MinimizeMeanSquaredError
1.036	200	0.6	13	MinimizeMeanAbsoluteError
1.036	312	0.6	5	MinimizeMeanAbsoluteError
1.036	312	0.6	5	MinimizeMeanSquaredError
1.036	440	0.5	15	MinimizeMeanAbsoluteError
1.036	512	0.5	11	MinimizeMeanAbsoluteError

RMSE	**PR**	**threshold**	**K**	**minimization**
1.036	512	0.5	11	MinimizeMeanSquaredError
1.037	1445	0.3	7	MinimizeMeanAbsoluteError
1.037	1635	0.3	9	MinimizeMeanAbsoluteError
1.037	1635	0.3	9	MinimizeMeanSquaredError
1.037	167	0.7	5	MinimizeMeanAbsoluteError
1.037	200	0.6	13	MinimizeMeanSquaredError
1.037	213	0.6	9	MinimizeMeanAbsoluteError
1.037	364	0.5	19	MinimizeMeanAbsoluteError
1.038	1641	0.3	5	MinimizeMeanAbsoluteError
1.038	182	0.6	11	MinimizeMeanAbsoluteError
1.038	182	0.6	11	MinimizeMeanSquaredError
1.038	213	0.6	9	MinimizeMeanSquaredError
1.039	1444	0.4	7	MinimizeMeanAbsoluteError
1.039	1445	0.3	7	MinimizeMeanSquaredError
1.039	364	0.5	19	MinimizeMeanSquaredError
1.039	472	0.5	13	MinimizeMeanSquaredError
1.040	1034	0.6	3	MinimizeMeanAbsoluteError
1.040	1034	0.6	3	MinimizeMeanSquaredError
1.040	116	0.6	15	MinimizeMeanSquaredError
1.040	1444	0.4	7	MinimizeMeanSquaredError
1.040	1641	0.3	5	MinimizeMeanSquaredError
1.040	392	0.5	17	MinimizeMeanAbsoluteError
1.040	392	0.5	17	MinimizeMeanSquaredError
1.041	116	0.6	15	MinimizeMeanAbsoluteError
1.041	472	0.5	13	MinimizeMeanAbsoluteError
1.042	1034	0.5	3	MinimizeMeanAbsoluteError
1.042	1034	0.5	3	MinimizeMeanSquaredError
1.042	1039	0.4	3	MinimizeMeanAbsoluteError
1.043	1039	0.4	3	MinimizeMeanSquaredError
1.043	1659	0.3	3	MinimizeMeanAbsoluteError
1.043	1659	0.3	3	MinimizeMeanSquaredError
1.046	1659	0.3	1	MinimizeMeanSquaredError
1.047	1651	0.8	1	MinimizeMeanAbsoluteError
1.047	1651	0.7	1	MinimizeMeanSquaredError
1.047	1653	0.6	1	MinimizeMeanAbsoluteError

RMSE	PR	threshold	K	minimization
1.047	1653	0.6	1	MinimizeMeanSquaredError
1.047	1653	0.5	1	MinimizeMeanSquaredError
1.048	1651	0.7	1	MinimizeMeanAbsoluteError
1.048	1651	0.8	1	MinimizeMeanSquaredError
1.048	1653	0.5	1	MinimizeMeanAbsoluteError
1.048	1659	0.4	1	MinimizeMeanAbsoluteError
1.048	1659	0.3	1	MinimizeMeanAbsoluteError
1.048	1659	0.4	1	MinimizeMeanSquaredError

A.8 Artificial Neural Network

RMSE	PR	learningRate	epochs
1.031	1659	0.04100	310
1.031	1659	0.04100	510
1.032	1659	0.02100	310
1.032	1659	0.08100	910
1.032	1659	0.02100	410
1.032	1659	0.02100	210
1.032	1659	0.08100	810
1.032	1659	0.02100	110
1.032	1659	0.02100	710
1.032	1659	0.02100	610
1.032	1659	0.04100	610
1.032	1659	0.06100	510
1.032	1659	0.04100	710
1.032	1659	0.06100	810
1.032	1659	0.06100	710
1.033	1659	0.08100	710
1.033	1659	0.04100	410
1.033	1659	0.12100	210
1.033	1659	0.08100	310
1.033	1659	0.06100	310
1.033	1659	0.08100	410
1.033	1659	0.06100	210
1.033	1659	0.12100	410

RMSE	PR	learningRate	epochs
1.033	1659	0.06100	410
1.033	1659	0.10100	210
1.033	1659	0.04100	210
1.033	1659	0.06100	910
1.033	1659	0.02100	510
1.033	1659	0.10100	110
1.033	1659	0.04100	110
1.033	1659	0.08100	610
1.033	1659	0.04100	910
1.033	1659	0.06100	610
1.033	1659	0.08100	510
1.033	1659	0.10100	710
1.033	1659	0.02100	810
1.033	1659	0.02100	910
1.033	1659	0.04100	810
1.034	1659	0.14100	410
1.034	1659	0.10100	810
1.034	1659	0.08100	210
1.034	1659	0.10100	310
1.034	1659	0.14100	210
1.034	1659	0.12100	910
1.034	1659	0.10100	910
1.034	1659	0.14100	810
1.034	1659	0.20100	310
1.034	1659	0.16100	810
1.034	1659	0.12100	510
1.034	1659	0.18100	110
1.034	1659	0.00100	910
1.034	1659	0.14100	610
1.034	1659	0.14100	110
1.034	1659	0.08100	110
1.034	1659	0.06100	110
1.034	1659	0.10100	510
1.034	1659	0.16100	610
1.034	1659	0.18100	610

RMSE	PR	learningRate	epochs
1.034	1659	0.12100	810
1.034	1659	0.12100	710
1.034	1659	0.20100	610
1.034	1659	0.10100	610
1.035	1659	0.14100	910
1.035	1659	0.16100	310
1.035	1659	0.16100	210
1.035	1659	0.14100	310
1.035	1659	0.18100	210
1.035	1659	0.14100	710
1.035	1659	0.12100	310
1.035	1659	0.16100	910
1.035	1659	0.10100	410
1.035	1659	0.18100	310
1.035	1659	0.22100	110
1.035	1659	0.16100	410
1.035	1659	0.00100	810
1.035	1659	0.20100	810
1.035	1659	0.00100	610
1.035	1659	0.16100	110
1.035	1659	0.12100	110
1.035	1659	0.04100	10
1.035	1659	0.20100	110
1.035	1659	0.20100	410
1.035	1659	0.16100	510
1.035	1659	0.22100	810
1.035	1659	0.20100	510
1.035	1659	0.22100	610
1.035	1659	0.18100	710
1.035	1659	0.18100	510
1.035	1659	0.00100	710
1.035	1659	0.18100	410
1.035	1659	0.12100	610
1.035	1659	0.16100	710
1.036	1659	0.22100	710

RMSE	**PR**	**learningRate**	**epochs**
1.036	1659	0.24100	410
1.036	1659	0.20100	210
1.036	1659	0.18100	810
1.036	1659	0.24100	610
1.036	1659	0.24100	310
1.036	1659	0.18100	910
1.036	1659	0.20100	910
1.036	1659	0.22100	310
1.036	1659	0.22100	910
1.036	1659	0.22100	210
1.036	1659	0.00100	410
1.036	1659	0.24100	710
1.036	1659	0.22100	510
1.036	1659	0.20100	710
1.036	1659	0.22100	410
1.036	1659	0.24100	510
1.036	1659	0.14100	510
1.036	1659	0.00100	510
1.036	1659	0.24100	810
1.037	1659	0.26100	210
1.037	1659	0.24100	910
1.037	1659	0.00100	310
1.037	1659	0.24100	210
1.037	1659	0.02100	10
1.037	1659	0.08100	10
1.037	1659	0.06100	10
1.037	1659	0.26100	710
1.037	1659	0.12100	10
1.037	1659	0.24100	110
1.037	1659	0.26100	410
1.037	1659	0.26100	510
1.038	1659	0.26100	610
1.038	1659	0.26100	110
1.038	1659	0.00100	210
1.038	1659	0.26100	310

RMSE	PR	learningRate	epochs
1.038	1659	0.26100	910
1.038	1659	0.14100	10
1.038	1659	0.00100	110
1.038	1659	0.10100	10
1.038	1659	0.26100	810
1.039	1659	0.28100	110
1.039	1659	0.28100	710
1.040	1659	0.28100	310
1.040	1659	0.28100	910
1.040	1659	0.28100	210
1.040	1659	0.20100	10
1.040	1659	0.16100	10
1.040	1659	0.28100	410
1.040	1659	0.28100	610
1.040	1659	0.28100	810
1.040	1659	0.28100	510
1.041	1659	0.18100	10
1.042	1659	0.24100	10
1.042	1659	0.22100	10
1.044	1659	0.26100	10
1.047	1659	0.28100	10
1.052	1659	0.00100	10

A.9 Support Vector Machine

RMSE	PR	C	kernelGamma
1.029	1659	128.00000	32.00000
1.031	1659	32.00000	1.00000
1.031	1659	256.00000	0.25000
1.031	1659	32.00000	32.00000
1.031	1659	1024.00000	0.25000
1.031	1659	0.25000	8.00000
1.031	1659	0.50000	1.00000
1.031	1659	2.00000	64.00000
1.031	1659	8.00000	4.00000

RMSE	**PR**	**C**	**kernelGamma**
1.031	1659	32.00000	2.00000
1.031	1659	0.00781	16.00000
1.031	1659	64.00000	2.00000
1.031	1659	512.00000	0.25000
1.031	1659	64.00000	4.00000
1.031	1659	32.00000	64.00000
1.031	1659	64.00000	0.50000
1.031	1659	8.00000	64.00000
1.032	1659	128.00000	2.00000
1.032	1659	16.00000	4.00000
1.032	1659	128.00000	0.25000
1.032	1659	0.25000	4.00000
1.032	1659	64.00000	64.00000
1.032	1659	0.06250	4.00000
1.032	1659	0.01563	32.00000
1.032	1659	1.00000	2.00000
1.032	1659	0.25000	32.00000
1.032	1659	64.00000	32.00000
1.032	1659	0.06250	8.00000
1.032	1659	0.06250	32.00000
1.032	1659	16.00000	0.50000
1.032	1659	32.00000	4.00000
1.032	1659	0.00391	32.00000
1.032	1659	0.25000	16.00000
1.032	1659	1024.00000	1.00000
1.032	1659	8.00000	1.00000
1.032	1659	0.50000	16.00000
1.032	1659	32.00000	0.50000
1.032	1659	16.00000	32.00000
1.032	1659	128.00000	64.00000
1.032	1659	0.01563	16.00000
1.032	1659	0.03125	16.00000
1.032	1659	0.03125	8.00000
1.032	1659	256.00000	32.00000
1.032	1659	1.00000	16.00000

RMSE	**PR**	**C**	**kernelGamma**
1.032	1659	0.00391	64.00000
1.032	1659	512.00000	32.00000
1.032	1659	16.00000	64.00000
1.032	1659	0.12500	8.00000
1.032	1659	256.00000	2.00000
1.032	1659	0.50000	8.00000
1.032	1659	8.00000	0.50000
1.032	1659	2.00000	2.00000
1.032	1659	2.00000	4.00000
1.032	1659	256.00000	0.50000
1.033	1659	1.00000	64.00000
1.033	1659	0.12500	4.00000
1.033	1659	0.50000	64.00000
1.033	1659	8.00000	2.00000
1.033	1659	64.00000	16.00000
1.033	1659	0.06250	64.00000
1.033	1659	128.00000	16.00000
1.033	1659	0.01563	8.00000
1.033	1659	0.03125	4.00000
1.033	1659	4.00000	16.00000
1.033	1659	0.25000	64.00000
1.033	1659	0.03125	64.00000
1.033	1659	0.06250	16.00000
1.033	1659	8.00000	32.00000
1.033	1659	512.00000	0.50000
1.033	1659	0.12500	64.00000
1.033	1659	0.03125	32.00000
1.033	1659	0.00781	64.00000
1.033	1659	128.00000	0.50000
1.033	1659	16.00000	1.00000
1.033	1659	2.00000	32.00000
1.033	1659	16.00000	16.00000
1.033	1659	4.00000	8.00000
1.033	1659	2.00000	8.00000
1.033	1659	0.00781	32.00000

RMSE	**PR**	**C**	**kernelGamma**
1.033	1659	0.12500	16.00000
1.033	1659	8.00000	16.00000
1.033	1659	8.00000	8.00000
1.033	1659	1.00000	4.00000
1.033	1659	2.00000	16.00000
1.033	1659	0.50000	32.00000
1.033	1659	4.00000	1.00000
1.033	1659	4.00000	4.00000
1.033	1659	0.50000	4.00000
1.033	1659	4.00000	64.00000
1.033	1659	2.00000	0.50000
1.033	1659	1.00000	8.00000
1.034	1659	0.12500	32.00000
1.034	1659	16.00000	2.00000
1.034	1659	32.00000	8.00000
1.034	1659	256.00000	4.00000
1.034	1659	0.12500	2.00000
1.034	1659	16.00000	8.00000
1.034	1659	32.00000	16.00000
1.034	1659	512.00000	64.00000
1.034	1659	0.01563	64.00000
1.034	1659	512.00000	2.00000
1.034	1659	4.00000	32.00000
1.034	1659	4.00000	2.00000
1.034	1659	256.00000	64.00000
1.034	1659	0.50000	2.00000
1.034	1659	128.00000	4.00000
1.034	1659	0.06250	2.00000
1.034	1659	1024.00000	32.00000
1.034	1659	1.00000	32.00000
1.034	1659	1.00000	1.00000
1.034	1659	4.00000	0.50000
1.035	1659	0.00781	8.00000
1.035	1659	0.25000	1.00000
1.035	1659	0.25000	2.00000

RMSE	**PR**	**C**	**kernelGamma**
1.035	1659	512.00000	16.00000
1.035	1659	2.00000	1.00000
1.036	1659	0.00195	64.00000
1.036	1659	0.00391	16.00000
1.036	1659	64.00000	8.00000
1.036	1659	256.00000	16.00000
1.036	1659	0.01563	4.00000
1.036	1659	64.00000	0.25000
1.037	1659	16.00000	0.25000
1.038	1659	512.00000	1.00000
1.038	1659	32.00000	0.25000
1.039	1659	0.01563	0.01563
1.039	1659	0.12500	0.25000
1.039	1659	0.00781	0.25000
1.039	1659	0.00391	0.25000
1.039	1659	0.06250	0.06250
1.039	1659	1.00000	0.06250
1.039	1659	0.00391	8.00000
1.039	1659	32.00000	0.03125
1.039	1659	0.00195	0.50000
1.039	1659	0.00781	0.01563
1.039	1659	0.01563	0.06250
1.039	1659	64.00000	1.00000
1.039	1659	8.00000	0.03125
1.039	1659	8.00000	0.12500
1.039	1659	1.00000	0.50000
1.039	1659	0.00098	1.00000
1.039	1659	0.00098	2.00000
1.039	1659	4.00000	0.01563
1.039	1659	1024.00000	16.00000
1.040	1659	0.00098	0.01563
1.040	1659	0.00781	0.03125
1.040	1659	0.00391	0.12500
1.040	1659	0.01563	0.12500
1.040	1659	0.00391	4.00000

RMSE	PR	C	kernelGamma
1.040	1659	0.50000	0.03125
1.040	1659	0.03125	0.03125
1.040	1659	0.00781	4.00000
1.040	1659	0.12500	1.00000
1.040	1659	0.12500	0.12500
1.040	1659	2.00000	0.03125
1.040	1659	0.12500	0.50000
1.040	1659	16.00000	0.06250
1.040	1659	0.06250	0.12500
1.040	1659	0.50000	0.12500
1.040	1659	0.00195	8.00000
1.040	1659	0.12500	0.01563
1.040	1659	0.03125	0.06250
1.040	1659	0.50000	0.25000
1.040	1659	0.00391	0.06250
1.040	1659	0.06250	0.25000
1.040	1659	0.00195	0.25000
1.040	1659	0.03125	0.25000
1.040	1659	128.00000	8.00000
1.040	1659	0.00098	4.00000
1.040	1659	0.00195	2.00000
1.040	1659	0.25000	0.25000
1.040	1659	0.00098	16.00000
1.040	1659	0.03125	0.50000
1.040	1659	0.00781	0.06250
1.040	1659	0.00098	0.03125
1.040	1659	0.00391	2.00000
1.040	1659	256.00000	0.01563
1.040	1659	0.12500	0.03125
1.040	1659	16.00000	0.03125
1.040	1659	4.00000	0.12500
1.040	1659	1.00000	0.01563
1.040	1659	0.00098	32.00000
1.040	1659	32.00000	0.01563
1.040	1659	2.00000	0.25000

RMSE	**PR**	**C**	**kernelGamma**
1.040	1659	0.00781	0.50000
1.040	1659	0.00195	16.00000
1.040	1659	4.00000	0.03125
1.040	1659	0.25000	0.12500
1.040	1659	0.25000	0.50000
1.040	1659	8.00000	0.25000
1.040	1659	1.00000	0.12500
1.040	1659	0.00195	0.12500
1.040	1659	0.00098	0.50000
1.040	1659	0.00781	1.00000
1.040	1659	0.00391	0.50000
1.040	1659	2.00000	0.12500
1.040	1659	0.00195	0.01563
1.040	1659	0.50000	0.50000
1.040	1659	0.00098	0.25000
1.040	1659	0.00195	0.03125
1.040	1659	0.00098	0.06250
1.040	1659	16.00000	0.01563
1.040	1659	0.00195	1.00000
1.040	1659	0.01563	2.00000
1.041	1659	0.25000	0.01563
1.041	1659	0.01563	1.00000
1.041	1659	64.00000	0.03125
1.041	1659	0.00391	0.03125
1.041	1659	0.03125	0.12500
1.041	1659	0.06250	0.03125
1.041	1659	0.06250	0.01563
1.041	1659	0.03125	0.01563
1.041	1659	0.12500	0.06250
1.041	1659	0.06250	0.50000
1.041	1659	0.50000	0.06250
1.041	1659	16.00000	0.12500
1.041	1659	0.06250	1.00000
1.041	1659	128.00000	0.01563
1.041	1659	0.50000	0.01563

RMSE	**PR**	**C**	**kernelGamma**
1.041	1659	4.00000	0.25000
1.041	1659	0.00098	64.00000
1.041	1659	64.00000	0.01563
1.041	1659	0.01563	0.25000
1.041	1659	0.00391	0.01563
1.041	1659	0.00391	1.00000
1.041	1659	0.00781	0.12500
1.041	1659	0.00781	2.00000
1.041	1659	2.00000	0.01563
1.041	1659	0.01563	0.50000
1.041	1659	1.00000	0.03125
1.041	1659	0.25000	0.06250
1.041	1659	0.00195	32.00000
1.041	1659	512.00000	0.01563
1.041	1659	2.00000	0.06250
1.041	1659	0.25000	0.03125
1.041	1659	8.00000	0.01563
1.041	1659	0.00195	4.00000
1.041	1659	0.00098	8.00000
1.041	1659	0.03125	1.00000
1.041	1659	4.00000	0.06250
1.041	1659	32.00000	0.06250
1.041	1659	0.00195	0.06250
1.041	1659	1.00000	0.25000
1.041	1659	8.00000	0.06250
1.042	1659	1024.00000	0.12500
1.042	1659	128.00000	0.03125
1.042	1659	0.01563	0.03125
1.042	1659	1024.00000	0.50000
1.042	1659	0.00098	0.12500
1.042	1659	0.03125	2.00000
1.043	1659	64.00000	0.06250
1.043	1659	512.00000	4.00000
1.044	1659	256.00000	8.00000
1.044	1659	1024.00000	64.00000

RMSE	PR	C	kernelGamma
1.045	1659	1024.00000	2.00000
1.045	1659	32.00000	0.12500
1.047	1659	256.00000	0.03125
1.049	1659	128.00000	1.00000
1.049	1659	512.00000	8.00000
1.050	1659	1024.00000	0.01563
1.052	1659	256.00000	1.00000
1.055	1659	128.00000	0.06250
1.055	1659	64.00000	0.12500
1.065	1659	1024.00000	8.00000
1.071	1659	512.00000	0.12500
1.073	1659	128.00000	0.12500
1.074	1659	1024.00000	4.00000
1.079	1659	256.00000	0.06250
1.086	1659	256.00000	0.12500
1.098	1659	512.00000	0.03125
1.209	1659	1024.00000	0.03125
1.239	1659	512.00000	0.06250
NaN	1659	1024.00000	0.06250

Bibliography

[1] K. C. Almeroth J. E. Ingvaldsen G. Nygreen M. Tavakolifard, J. A. Gulla and E. Berg. Tailored news in the palm of your hand: A multi-perspective transparent approach to news recommendation. 2013.

[2] A. D. Fidjestøl J. E. Nilsen K. R. Haugen X. Su J. A. Gulla, J. E. Ingvaldsen. Learning user profiles in mobile news recommendation. *Journal of Print and Media Technology Research*, II(3):183–194, 2013.

[3] J Ben Schafer, Joseph Konstan, and John Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 158–166. ACM, 1999.

[4] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *Internet Computing, IEEE*, 7(1):76–80, 2003.

[5] James Bennett and Stan Lanning. The netflix prize. In *Proceedings of KDD cup and workshop*, volume 2007, page 35, 2007.

[6] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th international conference on World Wide Web*, pages 271–280. ACM, 2007.

[7] Yehuda Koren. The bellkor solution to the netflix grand prize. *Netflix prize documentation*, 2009.

[8] Dan Cosley, Shyong K Lam, Istvan Albert, Joseph A Konstan, and John Riedl. Is seeing believing?: how recommender system interfaces affect users' opinions. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 585–592. ACM, 2003.

[9] Xavier Amatriain, Josep M Pujol, and Nuria Oliver. I like it... i like it not: Evaluating user ratings noise in recommender systems. In *User Modeling, Adaptation, and Personalization*, pages 247–258. Springer, 2009.

[10] N. Thurman. Making 'the daily me': Technology, economics and habit in the mainstream assimilation of personalised news. *Journalism: Theory, Practice & Criticism*, pages 395–415, 2011.

[11] Diane Kelly and Jaime Teevan. Implicit feedback for inferring user preference: a bibliography. In *ACM SIGIR Forum*, volume 37, pages 18–28. ACM, 2003.

[12] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*, pages 263–272. IEEE, 2008.

[13] Denis Parra and Xavier Amatriain. Walk the talk: Analyzing the relation between implicit and explicit feedback for preference elicitation. In *Proceedings of the 19th International Conference on User Modeling, Adaption, and Personalization*, UMAP'11, pages 255–268, Berlin, Heidelberg, 2011. Springer-Verlag. ISBN 978-3-642-22361-7. URL http://dl.acm.org/citation.cfm?id=2021855.2021878.

[14] Masahiro Morita and Yoichi Shinoda. Information filtering based on user behavior analysis and best match text retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 272–281. Springer-Verlag New York, Inc., 1994.

[15] Mark Claypool, Phong Le, Makoto Wased, and David Brown. Implicit interest indicators. In *Proceedings of the 6th international conference on Intelligent user interfaces*, pages 33–40. ACM, 2001.

[16] Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009.

[17] Kirk Baker. Singular value decomposition tutorial. 2005.

[18] Yehuda Koren. Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 4 (1):1, 2010.

[19] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[20] Faraz Makari, Christina Teflioudi, Rainer Gemulla, Peter Haas, and Yannis Sismanis. Shared-memory and shared-nothing stochastic gradient descent algorithms for matrix completion. *Knowledge and Information Systems*, pages 1–31, 2014. ISSN 0219-1377. doi: 10.1007/s10115-013-0718-7. URL http://dx.doi.org/10.1007/s10115-013-0718-7.

[21] Nguyen Thai-Nghe, Lucas Drumond, Tomás Horváth, Alexandros Nanopoulos, and Lars Schmidt-Thieme. Matrix and tensor factorization for predicting student performance. In *CSEDU (1)*, pages 69–78. Citeseer, 2011.

[22] Balázs Hidasi and Domonkos Tikk. Fast als-based tensor factorization for context-aware recommendation from implicit feedback. In *Machine Learning and Knowledge Discovery in Databases*, pages 67–82. Springer, 2012.

[23] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[24] Harry Zhang. The optimality of naive bayes. *A A*, 1(2):3, 2004.

[25] George H John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 338–345. Morgan Kaufmann Publishers Inc., 1995.

[26] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, 1967.

[27] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4): 115–133, 1943.

[28] Victor Hugo C de Albuquerque, Auzuir Ripardo de Alexandria, Paulo César Cortez, and João Manuel RS Tavares. Evaluation of multilayer perceptron and self-organizing map neural network topologies applied on microstructure segmentation from metallographic images. *NDT & E International*, 42(7): 644–651, 2009.

[29] Martin Riedmiller. Advanced supervised learning in multi-layer perceptrons—from backpropagation to adaptive learning algorithms. *Computer Standards & Interfaces*, 16(3):265–278, 1994.

[30] Tobias Ebert, Oliver Bänfer, and Oliver Nelles. Multilayer perceptron network with modified sigmoid activation functions. In *Artificial Intelligence and Computational Intelligence*, pages 414–421. Springer, 2010.

[31] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[32] J. MacQueen. Some methods for classification and analysis of multivariate observations, 1967. URL http://projecteuclid.org/euclid.bsmsp/1200512992.

[33] Dan Pelleg and Andrew W. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *Seventeenth International Conference on Machine Learning*, pages 727–734. Morgan Kaufmann, 2000.

[34] Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press Cambridge.

[35] Douglas H Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine learning*, 2(2):139–172, 1987.

[36] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 79–86. ACM, 2010.

[37] Yi Zhang. Bayesian graphical models for adaptive filtering. In *SIGIR Forum*, volume 39, page 57, 2005.

[38] Gábor Takács, István Pilászy, Bottyán Németh, and Domonkos Tikk. Scalable collaborative filtering approaches for large recommender systems. *The Journal of Machine Learning Research*, 10:623–656, 2009.

[39] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. *Advances in Neural Information Processing Systems*, 24:693–701, 2011.