Our awesome title

Thomas Almenningen, Martin Christian Havig and Herman Schistad

May 28, 2014

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

Supervisor: Heri Ramampiaro
Supervisor: Helge Langseth

Acknowledgments

Our deepest gratitude goes to our supervisors at NTNU *Heri Ramampiaro* and *Helge Langseth*, who throughout nearly a full year of research has not only supported us, but with their enthusiasm and knowledge made us understand and appreciate the field of recommender systems.

Equally our sincerest appreciation to our supervisor at Telenor Research *Hai Thanh Nguyen*, who has given us valuable insight and guidance through weekly meetings the last six months. Not only has his expertise in the field of machine learning made our end result better, but his role as coordinator between; the Sobazar headquarters in Oslo; various researchers at Telenor and our supervisors at NTNU have been invaluable, every week.

In addition, we thank *Anders Kofod-Petersen* who has given his constructive and useful insights on a range of topics spanning from the layout and focus of our thesis to inner workings of algorithms and metrics. We also thank *Humberto Castejon* and *Cyril Banino-Rokkones* for their support and ideas when initiating this project. Valuable input and help was provided by the Sobazar team and especially *Markus Krger* and *Matias Holte* who we had a meaningful and interesting meeting with halfway through. Finally our thanks to *Rana Juwel* who has assisted us with regards to content-based approaches for recommending based on the Sobazar-dataset.

There are many to thank, but in summary we had the luxury of having multiple researchers and supervisors, not only interested in the project, but in helping us throughout these last months. We are grateful for all your support and contributions. Again, thank you.

Abstract

In recommender systems we seek to predict the rating or preference that a user would give to an item. With the advent of modern application such as movie and music streaming, E-commerce, social networks and many other platforms recommender systems have gained an important role in both a commercial and academic setting. When analyzing various approaches to *how* an application generates recommendations, the prevalent technique is explicitly asking the users for a numeric rating for every item they consume and from these observed explicit ratings predict how the user would rate other items. However, this procedure often requires important changes to the application frontend and also an active effort from all users. In this thesis we instead look at how these ratings can be inferred from looking at session data and the transaction history for every user.

We analyze which recommender models are suitable for such an untraditional data source, whilst accounting for a sparse dataset with few events often called a cold start environment. Multiple novel ways of generating ratings are presented taking into account recentness in the data as an important factor. This, combined with traditional recommender system approaches makes for a new and interesting way of making recommendations in small datasets with high sparsity. We then provide valuable insight into using and evaluating implicit ratings in recommender system algorithms, spanning from simple global-averages to state-of-the-art matrix-factorizations. A promising result is finally made, supported by a range of relevant metrics, promoting further research into the area of implicit ratings on sparse and small datasets.

Finally, a discussion is made on the various evaluation metrics for both implicit ratings and our recommender models, as here as well the untraditional approach of choosing implicit ratings as input to our models makes many of the conventional metrics inefficacious.

heri-notes: Hovekonklusjon og hovedfunnene skal med her

Contents

List of Figures

List of Tables

1

Introduction

Contents

1.1 Motivation

In today's day and age the increasing amount of data overwhelm our human processing capabilities in many information seeking tasks. To cope with this overload researchers have introduced recommender system to filter the ever increasing information and only present a small selection of items which reflects the users tastes, interests and priorities. Recommender systems are an active research field and has been successfully applied to many different services ranging from e-commerce sites such as *Amazon*, movie and TV-series streaming services like *Netflix* and in different music applications such as *Last.fm* and *iTunes*.

Many of the largest commerce Web sites have been using recommender systems to help their customers find products to purchase for nearly two decades. Schafer et. al. [92] identified three ways, in which recommender systems increase E-commerce sales: (1) Browser into buyers: Recommender systems can help customers find products they wish to purchase, (2) Cross-sell: Recommender systems improve cross-sell by suggesting additional products for the customer to purchase and (3) Loyalty: In a world where the competitor only is one click away, gaining customer loyalty is an essential customer strategy. Recommender systems improve loyalty by creating a value added relationship between the site and the customer.

SoBazaar is a new fashion e-commerce application for web and hand held devices developed by Telenor, Norway's largest Telco company. The application aggregates fashion products from various brands and stores into one *webstore*. The app is planned to be *officially launched* this summer, meaning that there currently is a limited amount user-item interaction data available, making it a classic cold-start scenario. We have access to data coming from multiple sources including user information from Facebook, rich meta-data description of the items from the retailers as well as information about the users browsing and buying habits collected by the application.

In a classic recommendation scenario one has access to rich explicit information regarding the users preferences, often in the form of ratings. We have a very limited amount of user-item interaction information available and must therefore determine the best way to leverage the implicit information collected by the application in combination with the user combination collected from Facebook and item meta-data to improve the recommendations.

This scenario differs from making recommendations on movies, books and music, not just in the lack of rich explicit ratings, but also in the context of the domain the recommendations will be done, namely the fashion domain. Firstly, fashion consumption is largely determined by seasons. E.g. one does not buy winter jackets in the middle of summer. Some clothes do also go out of fashion, the same can not be said for *all* movies. Secondly, there are a whole different set of important aspects regarding the items or products when recommending in the fashion domain, such as: brand, color and size.

For the average consumer of a movie the producer might not affect greatly the way the consumer views the movie, but when it comes to fashion the consumer might mainly look at the brand of the product when deciding what to consume. The social aspect also affects fashion recommendation on another level than movies. The affiliation of a member of a social group might not be much affected by what movies the member likes or dislikes, but what the individual wears can greatly affect it [105]. How a consumer consumes differs from the named domains, and then again how to recommend will differ. Fashion is often used to show off to fellow peers, and will often produce satisfaction for

the individual showing of. Whereas the satisfaction of a movie or book can be just as great in solitude, rather than in the company of others. This magnifies the importance of the users or consumers social group. This is where Facebook and other social networks can be of great help when recommending fashion products.

heri-notes: Motivating example ville vre nyttig for f leseren til forst bedre tankegangen

1.2 Problem Statement and Goals

heri-notes: overordnet problemstilling

The primary aim of this thesis is to propose a design for a recommender system, that may be used in an e-commerce fashion application. The designed system should be able to generate recommendations based on user-interactions with the applications, and optionally use product information to improve the recommendation quality. In addition, the design should provide a solution to the problem of giving accurate recommendations to new users of the system.

Recommender systems have been applied to a wide range of different domains including music, movies, e-commerce, news and many others. Since each application domain has its own specific needs, the methods used for recommendations differ. This leads us to our first research goals, which are the following:

- G1: Gain a better understanding of the fashion domain.
- G2: Identify the specific challenges of making fashion recommendations.
- G3: How can existing technologies be adapted to mitigate or overcome these challenges?

Most recommender systems base their recommendations of previous feedback given by the user. A central problem for recommender systems is therefore the cold-start problem. How do you recommend items when you have little or no user feedback. SoBazaar is a *brand new* application, and have therefore naturally recorded a limited amount of user-interactions. Poor recommendations can result in customer defection and loss of revenue for Telenor. The above reasoning lead us to the following goal, which is to:

- G4: Find the existing solutions to the cold-start problem presented in the recommender system literature and present the possible solution(s) best suited for our application and needs.

We focus mainly on finding *complete* solutions to the cold-start problem, that can handle both new users, new items and general sparsity related problems.

As previously mentioned, recommender systems base their recommendations on feedback given by the user. SoBazaar records all the users interactions with the application, which then could be interpreted as user feedback given by users to items. We have multiple types of interactions such as e.g. browsing, wanting and purchasing items. We would like to figure out how these events could be used to learn the users true preferences. This lead us to the following goals:

- G5: Explore the existing solutions of how to infer user preference from implicit feedback data.
- G6: Establish user interaction patterns to support our assumptions.

- G7: Find different methods of combining various event types into *implicit ratings*?
- G8: Evaluate the *implicit ratings*.

We would like to combine the solutions found through our work with the above mentioned goals and combine these in a proposal to a design for a fashion e-commerce recommender system.

TODO? Assumptions and Constraints heri-notes: scope, what this thesis is not about, limitations Hai said: Maybe remove the evaluate recommender box? If filterbots makes it to the 'final system', include it in a more detailed system 'drawing'? Extend introduction with ref to the chapters/section the boxes are referring to

1.3 System Overview

The following figure shows an overview of the process from data collection to recommendations.

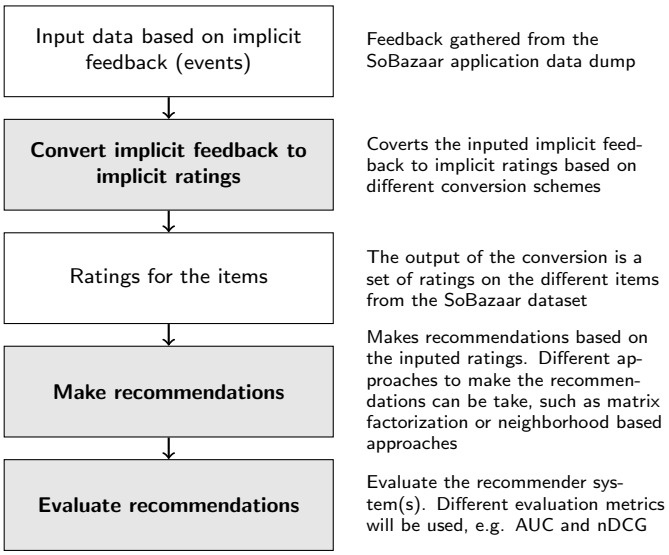


Figure 1.1: Overview of the system. Boxes in white represents input and output data. Boxes in gray represents processes

1.4 Outline

Chapter	Description
Chapter 1	The Introduction chapter gives an overview of the project to the reader. It also outlines the purpose and motivation of the project.
Chapter 2	The Dataset chapter presents the results from our dataset analysis.
Chapter 3	The Preliminary Study chapter documents knowledge, research and technology that is relevant to the project, and how and why some of them were prioritized over others when it comes to how they are used in the project.
Chapter 4	The implementation chapter describes the design of the system and how the design has be implemented.
Chapter 5	Evaluation chapter discussed the development process, testing of results and major issues.
Chapter 6	The Conclusion chapter sums up the project and describes the findings and reflects on them. It also describes further work to be done.
Appendix	The appendix contains extended information such as a full list of the requirements.

Table 1.1: Structure and chapters of the report.

2

The SoBazaar Data

Contents

This sections will introduce the SoBazaar application and explore the SoBazaar data set. This includes how the data was preprocessed, a summary of the dataset, statistics and graphs about the data and findings from the data.

2.1 The SoBazaar Application

> *"SOBAZAAR is a newly developed social shopping experience a daily fashion bazaar on your mobile."*
>
> – About SoBazaar

SoBazaar is an online marketplace for fashion, but with a social twist. The application lets you click through thousands of products. You can find stores like Moods of Norway, BIK BOK, Bianco and many more. If you find something you like you can purchase it directly from the application or store it for later with the *love it* functionality. The following screenshots are taken from the application and highlights some functionality currently found in the application.

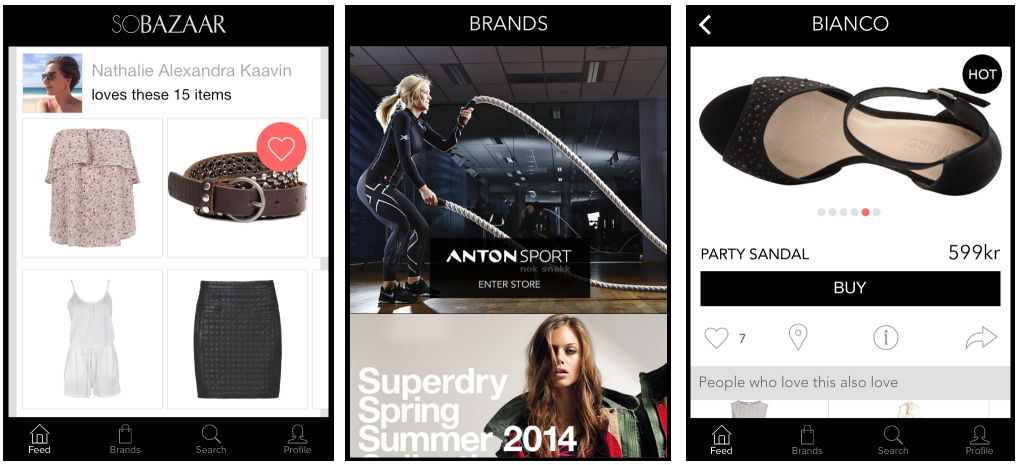


Figure 2.1: Screenshots from the SoBazaar Application. From the left to right: The SoBazaar newsfeed, the brand browser screen and the product detail screen

To help the customers find products the feed currently shows the activity of other users, notifies you about sales, presents the most popular items and the application also features an editors pick page among other things. The following screenshots show some functionality currently in place to help the customer find items to buy.

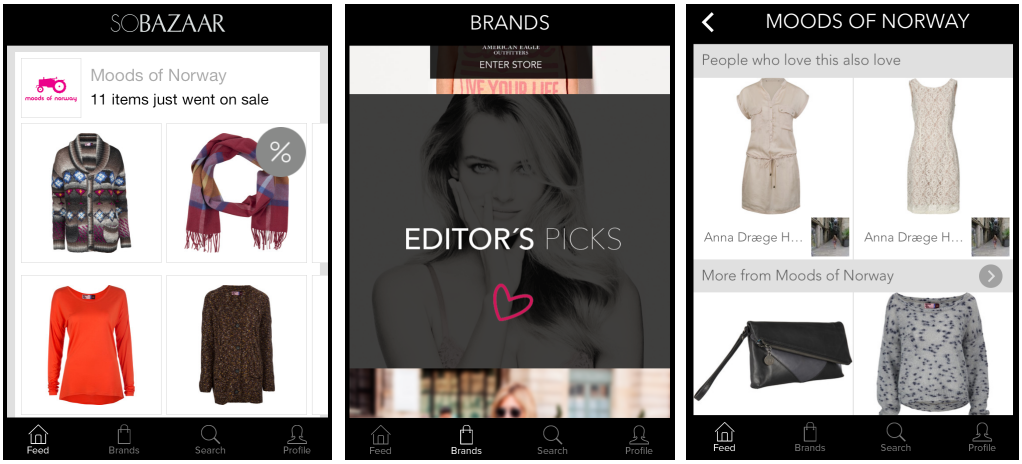


Figure 2.2: Screenshots from the SoBazaar Application. From the left to right: A newsfeed sale notification, editors picks and related product widgets from the product screen

The application is currently in a beta phase and is planned for a final release after the summer.

2.2 Data Cleaning

Some better way of saying this; Data Preprocessing, Data Prefiltering, something

The data from SoBazaar came as is from the SoBazaar database, with the exception that the data had been anonymised for privacy reasons. It contained therefore some unnecessary events. The data was therefore cleaned by removing events containing test environment flags such as: test environment and applications run from simulator. What was left after the cleaning data was mainly events triggered by the users, and no created during applications testing.

2.3 Dataset Summary

The data from SoBazaar is gathered based on the actions of the users in the application. When a user accesses a store or an item, the information regarding the user and the item is stored, this is often known as implicit feedback 4.2. Events such as purchasing an item and "wanting" and item is stored in the same manner, but can in many cases be considered as explicit feedback.

2.3.1 Event Metadata

When an event is triggered a set of information is stored regarding the event. This data is used to make recommendations for the users though converting the implicit feedback to implicit ratings.

Variable	Explanation
price	The price of the item which triggered the event
product_id	The id of the item which triggered the event
storefront_id	The store id from which the item originated
event_id	What kind of event was triggered [1]
event_location	The location of the user when the event was triggered
ts	Unix timestamp in milliseconds of when the event was triggered
session	Which session number the event belongs too [2]
user_id	The unique id of the user who triggered the event

Table 2.1: Metadata collected from an event. The complete list can be found in table A.1

[1] Complete list of the different types of events can be found in table A.2

[2] This value is added at a later time. For two events to end up in the same session, the event has to be triggered within a certain period of time, and both be after the same application started-flag

Attribute	Count
Unique users ids	1235
Unique item ids	3386
Unique brands	16
Purchases	1484
Wants	7726
Item Clicks	14036

Table 2.2: Overview of the key figures in the SoBazaar dataset

input the updated data when/if we get a new dump from SoBazaar

2.4 Graphs

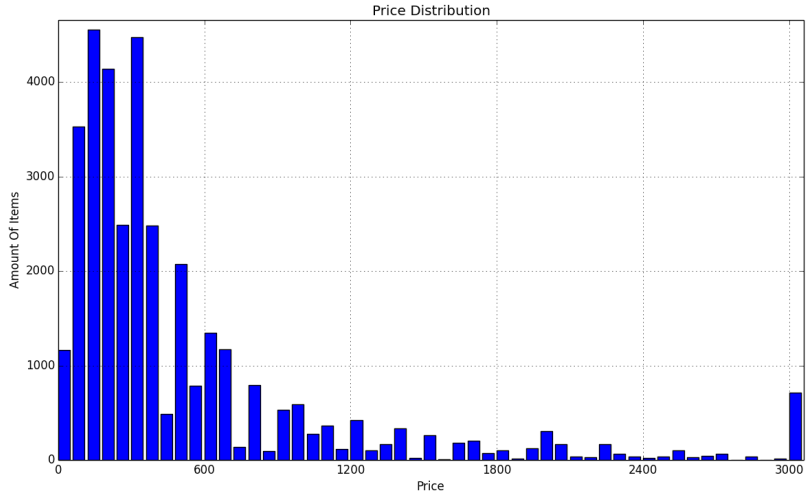


Figure 2.3: Here we see how them items are distributed on amount of items and price. The red bars indicates the amount of items with the belonging price. All items priced over 3 000 are put in the same bucket, and is the reason for the last spike we see at 3 000. We can see from this graph that the majority of the items are priced under 1 000 NOK

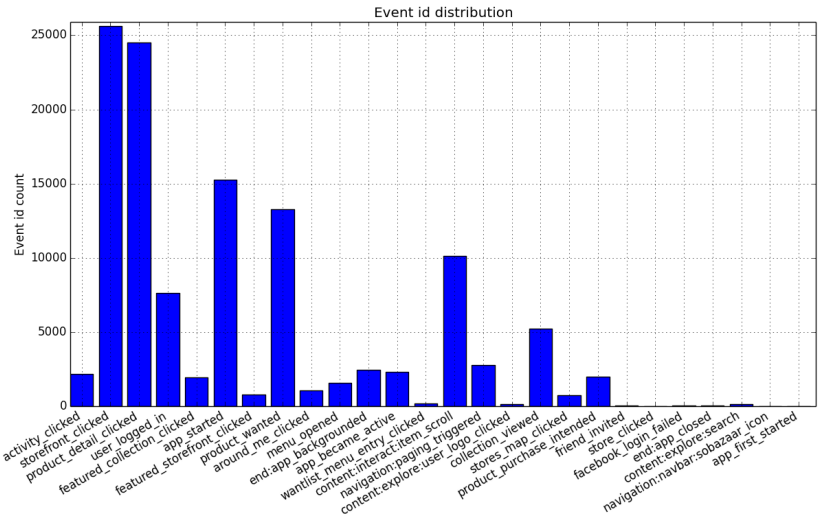


Figure 2.4: This figure displays the count for each of the different events which can be triggered in the SoBazaar application. The "collection_viewed"-event and "storefront_clicked"-event are the most common events to be triggered. Both of these types of events will send the user to an item overview, and indicates that many users browse the items through looking at the thumbnails rather than accessing the items, since the two named event types are gateways to collections of items, and if most of the users actually accessed most of the items the "product_detail_clicked", "product_wanted" and "product_purchase_intended" would have had the majority of the events triggered.

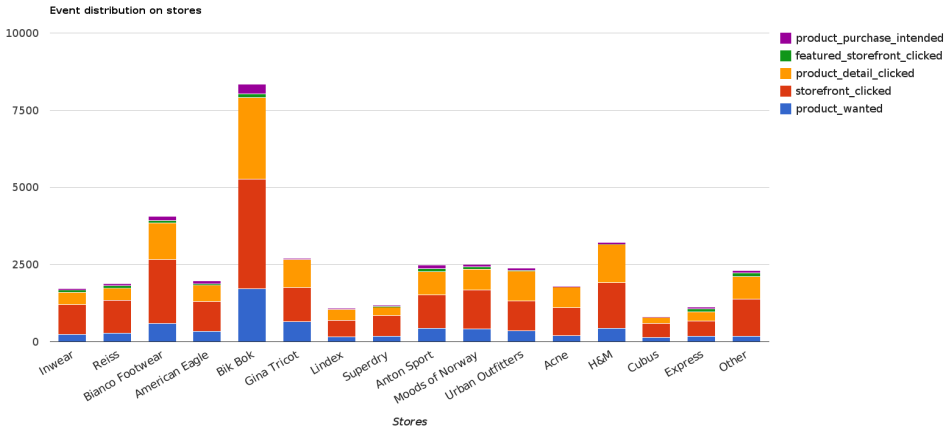


Figure 2.5: The events triggered in context with the different storefronts. The events are segmented to show the different event counts on the different storefronts, and stacked to show the complete count, to be able to clearly see how the events are distributed over the different stores. We can see that "Bik Bok" is the most popular store on all fields. One interesting find to take from this graph is the "storefront_clicked" to the item interaction related events ("product_detail_clicked", "product_wanted" and "product_purchase_intended") ratio. For instance "Bik Bok" has a much higher item interaction count than storefront access count, whereas stores such as "Reiss" and "Inwear" are mostly accessed and the items not interacted with. Different aspects affecting this might be price, style and item presentation.

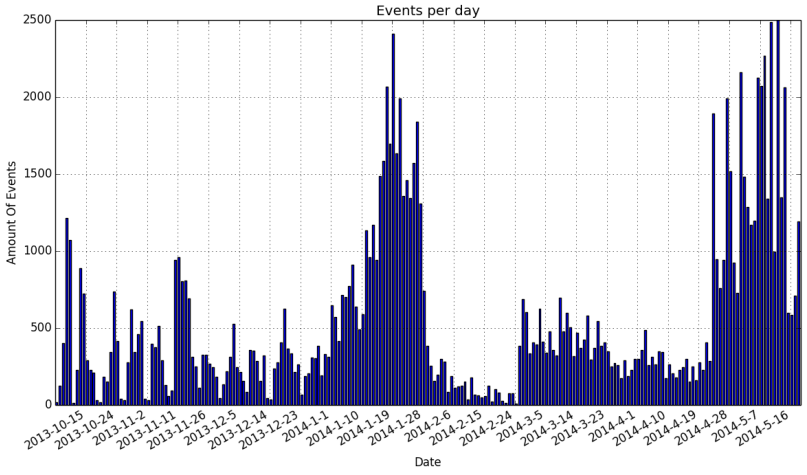


Figure 2.6: This figure shows the event distribution per day over the time period the events were stored. The spikes we see happens on a weekly basis, and is centered around the weekends. The larger spike from the start of January to the start of March might be due to increase in publicity or other outside factors.

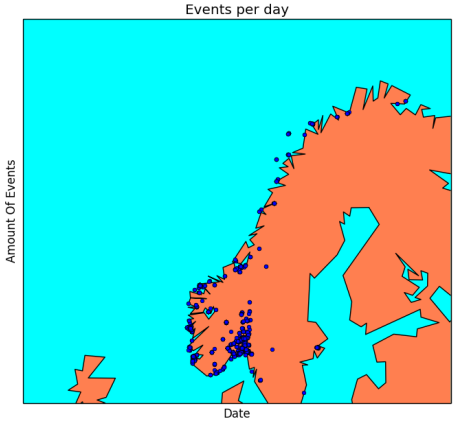


Figure 2.7: This figure shows the location of the user at the time of the different event triggers. It is cropped to show events triggered in and around Norway. We can see that the majority of the users are located in and around Oslo.

TODO: Cut at 70? (on the x-axis)

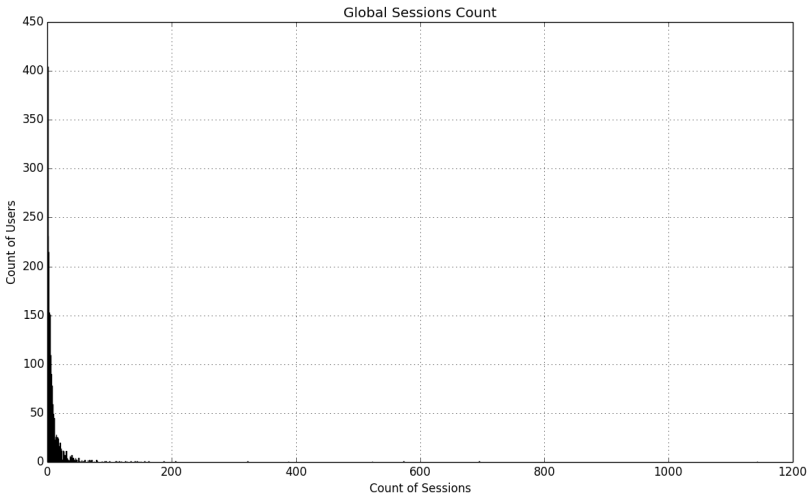


Figure 2.8: The maximum number of sessions for each user grouped to show how the count of how many users has the different amount of sessions. The majority of the users of SoBazaar has a session count of 20 and less. This means that they have not used the applications more than 20 separate times, over the time the data was collected.

TODO: cut on x-axis, make pretty

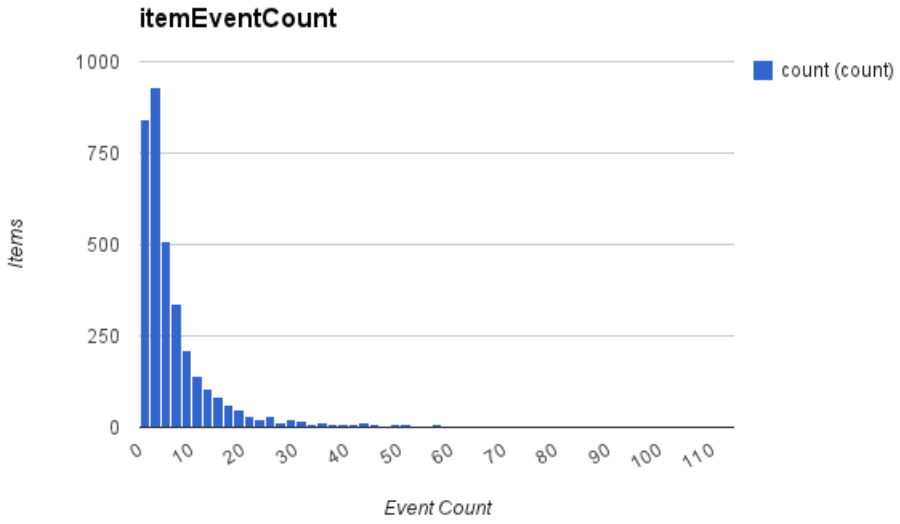
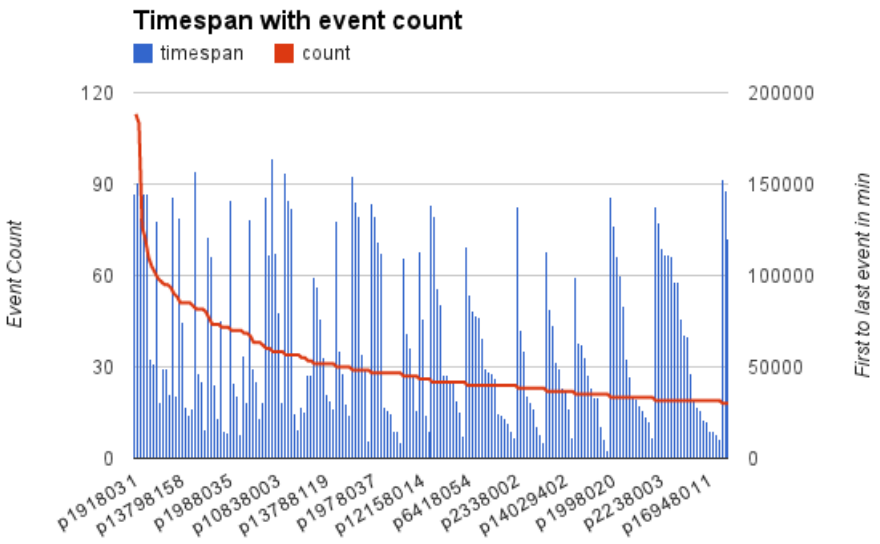
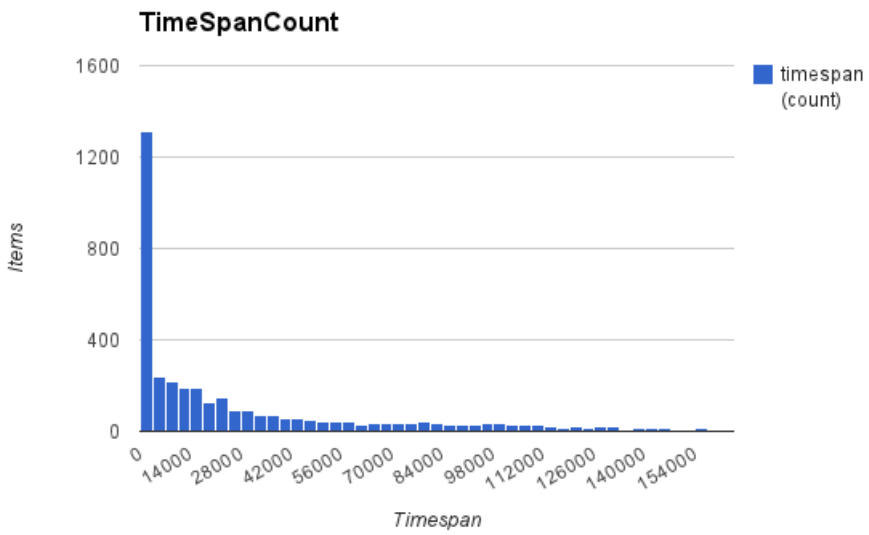


Figure 2.9: This figure shows the unique count of events on each item in the SoBazaar dataset. The majority of the items has only had a interaction count of 10 or less. Interaction count is "product_detail_clicked", "product_wanted" and "product_purchase_intended". This means that there will not be more than 10 events for the majority of the items, which might lead to an issue when doing collaborative filtering on the data. When the majority of the events only has 10 events and there are over 4 000 items and 1 200 users, the probability that multiple users have interacted with similar items will be low.



First to last event in days/weeks not minutes, remove text on a-axis (only confusing). Why is this important, clearify text, what can we learn from the plots?

Figure 2.10: This figure shows the total life span of each item mapped together with the amount of events triggered on them. The life span of an item is the time since the first event on the item till the last event on the item. Time is shown in minutes, so the longest time span of an item is about 105 days, which is close to the time span of the events gathered from SoBazaar. Even though an item has had a long time span does not mean that the item has been of measurable interest to the users in the SoBazaar application.



TODO: Change x axis from minutes to days (or preferably weeks). Does this give us any hints regarding time decay factors?

Figure 2.11: This figure shows the count of items which has the different time spans. The numbers on the x-axis is in minutes. This figure makes it clearer than figure 2.10 how long the majority of the the items have lived. Most of the items has a time span of less than 14 000 minutes, which is less than 10 days.

TODO - Use seconds?

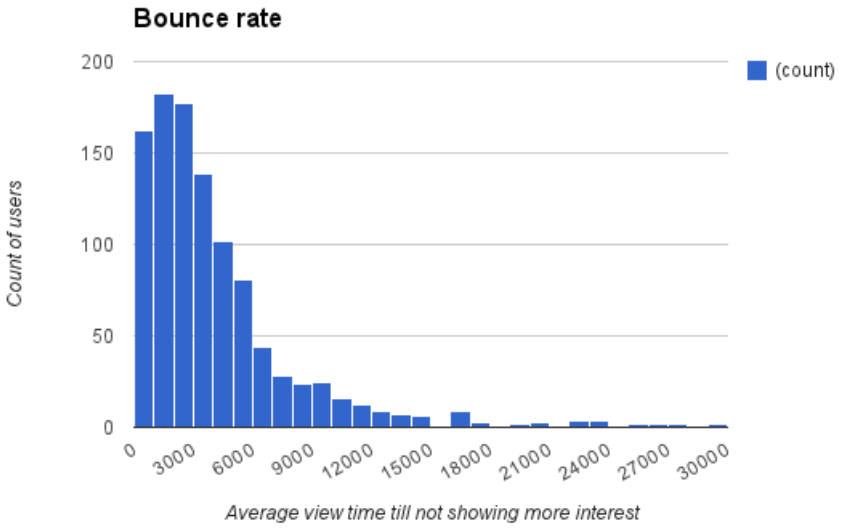


Figure 2.12: This figure shows the time the users use before not taking any more action towards the item (purchase it or want it). The time is in milliseconds. The majority of the users have a view time of less than 6 000 milliseconds before they moves on to another item.

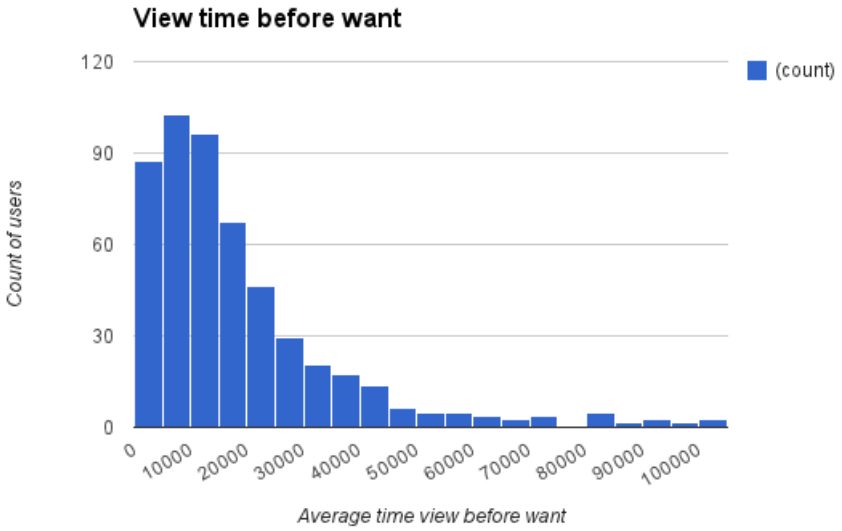


Figure 2.13: This figure shows the time the users use before wanting the currently viewed item. The time is in milliseconds. The majority of the users have a view time of less than 30 000 milliseconds before they want the currently viewed item.

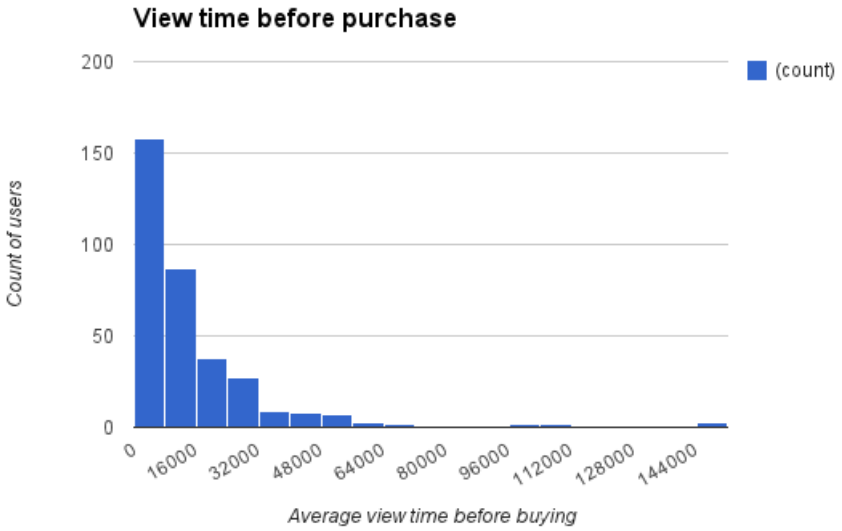


Figure 2.14: This figure shows the time the users use before purchasing the currently viewed item. The time is in milliseconds. The majority of the users have a view time of less than 32 000 milliseconds before they decide to buy the currently viewed item.

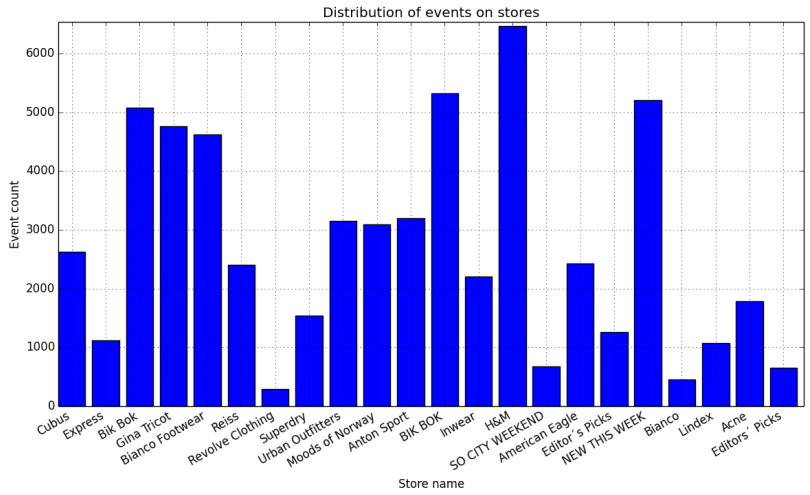


Figure 2.15: mtodo

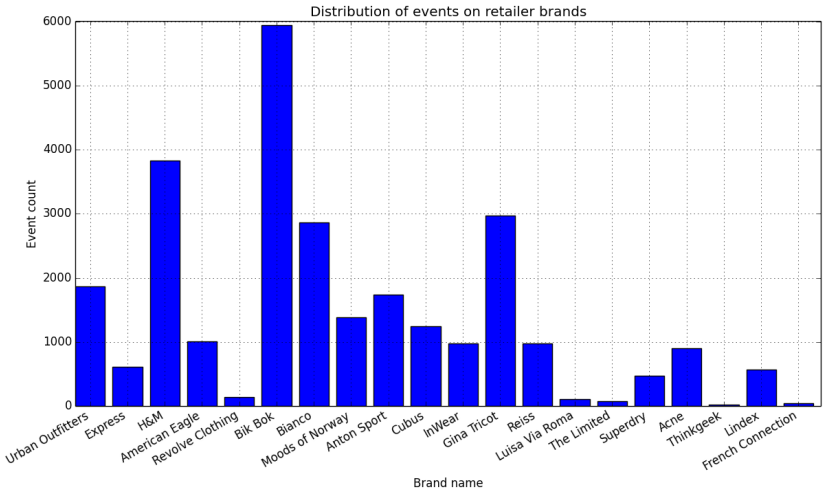


Figure 2.16: mtodo

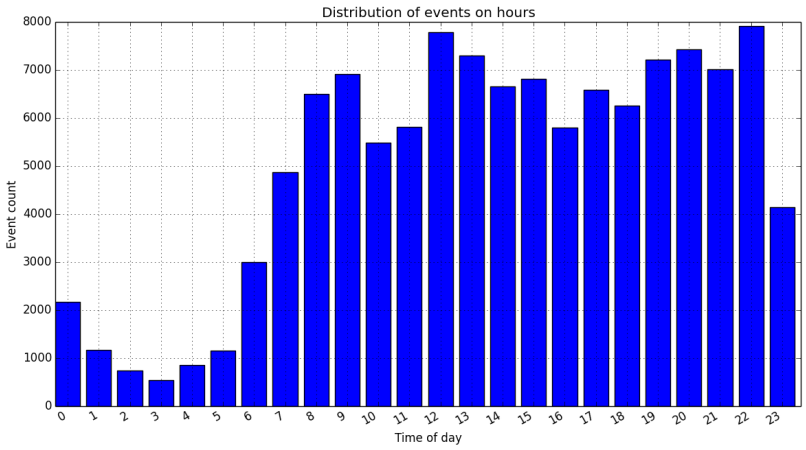


Figure 2.17: mtodo

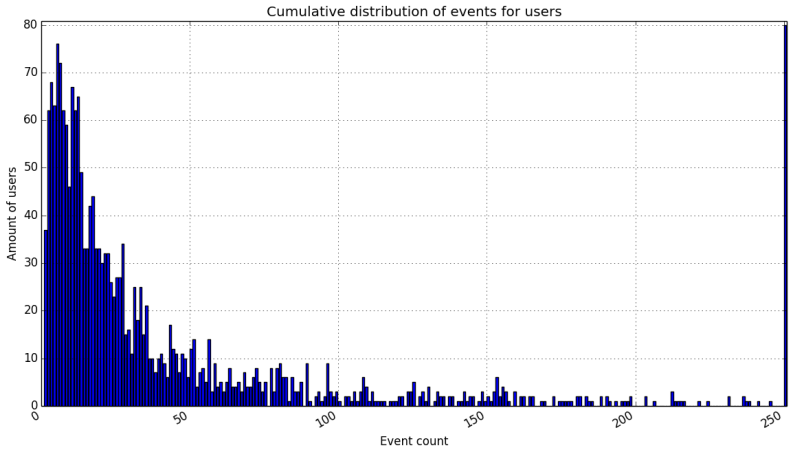


Figure 2.18: mtodo

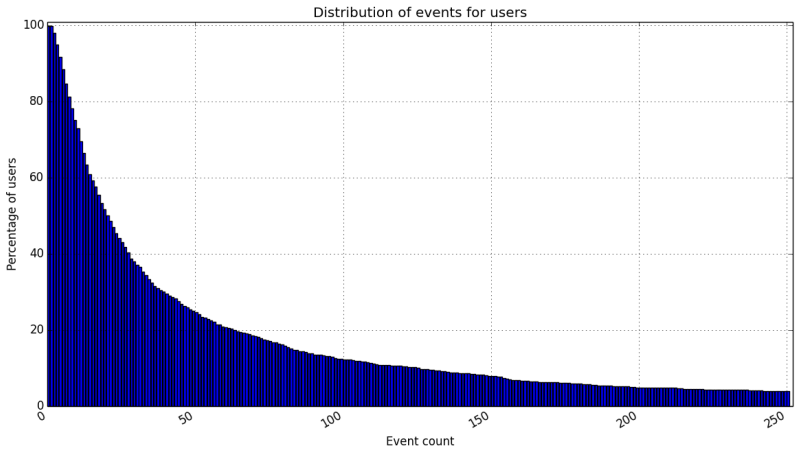


Figure 2.19: mtodo

2.4.1 About the view time before tanking an action

As seen from the figures 2.12, 2.13 and 2.19, the bounce rate is quite small compared to the time it takes for a user to decide to want or buy an item. This could be used as an indication of negative feedback, but since there is no explicit feedback to to test this, it might lead to rating an item negatively when the user in fact would like to rate it positively.

2.5 Session Findings

Something about session findings perhaps

TODO: Remove all arrows with less than e.g. 50 events

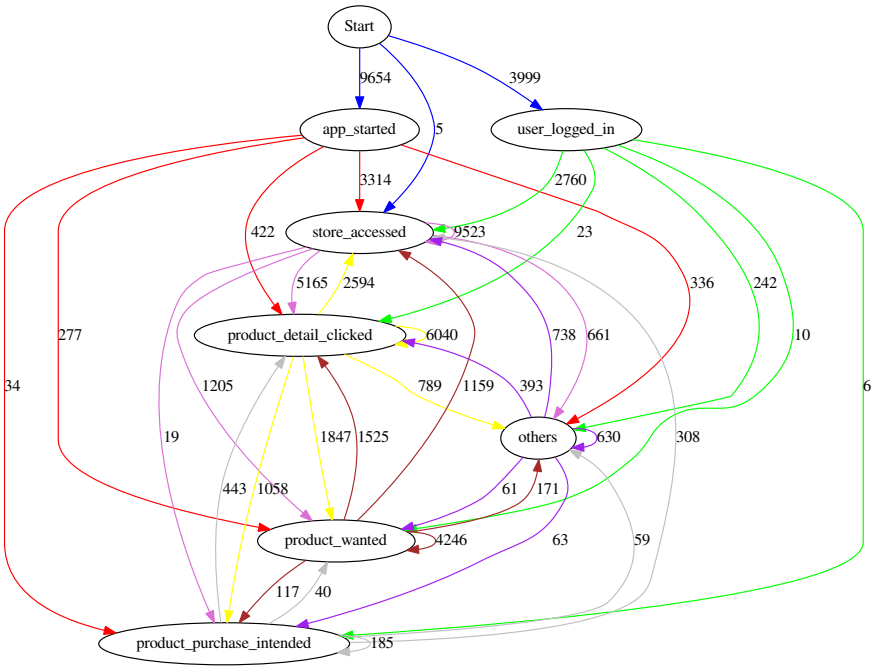


Figure 2.20: A minimized view of the different states of the system and how they interact with each other.

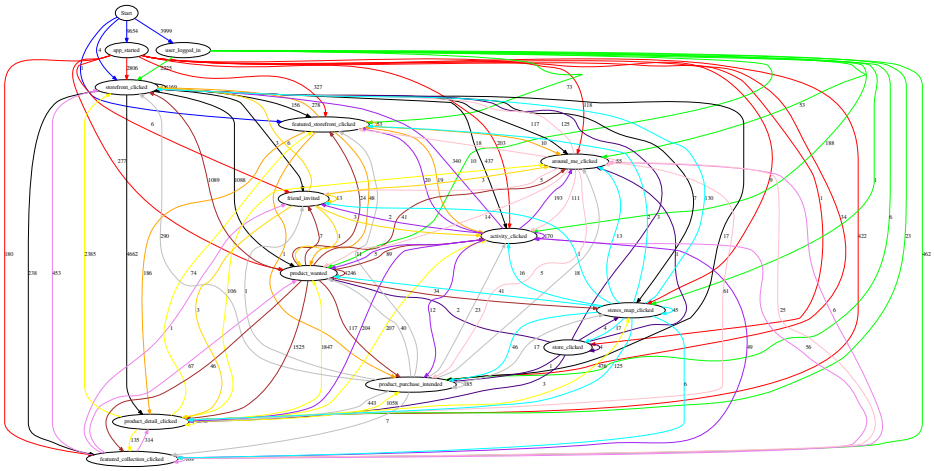


Figure 2.21: The different states of the system and how they interact with each other.

This is way too much information. Consider adding a threshold in order to reduce number of arrows

3

State Of The Art

Contents

This section will present and discuss previous work in the field of recommender systems. First out is recommender system fundamentals which cover the main recommender system techniques which the techniques described later in the chapter are based. Next up is a round up of some existing solutions to the cold start problem, followed by fashion recommender systems, session based approaches and lastly a summary of how recommender systems can be evaluated.

3.1 Recommender Systems foundations

Recommender systems have become an important research topic since the introduction of Tapestry [52], the first collaborative filtering system back in 1992. Recommender systems now play an important role in many of the most popular web-sites such as Amazon, YouTube, Netflix, TripAdvisor, Last.fm, and IMDb. In its most common formulation the recommendation problem is reduced to the problem of estimating the preference/rating of items that have not been seen by a user. Usually, this estimation is based on one or more of the following assumptions:

- You are like your friends
- You are like people who do similar things that you do
- You like things that are similar to things you already like
- You are influenced by experts and the opinions of others.

Once we have estimated these ratings we can recommend the items with the highest rating to the user. These recommendations relate to various decision-making processes, such as what items to buy, what music to listen to, or what online news to read. Recommender systems are usually classified into the following categories, based on how the recommendations are made [27].

- *Content-based recommendations:* The user will be recommended items with similar content to the ones the user preferred in the past;
- *Collaborative recommendations:* The user will be recommended items that people with similar tastes and preferences have liked in the past;
- *Hybrid approaches:* These methods combine collaborative and content-based methods.

3.1.1 Content Based Filtering

In a content-based system, we must construct a user *profile* $ContentBasedProfile(c)$ or each user c, which is a record or collection of the attributes which characterizes each item $Content(s)$ of all the items $s_i \epsilon S$ previously rated by user c. For example in a fashion recommender system the content-based recommender system tries to understand the commonalities among the items user c has rated highly in the past (color, brand, store, price, etc.). Then recommend items that have a high degree of similarity to these items. $ContentBasedProfile(c)$ can be designed as a vector of weights $(w_{c1}...w_{ck})$, where each weight w_{ci} denotes the importance of the keyword k_i to user c.

Items that can be recommended to the user can often be stored in a database table. Figure 3.1 shows a simple database with rows describing 5 items that have been rated by 3 users. The column names starting with X_n are the properties of the items, often referred to as "attributes".

Item	Arya	Cersei	Draenerys	X_1 (Price)	X_2 (Formality)	X_3 (Material)
Long Flowing Dress	1	3	5	High	High	Chiffon
Trousers	5	1	2	Low	Low	Cotton
Red Dress	1	5	4	High	High	Cotton
Red Gown	1	5	3	Very High	Medium	Silk
Quilted Tunic	5	1	1	Low	Very Low	Cotton

Figure 3.1: A clothing database. Rows are items, columns are users and item attributes

From the rating matrix and content properties one can then construct a $ContentBasedProfile(c)$ for each user c, for the user Arya one could imagine it would look something like this.

Arya

Attribute-Value		Rating
Price	Low	1.0
Price	High	0.15
Formality	Low	1.0
Material	Cotton	0.9

Figure 3.2: Content Profile Example

The recommendation process consists of matching up the attributes of the user profile against the attributes of an item. The result is a relevance judgment that represents the user's level of interest in that object. The utility $u(c, s)$ of item s for user c is estimated based on the utilities $u(c, s_i)$ assigned by user c to items $s_i \epsilon S$ that exhibit a similarity to item s. E.g. for the user Arya items with the attributes low price and low formality could safely be recommended as they fit her user profile, and have similar characteristics to the items which she previously have rated highly. The utility function $r(u, i)$ is usually defined as:

$$r(u, i) = score(ContentBasedProfile(u), Content(i)). \tag{3.1}$$

3.1.2 Collaborative Filtering

The goal of collaborative filtering methods is to suggest new items or to predict the utility $u(c, s)$ of a certain item s for a particular user c based on the user's previous activities and/or likings and similarity to other users [91]. In a typical CF scenario, there is a list of n users $C = c_1, ...c_n$ and a list of m items $S = s_1, ...s_m$. Each user c_i has a list of items S_{si}, which the user have expressed her opinion about, which makes up our rating matrix of size $S \times C$. More formally, the utility $u(c, s)$ of item s for user c is estimated based on the utilities $u(c_j, s)$ assigned to item s by the users $c_j \epsilon C$, which can be considered "similar" to the active user c. This is exemplified in Figure 3.3. For example, in our fashion recommender system, in order to recommend clothes to user c, the collaborative filtering method must find the "peers" of users c, which share the same tastes in clothes (user which tend to enjoy similar clothes). Then, recommend the clothes that are most liked among these "peers".

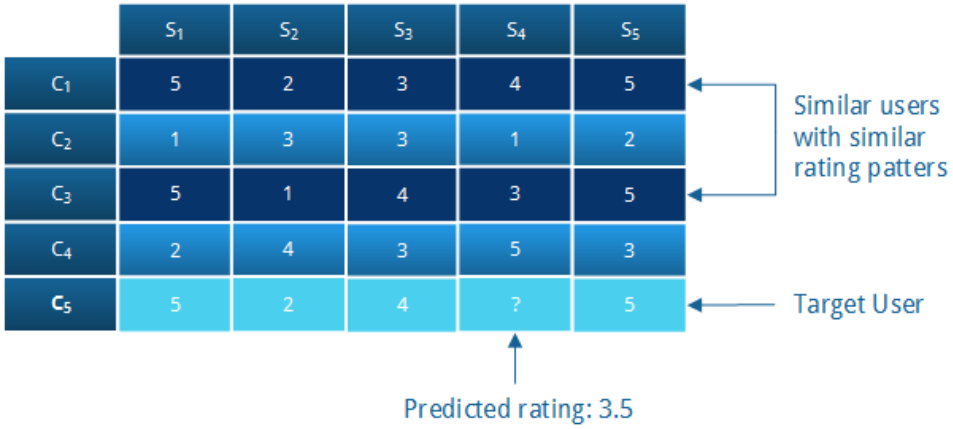


Figure 3.3: Collaborative filtering rating matrix

Researchers have devised a number of collaborative filtering algorithms that can be divided into two main categories: Memory-based and Model-based algorithms [100].

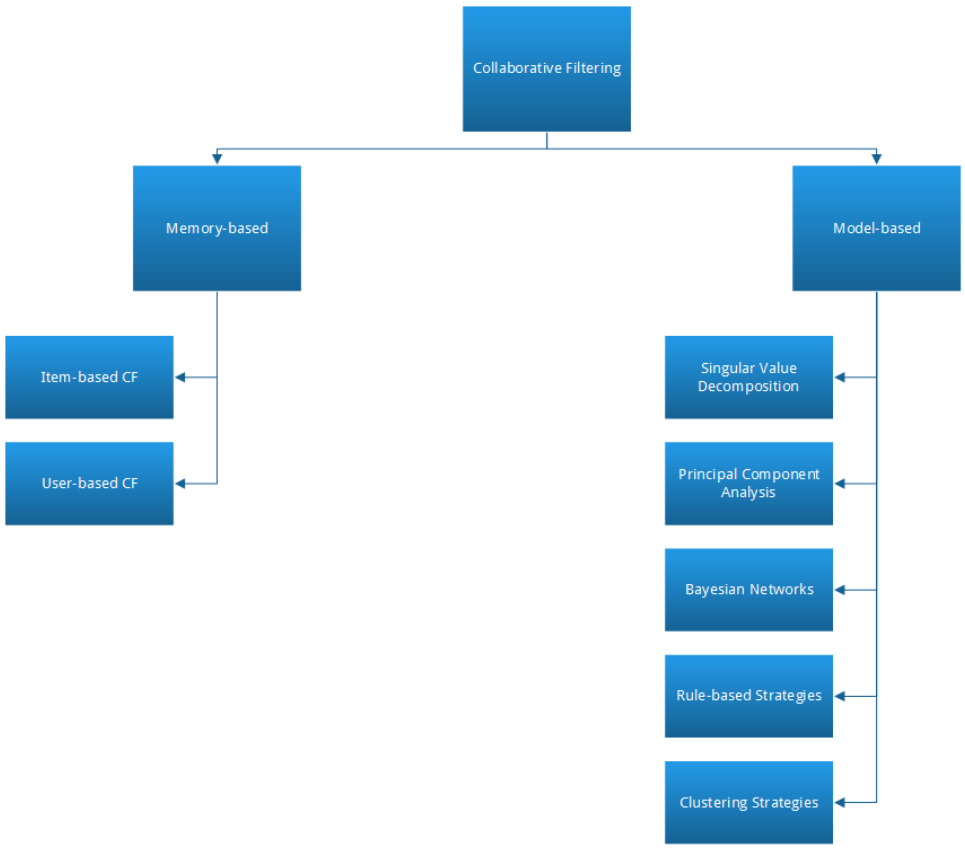


Figure 3.4: Classification of collaborative filtering techniques

Memory-based Methods

Memory-based Collaborative Filtering methods utilize the entire user-item database to generate predictions [72]. More formally, the value of an unknown utility $u(c, s)$ for user c and item s is usually computed by taking the weighted average of the utilities assigned by the N most similar users for the same item s. The similarity between user c and c', $sim(c, c')$ is used as the weight. The more similar a user c' is to c, the more weight is given to the utility $u(c', s)$, and thus, will carry more weight in the prediction for $u(c, s)$.

$$u(c, s) = k * \sum_{c' \epsilon C} sim(c, c') * u(c', s) \tag{3.2}$$

Where k serves as a normalization factor, usually being $1/|C|$. Various approaches have been used to compute the similarity $sim(c, c')$ between the users. Generally these approaches are based on the rating similarities for items both users have rated. The most popular similarity measure is The Pearson Correlation Coefficient. Equation 3.3 shows how to calculate the Pearson Correlation Coefficient between two users c and c', Here $S_{cc'}$ is the set of items both users have in *common*.

$$sim(c, c') = \frac{\sum_{s \epsilon S_{cc'}} (u(c, s) - \bar{u_c})(u(c', s) - \bar{u_{c'}})}{\sqrt{\sum_{s \epsilon S_{cc'}} (u(c, s) - \bar{u_c})^2 (u(c', s) - \bar{u_{c'}})^2}} \tag{3.3}$$

Where u_c is the mean utility of user c. The Pearson Correlation Coefficient and other similarity measures such as cosine based approaches are more commonly known user-based collaborative filtering.

Item-based Top-N Recommendation methods calculates the similarity between items instead of users. In these approaches, the historical information is analyzed to identify the relations between items such that a purchase of another item (or set of items) often leads to the purchase of another item. These models are often used since they quickly can recommend a set of items, and have shown to produce recommendation results comparable or better than traditional user-based approaches [65].

The algorithm first computes the k most similar items for each item according to the ratings given by users they both share. Once the most similar items are found, the prediction is then computed by taking the weighted average of the target user's ratings on these similar items.

$$u(c, s) = \frac{\sum_{allsimilaritems,S} (sim(s, S) u(c, S)}{\sum_{allsimilaritems,S} (|s, S|)} \tag{3.4}$$

Items that often are rated similarly by users are considered more similar than items which share few similar ratings. Figure 3.5 illustrates the process of finding the item-similarities.

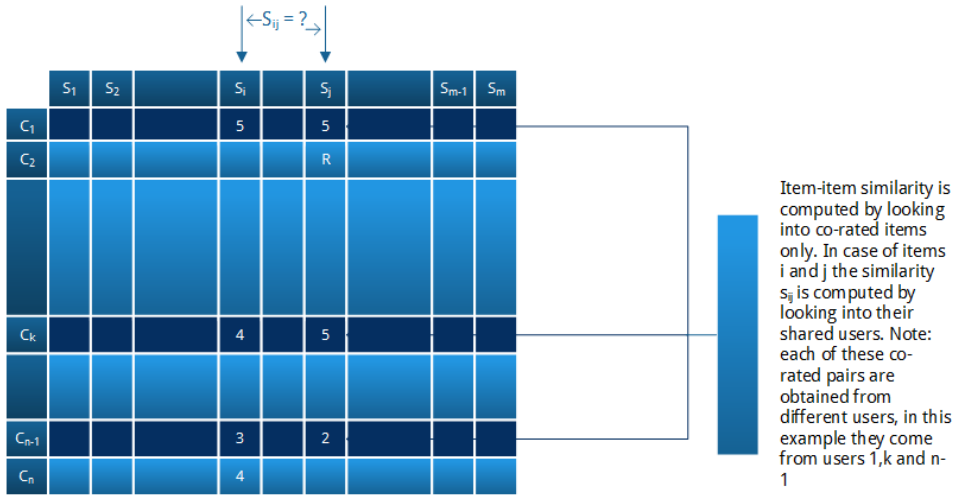


Figure 3.5: Item-item similarity

There are a number of ways of computing the similarity between items. E.g. by means of cosine-based similarity. In this case, two items are though of as two vectors in an C dimensional user space. The similarity between the items is found by computing the cosine of the angle between the two vectors.

$$sim(s', s) = cos(\vec{s'}, \vec{s}) = \frac{\vec{s'} \cdot \vec{s}}{\|\vec{s'}\|^2 * \|\vec{s}\|^2} \tag{3.5}$$

The item similarities can then used to find the Top-N recommendations. Each user has a set of items S_c previously rated by the user which we want to compute top-N recommendations for. First, we identify the set C of candidate items recommended items by taking the union of the k most similar items and removing each of the items in the set S_c the user already has rated; then calculate the similarities between each item of the set C and the set S_c, using only the k most similar items for each item in S_c. The resulting set of items in C are sorted in descending order of similarity and will be the recommended as the item-based Top-N list [65].

Model-based Methods

As the name implies, Model-based approaches provide recommendations by first developing a model of the user ratings, which is then used to make predictions [101]. These algorithms develop a model of user ratings rather than identify a neighborhood of similar users or items. These models can be built using various strategies, such as Singular Value Decomposition (SVD), Principal Component Analysis (PCA), Rule-based Strategies, Clustering Strategies and Bayesian Networks.

Latent factor models is probably the most representative approach. Latent factor models transform both items and users to a latent factor space. The latent factor space tries to explain the ratings by characterizing both items and users on factors automatically inferred from the data. The most popular latent factor models are based on matrix factorization techniques [67].

The main idea behind matrix factorization is just as its name implies, factorize a matrix, finding two or more matrices such that when you multiply them you get back the

original matrix. Matrix factorization can be used to discover latent factors underlying the interactions between the users and items. These factors *explain* how a user rates an item (i.e. that a user would give high ratings to a certain shirt if he likes the brand, or if the color is nice). If we can discover these factors, we should be able to predict a rating with respect to a certain user and a certain item based on the correlation between their factors.

A matrix factorization model map both users and items to a joint latent factor space of dimensionality f, where f is the number of latent factors. The number of latent factors are usually determined by using a hold-out dataset or cross-validation by evaluating the prediction error experimenting with different values. It is also worth mentioning that this in some ways can be seen as a trade-off between model building complexity and accuracy as having more features makes the model building more expensive. Each user c is associated with a vector $p_c \epsilon \mathbb{R}^f$, and each item s is associated with a vector $q_s \epsilon \mathbb{R}^f$. Giving us a matrix Q containing the user factors and a matrix P containing the item factors as exemplified in Figure 3.6.

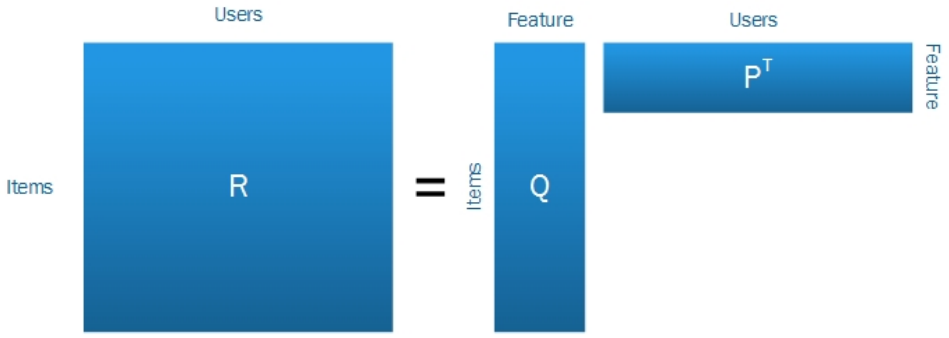


Figure 3.6: Matrix decomposition of the rating matrix R

User-item interactions are modeled as inner products in that space. For a given item s, the elements of q_s measures the extent to which the item possesses those factors, positive or negative. Likewise, for a given user c, the element p_c measures the extent of interest that user has in items that are high on the corresponding factors. The resulting dot product $u(\hat{c}, s)$ captures the overall interest of the user in the characteristics of the items.

$$u(c, s) = p_c^T q_s = \sum_{k=1}^{f} q_{sk} p_{kc} \tag{3.6}$$

The problem then, is to discover the user factor matrix P and the item factor matrix Q such that their product approximates the original rating matrix R.

$$R \approx Q \times P^T = \hat{R} \tag{3.7}$$

To learn the factor vectors the system minimizes the regularized square error on the set of known rating K.

$$min_{q,p} = \sum_{(c,s)\epsilon K} (u(c, s) - p_c^T q_s)^2 + \lambda(\|q_s\|^2 + \|p_c\|^2) \tag{3.8}$$

However, it is important to remember that our goal is generalize beyond the observed ratings, in a way that we can predict future unknown ratings. The system should therefore avoid overfitting the data by regularizing the learned parameters, whose magnitudes are penalized. λ controls the extent of regularization, and much like f, often determined by cross-validation. Two possible approaches to minimizing Equation 3.8 is to use Stochastic Gradient Descent or Alternating Least Squares [67].

Consider the following example where we have the following rating matrix R shown in Table 3.7 containing the rating of four users C for four items S, giving us a $C \times S$ matrix with explicit ratings on a scale from 1 to 5.

$$\begin{bmatrix} 5.00 & 5.00 & 2.00 & - \\ 2.00 & - & 3.00 & 5.00 \\ - & 5.00 & - & 3.00 \\ 3.00 & - & - & 5.00 \end{bmatrix}$$

Figure 3.7: Rating matrix R

Given that $f = 3$, we might end up with the following matrix P and Q

$$\begin{bmatrix} 1.81 & 1.62 & 0.74 \\ 2.66 & 1.71 & -1.08 \\ 1.73 & -0.23 & 0.78 \\ 3.16 & -0.24 & 0.90 \end{bmatrix}$$

Figure 3.8: User factor matrix P

$$\begin{bmatrix} 1.12 & 1.49 & 0.48 \\ 1.31 & -0.52 & 0.59 \\ 1.13 & 0.67 & -0.52 \\ 1.39 & 0.05 & 0.45 \end{bmatrix}$$

Figure 3.9: Item factor matrix Q

Equation 3.7 then gives us the following rating prediction matrix $\hat{R}$.

$$\begin{bmatrix} 4.79 & 5.01 & 1.97 & 3.61 \\ 1.97 & 1.96 & 2.85 & 4.80 \\ 2.75 & 4.71 & 1.40 & 2.94 \\ 2.93 & 3.30 & 2.74 & 4.78 \end{bmatrix}$$

Figure 3.10: Rating prediction matrix $\hat{R}$

As you can see the values of known ratings in Table 3.7 are fairly similar to the corresponding ratings in the rating prediction matrix.

3.1.3 Hybrid approaches

A term *hybrid recommender systems* is used to describe any recommender system that combines multiple recommendation techniques together to provide recommendations. Burke et al. [37] identified seven different classes of hybrid recommender systems:

- Weighted: The score of different recommendation components are combined numerically.
- Switching: Switching between recommender systems depending on the situation.
- Mixed: Recommendations from different recommenders are presented together.
- Feature Combination. Features derived from different knowledge sources are combined together and given to a single recommendation algorithm
- Feature Augmentation: One recommendation technique is used to compute a feature or set of features, which is then part of the input to the next technique.
- Cascade: Recommenders are given strict priority, with the lower priority ones breaking ties in the score of the higher ones
- Meta-level: One recommendation technique is applied and produces some sort of model, which is then the input by the next technique

Most commonly hybrid systems are built by combining collaborative and content-based methods in an attempt to mitigate the limitations the approaches suffer individually. Adomavicius and Tuzhilin [27] lists the following approaches to building hybrid recommender systems:

- Implementing the systems separately and combining their predictions
- Incorporating content-based characteristics into a collaborative approach
- Incorporating collaborative characteristics into a content-based approach
- Constructing a general unifying model that incorporates both content-based and collaborative characteristics

heri-notes: Which approach relevant to this work?

3.1.4 Recommender System Challenges

Below we briefly mention some of the main challenges one faces when working with recommender systems.

Scalability

As the number of existing users and items blow up, traditional CF algorithms suffer serious scalability problems. The model building phase of *Traditional* collaborative filtering methods have a complexity of O(MN) where M is the number of users and N is the number of items. For systems with millions of users and items, even a complexity of n is too large. Another fundamental issue is how to embed the core recommendation techniques in real operational systems and how to deal with massive and dynamic sets of data produced by the interactions of users with items. Recommender systems are

expected in many cases to provide rapid recommendations online, it is therefore also important to consider how fast the system provides recommendations.

Better scalability and improved accuracy make the item-item collaborative filtering approaches more favourable in many cases. The computational complexity of item-to-item based algorithms are up to two orders of magnitude faster than traditional user-based algorithms [43]. Dimensionality reduction techniques such as SVD can deal with the scalability by providing more compact representations and quickly produce good recommendations. However, most dimensionality reduction techniques must undergo expensive matrix factorization steps.

Sparsity

In practice, many recommender systems deal with very large item collections. This means that the number of ratings obtained is usually very small compared to the number of ratings that it needs to predict. Efficient prediction of ratings from a small number of examples is therefore important. The *reduced coverage* problem occurs when the number of users' rating may be very small compared to the number of items. This may lead to that the recommender is unable to provide recommendations for a large portion of the items. *Neighbour transitivity* refers to the problem in which users with similar tastes may not be identified due to a lack of co-rated items, making collaborative filtering futile, since it relies on comparing users to predict unknown ratings.

Cold-start

Conceptually, the cold-start problem can be viewed as a special instance of the sparsity problem, where most elements in a certain row or column are zero. The cold-start problem further emphasizes the importance of the sparsity problem. Whenever a new user or item enters the system, it is difficult to find similar ones as there is little or no information available. New items can therefore not be recommended until they have been recommended by a substantial amount of users. Similarly, giving *good* personalized recommendations to new users based on a few ratings is difficult, since it does not give a good overall picture of a users tastes and preferences. These problems are known as the *cold-start user* and *cold-start item* problems.

Shilling attacks

In recommender systems where everyone can give ratings, people may give lots of positive ratings to their own items and negative ratings to their competitors. It is often necessary for collaborative filtering systems to introduce precautions to discourage such kind of manipulation.

3.1.5 Terminology

Explanations / Transparency

heri-notes: Hvorfor har dere med dette?

Tintarev et al. [102] lists seven roles a that can be played by explanations in recommender systems:

- Transparency: Explaining how the system works
- Scrutability: Allowing the users to tell the system it is wrong

- Trust: Increasing user confidence in the system
- Effectiveness: Help users make good decisions
- Persuasiveness: Convince users to try or buy
- Efficiency: Helping users to make decisions faster
- Satisfaction: Increasing the ease of use or enjoyment

In collaborative filtering systems the explanations is of the form "Other users similar to you liked this item", while in content-based style explanations, the item's attributes which most affected the item to be recommended to the user are illustrated. For example, in a fashion recommender, an explanation may be of the form "This shirt was recommended because it's a Ralph Lauren who you seem to like".

3.2 The Cold-start Problem

In the literature, the term cold is used about an object in a system, or a whole system, which is new [94, 85]. Cold-start scenarios in recommender systems are situations in which little/no prior events, like ratings or clicks, are known for certain users or items. The cold-start problem can be divided into three sub problems:

- *Cold-start system*: A situation where we only have new users and little or no ratings for the items.
- *Cold-start item*: The problem of recommending items that are new to the system, which have not received any ratings.
- *Cold-start user*: The problem of giving accurate recommendations to a user who is new to a recommender system.

For example in a scenario where the average item in an item collection have 5 000 ratings, a new item with only 5 ratings would be considered a *cold-item*. Likewise, in a recommender system where the average user has rated 25 items, a user who only has rated 2 items, would be considered a *cold-user*.

The cold-start system problem is mainly a collaborative filtering problem, and can be seen as a combination of the cold-start user and cold-start item problem where the majority of the users are new to the system and have expressed few preferences, resulting in a very sparse user-item matrix, rendering traditional collaborative-filtering methods futile. Most traditional algorithms only work effectively in environments where the datasets has high information density. In fact, in extreme cases, when data is very scarce, simple non-personalized recommendations based on global averages can outperform collaborative-filtering algorithms [85]. The reason why the cold-start system problem is not so evident in content-based systems is due to *User Independence*, meaning that the system only exploits ratings provided by the active user to build her profile. Instead, collaborative filtering methods need ratings from other users to find the "peers" of the active users.

In content-based systems, new items can easily be recommended using the content information of the item, making it a popular solution to the *cold-start item* problem. This problem is more severe in collaborative-filtering systems where items are only

recommendable if they have been rated by substantial amount of users. New items will therefore not be recommendable before multiple users somehow stumble upon the new item while e.g. browsing the item collection, unless additional measures are taken to solve this problem. To *solve* the new-item problem, there are two commonly used (simple) solutions often used in E-commerce websites:

heri-notes: cite

- Advertising at the front-page of the website, putting the new items in an eye catching position. This solution, however, may this result in that some users, which don't like these new items, might leave the website.
- Requesting the user to choose one or more of his/hers categories while registering for the site, and recommend items from the selected categories. This approach however, requires active user involvement and complicates the sign up process. Many users might chose not to give up any personal interest information, thus the user group covered by this solution could end up not being large enough.

The cold-start user problem is present both in content-based and collaborative-filtering systems. Since Collaborative Filtering is based on the idea that like-minded users have similar tastes and preferences, a new user therefore naturally poses a challenge to a CF recommender, since the system has no knowledge about the preferences of the new user, and can therefore not find any like-minded users. The system must therefore acquire some information about the new user before it can start making personalized recommendations. In a typical domain, for example in the domain of books, the number of items is very large (in the order of tens of thousands) while the number of items rated by every single user is in general small (in the order of dozens or less). This means that it is very unlikely two random selected users have rated any items in common and hence they are not comparable. The system will therefore most likely struggle to find users with tastes that are *truly* similar to the target user. Similarly, in content-based systems, the lack of ratings given by the target user, means that the target user will have a limited content-profile, since the users content profile is constructed using content-information from his/hers rated items. In both cases, recommendation quality is most likely bound to suffer.

heri-notes: hvordan kan dette problemet vre relevant for oppgaven deres?

This section will present a few different solutions to the cold-start problem, focusing mainly on *complete* solutions to the cold-start problem.

3.2.1 Trust Aware Recommender Systems

One promising direction to solve the cold-start problem is the incorporation of a trust network. A trust network can significantly help alleviate the cold-start user problem, primarily since the trust statements between users can be propagated and aggregated, and consequently connect more people and products. By making clever connections in the trust network, newcomers can immediately gain access to a wide range of connections.

For example, when looking for movie recommendations we often turn to our friends which we share a similar taste in movies with. Trust can be defined as: "believe in the reliability, truth, or ability of", and in the context of recommender systems a trusted user would be a user you trust to provide you with good recommendations. E.g. in the case of the Epinions dataset [11], users can explicitly state whether they trust or distrust a user [1, -1], i.e. reviewers whose reviews and ratings they have consistently

found to be valuable or reviewers which they find consistently offensive, inaccurate or not valuable. In decentralized environments where everyone is free to create content and there is no centralized quality control entity, evaluating the quality of the content becomes an important issue. This phenomenon can be observed in online marketplaces such as E-bay where users can create "fake" auctions and in peer-to-peer networks where peers can enter corrupted items. In these environments, it is often a good strategy to delegate the quality assessment task to users themselves. E.g. *Ebay.com* allows users to express their level of satisfaction after every interaction with another user. Trust relationships of users are often employed in order to correlate more potential raters for the active users who require recommendations [77, 78]. Massa et. al. [77] also show that some of the weaknesses of recommender systems such as data sparseness and their susceptibility to shilling attacks could be alleviated by incorporating trust.

In [77], Massa et al. proposes a Trust-Aware recommender system architecture. To capture all the trust statements we need a CxC matrix, where C is the number of users, since each user is allowed to express a trust value in every other user. This matrix will make up our trust network among the users. If u trusts v, then there is a value $t_{u,v}$ for this trust which is a real number in $[0, 1]$. Zero means no trust and one means full trust. This additional information can be organized in a trust network and a *trust metric* can be used to predict the trustworthiness of other users as well (for example, friends of friends). The idea here is to not search for similar users as CF does but to search for trust-able users by exploiting trust propagation over the trust network. The items appreciated by these users are then recommended to the active user.

Consider the example shown in Figure 3.11. User A has issued a trust statement in B and C; hence B and C are in the web of trust of A. Using these explicit trust statements, it is possible to predict trust in unknown users by propagating trust, making it possible to infer something about how much user A could trust D.

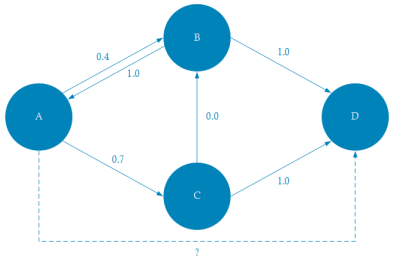


Figure 3.11: Trust Network. Nodes are users and solid edges are trust statements. The dotted edge is one of the undefined and predictable trust statements (Adopted from [77])

In addition to the trust network we will also have a rating matrix of size CxS, where S is the number of items. This rating will not differ from a standard rating matrix, which are used in traditional collaborative filtering systems. The value $u(c, s)$, is the rating given by user c to item s, the rating scale may differ from system to system.

The systems takes as input the trust network and the ratings matrix and produces, as output, a matrix of predicted ratings that the users would assign to the items. Figure 3.12 shows a conceptual overview of the trust-aware recommender system architecture.

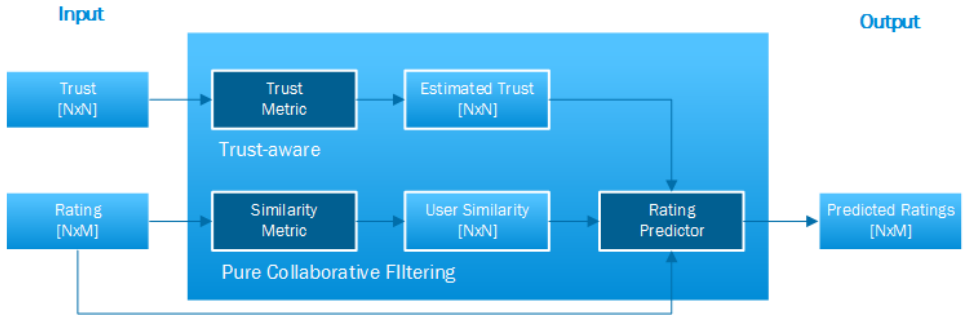


Figure 3.12: Trust-Aware Recommender System Architecture (Adopted from [77])

The *Trust Metric* module takes the trust network as input, and exploits trust propagation in order to predict, for every user, how much she could trust other users. Trust metrics can either be local and global. Global trust metrics produces an estimated trust matrix with all the rows equal, meaning that the estimated trust in a certain user (column) is the same for every user (row). A simple local trust metric could e.g. for each user assign to every other user a predicted trust based on her minimum distance from the source user. More sophisticated ones could also be employed. If we again consider Figure 3.11, we could employ a local trust metric where the predicted trust is based on the minimum distance from the source user. If we set the maximum propagation distance d, a user at distance n from the source user will have a predicted trust value of:

$$t_{u,v} = \frac{d - n + 1}{d} \tag{3.9}$$

Giving users not reachable within the maximum propagation distance a trust of 0. Using user A as the source user, the users at distance 1 (B and C) would get a trust value of $(4 - 1 + 1)/4 = 1$, while the user at distance 2 (D) would get a predicted trust value of $(4 - 2 + 1)/4 = 0.75$. Meaning that we will have a linear decay in trust based on the distance from the source user.

Massa et al. [78] experimented with both local and global trust metrics. They used the PageRank algorithm as a global trust metric. PageRank tries to infer the authority of every single user by examining the structure of the network. The algorithm follows a simple idea: if a link from user A to user B represent a positive vote casted by A to B, then the global rank of a page depends on the number (and quality) of the incoming links. The trust values assigned by users to users are used to predict the trustworthiness of unknown users. Their findings, not surprisingly, indicate that Global Trust Metrics are not suited for the task of finding good neighbours, especially for providing personalized recommendations, but is more suited to applications such as *Ebay.com* to find untrustworthy users. As a local trust metric they used MoleTrust, which is a depth-first graph walking algorithm with a tuneable trust horizon which allowed them to experiment with different propagation distances. They found trusted users to be good predictors. For the cold-start users they achieved a MAE of 0.674 when looking at friends of friends, compared to traditional collaborative filtering which scored 1.094. The difference is very high, and particularly relevant as it is important for recommender systems to generate personalized recommendations as soon as possible for new users, so that these users appreciate the system and keep using it.

The *Similarity Metric* module computes the user similarities, this is one of the standard steps of any traditional collaborative filtering technique, user similarities can be found e.g. by using the Pearson Correlation Coefficient. The intuition is that, if a user rates in a similar way to another user, then her ratings are using for predicting the ratings for that users.

The *Rating Predictor* can use the neighbours from the user similarity matrix, the estimated trust matrix or a combination of both in order to calculate the predicted ratings.

Jamali et al. [60] propose two different methods for getting around the cold-start user problem using a trust network. Their first approach called *Random Walk* only utilize the trust network to provide recommendations. Starting from the active user u, we perform a random walk on the trust network. Each random walk stops at a certain user. Then the items rated highly by that user will be considered as recommended items, ordered according to the ratings expressed by that user. Several random walks are performed to gather more information and compute a more confident result. The estimated rating of each item is the average of ratings for that item over all raters considered. At the end, we output items with the highest estimated rating as top-N recommended items. Their second approach called *Combined Approach* uses both user-user similarities and the trust network to provide recommendations. In this approach we compute the top K trusted users in the network and rank the items rated by these trusted users to compute top-N recommended items. The top K trusted users can either be found by *Breadth First Search* or *Random Walk in the social network*. We use the collaborative filtering approach to compute another set of top-N recommended items. Finally, we merge these two lists to produce a combined list of top-N recommended items. Items returned by CF is denoted as CF_u, while the items returned by Trust-based approach are denoted TR_u.

$$\hat{u}(c,s) = \begin{cases} \frac{u_{tr_{c,i}} + u_{cf_{c,i}}}{2} & i \in TR_u; i \in CF_u \\ \hat{u_{tr_{c,i}}} & i \in TR_u; i \notin CF_u \\ \hat{u_{cf_{c,i}}} & i \in CF_u; i \notin TR_u \end{cases} \tag{3.10}$$

The top-N items with the highest value of $\hat{u}(c,s)$ will be returned as the top-N recommended items. The authors also experimented with weighted averaging in the case where the item appear in both TR_u and CF_u.

The top-N items with the highest value of $\hat{u}(c,s)$ will be returned as the top-N recommended items. The authors also experimented with weighted averaging in the case where the item appear in both TR_u and CF_u. Their approaches showed great improvements in recall for cold-start users, improving the performance by 50% over standard CF methods. The main improvements however, are the coverage of the trust-based approaches, while still maintaining the same or even slightly better precision than the standard CF methods.

Papagelis et al. [84] proposed to alleviate sparsity using trust inferred from user-user similarity. This approach does therefore not require users to explicitly express their trust in other users, unlike the approaches described above, the trust information is inferred from the underlying social network of the rating matrix. Their approach is based on the assumption that the more similar two users are, the greater their established trust would be considered.

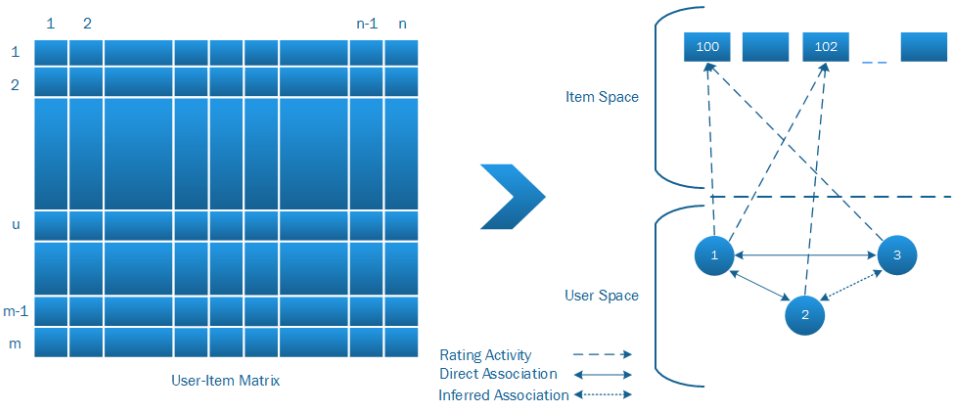


Figure 3.13: Underlying Social Networks in Recommender Systems

Due to the number of ratings that exist in recommendation systems, underlying social networks are very sparse. There are cases in which insufficient or loss of information is detrimental for the recommendation algorithms. Consider Figure 3.13, classic CF will associate only the users which have co-rated an item (User 1 and 2 and user 1 and 3). To deal with the problem of a sparse social network, it is possible to infer trust between a source user S and a target user T through an intermediate user N (User 2 and 3 are connected through the intermediary user 1), as shown by the *Inferred Association* arrow. According to this process, trust is propagated in the network and associations between users are built, even if they have no co-rated item. Trust paths can be of variable length, depending on the number of associations that one needs to traverse in order to reach the target user.

For example, if the trust $t_{(1,2)} = 0.7$ based on 5 co-rated items and $t_{(1,3)} = 0.35$ based on 2 co-rated items, then the trust between user 2 and 3 through 1 is, $\frac{0.7*5}{5+2} + \frac{0.35*2}{7} = 0.6$.

In order to express the subjective notion of trust, the authors set up a confidence model that is assigned to each direct association of the network that expresses the reliability of the association. Confidence is related to the number of co-rated items between two users. The confidence scores are all expressed in relation to the most confident association for each user.

$$c_{(s,t)} = \frac{n(I_s \cap I_t)}{n(I_s \cap I_{u_{MAX_{CONF}}})} \tag{3.11}$$

Using the above example, assuming that the maximum number of co-rated items user 1 has with any user is 7, $c_{(1,2)} = \frac{5}{7}$.

The authors achieved improved accuracy for all sparsity levels. With a sparsity level of 99.9% the 2-HOP CF (friend of friends) increased the MAE performance by 17% over standard CF methods.

Victor et al. [104] points out that cold-start users not only have expressed few ratings, but also typically have expressed trust in few users. In order for trust-aware recommenders to help cold-start users they need to have expressed trust in atleast one user. But choosing who to connect to is often a difficult task. To help cold-start users find trusted users, Victor et al. propose using key figures or mavens (users who write many reviews), frequent raters (users who evaluate many items) and connectors (users with many trust connections). By connecting to these key figures, cold-start users shown

a significant increase in coverage while still maintaining good accuracy. They also show that connecting to key figures are more beneficial to a cold-start user than connecting to a random user.

3.2.2 Filterbots

Park et al. [85] propose using filterbots to improve the cold-start performance of collaborative filtering methods. Their filterbots are a varition of RipperBots, described in detail in [53]. A filterbot is an automated agent that rates all or most items using information filtering (IF) techniques. The filterbots injects psuedo users or bots into the system. These bots rate items algorithmically according to item features and user profiles. For their movie recommendation systems the authors used 7 global bots which e.g. rated movies based on average item rating, a critic bot that generates ratings based on the average critic (pre-selected users) ratings, an award bot that generates rating based on the awards a movie has won, and so on. These ratings generated by these bots are injected into the user-item matrix along with actual user-item ratings. Standard CF algorithms are then applied to generate recommendations.

Their approach clearly demonstrated better robustness to all three cold-start situations than standard item-based and user-based collaborative filtering. The improvements were most evident on the datasets with a high degree of sparsity.

3.2.3 Wisdom of the better few / Seed users

Liu et al. [73] propose an approach in which they elect a few representative users and items. The representative set should represent a set of active users or items who well represent the entire population but with little taste overlap. In their approach they wish to find a rank-k factorization of the form $Y \approx XR$ or $Y \approx CX$ where X is a loading matrix consisting of free parameters and R and C which is the component matrix consisting of actual rows or columns from Y. The representative users and items are found using dimensionality reduction techniques by reducing the column space of the rating matrix from m to k. And then applying basis selection based on the maximum-volume principle to select the k most representative users or items. In order to be able to recommend new items to the users it must first be rated by the k representative users, likewise for new users to be rated they need to rate the k most representative items. Their method therefore easily allows new users and items to be *folded in*.

3.2.4 Intelligent Selection / Interview Process

> *"Ask the right questions if you're going to find the right answers"*
> – Vanessa Redgrave

As pointed out by Rashid et al. [89], the most direct way of acquiring information for use in personalized recommendations from a new users is to present item for the user to rate. However, they argue that the system must be careful to present useful items to garner information. A food recommender should probably not ask whether a new user likes vanilla ice cream since most people like vanilla ice cream. Therefore, knowing that a new user likes vanilla ice cream tells you very little about the user. The choice of what questions to ask a new user, then, is critical. The authors performed a study of

different item selection strategies that collaborative filtering recommender systems can use to learn about new users. They presented the users with a questionnaire with items asking them to rate/select the ones they like. Their strategies can be divided into five classes, which they evaluated based on user effort and accuracy:

- *Random:* strategies: Strategies that avoid bias in the presentation of bias
- *Popularity:* Select among the top N items where the probability that an item is selected is proportionate to the items popularity.
- *Pure entropy:* Present the items with the highest entropy that the user has not seen
- *Balanced strategies:* A balanced approach combining both popularity data and entropy.
- *Personalized:* As soon as some information is known about a user, present items specifically tailored to that user using e.g. item-item similarity

The authors found Popularity and balanced strategies to perform the best. Their recommendation for an e-commerce recommender is to start recommending the most popular items, rather than the highest rated ones, and then use item-item strategies to personalize the recommendations as quickly as possible. This study was later extended by Rashid et al. [90] where they more closely examined information theoretic strategies for item selection. In the article they introduced three new strategies, which again was evaluated based on user effort and accuracy:

- *Entropy0*: Entropy Considering Missing Values
- *IGCN:* Information Gain through Clustered Neighbors

The authors point out that approaches like popularity is likely to worsen the *prefix-bias*, meaning that popular items garner even more evaluations. The accuracy differences between the approaches is fairly small, IGNC performed the best closely followed by Entropy0 and Popularity. However, the expected utility of the profiles built using popularity is much lower than the information theoretic approaches.

The question then, is how many items you should ask a user to rate. Cremonesi et al. [42] performed a set of experiments where they looked at the trade-off between user-effort and accuracy. More specifically, how many ratings are enough to provide good quality recommendations to new users? The authors conclusion is that between 5 and 20 ratings are optimal for the movie domain. They concluded that 10 ratings is *enough*, but that this number depends on the recommendation method and the dataset used.

3.2.5 Hybrid Methods

Another line of search for solving the cold-start problem is to utilize features of items and users. The content features can be used to capture the similarities between users and items, thus reducing the amount of data required to make accurate predictions. User data that may be collected typically includes age, gender, nationality, marital status, income, educational level and occupation. Item data could e.g. be the price of a product,

title, description, editorial ratings and so. The idea is that people with a more common background share a more similar taste than someone with a random background, and therefore good recommendations can be made as long as we know something about the new user's background.

This section will present some latent factor models presented recently proposed that incorporate both user/item features in addition to user-item interactions. In Matrix factorization methods, the regularization is mostly based on a zero-mean Gaussian prior on the factors, we refer often referred to as ZeroMean. However in the following models the dyadic response matrix Y is estimated by a latent factor model such that $Y \approx U^T V$, where the latent factor matrices, P and Q, are estimated by regression such that $P \approx FX$ and $Q \approx MZ$. X and Z denote user attribute and item feature matrices, and F and M are weight matrices learned by regression. The main difference between the following methods is how they estimate these weight matrices.

Agarwal et al. [28] propose a class of latent factors models called regression-based latent factor model (RLFM) that incorporates both user/item features and past interaction data into a single model. Their approach utilizes features of items and users as the prior distribution for latent profiles in matrix factorization. Regularizing latent factors through regression has important consequences when modeling sparse dyadic data. For users/items with little data, one obtain reliable factor estimates by using the regression estimates as a fallback. This allows the model to effectively deal with both cold start and warm start situations. Their method assumes a Gaussian prior, but replaces the zero mean with a feature-based regression, thus it simultaneously regularizes both user and item factors through known features. Users and items are anchored around a global feature-based one where profiles are constructed by estimating deviations from the global ones in a smooth fashion. The deviation depends on the amount of information available, e.g. items/users with sparse data are aggressively "shrunk" to the global one. New items and users start out with profiles based on their known features that gets refined smoothly with the availability of more data. The model also supports dynamic updates, which gives more weight to recent data. Their proposal is a batched online learning scheme which updates the model on fixed time intervals or after a predetermined amount of new observations have been made.

Their model outperformed all other models on both the MovieLens and EachMovie datasets, and their dynamic model in particular significantly outperformed all other models.

Agarwal et. al [29] propose a Matrix factorization method to predict ratings in recommender system applications where a "bag-of-words" representation of item meta-data is natural. Their method regularizes both user and item factor simultaneously through user features and the bag of words associated with each item. The key idea of their method is to let user factors take values in an Euclidean space of existing factorization models, but assign item factors through a richer prior based on Latent Dirichlet Allocation (LDA). The main idea behind LDA is to attach a discrete latent factor to each word of an item that can take K different values (K-topics) and produce item topics by averaging the per-word topics in the item. An article where 80% of the words are assigned to politics and the rest to education would be though of as a political article related to the issue of education. This allows us to model the affinity between user i and item j as $s'j\hat{z}_j$, where $\hat{z}_j$ is the multinomial probability vector representing the soft cluster membership score of of item j to the K different latent topics.

Stern et al. [99] presents a probabilistic model called Matchbox. The system makes use of content information in the form of user and item meta-data in combination with

collaborative filtering information from previous user behaviour in order to predict the value of an item for a user. Much like [28] the factors are regularized by incorporating more flexibility in the Gaussian priors through regression on user and item factors. Their model is dynamic, meaning that it allows an item's popularity, a user's taste or user's personal rating scale to drift over time, as well as having the option to be trained incrementally using Assumed Density Filtering (ADF). This means that the value of weight matrices F and M will drift over time, this is accomplished by the addition of Gaussian noise each time step. Inference is accomplished a combination of message passing and expectation propagation.

The authors show that they can achieve state-of-the-art performance when training the model in an on-line manner, which is especially beneficial for dynamic domains where it is important to always have an up to date model. Matchbox was able to train the model for the Netflix Dataset in about 2 hours using 8 cores, meaning that it is able to add up to 14000 ratings per second. These methods also provide quick recommendations, which is important in an online applications, the system was able to generate 2,500,000 recommendations in 0.25 seconds using Approximate KD Trees.

Gantner et al. [50] propose a method on how to map additional information such as user and item features to the latent features of a matrix (or higher dimensional) factorization model. At the core of their approach is a standard factorization model, optimized to the recommendation task. The extensions include a mapping function that compute adequate latent representations for new entities from their attribute representations. This mapping function could allow new items and users latent features to be found only based on content-information and further on be used as if they were normally trained latent features. The training of the factorization model with a mapping extension consists of the following steps:

1. Training the factorization model using the data S, and then
2. Learning the mapping functions from the latent features of the entities

The authors use BPR-MF, a matrix factorization model based on the Bayesian Personalized Ranking (BPR) framework as their factorization model. The authors experimented with two different ways of mapping item/user attributes to the factor space (Only attribute-to-feature mapping for items are presented in the article):

1. k-NN Mapping: Weighted k-NN regression for each factor. Determine the k-nearest neighbors as the most similar items according to the cosine similarity of the attribute vectors.
2. Linear Mapping: Each item factor is expressed by a weighted sum of the item attributes. Suitable parameters for the mapping function is learned by optimizing the model for the squared error on latent features.

The authors found that linear mapping worked the best, and that their method yields accuracy comparable to state-of-the-art methods.

3.2.6 A Discussion on the Cold-start Solutions

Trust-aware recommenders

Massa et al. [78] found trusted users to be good predictors. When looking at directly trusted users they improved the MAE from 1.094 using traditional collaborative filtering

to 0.674 for cold-start recommendations. By propagating the trust they were able to drasticly increase the coverage. The average number of directly trusted users were 9.88, while the average number of comparable users using the pearson correlation factor was 160.73. Propagating at a distance of 2 it is possible to reach 399.89 users, increasing it to 3 and 4 respectively it is possible to reach respectively 4,386.32 and 16,033.94 users. Jamali et al. [60] got even better results with their *Trustwalker* approach by combining trust-based and item-based recommendations. Massa et al. [77] also argue that it is more useful for a recommender system to ask for one trust statement than asking for one rating for new users.

Requiring users to explicitly express trust, is not something users necessarily will frown upon. Services like Instagram, Facebook and many others offers a *follow* function to their users, filling their news feeds with content from the users which they have chosen to follow. For SoBazaar we imagine that you e.g. could chose to follow people either because they have a good taste in clothes or that you simply are friends, and you want to keep up with what your friends are buying. We imagine the *follow user* functionality, that has not yet been implemented, could be used to collect the trust statements. We believe that trust aware recommender systems is something that should be looked into at a later point when this functionality is in place, to further enhance the recommendation quality.

Propagating trust is expensive. The trust propagation must be computed in addition addition to the user-user or item-item similarities, and it therefore scales worse than collaborative filtering methods. It is however, a good general model for sparsity and increasing robustness of recommender systems, with the downsides being scalability challenges and the added complexity to the system.

Interview Process

Rashid et al. [90] got the best results using information theoretic approaches and argues that simpler methods such as most popular is likely to worsen the prefix bias. The authors found Information Gain through Clustered Neighbours (IGCN) to have the best performance overall, which scored 5 out of 5 stars for accuracy, and would be a good candidate to find items for the user to rate.

Our rational behind including these articles is that we envision a simple "hot or not" tinder like interface to be used to present items to new users when the first log in to the system. And then ask new users that download the app to rate e.g. 10 items when first logging in. It is worth mentioning that the authors of these articles mainly worked on a solution to the cold-start new user problem. The user-effort dimension of their evaluation could also largely be ignored as they made a system for movie recommendations. The implications of this is that a user must have watched a movie, in order to rate it. This is not as important for the fashion domain, as taking a quick look at an item should be sufficient to like/dislike it, so we should give more weight to the accuracy of the system after the interview process than user effort. It is also worth noting that calculating entropy using implicit ratings is tricky, since the rating distribution does not range from dislike to like. We can therefore not find *high-entropy* or *controversial* items which users either tend to like or dislike, as we have no data about items users dislike. We are also currently constrained to unobtrusively learn user-profiles from the natural interactions of users with the system, meaning that we can not require the user to rate e.g. 10 items before we can start providing recommendations, as this functionality has not been implemented.

Discussion on suitedness of implicit ratings for entropy calculations

.

The scalability of the approach is also fairly good. It requires another module in addition to collaborative filtering which is used in the non-personalized step until the user have rated a predetermined amount of items. When enough items have been rated the CF algorithm is used to produce recommendations. We really like this approach as it is simple and elegant. Given that a information theory approach is used this would be a good model for dealing with sparsity, as the number of ratings would sky-rocket in addition to having ratings for a large portion of the item collection (not only limited to the most popular items). The negative aspects of this approach is mainly limited to the fact that it requires active user involvement.

Seed users

In our opinion, this approach is not that suited for our domain, as it fairly dynamic and we are working with a large item collection. For an item to be recommendable it must be rated by all representative users, which is highly unlikely given the size of the item collection itself. E.g. if we have 15 representative users and a spring collection launches containing 6000 items, for all these items to be recommendable these 15 representative users must rate all these items.

Filterbots

Park et al. [85] clearly demonstrated the robustness of their Naive Filterbot compared to item-based and user-based approaches in all three cold-start scenarios. The results in [28, 29] also shows that the Naive Filterbots performance is very close to the state-of-the-art latent factor models.

To incorporate filterbots in our system we would first have to define what filterbots we wound want to use. We could e.g. use a Brand-bot that calculates ratings of brands over all users. The rating of a brand is the average rating of the items of the given brand, which then is injected into the user-item matrix. It is worth noting that selecting what bots to add to the system and and coding them would require some engineering effort, and involve some testing to validate your bots.

Park et al. [85] claim that the added computational complexity of adding seven global bots is almost negligible. The downside of this approach is the additional engineering effort required and the fact that it's performance is not on pair with the more sophisticated latent factor methods.

Hybrid recommenders

Latent factor models are currently the main paradigm within the recommender system field and are currently considered the state-of-the-art recommendation methods. The hybrid methods achieved state-of-the-art performance as well as having good fallback methods based on user and item features to solve the cold-start problem.

To implement these recommenders we would first have to select and extract user-features from Facebook and item-features from our item database. Another concern of ours is that our dataset is currently to small for latent factor methods, and is therefore likely to produce sub-optimal results.

As most latent factor models, model building is expensive. Matchbox and RLFM have to option of being trained online, which should further could increase the cold-start performance, as the model always will be up to date. Latent factor models are

also known to provide quick recommendations. These methods combine state-of-the-art performance, elegant solutions to the cold-start problem incorporating meta-information as fallback in addition to having the option to be incrementally trained.

Summary

It is hard to compare the performance of the different methods as they have experimented with different datasets and evaluation measures...

I would also argue that having the option to incrementally update the model is an important feature to further improve cold-start performance. As having a model that is already updated will instantly incorporate data about new users and items.

Compare the models based on the following properties:

- Accuracy: How accurate is the method...
- Cold-start performance: How well does it handle the cold-start related problems?
- Scalability: How well does the method scale for larger datasets
- User-effort: How much user involvement is required?

Method	Accuracy	Cold-start performance	Scalability	User-effort
Trust-aware RS				
Filterbots				
Seed users				
Intelligent selection				
Hybrid Methods				

Table 3.1: Evaluation of cold-start methods

We believe it would be interesting too see how the hybrid methods (RBLF) and Naive Filterbots could be combined with our implicit ratings to improve the cold-start performance of our system.

3.3 Fashion Recommendation

3.3.1 Theory

This subsection will look into some background research on fashion. And look into what fashion is and why a consumer behaves like the consumer does.

What is fashion

There are a lot of different ways of defining what fashion really is.

- The entire spectrum of attractive clothes styles at any given time - Anne Hollander
- Fashion is dress in which the key feature is rapid and continual changing of styles - Elisabeth Wilson

helge - I problem statement-delen br dere definere (uten bruke forelpig ikke definerte termer) hva som er nskede egenskaper, og prioriter disse. Ta igjen dette her. Col-start og user effort m vre mest sen-tralt forelpig eller? Holder gi kvalitativ vurder-ing (type +/- eller ++/+/-), s fravr av sammenlign-bare forsk virker ikke viktig p meg. Domenet er for spesielt til at vi kan generalisere ggodt tror jeg

What are the most important 'attributes' to consider? heri-notes: hvordan re-lateres dette til arbeidet deres?

- Fashion is usually first raised by a small group of people and then a trend is formed with more and more followers and copycats till it becomes outdated - Cheng & Huang
- The social norm recognized and advocated by a particular social class at one time. It affects all the fields in society, especially and famously in clothing. Sometimes, short-lived fashion is referred to as style - Fang Ma [75]

As seen from the different definitions mentioned above, what is reoccurring is; clothes, popularity, time and a cultural grouping. One of the main drives of fashion is the need and want for belonging, for the individuals to become a part of something bigger and sharing a common though or view. So fashion is what a **social group**, or a **set of groups**, recognizes and highly advocates **at any one time**. More generally, fashion is a popular style or practice, other categories such as music style and hairstyle can also be viewed as fashion, but in this case the focus will be directed towards clothing.

Task of fashion marketing

Fashion is subject to constant change as seen from the different definitions of fashion.

Some of these changes are due to human changes such as adoption of a new line of clothing, or something less controllable, such as the changing of the seasons.

How much of a product should be made to satisfy the need of the consumer, but still remain a desirable product the consumer would find itself unique and special with?

What is a reasonable price for the product, and how much is the name of the designer worth? Who can distribute the product without the product loosing its value and fashion status? These are just some of the questions the fashion industry has to answer. Without them answered the potential of the product is more difficult to reach. Which could lead the consumer not to feel the uniqueness and prestige of the item. Fashion trends comes and goes, and the new fashion starts with the refusal of what is old.

In fashion there is a big difference between men and women in what, where, when and how they buy. How to understand the behaviour of the consumers and how they act can come from a vast set of areas, the main factors influencing the consumer according to [68] is:

Factors	Examples
Physiological factors	Physical protection, commodity
Socio-cultural factors	Family, friends, work, social groups
Personal factors	Age, life cycle, occupation, personality
Psychological factors	Product reliance or sympathy
Rational factors	Brand of product, quality, designer, price

Table 3.2: Main factors influencing the consumer when it comes to their buying behaviour

When it comes to fashion it is mainly a socio-cultural phenomenon.

One central factor when it comes to shopping and fashion is price, a rational factor. The consumer acts rational, when it comes to price and quality [55]. In the case of fashion, and a product connected with prestige, this rational behavior might not apply.

There is a set of product criteria a consumer evaluates when it comes to the acquisition of a product [46], attributes found to have the most significant impact is styling, brand , price, place(store), fabrication/fiber content. The complete list is shown in 3.3

TODO: Fix some kind of left align centering og content

Concrete Attributes (Product Features)		Abstract Attributes (Attitude-Based)
Intrinsic (Hedonic)	Extrinsic	
Style	Price	Fun
Color	Brand	Entertainment
Patten	Country of origin	Enjoyment
Fabric/fiber	Place(Store)	Need
Appearance	Salespeson's evaluation	Function
Fashionability	Approval of others	
Durability	Coordination with wardrobe	
Comfort		
Quality		
Fit		
Care		

Table 3.3: The attributes effecting the consumer when in the process of consuming products [46]

The modern consumer finds pleasure with the consumption experience itself, not just the product, and this especially applies to the fashion domain. The purchase is often not done by need, but for pleasure.

The society nowadays is driven by consumption and change. Clothes lets the user claim a position of either respectability or outrageousness, economic and social values [32]. The clothes of the user is a way of telling the world who the user is. Fashion marketing starts and ends at the customer. The market must identify the way the consumer dresses himself or herself, and the product has to be produced according to the wants and needs of the user. Since this want and need of the user is ever-changing and changing faster and faster, focus must be given to the users actions and want.

More generally, the fashion marketing must answer the following questions [105]:

These are not really questions?

- Find consumer needs
- The most adequate consumer segment and how to approach
- Ideal positioning to reach this segment
- Design level, colors, quality that the target segment requires
- Price to establish
- Channel distribution demands
- Marketing strategies and policies that best suit the market segment

For the market to best answer the customers need, the market must have the best answers to the list above. This makes the domain of fashion more difficult to make recommendations for than many other domain, such as movie recommendations and music.

Consumer buying behavior

A lot of information about the consumers behavior is lost due to the reasons for their behavior is held in an unconscious or implicit level. The reason for a person is interested in a specific product could be based on some distant memory of the consumers life. This could affect how a consumer views a particular brand or product for better or worse. Brand choices are often made intuitively, based on their subconscious, and the consumer cannot tell why they made that specific choice [105].

Culture is one of the main factors to determine consumer behavior. Culture can be segmented into three parts: Culture, subculture and social class. All consumers are included in many smaller subcultures such as nationality, religious subcultures and geographical subcultures. Subcultures can be a efficient way of constructing marketing campaigns and aim similar products at, since they tend to form market segments. The forming of a subculture happens through individuals seeking out other individuals with similar tastes regarding a variety of aspects [105].

There are a lot of different behavior emerging from subcultures, such as peer pressure. Social psychology is used to understand the behavior of the individuals in subcultures [105].

The brand of the product might also greatly affect what the consumer buys and what the consumer does not buy. A study done on the behavior of the consumer [71] showed that knowing the brand of two almost identical products made the consumer crowd shift towards the more well known brand. Whereas before knowing the actual brand of the product, the crowd had a more equal distribution on the products.

Customer satisfaction

There are two main concepts when it comes to customer satisfaction: Transaction specific and cumulative specific. The transaction specific satisfaction of the consumer is base on the expectations in the pre-purchase stage and the perceived performance of the product in the post-purchase stage. Where the cumulative looks at the purchase as a whole, such as: the product, the purchase and the service received [69]. The transaction specific focuses on the post-purchase, if the expectations of the product during the pre-purchase is met in the post-purchase stage, the likeliness of a repeat purchase is increased.

3.3.2 Challenges

There is a set of challenges when it comes to making recommendations in the fashion domain compared to other domains.

Recentness of items

As seen from the different definitions of fashion, time is central in fashion. Therefore is time also central when it comes to making recommendations in this domain. What is of interest for the customer one month might not be of interest the next. The interest of the customer is not only affected by what is categorized as current fashion, but might be affected by other aspects, such as the current season. The recentness of an item and how long an item is of interest for a customer is greatly affected by the customers social groups, and the trend in this group.

How to use user feedback

The feedback from the user will mainly be implicit 4.2. As seen earlier in this section, it can be assumed that an increased interest in an item has some correlation with increased interaction with the item. To what degree this increased interest can be mapped to a more tangible feedback varies from customer to customer. How the item interaction feedback retrieved will be used is explored in section 4.2

Product semantics

The product database consist of items from multiple stores with multiple languages and multiple ways of labeling, describing and categorizing their products. This poses an issue for recommending items based on their content.

heri-notes: hvorfor er disse utfordringene relevante for oss?

3.3.3 Fashion in E-commerce

E-commerce is usually known as trading services and products via the Internet. There are many different services and products traded, such as fashion products. Fashion is one of the sectors in e-commerce increasing the most in many countries. In UK the online fashion sector has grown 258% in five years [45]. Three months after GANT [15] launched their new e-commerce solution, their orders online increased by 340% [8].

Previous sections has mentioned that a consumer is consuming for the satisfaction of consumption, which might not completely agree with the GANT example, but numbers from Statistics Norway [24] shows that fashion e-commerce is increasing at a high rate.

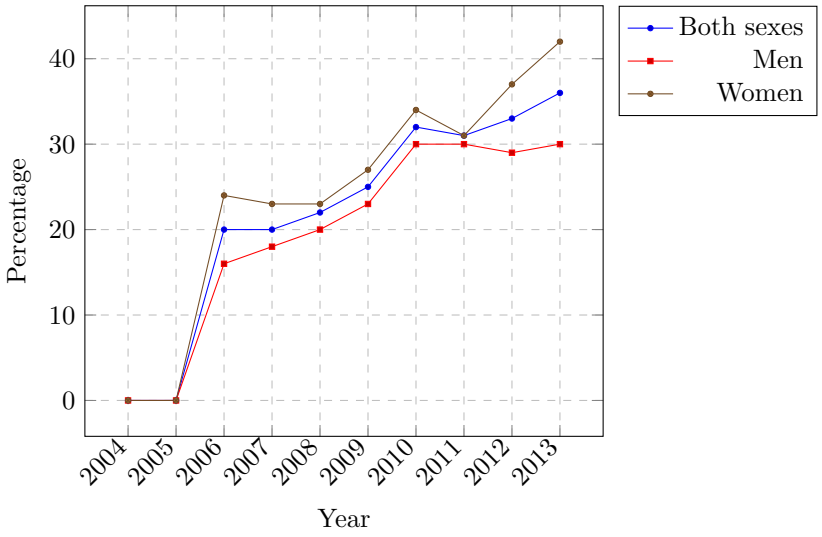


Figure 3.14: This figure is taken from [24] and shows the growth of online purchases through e-commerce applications in Norway

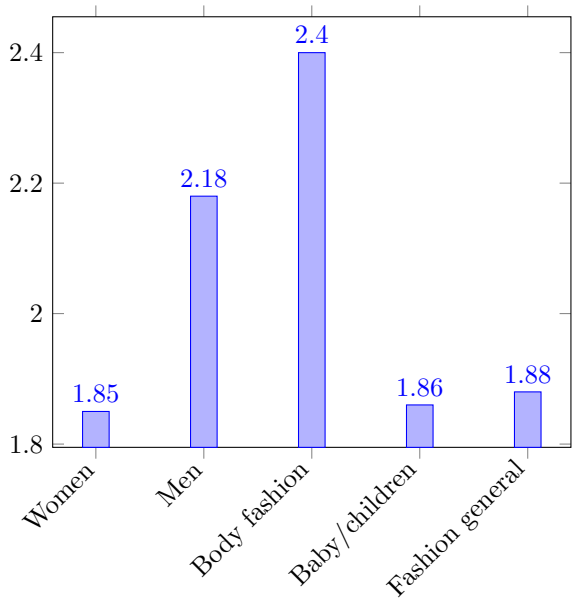


Figure 3.15: This figure is taken from [26] and shows the different values for conversion rate in the fashion domain. Conversion rate is the portion of users who visits a website, who goes beyond only browsing items and reaches some goal achievements. These goal achievements can be actions such as product purchase or advertisement interaction. The conversion rate is then $Conversionrate = \frac{\#ofGoalAchievments}{Visits}$ [81]. As seen from this figure, womens fashion has the lowest conversion rate in the different types of e-commerce fashion. The average conversion rate in e-commerce was around 3% in 2013 [81].

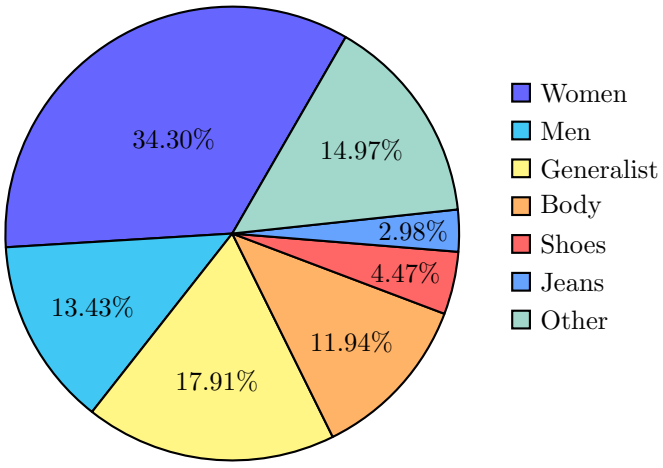


Figure 3.16: This figure is taken from [26] and shows how the different e-commerce fashion retailers focuses their products. This figure is based on 70 different fashion companies. It is clear that the large portion of the e-commerce fashion companies are putting their focus on women's clothing.

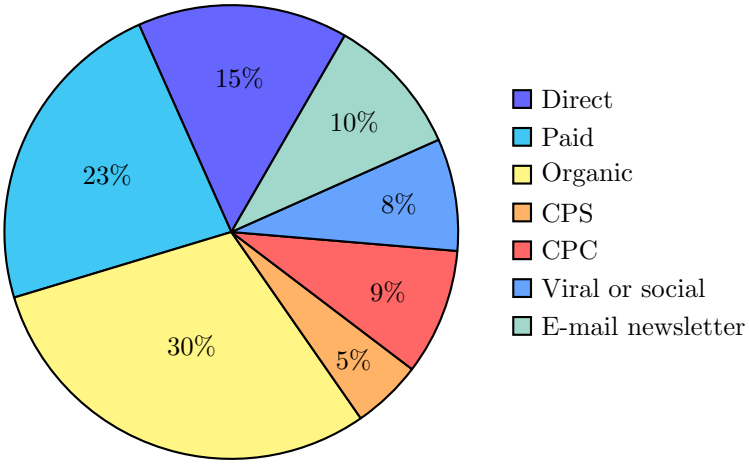


Figure 3.17: This figure is taken from [26] and shows where the users originated from in e-commerce fashion. Paid and Organic are search engine origins, and is 53% of traffic origin in e-commerce fashion. Direct traffic is at 15%, the average direct hit in e-commerce application is 22.3%. [26] explains this with the fact that the same products are often available on multiple e-commerce web sites, and the user browses for the best offer. Even though the viral/social origin segment is rather small (8%) the average for e-commerce companies is 4%. The social media is much more influential on e-commerce companies regarding fashion than other e-commerce companies.

3.3.4 SoBazaar, the e-commerce application

As briefly mentioned in the introduction chapter 1.1 SoBazaar is a fashion e-commerce application for web and hand held devices. The application is developed by Telenor and aggregates fashion products from various brands and stores, mainly fashion related. Users of the application can choose to log in with their social media account on Facebook [1] to store and share their fashion findings in the application. This allows the user to have one entry point to get the latest updates on fashion, and see what the user's social network is up in regards to fashion. When the user finds an item especially interesting and is interested in purchasing the item, the user is redirected to the store from which the product originated.

The fact that SoBazaar utilizes Facebook and a large set of fashion stores lets the users of the application gather at one place to find users with similar taste in fashion. As seen from 3.3.1 one important aspect in fashion is the feeling of belonging, which is made easy through the utilization of Facebook.

3.3.5 Competitors to SoBazaar

As seen from 3.3.3 SoBazaar is not the only e-commerce application for fashion, and has therefore some competitors. SoBazaar is built up of a collection of e-commerce store front applications and social interactions, but SoBazaar is not the first of its kind. There is a set of other similar applications providing the user with similar possibilities

TODO: Extend with Asos
TODO: Extend with Kwoller
TODO: Extend with Mallzee
TODO: Mention that many new apps have popped up recently, attempting to do the

[1] Facebook is an online social media service with around 1.28 billion monthly active users [12]

as SoBazaar. This section will be used to explore some of these systems, and look into their strengths and weaknesses regarding item recommendation for the user.

Myntra

"Myntra.com is a one stop shop for all your fashion and lifestyle needs" - about Myntra [4].

Myntra is one of India's largest e-commerce stores for fashion and aims to provide a hassle free shopping experience for the user. They aim to bring the newest and most in-season fashion products available to the user trough the web store front. The brand base of Myntra consists of 500 leading brands from both inside and outside India.

The web page uses a set of recommendation approaches to inform the user of what they might like, and to increase the user's awareness of different kinds of items. Such as, similar item and most popular.

Figure 3.18: In this figure we can see in red how Myntra is suggesting items which are similar to the item the user is currently looking at

Strengths	Weaknesses
Suggest similar items connected to the currently viewed	No personalized recommendations
Popular list for the different brands and stores	
Ability to add item to a "want list"	

Table 3.4: This table is the list of the recommendation related strengths and weaknesses of the e-commerce fashion web site Myntra [4]

Flink

"Flink is THE brand-new app to discover, get inspired and share trendy looks from top fashion bloggers" - About Flink [14].

Flink is a fashion discovery application for iPhone. It allows the user to browse fashion blogs, hot brands and new trends.

The content displayed can be "liked" and can be a collection of clothes from different brands. If the user is interested in the item, the application can redirect the user to the web page where it is sold.

Strengths	Weaknesses
Can follow other users	No personalized recommendations
Connect with facebook	
Ability to add item to a *want list*	

Table 3.5: This table is the list of the recommendation related strengths and weaknesses of the mobile fashion application Flink [14]

Lyst

"Lyst.com is a fashion shopping site that gives you your own shopping experience, so you can discover more of the fashion you love" - About Lyst [18].

Lyst offers items from thousands of the leading brands and stores of the world.

The site allows the user to follow different stores or brands to receive the latest items they have to offer. The user is given a personalized "stylefeed", which displays items from the different brands or stores the user is following. It is also possible for the user to add items to the their profile.

On first login the user is presented with a set of brands and store the user can like or dislike, to get the "stylefeed" started. On access of an item, the user is presented with related items, and the ability to add the item to a collection. When the user wants to buy an item, the site will redirect the user to the store selling the item.

Strengths	Weaknesses
Can follow brands and stores	Limited personalized recommendations
Connected with facebook	
Ability to add item to a "want list"	
"Stylfeed" based the user's follow list	
Show related items	

Table 3.6: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site Lyst [18]

Motilo

"Motilo was launched in 2011 to answer that perennial fashion dilemma all women face what shall I wear tonight?" - About Motilo [19].

Items on the web page are gathered by the Motilo stylists. This gives the page a fresh set of items for the user to select from.

Motilo gives the user the ability to put together item sets through dragging and dropping the items into a "fashion dilemma", or simply like items. The user can ask friends, the Motilo community or the Motilo stylists about suggestions regarding what to wear. If the user wants to buy an item, Motilo redirects the user to the page which sells the item in question.

Strengths	Weaknesses
Connected with facebook	Manual/limited personalized recommendations
Ability to add item to a "want list"	
A feed with the most trending item collections	
Ask Motilo stylists for suggestions	

Table 3.7: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site Motilo [19]

Farfetch

"farfetch.com forms the hub of a global fashion community that unites independent boutiques around the world with fashion lovers" - About Farfetch [2]

Farfetch is a collection of over 1000 boutiques from all over the world gathered on one web page. The user can shop directly on the page, and get the item delivered to the doorstep with only one checkout process.

When browsing an item the user is presented with a set of recommendations related to the current item, and previous browsing history. The item can be added to a "want list" or to the shopping chart.

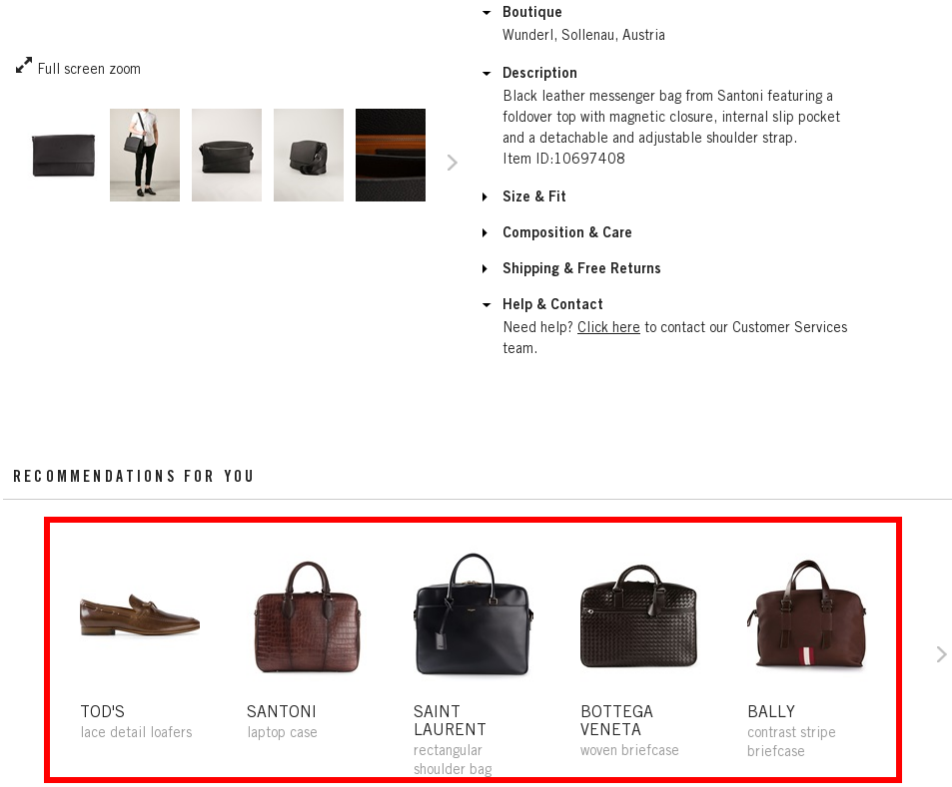


Figure 3.19: In this figure we can see in red how Farfetch is recommending items which might be of interest to the user. As we can see the first item is a shoe, which was the last item accessed by the user, the next four items are related to the currently viewed item

Strengths	Weaknesses
Ability to add item to a "want list"	No option to follow other users
A feed with the most popular items	
A feed with new items	
A list of recommendations for the user	

Table 3.8: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site Farfetch [2]

ModCloth

"A top e-retailer of indie clothing, accessories, and decor, and provide an engaging shopping experience where you, our customer, can have a voice" - About ModCloth [3]

ModCloth focuses on giving what the community is looking for. The user is given the opportunity to both be the seller and the buyer. The item base is affected by the user trough voting.

Strengths	Weaknesses
Ability to add item to a "want list"	No personalized recommendations
A feed with the most popular items	
A feed with new items	
A list of similar items	

Table 3.9: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site ModCloth [3]

UsTrendy

"UsTrendy allows you to shop and discover one-of-a-kind fashions from all over the world." - About UsTrendy [7]

UsTrendy has a large item database of more than hundred thousand unique items.

When the user is viewing an item, UsTrendy displays other items the user might like, which have common traits with the one the user is currently watching. The currently viewed item can be added to a sopping cart.

Strengths	Weaknesses
Ability to add item to a "want list"	No personalized recommendations
A feed with the most popular items	
A feed with new items	
A list of similar items	

Table 3.10: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site UsTrendy [7]

Polyvore

"Polyvore is a new way to discover and shop for things you love." - About Polyvore [5]

In Polyvore the user can put together sets of items and show them off to their friends and others. The items shown on Polyvore are gathered based on the community of Polyvore.

When accessing an item the user is shown similar items to the one which is currently being watched. When the user want to purchase an item, the user is redirected to the page which sells the item.

Strengths	Weaknesses
Ability to add item to a "want list"	No personalized recommendations
The user can follow other users	
Crawl other fashion sites to add to their item base	
A feed with trending items	
A list of recently viewed items	

Table 3.11: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site Polyvore [5]

Clothia

"An online destination where you can mix and match outfits, share looks you love, even try on clothes virtually via your webcam using augmented reality technology" - About Clothia [1]

The user can put together a set of clothes from the web site and make a "set". The set can be shared with other users and like by other users. If the user is interested in buying an item, the user is redirected to the page from which the item is sold.

Strengths	Weaknesses
Ability to add item to a "want list"	Lack personalized recommendations
The user can follow other users	
A feed with trending items	

Table 3.12: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site Clothia [1]

Trendabl

"Trendabl is a community of people who love fashion" - About Trendable [6]

The user is shown a feed with the newest items, and is free to browse different sets of collections, such as collections with shoes and pants. If the user wants to buy an item it can be added to a shopping chart.

Strengths	Weaknesses
Ability to add item to a "want list"	No personalized recommendations
The user can follow other users	
System recommends the top users in the system for the user to follow	

Table 3.13: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site Trendabl [6]

Rue La La

"Rue La La is the destination for the most desired brands" - About Rue La La [22]

Rue La La is a sale on site e-commerce web site. It is built up of a set of different boutiques, which can be browsed by the user. When the user is watching an item, Rue La La shows other items from the current boutique.

Strengths	Weaknesses
Ability to add item to a "want list"	No personalized recommendations
List of most popular items	

Table 3.14: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site Rue La La [22]

Zalando

"Clothes, accessories, sports items, beauty products" - About Zalando [25]

Zalando has a large set of items. When browsing an item the user is shown a set of similar items the user might also like, and a set of items which might "go well" with the currently viewed item.

Strengths	Weaknesses
Ability to add item to a "want list"	No personalized recommendations
Shop on site	
Similar items	

Table 3.15: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site Zalando [25]

Ellos [10]

Ellos is a e-commerce web site, which specializes in fashion.

When browsing an item, similar items to the one currently being watched is presented to the user. Other items which might go well with the item is also presented for the user.

Strengths	Weaknesses
Ability to add item to a "want list"	No personalized recommendations
Shop on site	
Similar items	
On site most popular list	
Items which might go well with the current	

Table 3.16: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site Ellos [10]

LookBook

"LOOKBOOK is the #1 source for fashion inspiration from real people around the world." - About LookBook [17]

LookBook is a leading online community which is centered around the looks of the users. The users can share their own looks and keep up with other users through watching their uploads. With over 1.2 million members LookBook is constantly up to date on the newest fashion trends.

Strengths	Weaknesses
Ability to add item to a "want list"	No personalized recommendations
Shop on site	
Similar items	
Most popular items list	
Hot items list	

Table 3.17: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site LookBook [17]

Fashiolista

"Let Fashiolista's community be your style guide in the online fashion jungle" - About Fasho [13]

The items on Fashiolista is selected by the users of Fashiolista, and the sites is therefore customized to fit the user crowd's wishes and interests.

When accessing an item, the user is presented with the item, and a set of other items from the store the current item originated from. Other users who liked the item are also shown, so the user can browse their personal want list.

Strengths	Weaknesses
Ability to add item to a "want list"	No personalized recommendations
On site most popular list	

Table 3.18: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site Fashiolista [13]

ShopStyle

"POPSUGAR is a global women's lifestyle brand focused in media, commerce, and technology" - About ShopStyle [23]

ShopStyle is a commerce brand of POPSUGAR. ShopStyle displays items from other e-commerce web sites, and redirects the user directly to the e-commerce web site from which the item clicked originated. Items can be liked on ShopStyle, and viewed in less detail at the web page of ShopStyle.

Strengths	Weaknesses
Ability to add item to a "want list"	No personalized recommendations
Similar items	
Editor's picks	

Table 3.19: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site ShopStyle [23]

MyHabit

"MyHabit is a private fashion sale site offering up to 60

MyHabit was founded by Amazon in response to the desire from the users of Amazon to shop fashion in an easy manner.

The site displays a feed. This feed is fed by a team from MyHabit, and is constantly updating with new sales and new products. The items put into the feed are handpicked.

When browsing items on MyHabit, other similar items are suggested to the user.

Strengths	Weaknesses
Shop on site	No personalized recommendations
Similar items	

Table 3.20: This table is the list of the recommendation related strengths and weaknesses of e-commerce fashion web site MyHabit [21]

Competitors Recommendation Overview

In table 3.21 we see that there is a very low count of fashion related e-commerce applications, which actually produces personalized recommendations for their users.

Most of applications are taking a simpler approach when making recommendations for the user, like most popular or similar items. There is no obvious relation between recommendations and that the site has a form of "want list", but the system which allows the users to follow each other are usually not in-application-purchase-applications.

There where no indications of the "want list" being used directly to some personalized recommendations and neither was it any indication that the "follow list" of other users helped produce any personalized recommendations, other than recommending items from the followed user's feed. The "follow list" was also in some cases used to suggest other users to follow. The "want list" was primarily there so that the user could go back to a liked item, and maybe interact with it later.

Competitor	In App Purchase	Most Popular	Similar Items	Want List	Follow Other Users	Personalized Recommendations
Myntra	✔	✔	✔	✔	✗	✗
Flink	✗	✔	??	✔	✔	✗
Lyst	✗	✔	✔	✔	✔	✗
Motilo	✗	✔	✗	✔	✔	✗
Farfetch	✔	✔	✔	✔	✗	✗/✔ [2]
ModCloth	✔	✔	✔	✔	✗	✗
UsTrendy	✔	✔	✔	✔	✗	✗
Polyvore	✗	✔	✔	✔	✔	✗
Clothia	✗	✔	✔	✔	✔	✗
Trendabl	✔	✔	✔	✔	✔	✗
Zalando	✔	✔	✔	✔	✗	✗
Ellos	✔	✔	✔	✔	✗	✗
LookBook	✗	✔	✔	✔	✔	✗
Fahsiolista	✗	✔	✗	✔	✔	✗
ShopStyle	✗	✗	✔	✔	✗	✗
MyHabit	✔	✗	✔	✗	✗	✗

Table 3.21: This table is the list of the different properties of some of the different competitors to SoBazaar. The properties are in regards of their recommendation ability, and how they let their user expand their item set

3.3.6 Fashion Recommender Systems

This subsection will look at different methods other fashion related systems have used to recommend fashion related products to the user.

Photograph based approach

Fashion and the products it regards are highly dependent on visuals. A fashion product would not be very interesting if no one saw it. An approach to use the importance of how

[2] How the recommendations are produced is not mentioned

the product looks regarding recommending is to utilize images of the product. Fashion Coordinates Recommender System Using Photographs from Fashion Magazines [58] is a system doing this. They teach their system by using fashion magazines with full body images. They segment the image into two parts, top and bottom. From this the system learns which top matches to which bottom and collects visual features of the products. From this the system can recommend other tops to go with a selected bottom, or other way around. The proposed system scored better[3] than both a more naive approach and a random selection. Runtime was at 0.04 seconds per recommendation.

Hot-or-not

A recommender system called SuitUp [64] did a survey on some of their potential users. One interesting finding was that many of the users enjoyed the Hot-or-Not feature of the system. This feature gives the user a set of items and the option to either like or dislike. This did not only make the participants in the survey more engaged in the system, but also produced ratings, both negative and positive, for the system. For cold-start users and in a cold-start system this extra information and ratings make it much easier to make recommendations for the users.

Scenario-Oriented Recommendation

Shen et al. [96] proposed a recommender system which produce recommendations not only based on the metadata of the products, but also on user written input. Knowledge used to handle the user input, is derived from Open Mind Common Sense [98].

The user uploads his or her clothes and adds brand (e.g.Nike), type (e.g.jeans), material and a description about the item (e.g."I put these on when I get home"). The systems makes recommendations based on the scenario the user is needing help to find suiting clothes to wear. The typical use case of the system is when a user is unsure about what to wear under different circumstances, but knows something about the scenario or occasion the clothes will be worn in. For instance: "I am going to the beach". It is also possible to interact with other users and the system relates similar users.

The different describing fields about the items are given a six-tuple style value. The six tuples are: luxurious, formal, funky, elegant, trendy and sporty, where each is given a value from 0 to 10 based on how much the describing field of the items is the current tuple. The different describing fields are given a default value, which can be changed by the user if that this necessary. But it seems like this has to be done for all brands, material and type manually to initialize the system.

This is an interesting approach to fashion and clothes recommendations but the need for user scenario input and a six-tuple description of the different describing fields, might not be desirable for the user or the system administrator in the long run.

Yu-Chu et al. [108] and Ying et al. [107] are other similar systems. Ying et al. [107] made a recommender based on the a similar concept, namely, what to wear in different situations. The system recommends two sets of top-to-toe clothing based on the current season, the occasion and the items the user has uploaded. The uploaded items must be given a set of descriptions, including the occasion to use the item.

To recommend, the system uses the user profile of the user, which describes the interests of the user and the information given by the user about the different items in his or hers wardrobe.

[3] Accuracy of 50% on the top 5 suggested items, whereas naive and random managed 18% and under 5% respectively

Photograph Recommendation Integrated with Occasion

Liu et al. [74] combined the two approaches from Iwata et al. [58] and Ying et al. [107] and Shen et al. [96] and suggested a system which recommend clothes both based on the photographies and the occasion the clothes are to be worn.

They do this by incorporating fashion rules like, what can you wear to which occasion, and what can you wear as a complete set to different occasions. The recommender learn the clothing recommendations through a latent support vector machine framework. They use this framework to match four potential functions: visual features vs. attribute, visual features vs. occasion, attributes vs. occasion and attribute vs. attribute. These are used together in a scoring function for clothing recommendation.

Their system preformed better than the baselines, but was highly dependent on the human detection accuracy, since the learner was learning from fashion photographs.

3.4 Sessions

Different ways of personalizing recommendations, such as collaborative filtering and content based filtering, have been explored in 3.1. A common issue with these types of recommendations is that when the data is sparse the recommendations suffer. Through utilizing the usage patterns from the users to construct sessions, some of the sparsity gaps can be filled. The patterns can help to understand the pre-purchase patterns, products they see and what they buy. This information can also reveal interesting relationships between products.

The data from SoBazaar is based on the actions of its users. From this user sessions can be constructed. A user session is the sequence of events from the time the user opens the application till the user closes the application, a user will therefore often end up with producing many sessions.

This section will look into how sessions can be used to tell more about the users and approaches to improve product recommendations based on the user sessions.

3.4.1 Mining

Data mining can be split into two main steps [39]:

- *Data Preparation* - In the data preparation step the sessions are built. They are built based on the usage during a single visit to the application, web page or system the user is accessing. It is the sequence of events produced by the user actions during the visit.
- *Pattern Discovery* - In the pattern discovery step the system use the sessions gathered in the previous step to discover association rules, sequential patterns, usage clusters, page clusters, user classification or any other pattern discovery method.

Data Preparation

When a user accesses an application or web page, the system can store the footprints of the user. Within these footprints the system can embed information such as, which browser used by user, location of the user, event triggered by user and time stamp. This bundle of information is called an event, a the set of events produced by the user during an application visit or web page visit, is called a session.

Pattern Discovery

Pattern discovery can be done after the data has been collected from the events. The patterns discovered can help the system to infer relationships between users and items.

Association Rules The association rule describes how item X and Y are coupled, this coupling is the percentage of transactions that contain both item X and item Y.

One simple way of discovering this association is through the aprior algorithm [30].

With this set of rules for the items, item taxonomies can be used to infer knowledge regarding the items. For instance; sessions when buying blue skirts often include buying of red shoes. More generally the association rules would allow the system to make assumptions regarding which items to suggest in context of other items.

Sequential Patterns The sequential patterns can be used to build Markov models to predict the next step the user will take based on the sequential pattern of the events from the sessions [44].

An issue with using the Markov model is that if first-order is being used, the accuracy can be faulty, since the model will not be looking far enough behind. Handling this with an higher-order model might solve the inaccuracy due to too low order, but might cause an issue with high state space complexity, and might also reduce coverage. Which can be handled through training a set of K different Markov models, with K different orders, All-K^{th}-Order Markov model, but the space complexity issue will still be present.

The Markov model predicts future actions through looking at past actions. So for first-order, the model looks at the last event performed to predict the next event of the user. The higher the order, the further back the model is looking in the event set and will produce a more complicated model, the higher the order. This can be handled through the usage of a selective Markov model [44]

One issue with the Markov model is that a user will not get predicted new items if the sessions used to predict future actions are only based on the own users sessions. One way of making use of the patterns and producing new recommendations is trough matching sequential patterns from users with one-another.

something not right

Usage Clusters The usage clusters are usually built up from the different features from the events in the session, where the features are collected and used to model the user.

Event features
Amount of purchases
Amount of views
Amount of wants
Time spent on items
Overall average time on items
Click to buy rate
Search query [109]
What was done before a specific action

Table 3.22: Set of event features used for finding usage clusters and model the users behavior

Page Clusters Page clustering will help the system to understand which pages belong together. For instance when searching for Harry Potter, the user might have interest in knowing more about the author J.K. Rowling, and the system might benefit from clustering queries of here with Harry Potter [109].

3.4.2 Analyzing

Analyzing the model of the user

3.4.3 Recommending

Using the model to help personalizing the recommendation for the user

3.4.4 SoBazaar Session Examples

A user session of a user randomly chosen user.

Show three common user session patterns (average purchase time, want time and view time) Will be updated with the new data from SoBazaar


```
{ "event_id" : "app_started", "product_id" : "NULL", "ts" : NumberLong("1382689141084") }
{ "event_id" : "storefront_clicked", "product_id" : "NULL", "ts" : NumberLong("1382689144152") }
{ "event_id" : "storefront_clicked", "product_id" : "NULL", "ts" : NumberLong("1382689172026") }
{ "event_id" : "storefront_clicked", "product_id" : "NULL", "ts" : NumberLong("1382689179152") }
{ "event_id" : "product_detail_clicked", "product_id" : 2028030, "ts" : NumberLong("1382689192035") }
{ "event_id" : "product_purchase_intended", "product_id" : 2028030, "ts" : NumberLong("1382689197749") }
{ "event_id" : "product_detail_clicked", "product_id" : 2038024, "ts" : NumberLong("1382689263384") }
{ "event_id" : "storefront_clicked", "product_id" : "NULL", "ts" : NumberLong("1382689284693") }
{ "event_id" : "storefront_clicked", "product_id" : "NULL", "ts" : NumberLong("1382689297864") }
{ "event_id" : "storefront_clicked", "product_id" : "NULL", "ts" : NumberLong("1382689322201") }
{ "event_id" : "storefront_clicked", "product_id" : "NULL", "ts" : NumberLong("1382689326804") }
{ "event_id" : "storefront_clicked", "product_id" : "NULL", "ts" : NumberLong("1382689352841") }
{ "event_id" : "product_detail_clicked", "product_id" : 2028032, "ts" : NumberLong("1382689357563") }
{ "event_id" : "product_detail_clicked", "product_id" : 2038025, "ts" : NumberLong("1382689363957") }
```


purchase time: 14068.69687 want time: 17859.1539

3.4.5 Discussion

Limitations

What are the limitations of a session based approach to recommendations.

Future

Time spent and scrolling on page have shown to suggest strong positive relationship with explicit interest [41].

Relative preferences derived from clicks are reasonably accurate [62]

3.5 Evaluation

In most cases a system designer that wish to employ a recommender system must choose between a set of candidate approaches. A first step towards selecting an appropriate algorithm is to decide which properties are the most important for the application. Recommender systems have a variety of properties such as accuracy, robustness, scalability, etc. The following section will discuss how to compare recommender systems based on the set of properties relevant for the application and how to evaluate the performance of the system. We have several different experimental settings which can be used to

evaluate recommender systems. We will focus on two types of experiments: Offline experiments where the system is evaluated without user interactions and Online experiments where real users interact with the system. This section will also cover some of the most frequently used evaluation metrics, a discussion on the suitedness of different evaluation measures for implicit feedback, and summary of the methodologies and evaluation measures used to evaluate the cold-start performance of recommender systems.

Shani et al. [95] lists the following guidelines for general experimental studies:

- Hypothesis: before running an experiment one must form an hypothesis. For example, an hypothesis can be that algorithm A better predicts user ratings than algorithm B. In that case the experiment should test the prediction accuracy, and not look at other factors.
- Controlling variables: when comparing a few candidate algorithms on a certain hypothesis, it is important that all variables that are not tested are fixed.
- Generalization power: when drawing conclusions from experiments, we may wish that our conclusions generalize beyond the immediate context of the experiments. When choosing an algorithm for a real application, we may want our conclusion to hold on the deployed system, and generalize beyond the experimental data set. To increase the probability of generalization of the recommender results one must typically experiment with several data sets or applications.

3.5.1 Offline Evaluation

Offline experiments are performed using pre-collected dataset*s* and a protocol that models the user behavior to estimate recommender performance through different evaluation measures. Offline experiments are attractive because they require no interactions with real users, and thus allows multiple researchers to compare a wide range of algorithms, using the same data, at a low cost. The downside of offline experiments is that they can answer a very narrow set of questions, typically questions about the predictive power of the algorithm, and does not measure other user factors.

Datasets for Offline Evaluation

The main aim of a recommender system is to identify the set of items in a dataset that might be interesting to a user based on their expressed preferences. For a fashion recommender this would mean estimating how much a user might like an item, by e.g. predicting what rating a user might give an item. In recent years, various test collections for different domains such as books, music, movies have been made available to the public. These datasets usually consist of user ratings in the form of $(UserID, ItemID, Rating)$, but may also include a timestamp.

In recent years more or more datasets have been made available which contains additional information such as demographic information about the users, trust-networks, user-assigned tags and etc. Under we have listed a few selected popular datasets containing additional information:

- MovieLens 100k dataset [20]: The movielens dataset incorporates demographic information about the user in addition the traditional rating matrix

- Epinions dataset [11]: The Epinions dataset includes a trust-network, which specifices who-trust-whom in a social network based on customer reviews for the website Epinions.com
- The Million Song Dataset [35]: The million song dataset is a implicit feedback dataset. This data also includes information on the users, and audio features and song meta-data.
- The Book-Crossing Dataset: The bookcrossing dataset consists of both implicit and explicit feedback, demographic information about users and some content based information about the books.

Explicit-feedback The definition of explicit is defined as 'stated clearly and in detail, leaving no room for confusion or doubt'. Explicit feedback are more precise than implicit feedback, but more difficult to collect since it requires active user involvement. One serious implication of this is that the amount of feedback often is scarce since many users opt not to provide any feedback. Explicit feedback mechanisms allow the users to unequivocally express their ratings on a scale (usually in the form of a Likert scale (strongly disagree strongly agree). Thus explicit feedback is able to capture both negative and positive feedback, while implicit feedback *only* can be positive. It is worth noting that explicit feedback tend to concentrate on either side of the rating scale, as users are more likely to express their preference if they feel strongly for or against an item [61].

Implicit-feedback Unlike explicit feedback, we do not have any direct input from the user regarding their personal preferences. In particular we do not have any substantial evidence of which items the user dislikes, such as low ratings. What we do have is indications of whether a user likes an item trough clicks, wants and purchases. Where wants and purchases can be viewed as a form of explicit rating only counting positively. Implicit-feedback is more easily collected than explicit-feedback, and usually more abundant. Types of implicit feedback include purchase history, browsing history, search patters, or even mouse movements. For example, a user who purchases many clothes from the same brand probably likes that brand. For a larger discussion surrounding the differences between implicit and explicit feedback, see Section 4.1.2

Validation Methods

Validation techniques are motivated by two fundamental problems; model selection and performance estimation. Almost all pattern recognition techniques have one or more free parameters, and we want a way to select the *optimal* parameter*s* or model for a given problem. Once we have chosen a model, we wish to estimate how well it is doing.

The Holdout Method When using the holdout method you split the dataset into two groups; a training set used to train the classifier and a test set used to estimate the error rate of the trained classifier. The Netflix Prize Competition [34] provided a training set consisting of 100,480,507 ratings given by 480,189 users to 17,300 movies. The testset consisted of 2,817,131 items where the ratings were unknown. The submitted predictions were scored against the true ratings in terms of root mean squared error (RMSE), and the goal was to minimize this error.

The holdout method has two basic drawbacks; (1) In problems with sparse datasets we may not be able to afford the luxury of setting aside a portion of the dataset for testing, (2) Since it is a single train-and-test experiment, the holdout estimate can be misleading if we happen to get an 'unfortunate' split.

Cross-Validation K-fold Cross-Validation creates a K-fold partition of the dataset. For each of the K experiments, use $K-1$ folds for training and the remaining one for testing. By setting $K = 2$, this is the same as the holdout method. The estimated error is found by taking the average error from all the experiments.

Leave-one-out is the degenerate case of K-fold Cross-Validation, where K is chosen as the total number of examples. For a dataset with N examples we perform N experiments. For each experiment we use $N-1$ examples for training and the remaining example for testing. Again, the true error is found by taking the average error rate from the experiments.

In practice the number of folds often depends on the size of the dataset. For large datasets, even 3-Fold Cross Validation will be quite accurate, which for sparse datasets, one may wish to train as many examples as possible. A common choice is K value between 5 and 10.

Advantages One positive aspect with k-fold cross-validation is that the result is averaged over the K experiments. The strongest argument for using cross-validation is the potential of using the entire training set for testing (albeit not at once), creating the largest possible test set for a fixed training data set.

Disadvantages The main disadvantage with K-fold Cross-Validation is the training algorithm has to be run K times, and thereby increasing the runtime for producing an evaluation of the system. If the dataset is large, an increase of K runs could prove to be disadvantageous.

Helge: Fler at bootstrapping er p et annet niv enn holdout og cross-validation. Passer det, og passer det her? Fix

Bootstrapping [47] Bootstrapping is used to estimate properties of an estimate, such as bias, variance, confidence, intervals and prediction error. This is done trough measuring these properties when sampling from an approximated distribution, creating an empirical distribution of the observed data. Since the entire population is unknown it will not be possible to calculate the true error from the sample data. The idea then is that with this information from sample data it is possible to say something about the population. Since it will not be possible to perform inference on:

sample data $\rightarrow$ *population* this is modeled as: *re-sample data* $\rightarrow$ *sample data*. The *re-sample data* is a re-sampling of the sample data.

In practice an example to bootstrapping is when we want to calculate the average height of the population worldwide. The issue here is that it is not as doable to measure everyone, so a subset of the population is used. Since the average on this number only will be an estimate of the actual world wide average a sense of error margin must be introduced. Bootstrapping is then used to reduce the error of margin through re-sampling the sample data a large number of times [4], and new averages are calculated. With these averages a histogram can be produced, which provides an estimate of the distribution of this average.

[4] Numbers vary on sample size, but is often 1 000 to 10 000

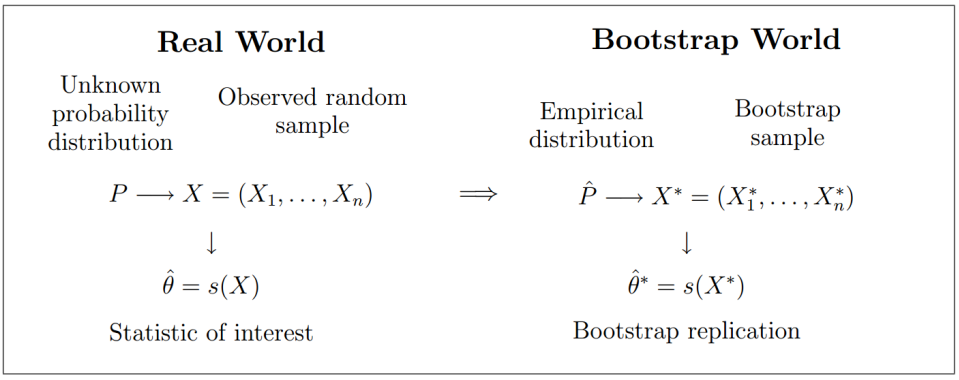


Figure 3.20: A figure of the principles bootstrapping. Taken from [48]

Disadvantages One prominent issue with bootstrapping is that important properties of the actual data might not be caught when undertaking the bootstrapping analysis [5].

Finding the optimal parameter settings Finding the optimal parameters for a model is usually a crucial task in engineering approaches to classification and modeling tasks. An automated approach is particularly desirable when the number of parameters is high. In order to work well many algorithms and modeling techniques in computer science rely on a careful choice of their parameters. The usual approach is to select parameters based on some prior experience on the problem at hand with a limited heuristic search of possibly optimal parameters. When such prior knowledge is absent or cannot be directly applied to the technique being used, the optimal parameters have to be found by *blind* search, by e.g. using genetic algorithms or some kind of exhaustive grid-search procedure.

The *de facto* standard way of performing model selection optimization is grid search, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm. To *guide* the grid-search one typically uses a performance metric, typically measured by cross-validation on the training set. After evaluating the performance all the pre-specified combinations of parameter settings, the grid search algorithm outputs the setting that achieved the highest score in the validation procedure.

Many of the most popular Machine Learning libraries now come with methods for performing grid search.

Offline Evaluation Metrics

When evaluating a recommender system, you wish to estimate a user's satisfaction for a given recommendation. Traditionally recommender systems have been evaluated by means of predictive accuracy. However, there is now a widely agreed that accurate predictions are crucial but insufficient to deploy a good recommendation engine [95, 79]. Some of the properties can be traded-off, one such example is the trade-off between accuracy and diversity. It is important to understand and evaluate these trade-offs and

[5] "Bootstrapping" comes from the phrase, "to pull oneself up by one's bootstraps" [49]

their effect on the overall performance. This subsection will cover the most popular metrics used for offline evaluation, a discussion of evaluation measures for implicit feedback, and a summary of the cold-start evaluation methodologies found in the literature.

Predictive Accuracy Metrics Predictive accuracy metrics measure how close the predicted ratings are to the true user ratings. More formally, the system tries to predict ratings $r(\hat{c}, i)$ for a test set T of user-item pairs (c, i) for which the true ratings are known. Traditionally, mean absolute error (MAE) has be used to evaluate the performance of collaborative-filtering algorithms, but other measures such as root mean squared error (RMSE) are also commonly used.

Mean Absolute Error (MAE) MAE measures how close the predictions are to the actual outcome.

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |r(\hat{c}, i) - r(c, i)| \tag{3.12}$$

$r(c, i)$ is the actual outcome and $\hat{r}(c, i)$ is the predicted value. As the name suggest, MAE calculates the average absolute error.

Root Mean Squared Error (RMSE) Often used to measure difference between a set of predicted values with a set of actual values.

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (r(\hat{c}, i) - r(c, i))^2 \tag{3.13}$$

$$RMSE = \sqrt{MSE} \tag{3.14}$$

In MSE 3.13 $r(\hat{c}, i)$ is the predicted value and $r(c, i)$ is the actual value. Both MAE and MSE are used to measure how correct the predictions are compared to the actual values. $RMSE$ is the square root of MSE and is one of the most used metrics to compare recommender algorithms in collaborative filtering, and was the main metric used in the Netflix price competition to evaluate the performance of the competitors recommender systems. $RMSE$ is always bigger or equal to MAE, $RMSE$ penalize an error more than MAE.

Measuring Usage Prediction In many applications the recommender system does not predict the user's preferences of items, but tries to recommend to users items that they may use. This is often done by giving the user a top-K set of recommendations. In an offline evaluation of usage prediction, we typically have a dataset consisting of items each user has used. We then select a test user, hide some of her selections, and ask the recommender to predict a set of items the user will use. We then have four possible outcomes for the recommend and hidden items.

	Relevant	Not Relevant
Recommended	True-Positive (TP)	False-Positive (FP)
Not Recommended	False-Negative (FN)	True-Negative (TN)

Table 3.23: This table is showing the different categories recommended items can end up in.

True-Positive (TP)	The recommended item is of interest to the user
False-Positive (FP)	The recommended item is not of interest to the user
False-Negative (FN)	The item is of interest to the user, but is not recommended
True-Negative (TN)	The item is not of interest to the user, but is not recommended

Table 3.24

This model assumes that not relevant items would not have been relevant if they had been recommended to a user. This assumption may be false, such as when the set of not relevant items contains some interesting items that the user did not select. For example, a user may not have relevant an items because she was unaware of its existence, but after the recommendation exposed that item, the user can decide to select it. We can count the number of examples that fall into each cell in the table and compute the Precision, Recall, Fallout and *ROC*.

Precision Precision is the fraction of retrieved items that are relevant.

$$Precision = \frac{TP}{TP + FP} \tag{3.15}$$

Precision takes all recommended items into account, but it can also be evaluated at a given cut-off point, only considering the top n results returned. This measure is called precision at n or P@n.

Recall Recall is the fraction of the items that are relevant to that are successfully recommended.

$$Recall = \frac{TP}{TP + FN} \tag{3.16}$$

Recall can therefore be seen as the probability that a relevant item is retrieved by the recommender.

Fallout Fallout is the amount of retrieved items which is not relevant amongst all the non relevant items (false positive).

$$Fallout = \frac{FP}{FP + TN} \tag{3.17}$$

Fallout can therefore be looked at as the probability that a non-relevant item is recommended.

F-measure F-measure combines the precision and the recall.

$$F_\beta = \frac{(1+\beta^2) * (Precision * Recall)}{(\beta^2 * Precision + Recall)} \tag{3.18}$$

Based on the value of β F-measure will weight precision or recall more. For a β over 1 F-measure will emphasize precision over recall, and opposite for β between 0 and 1.

Accuracy Accuracy is the amount of correctly recommended items over all the items.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.19}$$

Receiver Operating Characteristics (ROC) The ROC is the recall rate (TPR) against the fallout rate (FPR). The goal is to maximize the recall while minimizing the fallout.

$$TPR(T) = \int_T^\infty P_0(T)dT \tag{3.20}$$

$$FPR(T) = \int_T^\infty P_1(T)dT \tag{3.21}$$

T is a threshold parameter. The ROC curve is TPR plotted together with FPR at various T.

$$AUROC = \int_\infty^{-\infty} TPR(T)P_0(T)dTdT \tag{3.22}$$

Equation 3.22 can be used to calculate the area under the curve. $AUROC$ is the probability that the recommender system will rank positive examples higher than negative examples. The Area Under is a commonly used evaluation method for binary choice problems. If somebody makes random guesses, the ROC curve should be a diagonal line stretching from (0,0) to (1,1), as shown by the blue line in Figure ??, scoring an AUC of 0.5. A perfect model will score an AUC of 1.0. In practice, almost all models will fit somewhere in between.

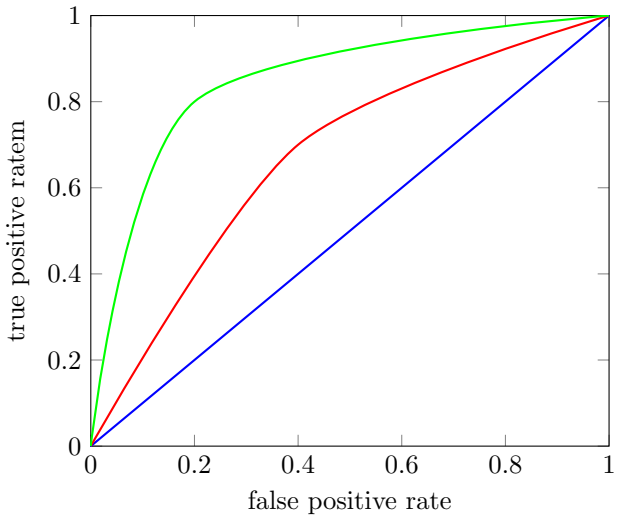


Figure 3.21: ROC curves

Limitations As mentioned by Powers et al. [87], recall, precision, F-measure have a bias. Recall, precision and F-measure ignore performance in correctly handling negative examples, they propagate the marginal Prevalences and biases, and they fail to take account the chance level performance. Another drawback or limitation with the measuring of user predictions is that it does not take into account the ranking of the items. To handle this, rank based metrics can be used.

Rank Based Metrics Rank accuracy metrics measure the ability of a recommendation method to produce a recommended ordering of items that matches how the user would have ordered the same items. Shani et al. [95] lists two different approaches for measuring the ranking accuracy: Try determining the correct order of a set of items for each user and measure how close a system comes to this correct order, or we can attempt to measure the utility of the system's ranking to a user.

Herlocker et al. [56] argue that rank accuracy metrics may be overly sensitive for domains where the user just wants an item that is 'good enough' (binary preferences) since the user won't be concerned about the ordering of items beyond the binary classification. These metrics are therefore most suitable to evaluate algorithms that are used to present ranked lists to the user in domains where the user preferences are expressed using numerical values.

AP correlation *APcorrelation* [106] measures the overall precision and is a variant of *Kendall's tau*. It counts the amount of items correctly placed in a ordered predicted rank list *list*1 and a list of the actual rank ordering of the preferences of the user *list*2.

$$AP = \frac{2}{N-1} * \sum_{i=2}^{N} (\frac{C(i)}{i-1}) - 1 \tag{3.23}$$

How to calculate the *APcorrelation* value is shown in 3.23. $C(i)$ is the number of items ranked correctly above rank i. The value of AP is between -1 and 1, where a score of 0

means that $list1$ can be considered a randomly generated list and 1 is a perfect match with the actual list $list2$.

if this is the setup, write a little intro

Mean Percentage Ranking (MPR) This measure is a recall-oriented metric. A known issue with implicit feedback is that it often lack the actual user's preference. This approach is used to measure the user satisfaction of items in an recommended ordered list.

$$MPR = \frac{\sum_{u,i} r_{ui} * rank_{ui}}{\sum_{u,i} r_{ui}} \tag{3.24}$$

How to calculate MPR is shown in 3.24. A list of all the items for user u is ordered based on the rank $rank_{ui}$ of the u. Where $rank_{ui}$ is the percentile rank of item i in this list for u. $rank_{ui} = 0$ means that i is the most preferred item for u. r_{ui} indicates whether u has consumed i or not. This makes a MPR value of 0% to be the most preferred value, and a value of 50% meaning a near randomly produced list.

Mean Average Precision (MAP) MAP [76] measures quality across recall levels.

$$ap@n = \sum_{k=1}^{n} \frac{P(K)}{min(m,n)} \tag{3.25}$$

$$MAP@n = \sum_{i=1}^{N} \frac{ap@n_i}{N} \tag{3.26}$$

How to calculate the MAP value is shown in 3.26. 3.25 calculates the average precision at n for a user. From 3.25, $P(K)$ is the precision at k in the item list, n is the maximum number of predicted items and m is the actual length of the predicted items list. 3.26 calculates the mean of all the values from 3.25.

Normalized Discounted Cumulative Gain (nDCG) $nDCG$ measures the graded relevance of the recommended item, the ranking quality or the usefulness of the recommended item based on its rank position. It is often used to measure the performance of web search recommendation systems.

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(i+1)} \tag{3.27}$$

$$nDCG_k = \frac{DCG_k}{IDCG_k} \tag{3.28}$$

How to calculate the $nDCG$ value is shown in 3.28. Where k is the maximum amount of suggested items, and rel_i is the graded relevance of the result at position i. $IDCG_k$, from 3.28, is the ideal DCG_k value. This is the result list sorted on relevance.

Half-life utility [36] Assume that the further down an item is in the list the less chance there is for that item to be viewed by the user. The rate of the decaying probability is exponential.

$$HL_u = \sum_i \frac{\delta(i)}{2^{\frac{i-1}{\alpha-1}}} \tag{3.29}$$

$\delta(i)$ is 1 if the user is interested in the item at position i and 0 if not. α is the viewing half-life, or half-life parameter. The half-life utility of all the users are shown in 3.30

$$HL = 100 * \frac{\sum_u HL_u}{\sum_u HL_u^{max}} \tag{3.30}$$

HL_u^{max} is the maximum possible value of the half-life utility value.

some overlapping of the algorithms (not really comparing two ranked list, but still evaluating a recommender system producing ranked lists) something like this perhaps

Beyond Accuracy There is an emerging understanding that good recommendations accuracy alone does not give the users of the recommender system an effective and satisfying experience [56]. The following *measures* attempts to assess a recommender systems usefulness beyond being able to provide accurate recommendations to the users.

Coverage The term coverage can refer to several distinct properties of the system. Most commonly, the term coverage refers to the proportion of items the recommendation system can recommend, also known as *item-space coverage*. The simplest measure of catalog coverage is the percentage of all items that can ever be recommended. Coverage can also be the proportion of users interactions for which the system can recommend items, known as *user-space coverage*. In many applications the recommender system may not provide recommendations for some users due to e.g.
low confidence in the accuracy of predictions for that user. In such cases one may prefer a recommender that can provide recommendations to a wider range of users. However, an increase in coverage is only beneficial if the accuracy does not drop significantly.

Perceived quality To gather the perceived quality the system must ask the user to examine the recommended item, and give feedback regarding the their actual interest in the recommended item. For the feedback from the user to be as complete as possible, the system must supply the user with the reason to why the item was suggested, and the metadata of the item. When the user has this overview of the item, the user's feedback regarding the item can produce some quality measure. One way of having the user to give this feedback is to re-rate the recommended item on a similar scale as the item was rated. A rating scale from 1 to 5 is often used for both [93].

Novelty and Serendipity Some recommender systems produce highly accurate recommendations and also have reasonable coverage - and yet that are useless for practical purposes. For instance, a music recommender can recommend Rihanna to every customer who have not yet listened to Rihanna. However, statistically, this is highly accurate as most people have listened to Rihanna or at least knows about her and have consciously chosen not to listen to her. Much more valuable would be a recommendation to a kick-ass indie-rock band that the active user would love, but will never hear about in the news. We therefore need a new dimension for analyzing recommender systems that consider "nonobviousness" of the recommendations. One such dimension is *novelty*. Another related dimension is *serendipity*. A serendipitous recommendation helps a user find a surprisingly interesting item the user might not have discovered otherwise. To clarify the difference between the two, a novel recommendation could be to recommend an unknown album from one of the users favorite artists, that the user likely eventually

would have discovered herself. However, a recommendation by an unknown artist is more likely to be serendipitous. Serendipity is therefore a measure of how surprisingly the successful recommendations are.

As novelty is the the degree of new and interesting items recommended for the user, the system must ask the user for feedback to be able to make any assumptions around the novelty. The novelty together with perceived quality allows the system to make informed decisions about the ratio of new items to recommend for the user. The novelty is expected to change over time, some times the user would like to receive recommendation on new items, and other times recommendations closer to the user's preferences. For the system to be able to detect these changes in preferences, and acting accordingly would be beneficial in regards of the user's satisfaction.

$$Novelty(u) = \frac{1}{N} \sum_{i=1}^{N} 1 - Knows(u, i) \tag{3.31}$$

In 3.31 $Knows(u, i)$ represents a binary functions which returns 1 if the user u knows the item i, and 0 if not. The set of items used for the calculation is the set of recommended items for the user.

The system should be able to produce items both items known to the user, and items unknown to the user. For trust in the system and novelty respectively.

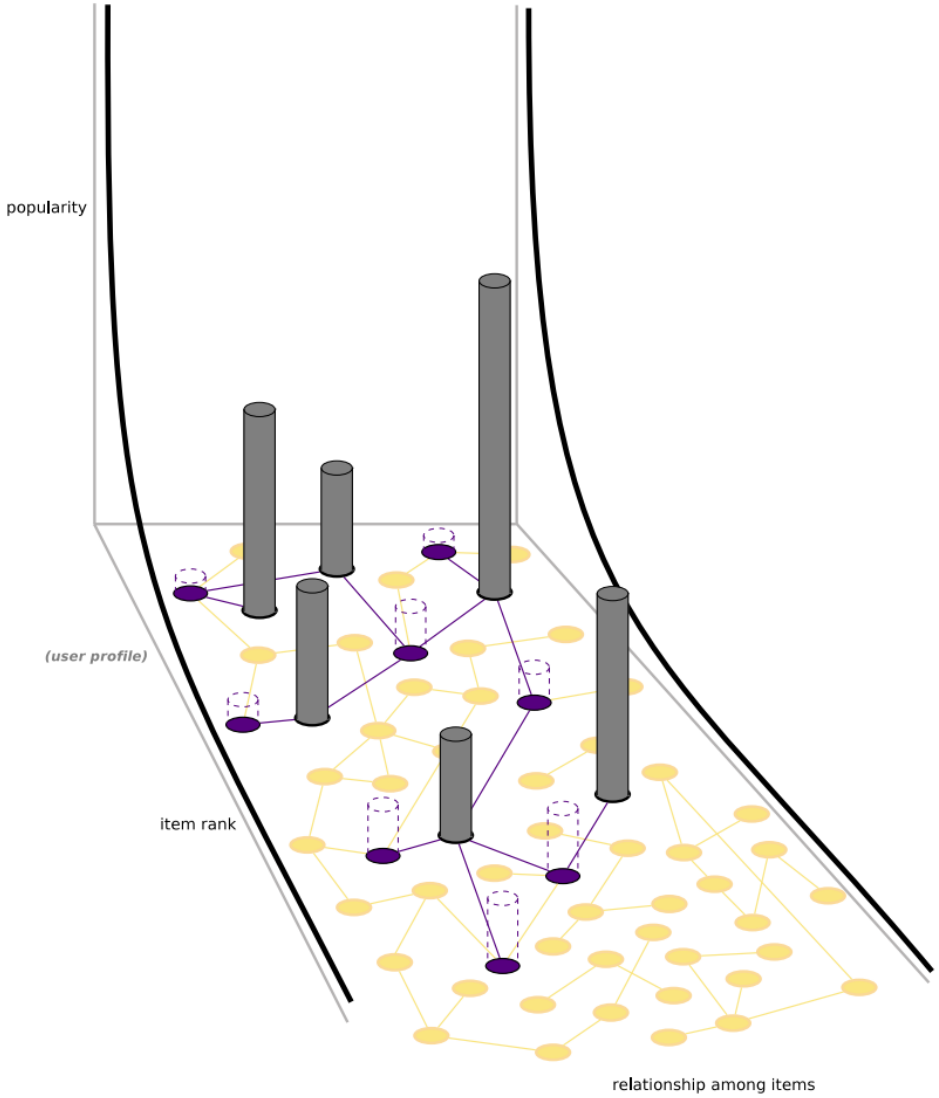


Figure 3.22: Long tail show with the item similarity between items. Similarity shown though edges between the nodes. The gray columns represents the user profile, the violet represents items which might be of interest to the user, and the height is the relevance. Adapted from [38]

todo - not sure if its ok to use this image (not self made) figure from music recommendation, but long tail applies to fashion
todo - if so, something about longtail

Diversity Diversity is generally defined as the opposite of similarity. In some cases suggesting a set of similar items may not be as useful for the user, because it may take longer to explore the range of products. E.g. when presenting a list of 5 recommendations, the system should not recommend 5 Ralph Lauren shirts with different colors. As diversity may come at the expense of other properties such as accuracy, one should evaluate the decrease in accuracy vs. the increase in diversity.

Other evaluation methods worth knowing about includes confidence, trust, utility, robustness, adaptivity, scalability mentioned in [56, 95].

User-Centric Evaluation User-centric evaluation focuses on the perceived quality of the recommendation system [88, 66]. To gather this kind of information, user feedback is required. User-centric evaluation is meant to handle the short comings of the offline evaluation metrics, such as predictive accuracy metrics. User-centric evaluation can make evaluations on items which the user has not yet show interest in. This allows user-centric evaluation to make evaluations on information about the system, such as the perceived quality and novelty of the recommendation system. When the user feedback has been gathered the data must be analyzed.

move offline beyond accuracy to here maybe?

Evaluation using Implicit Feedback

Accuracy metrics such as RMSE and MAE are not especially well suited for implicit feedback datasets, as they require knowing which items are undesired by a user [57]. One way to handle the issue with missing negative feedback is through conversion of the implicit feedback to explicit feedback, and thereby producing a sense of negative feedback. One issue with this is what is considered as negative feedback in the sense of implicit feedback. The reason for a user not to access an item is not necessarily grounded in dislike, but could simply be an overlook. On the other hand, if the user accessed the item for a short time without buying it, the user might not like it. These questions raises the need for a different approach when wanting to measure a recommender system using implicit feedback.

Joachims et al. [63] shows that clicks can be biased, and has to be interpreted relative to the order of presentation and relative to the other abstracts.

extend

Two ways of measuring the system is through ranking and usage prediction described earlier.

Rank Based The ranking approach can be considered more suited to test recommendation systems using implicit feedback since a subset of the items are meant to be recommended to the user. This subset is usually a ranked top K list of items where the items in this list is the items the system predicts will be most liked by the user.

One issue with this approach is that the items in the predicted top K list has to be ranked in some way, the same goes for the list this list is to be compared to. Thereby producing the requirement for a ranked list of preferred items from the user. If there is such a ranked list, ranking accuracy will help to tell how well the system is suggesting items for the user, such as [106].

maybe hard to differentiate between the names HELGE: Sprk!

Accuracy Ranking When there is lack of explicit user feedback and a ranked list of the items cannot be produced, ranking the predicted list and scoring this list based on the actual user preferences can be used to evaluate the system. The actual user preferences are a list of binary preferences for the items. This list is often possible to construct from implicit feedback, but will seldom be a complete list, and most often be a list with only positives (not possible to produce not interesting). Some of the different approaches when dealing with implicit feedback are: Li et al. [71] who used MPR and MAP to calculate their performance. Pan et al. [83] who used a set of top-k evaluation metrics: precision, recall, F1, nDCG, ROC and 1-call. Sindhwani et al. [97] who used

roc curve and precision-recall curve to evaluate their experiments. Pan et al. [82] who used MAP and Half-life Utility.

$nDCG$, MAP and Half-life utility has a similar setup. They all have a decaying factor, and rewards the system in different ways for how high in the list a relevant recommended item is.

On the other hand metrics as Precision alone has no decaying factor, and will not penalize a system with uninteresting items suggested higher in a ranked list.

Evaluation of Cold-start Recommendations

Helge: Pros og Cons plz

Fix pros pg cons, for mye "synsing

The cold-start problem can be considered a sub problem of coverage. The cold-start problem occurs when the recommender system cannot draw any inferences for user or items which it has not yet gathered sufficient information. When evaluating the cold-start system performance one is interested in measuring the system accuracy for these users and items.

The evaluation metric used depends on the type of feedback available. Most experiments carried out have used *traditional* explicit feedback datasets such as MovieLens, EachMovie, Netflix etc. Accuracy metrics such as MAE [89, 90, 77, 78, 99] and RMSE [28, 29] are therefore the most used ones. In the experiments where binary rating data have been used Precision@N [73, 50], ROC curves [28, 50, 94] and Area Under Curve (AUC) [73, 50] seems to be the preferred evaluation metrics.

Another way to discriminate between different recommender techniques is coverage. The recommender system may not be able to make predictions for every item. For this reason, it is important to measure the portion of ratings that an RS is able to predict (ratings coverage). However, this quantity is not always informative about the quality of a recommender system. A RS is likely to be good at predicting nearly all the ratings for heavy users and not be able to do the same for users who have rated few items. For this reason, one should also compute the users coverage, defined as the portion of users which the RS is able to predict at least one rating for. Good et al. [53] measure the item-space coverage, while Massa et al. [77, 78] measures both the item-space coverage and user-space coverage of their methods.

Massa et. al [77] argue that performance measures such as Mean Absolute User Error (MAUE) is a good measure for cold-start recommendations since every user is taken into account once and a cold start user is as influential as an heavy rater. Similarly, Park et al. [85] measure normalized MAE (NMAE) by macro-averaging, which first calculates the mean average error of each users and taking the average of all users.

To simulate the cold-start scenario, different approaches have been employed: One popular way to simulate a cold-start user scenario used by [99, 70] is to split the dataset in two disjoint sets, a training set containing 90% of the users and the remaining 10% being in the test set. For each test user one trains a model with a random subset T% of their ratings e.g. 5% or 75%, and then use the model to predict their remaining ratings. The same methodology can also be used to simulate a cold-start item scenario. Another highly similar way to simulate the cold-start user and cold-start item scenario was used in [89, 90]. For the cold-start user scenario one selects a subset of the users with e.g. more than 200 ratings. One then trains the model with a subset of the ratings. In the case of [89] 30, 45, 60 and 90 ratings was used. After training the model one computes the error on the hidden ratings for the same user. Stern et. al. [**?**] also used a similar method which they called Given n, in which they trained their model using 2, 5 and 10 ratings for each test user and predict the remaining values.

The advantage of the latter approach is that they use a selection criteria for the test users to avoid users with really few ratings which might be crucial on a cold-start dataset. The obvious problem with this approach is that for cold-start datasets these users might stand for a large portion of the ratings. There are both pros and cons of using a fixed subset of ratings rather than using the percentage of ratings, the best choice will most likely depend on the dataset. Using a fixed subset of ratings would be preferable e.g. if the number number of ratings given by the users are fairly uniform.

Another *simpler* approach employed by [78, 60] is to determine a cutoff point for what is considered a cold-start user. E.g. that every user with less than 5 ratings is considered a cold-start user. Then separately measure the error on predictions made to these users.

The problem with this approach is that it does not measure how well the recommendation quality improves as users provide an increasing amount of ratings, giving a less detailed view of the systems performance. The number of ratings predicted is also likely to be fairly small given a cold-start dataset.

To simulate a cold-start system scenario Agarwal et al. [28] split the dataset in two using 75% of the dataset for training and 25% for testing. They then train the model using 30%, 60% and 75% of the data and compare their performance on the testset.

Good and simple model as it is likely to generate three different training sets with different sparsity levels, which is exactly what we want to test. However, if the training examples are drawn at random the experiment should be repeated multiple times to avoid getting *unfortunate* splits.

3.5.2 Online Evaluation

Instead of doing offline evaluations on the system, one could also run large scale experiments on a deployed system. Such experiments evaluate the performance of recommender systems on real users which are oblivious to the conducted experiment. The real effect of a recommender system depends on a variety of factors such as users intent, the users context and how the recommendations are presented to the user. All these factors are hard to capture in an offline setting. Thus, the experiment that provides the strongest evidence as to the true real value of the system is an online evaluation, where the system is used by real users to perform real tasks.

Online Evaluation Metrics

Online studies in recommendations and advertisement usually measure the click-through-rate (CTR) of the recommendations, which aligns with financial incentives and implicitly factors in accuracy, novelty, diversity, etc., according to the preferences of the distribution of users. The click-through-rate of an algorithm is defined as the number of clicks your recommendations get divided by the total number of recommendations that have been made. A high CTR therefore indicates that your system is doing well.

$$CTR = \frac{Clicks}{Recommendations} \tag{3.32}$$

A/B Testing

A/B testing or bucket testing is used to test a system on live audience. In its' simplest form the audience is split into two groups, but it is also possible to make multiple groups

of users. The different groups are presented with different altered versions of the system and is asked to use the system as they normally would. The goal is to figure out which version is producing the best results, whether it is user satisfaction or revenue. How the altered versions are scored is usually done trough a measure of the applications main goal, for instance with an e-commerce application where the main goal is to sell items, the score could be revenue produced by that version.

Example In the case with the SoBazaar application, on way of doing a A/B testing on the system is as follows: 3 groups of users are made, one with the unaltered system, one with method A to produce recommendations and one with method B.

Group Split The groups of users can either be selected to fit the global distribution of users, or be a set of similar minded users. The latter case might benefit a system where the intended goal is to specialize the system for a set of users, or actually partition the system into fitting a subsets of its' users.

Group Size The size of the groups depends on the amount of splits and intended stability of the system. The group with the original system will be the biggest group to maintain the established image of the application. For a system with a well established image, and a large set of faithful users, changes in the system might produce unwanted results. The two groups with the altered versions of the system can be of the same size to make it simpler to measure the performance of the two system against one another.

System Measuring After a set period of time, the two altered versions are compared to one another and the original system. This is done trough measuring either the revenue produced or clicks. For a recommender system it is not just interesting to look at the final income number, but also if the user actually showed any interest in the products recommended for the users. If a larger portions of the users from version A showed an increased interest in the recommended items compared to the users from version B, that might indicate that version A is a stronger system than system B

Disadvantages An issue with A/B testing is that when splitting the users into subgroups of users, these subgroups might not possess the same user properties as the complete set of groups had. This might lead to a biased score for the group, which might not reflect the actual score of the system when releasing it on the full set of user groups.

Multi-armed bandit experiments [9]

A multi-armed bandit is a type of experiment where:

- The goal is to find the best or most profitable action
- The randomization distribution can be updated as the experiment progresses

The "multi-armed bandit" described a hypothetical example where you face several slot machines ("one-armed bandits") with potentially different payouts. You wish to find the slot machine with the best payout rate, but you also want to maximize your winnings. The fundamental tension is between "exploiting" arms that have performed

well in the past and "exploring" new or seemingly inferior arms in case they might perform better.

In normal A/B testing, you will split the traffic equally between both systems, meaning that both get 50% of the traffic each, all the time. When using bandits you can continuously adjust the traffic each variation receives based on its performance. Variations that appear to do well gets more traffic, and variations that clearly is underperforming gets less. The adjustments are made based on a statistical test that considers both the sample size and performance metrics together.

Advantages The main advantage of using multi-armed bandit experiments is that it usually performs better than A/B testing when we look at average conversion rates, which in turn will decrease the cost of the experiment. Saved testing time is another advantage highlighted in the Google article.

Disadvantages Statistical significance. Instead of sending equal traffic to each of the test pages, the page which performs better will start to get more traffic. If you are running tests across pages which do not get huge amounts of traffic, one variant can run away while the others are left in the dust without getting a fair chance. Of course, if your variation is performing badly you will loose some sales or conversions in the process of A/B testing, but that is the price one have to pay for finding out if a variation really did perform badly.

3.5.3 Discussion

The main idea behind a recommendation system is to produce a set of items which are of interest to the user. For the system to be successful, this sets of items needs quality. But what needs to be considered when determining the quality of the recommendations? According to a user study [88] the most central aspects to this quality is:

Perceived accuracy

The degree of how well the user perceives the recommended items match the actual want of the user.

Novelty

The degree of new and interesting items recommended for the user.

Attractiveness

How well the recommended items are able to evoke interest or desire in the user.

Diversity

How different the recommended items are.

Context compatibility

How the system uses contextual factors to supply the recommendations with more personalized recommendations.

3.5.4 The Good

Since the data at hand is mainly implicit feedback and this data is sparse, some natural ways of approaching the evaluations task would be:

MPR

- good
- Possible to produce a score for a recommender system which relies on implicit feedback
- Usable even though there are no feedback indicating undesired items
- Does not need a ranked list of the actual preferences of the user
- bad
- Needs distinct ranking of the different items

MAP

- good
- Possible to produce a score for a recommender system which relies on implicit feedback
- Usable even though there are no feedback indicating undesired items
- Does not need a ranked list of the actual preferences of the user
- Differentiates between predicted more desired items and those that are not
- bad
- Needs distinct ranking of the different items

3.5.5 The Bad

RMSE

- good
- Well known, so it produces a clear score of the system
- bad
- Needs the actual rating of the items
- Needs a predicted rating for the items
- Not easy to gather reliable information about undesired items trough implicit feedback

4

Algorithmic background

Contents

4.1 Recommendation methods

This section aims to describe the algorithms and methods which we have developed or borrowed from others which will be included in our experiments.

TODO: Motivate the explanation of the methods

One could imagine to final system to leverage multiple recommendation techniques. When users are new to the system they are recommended the most popular items, until enough data is collected to provide personalized recommendations. Another solution would be to use pre-computed item similarities and recommend similar items to the item currently being viewed by the user. We also wish to use these methods as benchmarks.

4.1.1 Recommender algorithms

The section will describe the different recommendation algorithms included in our experiments.

Most-popular Recommender

We developed a simple most-popular recommender that uses result dithering to *randomize* the recommendations to the users. One could imagine multiple categories of most popular recommendations: Most wanted, most purchased or a combination. Dithering adds *noise* to the algorithm, which permutes the results in such a way that the top few results have a high probability of remaining on the top spots, but as one goes deeper into the results, the degree of mixing increases dramatically. It is important to note that dithering is *guaranteed* to make off-line performance worse, but is likely to make the actual performance better as the user is not presented with the same list of recommendations every time. We have experimented with two different methods of dithering:

- $Score = log_2(rank + x) - runif(y, z)$
- $Score = log_2(rank + x) - y * rexp(z)$

The $runif$ function draw samples from a uniform distribution where y specifies the lower boundary of the output interval and z specifies the upper boundary. The probability density function of the uniform distribution is $P(x) = \frac{1}{z-y}$ anywhere within the interval $[y, z)$, and zero elsewhere. The $rexp$ function draws a random number from an exponential distribution, where z specifies the λ value, which is set to $\frac{1}{z}$. Figure ?? shows the exponential probability density function given different λ values.

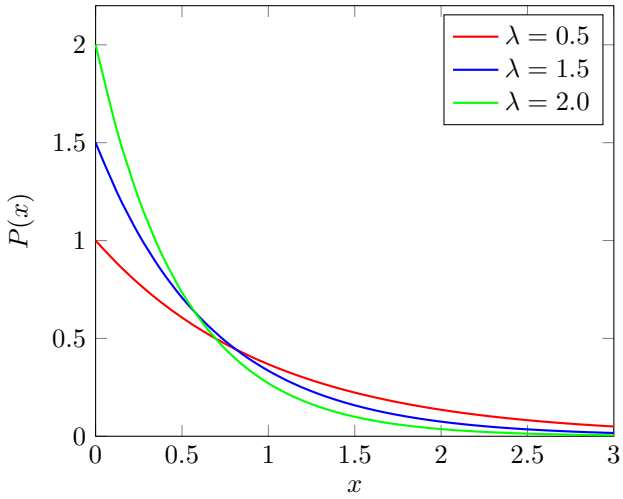


Figure 4.1: Probability density function, exponential distribution

Given $x = 5$, $y = 0$ and $z = 6$ using the uniform distribution we get the following permutations of the original ranking in 10 runs:

#Run	Result									
1	8	3	24	20	1	19	22	15	42	36
2	2	0	1	32	20	3	35	34	43	10
3	7	1	3	12	2	25	0	9	24	27
4	0	4	3	5	7	16	26	22	13	33
5	0	10	8	1	15	5	30	17	11	35
6	7	4	6	12	2	1	19	0	27	9
7	1	5	0	2	9	3	20	12	4	31
8	0	1	2	5	39	4	15	41	10	22
9	4	6	0	3	1	29	36	31	35	20
10	5	1	0	8	3	18	25	24	2	28

Given $x = 1$, $y = 3.0$ and $z = 2.5$ using the exponential distribution we get the following permutations of the original ranking in 10 runs:

#Run	Result									
1	2	4	0	35	3	1	15	72	9	5
2	0	10	11	17	1	3	8	41	15	5
3	44	1	0	15	9	4	5	59	26	2
4	0	98	1	4	2	41	8	26	11	94
5	0	6	1	2	70	4	19	14	8	3
6	2	5	0	16	15	18	1	3	32	6
7	2	65	4	0	3	45	8	1	48	36
8	3	49	5	0	2	82	8	77	11	4
9	0	1	21	8	4	85	2	6	47	3
10	0	1	21	70	11	20	2	10	9	3

where 0 is the most popular item before the permutation. The results show that alternative two has a larger occurrence of items that are further down in the list. This is due to the fact that the exponential distribution can output large values, in theory up to infinity, with a decreasing probability. The values of x, y and z can be modified to achieve the desired degree of mixing.

Item-Average

Item average is a simple recommender that always estimates the preference for an item to be the average of all known preference values for that item. No information about the individual users is taken into account. This recommender can therefore be considered a *highest rated* recommender, as it is likely to recommend the highest rated items. The following equation shows the rating prediction procedure:

$$u(c, s) = k * \sum_{c' \epsilon C} u(c', s) \tag{4.1}$$

where k is a normalization factor $(1/|C|)$. This is very similar to collaborative filtering, except for the fact that the user similarity $sim(c, c')$ has been taken out of the equation. This is the same as saying that all user similarities are the same. It is also worth mentioning that this method is not suited for binary ratings, as the result is likely to be highly random, and should therefore not be used without item ratings to average.

User-based Collaborative Filtering

Recommend items by finding similar users. This is often harder to scale because of the dynamic nature of users. The pearson correlation coefficient is used to calculate the user similarities. For a more in depth description of user-based collaborative filtering see Section 3.1.2.

Item-based Collaborative Filtering

Calculate similarity between items and make recommendations. Items usually don't change much, so this often can be computed offline. For a more in depth description of item-based collaborative filtering see Section 3.1.2.

ALS-WR

Alternating-least-squares with weighted-λ-regularization (ALS-WR) was designed fro the Netflix Prize Competition [34], where it obtained an RMSE score of 0.8975, which was one of the best results based on a pure method. in theory up to infinity Alternating-least-squares is a method to solve Equation 3.8. Since both q_s and p_c are unknown, the equation is not convex. However if we fix one of the unknowns, the optimization problem becomes quadratic and can be solved optimally. The ALS technique rotate between fixing the q_s's and fixing the p_c's. When all the p_c's are fixed, the system recomputes the q_s's by solving a least-squares problem, and vica versa. This ensures that each step decreases the error until convergence. What makes ALS favorable over the simpler and faster stochastic gradient descent is two things. ALS can be parallelized since the system computes the q_s's independently of the other item factors, the same can also be applied to the user factors. The second case if for systems centered around implicit data. Because the training set cannot be considered sparse, looping over each single training case as gradient descent would not be practical, but ALS can efficiently handle such cases [57].

ALS solves the low-rank matrix factorization as follows:

- Step 1: Initialize the matrix M by assigning the average rating for that movie as the first row, and a small random numbers for the remaining entries;
- Step 2: Fix P, solve Q by minimizing the objective function (the sum of squared errors);
- Step 3: Fix Q, solve by minimizing the objective function similarly;
- Step 4: Repeat Steps 2 and 3 until a stopping criterion is satisfied.

Zhou et. al. [110] used the difference in RMSEs between the rounds as a stopping criterion. Without regularization ALS might lead to overfitting due to the many free parameters. Regularization was therefore introduced in the form of weighted-λ-regularization to prevent the model from overfitting.

$$f(P,Q) = \sum_{(c,s)\epsilon C} (u(c,s) - p_c^T q_s)^2 + \lambda(\sum_c n_{p_c}\|p_c\|^2 + \sum_s n_{q_s}\|q_s\|^2) \tag{4.2}$$

where n_{p_c} and n_{q_s} denote the number of ratings of user c and item s respectively. S_c denote the set of items s that user c rated, then n_{p_c} is the cardinality of S_c; similarly C_s denotes the set of users who rated item s, and n_{q_s} is the cardinality of I_s. A given column of P, p_c is found by solving a regularized linear least squares problem involving the known ratings of user c, and the feature vectors q_s of the items that user c has rated. Similarly, we can compute individual q_s's via a regularized linear least squares solution, using the feature vectors of users who rated item j, and their ratings of it.

Content-based

TODO: Complete motivation

As previously mentioned content-based recommendation does not suffer as much from the cold-start problem as collaborative filtering approaches.

To build features for content-based recommendations we examined the content in the product database. To find the product-type, material, style and color we analyzed the content of the title, description and meta-description fields extracting the most used

keywords after removing the stopwords, which we combined and grouped for the different attributes. Since product descriptions could be either in Norwegian or in English we had to include the words from both languages. We also experimented with stemming from the nltk software package in python, but since it did not improve our results it was not used in the final implementation. E.g. to determine if a product-type falls under the *sweater* category we check for the following keywords: *sweater, cardigan, jumper, hoody ,genser and genseren.* We attempt to extract the following features from the product-database content:

- Price: Grouped into price brackets
- Brand/Storename
- Category: E.g. dress, jacket, top, pants and boots
- Material: E.g. Cotton, wool, polyester, leather
- Style: E.g. Classic, Modern, Comfy, Sexy
- Color

It was a pleasant surprise to find out that 5,060 out of 5,855 items could be found in the product database and be assigned features. The following table shows an overview over how many items we were able to extract the different features for.

Attribute	Percentage
Price	1.000
Brand	0.742
Category	0.628
Material	0.408
Style	0.314
Color	0.162

Price, brand and product-type can be found for *most* items, while material style and color could only be found for a minority of the items. The dataset is in no way ideal for content-based recommendation as it highlights on of the main weakness of content-based methods, namely that they require rich descriptions of items to build well informed user profiles. As mentioned in [80] this is most often not the case in real applications.

Mention what item-based methods will be used

4.1.2 Cold-start Solutions

Due to limitations, mainly in the form of lacking data the number of cold-start solutions we are currently able to experiment with is fairly limited. As we never got access to user-data, we have to cross out RBLF and other methods requiring user features. This leaves us with Naive Filterbots [85] which easily could be combined with our implicit ratings.

Filterbots

The main reasons for experimenting with filterbots is simplicity, the fact that it easily could be combined with our implicit ratings in addition to the fact that previous experiments [28, 29] show that its performance is close to the state-of-the-art methods. It is worth to keep in mind that filterbots was developed for traditional collaborative filtering methods and it will be interesting to see how the filterbots affect the performance of ALS-WR. Filterbots can potentially help solve the cold-start user problem by making it possible for new users to connect to users that capture the general underlying trends of the entire user group. Filterbots are mainly meant to improve performance when data is scarce and not to degrade performance when data is plentiful.

Similarly as in [85] we decided to experiment with global bots. We have currently implemented the following bots: A *BrandBot* which rates an item based on the brands average rating, an *AverageBot* which rates an item based on their average rating given to it, a *CriticBot* which selects n critics among the most active users and rate items based on their average rating, a *PopularityBot* which rates an item based on its popularity, more ratings equals a higher rating. A filterbot takes the training set as input and generates a set of ratings which are then added to the training set. Both the *AverageBot* and *BrandBot* can be seen to incorporate the implicit rating factors such as recency into the ratings generated. However, for the *PopularityBot* and *Critic* but we wish to implement additional functionality to ensure that the new users are connected with items that are relevant right now and not consider outdated items or users which are no longer active. Since we have access to timestamps we could e.g. make the *PopularityBot* rate items based on their popularity the last couple of weeks instead of over the entire dataset period. Similarly, critics can be selected based on their activity the couple of weeks/months.

TODO: Describe implementation?

4.2 Implicit feedback to implicit ratings

In this section several *existing* methods will be presented, all solving the following problem:

Input: A dataset with **implicit feedback** containing event logs with the following information available: event type, product id, user id and timestamp of the event.

Output: For every user u, a rating between 1 and k on every item that the user has interacted with. These ratings are called **implicit ratings**.

This problem definition is further used in Section 5.1 where multiple novel ways of quantification are presented based on ideas found in this section. First however, a deeper understanding on the relationship between implicit and explicit feedback is needed.

Explicit feedback is present in many of the largest recommender systems today and hence, is extensively researched [27]. The user is commonly asked to rate item i on a Likert scale from 1 to k, ranging from strongly disagree to strongly agree with an item. Its advantages are, among others the ability to get precise feedback from the user and capturing both positive and negative preferences. However, although having a high popularity, the method has multiple weaknesses. The most prominent weakness is the difficulty of collecting ratings: the method requires the user to spend time rating items and the amount of feedback is often scarce, creating sparse data sets. Further, explicit ratings are often subject to inconsistencies known as natural noise [31] and users might also be pressed to report different preferences due to peer pressure [33]. The fact that we are introducing a user overhead, makes it difficult to have a complete view on the user preferences [61].

There are also practical reasons for not having explicit feedback in form of ratings, dependent on how a website/application interacts and works with its customers. One of the most popular recommender sites in the world, Netflix, ask their users to rate items after consumation. However, this process does not *cost* anything for the user, and the possibility to rate is included in the monthly subscription price, whilst in a webstore rating an item often require the user to first buy it, consume it and finally rate it.

Implicit feedback on the other hand utilizes, often already collected, analytics data and hence does not require any extra effort neither for the user nor the frontend-developers. This creates a more pleasant experience for the user, but can also yield better recommender results since often the event log is less sparse than rating datasets.

It is important to understand that there are several fundemental differences between implicit and explicit ratings, in everything from *meaning* to *evaluation techniques*. Partly inspired by Hu et al. [57], we can identify the following key charecterstics and differences between them:

Explicit ratings:

- Contains both positive and negative feedback. In other words, a user is able to explicitly tell the system that he/she does not like an item, as well as the opposite.
- Indicates preference, often on a Likert scale (or similar), where scores range from total dislike to high satisfaction with an item.
- There is medium noisiness in the data, but the amount is highly dependent on the domain in question [31]. With little noisiness we achieve more precise ratings.

- Metrics in use are commonly RMSE, MAE or other evaluation schemes where we consider how well we match a test-set.
- Both evaluation schemes and recommendation techniques using explicit feedback is heavily researched and used. Many of these are easy to implement, and multiple open-source tools and projects are at researchers and developers disposal [**?**].

Implicit ratings:

- Usually only contain positive feedback, since we base our models on user activity on an item representing something good. However, looking at metrics such as bounce rates so forth, negative feedback can be produced with some effort.
- Indicates confidence, that is a recurring event is more likely to reflect the user opinion, but not necessarily his/hers preference.
- There is a high degree of noise and this is one of the main challenges with implicit rating and its precision. Researchers and developers need to be sure to select correct set of events and session-metrics to base implicit ratings on.
- As users do not provide numerical scores, a precision-recall evaluation scheme is often preffered to more preference-based metrics.
- Strengths include not requiring extra feedback for the user, having less sparsity and catching actual behaviour in the application. Less influenced by peer pressure and so forth.

Notice that several events such as purchasing an item may be considered as explicitly giving positive feedback, but as we argue in this paper all events in the SoBazaar dataset (including purchases) are treated as implicit feedback. The reasoning behind this lies in the fact that purchasing, in contrast to positivly rating, an item happens before actual consumption. Hence the user may not like the item in question, it may be a gift for another person or a variety of other reasons. Whilst in the scenerio of rating, the user has already consumed it and is explicitly answering the question *To which degree did you like it?*.

heri-notes: merpresis formulering

We will look into more of these charecteristics in our thesis, carefully examining the state of the art with regards to both weaknesses and advantages and finally introduce some novel techniques facilitating the creation of implicit ratings.

4.2.1 Binary implicit ratings

One of the most common techniques of quantifying implicit feedback to implicit ratings is to use binary ratings - that is if a user has shown any interest in an item we give that item a weight of 1, similarily all items the user has not shown interest in (e.g. not clicked on) we give a rating of 0. This method was introduced and used by Hu et. al. [57] where they formalized the notion of confidence which the r_{ui} variables measure, e.g. if a user u had clicked on an item i 2 times it would translate to the score $r_{ui} = 2.0$. This was further converted to a p_{ui} value derived by binarizing the r_{ui} values:

$$p_{ui} = \begin{cases} 1 & \text{if } r_{ui} > 0 \\ 0 & \text{if } r_{ui} = 0 \end{cases} \tag{4.3}$$

As mentioned earlier in this section, and as it is also noted in the original research, when we have implicit rather than explicit feedback there may be multiple reasons beyond preference that decide consumation of an item. It may be due to the items availability or price; the users knowledge of the item caused by e.g. the user interface or one of many other factors different from the user preferring or disliking the item. To remidate this they introduce a variable c_{ui} which measure the confidence in observing p_{ui}, where their main hypothesis is that as the r_{ui} grows the confidence is stronger. Hence, a plausible choice for c_{ui} would be:

$$c_{ui} = 1 + \alpha r_{ui} \tag{4.4}$$

Where α controls the rate of increase, and is set to 40 in their experiments as this produced good results. They now use these two scores (p_{ui} and c_{ui}) in parallel when finding latent factors and training their model. In short they find both a user and item vector (x_u and y_i, respectively) where the preference is assumed to be their inner product: $p_{ui} = x_u^T y_i$. Notice as well how this method stand out from similar matrix factorization methods (see Section **??**) by accounting for varying confidence levels:

$$min_{x,y} \sum_{u,i} c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2) \tag{4.5}$$

Where the $\lambda(\sum_u ||x_u||^2 + \sum_i ||y_i||^2)$ term is necessary for regularizing the model, so that is will not overfit the training data. λ is highly data dependent and should be set by cross-validation.

However, notice the primary term where the difference between the binary weight and our user/item-models are minimized and perhaps more importantly multiplied by our confidence. As the details on this topic are quite detailed and diverges a bit from the generation of implicit ratings, we will stop our analysis of it here. But is shows an important aspect which we will return to in subsequent sections, namely the confidence in repetitive events and our continuous challenge to obtain higher confidence in our implicit ratings. For further information on matrix factorization and latent factors refer to Section 3.1.2 or to Hu et. al. original research [57].

4.2.2 Implicit ratings for binary domains

In the previous sub-section we saw how r_{ui}, a score for user u on item i, was set based on e.g. number of clicks but perhaps more commonly the duration or amount of which the user has consumed the item. This way, in a domain such as television a user would

obtain $r_{ui} = 1.0$ if the whole programme was watched, and similarily $r_{ui} = 0.5$ if half of the show was watched, and so forth. Now, this works well in domains where actions on an item are continious, but in many domains such as a e-commerce store we mostly have binary events such as clicks or purchases. One scheme, proposed by [51] is to differentiate between different events and by weighting these by importance or relevence we can create ratings using the whole scale between 0.0 and 1.0.

This is just one example of how the domain and data available affects our methods, techniques and reliability to our predicted numbers. This is common theme when dealing with implicit ratings, and it is an important aspect that the reader should be aware of and keep in the back of his/hers mind. Now, in a proposed e-commerce domain where our data is based on web-access logs we may have the following events to choose from:

Event type	Description
0	Item purchased
1	Item placed in shopping cart
2	Item placed in wish list
3	Item browsed based on search result
4	Item browsed

Then, our heuristic would count the frequency of each event for an item i on user u. However, just counting is not a bounded function so if one give *Item browsed* the weight of 1 and the *Item purchased* event a weight of 100, then a user browsing an item more than 100 times would get a higher implicit rating than a user buying it.

Instead, [51] proposes using a global rating mapping, that uses *levels of frequency*.

Event type	Scores
0	100
1	70, 77, 80
2	30, 40, 45, 48, 50
3	20, 25, 28, 30
4	10, 15, 18, 20

Table 4.1: Scores per event type, increases as frequency of each event increments

Now, a user browsing an item i two times, would get a score of 40 and a if buying it the score would be 100 (e.g. the event type with highest interest level supersede all others). Equally, if a user browsed the item 100 times, it would make no difference in score compared to browsing it 4 times. The scores given to the various event types may vary from what kind of dataset one have, the domain and should be evaluated using one or more of the methods mentioned in Section 3.5.

If we wanted ratings between 1 and k one can transform the score s provided by table 4.1:

$$ImplicitRating(s, k) = 100 * \frac{k-1}{100} + 1 \tag{4.6}$$

The advantage of this heuristic is that it requires no training, it is simple to understand and works reasonably well, if weights are chosen correctly. The latter is also its

largest weakness as finding these weights may be difficult. If one had explicit feedback, as well as the implicit, one could train a model automatically choosing weights. Further, when using the scores defined as above we only use a small percentage of the scale 0-100. This leads to problems when a user triggers one type of event many times more than another, as both would reach the highest score of the given event type, but there would be no difference in scoring even though they had very different activity. We would like a scheme where we use the whole scale of scores and where there were no global ceiling, but rather let the scoring be normalized to the number of events for the user in question.

Improvements to this model are presented in Section **??**

4.2.3 Regression analysis

Introduction A different method for converting events to ratings is finding a mathematical model which fits our observed data. Using this model we can predict new ratings by extrapolation or intrapolaton. Our observed data can be in N-dimensions, but in the figure below we consider two dimensions of observed properties, in which the goal is to find a regression line matching the data points with minimal errors. An example regression line matching five data points is provided below:

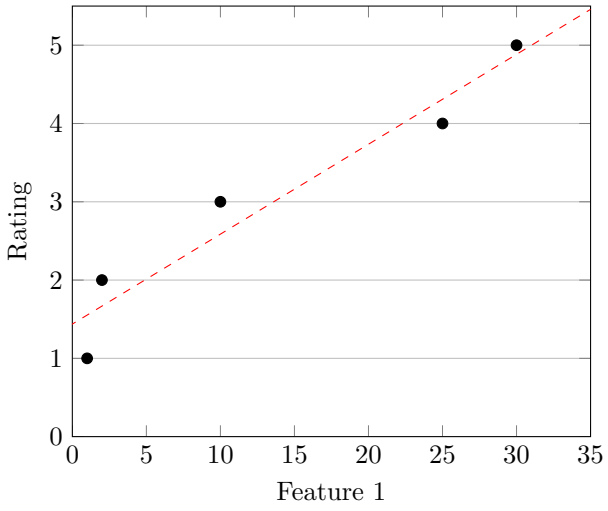


Figure 4.2: Regression line fitting the 5 observed datapoints

In this example we consider the relationship between one feature (the independent variable) and the rating (ground truth/dependent variable), using five observed datapoints. We can then, given a feature-value, predict future ratings. The process of finding the regression line is better generalized by solving the following equation, where we are given a number of independent and dependent values:

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \cdots + w_p x_p \tag{4.7}$$

where we want to find the values of the w-parameters, also called the unknown parameters. This is done by using multipe known values for x and y, the independent and dependent variables respectivly. In Figure **??** we have the following datapoints:

Feature-value (x)	Rating (y)
1	1
2	2
10	3
25	4
30	5

Which, solving Equation 4.7, five different equations are used with the unknown parameters w_0 and w_1:

$$\begin{aligned} 1 &= w_0 + 1w_1 \\ 2 &= w_0 + 2w_1 \\ 3 &= w_0 + 10w_1 \\ 4 &= w_0 + 25w_1 \\ 5 &= w_0 + 30w_1 \end{aligned} \tag{4.8}$$

Using a minimization method called *Least squares* we try to minimize the residual $e = y_i - w^T x_i$, using sum of squares between the observed values in the dataset and the values predicted by the linear approximation. In other words we want a minimal distance between all observed datapoints the the final regression line. Mathematically it solved the problem of the following form:

$$\hat{w} = \arg\min_w \sum_i (y - w^T x_i)^2 \tag{4.9}$$

Carrying out this minimization for our equations 4.8, we obtain an intercept value (w_0) of 1.4379 and a slope (w_1) of 0.1149, which matches the regression line plotted in Figure **??**. Predicting a new rating based on a feature value in now very simple - as an example we assume a new independent value is provided with the value 15 and an unknown rating, that is $x = 15$ and we can solve the equation $y = 0.1149 \cdot 15 + 1.4379$ which yields the y-value or rating 3.1614.

Using regression analysis on implicit feedback The process of using regression analysis in order to predict ratings based on implicit feedback has already been done, in a different domain, by Parra et. al. [86]. Where they first did a quantitiative user study asking 114 active users of the music service Last.fm to rate items which was found in their activity history. These ratings were used as dependent variables and then using various schemes described below they mapped these to the independent variables. Now, a key point in their study is the fact that all models found used explicit feedback from a user study as a *ground truth*. The necessity of such a ground truth is the largest weakness in terms of regression analysis on implicit datasets - as it is per definition non-existent. A possibility is to give various events scores based on importance in the system, with e.g. a purchase having top-score and clicking on an item a low score. This is discussed further in Section **??** where an implimentaiton and evaluation of such a design is presented as well.

The study looked at three different features for each item, found in the dataset:

- User activity on the item
- Global popularity of the item
- How recent the user had acitivity on the item

For each feature all items are categorized into three buckets, so that items are equally distributed across one feature. Thus bucket one for the feature of global popularity contains items with low popularity and bucket three those with the highest popularity, and so forth. The main reason for having this sampling strategy is to create a homogenous set of items where outliers are accounted for, and we are less prone to over or underfitting.

Four models are proposed, utilizing the features above and the ratings provided in the user study:

$$\begin{aligned} r_{iu} &= w_0 + w_1 if_{iu} \\ r_{iu} &= w_0 + w_1 if_{iu} + w_2 re_{iu} \\ r_{iu} &= w_0 + w_1 if_{iu} + w_2 re_{iu} + w_3 gp_i \\ r_{iu} &= w_0 + w_1 if_{iu} + w_2 re_{iu} + w_3 if_{iu} \cdot re_{iu} \end{aligned} \tag{4.10}$$

where every independent variable (if_{iu}, re_{iu} and gp_i) is the bucket number between 1 and 3. if_{iu} indicates how many times user u used item i, re_{iu} how recently user u used item i and gp_i the global popularity of item i.

Evaluating a regression model is done by confirming the *goodness of fit*, and the most common metric is to calculate the R^2 which based on residual values in Equation 4.9 yields a number between 0 and 1, indicating a bad or good fit, respectivly. By accounting for recentness in their regression model (model 24) they achieved an improvement in the R^2 value by *10%*.

Summarizing, the following requirements are needed in order to perform regression analysis on implicit feedback:

- A ground-truth score.
- A set of numeric features.
- Each feature have to be transitive, that is a linear relationship in the values have to exist.

The fact that the features have to be transitive implies that one can not use e.g. *item category* as training feature, unless the brands (or rather the brand ids) themselves are sorted by price or equality, then renumbered.

4.2.4 Relative preferences using buying frequency

A hybrid approach using sequential pattern analysis and collaborative filtering techniques is presented by Choi et al. [40]. In their method, coined HOPE, they calculate an implicit rating by finding the absolute preference AP from users and items. These preferences are further used to find the relative preference which finally are normalized into an implicit rating. Once the implicit ratings are derived they calculate similarity scores and find K-nearest neighbours in a traditional fashion. Finally these neighbours are used in order to calculate a CF-based predicted preference (CFPP). This score is integrated with a Sequential Pattern Analysis-based predicted preference (SPAPP) which

is derived from matching subsequences of common sequential patterns between users. This process is summarized in Figure 4.3.

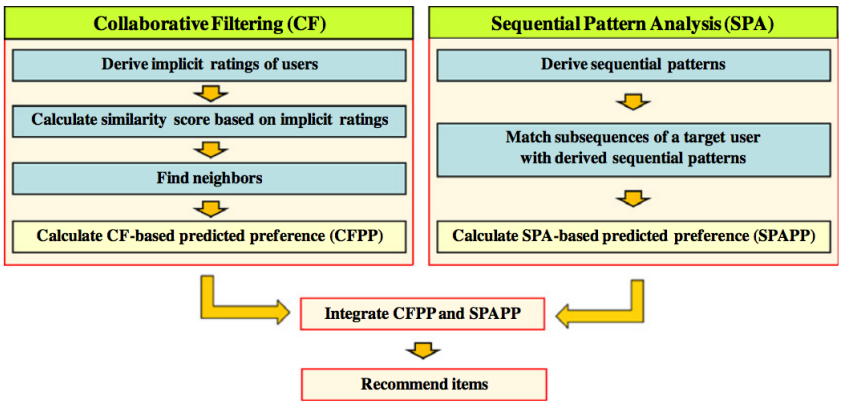


Figure 4.3: Overall framework of HOPE system, calculating implicit ratings and combining these with sequential pattern analysis

We will continue this section focusing on the convertion from transaction data into implicit ratings, rather than going in depth into sequential pattern analysis and calculating the CFPP and SPAPP-scores, as the interested reader rather should consult the original research for a indepth discussion. When calculating the AP score we focus solely on the purchase data of the user u, but it can easily be extended to other types of events. However, the score does makes more sense when using it on domains having a high degree of repeated actions - such as e-commerce stores selling convienience products with short life spans or a music service having users listening to a song multiple times. The absolute preference is calculated as follows:

$$AP(u,i) = \ln(\frac{trans(u,i)}{\sum_{e \in E} trans(u,e)} + 1) \tag{4.11}$$

where $trans(u,i)$ is the number of transactions for user u on item i, and E is the set of all items in our system. As an example if a user u purchases an item i four times out of ten transactions the $AP(u,i) = \ln(1.4)$, but another user p who has bought the same item one time out of one transaction the $AP(p,i) = \ln(2.0)$, and as one can see we should consider a relative preference so that repeated actions are rewarded. Further, the absolute preference only takes into account the frequency of purchases and because the frequency is heavily dependent on price, item category and lifespan of an item so in the original research they propose the following equation in order to calcualte the relative preference@:

$$RP(u,i) = \frac{AP(u,i)}{\max_{c \in U}(AP(c,i))} \tag{4.12}$$

where U denotes every user who purchased item i. The reason for using a maximization function, is to make $RP(u,i)$ range from 0.0 to 1.0 (i.e. normalization) and one can thus find a rating on a common Likert scale by multiplying with k:

$$ImplicitRating(u,i) = \lceil k * RP(u,i) \rceil \tag{4.13}$$

Here we round up in order to range from 1 to 5, but one could either round down instead and have ratings range from 0 to 4 or one could use the normalization equation defined in Equation 5.3, and setting $a = 0$ and $b = 5$.

todo: Do not reference ahead in the thesis. Move equation about normalization here

4.2.5 Challenges and weaknesses

Almost all of the research on implicit feedback has considered how behaviors can be used as positive, rather than negative evidence. This is not to say that we can't use user behaviors to get negative feedback, examples may be analyzing bounce rates (e.g. reading an article for less than 5 seconds) or deleting/hiding something from a feed of items. There are however much fewer of these types of events.

Another challenge facing implicit feedback research is the notion of degree of personalization offered by the system. In particular, individual differences can greatly impact the effectiveness of using behavior as implicit relevance feedback. People behave differently and have varying approaches to information-seeking; thus, it is difficult to generate, and dangerous to apply, all-purpose rules for describing how behavior can be used as implicit relevance feedback.

In a classic recommender system we need three parts: ratings, a recommender model and evaluation. Each of these parts needs to compliment each others advantages and disadvantages. If we start top-down then we say that the evaluation metrics used need to consider the fact that we are basing our model on implicit ratings, and thus a preference based metric is perhaps not the best metric (See Section 3.5). Further given that we choose a precision/recall scheme then we need to choose good parameters showing both the strengths and weaknesses found in the data and recommender model. How many recommendations should we provide the user in order to calculate precision and recall? And which metrics should we focus on; F1; MPR or others? When considering the recommender model there are as many countless possibibilites - ranging from traditional approcahes such as Pearson-correlation and K-nearest neighbours to more advanced and perhaps more specialized approaches such as SVD++. A common theme when using more advanced models are the number of parameters available for adjustments increases almost linearily. In a simple KNN-approach we can get various results by varying K - whilst in SVD++ we can vary the learning rate, alpha, number of features and many others **??**. The approach we use in order to create our recommender model should consider the *type* of ratings (explicit or implicit), the sparsity, the domain, number of items vs. number of users and many others. And lastly when we are creating ratings we should, as this section has mentioned several times, choose an approach matching the domain - as there can be large differences between e.g. an online TV-streaming service and an E-commerce store selling fashion.

Are MPR and F1 introduced?

This large choice of different models and their parameters in all stages from generation of ratings to evaluation, makes the pipeline quite vulnerable to fluxuations in results by small adjustments early in the pipeline. That is, by adjusting parameters in our first stage (generation of implicit ratings) it can have great effects on both our mathematical model and evaluation metrics. The number of different approaches makes it difficult to effectivly show all possibilites to the user and for some domains, even those focused on in this paper, it would not make sense to test some models or metrics.

When considering the domain of e-commerce websites a common approach is to both count transactions and clicks [**?**], but ...

heri-notes: dere presenterer mange fine teorier her. det er dog ikke klart for meg hva som er deres egne og hva dere har lnt fra andre

5

Implementation

Contents

5.1 Generating implicit ratings

In this section we continue solving the problem of generating ratings based on implicit feedback found in analytics logs, as defined in Section 4.1.2. Our goal is two-fold:

- Find novel ways of creating implicit ratings, remedying as many weaknesses and challenges, depicted in Section 4.2.5, as possible
- Customize existing and new algorithms to our fashion domain, described in Section **??**

But first, we need a clear insight into evaluation of generated ratings as this will guide our choice of methods and give us clear metrics for comparison of various techniques.

5.1.1 Evaluating generated ratings

The most important factor when creating ratings is to understand which implicit data are available and what they mean and imply. When ratings are not explicit, the implicit ratings becomes the recommender systems equivelent of a ground truth and all later stages in the recommender pipeline (See Section **??**) are dependent on the ratings representing a users preferences. Hence, when evaluating conversion methods we can do an initial analysis without any metrics by answering *Given our data, does this generalization make sense?*.

As a good example we our fashion store, SoBazaar, but any other e-commerce application is applicable. The naive way of creating implicit ratings in such a domain would be to count the number of times the user clicked on an item, and draw the conclusion that the higher number of clicks equalled a higher rating. However,

5.1.2 Levels of frequency, with global popularity

??

Considering the weaknesses presented in Section **??**, namely choosing good weights and only using a small percentage of the scale between 0-100. Further, the method presented does not differentiate between active and less active users, thus ignoring the entropy once the maximal level is reached. Instead we extend the method by considering the global popularity of each item, so that the user that..

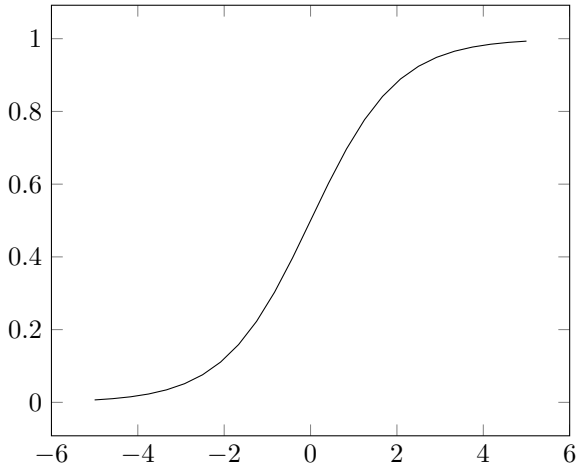
Finish the subsection about improving leveled freq

5.1.3 Introduction to the sigmoid-function

Extending our models further we want to capture our intuition that recent events should count more towards a good rating, compared to old events. We differentiate between two ways of classifying an event e_u as old or recent; one where we count the number of days between the newest event e_n and event e_u; and the second where we count the number of other events between e_n and e_u. However, intuitively we consider a user to have multiple relevant items concurrently and we know that in domains such as technology, fashion and other consumer-products an item has an age of relevancy, somewhat metaphoric to a seasonal threshold. An example of this could be a fashion store recommending warm clothes in the months between December to March, but then want to "change" product pool based on the users behaviours who are probably looking for lighter clothes (changing season).

By considering recentness we also implicitly add negative feedback to events, as in practice we are penalizing the ratings for old events. This is an important aspect to keep in mind when working with implicit feedback, as discussed in Section 4.1.2 as modern recommender engines work better when we are assuming ratings are based on both positive and negative feedback.

In order to catch our intuition mathematically we use a logistic function, which is a mathematical function having an "S" shape and a common special case of the more general sigmoid-function. In its most simplest case the logistic function is defined as:



$$f(x) = \frac{1}{1 + \exp^{-x}} \tag{5.1}$$

Figure 5.1: Logistic function having a S-shape with y-values ranging from 0 to 1.

Here the value of $f(x)$ is asymptotically limited between 0 and 1, dependent on the value of x. The steepest point of the curve happens when $x = 0$. By adjusting the exponent of $e^{(-x)}$ we can skew the curve in order to map to our data, giving us a *function of relevancy* ranging from an item being very relevant ($f(x) = 0$) and not relevant ($f(x) = 1$).

By adding two variables to the logistic function we can fine tune both the steepness and range of $f(x)$. Hence we adjust Equation 5.1 to include s, the *steepness coefficient*, and r, the *shift coefficient*. By default these are 1 and 0 respectively, but by adjusting s closer to 0 we decrease the steepness, creating a more gradual curve. Setting the r to a larger number we shift the steepest point of the curve to $x = r$, hence if we set r to 20, the steepest point (largest acceleration) in our curve would be located when $x = 20$.

show equation using these coefficients

5.1.4 Considering number of days since event

In order to better understand the usage of the logistic function, we consider an example event log.

Number of days since most recent event	Unique event types
5	1,2,3
10	1
15	1,2

As discussed in Section 4.1.2 and Section **??** we will use levels of frequency to order or event types by scores, or rather importance. But, instead of sampling scores between a given range we use a interval start and stop value - and use the whole range of float values in this interval as possible scores. We can imagine, for the purpose of this example that we have the following score intervals:

Event type	Min. score	Max. score
1	20	60
2	60	80
3	80	100

Table 5.1: Example of a scoring scheme using continious scores between a min. and max value as possible implicit scores

todo: do evaluation on different schemes

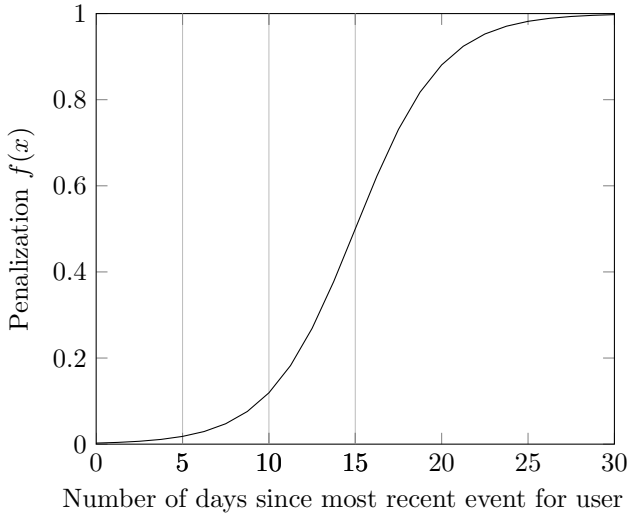
As discussed earlier, the way we assign these scores is at the moment fairly naive, some results using various schemes are given in Section **??**. The main thing to note is that we use non-overlapping intervals in order to do various optimizations in our algorithms and also that the interval for the most common event (typically a product click or similar) is $3x$ as big as our higher valued event types. This is done in order to create a larger differentiation of scores between events of same type in the same time space.

Using the events in Table 5.2 we set our shift coefficient r to 14.0, and the steepness coefficient s to 0.4 in order to both match our domain specific goals (short life span of products and seasonal activity) and get a good spread in final ratings as our dataset is small. This yields the following logistic function:

As one can see, an event happening 15 days after the most recent event for user u will get a penalization of 0.5 whilst an event with $x = 5$ recieves 0.018. Following up on Table 5.2 we can calculate the various penalizations and final scores for each day, by taking the highest possible score for event i and penalizing it in the following manner:

$$s_e(x, u) = b_e - (b_e - w_e) \cdot p_x(u) \tag{5.2}$$

where p_x is the penalization after x days. b_e and w_e are the best and worst scores achievable for event e, respectivly. The final score $s_e(x, u)$ is presented for each event below:



Num. days (x)	Event types	Penalization $p_x(u)$	Scores	Highest score
5	1,2,3	0.018	59.1, 79.64, 99.64	**99.64**
10	1	0.1192	55.23	55.23
15	1,2	0.5	50.0, 70.0	70.0

Table 5.2

Perhaps represent this differently?

When selecting a score for user u we select the highest valued one, in this case 99.64. In fact, we can optimize our algorithm by starting at the most recent events and not calculating scores for events types that yield a lower score than the current highest score. In the scenario above we could take all events on $x = 5$, then taken the event type with highest maximum value (3) and ignored all other events. Note that this is only true if you have non-overlapping event type scores/intervals, as we have per Table 5.1. We can now normalize s and get a rating r between a and b by using the following equation, knowing that $X_{max} = 100$ and $X_{min} = 0$:

$$X' = a + \frac{(X - X_{min}) \cdot (b - a)}{X_{max} - X_{min}} \tag{5.3}$$

Setting a to 0 and b to 5, as is common in recommender systems we get the rating $X' = 4.982$ when $s = 99.64$. Intuitivly this makes sense, if we assume event type 3 to be the highest valued event in our system it would be equivalaent to a user buying a product - or similar. Thus, if user u bought product i only 5 days ago this would get 4.98 as final rating. If the user does not interact with the product again in 30 days we can re-calculate the score, now using $x = 35$ which yields a penalization of 0.999 and score $s = 100 - ((100 - 80) * 0,999) = 80.2$, normalized in the same likert scale as above we get a new normalized rating 4.01 - thus still a high rating, but not as relevant for the user as 30 days earlier.

Ordering	Event type	Item id
0	1	1001
1	1	1003
2	3	1002
3	1	1004
4	2	1002
5	2	1006
6	1	1005
7	1	1001
8	3	1005
9	1	1003

5.1.5 Considering ordering of events

In the our previous method using the number of days since the most recent event we encounter several weaknesses when a user is either very active or have events with a high degree of sparsity. In the latter a user interacting with products every 20th day would see a divide in ratings since old events are placed after at the top of the S-curve ($f(x) > 0.9$) and new events achieves a penalization in the lower values ($f(x) < 0.1$). Similarily, when the user is highly active we obtain a large number of items having the same penalization weights and in essence a large duplication of ratings, provided the spread of event types are not large, which in many systems are unlikely. One possibility would be to use a finer granularity on the x values, such as seconds or minutes since the most recent event, but instead we extend our method by not taking into consideration the *time*, but instead the *ordering* of events.

As before we use an example event log where we have 10 events of three different types (1,2 and 3) on 6 different item IDs (1001-1006).

We want to continue using the sigmoid function, but in this case we will differentiate based on how many events we observe for a user. Further, if a user has e.g. more than 100 events in the event log we have two options: we can set a ceiling, saying that events older than a threshold recieves the maximum penalty or we can distribute all items evenly and extend the max-value of x-axis. Probably one would want a combination of the two, having a pretty large threshold and evenly distribute the items. In the case of evenly distribute the values you would need a *distribution factor* f expressing the relationship between steepness and shift coefficients. We use the following equation where c is the number of events for user u.

$$f_u(x, c) = \begin{cases} 1 & \text{if } c > 1000 \\ \frac{1}{1+e^{-(f/c)\cdot(x-c)}} & \text{else} \end{cases} \tag{5.4}$$

5.1.6 Linearly blending the results

Move this introduction to the pre-study?

At this point we have found multiple novel ways of calculating the implicit ratings, based on our implicit feedback. However, as one may observe each method has its weknesses and strengths. A sigmoid-function considering the number of days between events is good for including our implicit knowledge about seasons into the ratings, but is not as effective if users has high spread in between events or low activity. Further, there may exist some clothes that has longer life-span than others, e.g. warm jackets that

generally are bought from September to March (7 months) compared to shorts which are generally bought from May to August (4 months), depending on where the store reside. Our second sigmoid function has the strength of always keeping the ratings for a user fresh, also for less active users, but it is weaker in differentiating between seasons - which can be seen if a user is on a hiatus between January and August, not using the application. Upon return all ratings would be based on his/hers winter activity, not penalizing the fact that the type of clothes generally bought in the store at this time are different than in January.

Optimally we would like to combine these two methods, taking their strenghts and weaknesses together trying to average them out in order to end up with generally better ratings - where we cannot trivally imagine scenarios as depicted above where our models would fail. The process of combining such ratings are in the Recommender Systems community called **blending** and is in many ways a seperate research area in itself, if done advanced enough. However, in the case of the naive and linear blend on can achieve results a magninute higher than for each method seperatly as seen in **??** .

find some refs using linear blending

When linearly blending M models m, we choose M factors f all adding up to 1.0, representing the weight of model m_i in the final blend. Then when calculating the final rating for item j we sum over all models:

$$r_j = \sum_{i=1}^{M} f_i * m_j \tag{5.5}$$

As one may observe given the linear blend between with factors $f_1 = 0.7$ and $f_2 = 0.3$ and the two ratings $m_1 = 5$ and $m_2 = 3$ for a given item, we can calculate the final rating as $0.7 \cdot 5 + 0.3 \cdot 3 = 4.4$. A weakness when linearly blending models in this way is the need for manually finding good weights for the M factors. There exists methods where this given a good test set can be done automatically, such as using Linear Regression or KNN blending. Many other blending schemes exists as well, such as Binned Linear Regression, Bagged Gradient Boosted Decision Tree (BGBDT), Neural Networks and Kernel Ridge Regression Blending [59] [103].

As we have multiple proposed models we present our results given various evaluation metrics and combinations of weights. Note that when a model has the weight 1.0 its equal to that the model has not been blended with any other model, and is included in the table below as a baseline.

Naive	**Sigmoid Count**	**Sigmoid Recent**	**RMSE**	**MAE**
1.0	0.0	0.0	X	X
0.0	1.0	0.0	X	X
0.0	0.0	1.0	X	X
0.6	0.2	0.2	X	X
0.5	0.3	0.2	X	X
0.4	0.3	0.3	X	X
0.3	0.3	0.4	X	X
0.2	0.4	0.4	X	X
0.1	0.4	0.5	X	X
0.0	0.5	0.5	X	X

As one may see, the best results are achivied when blending with...

todo: analyze and finish this table

5.2 Experimental Plan

Section 3.5 covers a wide range of evaluation metrics that measure different properties of the recommender system. This section will cover our experimental plan, starting off by looking at the goals for our experiments. The remaining parts of the section will describe the datasets used for evaluation, our evaluation methodology and our evaluation metrics of choice. The section will also describe the datasets used for evaluation, our evaluation methodology

We have the following goals for our experiment:

- Does our proposed implicit rating methods improve the recommendation quality over binary preference data?
- Compare the different implicit rating functions.
- Select the best combination of methods for the SoBazaar recommender system.

Supervisors: Any suggestions?
TODO: Discussion on how

Then the question is, how can we determine whether a method is better than another. The main reasons for implementing a recommender system is the desire to improve user satisfaction and to increase the economic success of a platform. Although both goals are interrelated they may be competing in some scenarios. The user might be more interested in purchasing the products with the best price-performance ratio, while the *owners* are more interested in showing the products that lead to the highest revenue for the business. For this purpose, a commercial recommender could/should consider implementing a reward attribute for items that show how much the company profits from its sale. This information can then e.g. be used in a combination the recommendation list of the recommender to produce the final recommendations.

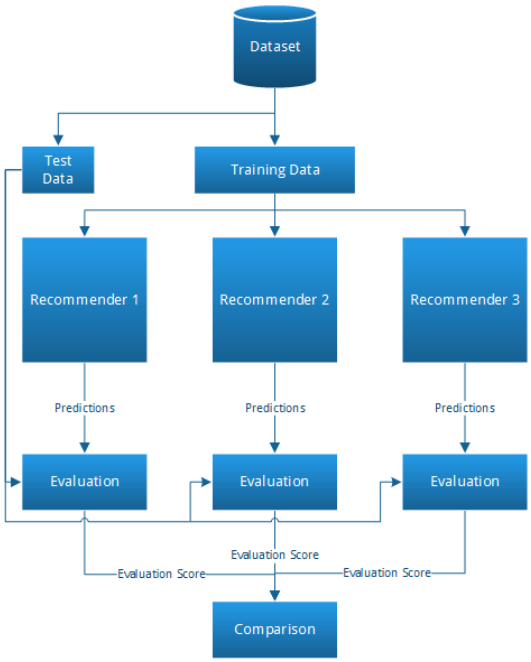


Figure 5.2: A traditional evaluation pipeline for evaluating recommender systems

In order to further specify our goals we therefore have to take a closer look at the recommender's task, its interface and the available data. The most typical task for a e-commerce recommender system is to determine an order of items, often with the purpose of creating a top-k list of items that is shown in a sidebar or on a dedicated page.

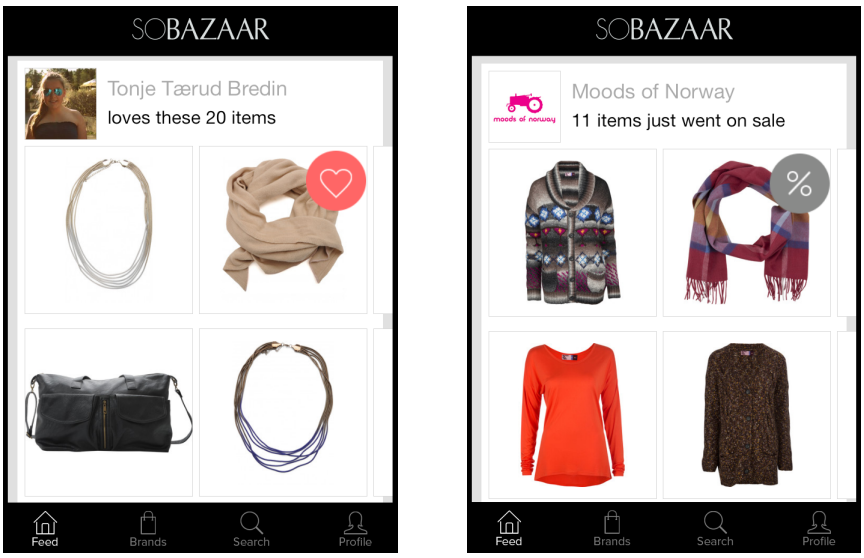


Figure 5.3: SoBazaar news feed

The above figure shows the SoBazaar feed. Recommendations are likely to be shown in a similar fashion, but instead as *recommended for you*. The interface currently let you scroll sideways over up to 20 items.

Ultimately, the goal of the experiment is to evaluate and measure the properties of the system, which we have identified as the most important for the systems success, and select the method that performs the best overall with respect to these properties.

5.2.1 Selecting datasets for evaluation

In addition to evaluate the methods on the SoBazaar dataset we want to make sure that our solution generalizes beyond our experimental dataset, in accordance to the general guidelines for experimental studies [95]. The data used for offline evaluation should match as closely as possible the data we expect the recommender system to face when it is deployed [54]. When selecting datasets for evaluation we focused on the following dataset properties:

- Size of dataset: Preferable as close as possible to SoBazaar (6 months from now) in terms of number of ratings, users and items.
- Different types of user feedback: Preferable different types of implicit feedback such as browsing and buying behavior.
- Domain: Preferably a domain as close to possible as the e-commerce domain with respect to the importance of factors such as recentness.

- Presence of features: To evaluate the hybrid methods.
- Timestamps: To evaluate the recentness mapping

We were unable to acquire any e-commerce datasets containing user browsing history, purchases etc. And we therefore had to turn for other domains for datasets...

Book-Crossing Dataset?

Use implicit feedback as *item-clicked* and explicit ratings greater than 5 as *item-liked*.

5.2.2 Simulating user behavior?

The SoBazaar Dataset

The SoBazaar dataset is smallest and sparsest of our datasets used for evaluation. The dataset contain ratings 15,252 given by 1,235 users to 3,386 items. We also have access to semi-structured product information collected/crawled from the online retailers for *some* items. In addition user data from the users can also be downloaded from Facebook.

Having such a small and sparse dataset has several implications. Firstly we have to avoid *wishful thinking* as we have very thin data, meaning that we cannot rely on getting reliable results. Secondly, our evaluation methodology must be *tailored* for small sparse datasets. When using cross-validation the number of folds depends on the size of the dataset. For large datasets, even 3-fold Cross Validation will be quite accurate, while for very sparse datasets, we may have to use leave-one-out in order to train on as many examples as possible. The advantages of using a large number of folds is that the bias of the true error rate estimators will be small (the estimator will be very accurate), with the disadvantages being that the variance of the true error rate estimator will be large in addition to increased computation time. To exemplify this we ran a small experiment on the SoBazaar data using IBCF, with $k - NN = 3$, experimenting with different K-fold values:

K-fold	min_{RMSE}	max_{RMSE}	Average	Variance
3	0.783	0.789	0.785	8.730×10^{-6}
5	0.759	0.802	0.781	3.363×10^{-4}
8	0.740	0.781	0.758	1.656×10^{-4}
10	0.718	1.026	0.810	0.0125

Table 5.3: Evaluation results from experimenting with different k-fold splits on the SoBazaar dataset

TODO: This table is not really necessary to include...

When increasing the number of folds we could see that it was unable to generate any recommendations at all for some folds, or getting really poor results, meaning that we get some difficult splits. E.g. when using 30 folds, IBCF was unable to provide any recommendations for 8 folds out of 30. By using more than 10 folds it is increasingly likely that we end up with a few or more *unrecommendable* instances in the test set, yielding no test result. Based on these results, and the general consensus that more folds are better for small datasets we believe that using between $5 - 10$-folds would be a good choice for model validation. Another alternative well suited for sparse datasets

is the *all but one* or the *leave one out* method, in which we remove one rating from the test users and try to predict the hidden rating.

Another important concern is whether or not to take the timestamps into consideration, which directly speaks against the use of cross-validation, as we wish to use the past interactions to predict future actions. When using the *leave one out* method one could e.g. select a predetermined set of test users based on some criteria and remove their latest rating and try to predict it and repeat the process any number of times. This is particularly relevance as our implicit mapping function takes recency into account.

Overview of the Datasets

Table 5.4 shows an overview of the datasets used for evaluation.

Dataset	Ratings	Users	Items	Sparsity	Rating Scale
SoBazaar	27,873	1,511	5855	99.69657	Implicit Ratings(1-5)
Dataset 2	-	-	-	-	

Table 5.4: Overview of the datasets used for evaluation

5.2.3 Simulating the Cold-Start Problem

To simulate the cold-start problem and evaluate how well our the different methods tackle the different cold-start situations we used the following evaluation methodology. As mentioned in Section 3.5.1 there is no common framework for assessing the cold-start performance of recommender systems. Our goal is to come up with *comprehensive* framework to assess the cold-start performance of our recommender systems. The following inputs changes the dataset over time:

- Existing users watch new items in the catalogue
- New users join the system and view their first item
- New items are added to the catalogue

The first input source has the effect of increasing the dataset density, the average user profile length, and the average number of views per item. The second input factor has the effect of decreasing both the dataset density and the average user profile length, as the new users that join the system have interacted with only a few items. Similarly, the third input factor has the effect of decreasing both the dataset density and the average number of views per item.

To simulate the cold-start user problem we propose splitting the users into two disjoint sets, similarly as in [99, 70], using 90% of the users for training and setting aside the remaining 10% for evaluation. We then train the model with e.g. 5, 15, 25 and 35 ratings and predict the remaining values. Alternatively one could train the model using e.g. 25% and 75% of each test users ratings. Similarly, to simulate the cold-start item problem we again split the items into two disjoint sets, using 90% of the items for training and the remaining 10% for evaluation. We then train the model with e.g. 20, 40, 60 and 80 ratings and predict the remaining values. The selection criteria for test items and users can differ from dataset to dataset. E.g. in [89, 90] the authors selected a subset of the users with more than 200 ratings, but you can not expect 10% all the

users for all datasets to have provided 200 ratings, so this number might be lowered if necessary. The implications of removing the top 10% of the raters from the SoBazaar dataset is fairly large as they stand for a large portion of the few ratings we have.

To evaluate the cold-start system performance we use the same method as described in [28] where the authors propose using a 75:25 training/test split, where we at random draw e.g. 35%, 50% and then finally use all (75%) of the ratings in the training set and predict the remaining 25%.

TODO: Add some justification...

For the SoBazaar dataset we select 10% of the users as test users, for a user to be selected as a test user, the user must have provided at least 20 ratings. The test user are drawn at random from the eligible candidates. We then train the model using 10%, 40% and 75% of their ratings and try to predict their remaining ratings. The reason for choosing percentages over hard limits is due to the fact that the ratings are distributed unevenly among the users. As we have a very low number of ratings and a large item collection we had to use only 5% of the items as test items, where each test item have been rated by atleast 15 users. We train the model using 10%, 40% and 75% of their ratings and try to predict their remaining ratings. As we have access time timestamps, we split the users and items based on timestamp. E.g. for the 10% user split we train the model with their initial ratings and try to predict their following ratings. To evaluate the cold-start system performance we split the dataset in a test and training set using 20% of the ratings for testing and then train the model using 40%, 60% and 80% of the ratings for training. It is important to note that this process should be repeated multiple times, as the chance of getting an *unfortunate* split is highly probable due to the dataset size. For the cold-start system task we also split the dataset based on timestamps, meaning that the test set consists of the most recent ratings.

5.2.4 Evaluation Metrics

A large variety of metrics have been published, and some of these metrics are highly correlated [56]. There is little guidance for evaluating recommender systems and choosing metrics. However, there are some important questions one should ask oneself when selecting evaluation metrics:

- Which aspects of the usage scenario and the data influence the choice?
- Which metrics are applicable?
- What does these metrics express?
- What are the differences among them?
- Which metric represent our user-case best?
- How much do the metrics suffer biases?

It is safe to assume that the users are more interested in the top ranked items, than rating predictions for the entire item collection. Evaluation of top-k recommendations suggests a classification or ranking task, evaluation should therefore focus on classification or ranking metrics.

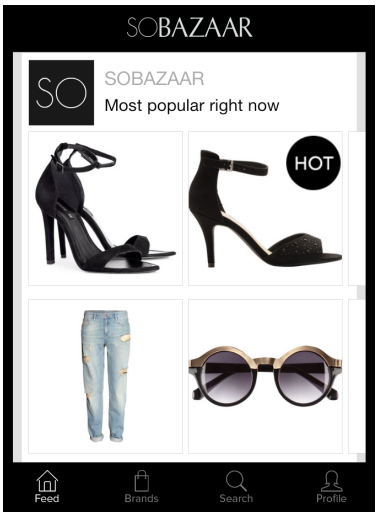


Figure 5.4: The Figure shows how the most-popular recommendations are shown in the application

Much like the social feed shown in Figure 5.3 up to 20 popular items can be shown. It would also be beneficial to rank these recommendations putting the most popular/relevant items in the first 4 slots and sorting the remaining items in descending order based on popularity. This implicates that a metric that measures the overall ranking is not appropriate, and that we only should measure the ranking quality of the k items being shown, the other items are irrelevant. The above reasoning lead us to take a closer look at classification and ranking accuracy metrics.

The area under curve (AUC) is a popular classification accuracy metric. ROC curves provide a graphical representation for the performance of a recommender system, by plotting the recall (True positive rate) against the fallout (False positive rate) for increasing recommendation set size. A perfect recommender would therefore yield a ROC curve that goes straight up towards 1.0 recall and 0.0 fallout until all relevant items are retrieved. Afterwards it would go straight towards 1.0 fallout while the remaining irrelevant items follow. One therefore obviously aims to maximize the area under the curve (AUC). The higher up all relevant items (True positives) are in the recommendation list, the higher the AUC score will be. AUC can therefore be used as a single measure for the overall quality of a recommender system. Table 5.5 shows an example of the AUC values when varying the position of a single relevant document through the recommendation list.

#Example	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	AUC
1	✔	✗	✗	✗	✗	✗	✗	✗	✗	✗	1.000
2	✗	✔	✗	✗	✗	✗	✗	✗	✗	✗	0.889
3	✗	✗	✔	✗	✗	✗	✗	✗	✗	✗	0.778
4	✗	✗	✗	✔	✗	✗	✗	✗	✗	✗	0.667
5	✗	✗	✗	✗	✔	✗	✗	✗	✗	✗	0.556
6	✗	✗	✗	✗	✗	✔	✗	✗	✗	✗	0.444
7	✗	✗	✗	✗	✗	✗	✔	✗	✗	✗	0.333
8	✗	✗	✗	✗	✗	✗	✗	✔	✗	✗	0.222
9	✗	✗	✗	✗	✗	✗	✗	✗	✔	✗	0.111
10	✗	✗	✗	✗	✗	✗	✗	✗	✗	✔	0.000

Table 5.5: Varying the position of a single relevant item on a four out of ten recommendation list

One should also be aware of that the AUC scores can be highly inaccurate, especially for cold-start users, users which the recommender is unable to provide more than a handful of recommendations for. This is due to the fact that all unknown ratings are appended in random order after the known recommendations. E.g. lets say the recommender is able to generate 10 recommendations for a user out of 4000 items, and that one out of two test items for that user is not in the recommended list from the recommender. Where the last test item is appended in the recommendation list can mean the difference between a AUC score of above 0.9 to less than 0.6 if the item is appended at the end of the list, the results can be even more severe if none of the items is in the initial recommended list, then the results for that user will be completely random. The most obvious solution to this problem would be to use resampling or to repeat the experiment multiple times where the items not in the recommendation list are drawn at random and the results averaged over all the trials.

However, a frequently uttered point of criticism is that users are often more interested in the items at the top of a recommendation list but that the AUC measure is equally affected by swaps at the top or the bottom. To *complement* AUC, we also wish to measure the rank accuracy, to get a better overall picture of the systems performance. This means that we want to measure whether or not highly rated items such as purchases, likes etc. are ranked higher than less relevant items such as clicks, as having these highly ranked items higher up in the recommendation list aligns with our previously mentioned financial incentives.

When using rank accuracy metrics, it is worth knowing whether one is measuring total or partial orderings. Most rank accuracy metrics (e.g. Kendall's tau and Spearman's rho) compare two total ordering. The problem with these measures is that we in most cases are not provided with a full ranking of the items as most recommendation algorithms only generate a partial list of items that are likely to be preferred by a user. The remaining items would therefore have to be concatenated in random order. The recommendation list can also consist of several items with similar rating that can appear in varying orders. Therefore, in order to create a full ranking of the items all preference values for the user have to be known. Since the user can express the same rating for similar items, the list will again contain groups of items that can appear in arbitrary order. However, the largest problem is posed by items for which no rating is known. These items could hold an arbitrary place within the ranking. Again, consider

an example where a user e.g. have rated 5 items out of 4000, and the recommender is able to recommend only 10 items for that user. To measure the rank accuracy we would have to randomly add 3995 items to the users known preference list and 3990 to the users prediction list. Comparing the order of these lists would not make much sense. The bottom line is that in most cases a rank metric for partial ordering would be more appropriate for comparing recommendation lists that are produced by recommenders to item rankings from known user preferences.

We assume that we can recommend at most k items for each user at a time. It also pays to submit all k recommendations, because we are not penalized for bad guesses. We also assume that the order matters, so it is better to submit more certain recommendations first, followed by recommendations we are less sure about. Which means that we basically select the k best candidates in order. To reduce the number of randomness in the results one could choose to only look at the ranking of the top k. However, the problem is that the likeliness of finding a *hidden* item in e.g. the top 20 recommended items is not very large when one is working with large item catalogs. The problem then, is how to select the value of k. Setting it to low you risk ending up with many 0.0 values, and when setting it to high you risk including to many *randomly* concatenated ratings in the results.

Mean average precision (MAP@k), described in Section 3.5.1 is a popular metric for search engines and is applied, for example, to report results at the Text Retrieval Conference (TREC). The main weakness with regard to recommender systems is that it assumes that the user is interested in finding many relevant documents for each query, and does not look at the relevance of the items. Consider the following examples where we have three lists of hidden items and three recommendations lists. All none relevant items are labeled with the item-id 0.

Example	Actual	Recommended	AP@4
1	[1,2,3,4]	[1,0,0,0]	0.250
2	[1,2,3,4]	[2,0,0,0]	0.250
3	[1,2,3,4]	[3,0,0,0]	0.250

Table 5.6: AP@4 Scores

As you can average precision does not consider the order of the actual item list. We want a way to reward the recommender for getting the highly ranked items right.

Normalized Discounted Cumulative Gain ($nDCG_k$), described in Section 3.5.1 is another metric to measure the rank accuracy. It is based on two main assumptions; (1) Highly relevant items are more useful than marginally relevant ones, (2) The lower the ranked position of a relevant item, the less useful it is for the user, since it is less likely to be *examined*. The maybe most interesting aspect of nDCG is that it contains an utility function rel_i. One can replace the original utility function and replace it with a function that is more relevant to the application. Two such examples include:

- $rel_i = 1/log(i+k)$, where i is the index of the item in the actual sorted rating list, where k is a constant.
- $rel_i = u(i)$, simply assign the rating of an item as its relevance.

For our application we believe it to be beneficial to use the rating of the item as a relevance score, as we could have multiple items with highly similar ratings (thus the

same relevance). When using a low k value for the logarithmic alternative this could mean that the relevance between two almost similarly rated items might end up being disproportionately large. We will also most likely look at the 20 top items, which means that the difference between the lower ranked items will be fairly small, as the logarithmic function converges. A third alternative would be to also use some linear decay function. The following table shows the *nDCG* score for a set of examples where the function $rel_i = 1/log(i+1)$ is used to assign the relevance score to item i in the sorted list of actual ratings. E.g. item one is assigned a relevance score of $1/(log(2)$ and so on. We hope the following table will give the reader a better idea of how *nDCG works*.

Example	Actual	Recommended	nDCG
1	[1,2,3,4]	[1,0,0,0]	0.470
2	[1,2,3,4]	[2,0,0,0]	0.360
3	[1,2,3,4]	[3,0,0,0]	0.330
4	[1,2,3,4,5,6,7,8,9,10]	[1,2,3,4,5,6,7,8,9,10]	1.000
5	[1,2,3,4,5,6,7,8,9,10]	[10,9,8,7,6,5,4,3,2,1]	0.900
6	[1,2,3,4,5,6,7,8,9,10]	[1,2,0,0,0,0,0,0,0,0]	0.440
7	[1,2,3,4,5,6,7,8,9,10]	[1,0,0,3,0,0,0,0,0,0]	0.390
8	[1,2,3,4,5,6,7,8,9,10]	[4,0,0,0,8,0,0,0,0,0]	0.280
9	[1,2,3,4,5,6,7,8,9,10]	[0,0,0,0,5,0,0,0,0,10]	0.130
10	[1,2,3,4,5,6,7,8,9,10]	[0,0,0,0,0,0,0,0,9,10]	0.110

Table 5.7: nDCG Test Examples

As you can see from the above mentioned examples nDCG will give us an indication whether or not our recommender ranks *highly relevant* items above those who are less relevant. As for the limitations, nDCG is designed for situations of non-binary notions of relevance, thus it cannot be used in our experiment where we wish to compare binary ratings with implicit ratings. The main difficulty encountered in using nDCG is the unavailability of an ideal ordering of results when only partial relevance feedback is available. This is often the case when we have several equally good results. This is especially true when the metric is limited to only first few results as it is often done in practice.

We chose take a closer look at the different ways of calculating nDCG. There are two commonly used methods used to calculate the DCG and IDCG scores.

- Method 1 $DCG_p = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{log_2(i)}$
- Method 2: $DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i}-1}{log_2(i+1}$)

The following table shows the differences between Method 1 and Method 2 on a few selected examples. Here we set the value of rel_i equal to the implicit rating of item i.

#Ex	Actual	Recommended	$nDCG^1$	$nDCG^2$
1	$[1_5, 2_{3.8}, 3_{3.55}, 4_{2.78}, 5_{1.3}]$	[5,0,0,0,0]	0.113	0.026
2	$[1_5, 2_{3.8}, 3_{3.55}, 4_{2.78}, 5_{1.3}]$	[0,0,1,0,0]	0.217	0.275
3	$[1_5, 2_{4.8}, 3_{3.55}, 4_{2.78}, 5_{2.2}, 6_{2.0}, 7_{1.77}, 8_{1.3}]$	[2,3,4,5,6,7,8,0]	0.827	0.682
4	$[1_5, 2_{4.8}, 3_{3.55}, 4_{2.78}, 5_{2.2}, 6_{2.0}, 7_{1.77}, 8_{1.3}]$	[1,2,0,0,0,0,0,0]	0.592	0.805
5	$[1_5, 2_{4.93}, 3_{2.88}, 4_{1.85}, 5_{1.8}, 6_{1.77}, 7_{1.63}, 8_{1.52}]$	[1,0,0,0,0,0,0,0]	0.394	0.544
6	$[1_5, 2_{4.93}, 3_{2.88}, 4_{1.85}, 5_{1.8}, 6_{1.77}, 7_{1.63}, 8_{1.52}]$	[3,4,5,6,7,8,0,0]	0.542	0.206

Table 5.8: Comparison of nDCG scores for different methods of computing DCG. Each item in the Actual list is on the form $ItemId_{Rating}$, the first item in the actual list of example one therefore has en ItemId of 1 and a Rating of 5. All non relevant items are given the ItemId 0 in the Recommended list.

As you can see Method 1 is a bit more *balanced* than Method 2, as it does not give as much weight to highly rated items. Especially example 5 and 6 highlights this as Method 2 prefers retrieving one highly rated item over retrieving multiple less relevant ones. Method 2 is ultimately closer to what we want in addition to being commonly used by web search companies and on the data science competition platform Kaggle [16]. We have also chosen to truncate the recommendation list and only calculate the nDCG of the k top items. Because nDCG has a relatively low positional discount, it allows us to set k fairly high, but this only makes sense if we believe that the user will read large portions of the recommendation list.

In addition to looking at the above mentioned metrics it would be interesting to see how the different sparsity levels affect both the user- and item-space coverage of the different methods when evaluating the cold-start system performance, as having a recommender that can only recommend a limited set of items to a small portion of the users is *unfortunate*. The user-space coverage is the number of users the recommender is able to produce recommendations for while the item-space coverage is measured by looking at how many of the items are recommendable.

TODO: Spend any more time looking at partial order ranking measures?...

5.2.5 Comparing ratings

Figure **??** shows the process of evaluating a set of generated ratings. The *million dollar question* is: how to do we evaluate a set of ratings?

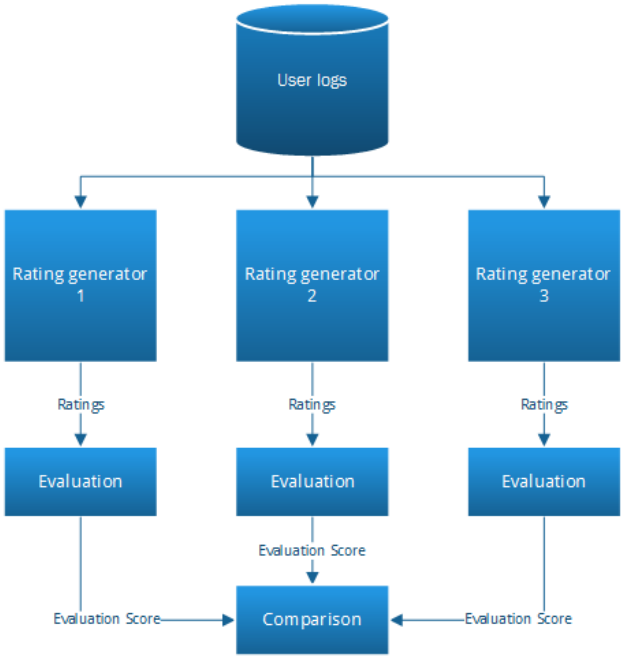


Figure 5.5: The Figure shows the process of comparing ratings

Evaluating and comparing a set of ratings is not something you often encounter in the literature. We have a few initial theories of how it can be done:

TODO: Are there any articles on something similar?

1. Using the log data as the ground truth. Is it possible to justify that one method is better than another with observations from the log data?

2. Can we put the ratings into a traditional recommender system evaluation pipeline and use the results from that to compare the ratings. In that case, what evaluation metrics are suitable?

The bottom line is that evaluating the quality of a set of generated ratings is hard. We could say that we believe that better ratings gives us better recommendation results, and use this as a basis for our evaluation. However, this means that the results we get from testing the different algorithms with different rating sets should be comparable. The problem with this is that most evaluation metrics consider the supplied ratings as the ground truth and bases its evaluation on these numbers. This automatically disqualifies any evaluation metrics that consider the rating values such as e.g. MAE.

5.2.6 Implicit Ratings vs. Binary Preference

This subsection will attempt to explain how we will attempt to determine whether our implicit gives some added qualities in form of improved recommendations produced using binary preferences.

Problems: Methods that can incorporate both types of feedback... Problems: Methods for implicit ratings (Explain why traditional item-based cf is not suited...)

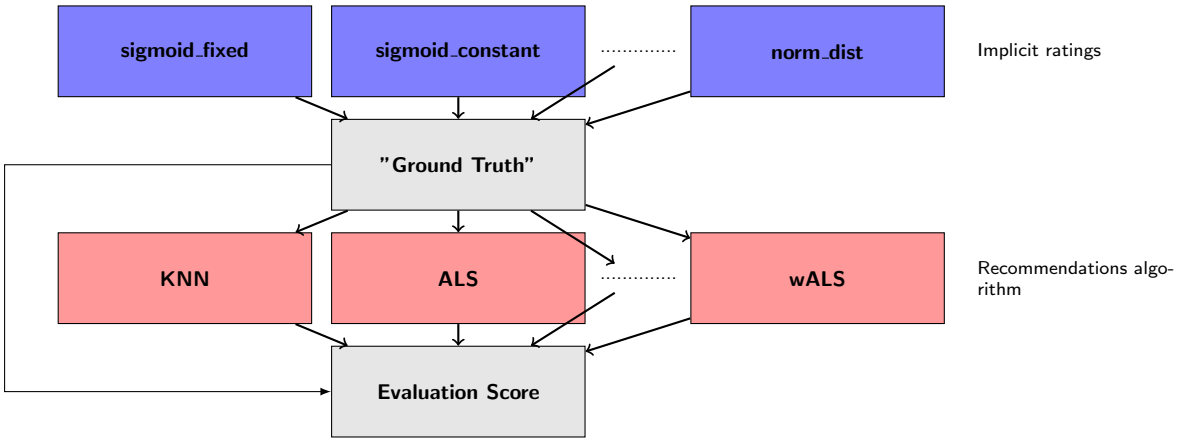


Figure 5.6: Overview of how the generated ratings from the implicit feedback can be evaluated.

The blue boxes in figure 5.6 are functions used to produce the implicit ratings based on the implicit feedback. Based on this, the system gets an estimated *ground truth*, which can be used to evaluate the different recommendation algorithms show in the red boxes. Based on the evaluation score something can be said about the implicit feedback conversion to implicit ratings and the recommendation algorithms used.

5.2.7 A Comparison of Implicit Rating Mapping Functions

5.2.8 Combining Implicit Ratings With Existing Cold-Start Solutions

Only use cold-start splits for this sections tables.

Comparison of recency functions in filterbots? E.g. does the PopularityBot perform better when only considering the last x weeks or all the data? Similarly for the criticBot

5.3 Experimental Setup

5.3.1 Computer Specs

5.3.2 Requirements

5.3.3 Parameter Settings

6

Evaluation

Contents

6.1 Experimental Results

In this section we present the experimental results...

6.1.1 The SoBazaar Dataset (Implicit Ratings)

This subsection will present the experimental results for the SoBazaar dataset using implicit ratings.

	AUC			nDCG@20			MAP@20			HLU			IS Coverage			US Coverage		
Model	40%	60%	80%	40%	60%	80%	40%	60%	80%	40%	60%	80%	40%	60%	80%	40%	60%	80%
MostPopular	0.5982	0.6028	0.6071	0.0254	0.0256	0.0187	0.0136	0.0156	0.0089	5.7803	5.7803	2.8902	1.0	1.0	1.0	1.0	1.0	1.0
MostPopular + FB	0.7557	0.7687	0.7714	0.0254	0.0256	0.0174	0.0157	0.0156	0.0085	5.7803	5.7803	2.8902	1.0	1.0	1.0	1.0	1.0	1.0
Rec 3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Table 6.1: SoBazaar Cold-start System Evaluation Results using Implicit Ratings (count_sigmoid_fixed_sr-3.5.txt)

	AUC			nDCG@20			MAP@20			HLU			IS Coverage			US Coverage		
Model	10%	40%	75%	10%	40%	75%	10%	40%	75%	10%	40%	75%	10%	40%	75%	10%	40%	75%
MostPopular	0.7385	0.7391	0.7533	0.0247	0.0400	0.0474	0.0087	0.0135	0.0219	5.7143	7.4561	8.7336	1.0	1.0	1.0	1.0	1.0	1.0
MostPopular + FB	0.8569	0.8587	0.8672	0.0254	0.0400	0.0474	0.0089	0.0135	0.0219	2.0179	5.7143	8.7336	1.0	1.0	1.0	1.0	1.0	1.0
Rec 3	-	-	-	-	-	-	-	-	-		-							-
Rec 4	-	-	-	-	-	-	-	-	-		-							-

Table 6.2: SoBazaar Cold-start Item Evaluation Results using Implicit Ratings (count_sigmoid_fixed_sr-3.5.txt)

	AUC			nDCG@20			MAP@20			HLU			IS Coverage			US Coverage		
Model	10%	40%	75%	10%	40%	75%	10%	40%	75%	10%	40%	75%	10%	40%	75%	10%	40%	75%
MostPopular	0.9775	0.9807	0.9811	0.0804	0.0937	0.1045	0.0449	0.0468	0.0533	4.0595	6.0879	7.6238	1.0	1.0	1.0	1.0	1.0	1.0
MostPopular + FB	0.9877	0.9895	0.9897	0.0804	0.0937	0.1045	0.0449	0.0468	0.0533	4.0595	6.0879	7.6238	1.0	1.0	1.0	1.0	1.0	1.0
Rec 3	-	-	-	-	-	-	-	-	-		-							-
Rec 4	-	-	-	-	-	-	-	-	-		-							-

Table 6.3: SoBazaar Cold-start User Evaluation Results using Implicit Ratings (count_sigmoid_fixed_sr-3.5.txt)

	AUC			nDCG@20			MAP@20			HLU			IS Coverage			US Coverage		
Model	40%	60%	80%	40%	60%	80%	40%	60%	80%	40%	60%	80%	40%	60%	80%	40%	60%	80%
NameID: MostPopular mode: item	0.9833	0.6840	0.6203	0.0000	0.0000	0.0000	0.0435	0.0606	0.0647	0.0000	0.0000	0.0000	0.9992	0.9992	0.9992	1.0000	1.0000	1.0000
NameID: MostPopular mode: system	0.5679	0.5913	0.5916	0.0029	0.0024	0.0015	0.0220	0.0204	0.0201	0.0000	0.4149	0.0000	0.9989	0.9990	0.9991	1.0000	1.0000	1.0000
NameID: MostPopular mode: user	0.7664	0.7514	0.7405	0.0000	0.0000	0.0000	0.0081	0.0029	0.0024	0.0000	0.0000	0.0000	0.9992	0.9992	0.9992	1.0000	1.0000	1.0000
NameID: Random mode: item	0.4893	0.4979	0.4985	0.0007	0.0015	0.0021	0.0000	0.0004	0.0016	0.0000	0.2301	0.0967	0.9992	0.9992	0.9992	1.0000	1.0000	1.0000
NameID: Random mode: system	0.5210	0.4918	0.5188	0.0031	0.0040	0.0009	0.0002	0.0023	0.0008	0.0000	0.0000	0.0000	0.9989	0.9990	0.9991	1.0000	1.0000	1.0000
NameID: Random mode: user	0.5055	0.5108	0.4934	0.0012	0.0008	0.0005	0.0014	0.0007	0.0001	0.0000	0.0000	0.0000	0.9992	0.9992	0.9992	1.0000	1.0000	1.0000
NameID: Zero mode: item	0.0170	0.3164	0.3794	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
NameID: Zero mode: system	0.5619	0.5504	0.5614	0.0000	0.0045	0.0018	0.0000	0.0010	0.0007	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
NameID: Zero mode: user	0.7184	0.7115	0.6864	0.0079	0.0031	0.0009	0.0024	0.0008	0.0006	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
NameID: svd mode: item	0.8101	0.6013	0.5748	0.0134	0.0132	0.0158	0.0362	0.0304	0.0275	0.6303	0.6135	1.0155	1.0000	0.9992	1.0000	1.0000	1.0000	1.0000
NameID: svd mode: system	0.5498	0.5549	0.5699	0.0179	0.0065	0.0044	0.0045	0.0057	0.0050	1.3274	0.0000	0.0000	0.9989	0.9990	1.0000	1.0000	1.0000	1.0000
NameID: svd mode: user	0.7814	0.7788	0.7743	0.0013	0.0012	0.0005	0.0189	0.0058	0.0034	0.0000	0.0000	0.0000	0.9992	0.9992	0.9992	1.0000	1.0000	1.0000
NameID: ItemKNN mode: item	0.2779	0.4035	0.4405	0.0000	0.0000	0.0000	0.0014	0.0059	0.0101	0.0000	0.0000	0.0000	0.6422	0.5735	0.6134	0.9174	0.9159	0.9180
NameID: ItemKNN mode: system	0.5555	0.5729	0.5729	0.0161	0.0015	0.0057	0.0067	0.0005	0.0011	3.0973	0.0000	0.0000	0.8922	0.9039	0.9455	0.9948	0.9530	0.9355
NameID: ItemKNN mode: user	0.7377	-1.0000	0.7337	0.0121	-1.0000	0.0003	0.0000	nan	0.0028	0.0000	-1.0000	0.8000	0.8491	0.8173	0.9508	0.9314	0.8943	0.9337
NameID: WRMF mode: item	0.8422	0.6315	0.5858	0.0070	0.0102	0.0091	0.0416	0.0358	0.0374	0.6303	0.9202	0.9671	0.9992	0.9992	0.9992	1.0000	1.0000	1.0000
NameID: WRMF mode: system	0.5230	0.5329	0.5592	0.0080	0.0014	0.0052	0.0030	0.0040	0.0032	0.8850	0.0000	0.8000	0.9989	0.9990	0.9991	1.0000	1.0000	1.0000
NameID: WRMF mode: user	0.7602	0.7580	0.7507	0.0013	0.0026	0.0000	0.0130	0.0044	0.0025	0.0000	0.5333	0.0000	0.9992	0.9992	0.9992	1.0000	1.0000	1.0000
NameID: UserKNN mode: item	0.9426	0.6679	0.6094	0.0000	0.0000	0.0000	0.0418	0.0294	0.0280	0.0000	0.0000	0.0000	0.9992	0.9992	0.9992	1.0000	1.0000	1.0000
NameID: UserKNN mode: system	0.5508	0.5672	0.5684	0.0023	0.0066	0.0014	0.0035	0.0112	0.0123	0.0000	0.0000	0.0000	0.9989	0.9990	0.9991	1.0000	1.0000	1.0000
NameID: UserKNN mode: user	0.7925	0.7979	0.7839	0.0016	0.0000	0.0000	0.0184	0.0072	0.0062	0.0000	0.0000	0.0000	0.9992	0.9992	0.9992	1.0000	1.0000	1.0000
NameID: BPRMF mode: item	0.9214	0.6576	0.6071	0.0012	0.0000	0.0000	0.0285	0.0352	0.0384	0.0000	0.0000	0.0000	0.9992	0.9992	0.9992	1.0000	1.0000	1.0000
NameID: BPRMF mode: system	0.5885	0.6359	0.6072	0.0027	0.0025	0.0045	0.0030	0.0043	0.0055	0.4425	0.0000	0.4000	0.9989	0.9990	0.9991	1.0000	1.0000	1.0000
NameID: BPRMF mode: user	0.7490	0.7408	0.7361	0.0018	0.0005	0.0007	0.0042	0.0016	0.0014	0.0000	0.0000	0.2667	0.9992	0.9992	0.9992	1.0000	1.0000	1.0000

Table 6.4: Testur

6.2 Development Process

Good

Bad

6.3 Result Evaluation

Testing of preliminary study

Testing of code functionality

Types of testing not used

6.4 Issues

7

Conclusion

Contents

7.1 Final Product

7.2 Related Work

7.3 Future Work

As previously mentioned in the evaluation section the main reasons for implementing a recommender system is the desire to improve user satisfaction and to increase the economic success of a platform. We believe it would be interesting to look into how one could implement a reward attribute for the items, that factors in how much Telenor will profit from its sale.

$$ExpectedReturn_i = P(Sale_i) * Reward(i) \tag{7.1}$$

The question then, is how this information can e.g. be used/incorporated into the recommender to increase profits without sacrificing (too much) user satisfaction.

However, the content-based filtering methods require rich descriptions of items and well built and well informed user profiles. These ideal cases are rare in real applications. This dependence on the quality and structure of data is the main weakness of methods based on content.

Appendix A

Data

Variable	*Example*
”app_version”	0.3
”user_agent”	”SOBAZAR 0.3 (iPhone; iPhone OS 6.1.4; Scale/2.00; nb_NO)”
”product_type”	”product”
”server_time_stamp”	”2013-10-24T11:33:17.632Z”
”dy”	24
”origin_ui”	”storefront”
”currency”	”kr”
”country_name”	”Norway”
”price”	1995
”product_name”	”DWS No47”
”tag_name”	”NULL”
”tag_id”	”NULL”
”storefront_name”	”BIK BOK”
”event_id”	”product_purchase_intended”
”age_target”	”Any”
”epoch_day”	16002
”mo”	10
”yr”	2013
”product_id”	2298002
”event_location”	Geo Location
”ipAddress”	IP
”contentDescription”	null
”sessionId”	null
”contentId”	null
”instKey”	”ed4c76251ac47da54299d8c0bce3dca6”
”viewer”	null
”ts”	NumberLong(”1382614397632”)
”gender_target”	”Female”
”client_time_stamp”	”NULL”
”login_type”	”NULL”
”transaction_id”	”N/A”
”service_id”	”SOBAZAR”
”platform”	”iPhone”
”epoch_week”	2286
”storefront_id”	23002
”hr”	11
”tag_position”	”NULL”
”time_stamp”	”2013-10-24T13:33+0200”
”retailer_brand”	13001
”storefront_position”	2

TODO: Add an explanation to each row, and maybe remove example?

Event Name	*Explanation*
”product_wanted”	-
”product_detail_clicked”	-
”storefront_clicked”	-
”activity_clicked”	-
”around_me_clicked”	-
”user_logged_in”	-
”featured_storefront_clicked”	-
”friend_invited”	-
”app_started”	-
”stores_map_clicked”	-
”product_purchase_intended”	-
”store_clicked”	-
”featured_collection_clicked”	-

Table A.2: Table of the different events that can be triggered by the user and an explanation

TODO: Add all the different event types

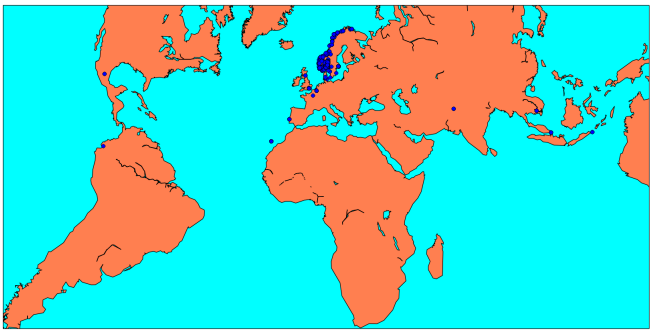


Figure A.1: This figure shows the location of the events from all over the world. For a cropped version with focus on Norway and the surrounding countries, refer to 2.7

Appendix B

Requirements

B.1 Functional Requirements

B.2 Non Functional Requirements

Appendix C

Design

Appendix D

Implementation

D.1 Implemented Functional Requirements

FR 1: Blablaba

FR 2: ~~Blablaba~~

D.2 Implemented Non Functional Requirements

NFR 1: Blablaba

NFR 2: ~~Blablaba~~

Bibliography

[1] Clothia. https://clothia.com. Clothia web page, [Online; accessed 7-05-2014].

[2] Farfetch. https://www.farfetch.com. Farfetch web page, [Online; accessed 8-05-2014].

[3] Modcloth web page. http://www.modcloth.com/. ModCloth web page, [Online; accessed 7-05-2014].

[4] Myntra. https://myntra.com. Myntra web page, [Online; accessed 8-05-2014].

[5] Polyvore web page. http://www.polyvore.com/. Polyvore web page, [Online; accessed 7-05-2014].

[6] Trendabl. https://trendabl.com. Trendabl web page, [Online; accessed 7-05-2014].

[7] Ustrendy. http://www.ustrendy.com/. UsTrendy web page, [Online; accessed 7-05-2014].

[8] Magento enterprise customer success story gant. http://info.magento.com/rs/magentocommerce/images/Gant1.pdf, 2012. Magento Enterprise Customer Success Story GANT online article, [Online; accessed 12-05-2014].

[9] Multi-armed bandit experiments. https://support.google.com/analytics/answer/2844870?hl=en, 2013. Google web page, [Online; accessed 29-05-2014].

[10] Ellos. http://www.Ellos.com, 2014. Ellos web page, [Online; accessed 9-05-2014].

[11] Epinions. https://snap.stanford.edu/data/soc-Epinions1.html/, 2014. Epinions Datasets, [Online; accessed 4-04-2014].

[12] Facebook. http://www.facebook.com/, 2014. Facebook webpage, [Online; accessed 21-05-2014].

[13] Fashiolista. http://www.Fashiolista.nu, 2014. Fashiolista web page, [Online; accessed 9-05-2014].

[14] Flink. https://itunes.apple.com/us/app/id798552697, 2014. Flink iPhone fashion application, [Online; accessed 8-05-2014].

[15] Gant. http://www.gant.com/, 2014. GANT webpage, [Online; accessed 12-05-2014].

[16] Kaggle. https://www.kaggle.com, 2014. Kaggle web page, [Online; accessed 8-05-2014].

[17] Lookbook. http://www.LookBook.nu, 2014. LookBook web page, [Online; accessed 9-05-2014].

[18] Lyst. https://www.lyst.com, 2014. Lyst web page, [Online; accessed 8-05-2014].

[19] Motilo. https://www.motilo.com, 2014. Motilo web page, [Online; accessed 8-05-2014].

[20] Movielens. http://grouplens.org/datasets/movielens/, 2014. Movielens Datasets, [Online; accessed 4-04-2014].

[21] Myhabit. http://www.MyHabit.com, 2014. MyHabit web page, [Online; accessed 9-05-2014].

[22] Rue la la. http://www.RueLaLa.com, 2014. Rue La La web page, [Online; accessed 9-05-2014].

[23] Shopstyle. http://www.ShopStyle.com, 2014. ShopStyle web page, [Online; accessed 9-05-2014].

[24] Statistics norway. https://www.ssb.no, 2014. Statistics Norway webpage, [Online; accessed 21-05-2014].

[25] Zalando. http://www.Zalando.com, 2014. Zalando web page, [Online; accessed 8-05-2014].

[26] Jorij Abraham. ecommerce benchmark for the fashion industry. http://www.ecommercefoundation.org/about-us/ecommerce-benchmark-fashion-retail, 2012. ecommercefoundation webpage, [Online; accessed 12-05-2014].

[27] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, June 2005.

[28] Deepak Agarwal and Bee-Chung Chen. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '09, pages 19–28, New York, NY, USA, 2009. ACM.

[29] Deepak Agarwal and Bee-Chung Chen. flda: Matrix factorization through latent dirichlet allocation. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 91–100, New York, NY, USA, 2010. ACM.

[30] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[31] 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.

[32] M. Barnard. *Fashion as Communication*. Routledge, 2002.

[33] Robert M Bell and Yehuda Koren. Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on*, pages 43–52. IEEE, 2007.

[34] James Bennett, Stan Lanning, and Netflix Netflix. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[35] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[36] John S. Breese, David Heckerman, and Carl Kadie. Empirical analysis of predictive algorithms for collaborative filtering. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, pages 43–52, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[37] Robin Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

[38] Ò. Celma. *Music Recommendation and Discovery in the Long Tail*. PhD thesis, Universitat Pompeu Fabra, Barcelona, 2008.

[39] Yoon Ho Cho, Jae Kyeong Kim, and Soung Hie Kim. A personalized recommender system based on web usage mining and decision tree induction. *Expert Systems with Applications*, 23(3):329 – 342, 2002.

[40] Keunho Choi, Donghee Yoo, Gunwoo Kim, and Yongmoo Suh. A hybrid online-product recommendation system: Combining implicit rating-based collaborative filtering and sequential pattern analysis. *Electronic Commerce Research and Applications*, 11(4):309–317, 2012.

[41] Mark Claypool, David Brown, Phong Le, and Makoto Waseda. Inferring user interest. *IEEE INTERNET COMPUTING*, 5:32–39, 2001.

[42] Paolo Cremonesi, Franca Garzottto, and Roberto Turrin. User effort vs. accuracy in rating-based elicitation. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 27–34, New York, NY, USA, 2012. ACM.

[43] Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Trans. Inf. Syst.*, 22(1):143–177, January 2004.

[44] Mukund Deshpande and George Karypis. Selective markov models for predicting web page accesses. *ACM Trans. Internet Technol.*, 4(2):163–184, May 2004.

[45] Divante. E-commerce in fashion industry. http://www.slideshare.net/divanteltd/e-commerce-in-fashion-industry, 2014. SlideShare webpage, [Online; accessed 12-05-2014].

[46] Kathryn Christine Dutton. The affect of garment attributes on the purchase intentions of fifteen to twenty-five year old females. page 142, 2006.

[47] B. Efron and R.J. Tibshirani. *An Introduction to the Bootstrap*. Chapman & Hall/CRC Monographs on Statistics & Applied Probability. Taylor & Francis, 1994.

[48] Michael Eichler. Introduction to the bootstrap. http://galton.uchicago.edu/~eichler/stat24600/Handouts/bootstrap.pdf, 2003. Introduction to the Bootstrap webpage, [Online; accessed 21-05-2014].

[49] Madison City Express. Madison city express (newspaper), 2 feb. 2/5, 1834.

[50] Z. Gantner, L. Drumond, C. Freudenthaler, S. Rendle, and L. Schmidt-Thieme. Learning attribute-to-feature mappings for cold-start recommendations. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*, pages 176–185, Dec 2010.

[51] Pranab Ghosh. From explicit user engagement to implicit product rating.

[52] David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35(12):61–70, December 1992.

[53] Nathaniel Good, J. Ben Schafer, Joseph A. Konstan, Al Borchers, Badrul Sarwar, Jon Herlocker, and John Riedl. Combining collaborative filtering with personal agents for better recommendations. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 439–446, Menlo Park, CA, USA, 1999. American Association for Artificial Intelligence.

[54] Asela Gunawardana, Guy Shani, and Lyle Ungar. A survey of accuracy evaluation metrics of recommendation tasks. *Journal of Machine Learning Research*, pages 2935–2962.

[55] C.-Hennig Hanf and Bernhard Wersebe. Price, quality, and consumers' behaviour. *Journal of Consumer Policy*, 17(3):335–348, 1994.

[56] Jonathan L. Herlocker, Joseph A. Konstan, Loren G. Terveen, and John T. Riedl. Evaluating collaborative filtering recommender systems. *ACM Trans. Inf. Syst.*, 22(1):5–53, January 2004.

[57] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 263–272, Washington, DC, USA, 2008. IEEE Computer Society.

[58] Tomoharu Iwata, Shinji Watanabe, and Hiroshi Sawada. Fashion coordinates recommender system using photographs from fashion magazines. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, pages 2262–2267. AAAI Press, 2011.

[59] Michael Jahrer, Andreas Töscher, and Robert Legenstein. Combining predictions for accurate recommender systems. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 693–702. ACM, 2010.

[60] Mohsen Jamali and Martin Ester. Using a trust network to improve top-n recommendation. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 181–188, New York, NY, USA, 2009. ACM.

[61] Gawesh Jawaheer, Martin Szomszor, and Patty Kostkova. Characterisation of explicit feedback in an online music recommendation service. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 317–320. ACM, 2010.

[62] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Trans. Inf. Syst.*, 25(2), April 2007.

[63] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM TRANSACTIONS ON INFORMATION SCIENCE (TOIS*, 25(2), 2007.

[64] Kerry Kao. Suitup! http://www.kerrykao.com/suitup/final/, 2010. Social space centered around clothing items, [Online; accessed 15-03-2014].

[65] George Karypis. Evaluation of item-based top-n recommendation algorithms. In *Proceedings of the Tenth International Conference on Information and Knowledge Management*, CIKM '01, pages 247–254, New York, NY, USA, 2001. ACM.

[66] Bart P. Knijnenburg, Martijn C. Willemsen, and Alfred Kobsa. A pragmatic procedure to support the user-centric evaluation of recommender systems. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 321–324, New York, NY, USA, 2011. ACM.

[67] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, Aug 2009.

[68] P. Kotler. *Marketing Management*. Pearson One. Pearson Prentice Hall, 2009.

[69] LALITA KUMARI. A study: Employee's job satisfaction, its antecedents and linkage between customer satisfaction and employee satisfaction. 2012.

[70] Xuan Nhat Lam, Thuc Vu, Trong Duc Le, and Anh Duc Duong. Addressing cold-start problem in recommendation systems. In *Proceedings of the 2Nd International Conference on Ubiquitous Information Management and Communication*, ICUIMC '08, pages 208–211, New York, NY, USA, 2008. ACM.

[71] Yn Li, Jia Hu, ChengXiang Zhai, and Ye Chen. Improving one-class collaborative filtering by incorporating rich user information. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, CIKM '10, pages 959–968, New York, NY, USA, 2010. ACM.

[72] Jyh-Han Lin and JeffreyScott Vitter. A theory for memory-based learning. *Machine Learning*, 17(2-3):143–167, 1994.

[73] Nathan N. Liu, Xiangrui Meng, Chao Liu, and Qiang Yang. Wisdom of the better few: Cold start recommendation via representative based rating elicitation. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 37–44, New York, NY, USA, 2011. ACM.

[74] Si Liu, Jiashi Feng, Zheng Song, Tianzhu Zhang, Hanqing Lu, Changsheng Xu, and Shuicheng Yan. Hi, magic closet, tell me what to wear! In *Proceedings of the 20th ACM International Conference on Multimedia*, MM '12, pages 619–628, New York, NY, USA, 2012. ACM.

[75] Fang Ma; Huijing Shi; Lihua Chen; Yiping Luo. *A Theory on Fashion Consumption*. Sciedu Press, 2012.

[76] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[77] Paolo Massa and Paolo Avesani. Trust-aware collaborative filtering for recommender systems. In *In Proc. of Federated Int. Conference On The Move to Meaningful Internet: CoopIS, DOA, ODBASE*, pages 492–508, 2004.

[78] Paolo Massa and Paolo Avesani. Trust-aware recommender systems. In *Proceedings of the 2007 ACM Conference on Recommender Systems*, RecSys '07, pages 17–24, New York, NY, USA, 2007. ACM.

[79] Sean M. McNee, John Riedl, and Joseph A. Konstan. Being accurate is not enough: How accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, pages 1097–1101, New York, NY, USA, 2006. ACM.

[80] Frank Meyer. Recommender systems in industrial contexts. *arXiv preprint arXiv:1203.4487*, 2012.

[81] Jakob Nielsen. Conversion rates. http://www.nngroup.com/articles/conversion-rates/, 2013. Conversion Rates webpage, [Online; accessed 12-05-2014].

[82] Rong Pan, Yunhong Zhou, Bin Cao, Nathan N. Liu, Rajan Lukose, Martin Scholz, Qiang Yang, Rong Pan, Yunhong Zhou, Bin Cao, Nathan N. Liu, Rajan Lukose, Martin Scholz, and Qiang Yang. One-class collaborative filtering, 2008.

[83] Weike Pan and Li Chen. Gbpr: Group preference based bayesian personalized ranking for one-class collaborative filtering. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence*, IJCAI'13, pages 2691–2697. AAAI Press, 2013.

[84] Manos Papagelis, Dimitris Plexousakis, and Themistoklis Kutsuras. Alleviating the sparsity problem of collaborative filtering using trust inferences. In Peter Herrmann, Valrie Issarny, and Simon Shiu, editors, *Trust Management*, volume 3477 of *Lecture Notes in Computer Science*, pages 224–239. Springer Berlin Heidelberg, 2005.

[85] Seung-Taek Park, David Pennock, Omid Madani, Nathan Good, and Dennis DeCoste. Naïve filterbots for robust cold-start recommendations. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, pages 699–705, New York, NY, USA, 2006. ACM.

[86] Denis Parra and Xavier Amatriain. Walk the talk. In *User Modeling, Adaption and Personalization*, pages 255–268. Springer, 2011.

[87] David M. W. Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness and correlation. 2007.

[88] Pearl Pu, Li Chen, and Rong Hu. A user-centric evaluation framework for recommender systems. In *Proceedings of the Fifth ACM Conference on Recommender Systems*, RecSys '11, pages 157–164, New York, NY, USA, 2011. ACM.

[89] Al Mamunur Rashid, Istvan Albert, Dan Cosley, Shyong K. Lam, Sean M. McNee, Joseph A. Konstan, and John Riedl. Getting to know you: Learning new user preferences in recommender systems. In *Proceedings of the 7th International Conference on Intelligent User Interfaces*, IUI '02, pages 127–134, New York, NY, USA, 2002. ACM.

[90] Al Mamunur Rashid, George Karypis, and John Riedl. Learning preferences of new users in recommender systems: An information theoretic approach. *SIGKDD Explor. Newsl.*, 10(2):90–100, December 2008.

[91] Claude Sammut and Geoffrey I. Webb. *Encyclopedia of Machine Learning*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[92] J. Ben Schafer, Joseph Konstan, and John Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 158–166, New York, NY, USA, 1999. ACM.

[93] J. Ben Schafer, Joseph Konstan, and John Riedl. Recommender systems in e-commerce. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 158–166, New York, NY, USA, 1999. ACM.

[94] Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM, 2002.

[95] Guy Shani and Asela Gunawardana. Evaluating recommendation systems. In *Recommender systems handbook*, pages 257–297. Springer, 2011.

[96] Edward Shen, Henry Lieberman, and Francis Lam. What am i gonna wear?: Scenario-oriented recommendation. In *Proceedings of the 12th International Conference on Intelligent User Interfaces*, IUI '07, pages 365–368, New York, NY, USA, 2007. ACM.

[97] Vikas Sindhwani, Serhat S. Bucak, Jianying Hu, and Aleksandra Mojsilovic. One-class matrix completion with low-density factorizations. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 1055–1060, Washington, DC, USA, 2010. IEEE Computer Society.

[98] Push Singh, Thomas Lin, Erik T. Mueller, Grace Lim, Travell Perkins, and Wan Li Zhu. Open mind common sense: Knowledge acquisition from the general public. pages 1223–1237. Springer-Verlag, 2002.

[99] David H. Stern, Ralf Herbrich, and Thore Graepel. Matchbox: large scale online bayesian recommendations. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 111–120, New York, NY, USA, 2009. ACM.

[100] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.

[101] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.

[102] Nava Tintarev and Judith Masthoff. A survey of explanations in recommender systems. In *Data Engineering Workshop, 2007 IEEE 23rd International Conference on*, pages 801–810. IEEE, 2007.

[103] Andreas Töscher, Michael Jahrer, and Robert M Bell. The bigchaos solution to the netflix grand prize. *Netflix prize documentation*, 2009.

[104] Patricia Victor, Chris Cornelis, Martine De Cock, and Ankur M. Teredesai. Key figure impact in trust-enhanced recommender systems. *AI Commun.*, 21(2-3):127–143, April 2008.

[105] G. Vignali and C. Vignali. *Fashion marketing & theory*. Access Press, 2009.

[106] Emine Yilmaz, Javed A. Aslam, and Stephen Robertson. A new rank correlation coefficient for information retrieval. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '08, pages 587–594, New York, NY, USA, 2008. ACM.

[107] Kenji Araki Ying Zhao. What to wear in different situations? a content-based recommendation system for fashion coordination. 2011.

[108] Lin Yu-Chu, Yuusuke Kawakita, Etsuko Suzuki, and Haruhisa Ichikawa. Personalized clothing-recommendation system based on a modified bayesian network. In *Proceedings of the 2012 IEEE/IPSJ 12th International Symposium on Applications and the Internet*, SAINT '12, pages 414–417, Washington, DC, USA, 2012. IEEE Computer Society.

[109] Zhiyong Zhang and Olfa Nasraoui. Mining search engine query logs for query recommendation. In *Proceedings of the 15th International Conference on World Wide Web*, WWW '06, pages 1039–1040, New York, NY, USA, 2006. ACM.

[110] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *Proc. 4th Intl Conf. Algorithmic Aspects in Information and Management, LNCS 5034*, pages 337–348. Springer, 2008.