KTH Computer Science
and Communication

Extending recommendation algorithms by modeling user context

THEODOROS VASILOUDIS

Master's Thesis at Spotify and CSC
KTH Supervisor: Hedvig Kjellström
Company Supervisor: Boxun Zhang
KTH Examiner: Danica Kragic

TRITA xxx yyyy-nn

Abstract

Recommender systems have been widely adopted by online e-commerce websites like Amazon and music streaming services like Spotify. However, most research efforts have not sufficiently considered the context in which recommendations are made, especially when the input is implicit.

In this work, we investigate the value of including contextual information like day-of-week in collaborative filtering recommender systems. For the investigation, we first implemented two algorithms, namely contextual pre-filtering and contextual post-filtering. Then, we evaluated these algorithms with user data collected from Spotify.

Experiment results show that the pre-filtering algorithm shows some promise against an item similarity baseline, indicating that further investigation could be rewarding. The post-filtering algorithm underperforms a popularity-derived baseline, due to information loss in the recommendation process.

Referat

Förbättrade rekommenadtionsalgoritmer genom att använda användarens kontext.

Rekommendationssystem har spridda användsningområden så som e-handels företag som Amazon och internetbaserade musiktjänster som Spotify. Mesta forskningen inom rekommendationssystem har inte tagit användares context i beaktning och speciellt inte då datan är av implicit typ.

I det här projektet har vi undersökt vikten av att inkludera information om användares context, så som veckodag, i traditionella rekommendationssystem baserat på collaborative filtering. Vi har implementerat två algoritmer, contextual pre-filtering och contextual post-filtering och utvärderat dem på användardata från Spotify.

Experimenten visar att post-filtering algoritmen presterar sämre än en standard popularitetsbaserad algoritm på grund utav informationsförlust i rekommendationssteget. Algoritmen baserad på pre-filtering visar lovande resultat jämfört med en standard item similarity algoritm vilket lovordar vidare undersökningar.

Aknowledgements

Any major effort will usually have many people contributing, either directly on indirectly. The same is true for this thesis, so I'd like to take a moment to thank the people who contributed towards the completion of this thesis. I'd like to start with my supervisor at KTH, Hedvig Kjellström, who enabled me to start this thesis at Spotify in the first place, and provided me with guidance throughout my work.

The opportunity to perform my thesis at Spotify was made possible by Mikael Goldmann, Henrik Lindström and Anders Arpteg who trusted me and guided me in the beginning of my thesis. Christopher Johnson put me in the right track for this thesis and was always available to answer my questions.

I want to make a special mention of Boxun Zhang, my supervisor at Spotify. His guidance, advice and academic rigor helped elevate the quality of this thesis and provided me with valuable lessons in scientific thinking and writing.

I would like to thank my colleague Oscar Carlsson for all the discussions during the performance of our theses at Spotify. Finally I would like to thank the rest of my squad at Spotify; Anders Eurenius, Gösta Forsun, Mathew Green, Marcus Issakson, Henrik Löf and Aleksandar Radulovic for all the advice and interesting discussions during my time there.

Contents

Chapter 1

Introduction

In this chapter we first provide an overview of the challenges of recommender systems and describe the goals of this thesis. Then, we briefly present the methodology followed in this thesis. We end the chapter by providing an outline of the thesis.

1.1 Motivation

Providing relevant recommendations to users is a challenge faced by many online services these days. The users are presented with an abundance of choices, like different products in an online store like Amazon[1], or songs in services like Spotify[2] and Pandora[3].

The goal of a recommender system is to help users to make choices, by recommending items that are relevant to their interests and current context. The approach followed by most systems is to use some form of collaborative filtering or content based recommendation system or, quite often, use a hybrid approach combining the two.

The problem with these approaches is that they do not take context into account. A definition of context is given by Abowd et al. [2]:

> *Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

The importance of including context in a personalization system is shown by Palmisano et al. [47] and Gorgolione et al. [24], where it is made clear that including context in a recommendation system can have a positive effect on the performance as it helps in modeling users in more detail and achieving a better understanding of their behavior.

[1] `http://www.amazon.com`
[2] `http://www.spotify.com`
[3] `http://www.pandora.com`

In the domain of music recommendation an "entity" can be a user or a song. Schedl et al. [58] also make a distinction between user and music context, which correspond to these two entities. User context can include a user's mood, social context or his location, factors that are "dynamic and frequently changing". Music context can include semantic labels, information on the release date of the track, and its geographic origin. In general they are factors relevant to the song that cannot be extracted from the audio signal.

These are important factors that can influence the way a user selects music, and hence they should be taken into consideration when designing a recommendation system.

1.2 Goals

One goal of this project is to extend a baseline collaborative filtering recommendation system in order to include user context in the decision process. Our goal is to improve the accuracy of the recommendations made by the system, evaluating the performance of our approach using some of the metrics proposed by Herlocker et al. in [29].

Another important goal is for the system to be an *extension* of existing approaches, thereby making it possible to be used together with already established methods in the field. This ensures that all the research that has already been made in the field, as well as investments in building recommender systems by companies can be used in conjunction with the algorithms developed.

Finally the systems designed should be scalable and be able to provide recommendations in cases where we have millions of users and items. For this reason we avoided overly complicated techniques and focused on methods with clear scalability potential.

1.3 Methodology

Our approach includes using learning techniques which detect behavioral patterns that might be present in user logs. Those may indicate users acting in specific manners under specific contexts. We follow two main methodologies in order to achieve this goal. One is using one traditional recommender system to provide recommendations which are then "contextualized" according to user behavior within the context. The other approach creates different traditional recommender systems which are trained on data that correspond to a contextual slice of the complete dataset. The appropriate recommender is used according to the context in which we are making the recommendation in.

For our experiments we used time as the contextual variable, as it is a contextual variable that can be obtained easily without the need for any inference, and it provides an intuitive way to separate the data on the assumption that user behavior will differ in different time-frames. The experiments were performed on a number

of different datasets, covering different periods within a day and within a year as well as different platforms. This allowed us to test the developed algorithms under many different settings and provide a better evaluation of the algorithms.

1.4 Thesis outline

In the introduction we provided the motivation for using contextual information in a recommender system. In Chapter 2 we will provide a thorough overview of the field of recommender systems and current research in the field. In Chapter 3 we will present a number of approaches that have been presented in context-aware recommender systems, providing a categorization of the various algorithms and examining the advantages and limitations of the current state-of-the-art. In Chapter 4 we will present the algorithms developed for this thesis, and a presentation of the experiments performed will be made in Chapter 5, along with an evaluation of the performance of the algorithms. We will close the thesis with Chapter 6 with our final thoughts on the developed methods and a look at future work.

Chapter 2

Background

In this section we present a comprehensive overview of past and current research on recommender systems. We will start by describing Collaborative Filtering (CF) in Section 2.1, continue with content-based algorithms in Section 2.2 and provide a short introduction to context-aware systems in Section 2.3. We will examine hybrid recommender systems in Section 2.4 and end the chapter with an overview of the evaluation of recommender systems in Section 2.5.

2.1 Collaborative filtering

Collaborative filtering is arguably the most widely used recommender system technique. CF algorithms are commonly categorized as memory-based and model-based [60]. Memory-based CF algorithms make use of the similarity between users to make recommendations [54] and are therefore also known as neighborhood-based algorithms. Model-based algorithms build statistical models out of users' ratings and their interactions with the system in order to make predictions such as what ratings the users would give to unknown items [40], [31], [67].

Memory-based approaches were some of the early algorithms developed for recommendations. These approaches usually examined the complete user-item matrix in order to discover similarities between users. The similarity between every user in the dataset was calculated using a vector-space model, as is common in the Information Retrieval field. Each user was represented by a vector containing his ratings for the items in the dataset, and measures such as the Pearson correlation coefficient (2.1) or cosine similarity (2.2) were used in order to compute the similarity between users. In Equation 2.1 and Equation 2.2 $\vec{x}, \vec{y}$ are the user vectors we are examining, $r_{x,i}$ is the rating of user x for item i, $\overline{r_x}$ is the average rating that user x has for items he has rated, and I_{xy} is the set of items that have been rated by both user x and user y.

$$pearson(\vec{x}, \vec{y}) = \frac{\sum_{i \in I_{xy}} (r_{x,i} - \overline{r_x})(r_{y,i} - \overline{r_y})}{\sqrt{\sum_{i \in I_{xy}} (r_{x,i} - \overline{r_x})^2 \sum_{i \in I_{xy}} (r_{y,i} - \overline{r_y})^2}} \tag{2.1}$$

$$cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \|\vec{y}\|} = \frac{\sum_{i \in I_{xy}} r_{x,i} r_{y,i}}{\sqrt{\sum_{i \in I_x} r_{x,i}^2} \sqrt{\sum_{i \in I_y} r_{y,i}^2}} \tag{2.2}$$

The rating prediction for a user-item pair was then made by selecting the nearest neighbors for the target user, and aggregating their ratings for the target item, typically using a weighted average function. In order to recommend the N best items for the target user, the items for which his nearest neighbors showed the most preference were selected. Improvements that were made to the base algorithm included normalizing user rows in order to counteract the influence of very active users who have interacted with many items, or including a normalizing factor in the rating prediction that modeled the average rating given by the target user.

While user-based CF systems were popular, they were not scalable to millions of users and items which is the size of datasets that companies like Amazon and Spotify are dealing with. The complexity of these neighborhood-based algorithms grows linearly with the number of users, making them unsuitable for large-scale applications. These systems also suffered from data sparsity, were pairs of users who have rated only a few items with the same ratings would be labeled as very similar were such a similarity would be unwarranted for so few data points.

In order to counter-act this problem an item-based CF algorithm was proposed by Sarwar et al. [56]. Linden et al. [41] described the *item-to-item collaborative filtering* algorithm used in the online retailer Amazon, which was based on ideas from Sarwar's work. The main idea behind this algorithm is to first match a user's purchased and rated items to similar items. Depending on the application items could be products in an online retailer or music tracks in a music streaming service. Then, it provides the user with a recommendation list based on those similar items. In order to discover similar items, an item similarity table is built by finding items that users tend to purchase together.

However, iterating through each item pair to find their similarity would be computationally inefficient because many of the product pairs will not have common purchasers. Instead, an iterative approach that calculates the "similarity between a single product and all related products" is used. The similarity between items is calculated by comparing the vectors that represent them. This is contrary to traditional CF systems which represent each user with an n dimensional vector, where n is the number of items in the system, and try to find similar users using some vector similarity measure, such as cosine similarity. In *item-to-item collaborative filtering* the length of the vectors being compared depends on the number of users that have interacted with an item, which in an online service the size of Amazon is bound to be much smaller than the total number of items available in the system.

This characteristic of the algorithm provides it with an advantage over traditional collaborative filtering techniques in performance and scalability. The expensive item-to-item similarity calculation is performed offline with a algorithmic complexity of $O(nm)$ where m is the number of users and n the number of items a user has interacted with. Once the similarity table is created, recommendations are made by finding items similar to a users' rated and purchased items and presenting them as a list to that user. Depending on the number of items a user has rated and purchased, the creation of this list is a computationally inexpensive procedure.

Other model-based techniques try to model the relationships present in the data of user ratings and interactions with the system, like purchases in an online retailer or streams in a music streaming service. Lemire et al. [40] utilized the differences in ratings between items in order to make predictions. They take pairs of items and try to determine how much better one item is liked than the other. This can be determined, for example, by subtracting the average rating of two items. This difference is used to predict the rating of one item for a user, given the rating he has given to the other.

Clustering techniques are also commonly used for making recommendations. Ungar et al. [67] clustered users and items using different k-means algorithms and Gibbs sampling [19]. They used the items rated by users to cluster the users and the users that rated the items to cluster the items.

Hofmann [31] adapted the probabilistic Latent Semantic Analysis (pLSA) technique [30] to tackle collaborative filtering problems. Using pLSA in such a context provides for higher accuracy and the capability to automatically identify user communities and item categories.

One major push in the development of recommendation algorithms was accomplished with the organization of the Netflix Prize [12]. This was a competition organized by the video streaming service Netflix[1] that provided a large dataset to researchers containing 100 million ratings with timestamps from 480 thousand subscribers on close to 18 thousand movies. The challenge was to improve the performance of its existing recommendation system, according to the Root Mean Square Error metric, which measures the algorithms ability to predict the rating that a user gave to a movie. The team that achieved the biggest reduction in the error was awarded $1,000,000. The best performing algorithms in that contest [36], [51], [62] were algorithms that employed techniques based on latent factor models.

Latent factor models learn feature vectors for users and items from the data, modeling users and items using a number of factors. The rating prediction for a user-item pair for these systems is then performed by taking the inner product of the user factor vector and the item factor vector. One technique developed by Bell and Koren [11] was based on alternating least squares (ALS). Using matrix factorization techniques the user-item matrix is factorized into two matrices, one containing the user factor vectors and one containing the item factor vectors. The number of factors f we choose determines the dimensions of the two matrices, which

[1] `http://www.netflix.com`

will have $m \times f$ and $f \times n$ dimensions for the user factor and item factor matrix respectively, where m is the number of users and n the number of items. The product Q of these two matrices should approximate the original user-item matrix. The goal is then to minimize the square of the error between the original matrix and the product matrix Q by optimizing the values in the user factor and item factor matrices. Since optimizing both matrices at the same time is a non-convex optimization problem, we choose to maintain one of the matrices fixed, and solve the system of linear equations for the other matrix, which is a tractable, convex optimization problem. In ALS we alternate between having the user and item matrix fixed. One of the major advantages of ALS is that the factorization can be performed in a distributed manner, and the performance scales linearly as more machines are added [32]. Using such techniques, computation of recommendations for huge datasets containing millions of users and items becomes tractable.

Collaborative filtering however has some inherent problems like dealing with the sparsity of the data. As users will tend to give ratings or interact with only a small number of items from the millions that might be available, the resulting user-item matrices are extremely sparse. This sparsity can give rise to the cold start problem and cause performance issues. The cold start problem refers to the inability of collaborative filtering systems to make accurate recommendations to users who have not rated enough items, as well as recommending items before a number of users have rated them. In order to mitigate the effect of data sparsity on performance, matrix factorization techniques like ALS and Singular Value Decomposition (SVD) have been employed [57] while techniques like pLSA can also improve performance.

Collaborative filtering systems also suffer from popularity bias, which is the phenomenon of more popular items being recommended more often due to their popularity. This is amplified by the *long tail* problem [6], where a small number of items makes up the majority of the user preferences and a large number of items forms the *long tail*, items that are interacted with by a small number of users. This creates a positive feedback loop and gives rise to a rich-get-richer phenomenon [21] in the system. The concept is described in more detail in Section 2.5.

Considerations for implicit data

Since we will be using implicit user data for our experiments it is important to note some considerations that need to be made when working with implicit versus explicit data.

By implicit data we mean that we measure user preference for items using interaction data, such as number of purchases, number of streams, or clickstream data. The main assumption we make is that a user who interact many times with an item, for example a user that streams the same track many times, probably likes that item. Hu et al. examine the problem of making recommendations for implicit feedback datasets in [32].

One of the main issues that the authors raise is the lack of negative feedback in implicit data. The fact that a user has not interacted with an item could indicate

that the user dislikes the item, but it could also indicate that the user is simply not aware of the item. This is an issue that does not arise in explicit rating data where the ratings indicate exactly what the users do and do not like. In explicit recommenders the data for which we do not have ratings on are treated as missing data and are not taken into consideration for the modeling. Attempting to do this with implicit data however would mean that we only take positive feedback into account when building the user profile, which can lead to poor modeling of the real user preferences.

The authors also note the problem of inherent noise in implicit feedback data. One example in the domain of music can be a user who lets the computer play tracks and then leaves the room. The user will have interacted with the items according to the implicit data, but we cannot be sure whether he would actually enjoy the tracks that were played during that time.

Another issue is that while "the numerical value of explicit feedback indicates *preference*, the numerical value of implicit feedback indicates *confidence*". What this means is that having more interactions with an item does not necessarily mean that a user likes that item more than another item with which he has less interactions with. An example in the music domain would be a user that has some favorite tracks that he no longer listens to often due to satiation with them, preferring to listen to more popular recent tracks more often. His absolute preferences might be with the older tracks, but his interactions indicate the more recent tracks. With this taken into consideration we can claim that the number of interactions with an item can provide us with an indication that a user likes an item but not an absolute preference measure.

2.2 Content-based recommender systems

In content-based recommender systems we look at the actual features of the items in order to find similarities or extract semantic information, which we then use to make recommendations to users, typically by recommending items that are similar to items that the user has already shown interest in.

In the domain of music, a lot of the features and techniques used in content-based recommender systems are based on techniques developed for Music Information Retrieval. Such techniques can be used in order to extract semantic information directly from the audio signal, by using descriptors such as Mel Frequency Cepstral Coefficients (MFCC) [42]. The information extracted can be low-level descriptors such as timbre or tempo, or even higher level semantic information such as the mood of the song [44], [63], or semantic labels such as genre, although the success for this particular task has been limited due to the difficulty of extracting semantic information from the audio signal [13]. Another common use-case is to calculate the similarity between audio tracks and use the similarity in order to provide recommendations to users. Audio similarity was explored by Tzanetakis in his PhD dissertation [65], where he explored the extraction of high level features from songs,

including rhythm and harmony.

Van den Oord et al. [68] presented an approach based on convolutional neural networks where the content of an audio track is used in order to extract latent semantic information and use it to provide recommendations. The approach presented is able to out-perform traditional bag-of-word models when tested on the Million Song Dataset [14] by a large margin. However the performance of the algorithm is still worse than using a collaborative filtering system. According to the authors that outcome is expected since many aspects of the songs that can influence the preferences of the users cannot be extracted from the audio signal alone. The authors note the inability to predict the popularity of a song as a major limiting factor in the approach. Trohidis et al. [63] made use of multi-label classification in order to assign moods to songs. The authors used MARSYAS, a feature extraction framework developed by Tzanetakis et al. [66] to extract rhythmic features like Beats per Minute and timbre features such as the aforementioned MFCC features. The recognition of emotions was performed using multi-label classifiers [64] on a set of songs labeled with emotions in the Tellegen-Watson-Clark model [61].

Some of the limitations and advantages of content-based recommendation systems are listed in [21]. The drawbacks include the lack of novelty in recommendations which can be an undesired effect of having a well performing similarity function. If we only recommend similar sounding tracks to the users, our recommendations will end up lacking in novelty and serendipity, two concepts we will explain further in Section 2.5. These systems also do not take user preferences and listening habits into account, which can to a large extent influence whether a user will actually enjoy a song or not.

Content-based approaches however mitigate a number of the problems that CF systems face. That includes the inability to recommend a new item (cold start problem), as we don't need to wait for users to make ratings in order to be able to recommend an item and essentially removes the problem of popularity bias as user ratings are not included in the recommendation process.

2.3 Context-aware recommender systems

Context-aware recommender systems (CARS) will be the main focus of this project and we will provide a more extensive overview for them in Chapter 3 so we will only mention a number of approaches in this section. The main idea behind context-aware recommender systems is to include contextual information about the user or the item in the recommendation process, thereby making the recommendation more relevant to the current context.

A number of researchers have used time as context [8], [23], [20] [18] as time data can be readily available in datasets and can be incorporated directly without the need for inference. One approach followed in [8] is to use existing collaborative filtering techniques but only on slices of data corresponding to the current context. Tensor factorization can also be used in order to model the time context, as is done

in [20] or any other context type as the authors of [33] did.

Other approaches include trying to extract user context from diverse sources of information [10], [49], [38]. Mobile sensors can be used to extract information such as temperature and location which can then be used to infer the user context. The aforementioned approaches will be discussed in more detail in Chapter 3.

2.4 Hybrid systems

Hybrid systems are systems that use a combination of the aforementioned techniques in order to improve the overall quality of the recommendation system.

There are many different ways to combine different techniques [60]. One could be to have some kind of combination of the results of two or more approaches in order to have a final score for the recommendation. In [46] the authors combine a content-based and a collaborative filtering method, using each method as a means to overcome the limitations of the other. More specifically the content-based method is used to overcome the cold-start problem of the CF method and the CF method is used to improve the quality of recommendations.

Another way would be to use them in sequence, for example creating a list of recommendations using a collaborative filtering system and then re-ranking that list using a context-aware recommender. This is an approach used in [28], a context-aware system which we will describe in Chapter 3.

Hybrid systems, when leveraged correctly, can often have better performance than using individual systems [17]. For that reason they are often employed in commercial applications, also in the forms of ensembles, where several CF and content algorithms may be used in conjuction in order to achieve the highest possible performance. In many cases researchers use different techniques, or different algorithms within the same discipline in order to achieve better recommendations. In [71] the authors combine model-based and memory-based techniques to achieve better performance than the individual algorithms.

2.5 Evaluation of recommender systems

There are a number of challenges researchers face when trying to evaluate recommender systems. Depending on the task we are trying to tackle, different evaluation metrics may give different results.

In [29] Herlocker et al. made a comprehensive study of the challenge that evaluating recommender systems poses. They differentiate among *domain features* of datasets. Those may include the user tasks that are supported by the recommender, such as *Find Good Items* or *Recommend Sequence*, or the need for novelty in the recommendation and how important we deem their quality. Other properties may pertain to inherent features of the data, such as whether the ratings are explicit or implicit and the presence of a timestamp. Finally they note the importance of

sample features, as in any other case where we have to evaluate a dataset. These can include the size and distribution of the data set and its sparseness.

An important concept relating to the distribution of the dataset and how users consume is the *Long Tail* concept [6], described in [72] and examined in depth in [21] specifically in the domain of music. The theory behind this idea is that a relatively small number of popular items dominate the user preferences, lying in the head of the distribution, and a very large number of items lies in the *long tail*, niche items that are not popular on their own but as a whole account for a considerable percentage of items consumed. In the music industry, Celma [21] cites numbers from the 2007 Nielsen "State of the industry report":

> *844 million digital tracks were sold in 2007, but only 1% of all digital tracks—the head part of the curve—accounted for 80% of all track sales. Also, 1,000 albums accounted for 50% of all album sales, and 450,344 of the 570,000 albums sold were purchased less than 100 times.*

The distribution of the consumption of items can follow a power-law distribution, although not necessarily, and the specifics of the distribution can affect the performance of the recommendation techniques used, as shown in [72]. It is therefore very important to consider the distribution of the dataset when deciding on the recommender system, and also when evaluating new techniques.

Some more recent work focused on accuracy measures is presented in [26]. In this work the authors provide guidelines in selecting appropriate similarity measures according to the user task. They also focus on the importance of having a statistically sound manner by which we verify the performance of the algorithms, by providing tests for statistical significance when comparing the performance of algorithms and ranking them.

Some of the metrics that are often used in evaluating the accuracy of recommender systems metrics are:

- Precision-Recall. These have a similar definition as the one given to them in the Information Retrieval field. Precision is a way of evaluating the algorithm's ability to provide recommendations that are relevant to the user, versus making irrelevant recommendations. Recall in the recommendation context is a measure of how well the recommendations we make cover the range of the users' taste, i.e. from all the items that the users interacted with in the test set, how many we are able to recommend.

- Receiver operating characteristic or ROC curves. These compare the true positive rate to the false positive rate. A variant which is precision-recall curves was used in the evaluation of the algorithms developed from this thesis.

- Utility measures. These try to measure the utility of a ranked recommendation list to users, according to the usefulness/relevance of the recommended items in respect to their position in the list.

- Hit Ratio [34], a recall-like measure for recommendation lists which measures the ratio of held-out items returned by the algorithm in the list, and can be evaluated at different list lengths.

In Section 5.2 we will provide a more detailed look at the metrics that we used in this thesis.

In [29] as well as [45] the importance of looking beyond just accuracy when evaluating recommender systems is noted. Recommender systems must be useful to users and in order to do that we should try to cover a number of aspects. The system should provide sufficient *coverage* over the available items and its *learning rate* should be such that new users are able to receive acceptable recommendations after only a few interactions with the system. Another important topic is the *novelty* and *serendipity* of the recommendations. Novelty refers to providing the user with items that he might enjoy and has not encountered before. Serendipity is a bit more involved than that, as it tries to measure the ability of the algorithm to provide the users with unseen items that are also outside the users' usual preferences. An example provided in [29] that can aid in discerning between the two is that recommending a movie from a user's favorite director that the user has not seen yet would be a novel recommendation, while a serendipitous recommendation would be one where we recommend a movie from a genre totally unrelated to what the user usually watches but the user ends up enjoying. The main problem with metrics such as serendipity is that they are usually difficult to measure as they are open to interpretation and depend on each individual user.

Another dimension we may want to examine when evaluating a recommender system is its robustness to attacks that attempt to influence its results. Online recommender systems can be the target of attacks by people looking to benefit from "gaming" the system in order to promote their items and gain, usually financially, from the increased exposure their items will receive. This problem was studied in [37] where the authors examine *shill attacks*, where the attacker creates fake user accounts and uses those to provide favorable ratings for the items he wants to promote. The susceptibility of different algorithms to attacks is examined as well as how easy or hard it is to detect such attacks. The authors provide some guidelines for recommender systems designers in order to better protect their systems against attacks such as the importance of protecting new items which can be more sensitive to attacks.

Before continuing we should make note of one of the major problems with the offline evaluation of recommender systems. The ground truth in offline testing is the interaction history of the users, and the goal that we try to achieve is to "recommend back" items that the users interacted with in their history. What we define in our measures as "irrelevant" user-item pairs are pairs that were not present in the ground truth. However, the users might have found the recommended items to be a good recommendation, relevant to their context. If the user never interacts with an item it will never appear in the test set, meaning that any serendipitous recommendation made by the algorithm will actually be counted as a failed recommendation when

it could be a better recommendation when compared to many items in the test set. In order to mitigate such problems one should run live tests of the recommender systems and gather feedback from actual user interactions with the system.

Chapter 3

Previous Work

In this chapter we will provide an overview of other approaches to context-aware recommender systems.

In Section 1.1 we provided a definition of context and indicated its importance in improving recommendations using Context-aware recommender systems (CARS). In the field of music, Schedl et al. [58] provide a separation between *user context* which can include the user's social context, weather conditions, time of day and other factors and *music context* which includes factors that "cannot be extracted directly from the audio, but are nevertheless related to the music item". These can include information about the artist and metadata such when or where the track was recorded. That information can be in the form of semantic labels, such as those crowd-sourced by the users of the service last.fm[1].

As far as user context is concerned there are many ways to attack the problem, as there exist a multitude of factors that we can include in what we define as user context. These can include the mood of the user, the time of day or day of year, the weather conditions, the location of the user, and the social context i.e. whether the user is alone or with a group. In general we could include any factor that fits the definition for context given in Section 1.1, if we make the assumption that it will improve the performance of our system.

The main challenge with user context is that most of the information about it has to be derived from sensors or other implicit data about user behavior. In some studies the users were asked to provide their context explicitly [9], [33] but as examined by Pu et al. [52], enforced preference elicitation can be detrimental to the user experience, so the authors of [52] recommend minimizing the need for user input. As we will see in the following sections, many of the recommender systems choose instead to use contextual information that is more readily available such as time and date and weather information.

A categorization for context-aware recommender systems is provided in [5], which classifies the techniques based on which part of the recommendation pipeline context is used in. The categorization provided is the following:

[1] `http://www.last.fm`

- Contextual pre-filtering: Figure 3.1(a). In this category the context is used as a means to pre-filter the data, so that we split the initial dataset that includes the contextual information into a number of different datasets depending on the values of the contextual variables, that are in turn used to train different traditional recommendation systems. Recommendations are then made by using the appropriate recommender system based on the target context.

- Contextual post-filtering: Figure 3.1(b). In this category the context is used as a means to adjust the output of a traditional recommender system, by incorporating context so that the recommendations better match the target context.

- Contextual modeling: Figure 3.1(c). This category of systems integrate context directly in the recommendation procedure. These systems are multidimensional recommender systems, and can be extensions of existing 2D ($User \times Item$) techniques that are able to handle multiple dimensions.

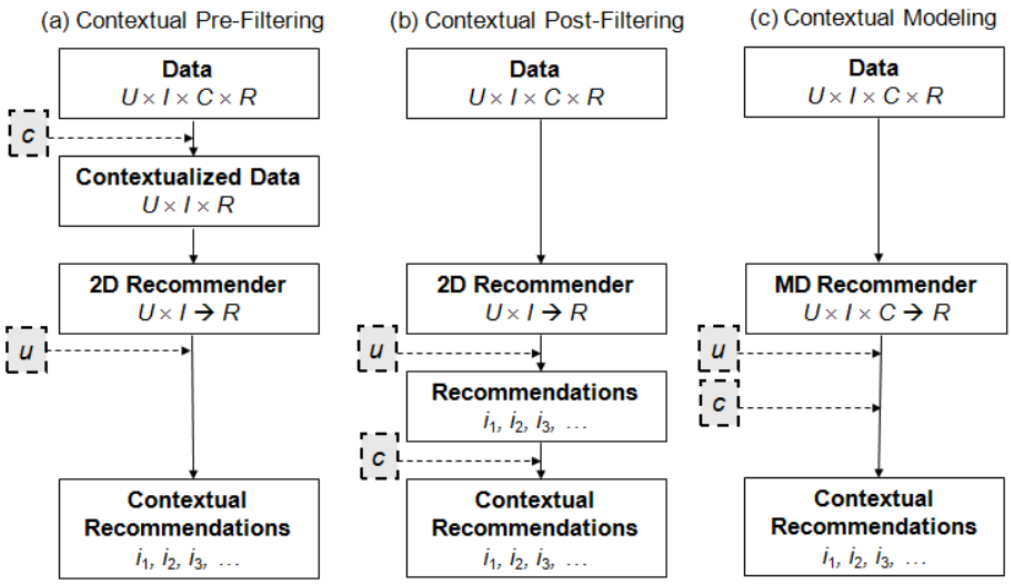


Figure 3.1. CARS categorization. Source: [5]

The pre-filtering and post-filtering approaches have the advantage of working in conjunction with traditional recommender systems, thereby allowing us to make use of existing technologies and easing the transition from a 2D to a multidimensional approach. This is especially true for companies who have already invested heavily in creating a high-performing 2D based recommendation system, as replacing it completely would not make technological or financial sense. Contextual modeling can

have the potential advantage of improved performance from including the context as a central feature of the recommendation system.

As already mentioned in Section 2.4 multiple approaches can be combined in an ensemble in order to achieve increased performance. The same is true also in the case of context-aware recommendation systems where an ensemble of techniques from one or more of the approaches described previously can be used together. Adomavicius et al. follow this approach in [3] where they combine different contextual pre-filters in order to approach the current context using a number of potentially generalized contexts and then combining the ratings generated from each filter in order to produce the final recommendation score. The way this is achieved is by determining which of the pre-filters performs better than the 2D approach used as a baseline and choosing to use the appropriate best-performing pre-filter for the target context.

In the following sections we present a number of research efforts in the field, classified using Adomavicius' categorization into contextual pre-filtering, post-filtering and contextual modeling algorithms. Where applicable we will also mention whether the algorithm focuses on user context or music context data, as defined by Schedl.

3.1 Contextual pre-filtering

As we mentioned in the introduction to this chapter, time is one of the contextual variables that are easier to obtain. In [8] Baltrunas et al. use time as a way to pre-filter the interaction data and split each user profile into micro-profiles, each representing a user in a specific context, defined as different non-overlapping time segments. The predictions are then made using the micro-profiles instead of the complete user profiles. The main idea behind this approach is that by limiting the training set for each mirco-profile into context that is relevant to the specific situation, the system will be able to model the temporal differences in user taste more accurately. However the problem of splitting the user profiles in separate time slices in a way that improves the prediction is a major challenge. That is made evident by the authors in their efforts to find the optimal split for the profiles, as well as in the experimental results. The authors were able to achieve better quantitative performance by using a split of user profiles into even and odd hours, a split that should not contain any semantic information, as we don't expect users to have different music taste in even and odd hours.

An approach that focuses specifically on deriving context from user event logs is provided in [38]. The authors once again use a variety of input sources to get context data which they then abstract into concepts such as "summer" or "weekend", also allowing for fuzziness for contexts that don't have clear boundaries between them, for example the difference between spring and summer, in contrast to the approach in [8] we just mentioned. The authors then use different techniques in order to incorporate the context into their predictions, for example by using a simple pre-filtering approach, or determining the popularity of an item in a specific context. Unfortunately the report is lacking in implementation details and the experiments

performed focused on parameters like number of similar users and varying the percentage of the dataset used for training the algorithms, which do not reveal so much about the actual performance of the algorithms against other techniques.

Using case-based reasoning [1] as a way to incorporate context in the recommendation procedure was explored in [39]. The authors used temporal data such as season and weekday, location and weather data as the contextual information. Users were defined by using profile information such as gender, age and their listening habits. The context of the users was logged together with their listening habits so that the authors had access to which tracks users listened to under specific contexts. The recommendation procedure in order to recommend tracks to a user under a specific context was to find similar users who listened to music in a similar context and use their listening history in order to make recommendations to the target user.

3.2 Contextual post-filtering

In contextual post-filtering traditional 2D recommenders are used as input, and trained on non-contextual data. The resulting recommendation lists are then adjusted by the post-filtering algorithm in order to better fit the target context.

This an approach is taken by Hariri et al. in [27] where the output of a 2D recommender is re-ranked according to contextual information as evidenced by the sequence of tracks played by the user. There the authors use the semantic tags provided by last.fm users in order to perform topic modeling [15] on the sequence of songs using Latend Dirichlect Allocation [16] (LDA) and then mine human-created playlists in order to discover frequent sequential patterns among the discovered topics. After that is done, the current sequence of topics selected by the user is matched against the mined patterns and those are used in order to predict the following topic and from that recommend a song.

An important aspect of this algorithm is the exploitation of the user's current playing information, with the goal of making the recommendation more relevant to the current state of the user. The fact that a user's preferences can change during an active session can cause problems however. A short session does not provide enough predictive power while longer sessions make it more likely that the user's preferences have drifted. The authors use an *all-K'th-order* method in order to mitigate this effect. If the algorithm cannot make any recommendations for a given size of a session, its size is iteratively reduced until a recommendation is generated.

This algorithm makes the underlying assumption that a user's context is modeled in the way that he selects music order, and that playlists also model a certain type of listening context. If the number of topics generated is very large though, a large data set is needed in order to be able to make recommendations on the highly varying sequences that a user may select to listen to. The fact that the dataset needed is very large though can also lead spurious relations appearing, especially for shorter user sessions.

Some more work that looks at playlists and ordering as defining music context is performed at [22]. Here the author makes use of content based techniques and artist graph information based on data from the social network Myspace[2] in order to generate playlists. A technique to calculate playlist similarity is also proposed which also uses topics generated using LDA from social tags in order to represent songs.

3.3 Contextual modeling

In contextual modeling, 2D recommenders are not used in the recommendation pipeline. Instead the complete multidimensional data, including users, items and the contextual dimensions are used to train the recommendation systems.

In [28] the authors develop an extension to the LDA algorithm that allows them to include the semantic tags as features of the items, when modeling the way that a user selects songs. Each user is modeled as a multinomial distribution over the discovered topics and each topic has a distribution over the set of items and features. Items are assumed to be generated for a user by sampling a topic from the user distribution and then, according to the selected topic, sampling the item and its features from the corresponding distributions. In order to use this model for recommendations, the probability $p(i|u, c)$ for a user u to select item i in the context c is calculated based on estimates of the user topic, the item topic and item feature distributions. The hyperparameters for these distributions are approximated using variational message passing, a Bayesian inference method proposed by Winn et al. [69]. One advantage of this technique when using semantic tags as features, is that it provides a grouping of these tags as a result of the topics discovered, so we have an overview of which tags the algorithms clusters together, allowing for a qualitative evaluation of the algorithm's performance and a visualization of the users preferences as well as the ability to calculate the similarity between artists and songs.

In [23] the authors present an entry to the Challenge on Context-aware Movie Recommendation [55] that uses time as context. The task was to predict the movies that users would rate in a set of weeks, the Christmas week and the week leading up to the Academy Awards in 2010. The authors used Pairwise Interaction Tensor Factorization (PITF) [53], a tensor factorization model developed in order to predict tags, to model the time context. Despite including context into the calculations though, the method proposed by the authors performs worse than other methods not using time or other contextual information. The authors attribute this to the lack of time in optimizing the model, but it should also serve as an indicator that including contextual information into a recommendation system can lead to worse performance. It could be the case that the contextual information actually holds no value that can improve the recommendation, or its value can be diminished during the processing steps, due to information loss.

[2] `http://www.myspace.com`

Apart from time, other information about the current user situation can also be leveraged, especially from mobile devices like smartphones which contain a number of sensors from which information about the context of the user can be derived. In [10] the authors propose a way to combine social and mobile data as well as sensor networks in order to enable a multitude of context-aware applications, which they name SocialFusion. This work looks at the problem from a broader application perspective and the authors touch upon issues such as security and privacy concerns. The authors propose mining this diverse set of inputs in order to make recommendations for individuals or a group of users, in order to discover patterns or frequent itemsets. They also present an experimental application, SocialFlicks which "recommends movie trailers to one or more users who are watching a common display". The undertaking described is a highly integrated system that has to deal with gathering and interpreting input from multiple sources, and its reliance on a network of sensors is probably something that places far away from being implemented in the foreseeable future.

In [49] Wang et al. propose another approach for music recommendations that uses a number of readily available data sources such as weather data and information from sensors that could be available on a smartphone, such as environment noise and luminance. The contextual information is used in conjunction with demographic data, such as the gender and age of users. The data are treated and then combined as factors into a fuzzy Bayesian Network which is used to infer the state of the user. The recommendation score is then calculated by taking into account the context evidence as gathered from the sensors and other input sources and combining that with preferences gathered from the users. This report suffers however from a lack of a quantitative evaluation so its utility to users cannot be concluded from it.

Karatzoglou et al. [33] proposed a multidimensional approach which directly takes advantage of the context in order to build the recommendation model. The authors list a number of advantages in using a multidimensional model, or *Multiverse Recommendations* as they describe their approach. The improvements include the lack of post or pre-filtering which can lead to information loss, improved computational complexity and the ability to handle an arbitrary number of context dimensions. The information about the users, items and context are stored in a multidimensional matrix. The context can have an arbitrary number of dimensions, for example time can be added as a dimension and location as another. While the results of this approach seem promising, the actual evaluation is somewhat limited. The experiments were performed on one dataset using semi-synthetic data where the contextual data were randomly generated, and two more datasets that used real user data but where limited in size, using data from a few hundred users which were asked to provide their context explicitly.

Chapter 4

Method

In this chapter we will present the two algorithms we implemented for this thesis. We focused on one algorithm from the *contextual post-filtering* paradigm and one from the *contextual pre-filtering* paradigm. The reason these types of algorithms were selected is mainly the fact that we can use any already established 2D recommendation algorithm as a base for the algorithms. Thereby all research efforts that have already been performed in the field of 2D recommender systems are still applicable in this setting, as opposed to contextual modeling, where an algorithm would have to be built from the ground up in order to take advantage of the contextual data.

Using a pre- or post-filtering approach also provides us with a straightforward way to compare the performance of the algorithms by comparing their performance to that of the non-contextual baseline algorithm. Making this comparison fair however presents us with some experimental design problems, especially for the pre-filtering algorithm, which we will discuss in detail in Chapter 5.

We will begin the chapter by describing the baseline algorithms that were used to create the 2D recommendation lists, followed by a description of the post-filtering method we implemented and close the chapter with a description of the pre-filtering method developed.

4.1 Baseline algorithms

Since the algorithms that were implemented for this project belong in the contextual pre-filtering and contextual post-filtering paradigms, a traditional 2D recommender had to be used as a source of recommendations for the algorithms. We chose to use two algorithms to test our algorithms, a popularity baseline and an item similarity baseline. The popularity baseline is a simple recommendation technique and the reason that we chose this method was its speed and simplicity. This allowed us to iterate quickly in the early stages of our research. The item similarity baseline is a more complicated algorithm and was used as a more realistic test of the performance the algorithms. Because the largest datasets we worked with contained tens of

thousands of users and hundreds of thousands of items, we had to make sure that the baseline algorithms were implemented in a high performance, parallel or distributed environment. We chose to use Graphlab [43], a high-performance, parallel machine learning framework. This allowed us to focus our efforts on the development of the contextual algorithms and not the baseline algorithms.

4.1.1 Popularity

The popularity algorithm is one of the simplest one can employ for recommending items, as it simply recommends the most popular items in the dataset to all the users. The popularity of the items is determined by the total number of streams for each track in the dataset, and all the users are presented with the same recommendation list, where the items are ranked according to their popularity. Despite its simplicity, the popularity algorithm can perform reasonably well, perhaps due to the power-law distribution observed in music consumption as examined in Section 2.5. In other words, the fact that popular items dominate the distribution of streams among users makes the popularity algorithm a viable approach. With a long enough recommendation list of the most popular items, there is a good chance that we will cover the preferences of a large number of users and be able to recommend items that the users will end up listening to.

Another advantage for this algorithm is lack of parameters and the fact that due to its simplicity we are able to make recommendations for datasets containing tens of thousands of users and hundreds of thousands of items very fast. Both of these factors contributed in making experimentation much easier, an important factor for a baseline algorithm.

This benefit however comes at a cost for the quality of the recommendations. Since the algorithm performs no personalization in the recommendations made, all users are presented with the same recommendation list. Its utility to the users is also limited, as popular items could be easily discovered by users on their own, so it is hard to use this algorithm in order to discover new tracks that the users might enjoy. As a result, metrics like coverage and novelty, described in Section 2.5, will suffer greatly. There is a limit to the performance of the algorithm in terms of accuracy, due to the variance of the users' listening habits. The method will perform well for homogeneous groups of users with similar listening habits, but will fail when faced with a group of users with diverse taste. The method also will fail in cases where the most popular items are excluded from the songs we are able to recommend. This is the case when we exclude songs that the users have already interacted with from the songs that we are allowed to recommend to them, in order to ensure novel recommendations and avoid *replay bias*. Replay bias is caused by the fact that users tend to replay their favorite songs often, and recommending them back to the user is an easy task that has no utility for the users.

If we exclude the items that make up the head of the distribution the difference in popularity between the items that reside in the tail is usually very small and is not a real indication of user preference.

4.1.2 Item similarity

We provided an introduction to *item-to-item collaborative filtering* in Section 2.1, so in this section we will briefly re-iterate the main assumptions behind the algorithm, and look at a few implementation details. The description we provide of the algorithm fits both the cases where we have explicit ratings available for the items, and the case where we only have implicit data, as was the case in our experiments.

Item-to-item collaborative filtering tries to overcome the problems of neighborhood based algorithms by looking at the similarities between items instead of the similarities between users. As the number of users and items grows the computation of similar users at recommendation time, as performed by most neighborhood-based algorithms, becomes very expensive computationally. What item-to-item CF does instead is to pre-compute the similarity between items and use these similarities to make recommendations to users, by predicting the rating the users would give to a new item according to the ratings they have given to similar items.

The main assumption behind this algorithm is that users will be more interested in items similar to those they have rated positively in the past, and less interested in items that are similar to those they have given negative ratings to. We should note that this negative feedback is not available for the case of implicit feedback ratings, which has a negative effect on the performance of the algorithm. The speedup for the algorithm comes from the fact that these similarities can be pre-computed and used in recommendation time. While this pre-computation could also be done for users in theory, we would be faced with the problem that user-to-user similarity is much more dynamic and can change dramatically as users rate more items. In contrast the relationships between items are much more static and should not change dramatically once a large enough number of ratings has been made on the items, allowing us to perform the expensive computation of the item similarity matrix periodically and not have to do an expensive search on the user-item matrix at recommendation time.

We will now provide a brief explanation of the recommendation process for this algorithm. The algorithm performs the rating prediction for a user u and an item i by examining the items that the user has rated and getting the similarity for each rated item to the target item i. After the most similar items are discovered, the rating is predicted by taking a weighted average of the ratings the target user has given to these similar items. Since each item is represented as a vector comprised of the ratings that users have given for that item we can use any vector similarity measure to compute the similarity between items. Two measures that we examined are the Jaccard similarity and the cosine similarity. Jaccard similarity, also known as Jaccard index, is defined in Equation 4.1. It measures the ratio of users the two items have in common versus the total number distinct users in both sets. X and Y are the item vectors with ratings for each user in the dataset as elements.

$$jac(\vec{X}, \vec{Y}) = \frac{|\vec{X} \cap \vec{Y}|}{|\vec{X} \cup \vec{Y}|} \tag{4.1}$$

Cosine similarity is computed as shown in Equation 4.2. It measures the similarity between the two vectors using the cosine of the angle between them, calculated as their inner product divided by the product of their magnitudes.

$$cos(\vec{X}, \vec{Y}) = \frac{\vec{X} \cdot \vec{Y}}{\|\vec{X}\|\|\vec{Y}\|} \tag{4.2}$$

We experimented with using cosine and Jaccard similarity and settled on using Jaccard similarity. The reasoning behind this decision is that any difference in recommendation quality should affect both the baseline and the contextual algorithms in the same manner. Since Jaccard similarity is less complex and therefore faster to compute we decided to use that for our experiments. In the parameter selection experiments we performed it also outperformed the cosine similarity, as can be seen in Subsection A.1.2.

4.2 Contextual post-filtering

Our first approach was a variation of a method described by Panniello et al. [48]. This is a post-filtering approach where the recommendations made by a traditional 2D recommender are contextualized using one of two methods, *Filter* or *Weight*, based on a contextual probability $P(u, i, c)$, where u is a user, i is an item and c is a context. In [48] the authors estimate this probability by retrieving a pre-defined number of nearest neighbors (NNs) for user u and examining how many from those neighbors have interacted with item i within the target context. The number of NNs who have interacted with item i divided by the total number of NNs retrieved gives us the contextual probability. In the *Weight* method this probability is then multiplied with the score produced by the 2D recommender to "contextualize" it and the list of recommendations is re-ranked based on the new score. In the *Filter* method we use the contextual probability to filter out recommendations that have a probability that is lower than a specified threshold.

In the variation we have implemented, instead of a contextual probability as already described, we calculated a contextual score. The score $ContScore(u, i, c)$ (4.3) is the sum of the playcounts of the nearest neighbors of u for item i in context c, divided by the number of neighbors. The assumption behind this approach is that items that users interact with more often in a certain context should be considered more important for that context and receive a higher score in the final list. In the initial experiments we performed, the performance of the two methods, using the contextual probability and using the contextual score did not show major differences in the quality of the recommendations, while using the contextual score provided a minor speedup in the running time of the algorithm so we selected to use that one for the rest of our experiments.

$$ContScore(u, i, c) = \frac{\sum_{u' \in NN(u)} r_{u'i}}{|NN(u)|} \tag{4.3}$$

In the *Weight* method, the final score for the recommendation was calculated using Equation 4.4 where $ContScore(u, i, c)$ is the contextual score and $2DScore(u, i)$ the original non-contextual score for the recommendation generated by the 2D algorithm. For the *Filter* method we use Equation 4.5, removing entries for recommendations that were below the set threshold, t. The threshold value we selected for the experiments was $t = 0.1$. This parameter is highly dependent on the dataset being used, so we took into consideration the distribution of values for the contextual score in the recommendation list, making sure that we have a large enough number of recommendations left after the filtering step to provide most users in the set with a non-empty recommendation list. The same threshold value was also used in [48]. The number of neighbors examined was another parameter that was set experimentally. Following the example of [48] we experimented with different values in the $10 - -200$ range and settled with 50 neighbors, as using more neighbors did not provide any clear benefit in the quality of recommendations, while increasing the size of the neighbor list has a negative effect on the execution speed of the algorithm. We can see the effect of neighborhood size on the performance of the algorithm in Figure 4.1.

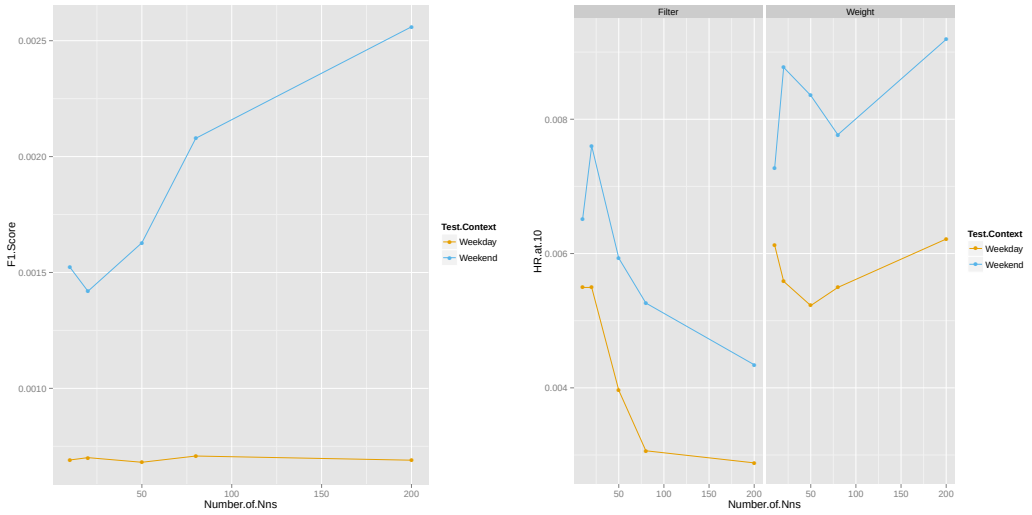
The measures we use in these experiments are F_1-Score and Hit Ratio at 10 (HR@10). F_1-Score is a summary precision-recall measure and HR@10 is a ranking measure, indicating how the algorithm performs when examining only the top 10 recommendations in the list. Both measures will be explained in more detail in Section 5.2 For F_1-Score we only include the plot for the *Filter* method, since the precision-recall measures remain the same for the *Weight* method regardless of neighborhood size. We can see no clear benefit when using more neighbors when we consider both the ranking HR@10 measure and the F_1-Score together, as we observe an increase for F1-Score for the weekend context but a decrease in HR@10.

We should note that the parameter selection experiments were performed only on the largest of the datasets, that is the 2 month October to December 2013 dataset. The reasoning behind this decision was that adjusting the parameters to a setting that performed reasonably well on the bigger datasets would allow us to test the technique's ability to generalize to other datasets, and we avoid overfitting the model by optimizing the parameter values to each dataset.

$$WeightedScore(u, i, c) = 2DScore(u, i) * ContScore(u, i, c) \tag{4.4}$$

$$FilteredScore(u, i, c) = \begin{cases} 2DScore(u, i) & ContScore(u, i, c) \geq t \\ 0 & ContScore(u, i, c) < t \end{cases} \tag{4.5}$$

Since the datasets we used for the post-filtering experiments contained thousands of users and tens of thousands of items, we had to perform a matrix factorization step on the user-item matrix to make the nearest neighbor search tractable. By factorizing the matrices we are able to model the user preferences using only a relatively small number of factors, providing a better separation between users, and



(a) F_1-Score for different number of NNs (b) HR@10 for different number of NNs

Figure 4.1. Effect of nearest neighbor count on performance of post-filtering

also making the calculation of the distance between two users much faster. For the factorization of the matrices we used a high-performance variation of the alternating least squares algorithm that we presented in Section 2.1. The variation used was cyclic coordinate descent (CCD) [50] and the specific implementation we used was CCD++[1], proposed by Yu et al. [70] which was chosen due its high performance in a parallel and distributed environment.

In order to retrieve the neighbors in an efficient manner, an approximate nearest neighbor search was performed on the user component of the factorized matrices. The nearest neighbor index building was performed using locality sensitive hashing [7], a probabilistic technique for finding approximate nearest neighbors in a high dimensional space. The implementation used was the *annoy*[2] library. It uses random projections, a technique to approximate the cosine distance between vectors by hashing the input vectors in random hyperplanes. It builds a tree by choosing a random hyperplane at every node of the tree, which divides the vector space into two subspaces. The tree construction is performed k number of times, creating a forest of k trees. The complete pre-processing of the data is illustrated in Figure 4.2

While using matrix factorization and approximate nearest neighbors allowed us to perform experiments on large datasets, it introduced many parameters to the algorithm that increased its complexity and made the design of our experiments harder. For CCD++ the most important parameter we had to choose was the number of latent factors used to represent the user-item matrix. We experimented with using 40 factors and 100 factors as Yu et al. did in their testing of the algorithm.

[1] `http://www.cs.utexas.edu/~rofuyu/libpmf/`
[2] `http://github.com/spotify/annoy/`

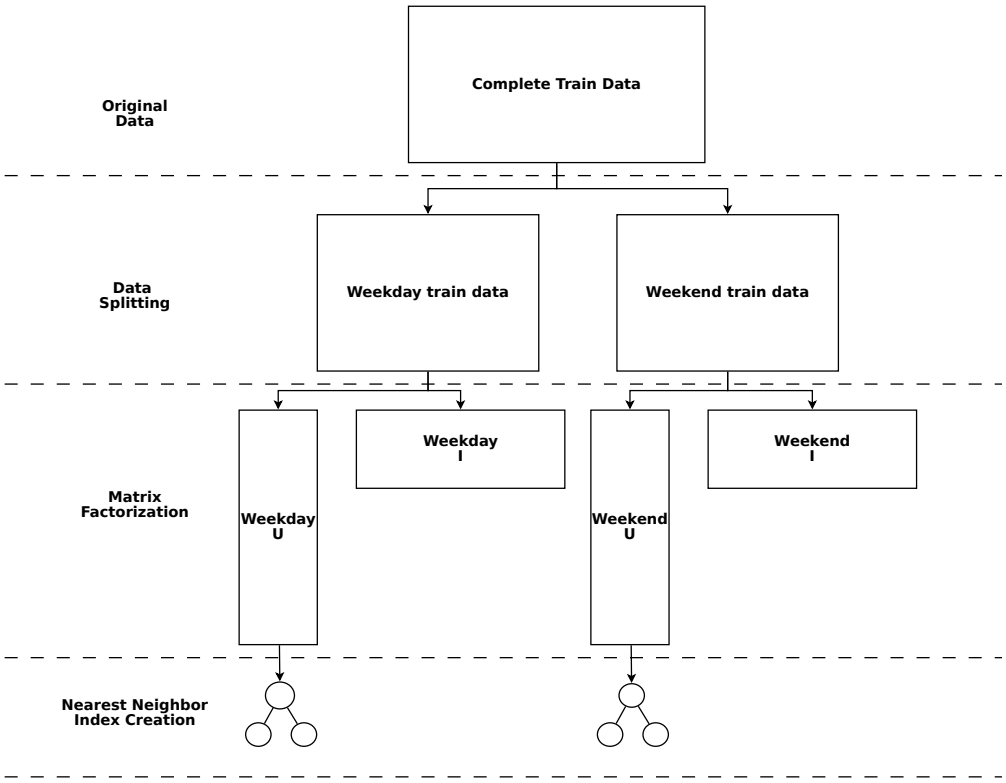


Figure 4.2. Pre-processing steps for post-filtering recommender

The experiments were performed on a dataset containing data from October to December 2013. We found that while using 100 factors did have a minor positive effect on the accuracy of the recommendations the effect was not significant, and given the degradation in running time that increasing the dimensionality causes we chose to use 40 latent factors. The results of these experiments are shown in Subsection A.1.1 of the appendix. For LSH we had to choose the number of trees that were built for the index. There we followed the advice of the library's author and created $2 * f$ trees, where f is the number of factors used in the matrix factorization step.

4.3 Contextual pre-filtering

The second algorithm we implemented was a pre-filtering approach. Using this approach we make recommendations using a traditional 2D recommender that was trained on the part of the complete data that is most relevant to the context we are making the recommendation in. Adomavicius et. al [4] present a *reduction-*

based approach. Using this algorithm we reduce the multidimensional problem of contextual recommendations to a traditional 2D recommendation problem. In this approach we create a different 2D recommender for each contextual dimension we are examining, using only data from the corresponding context to train each recommender. The appropriate recommender is then used to make recommendations for each context. The method we implemented is described as *exact pre-filtering* (EPF) in [4] as opposed to *generalized pre-filtering* which selects contextual information based on the best available *generalization* of the current context. In figure 4.3 we can see an illustration of how the baseline and pre-filtering recommenders were created for the experiments we performed, using the weekend-weekday context. The train data for the contextual recommenders are derived from the complete set using a process we will describe in more detail in Subsection 5.3.2

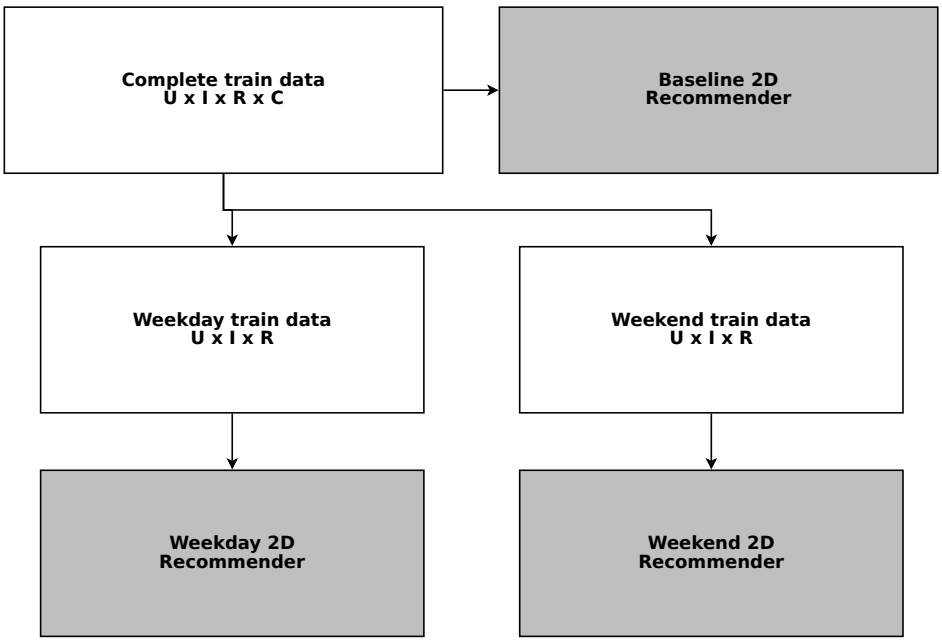


Figure 4.3. Creation of the different recommenders for pre-filtering

The assumption behind EPF is that by using more specific training data we will be able to provide recommendations that are more relevant to the current context, thereby negating the expected negative effect of the reduction in available train data. For example, a user's listening habits during the weekend could be different from his habits during the week. This difference will be reflected in the data and therefore the more specific recommenders will achieve better performance from one that has been trained on the less specific, complete dataset. One concern with reduction-based approaches is the selection of the type of context on which the datasets are

split. Having redundant contextual splits can lead to decreased performance due to the lack of semantic meaning in a context. For example, splitting the context into even and odd hours, as the authors of [8] did to test their hypothesis, does not contain any semantic meaning and we don't expect it to provide an improvement in performance. However the authors did obtain a result that outperformed their baseline, indicating that there are still issues with the understanding of the way that such a contextual split models user behavior. Apart from the lack of semantic meaning, the increased data sparsity from slicing down the data can be problematic. For a very specific type of context we might end up with too few training points to make a relevant recommendation and the baseline could end up always outperforming the very specific contextual recommender.

Chapter 5

Evaluation

In this chapter we present the experiment results and analyze the performance of the algorithms. We will investigate the differences in performance between the baseline and the developed algorithms. We begin the chapter with a description of the datasets that were used for the evaluation. Then, we present an analysis of the experiments performed, starting with a presentation of the metrics used for the evaluation, and a look at the performance of each algorithm starting with the post-filtering algorithm and continuing with the pre-filtering algorithm. We also dedicate a section to the experiment design challenges we had to tackle during the testing of the pre-filtering algorithm.

5.1 Datasets

Our main source of data was implicit user-item interactions measured as the number of times a user interacted with an item within the time and context specified. The playcount data were gathered from real users of Spotify. We aggregated streams from users in the United Kingdom during a particular time period of every day. Instead of using binary scores to indicate whether a user interacted with an item or not, as done in [32], streams were weighted according to the play source. For example we might weight streams that originated from a user searching for a specific track more heavily than streams that occurred due a track being the next song in a playlist. We also made sure to give small weights to streams originating from sources where recommender systems generated the playlists such as the Radio feature of Spotify. In order to remove artifacts such as skipped songs we only considered items which the user listened to for more than 30 seconds.

Data were gathered over specific periods of time and labeled as streams occurring in a specific context, namely whether the stream occurred during the weekend or on a weekday. A number training datasets were created for this purpose. We gathered the data using streams from different time periods, including stream data for two weeks in June 2013, November 2013 and February 2014. These datasets were used to explore the performance of the algorithms in different time periods. Larger

datasets were also created in order to provide the algorithms with more training data and to allow for further exploration of their performance. The post-filtering method was tested on datasets that reached up to 2 months of user data, gathered during October to December 2013, while the pre-filtering algorithm was also tested on datasets that contained 6 months and one year of user data, ranging from April 2013 to April 2014. In total we examined data from 6 different time periods. The largest dataset that we examined, that is the one year dataset, contained data from 30,353 users interacting with 1,103,127 items, for a total of 5,580,489 interactions. As we will mention later though, we only examined the interactions made with the 20,000 most popular items for the pref-filtering algorithm.

For each experiment two test sets were created, one with data from a weekday and one with data from a weekend. The contextual algorithm and the corresponding baseline were then both tested on these datasets.

For the post-filtering experiments we created test sets from different days following the training set. For example, for a train set gathering data in the 2013-06-10 to 2013-06-24 period, the test set for the weekend context was created using data gathered on 2013-06-29 which was a weekend day, and the test set for the weekday context was created from data gathered on 2013-06-27 which was a weekday.

For the pre-filtering experiments the complete data were split into a train and a test set, using a process that we will describe in detail in Subsection 5.3.2. In short, we split the complete set into a train and a test set and then used subsets of the complete test set to create the contextual test sets, thereby ensuring that user-item pairs that appeared in the training sets did not appear in any test set.

In all of the experiments we performed, we generated a list of 300 user-item recommendation pairs, recommending only items that the users had not already interacted with in the training set. That allowed us to avoid *replay bias* i.e. the fact that users will tend to replay their favorite tracks making the recommendation of such tracks an easy way to boost the accuracy scores of an algorithm, but providing no real utility to the users. We chose a list of length 300 in order to have sufficient depth in our recommendations, while at the same allowing us to be able iterate relatively quickly on our experiments.

5.2 Metrics

In this section we will present the metrics used for the evaluation of the algorithms, and briefly explain how each metric measures a different aspect of the algorithms. The metrics used for the evaluation were accuracy based metrics like precision-recall as well as ranking measures. In particular we used *overall precision*, *overall recall*, and F_1-*Score* as our precision-recall metrics, while *Hit Ratio* and *Mean Percentage Ranking* were used as our ranking metrics.

5.2.1 Precision-Recall measures

Overall precision (5.1) is defined as the ratio between the number of correct recommendations made divided by the size of the recommendation list. By correct recommendations we mean user-item pairs that appeared both in the recommendation list and the test set. It measures the ability of the algorithm to recommend songs that are relevant to the user versus recommendations that are considered irrelevant.

$$precision = \frac{|\{(u,i)\ in\ test\ set\} \cap \{(u,i)\ in\ recommendation\ list\}|}{|\{(u,i)\ in\ recommendation\ list\}|} \tag{5.1}$$

Overall recall (5.2) is defined as the ratio between the number of correct predictions versus the size of the test set. Recall measures the probability that an item the user considers relevant was actually recommended to the user, or in other words the algorithm's ability to cover the users' preferences.

$$recall = \frac{|\{(u,i)\ in\ test\ set\} \cap \{(u,i)\ in\ recommendation\ list\}|}{|\{(u,i)\ in\ test\ set\}|} \tag{5.2}$$

F_1-Score is defined as the harmonic mean of precision and recall, and can be used as a metric that provides a summary of both precision and recall. Its definition is given in Equation 5.3

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{5.3}$$

In addition to overall precision and recall we also create Precision-Recall (PR) curves, a variant of ROC curves, to measure the performance of the algorithm at different precision and recall levels. Following the example presented by Schein et al. [59] we created Global and Customer PR curves. Global PR curves are similar to the ROC curves suggested by Herlocker et al. in [29]. They are useful when "we are allowed to recommend more often to some users than others". A use case for this example would be playlist backfilling where we have to add a certain number of songs to a playlist in order to reach a certain length as indicated by the the user or other factors. They are constructed using Algorithm 1.

Algorithm 1 Global PR Curve Calculation

```
1: procedure GLOBAL PR CURVE(points)
2:     Order the recommended user-item pairs according to descending score
3:     for points do
4:         Pick number k, calculate precision and recall using only the
           top k recommendations, use PR values to plot the point
5:     end for
6: end procedure
```

The first step of the algorithm is to sort the recommendation list in descending order of score, thereby placing the recommendations the algorithm indicates as more relevant in the beginning of the recommendation list. We then repeat the step in Line 4 as many times as we indicate with the variable *points*, which determines how many points our curve will have. What we do in this step is select the top k recommendations and determine the recall and precision using only those points. The number k is determined by the number of points we want our curve to have and the size of the recommendations list. For our experiments we created 30 points for each curve.

The customer PR curve indicates the performance of the algorithm in cases where the same number of items has to be recommended to all the users. This is the typical case where we have to provide all our users with a list of songs that fit their preferences, for example in a song discovery setting. The algorithm illustrated in Algorithm 2 is similar to the one for the global PR curve, only this time the recommendation list is sorted according to users first and then secondary sorted on the score. The items recommended are also selected so that we pick the top k items *for each user*.

Algorithm 2 Customer PR Curve Calculation

```
1: procedure CUSTOMER PR CURVE(points)
2:     For each user, order the recommended user-item pairs according
       to descending score
3:     for points do
           Pick number k, calculate precision and recall using only the
4:         top k recommendations for each user, use the PR values to
           plot the point
5:     end for
6: end procedure
```

5.2.2 Ranking measures

The ranking measures we used were *Hit Ratio* (HR) and Mean Percentage Ranking (MPR).

HR [34] is a recall based measure that calculates the percentage of items recommended in the top-k part of the recommendation list that were *hits*, where top-k is defined per-user as was done in the customer PR curve. As such, the measure is equivalent to a per-user recall-at-k measure. This measure was calculated for $k = [10, 20, 30]$. HR indicates the performance of the algorithm near the top of the recommendation list for each user. It is an important measure as users are unlikely to search too deep in a recommendation list [25]. A formal definition is given in Equation 5.4. We will use the syntax HR@k in this thesis to indicate the HR score at the depth k.

$$HR(k) = \frac{|\{(u,i)\ in\ test\ set\} \cap \{(u,i)\ in\ top\text{-}k\ recommendation\ list\}|}{|\{(u,i)\ in\ test\ set\}|} \tag{5.4}$$

MPR [35] is a measure of a user's satisfaction with an ordered list of items. For every generated list of ranked recommendations, let $rank_{ui}$ be the percentile ranking of item i in the ordered list of all items for users u. A $rank_{ui} = 0\%$ would indicate that the item i is the most preferred item for user u. A $rank_{ui} = 100\%$ indicates that i is predicted to be less desirable for user u. The percentile ranking is then evenly distributed among the remaining items in the list by steps of $\frac{100\%}{|R|}$ where R denotes the list of recommendations. MPR is then defined by the expected percentile ranking, according to the preference shown by a user in specific items. The preference is determined by the number of times a user has streamed those items, and their position in the recommendation list, determined by $rank_{ui}$ as shown in Equation 5.5, where r^t_{ui} denotes the number of times user u has streamed item i in the *test set*.

$$MPR = \frac{\sum_{ui} r^t_{ui} rank_{ui}}{\sum_{ui} r^t_{ui}} \tag{5.5}$$

For MPR, lower values are more desirable since they indicate that items placed higher in the generated recommendation lists were listened to more often in the test set by the users.

5.3 Analysis

In this section, we present the analysis of the experiments performed and the performance of the two algorithms. We start by analyzing the results of the contextual post-filtering method, and then present a validation process and an explanation for the observed results. Then, we examine the challenges in experimental design for the pre-filtering method and how to overcome them. After a proper experimental process has been established, we proceed with a presentation of the pre-filtering results, along with the discussion of the algorithm's performance.

5.3.1 Contextual post-filtering

The first algorithm we implemented and performed experiments with was the post-filtering algorithm, using the *Weight* and *Filter* methods explained in Section 4.2. The parameters used for the algorithm were a neighborhood size of 50 for obtaining the contextual nearest neighbors and the threshold for the *Filter* method was set at 0.1. We tested the algorithm against a test set selected within the specified context on a day occurring shortly after the train set. As we will see below the algorithm ended up under-performing the baseline for reasons that we will explain in the following sections.

Comparison of methods

Figure 5.1 and Figure 5.2 show the performance of the *Filter* and *Weight* methods, compared against the popularity baseline. We can see that the *Filter* method outperforms the baseline in precision but at a cost in recall. The F1-score favors the filtering method but not significantly. Note that since the precision-recall metrics are identical for the baseline and *Weight* method, as the items in the recommendation list are the same, we have collapse their values into one box, indicated by the yellow color in the precision-recall figures. The ranking measures for the post-filtering methods consistently under-perform the baseline however.

In Figure 5.3 and Figure 5.4 we can see the problems of the post-filtering methods in the PR curves generated from the 2 month October-December 2013 dataset. The *Weight* method under-performs the popularity baseline in both the Global and Customer PR curves, indicating that the contextual score does not actually provide enough information to improve the ranking. Since the size of the recommendation lists returned by the *Filter* method is much smaller than those returned by the *Weight* and baseline methods, its recall measure suffers significantly as evidenced by the fact that its curve stops expanding towards higher recall values much sooner than the other two methods. We can however see the better performance precision wise in the Global curves in Figure 5.4.

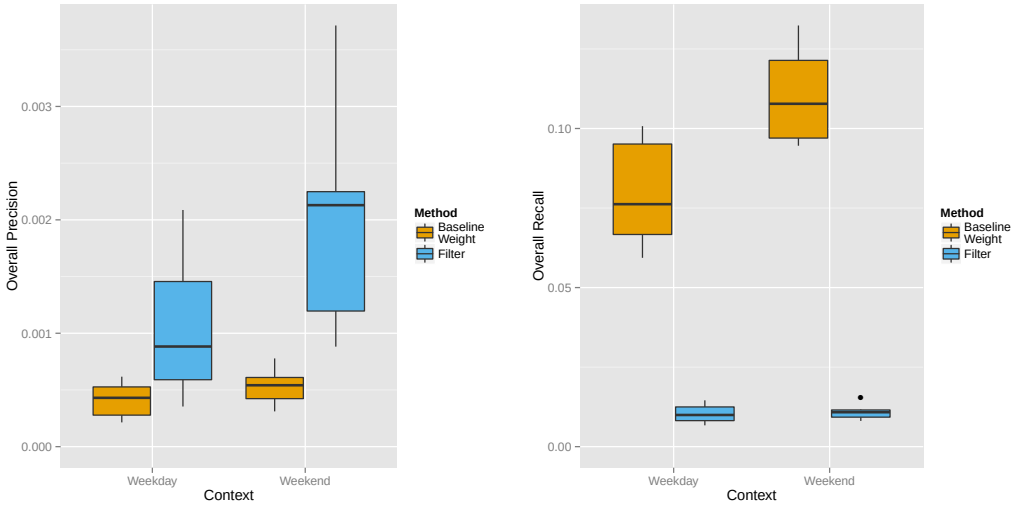
The performance of the *Filter* algorithm can be explained by the fact that it reduces the number of recommendations by a significant factor, depending on the threshold used. That results in increased precision as the recommendations become much more specific but at the same time the recall will suffer, as confirmed by our experiments. As we increase the threshold value the performance of the *Filter* method will approach the performance of the baseline until the two methods become identical for a high enough threshold value.

The performance of the *Weight* method however is an indicator for the value of the post-filtering method overall. Since the ranking is not improved by incorporating the contextual score, it indicates that the contextual score does not provide enough value in the recommendation. This realization lead to to a re-examining of the partly positive results in the *Filter* method which we present in the following section.

Validation of results

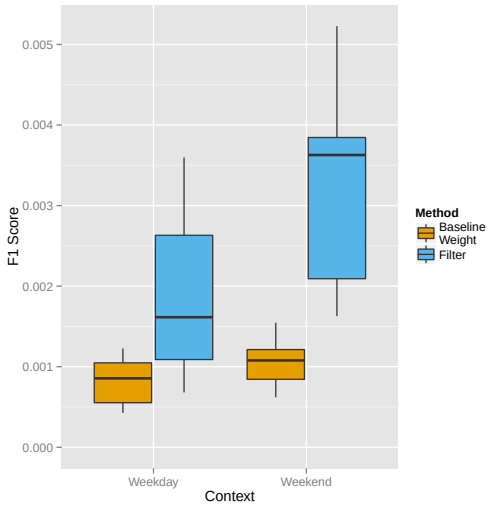
The fact that the *Weight* method under-performed the popularity baseline in all our experiments tells us that the value of the contextual score as an indicator of user preference in a context is limited. Since the same contextual score is also used as an indicator of whether or not to keep a recommendation from the initial list in the *Filter* method, a re-examining of the results had to be performed in order to determine the true reason of the increase in precision. The hypothesis we tested against was that the decreased length of the recommendation lists were the main factor for the increase in precision.

For our first experiments we created recommendation lists that had the same



(a) Post-filtering Precision

(b) Post-filtering Recall

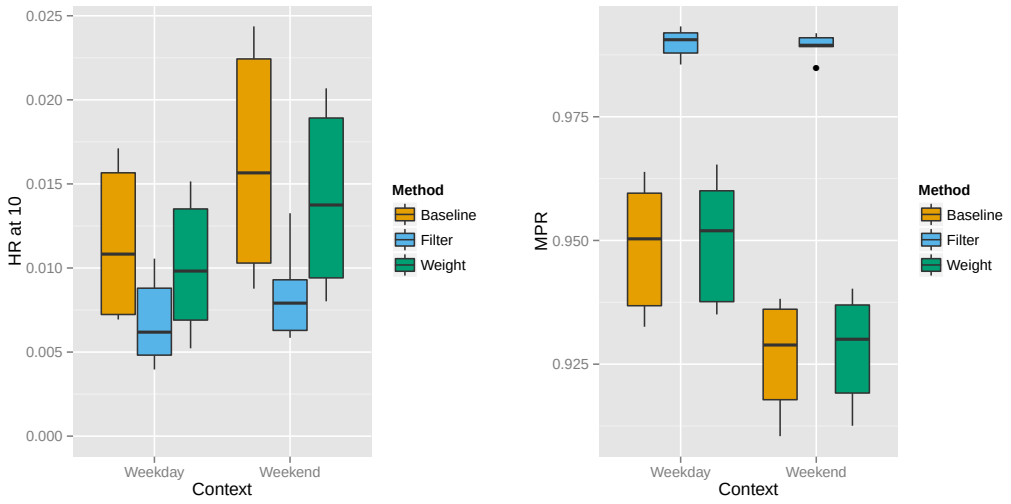


(c) Post-filtering F_1-Score

Figure 5.1. Precision-recall measures for post-filtering

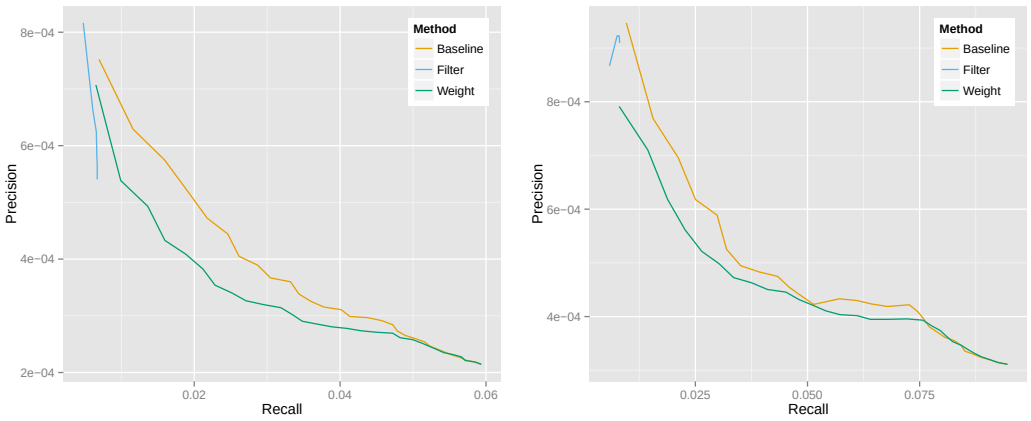
length as those created by the *Filter* method, however the recommendations that were retained were selected randomly. The positive effect of the *Filter* method remained in that case, so in order to make the test stricter we selected from each recommendation list the top-k items according to the score assigned to them by the baseline recommender, where k was the length of the recommendation list returned by the *Filter* method for that experiment.

As can be seen in Figures 5.5 and 5.6 selecting the top-k recommendations from



(a) Post-filtering HR@10 (b) Post-filtering MPR

Figure 5.2. Ranking measures for post-filtering



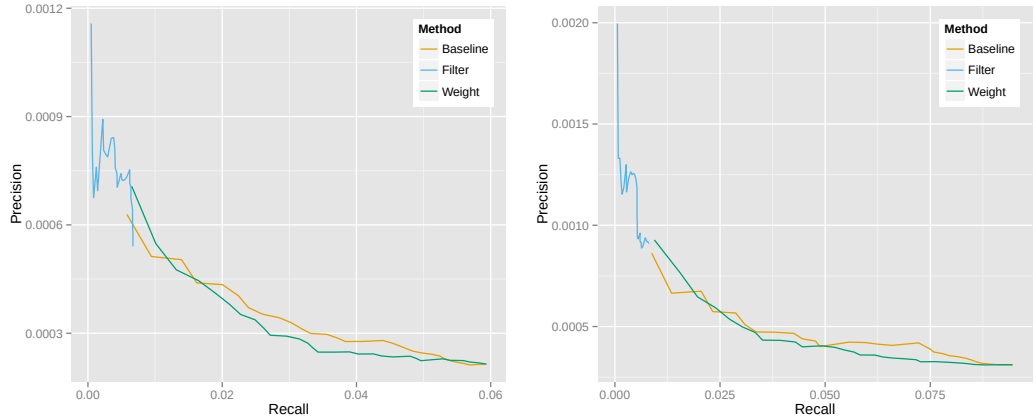
(a) Customer PR Curve Weekday (b) Customer PR Curve Weekend

Figure 5.3. Customer PR curves for post-filtering

the baseline algorithm actually out-performs the *Filter* method in all metrics, as we expected after we observed the poor performance of the *Weight* method.

Discussion

As we described in Section 4.2 post-filtering is an involved process with a number of parameters to be set, and any step of the process can influence the end performance of the algorithm. Our assumption for the poor performance of the algorithm is that



(a) Global PR Curve Weekday

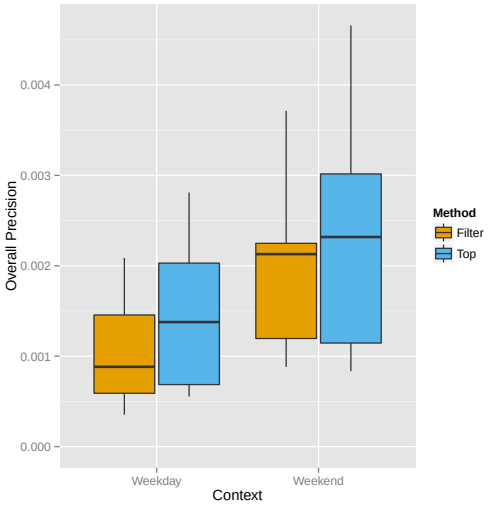
(b) Global PR Curve Weekend

Figure 5.4. Global PR curves for post-filtering

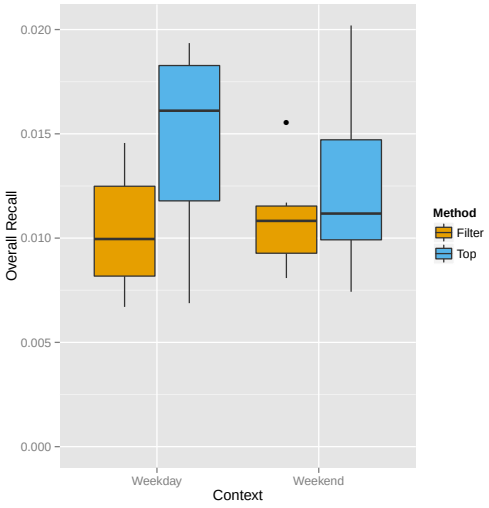
the value of the contextual information is in fact lost during one of the preparatory steps of the algorithm. To reiterate the process briefly, first we have to create approximate nearest neighbor indices for the users based on the user component of the factorized user-item matrix. We then use these approximate nearest neighbors at recommendation time in order to calculate the contextual score for each recommendation according to the preference that the nearest neighbors of the target user have shown in the target item, for the target context.

The matrix factorization step and approximate nearest search are by definition sources of information loss as we exchange accuracy for performance. Since the calculation of the contextual score depends on the results of two probabilistic operations in a row, we are faced with a case of *propagation of uncertainty*, that has an immediate negative influence on the result. In other words since we are performing an approximate nearest neighbor search on a matrix where we have reduced the item dimensions from tens of thousands to 40 as we did in the case of the user matrix, it is possible that the nearest neighbors that we obtain for the users are not representative of the true similar nearest neighbors in the initial data.

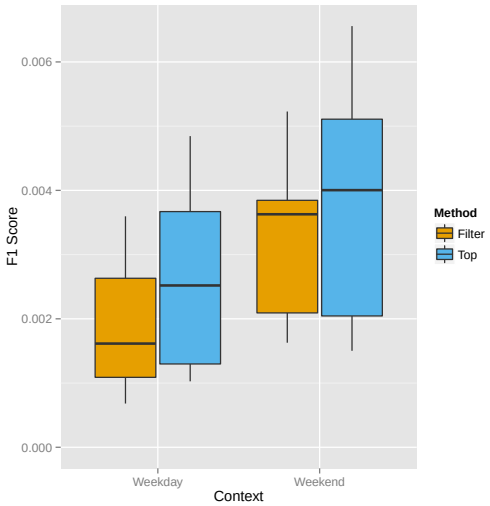
Another important consideration is the distribution of the contextualscores, where we observe that the majority of the user-item pairs, receive a contextual score of 0, as shown in Figure 5.7 which was generated from the recommendation list for the two month October to December 2013 dataset. For this plot we set a cutoff value at 0.30, as we only had less than 2000 user-item pairs where the contextual score was higher than that, from a list of more than 3,000,000 user-item pairs. In fact, for this dataset, more than 80% of the recommendations have a contextual score of 0. A contextual score of 0 means that none of the nearest neighbors of the target user has streamed the target track in that context. This extremely skewed distribution of contextual scores can be explained due to two reasons. First, many



(a) Filter vs Top Precision



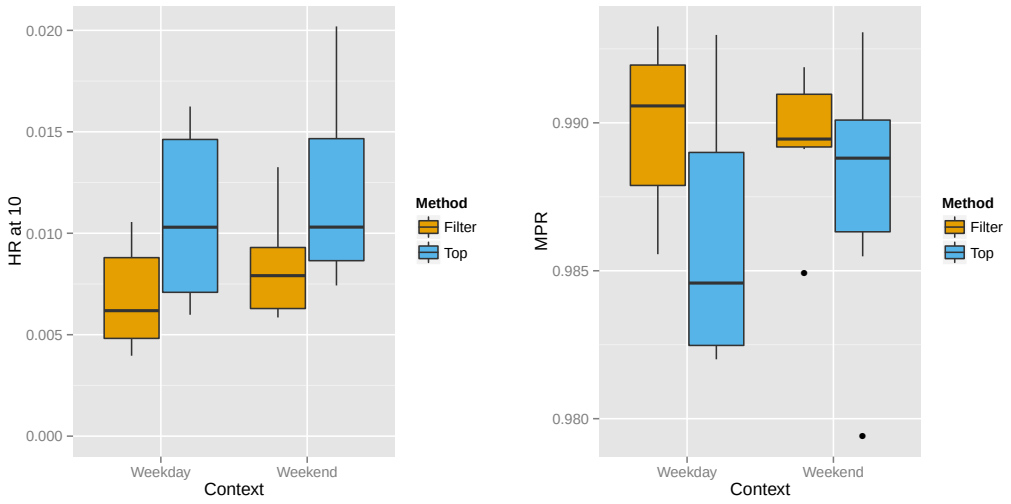
(b) Filter vs Top Recall



(c) Filter vs Top F_1-Score

Figure 5.5. Precision-recall measures for Filter against selecting the top recommendations

of the items that appear in the complete set do not appear in the contextual sets, and as a result will receive a contextual score of 0. Second, the power-law distribution of music item consumption means that apart from a number of popular items, the majority of the items listened to by the users will belong in the long tail. Due to the vast amount of items available, and the fact that most users will listen to a very large number of items just once, the probability that a user will have listened to the same item as a user determined as similar will have in a particular context



(a) Filter vs Top HR@10

(b) Filter vs Top MPR

Figure 5.6. Ranking measures for Filter against selecting the top recommendations

becomes very small.

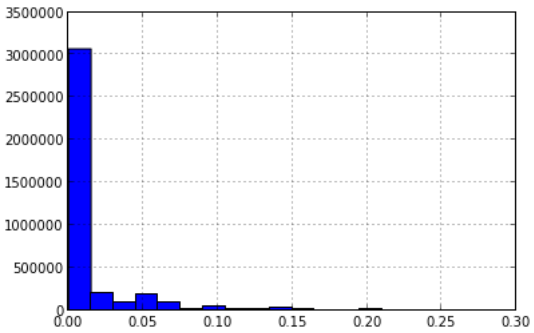


Figure 5.7. Distribution of contextual score values

For these reasons the majority of the contextual scores end up being zero, which provides us with no contextual information on most of the recommendations and leads to the overall worse performance of the algorithm.

After observing this result we can expect that changing the baseline algorithm to the item similarity one is not going to have a major effect in the performance of the post-filtering method. To verify this hypothesis we performed an experiment using item similarity as a baseline on the October-December 2013 dataset, where the post-filtering method was again unable to out-perform selecting the top entries of the algorithm.

5.3.2 Contextual pre-filtering

In this section we will present the performance of the pre-filtering algorithm. We will start by looking at the challenges with experimental design for pre-filtering algorithms, including a brief look at our first attempts at designing an experiment and how we corrected the initial mistakes made. We will then present the performance of the algorithm versus the two baseline algorithms, popularity and item similarity, focusing on the effect the time period over which data were gathered has on the performance of the algorithm.

Experimental design

When dealing with different train sets for algorithms that we want to compare against each other, as is the case with contextual pre-filtering, one must be very careful to ensure that the algorithms are tested on equal grounds.

Our initial experiment design for pre-filtering was similar to the one we used in the post-filtering case. Each recommender was trained on a slice of the complete data corresponding to some context, so in the experiments we performed we had one baseline recommender trained on the complete set, one trained on only the weekend data and one trained on only the weekday data. The test sets were again taken from a day according to the context, from a date shortly after the end of the training data. For example for the 2 month dataset, comprising of user data gathered during October to December 2013, the test data for the weekend contextual segment were gathered from the first Sunday after the train set and similarly for the weekday test data.

We performed the first batch of experiments using this setup and we got back results where the pre-filtering algorithm was out-performing the baseline algorithms on most of the measures, many times by a margin of 25% in the weekend set. Given that the size of the weekend train data was approximately one third of the size of the complete data that kind of performance improvement seemed very unlikely, especially when we consider the fact that a 2 month dataset will most likely not be large enough to determine some recurring pattern of user behavior. After investigating the design of the experiment we determined that the way that we excluded some items from being recommended was the cause for this abnormal performance increase.

As we mentioned in the previous section, the items that we recommended for each user could only be items that the user had not already interacted with. Since the users had more interactions on average in the complete set when compared to the contextual sets it was also more likely that they will interacted with a larger number of distinct items. As a result the recommender trained on the complete data had a smaller "pool" of songs from which to recommend to for each user, which lead to worse performance when compared to the contextual algorithms trained with the smaller datasets. Ensuring that the same number of distinct items appeared on all the train sets did not help either, because even if the number of distinct items is

the same in both sets, the users in the complete set will *on average* interact with more of the items, thereby again limiting the individual "pool" of items available for recommendation for *each user*.

In order then to correct the experimental design we had to move away from using a test set that came from separate day, as that way it was not possible to ensure parity in the train and test sets of all the segments. We instead adopted the experimental design proposed by Baltrunas et al. [8], which was specifically designed for pre-filtering recommender systems. An illustration of the idea is provided in Figure 5.8, adapted from [8]. The main goal of this design is to ensure that the train and test sets are completely disjoint for the complete as well as the contextual segments.

Initially the complete dataset is split into the contextual segments, in our case corresponding to the weekend and weekday data. We also split the complete set into a train and test set. We then extract the test sets for the contextual segments by extracting the user-item pairs that are present in both the complete test set and the contextual segment. After extracting the test set from the contextual segment, the remainder of the segment is used as the training set for the contextual recommender. The important difference here is that we don't split each contextual segment into a train and test set individually as that way we could end up with some user-item pairs that are present in the test set also being present in the train set. For our experiments we initially split the complete dataset into two parts, using 90% of the data for training and 10% for testing.

Pre-filtering vs. Baseline

In order to test the performance of the pre-filtering algorithm against the baseline we used data from different periods in the year, as well as gathered during different time-frames and on different platforms. We tested the algorithm against two different baselines, the popularity-based algorithm and the item similarity. Contrary to the post-filtering algorithm there were no parameters to be set in the pre-filtering algorithm itself, but rather in the algorithms used as a baseline, such as for example the similarity measure for the item similarity algorithm.

One important difference in the way that the experiments were performed, apart from the experimental design we already described, was limiting the items to the top 20.000 most popular ones in our experiments. The reason this limitation had to be imposed was the memory requirements of building the item similarity matrix. The number of items our complete sets contained made the calculation intractable for the complete set, since the similarity has to be calculated in a pairwise manner. However since the advantage of having fewer items to recommend from applies to both the baseline as well as the pre-filtering algorithms the comparison between their relative performance remains valid. Especially in the case of the popularity baseline the limitation makes no difference since the algorithm will just recommend the k most popular tracks to users either way, where k is the length of the recommendation list we generate, set to $k = 300$ for our experiments. Limiting the tail of the track

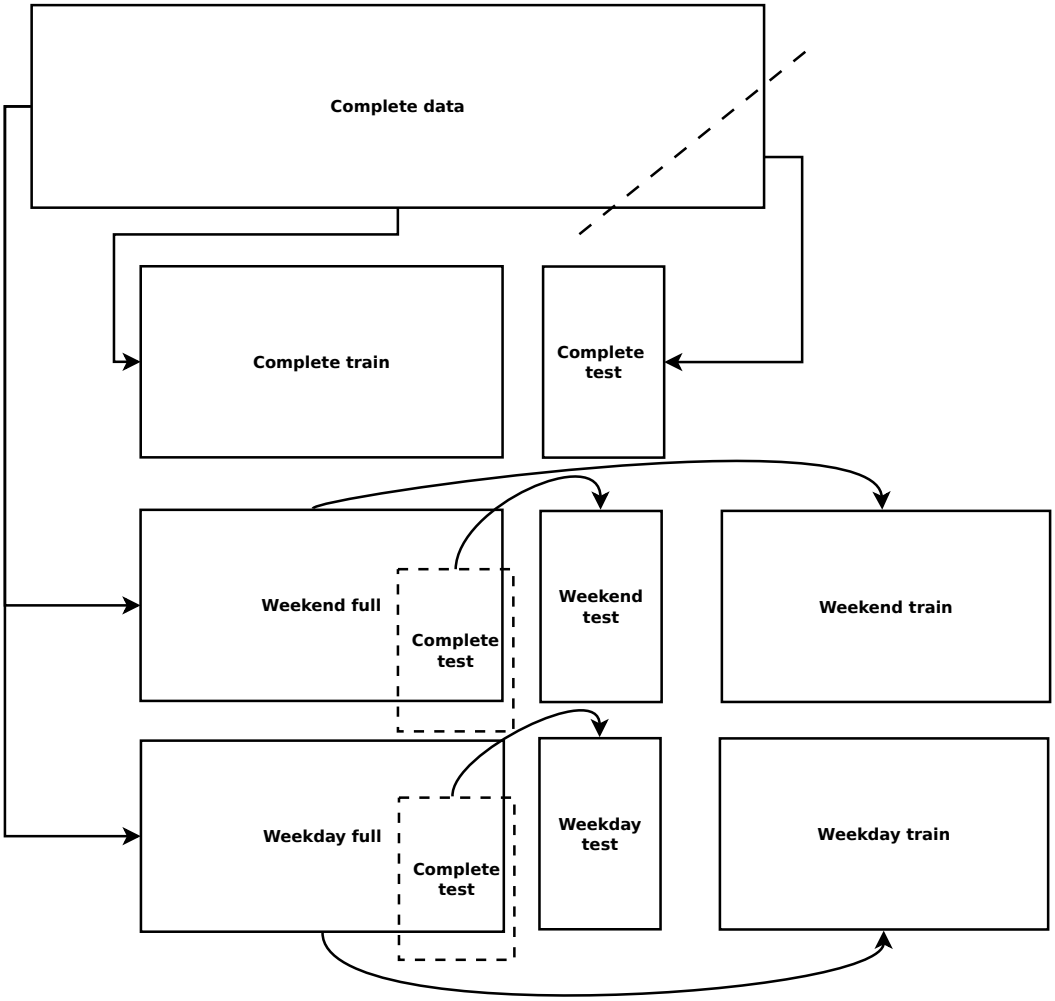
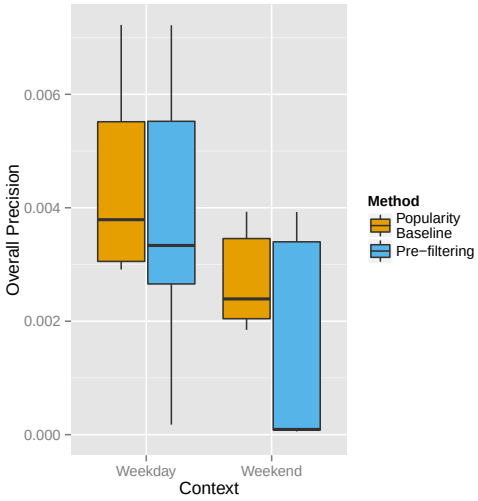


Figure 5.8. Illustration of experiment design for pre-filtering

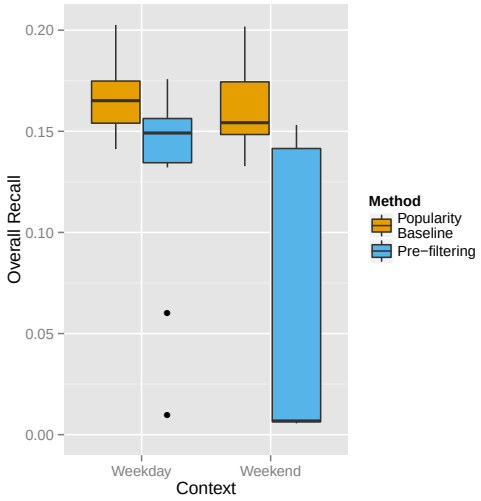
distribution then has no impact on the performance of the algorithm.

Our first look will be at the overall performance of the algorithm in all the datasets. Figures 5.9 and 5.10 show the aggregated performance of the algorithm versus the popularity baseline. Using the popularity as the input algorithm results in performance that is worse than the baseline. The justification for this drop in performance comes from the way that the popularity algorithm works. Since the contextual recommenders are trained on datasets that are smaller than those of the complete recommender, the popular items in those recommenders will be less diverse and perhaps missing some of the items that are in fact the most popular ones when we look at the complete dataset. In other words when we use the popularity of items to make recommendations, using smaller datasets will result in worse performance as smaller datasets will capture a smaller part of the taste of users. This results

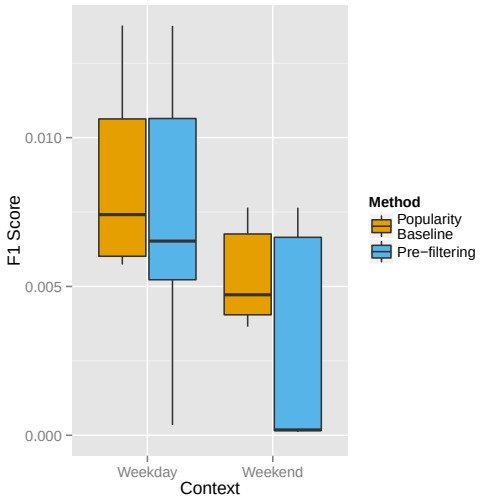
in the recommendation of items that are popular in a segment of the dataset, but the overall most popular songs will still dominate the user plays and those can be missing from the contextual datasets.



(a) Pre-filtering vs. Popularity Precision



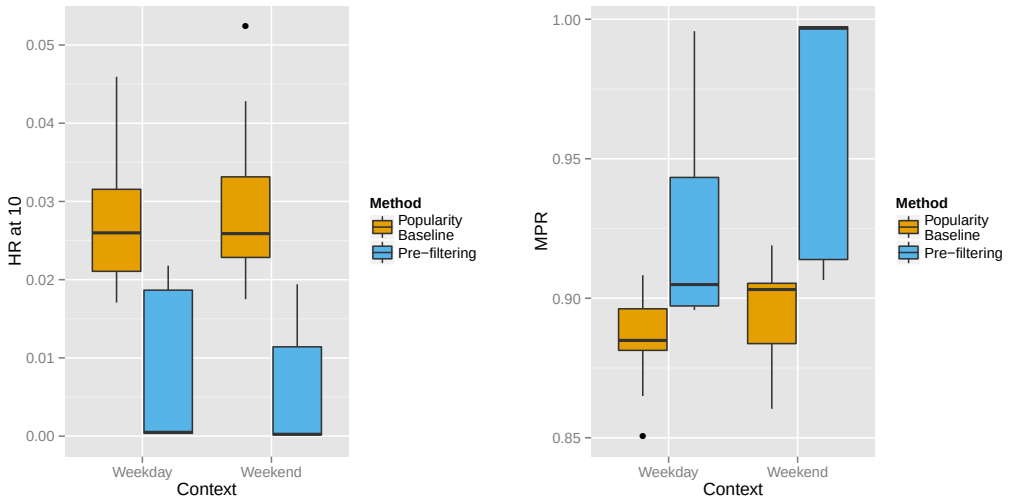
(b) Pre-filtering vs. Popularity Recall



(c) Pre-filtering vs. Popularity F_1-Score

Figure 5.9. Precision-recall measures for Pre-filtering vs. Popularity

Another observation we can make is the high variance of the results, something which is more obvious for the weekend results. This indicates another disadvantage of the popularity method, in that its success depends in a large way on the set of items that we exclude when making the recommendation. Exclusion of many items



(a) Pre-filtering vs. Popularity HR@10

(b) Pre-filtering vs. Popularity MPR

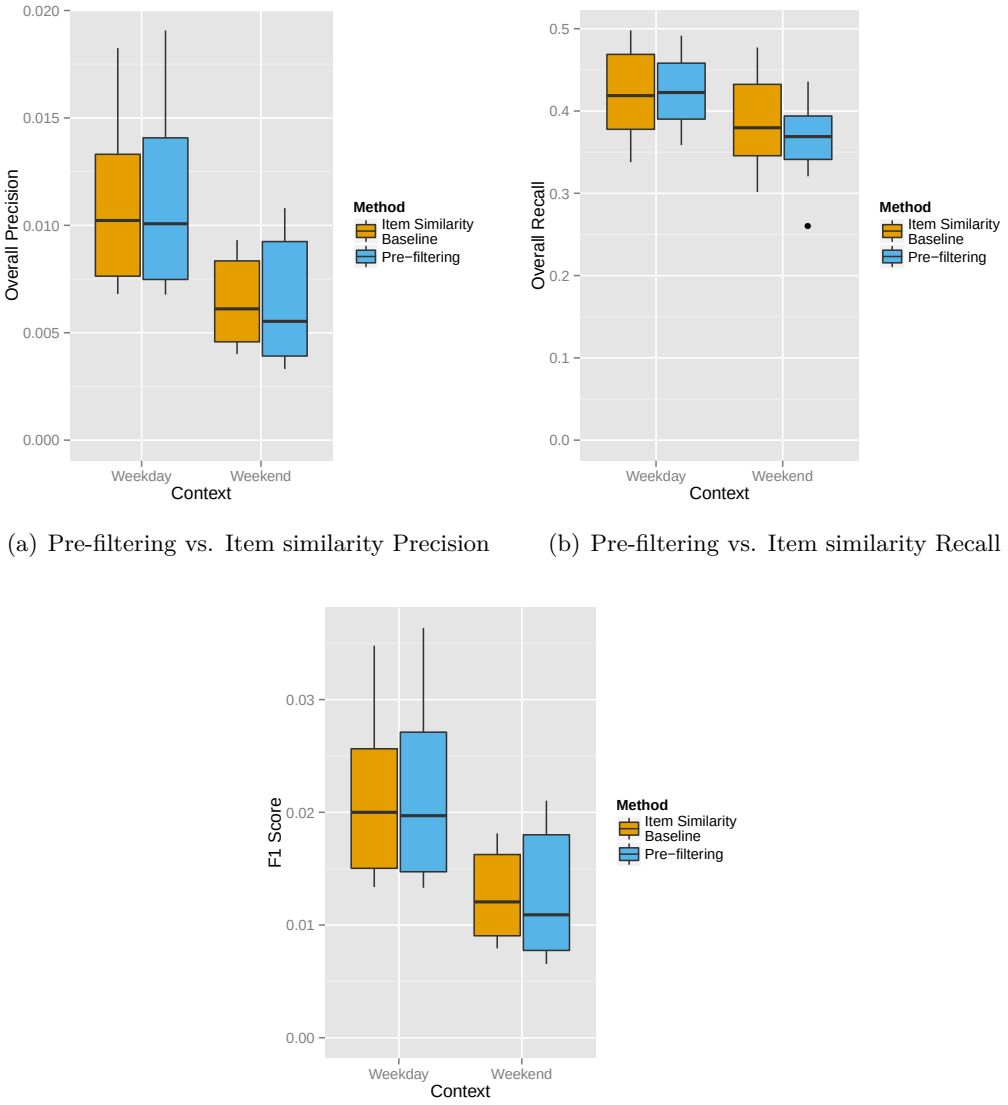
Figure 5.10. Ranking measures for Pre-filtering vs. Popularity

from recommendation as happens in the case of the weekend can lead to worse recommendations, depending on the dataset examined.

The situation changes when we use the item similarity algorithm as input as we can see in Figures 5.11 and 5.12. The performance of the baseline and the pre-filtering algorithms are now mostly on the same level, with the pre-filtering algorithm actually out-performing the baseline for the HR metric. We see that the large variations in performance are not a problem with the item similarity algorithm, since the recommendation lists generated are suited to the preferences of each user. We are now recommending items in a way that is tied to how users consume the items and the relationships that emerge between items, providing a better model for the users' preferences which is also made evident by the difference in magnitude between the metrics when using the popularity and the item similarity baselines.

Pre-filtering for larger datasets

One interesting observation we made was the continuous improvement in relative performance of the pre-filtering method against the baseline as we increased the size of the dataset. Figures 5.13 to 5.18 show the performance of the algorithm when examining only the datasets that contain at least 6 months of training data. In the case of the item similarity baseline, shown in Figures 5.13 and 5.14, we now see an advantage in all the metrics for the pre-filtering method, and a more pronounced gain for HR metric. While the performance gain is generally not significant, one has to consider the fact that the contextual recommenders were trained with much less data and nonetheless out-perform the baseline algorithm. The difference in



(a) Pre-filtering vs. Item similarity Precision

(b) Pre-filtering vs. Item similarity Recall

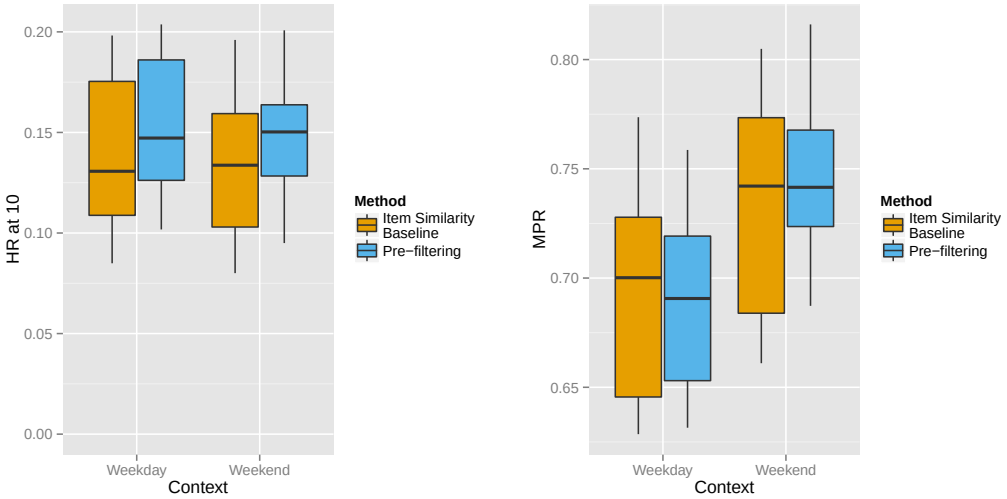
(c) Pre-filtering vs. Item similarity F_1-Score

Figure 5.11. Precision-recall measures for Pre-filtering vs. Item similarity

performance is also evident in the PR plots presented in Figures 5.15 and 5.16.

Our assumption for this change is that as we add more usage data in the model some clear patterns emerge in the way that the users consume music in the different contexts, which were not possible to be detected in the shorter datasets due to the short term drift of the users' preferences.

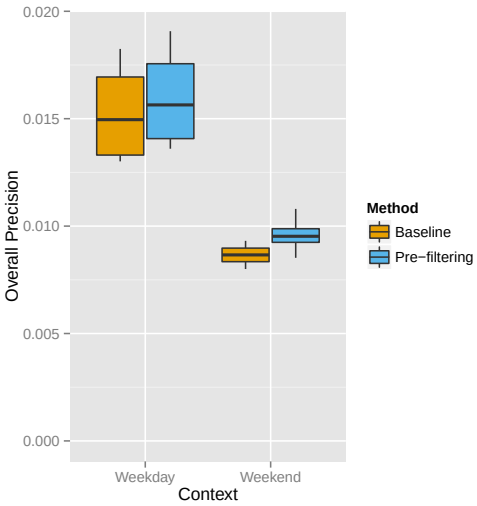
It is also interesting to observe the case of the popularity baseline for the big datasets, shown in Figures 5.17 and 5.18. We see that the pre-filtering algorithm



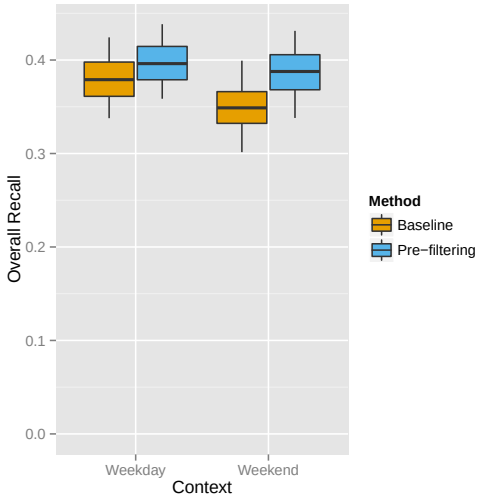
(a) Pre-filtering vs. Item similarity HR@10 (b) Pre-filtering vs. Item similarity MPR

Figure 5.12. Ranking measures for Pre-filtering vs. Item similarity

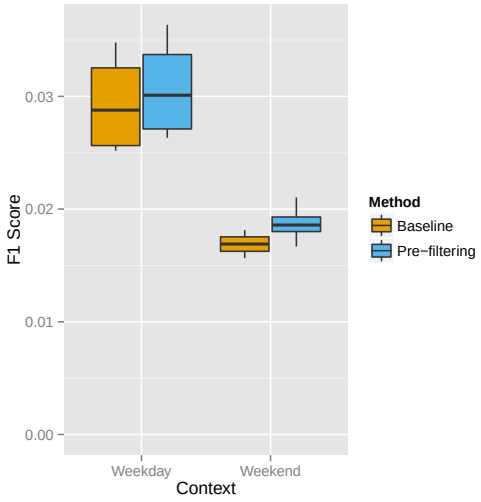
achieves parity with the baseline in most of the measures, in contrast to the much worse performance it had when we examined all the datasets. The cause for this change is the fact that for a large enough dataset, the items that are popular in the contextual datasets will tend to converge to the overall most popular items, hence the recommendation lists for the two methods, baseline popularity and pre-filtering, will be pretty much identical, since the length of the recommendation list we use will only cover the 300 most popular items anyway. In a sense we reach the limit of the popularity baseline's performance for pre-filtering, and we cannot do any better than the baseline when examining using the popularity as the input algorithm.



(a) Pre-filtering vs. Item similarity Precision

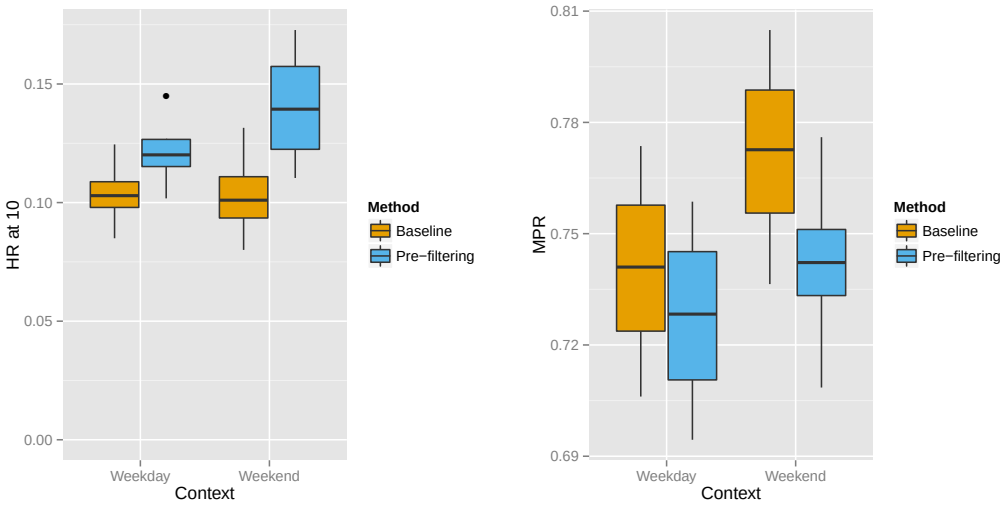


(b) Pre-filtering vs. Item similarity Recall



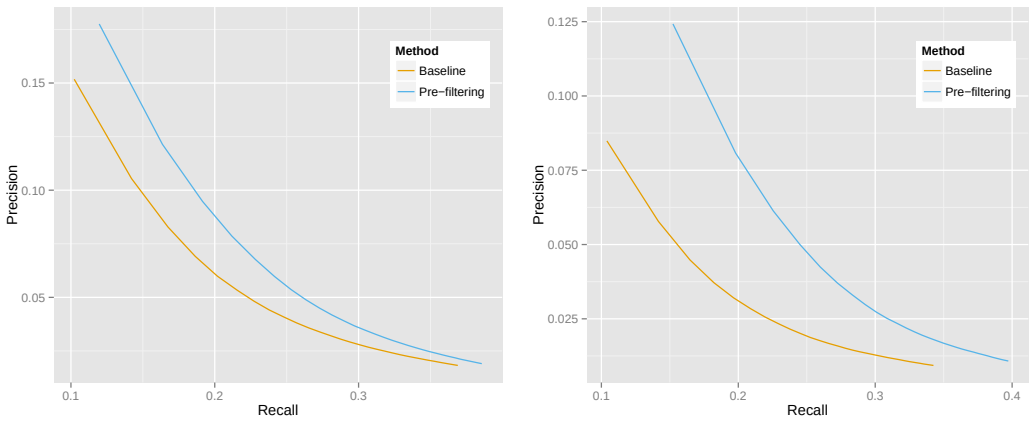
(c) Pre-filtering vs. Item similarity F_1-Score

Figure 5.13. Precision-recall measures for Pre-filtering vs. Item similarity, using datasets at least 6 months long



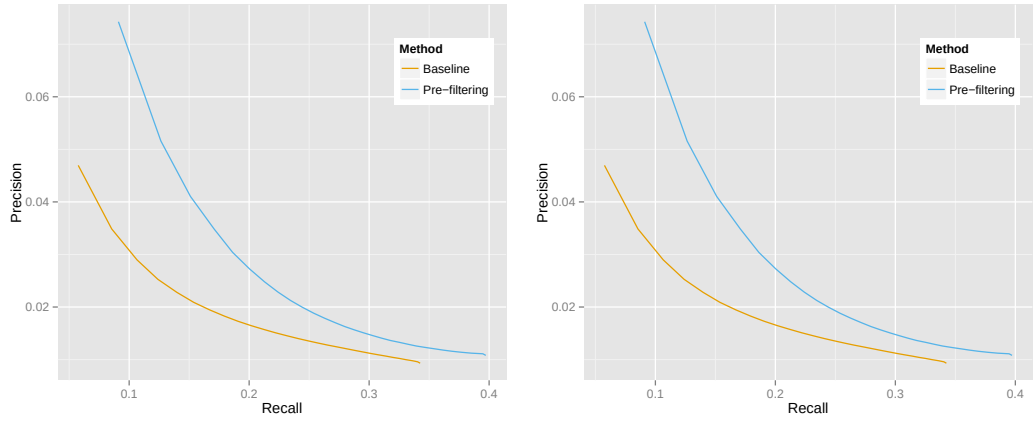
(a) Pre-filtering vs. Item similarity HR@10 (b) Pre-filtering vs. Item similarity MPR

Figure 5.14. Ranking measures for Pre-filtering vs. Item similarity, using datasets at least 6 months long



(a) Customer PR Curve Weekday (b) Customer PR Curve Weekend

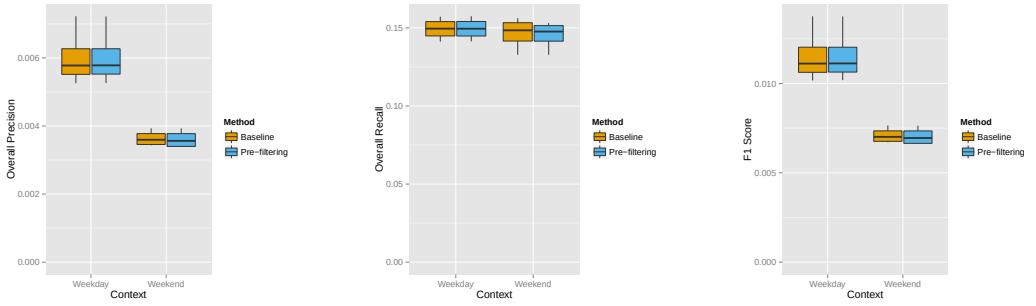
Figure 5.15. Customer PR curves for item similarity pre-filtering, 1 year set



(a) Global PR Curve Weekday

(b) Global PR Curve Weekend

Figure 5.16. Global PR curves for item similarity pre-filtering, 1 year set

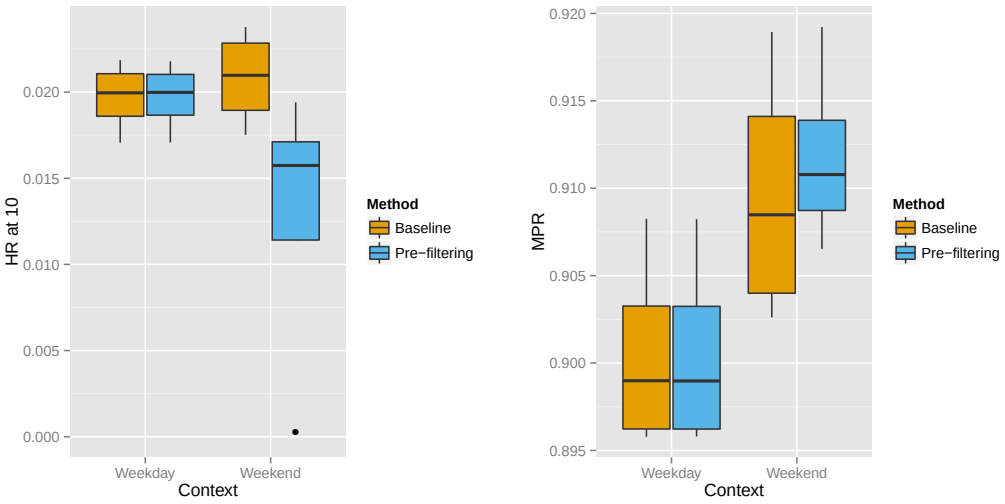


(a) Pre-filtering vs. Popularity Precision

(b) Pre-filtering vs. Popularity Recall

(c) Pre-filtering vs. Popularity F_1-Score

Figure 5.17. Precision-recall measures for Pre-filtering vs. Popularity, using datasets at least 6 months long



(a) Pre-filtering vs. Popularity HR@10

(b) Pre-filtering vs. Popularity MPR

Figure 5.18. Ranking measures for Pre-filtering vs. Popularity, using datasets at least 6 months long

Chapter 6

Conclusion

In this chapter we will provide our closing remarks for the work performed during this thesis, as well as directions for future work.

6.1 Discussion of results

Recommender systems have now become prevalent in a number of online applications and their importance in driving user satisfaction and sales has been established. Despite an increased interest on research in the field sparked by the Netflix Prize, context-aware recommendation systems are still an area that has not been explored thoroughly. This project's aim was to investigate the performance of context-aware recommendation methods on implicit user data, making use of already established collaborative filtering algorithms.

The post-filtering algorithm developed performed worse than the baseline, where the sparsity of the data proved to be a major challenge for the performance of the algorithm. Given the complexity of the steps involved in the algorithm, significant changes would have to be made to the algorithm in order to provide higher quality recommendations while ensuring that the algorithm also scales to handle millions of users and items.

The pre-filtering algorithm provided some more challenges in setting up the experiments properly. Its performance proved to be better than the item similarity baseline when a large enough dataset was used as input, which indicates that further investigation is warranted for this algorithm. The fact that this algorithm can use any already established algorithm as input and that we don't need to set any parameters apart from the choice of contextual dimensions make it an attractive option.

However we must be aware of the potential problems that could arise if we introduce too many contextual dimensions, which would split the complete dataset into segments that are too small to generate quality recommendations. The fact

that different recommenders have to be created and maintained for every contextual variable is also a concern, as such a process can be very expensive when we have to provide recommendations to millions of users for millions of items. Running live tests with the algorithm on real users to gather feedback on the utility of the recommendations to users is also necessary to determine its viability as an alternative to using the baseline algorithms.

6.2 Future work

Since the purpose of this thesis was to perform an exploration on the techniques available in the field, there are many aspects of CARS that we did not explore.

The most obvious one would be trying out the same algorithms using contextual variables other than the weekend-weekday split. While time is a contextual variable that can have semantic meaning and can be easily obtained, there are other contextual variables that we can obtain from users that have a semantic meaning, such as the platform being used, or maybe pre-filtering on demographic data that would be good candidates to examine. It would be interesting then to check the performance of the algorithms developed using different contextual variables, and perhaps splitting the data using more dimensions in order to observe the effect the additional specicifity can have when counteracted by the more severe reduction in data size. In order to verify the strength of the positive results we obtained for pre-filtering, we could also perform experiments where we use a more advanced baseline technique, and run the algorithm on all the available items.

Apart from additional experiments on the same techniques, exploration of other context-aware techniques would be of great interest. In particular, testing the performance of contextual modeling algorithms using the dataset available at Spotify would provide valuable insights into the performance of the algorithms, especially testing the scalability of the algorithms as they are tasked to handle millions of users and items. As we discussed in Section 3.3, topic modeling techniques can be used for the personalization of recommendations, and even though their performance individually may be worse that the best performing CF algorithms, contextual algorithms can also provide value as one of the features in an ensemble method.

Another interesting related field is context inference. The inference of high level concepts about users, like their mood or their social situation from implicit data can be very important in driving recommendations. Having such contextual variables available can be valuable as input for contextual recommendation algorithms. With the wide adoption of mobile devices with a multitude of sensors available, we now have a number of sources that we can use in order to infer the context of the user.

Bibliography

[1] Agnar Aamod and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.

[2] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, HUC '99, pages 304–307, London, UK, UK, 1999. Springer-Verlag.

[3] Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen, and Alexander Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Trans. Inf. Syst.*, 23(1):103–145, January 2005.

[4] Gediminas Adomavicius, Ramesh Sankaranarayanan, Shahana Sen, and Alexander Tuzhilin. Incorporating contextual information in recommender systems using a multidimensional approach. *ACM Transactions on Information Systems (TOIS)*, 23(1):103–145, 2005.

[5] Gediminas Adomavicius and Alexander Tuzhilin. Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer, 2011.

[6] Chris Anderson. The long tail. *Wired magazine*, 12(10):170–177, 2004.

[7] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[8] Linas Baltrunas and Xavier Amatriain. Towards time-dependant recommendation based on implicit feedback. In *Workshop on context-aware recommender systems (CARS 2009) in ACM Recsys 2009*, 2009.

[9] Linas Baltrunas, Marius Kaminskas, Bernd Ludwig, Omar Moling, Francesco Ricci, Aykan Aydin, Karl-Heinz Lüke, and Roland Schwaiger. Incarmusic: Context-aware music recommendations in a car. In Christian Huemer and

Thomas Setzer, editors, *E-Commerce and Web Technologies*, volume 85 of *Lecture Notes in Business Information Processing*, pages 89–100. Springer Berlin Heidelberg, 2011.

[10] Aaron Beach, Mike Gartrell, Xinyu Xing, Richard Han, Qin Lv, Shivakant Mishra, and Karim Seada. Fusing mobile, sensor, and social data to fully enable context-aware computing. In *Proceedings of the Eleventh Workshop on Mobile Computing Systems & Applications*, HotMobile '10, pages 60–65, New York, NY, USA, 2010. ACM.

[11] 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, Oct 2007.

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

[13] Thierry Bertin-Mahieux, Douglas Eck, and Michael Mandel. Automatic tagging of audio: The state-of-the-art. *Machine audition: Principles, algorithms and systems*, pages 334–352, 2011.

[14] 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.

[15] David M. Blei. Probabilistic topic models. *Commun. ACM*, 55(4):77–84, April 2012.

[16] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

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

[18] Pedro G Campos, Fernando Díez, and Iván Cantador. Time-aware recommender systems: a comprehensive survey and analysis of existing evaluation protocols. *User Modeling and User-Adapted Interaction*, pages 1–53, 2013.

[19] George Casella and Edward I. George. Explaining the gibbs sampler. *The American Statistician*, 46(3):167–174, 1992.

[20] Toni Cebrián, Marc Planaguma, Paulo Villegas, and Xavier Amatriain. Music recommendations with temporal context awareness. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 349–352. ACM, 2010.

[21] Oscar Celma. *Music recommendation and discovery: : The long tail, long fail, and long play in the digital music space*. Springer, 2010.

[22] Benjamin Fields. *Contextualize your listening: the playlist as recommendation engine*. PhD thesis, Department of Computing Goldsmiths, University of London, 2011.

[23] Zeno Gantner, Steffen Rendle, and Lars Schmidt-Thieme. Factorization models for context-/time-aware movie recommendations. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 14–19, New York, NY, USA, 2010. ACM.

[24] Michele Gorgoglione, Umberto Panniello, and Alexander Tuzhilin. The effect of context-aware recommendations on customer purchasing behavior and trust. In *Proceedings of the fifth ACM conference on Recommender systems*, pages 85–92. ACM, 2011.

[25] Laura A. Granka, Thorsten Joachims, and Geri Gay. Eye-tracking analysis of user behavior in www search. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '04, pages 478–479, New York, NY, USA, 2004. ACM.

[26] Asela Gunawardana and Guy Shani. A survey of accuracy evaluation metrics of recommendation tasks. *J. Mach. Learn. Res.*, 10:2935–2962, December 2009.

[27] Negar Hariri, Bamshad Mobasher, and Robin Burke. Context-aware music recommendation based on latent topic sequential patterns. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 131–138. ACM, 2012.

[28] Negar Hariri, Bamshad Mobasher, and Robin Burke. Query-driven context aware recommendation. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 9–16. ACM, 2013.

[29] Jonathan L Herlocker, Joseph A Konstan, Loren G Terveen, and John T Riedl. Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53, 2004.

[30] Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22Nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '99, pages 50–57, New York, NY, USA, 1999. ACM.

[31] Thomas Hofmann. Latent semantic models for collaborative filtering. *ACM Trans. Inf. Syst.*, 22(1):89–115, January 2004.

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

[33] Alexandros Karatzoglou, Xavier Amatriain, Linas Baltrunas, and Nuria Oliver. Multiverse recommendation: N-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 79–86, New York, NY, USA, 2010. ACM.

[34] 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.

[35] Yehuda Koren. Factorization meets the neighborhood: A multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 426–434, New York, NY, USA, 2008. ACM.

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

[37] Shyong K. Lam and John Riedl. Shilling recommender systems for fun and profit. In *Proceedings of the 13th International Conference on World Wide Web*, WWW '04, pages 393–402, New York, NY, USA, 2004. ACM.

[38] Dongjoo Lee, SungEun Park, Minsuk Kahng, Sangkeun Lee, and Sang-goo Lee. Exploiting contextual information from event logs for personalized recommendation. In Roger Lee, editor, *Computer and Information Science 2010*, volume 317 of *Studies in Computational Intelligence*, pages 121–139. Springer Berlin Heidelberg, 2010.

[39] JaeSik Lee and JinChun Lee. Context awareness by case-based reasoning in a music recommendation system. In Haruhisa Ichikawa, We-Duke Cho, Ichiro Satoh, and HeeYong Youn, editors, *Ubiquitous Computing Systems*, volume 4836 of *Lecture Notes in Computer Science*, pages 45–58. Springer Berlin Heidelberg, 2007.

[40] Daniel Lemire and Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. In *SDM*, volume 5, pages 1–5. SIAM, 2005.

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

[42] Beth Logan. Mel frequency cepstral coefficients for music modeling. In *In International Symposium on Music Information Retrieval*, 2000.

[43] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Graphlab: A new framework for parallel machine learning. *CoRR*, abs/1006.4990, 2010.

[44] Lie Lu, D. Liu, and Hong-Jiang Zhang. Automatic mood detection and tracking of music audio signals. *Trans. Audio, Speech and Lang. Proc.*, 14(1):5–18, December 2006.

[45] 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.

[46] Prem Melville, Raymod J. Mooney, and Ramadass Nagarajan. Content-boosted collaborative filtering for improved recommendations. In *Eighteenth National Conference on Artificial Intelligence*, pages 187–192, Menlo Park, CA, USA, 2002. American Association for Artificial Intelligence.

[47] Cosimo Palmisano, Alexander Tuzhilin, and Michele Gorgoglione. Using context to improve predictive modeling of customers in personalization applications. *Knowledge and Data Engineering, IEEE Transactions on*, 20(11):1535–1549, 2008.

[48] Umberto Panniello, Alexander Tuzhilin, Michele Gorgoglione, Cosimo Palmisano, and Anto Pedone. Experimental comparison of pre- vs. post-filtering approaches in context-aware recommender systems. In *Proceedings of the Third ACM conference on Recommender systems*, pages 265–268. ACM, 2009.

[49] Han-Saem Park, Ji-Oh Yoo, and Sung-Bae Cho. A context-aware music recommendation system using fuzzy bayesian networks with utility theory. In Lipo Wang, Licheng Jiao, Guanming Shi, Xue Li, and Jing Liu, editors, *Fuzzy Systems and Knowledge Discovery*, volume 4223 of *Lecture Notes in Computer Science*, pages 970–979. Springer Berlin Heidelberg, 2006.

[50] István Pilászy, Dávid Zibriczky, and Domonkos Tikk. Fast als-based matrix factorization for explicit and implicit feedback datasets. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, pages 71–78, New York, NY, USA, 2010. ACM.

[51] Martin Piotte and Martin Chabbert. The pragmatic theory solution to the netflix grand prize. *Netflix prize documentation*, 2009.

[52] Pearl Pu, Li Chen, and Rong Hu. Evaluating recommender systems from the user's perspective: survey of the state of the art. *User Modeling and User-Adapted Interaction*, 22(4-5):317–355, 2012.

[53] Steffen Rendle and Lars Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, pages 81–90, New York, NY, USA, 2010. ACM.

[54] Paul Resnick, Neophytos Iacovou, Mitesh Suchak, Peter Bergstrom, and John Riedl. Grouplens: An open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM Conference on Computer Supported Cooperative Work*, CSCW '94, pages 175–186, New York, NY, USA, 1994. ACM.

[55] Alan Said, Shlomo Berkovsky, and Ernesto W. De Luca. Putting things in context: Challenge on context-aware movie recommendation. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, pages 2–6, New York, NY, USA, 2010. ACM.

[56] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM.

[57] Badrul M Sarwar, George Karypis, Joseph A Konstan, and John T Riedl. Application of dimensionality reduction in recommender system–a case study. In *Proceedings of the ACM WebKDD 2000 Web Mining for E-Commerce Workshop*, New York, NY, USA, 2000. ACM.

[58] Markus Schedl, Arthur Flexer, and Julián Urbano. The neglected user in music information retrieval research. *Journal of Intelligent Information Systems*, 2013.

[59] 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*, SIGIR '02, pages 253–260, New York, NY, USA, 2002. ACM.

[60] Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Adv. in Artif. Intell.*, 2009:4:2–4:2, January 2009.

[61] Auke Tellegen, David Watson, and Lee Anna Clark. On the dimensional and hierarchical structure of affect. *Psychological Science*, 10(4):297–303, 1999.

[62] Andreas Töscher, Michael Jahrer, and Robert M Bell. The bigchaos solution to the netflix grand prize. *Netflix prize documentation*, 2009.

[63] K. Trohidis, G. Tsoumakas, G. Kalliris, and I. Vlahavas. Multilabel Classification of Music into Emotions. In *Proc. 9th International Conference on Music Information Retrieval (ISMIR 2008), Philadelphia, PA, USA, 2008*, 2008.

[64] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13, 2007.

[65] George Tzanetakis. *Manipulation, analysis and retrieval systems for audio signals*. PhD thesis, 2002.

[66] George Tzanetakis and Perry Cook. Marsyas: a framework for audio analysis. *Organised Sound*, 4:169–175, 12 2000.

[67] Lyle H Ungar and Dean P Foster. Clustering methods for collaborative filtering. In *AAAI Workshop on Recommendation Systems*, volume 1, 1998.

[68] Aaron van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2643–2651. 2013.

[69] John Winn, Christopher M Bishop, and Tommi Jaakkola. Variational message passing. *Journal of Machine Learning Research*, 6(4), 2005.

[70] Hsiang-Fu Yu, Cho-Jui Hsieh, Si Si, and Inderjit S Dhillon. Scalable coordinate descent approaches to parallel matrix factorization for recommender systems. In *ICDM*, pages 765–774. IEEE, 2012.

[71] K. Yu, A. Schwaighofer, V. Tresp, Xiaowei Xu, and H.-P. Kriegel. Probabilistic memory-based collaborative filtering. *Knowledge and Data Engineering, IEEE Transactions on*, 16(1):56–69, Jan 2004.

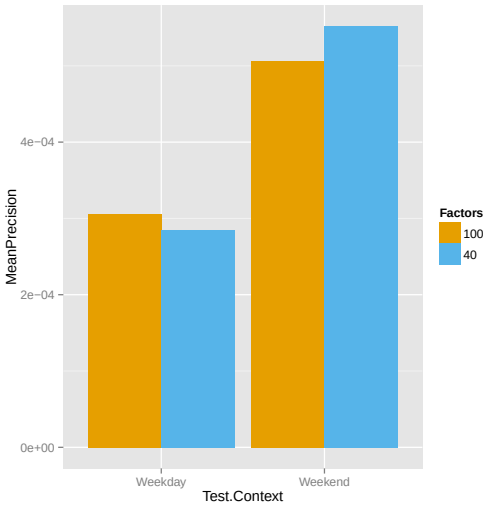
[72] Z. Zaier, R. Godin, and L. Faucher. Evaluating recommender systems. In *Automated solutions for Cross Media Content and Multi-channel Distribution, 2008. AXMEDIS '08. International Conference on*, pages 211–217, Nov 2008.

Appendix A

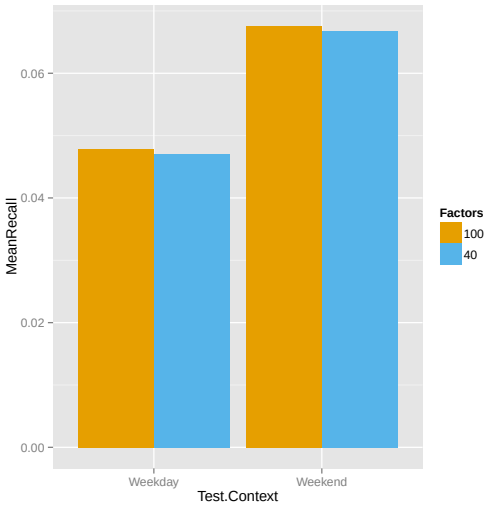
Plots

A.1 Parameter selection experiments

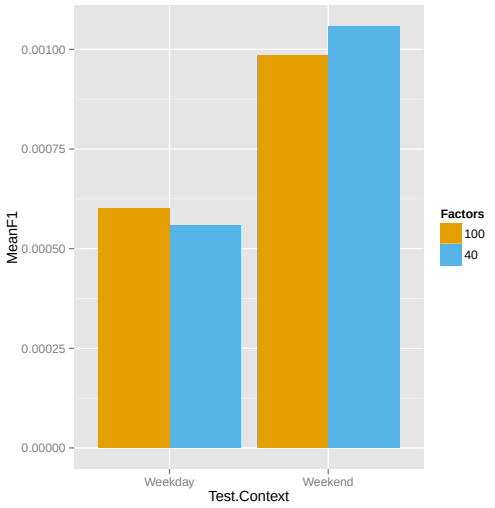
A.1.1 Selection of number of factors for CCD++



(a) 100 vs 40 Factors Precision

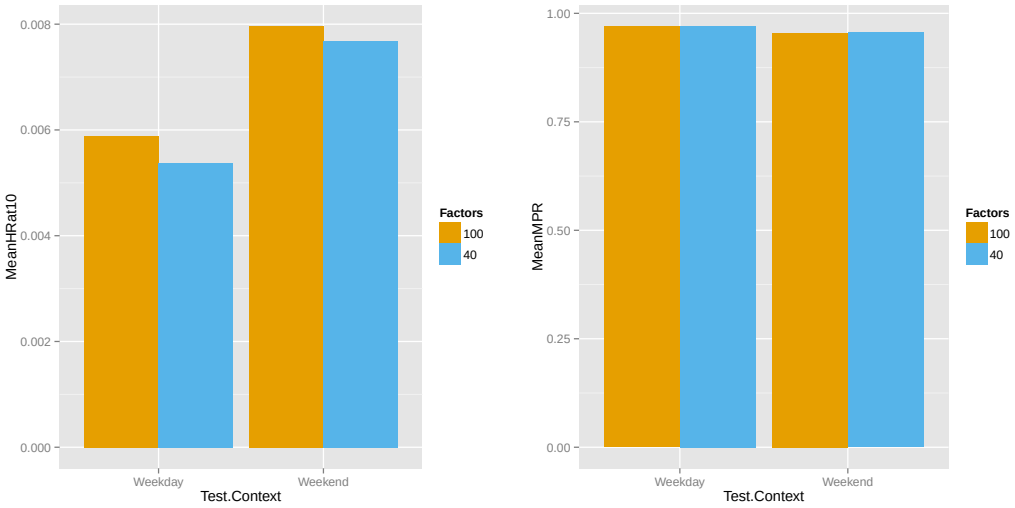


(b) 100 vs 40 Factors Recall



(c) 100 vs 40 Factors F_1-Score

Figure A.1. Effect of number of factors used in CCD++ on post-filtering, for precision-recall measures

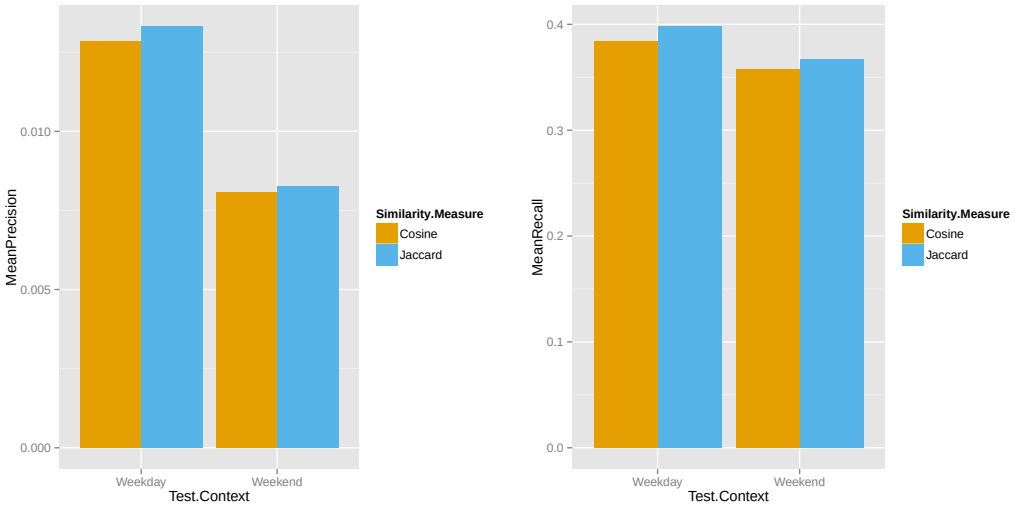


(a) 100 vs 40 Factors HR@10

(b) 100 vs 40 Factors MPR

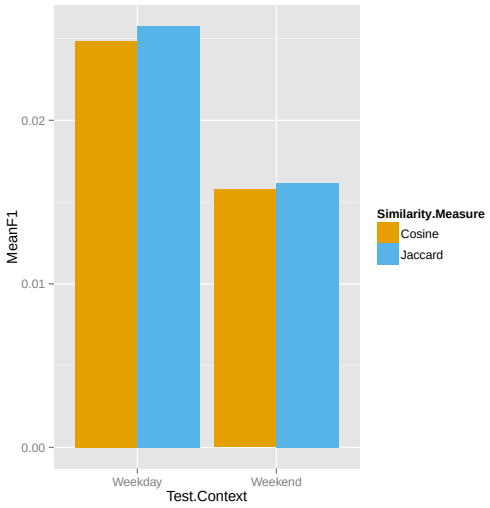
Figure A.2. Effect of number of factors used in CCD++ on post-filtering, for ranking measures

A.1.2 Selection of similarity measure for pre-filtering



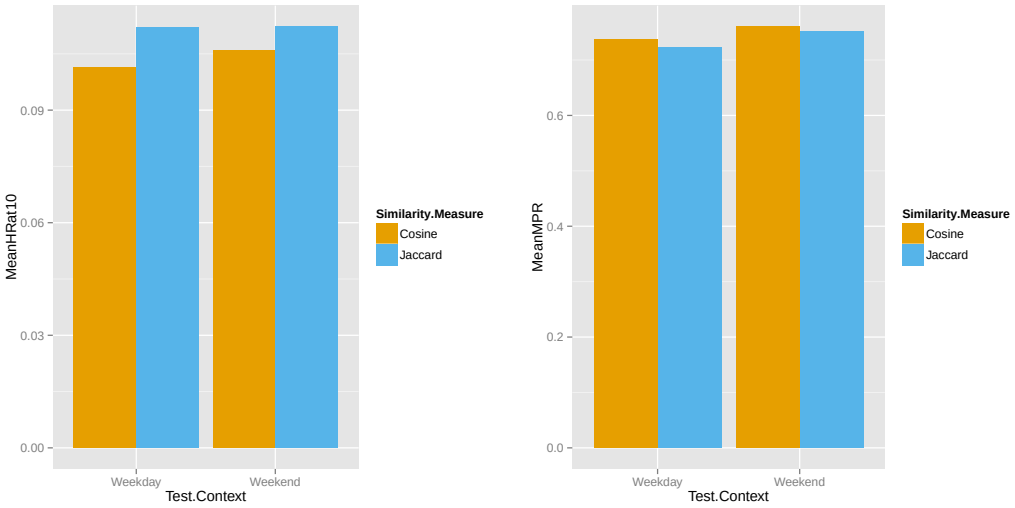
(a) Jaccard vs. Cosine Precision

(b) Jaccard vs. Cosine Recall



(c) Jaccard vs. Cosine F_1-Score

Figure A.3. Effect of similarity measure on pre-filtering, for precision-recall measures



(a) Jaccard vs. Cosine HR@10 (b) Jaccard vs. Cosine MPR

Figure A.4. Effect of similarity measure on pre-filtering, for ranking measures