Recommender Systems for e-commerce websites

An analysis of the MyMediaLite recommender library

Masterproef voorgedragen tot het bekomen van de graad van
Master’s Dissertation submitted to obtain the degree of

Master of Science in Business Administration

Jonathan Henskens
Under the guidance of
Prof. Els Clarysse
Recommender Systems for e-commerce websites

An analysis of the MyMediaLite recommender library

Masterproef voorgedragen tot het bekomen van de graad van
Master’s Dissertation submitted to obtain the degree of

Master of Science in Business Administration

Jonathan Henskens
Under the guidance of
Prof. Els Clarysse
Confidentiality agreement:

PERMISSION

I declare that the content of this Master’s Dissertation can be consulted and/or reproduced if the sources are mentioned.

Name student: Jonathan Henskens

Signature:
Dutch Abstract

In de huidige economische omgeving worden er meer en meer aankopen gedaan langs het internet. Echter door het internet is het aanbod aan goederen en diensten dat bedrijven aanbieden aanzienlijk gestegen en is het vaak moeilijk voor klanten om goede beslissingen te maken. Daarom werd een hulpmiddel ontwikkeld die de klanten zou kunnen sturen in hun keuzes door aanbevelingen te maken voor producten op basis van onderliggende gegevens, genaamd Recommender Systems. Aangezien deze systemen bestaan in vele vormen is er geen consensus over welke onderliggende algoritmes nu het best presteren. Daarom wordt in dit onderzoek een framework voorgesteld, MyMediaLite, die een veelheid aan algoritmen aanbiedt. Er zal worden nagegaan welke onderliggende algoritmen het meest gebruikt worden, alsook de voor- en nadelen ervan. Hieruit zal dan een analyse volgen van de beschikbare algoritmen binnen het MyMediaLite-framework, alsook een analyse van de bijbehorende andere functies die het framework biedt. Hieruit zal dan worden afgeleid of het MyMediaLite-framework voldoende is voor het toepassen van Recommender Systems in de context van e-commerce, meer bepaald het gebruik van deze systemen voor het aanbevelen van producten op e-commerce websites.
Preface

To finish my degree in Business Administration: Management and IT I had to write a thesis. This was a very challenging task, as the content was not the easiest without a lot of statistical knowledge. However I did learn a lot during the process of researching the different recommender algorithms.

For this I want to thank my promoter Els Clarysse for giving me the opportunity to get a more deep knowledge of this subject. I have always been interested in the underlying processes of recommender systems and through this thesis I feel like I can now say that I understand how these work. This will now most certainly result in thinking about all the available algorithms, when browsing on amazon.

I also want to thank my mother Marijke Fortuin for supporting me during this process. Furthermore I want to thank my friends for supporting me as well and for being there when I needed a well-deserved break.

Finally I want to thank Zeno Gantner for helping me figure out how I could use the MyMediaLite framework, as I struggled for a while to access it. Yet once he explained it to me it was clear to me how easy it actually was.
Table of Contents

List of abbreviations ............................................................................................................ 1
List of tables and figures ...................................................................................................... 1
Introduction .......................................................................................................................... 2
  Relevance of Recommender Systems in e-commerce ......................................................... 3
    Direct and Indirect effect .................................................................................................. 6
    Direct effect ..................................................................................................................... 6
    Indirect effect .................................................................................................................. 8
Recommender Systems: the concept .................................................................................... 11
Origin .................................................................................................................................. 11
Classification ....................................................................................................................... 12
  Content-based recommendations ...................................................................................... 13
  Collaborative-based Recommendations ........................................................................... 15
  Memory-based collaborative recommendations ............................................................... 16
  Model-based collaborative recommendations ................................................................... 18
    k-Nearest-Neighbor clustering ....................................................................................... 19
  SLIM: Sparse Linear Method ............................................................................................ 22
    Matrix Factorization ........................................................................................................ 24
Hybrid recommendations ...................................................................................................... 33
  Independent Combination of outputs ............................................................................... 33
  Incorporation of content-based characteristics in a collaborative recommender system .................................................................................................................. 34
  Incorporation of collaborative characteristics in a content-based recommender system .................................................................................................................. 35
    A unified model ............................................................................................................... 35
Recent developments ............................................................................................................ 38
MyMediaLite ........................................................................................................................ 40
  History ............................................................................................................................... 41
  Open Source ..................................................................................................................... 43
File Format ........................................................................................................................... 44
  Rating Files ....................................................................................................................... 45
  Positive-only feedback files ............................................................................................. 45
  Attribute files .................................................................................................................... 46
User and Item lists .............................................................................................................. 46
Item-recommendation files ................................................................................................ 47
Application .......................................................................................................................... 48
BPR .................................................................................................................................... 50
Recommender System library: Algorithms ................................................................. 56
Item recommendation tool ............................................................................................... 56
  Baseline Algorithms ....................................................................................................... 58
  Matrix Factorization algorithms ................................................................................. 59
  K-Nearest-Neighbor clustering algorithms (k-NN) ................................................... 63
Sparse Linear Methods (SLIM) ....................................................................................... 64
External Item Recommender ............................................................................................ 66
Summary ............................................................................................................................. 66
Rating prediction tool ...................................................................................................... 69
Additional features .......................................................................................................... 70
Iterations ............................................................................................................................. 70
Recommender-options ...................................................................................................... 71
Candidate items ................................................................................................................ 71
Evaluation methods ......................................................................................................... 72
Dataset ............................................................................................................................... 73
Conclusion ........................................................................................................................ 74
Bibliography ..................................................................................................................... 76
Articles ................................................................................................................................. 76
Websites ............................................................................................................................. 81
Figures (in order of appearance) ..................................................................................... 83
List of abbreviations

<table>
<thead>
<tr>
<th>Abreviation</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CB</td>
<td>Content-based</td>
</tr>
<tr>
<td>CF</td>
<td>Collaborative Filtering</td>
</tr>
<tr>
<td>MF</td>
<td>Matrix Factorization</td>
</tr>
<tr>
<td>kNN</td>
<td>k-Nearest-Neighbor</td>
</tr>
<tr>
<td>BPR</td>
<td>Bayesian Personalized Ranking</td>
</tr>
<tr>
<td>SLIM</td>
<td>Sparse Linear Methods</td>
</tr>
<tr>
<td>AUC</td>
<td>Area Under the Curve</td>
</tr>
<tr>
<td>PITF</td>
<td>Pairwise Interaction Tensor Factorization</td>
</tr>
<tr>
<td>LARS</td>
<td>Location-Aware Recommender System</td>
</tr>
<tr>
<td>BPR-Opt</td>
<td>Bayesian Personalized Ranking Optimization criterion</td>
</tr>
<tr>
<td>MML</td>
<td>MyMediaLite</td>
</tr>
<tr>
<td>MAE</td>
<td>Mean Average Error</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>GPU</td>
<td>General Public License</td>
</tr>
<tr>
<td>.csv</td>
<td>Comma-Separated values</td>
</tr>
<tr>
<td>.tsv</td>
<td>Tab-Separated values</td>
</tr>
</tbody>
</table>

List of tables and figures

Table 1: Comparison of algorithms.................................................................67

Figure 1: Penetration in-store and checkout recommender systems........................6
Figure 2: Direct revenue effect of in-store and checkout recommender systems.........7
Figure 3 Direct and Indirect effect of recommender systems on revenue ...............8
Figure 4 Singular Value Decomposition matrices..............................................31
Figure 5: Comparison of CB, CF and Hybrid recommenders ..................................37
Figure 6: Conceptual MyMedia framework.........................................................42
Figure 7: Rating data with date and time stamp..................................................45
Figure 8: Feedback Files (comma-separated).....................................................46
Figure 9: Attribute files (tab-separated)..........................................................46
Figure 10: user list...............................................................................................47
Figure 11: Item-recommendation file.................................................................47
Figure 12: Common approach to missing values....................................................51
Figure 13: BPR approach to missing values ..........................................................52
Figure 14: learnBPR procedure..............................................................................54
Figure 15: Convergence BPR vs stochastic gradient descent ..................................55
Figure 16: Comparison of different CF algorithms ..............................................56
Figure 17: Standard parameters for k-NN clustering ...........................................57
Introduction
In today’s economic landscape e-commerce has gained a lot of importance over the past 10 years. In Europe the share of enterprises’ turnover achieved through e-commerce channels has increased from 9% in 2004 to 17% in 2015. In the US this trend of the increasing share of e-commerce in turnover can also be observed. However in the US it is remarkable that the wholesale of products is much more driven by e-commerce than the retail sector. In 2014 the percentage of sales achieved through an e-commerce channel was at 42.5%, while retail sales were at 6.4% of total sales. This implies that in the business-to-business the e-commerce sector is of more importance in the US. However the general trend is the same in all sectors, a steady increase in the share of e-commerce driven turnover. Therefore it is important for businesses to do a greater effort for their e-commerce activities and thus implying the need for proper tools to boost this.

One of these tools that has made its introduction in 1992 is the use of recommender systems to boost sales. The main objective of this tool is to recommend products to customers which they might want to buy, based on their personal characteristics and purchasing history. This can be a recommendation connected to a purchase a customer has just made (cross-selling) or by recommending products based on their similarity to other users. Since e-commerce mostly takes place online on websites this makes it possible to offer a personalized shop to every customer that visits the website, thus increasing the relevance of every product shown on the website. This makes it possible to recommend products that a customer would have never heard of if it wasn’t for the recommendation. In this paper will be analysed what kind of benefits a recommender system can offer to an e-commerce website, focusing on item recommendations. More precisely an answer will be looked after for the question: “Is the MyMediaLite Recommender System Library a sufficient framework for applying recommender systems for item recommendation in an e-commerce website context?”.
Relevance of Recommender Systems in e-commerce

Recommender systems and its applications have been the subject of a lot of research in the past 2 decades. Many new approaches for the recommendation problem have been developed, as businesses are more and more taking part in e-commerce applications to sell their products. Two competitions have also made a big impact on the field of recommender systems, namely the Netflix Prize and the KDD Cup. The Netflix prize challenged people all over the world to improve their movie recommendations with 10% (RMSE) and in return the winning team would receive one million dollars. This motivated a lot of researchers to develop new ways to make recommendations more accurate. However why is a company like Netflix willing to pay one million dollar for an increase in accuracy of 10%?

One trend that can certainly explain part of this gained interest in recommender system, is the trend from mass production to a more personalized approach to marketing. Before products were made exactly the same for all its customers and no or very little personalization was applied to the marketing of its products or to the products themselves. However once the internet made its introduction, people were better informed on what kind of products were available with additional reviews already rating the products. However for customers this new media for buying products was a lot to cope with as the amount of information rose exponentially. The huge amount of information made it so that customers had a lot of difficulties deciding on which products they would buy and were overwhelmed by this decision, which used to be easy. This is where recommender systems came in, these tools provided recommendations for customers, thus filtering the amount of information that they had to process. The benefit of recommender systems was also that it used feedback from other customers to provide more accurate information on the different products. This took away some of the distrust that early adopters of web shops experienced, as products were rated by people that had already bought the product. In this stage however recommender systems mostly recommended items that were liked by its users in general and no personal recommendations were made yet.

In the second stage of the adoption of recommender systems the previously rated items were now used to analyse customer-patterns. These customer-patterns could now be analysed automatically thus enabling systems to be created to incorporate this. Recommender systems could now make recommendations, tailored to the personal needs of each customer. As mentioned in ‘Amazon.com Recommendations Item-to-Item Collaborative Filtering’ (Linden,
Smith, York, 2003), Amazon.com mentioned that in the context of web shops one should have one shop for every customer. This is exactly what the recommender systems try to provide. They analyse user profiles and link it to other customers to recommend certain products which fit into their preferences. Then they are able to display their webpages accordingly, rendering the main page into a page filled with recommendations based on your previously analysed preferences or your similarity with other users. Investing in a recommender system to provide a personally tailored website to its customer often improves revenue through many different effects, so it should be considered by companies that have an online presence.

The effects of incorporating a recommender system on a web shop have a very wide scope as it influences many elements of the customer experience. A first effect that should be considered is the volatility of websites. Most users, when checking a website, scroll over it and don’t pay a lot of attention on it if it does not interest them. This is one thing that a recommender system can improve, as it is able to quantify a person’s interests it is able to only show the items that the user will most likely be interested in. This by itself can already improve the customer’s experience, as a user will be less annoyed by products or articles that he could consider as spam. This however only works when the recommendations are sufficiently accurate, recommending the wrong items renders the website uninteresting for the user, so this should be avoided in all cases.

Another opportunity of recommender systems is to analyse its customers and try to cross-sell its other products based on the interest that other similar users showed. Cross-selling is the act of recommending other products when a user buys a product, thus leading to higher revenue. These recommender systems offer good assistance in this as patterns can be discovered that weren’t clear before. For example Walmart found that people who buy diapers, very often also buy beer (http://www.theregister.co.uk/2006/08/15/beer_diapers/). This is a pattern that is not as obvious and without a recommender system it would have probably not been discovered, but now Walmart is able to use this and recommend beer when people buy diapers. The method of cross-selling is usually used at the end of the purchasing process, thus at the checkout. So it is only a small part of the entire functionalities of recommender systems. However also while browsing through a web page this can be applied as the customer is already showing interests towards certain items then, recommendations in the sidebar can be made according to the shown interest.
In general however recommender systems should concentrate on one certain goal, improving customer loyalty. In the context of e-commerce acquiring loyal customers is one of the most important requirements to attain steady revenue streams. In e-commerce users are able to compare many different products with each other, based on reviews, ratings, product specifications, etc. So many users will make an informed decision, with the available information. This is a big contrast to how business was done before the internet, as customers were mostly limited to a few shops which were nearby, while nowadays customers are able to compare everything everywhere. Therefore it is important for companies that are active in the e-commerce space to offer something greater than just its product, namely a personal shop for every customer. This way a company can include only things that a customer is predicted to like and also include information like reviews and ratings to make it easier for customers to get a good image of a product. By doing this it can achieve a one-stop shop, which includes all the necessary elements to make a buying decision. By improving the experience of a customer in such a way, it can provoke loyalty towards a website as customers grow accustomed to the website and see it as their way to access the internet for certain products. Like for example Amazon, which has become a web shop for basically anything and because of the good application of recommender systems, the customers are able to find the products they are looking for quite easily. This good experience then becomes a standard and customers will compare this with other websites, which shows that it’s of great competitive value to make the websites dynamic so that every user will experience the same website in an entirely different way.

The before mentioned customer loyalty leads in many ways to higher revenues. In ‘Customer loyalty in e-commerce: an exploration of its antecedents and consequences’ (Srinivasan, Anderson, Ponnavolu, 2002) an analysis is made of the influence of e-loyalty of customers on their behaviour. They showed that customer that are loyal to an online website tend to promote this website to other people if their experience was good, which can be classified as an indirect effect of a good website. Also customers tend be willing to pay more for their products if they are loyal, as they feel comfortable buying items there and have trust in their services. However it should be noted that recommender systems only play a small part in this. A lot of this is also dependent on proper website-design, good customer service and user-friendly interfaces, however recommender systems can be a good tool to improve this.
Direct and Indirect effect
The effect of adopting a recommender system in a company’s online presence can be classified into 2 different effects: the direct effect and the indirect effect on revenue. The direct effect consists of all revenue from purchases made in recommended items. The indirect effect is considered as all the revenue that originates from repeat purchases of recommended items and the amount of purchases made in the same category as the recommended items. Also a division can be made in the application of a recommender system: checkout recommenders and in-store recommenders. Checkout recommenders are the applications which recommend other items, just before checking out and going to the payment page. In-store recommenders are the applications that recommend items while browsing through the shop or on the main page. In ‘The Value of Personalised Recommender Systems to E-Business: A Case Study’ (Dias, Locher, Li, El-Deredy, Lisboa, 2008) a case study was done of the web shop LeShop, the number 1 e-grocer in Switzerland. In it they analyse both the direct and indirect of recommender system on revenue and make a comparison of in-store and checkout recommenders.

Direct effect

![Figure 1: Penetration in-store and checkout recommender systems](image)

In their comparison and the graph (Dias, Locher, Li, El-Deredy, Lisboa, 2008, p.3) of the in-store recommender and the checkout recommender they compare it on 2 aspects, penetration and direct extra revenue. Penetration is calculated as the proportion of shoppers that bought at least
one of the recommended items, thus dividing this amount of customers by the amount of customers that have made a purchase from LeShop. Their first finding in this was that regularly updating the recommender model files leads to an increase in penetration. The average increase was at 0.26% for every update. Thus proving the importance of regularly, preferably in real-time, updating the recommender models. Another conclusion that could be made is that once the in-store recommender was introduced its penetration rose very fast, comparing to the checkout recommender. They believe that this is because of the length of exposure, as this is 9 times higher for the in-store recommender. Another thing that can also be observed is that the total penetration, which only counts customers that reacted on a recommendation once, is higher once the in-store recommender is introduced. This implies that users who used the in-store recommendations were users that did not use the checkout recommender before. This shows that different customers have different behaviours and needs towards recommenders depending on where they are presented.

Figure 2: Direct revenue effect of in-store and checkout recommender systems

In their second comparison (Dias, Locher, Li, El-Deredy, Lisboa, 2008, p.3) the direct revenue effect is analysed of both recommender forms, as shown in the graph. The direct effect is calculated as the percentage of total revenue that is acquired through the sales of the recommended items. Again here it’s clear that updating the recommender model data certainly
has an influence on the performance of the recommender systems. However it seems to only have a large positive effect on the in-store recommender. The same observation can be made for the influence on direct extra revenue from recommendations, namely that the in-store recommender’s share in revenue rises very fast once it is introduced. Again indicating that the in-store recommender tends to perform better than the checkout recommender, possibly for the same reason as was mentioned before. A final note on this analysis is that although the maximum share of revenue that this direct effect attains is lower than 0.30%, the amounts of revenue are substantial when it is analysed in its proper context. An increase of 0.30% seems rather low, however taking into account that these kind of e-commerce websites often produce revenues in the millions 0.30% is a decent amount of additional revenue.

**Indirect effect**

The indirect effect of recommender systems is calculated in this case study as the amount of revenue from repeat purchases which the customer was first introduced to by a recommendation. Additionally the revenue from categories to which the customer was first introduced by a recommender system are added. However all revenue from products which were recommended during their purchasing process are left out as this can be classified under the direct effect. This revenue is classified as indirect, because it was caused by a more long term effect of an earlier made recommendation (Dias, Locher, Li, El-Deredy, Lisboa, 2008, p.4):
By analysing this graph it’s clear that it does not take long before the indirect effects outperform the direct effects. At first the direct effects produce more revenue, however this is just because the indirect effects have not yet been triggered. The direct effects also seem rather low until the introduction of the in-store recommender system. It is at this point that the direct effect substantially increases. The indirect effect however stays at a substantial level and even when no updates on the model’s data are performed it remains stable around 0.15% and 0.20%, while the direct effect slowly fades. In general the indirect effect thus makes up for most part of the effect a recommender system has on revenue. The extra gain from the indirect effect adds up to at least 66% or an average of 336% when compared to the direct effect. So it can be concluded that the indirect effect of a recommender system is of more importance to a company’s revenue. In the case study it is also analysed from which categories of products the most additional revenue is obtained and in the case of LeShop this comes from Delicatessen (26.02%), Dairy (19.67%), Fruits & Vegetables (17.12%) and Butcher (9.04%). Noticeably these are all fresh products and according to the authors this could be a sign that recommender systems can help in overcoming the resistance that shoppers experience when it comes to buying fresh products on-line. However this could also be an indication that the customers were not aware that these products were also sold on-line and thus once realizing this repeat purchases were triggered as these are fresh consumption products.

So in conclusion this case study clearly shows that the indirect effects are greater than the direct effects of recommender systems. However the definition that was used in this case study for indirect effect does not include everything. As mentioned before customer loyalty in itself has a large influence on revenue as well. These effects however are not as easily measured in contrast with what the definition used in the case study would suggest. Good recommendations and the eventual satisfaction of a customer’s needs by purchasing and like a product can lead to the customer liking the site more, which can then lead to new customers as he might share his experience with other people (word of mouth). Then the satisfaction in itself might lead to the customer being so convinced of the website that he will also use the website for other purchases, which he hadn’t even considered before. However when these purchases weren’t recommended to him, this will show a loyal character towards the company. These kind of effects have not yet been properly quantified in research as this is hard to identify. Possible approaches to measure
this could include social grouping algorithms, identifying which users were introduced to a website through a previous user.

Recommender systems thus form a great opportunity for e-commerce companies to improve their revenue and service in general. Although its effect is proportionally rather small, in companies with huge revenues the eventual effect is rather substantial. As new research in the context of recommender systems is done and the accuracy of its recommendations increase, this will certainly lead to even better results. However for smaller companies this is also a great opportunity as it can help improve its services, which can lead to a company slowly becoming a standard. However smaller companies will have to analyse whether an investment in a recommender system is worth the cost as it does require maintenance and purchasing costs. One thing that companies that are using recommender systems should keep in mind is their goal. A recommender system can be improved in many different ways, so the goal is most important. Some recommender system try to achieve better service, while other try to recommend items for cross-selling to achieve the highest profit. This goal defines the kind of recommender system needed. A profit driven recommender system for example will recommend products with the highest profit margin, while still taking into account the characteristics of a user. Recommender systems that try to satisfy their customers as much as possible will accord less importance to the profit, but will purely concentrate on recommending items that will satisfy their customers’ needs.
Recommender Systems: the concept

Before analysing the MyMediaLite recommender system library the term ‘Recommender System’ should be explained together with a brief history to get a better understanding.

Origin

Recommender systems originally originate from the ‘Tapestry system’, which was created in 1992 by David Goldberg, David Nichols, Brian M. Oki and Douglas Terry as a system to filter mails based on collaborative filtering. The system achieved this through content-analysis, but more importantly the use of ‘collaborative filtering’, which “simply means that people collaborate to help one another perform filtering by recording their reactions to documents they read.” (1992, Goldberg, Nichols, Oki, Terry). In this ‘Tapestry system’ users could make annotations on mails that were recorded and then later on used to filter the mailing lists based on the interests of its users. The use of annotations implies the use of explicit feedback for recommendations or rating-based recommendations, something that will later on be adjusted by also using implicit feedback.

The tapestry system provided a database of annotations, separate from the database containing the mails. This is similar to the systems that would later on use ratings (yet not limited to) to recommend products, like on Amazon.com. These annotations or ratings were then aggregated to provide recommendations based on the global characteristics of the mails.

Later on in 1997, the term ‘Recommender System’ was coined by Paul Resnick and Hal R. Varian. For 2 reasons. First of recommenders do not explicitly work together since its users often do not know each other. Secondly these systems do not always filter out the items that are not wanted, but also recommend items that might be of interest to the user. At this time recommender systems also adopted the use of implicit feedback for recommendations as well as applying different aggregation techniques to provide recommendations. One example of such a system is Grouplens, a system for collaborative filtering of net-news, which monitored the reading times of users as a form of implicit feedback as well as a numeric rating between 1 and 5 as explicit feedback. Furthermore it also made use of weighted voting to aggregate the different recommendations and used pseudonyms to capture information on its users to respect their privacy. At this time the drawback of the cost was also taken into account, since the development and maintenance of these systems was often very costly it had to be determined whether the benefits actually outweigh the cost.
In general these recommender systems tried to maximize the same function, namely (Adomavicius, 2005, p.2):

\[ \forall c \in C, s'_c = \arg \max u(c, s) \]

\[(s \in S)\]

With \( u \) utility function that measures usefulness of \( s \) to user \( c \) and \( u \) defined as \( u: C \times S \rightarrow R \), where \( R \) is the totally ordered set. For example the set could be ordered by rating. The main problem however with recommender systems was the utility function \( u \) since it is not defined for the entire set of considered items, so these had to be extrapolated. This is where the differences between recommender systems algorithms became clear. As the utility function first had to be defined empirically and then optimized for a certain performance criterion, for example mean square error, which calculates the accuracy of the predictions. So the different recommender systems differ in the way they rate items. Then once the total order of items is obtained a certain amount of items was then selected to be recommended to the user.

**Classification**

In general 3 different kind of recommendations were made (Adomavicius, Tuzhilin, 2005, p.2):

- **Content-based recommendations:**

  Recommendations are made based on the characteristics of the items that the user preferred in the past.

- **Collaborative recommendations:**

  Recommendations are made based on the similarity between the taste of users and not the content of the items themselves. The user will be recommended items that similar users preferred.

- **Hybrid approach:**

  Recommendations are generated based on the similarities between users and the similarities between items as well as their given feedback.
Content-based recommendations

An early application of a content-based recommendation system was the web-based movie recommender INTIMATE (Mak, Koprinska and Poon, 2003). Like many content-based recommender systems it made use of text categorization to find similarities between movies to then recommend movies. First it analysed the short descriptions of movies as taken from IMDB to obtain a feature vector for every movie. It achieved this by automatically analysing the synopsis of the movies and transforming them into features with different weights for every noun, verb and noun phrase, which were calculated based on their importance within the synopsis. This process takes place in 6 different phases.

First of the system starts by pre-processing the synopsis, it removes some words to obtain only the relevant words. For this it uses 2 algorithms: ‘Lovins stemming’ and ‘stop words removal’. Stemming is used to remove the suffix from words to produce word stems. The stop words removal algorithm removes the words that do not offer any information. This is done to reduce the amount of unique words. Secondly the synopsis is turned into a feature vector based on the words that remain. The synopses are represented by nouns-only as this reduces variance. Thirdly the features are selected based on ‘document frequency’, so the features that show up most in the vector features are selected. Then the weights are accorded based on the ‘term frequency-inverted document frequency’ (TF-IDF) (Salton, 1989), which accords higher weights to the words that appear frequently. However these weights are reduced when these terms also appear in many other documents. Finally a decision tree is built on these weighted feature vectors to then be able to apply this to the characteristics of the user. This way the presence of certain feature vectors then determines which node of the tree the user is in and based on this the most relevant document is selected. (Mak, Koprinska and Poon, 2003)

Content-based recommendations often require automatic text processing algorithms, since the content of the items first has to be analysed and transformed in such a way that it is possible to find similarities between items. Therefore the concentration of these kind of recommendations is mostly on these automatic text processing algorithms, however not limited to. Once this is applied the same techniques that are used in other recommendations can be used to find similarities between items. To show how these text processing algorithms work, one of the earliest algorithms, TF-IDF, will be analysed (Salton, 1989).
TF-IDF or ‘term frequency/inverse document frequency measure’ is defined by ‘Term Frequency’ on one side and ‘Inverse document Frequency’ on the other side. Term frequency is defined by dividing the amount of times a keyword $k_j$ appears in a document $d_j$, represented by $f_{i,j}$ and the maximum frequency $f_{z,j}$ over all keywords. Thus calculating the relative frequency of a keyword $k_j$ in a document $d_j$ (Adomavicius, Tuzhilin, 2005, p.3):

$$TF_{i,j} = \frac{f_{i,j}}{\max_z f_{z,j}}$$

However here should be noted that the use of only the term frequency would not give an accurate estimation of the importance of a keyword if they appear in many other documents as well. Therefore the ‘inverse document frequency’ ($IDF_i$) is added to avoid keywords being overvalued. Inverse document frequency is defined as (Adomavicius, Tuzhilin, 2005, p.3):

$$IDF_i = \log \frac{N}{n_i}$$

With $N$ being the total amount of documents that can be recommend and $n_i$ the amount of documents in which the keyword appears.

Once the $TF_{i,j}$ and $IDF_i$ are calculated, these are then multiplied to obtain the weights for every keyword that is used in the document (Adomavicius, Tuzhilin, 2005, p.3):

$$w_{i,j} = TF_{i,j} \times IDF_i$$

From which the feature vector of the document can then be derived as (Adomavicius, Tuzhilin, 2005, p.3):

$$Content(d_j) = (w_{1,j} ... w_{k_j})$$

After this many more algorithms for automatic text processing were created with the same goal, but with reducing the amount of features in a vector or by applying a machine learning algorithm
to find the most efficient way to model the content of a document. However the underlying principle stayed the same. The obtained feature vectors from these kinds of algorithms are then used to recommend a document based on similarity measures, like the cosine similarity measure. For this it first produces a profile for every user based on their historical characteristics like ratings or purchasing behaviour. Then the similarity between this profile (\(\text{ContentBasedProfile}(c)\)) and the feature vector of the currently analysed document is calculated resulting in the following utility function (Adomavicius, Tuzhilin, 2005, p.3):

\[
u(c, s) = \text{score}(\text{ContentBasedProfile}(c), \text{Content}(s))\]

Which are then maximized by the applied recommender system algorithms. Here can be noted that this utility is one of the possible heuristics for recommender system algorithms.

The benefit of using a content-based model proved the importance of content-based recommendations, it was outperformed by many other models when a lot of ratings were available on the movies, however when very few ratings were available. Therefore this kind of recommendations are very important in such cases and in combination with collaborative recommendations, thus forming a hybrid, this could improve a recommender system greatly when analysing items on which little feedback is given. However a downside of this kind of recommendations is that it often only recommend items that are very similar to the ones the users already knows, thus missing the opportunity of recommending new, also available items that the user might also be interested in. Because of this some randomness should always be added to recommender systems.

**Collaborative-based Recommendations**

Next of the collaborative-based recommender system will be analysed. This kind of system uses the similarity between its users to predict what a user will be interested in and not the characteristics of the considered items. These recommendations can be based on both implicit and explicit feedback. Implicit feedback, as the term implies, is feedback that is captured automatically without an effort of the user. Examples of this are the clicks a user makes, the
amount of time a user stays on a page, the act of buying a product, etc. The benefit of this kind of feedback is surely that it is captured automatically and is thus easier to come by than explicit feedback. Explicit feedback is feedback that is given consciously by the user, thus explicitly given by the user. This feedback is usually in the form of ratings, but can also contain reviews and similar forms of feedback.

Most collaborative-based recommender systems use both forms of feedback to predict whether a user will be interested in an item. The ratings or explicit feedback are usually used as the target variable, since the recommender system will estimate how a user will rate the newly proposed item. Thus ranking the different items by their predicted rating. The implicit feedback is usually used as information on the characteristics of the user. By analysing this a recommender system tries to group users together to then provide recommendations on items that other similar users also liked. However implicit feedback can also be used to provide a kind of rating for an item. For example how long a user stays on a page can implicitly mean how much a person liked an item as a user that isn’t interested wouldn’t stay as long on the web page, but then also other things have to be taken into account like whether the user was also active during the time on that page.

In general the collaborative-based recommender can be divided in 2 parts, the grouping of similar users and the aggregation of the different ratings accorded to the items. Many algorithms were created to perform both of these operations and these can be subdivided into 2 classes: memory-based and model-based recommendations.

**Memory-based collaborative recommendations**

In memory-based recommendations ratings are predicted based on the historical ratings given by users on the different items. Thus estimating the unknown rating $r_{cs}$ for user $c$ and item $s$ by aggregating the ratings of other users who are most similar (Adomavicius, Tuzhilin, 2005, p.5):

$$r_{cs} = agg r_{c's} \ (c' \in \hat{C})$$

Where $\hat{C}$ is the set of users that are most similar that have rated the item. The used aggregation function differs between different recommender systems, however an aggregation function that
is used very often is the weighted sum. The weights used for this average are usually based on how similar the user is to the user for whom the recommendation will be made. Next to that a normalization factor is added to provide a more accurate weighted sum (Adomavicius, Tuzhilin, 2005, p.5):

$$ r_{c,s} = k \sum sim(c, c') \times r_{c',s} \ (c' \in \hat{C}) $$

With $k$:

$$ k = \frac{1}{\sum |sim(c, c')|} $$

By doing this the estimated rating is then obtained and calculated for every item. Then these obtained ratings are used to rank these different items and the top items are then selected to be recommended to the user. The similarity measure has many forms, one of which is the cosine measure, which is often used. To calculate this measure the compared users are represented as feature vectors with the ratings given to the different items that have been rated by both users. After which the cosine measure is calculated based on the difference between the 2 vectors (Adomavicius, Tuzhilin, 2005, p.5):

$$ sim(x, y) = \cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{||\vec{x}||_2 \cdot ||\vec{y}||_2} = \frac{\sum r_{x,s} r_{y,s}}{\sqrt{\sum r_{x,s}^2} \cdot \sqrt{\sum r_{y,s}^2}} $$

The before mentioned ratings from implicit feedback can also be applied within this weighted sum framework, however the ratings will then have to be aggregated with different aggregation methods to then result in a scalar rating. These ratings can then be applied to the framework in the exact same way. A note that should also be made is that these memory-based models often don’t perform well on small datasets as they require other users who have rated the same items. An example to solve this is ‘default voting’ (Breese, Heckerman, Kadie, 1998), in which missing ratings are assumed to have a default rating. Also the content-based recommendation
algorithms could be a solution for this. However another kind of recommendations has been proven to outperform this model by using the similarities between users to recommend items.

Model-based collaborative recommendations
In model-based recommendations, recommendations are not made by comparing all previous ratings of the considered user with all the previous ratings of other users. Instead a model is learned from the available ratings with different algorithms and is then used to make predictions. Many different models, like clustering, probability calculations, decision trees, matrix factorization and many more, can be used to model the available data on the different items and users. Often this technique makes use of machine learning algorithms to find relationships within the data to better understand its users and items to then apply this to new users and items. This technique is known to outperform most memory-based algorithms and has shown great potential for future development.

One of the early models that was applied to this kind recommendations is a probabilistic approach to collaborative filtering. In this collaborative filtering unknown ratings are calculated with the following underlying model (Adomavicius, Tuzhilin, 2005, p.6):

$$ r_{cs} = E(r_{cs}) = \sum_{i=0}^{n} i \times Pr(r_{cs} = i | r_{c,s'}, s' \in S_c) $$

In this model, ratings are assumed to be integers between 0 and $n$ and is calculated by the probability that user $c$ will give the same rating for the considered item $s$ as he did for the previously rated item. One way of doing this is by clustering the different users into different groups, i.e. categorizing every user into one class with the most similar characteristics and then calculating the average rating of this class. The amount of classes and the parameters of the defined groups depend on the data as the model is learned from this. However a limitation of this kind of technique is that each user is categorized into one class, however sometimes clustering into different classes could also be useful as the user’s interests might be very variant. For example one could be interested in hiking in their free time, but during work they might be interested in new technologies.
**k-Nearest-Neighbor clustering**

One possible algorithm for identifying similarities is the k-Nearest-Neighbor clustering algorithm. The method of k-Nearest-Neighbors was first introduced by E. Fix and J.L. Hodges in 1951 and has made a huge impact in the field of classification. The basis of this classification method is that new cases will be classified according to their nearest neighbours, the amount dependent on $k$, so that the target variable will be some kind of aggregation of these neighbours. The simplest and most often used method of aggregation is the principle of ‘majority voting’, in which the target variable is estimated based on the frequency of certain outcomes. However some kind of weighting is usually added to improve the accuracy. This classification method comes down to the following nearest-neighbour rule for binary nonparametric regression (for example the binary variable can be whether a user has bought the product or not):

First the 2 different sample outcomes X and Y are combined into a single sample: $Z_1, Z_2, ..., Z_{m+n}$ (with $m$ being the amount of X outcomes and $n$ the amount of Y outcomes). Then an indicator function is formed in the form of (Fix, Hodges, 1951, p.5-6):

$$
\delta_i = \begin{cases} 
1 \text{ if } Z_i \text{ is one of the X's} \\
0 \text{ if } Z_i \text{ is one of the Y's}
\end{cases}
$$

Then $r$ can be estimated based on weighted majority voting:

$$
\hat{r}(z) = \frac{\sum_{j=1}^{m+n} \delta_i K\{d_k^{-1}(z)(z - Z_j)\}}{\sum_{j=1}^{m+n} K\{d_k^{-1}(z)(z - Z_j)\}}
$$

With $\hat{r}(z)$ as the estimated classification of user or case $z$, $d_k(z)$ the distance from $z$ to the $k$th nearest $Z_j$ (neighbour) and $K$ the ‘kernel density function’. The kernel density function reflects how the likelihood of a certain variable, here the classification, to take on a certain value is distributed. This estimation can be seen as a nonparametric binary regression to the class of $z$, normalized by including the distance-based weights.
Although this method has been developed more than 50 years ago, it has been used very often in the context of classification problems, which is also the case in recommender systems. One of the reasons for this is its simplicity and its ability to perform well on large datasets. It should also be noted that this model is often used for binary classifications, however is not limited to. Linear, logistic and other regressions can also be performed by applying these techniques to the chosen nearest neighbours. This offered great opportunities for recommender systems as they could be used to predict a rating that a user would give to an item. Thus recognizing patterns in the data, to make estimations on new instances. However research showed that the k-NN classification could be easily improved by making the distance metric variable, as this was still far from perfect. Therefore ‘distance metric learning’ (Xing, Jordan, Russel, 2003) was developed. This was such an important part for the k-NN clustering since the quality of these kind of algorithms entirely depend on the underlying metrics that reflect the relationships in the data. As clustering is unsupervised it is thus very important that these relationship have meaning, as users or cases could be classified together based on information that is not relevant, resulting in useless classifications. In research is has been shown that even a linear approach to distance learning could improve the k-NN clustering technique greatly. The example of ‘distance metrics learning’ by Xing, Jordan and Russel (2003) resulted in the following (Xing, Jordan, Russel, 2003, p.2):

A set of points is given by \( \{x_i\}_{i=1}^m \subseteq \mathbb{R}^n \) with certain pairs of these items being ‘similar’:

\[
S : (x_i, x_j) \in S \text{ if } x_i \text{ and } x_j \text{ are similar}
\]

Then a distance metric should be learned that calculates the distance between points \( x \) and \( y \) that puts similar points close to each other, a possible distance metric could have the form of (Xing, Jordan, Russel, 2003, p.2):

\[
d(x, y) = d_A(x, y) = \|x - y\|_A = \sqrt{(x - y)^T A(x - y)}
\]

Where \( (x - y) \) will always be positive and if \( A \) is set to the diagonal in the multidimensional space, then by learning this metric different weights will be calculated on the different
dimensions or axes. Thus calculating the minimum over all of the dimensions. Then to minimize the metric a function, like the small square distance should be minimized. However another constraint has to be put in place, namely the sum of all distances (which have to be positive) is greater than or equal to 1, if not minimizing this metric could put all the points of the dataset in the same place, thus collapsing the entire set. This then results in the following optimization problem (Xing, Jordan, Russel, 2003, p.3):

$$\min \sum_{(x_i,x_j) \in S} \| x_i - x_j \|_A^2$$

$$\min \sum_{(x_i,x_j) \in D} \| x_i - x_j \|_A \geq 1$$

With D being a set of pairs of points that are known to be ‘dissimilar’, if this kind of information is available. Analysing the 3 different formulas, i.e. the distance metric and the 2 constraints, it should be noted that all of these are convex. So the optimization problem will also be convex and so a local minimum can be calculated by an algorithm that calculates the local minimum. This algorithm can have many shapes, one example is the Newton-Raphson method, which can be used to learn the local minima of a diagonal $A$. By doing this all the local minima are found and thus resulting in the smallest distances for every point, resulting in a clustered set of points.

This development in the k-NN technique led to many improvements in clustering and thus its applications had big implications for recommender systems. By finding the true nearest-neighbours, the recommendations based on the aggregation of these neighbours led to a big increase in performance. This learning method also shows the importance of machine learning in the context of recommender systems. The general models that were initially created, could now be improved by applying machine learning to optimize the parameters of the models, leading to an increase in performance over many recommender system algorithms.

One thing that should be noted is that clustering can not only be used to predict whether a user will like a certain item. It also possesses the power to cluster the items itself together, placing similar items together. In certain cases this can also be useful to recommending items, as users often like these similar items. However here the problem of too much similarity often comes up, as these similar items don’t differ a lot, users have many different tastes. Thus only recommending items that are similar would not lead to efficient recommendations as this is
rather limited. However in the case of for example tools, it could be useful for example to recommend the products of the same brand, same quality or same price. Therefore it is important in item-clustering that the attributes for clustering are chosen appropriately. Although adding some kind of randomness in methods like these are highly recommended. Using this kind of clustering would however not be classified under collaborative recommendations, but content-based recommendations.

SLIM: Sparse Linear Method
Another approach in model-based collaborative filtering is the Sparse Linear Method algorithms (Ning, Karypis, 2011). This algorithm aggregates from the different purchasing and rating profiles of its users by solving 2 norms for the optimization problem: the $l_1$-norm and the $l_2$-norm. The algorithm calculates a recommender score for every un-purchased or un-rated item through sparse aggregation of items that have been purchased or rated and then selects the top-N recommendation scores for every user. This sparseness can be interpreted as sampling the cases and not aggregating all available purchases or rating. This produces very efficient results, as the model is learned very quickly. This aggregation for a user $u$ of an item $t_j$ is given by Ning and Karypis in ‘SLIM: Sparse Linear Methods’ (2011, p2):

$$\bar{a}_{ij} = a_i^T w_j$$

With $\bar{a}_{ij} = 0$ and $w_j$ a vector of the aggregation coefficients of a size $n$. Therefore the entire model can be presented as (Ning, Karypis, 2011, p.2):

$$\bar{A} = AW$$

With $A$ being the binary user-item matrix with its given feedback (purchases or ratings). $W$ is a $n \times n$ matrix with the different aggregation coefficients, where every $j$-th column corresponds to $w_j$, which was previously mentioned. Each row of $\bar{A}$ corresponds to all of the recommendation scores of all the items for a user $u$ and is presented as $\bar{a}_i^T$. Each row is then calculated by $\bar{a}_i^T = a_i^T W$, so the multiplication of the transposed preferences feature vector of the user and the corresponding aggregation coefficients. Then the scores are sorted from high to low and the top-N items are selected for each user.
So to solve this model the aggregation coefficients of $W$ of size $n \times n$ have to be found, for this it makes of the $A$-matrix, which represents all the observed ratings or purchases of items respectively with its users, its dimensions are of size $m \times n$ (respectively all users $\times$ all items). For this the following optimization problem was created (Ning, Karypis, 2011, p.3):

$$\minimize_{W} \frac{1}{2} \|A - AW\|_F^2 + \frac{\beta}{2} \|W\|_F^2 + \lambda \|W\|_1$$

subject to \{ $W \geq 0$

$$\text{diag}(W) = 0$$

The $\frac{1}{2} \|A - AW\|_F^2$-term here measures how well the model fits to the data, as $AW$ is the estimated matrix and $A$ is the actually observed rating. So it can be viewed as the error rate of the model, therefore it is minimized as the error-rate should be as low as possible. The $\|W\|_F^2$ and $\|W\|_1^2$ term are considered as the regularization terms for the $l_F$-norm and the $l_1$-norm respectively. The $\beta$ and $\lambda$ parameters can be viewed as the severity of regularization that is used to learn the model. The 2 conditions that have to be fulfilled for $W$ make sure that only positive relationships can be learned and avoids that an identical matrix would be found as a solution and thus the same item would be recommended as was bought or rated.

One of the implication of the $l_1$-norm that should be noted is that because of its positive constraint, some relations will be disregarded, thus resulting in no value. This then produces a sparse matrix, which does not contain a value for every case. The $l_F$-norm is used in this framework to measure to model complexity and according to Ning and Karypis this leads to implicitly grouping together correlated items in the solution. To then compute $W$ every column $w_j$ is then minimized separately as these are assumed to be independent (Ning, Karypis, 2011, p.3):

$$\minimize_{w_j} \frac{1}{2} \|a_j - Aw_j\|_2^2 + \frac{\beta}{2} \|w_j\|_2^2 + \lambda \|w_j\|_1$$

subject to \{ $w_j \geq 0$

$$w_{j,j} = 0$$

Thus leading to a $W$-matrix which is optimized for both least squared errors ($l_2$-norm) and the absolute errors ($l_1$-norm). So then once $W$ is known this implicitly means that the users’
purchasing or rating profiles have been figured out, which can then be applied to new items in order to predict a recommendation score.

These kinds of algorithms have been tested and have been proven to perform about as well as the other state-of-the-art top-N recommendation algorithms. Its most important benefits are that it is able to learn the profiles of its users quite rapidly as the negative relations are immediately disregarded and in doing so it does not have to sacrifice a lot of precision.

**Matrix Factorization**

Yet another approach in model-based collaborative recommendations is the use of ‘latent factor models’. In this approach both users’ and items’ characteristics are analysed to create a profile for every user and item. Then the user profiles are compared to other users and based on this recommendations are made. An example of this could be that John likes comedy movies with very little character-depth, then the recommender system will recommend items that fit within this profile. What then actually occurs is that the predicted rating for John of a movie that is in the comedy genre with a lot of character-depth will be lower than the average user.

One of the possible models within latent factor models is the ‘Matrix Factorization method’ (Koren, Bell, Volisnky, 2009). This method was often used in the context of the Netflix contest. In this contest the company challenged people to improve its movie recommendation system’s accuracy with 10% for which the winners would gain 1 million dollar. This contest together with the KDD-contest caused a lot of progress in the field of recommender systems. One of which was the application of Matrix Factorization models and its further development. The benefit of using such models is that it included a lot of implicit feedback of users to provide recommendations. Eventually it proved to outperform the k-NN clustering methods greatly, thus proving its potential. Both their scalability (the ability to perform on every size of dataset) and their accuracy were higher and offered new opportunities to optimize this kind of recommender system.

In general a Matrix Factorization model defines items and users with vectors based on the given rating patterns. It factorizes the main rating matrix into different matrices to better understand the factors that are driving the ratings. Consider a space of dimensionality $f$ so that the user-item interactions are modelled as inner products in that space. Each item $i$ is given by a vector $q_i \in \mathbb{R}^f$ and each user $u$ is given by a vector $p_u \in \mathbb{R}^f$. $q_i$ represents every element item of
an item $i$ by indicating whether an item possesses these elements with a positive or negative measure. For a user $u$, $p_u$ represents the interest a user has in different items, divided in the different factors and showing the influence of each factor on the interest of the user. These are also presented as a positive or negative measure. These 2 vectors are then multiplied in a dot product, thus resulting in the interaction of user $u$ and item $i$, which then results in the predicted rating $\hat{r}_{ui}$ (Koren, Bell, Volinsky, 2009, p.3):

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

The most difficult part of this process is the mapping of the different user and item factor vectors. Once this is done a recommender system can easily compute the predicted ratings for the different item-user pairs, by using the previously mentioned formula.

The mapping of the different factor vectors can be done in many different ways, Singular Value decomposition (another model-based collaborative approach to recommendations) for example requires the full user-item rating matrix. However this is often not possible because of the amount of missing values, therefore different techniques were created to remove these missing values. One technique is called ‘imputation’, which replaces the missing values with estimated values based on the available values by for example the average of the other ratings. This however has been proven to overfit on the available data or be inaccurate. It is because of this that in Matrix Factorization another approach has been used to map this interaction matrix.

The approach in MF models the available values directly and avoids overfitting by regularizing the model. To do this the factor vectors are learned by minimizing the regularized squared error on the available ratings (Koren, Bell, Volinsky, 2009, p.3):

$$\min_{q^*, p^*} \sum_{(u,i) \in K} (r_{ui} - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2)$$
With \( K \) being the set of user-item pairs for which the ratings are known. The second term of the formula is the regularization term, which ensures that the system does not overfit on the available data. Cross-validation (the user of different splits of training set and test set) is often used to avoid this. The \( \lambda \) factor determines to which extent the available ratings are regularized. The regularization avoids overfitting, which implies that the previous ratings are formed into general notions and thus can be used for future predictions.

**Stochastic gradient descent**

There are several ways to minimize the given function for regularized squared errors, one of which is ‘stochastic gradient descent’. This is a learning algorithm that loops through the different ratings in the training set and for every case a prediction is made and then compared to the actual rating to calculate the error (Koren, Bell, Volinsky, 2009, p.4):

\[
e_{ui} \equiv r_{ui} - q_i^T p_u
\]

It then modifies the parameters proportional to \( \gamma \) in the opposite direction of the gradient or derivative in that point (Koren, Bell, Volinsky, 2009, p.4):

\[
q_i \leftarrow q_i + \gamma (e_{ui} \cdot p_u - \lambda \cdot q_i)
\]

\[
p_u \leftarrow p_u + \gamma (e_{ui} \cdot q_i - \lambda \cdot p_u)
\]

This way the optimal parameters are learned as the looping continues, leading to the eventual best set of parameters for every user-item pair. The looping process continues until all cases are treated or when convergence takes place then no more modifications are made to \( q_i \) and \( p_u \).

However because it has to run through all of the possible cases this involves many calculations. In recent research it has been shown that this kind of learning algorithm involves too many calculations to be feasible for large datasets.

**Alternating least squares**

An alternative to the stochastic gradient descent approach is the ‘alternating least squares’ approach. In this approach one of the feature vectors \( (q_i, p_u) \) is fixed. This causes the formula of the regularized least squares to become quadratic and can then be solved to find their
optima’s. It then rotates between fixing the user vector $p_u$ and the item vector $q_i$ by doing this the ‘regularized least squares’ is decreased in every step until no more change is made by fixing one of the vectors (convergence). This method has the benefit, comparing to stochastic gradient descent, that in sparse training sets it does not loop endlessly through all the empty cases, however it starts with a certain value, which reduces the workload of the learning algorithm. In research is also mentioned that because of the constant fixing of one feature vector for a user or an item it can perform its calculations in parallel, which also reduces the running time. (Koren, Bell, Volinsky, 2009, p.4)

Bias

The learning algorithms used for the rating estimations is already one of the reasons for the good performance of matrix factorization models. However one element that has really improved the accuracy of this model is the introduction of biases into the rating estimations. Ratings in general tend to be biases as some users might continuously rate items higher than other users or some products might have a persistent tendency to be rated higher because of the general perception, however not really representing the actual quality. Therefore explaining every tendency through user-item interactions would not be correct, so the bias should be added in the $\hat{r}_{ui} = q_i^T p_u$ formula to properly reflect the different influences. A first-order approximation of this bias is given by Koren, Bell and Volinsky (2009, p.4):

$$b_{ui} = \mu + b_i + b_u$$

With:

- $b_{ui}$ as the total bias in rating
- $\mu$ as the overall average rating
- $b_i$ as the item-bias
- $b_u$ as the users’ bias

As an example John could be rating the movie ‘The Shawshank Redemption’. The average rating for all movies is 3.7, since this movie is better than average it receives an extra 0.5, however John is a critical user who tends to rate movies 0.4 lower than an average user. So the estimate for
'The Shawshank Redemption' will be 3.8 (3.7 + 0.5 + (-0.4)). Applied to the original rating prediction formula this results in (Koren, Bell, Volinsky, 2009, p.4):

\[ \hat{r}_{ui} = \mu + b_i + b_u + q_i^T p_u \]

So it can be observed that the rating consists of the average rating, the bias towards the item, the users' bias and the interaction of the user with the item. However it should be noted that the actual 'matrix factorization' only takes place in the interaction term of the formula. The modified 'regularized least squares' formula or the optimization criterion is then given by (Koren, Bell, Volinsky, 2009, p.4):

\[
\min_{p^*, q^*, b^*} \sum_{(u,i) \in K} (r_{ui} - \mu - b_i - b_u - q_i^T p_u)^2 + \lambda (\|q_i\|^2 + \|p_u\|^2 + b_i^2 + b_u^2)
\]

Other approaches to calculating the bias in matrix factorization are mentioned by Koren (2010), adding factors like the influence of the day, which as he indicated in his paper improves the accuracy of the model. The users’ and items’ biases are then made dynamic, linking them to a certain day and modifying the bias to the characteristics of that certain day.

**Cold start problem**
One problem that occurs when using a Matrix Factorization approach to recommender systems is the ‘cold start problem’. Because of the lack of ratings it is often difficult to come to a good approximation of the ratings, this is called the cold start problem. One way of solving this, is the inclusion of implicit feedback of users and users’ characteristics (demographically). Implicit feedback could be their purchasing history, their browsing behaviour (e.g. whether a user has seen a page or not), whether or not a user has seen a video entirely, ... Users’ characteristics often include age group, gender, income level, zip code, ... These are often incorporated as Boolean variables as this indicates the presence or absence of the different characteristics, however integer variables can also be used.
Let $N(u)$ be considered as the set of items for which a user $u$ has showed an implicit preference and let $A(u)$ be considered as the set of attributes representing the characteristics of user $u$. Both $N(u)$ and $A(u)$ are elements of $\mathbb{R}^f$ ($N(u), A(u) \in \mathbb{R}^f$) and are both considered to be described by Boolean feature vectors. For the implicit feedback some kind of regularization should be added to give a more accurate representation. The implicit feedback can then be represented as (Koren, Bell, Volinsky, 2009, p.5):

$$|N(u)|^{-0.5} \sum_{i \in N(u)} x_i$$

And the users’ characteristics can be described as (Koren, Bell, Volinsky, 2009, p.5):

$$\sum_{i \in A(u)} y_a$$

Then incorporating this in the last mentioned model of matrix factorization this results in (Koren, Bell, Volinsky, 2009, p.5):

$$\hat{r}_{ui} = \mu + b_i + b_u + q_i^T [p_u + |N(u)|^{-0.5} \sum_{i \in N(u)} x_i + \sum_{i \in A(u)} y_a]$$

By doing this the bias, the implicit feedback, the user-characteristics, the average rating and the user-item interaction describe the rating-prediction problem. The incorporation of the implicit feedback, the characteristics and the biases show the strength of matrix factorization. It is able to incorporate a lot of information from many different sources, thus making more accurate predictions. One possible way to improve this more is by making the different terms in the formula variable, like the before mentioned daily influence. Biases could be made dependent on the current day, thus making the formula dynamic, which could again improve the performance.

During the Netflix contest many of these kind of models were used to develop a recommender system that made 10% more accurate recommendations (RMSE, root-mean-square error). They proved to be the most popular and also best performing, acquiring root-mean-square error rates improvements of 7%-9%. In ‘Matrix Factorization techniques for recommender systems’ (Koren, Bell, Volinsky, 2009) it is shown that with the inclusion of every extra ‘dimension’ (e.g. bias, implicit feedback, users’ characteristics) and by turning some terms into their dynamic form the accuracy augmented. Thus showing the true potential of Matrix Factorization, its variability in
possible inputs. During the Netflix price these models outperformed the k-NN clustering models in accuracy, while being quite memory-efficient.

**Singular Value Decomposition**

Singular Value Decomposition (SVD) (Eckart, Young, 1936) is a particular way of solving the matrix factorization problem. In normal matrix factorization the factorization of the ratings matrix can take place in many different ways, the underlying principle is just that a matrix gets split up into a product of other matrices. This then provides a framework to analyse which factors drive the ratings. This factorization reduces the dimensionality of the matrix, as it is subdivided into the product of different matrices, which when multiplied give the total rating matrix with its original dimensionality. In the context of collaborative filtering it has been used often, but has been proven to be expensive to run as it requires a lot of computation power and the results are often not accurate for small datasets. It also does not offer a proper explanation of the reason why an item is recommended to a person that is understandable for its users. However combined with other algorithms it is capable of recommending items with good accuracy.

Singular Value Decomposition for a matrix $A: m \times n$ with rank $r$ is defined as (Sarwar, Karypis, Konstan, Riedl, 2002, p.2):

$$SVD(A) = U \times S \times V^T$$

$U$, $S$ and $V^T$ are matrices of dimensions $m \times r$, $r \times r$ and $r \times n$. With $S$ being a diagonal matrix with only nonzero values on its diagonal, the values on its diagonal have the property of being ranked from high to low ($s_1 \geq s_2 \geq s_3 \geq \cdots \geq s_r$) and being positive values higher than 0. It is called the ‘singular matrix’ as it contains the nonzero singular values. $U$ and $V$ are both orthogonal matrices (their inner-product is equal to 1) and are called the ‘left singular vector’ and the ‘right singular vector’ respectively. Their first $r$ columns correspond with the orthogonal eigenvectors that are associated with the nonzero singular values in the singular matrix. The first $r$ columns of $U$ correspond with the nonzero eigenvalues of $AA^T$ and the first $r$ columns of $V$ correspond with the nonzero eigenvalues of of $A^TA$. This is done using the principle of eigenvalues, which says that a matrix can be rewritten as a product of the diagonal matrix and then stretching in to both sides with the eigenvalue vectors ($U$ and $V$). Thus the ‘singular matrix’,
show the variance in the ratings matrix that is not explained by either of the singular vectors. These vectors should also be seen as a feature vector contain information on the different items and ratings, however this cannot be explained in a sense making way to its users. (Sarwar, Karypis, Konstan, Riedl, 2002, p.2-3) The following image gives an overview of the Singular Value Decomposition (http://xieyan87.com/2015/06/stochastic-gradient-descent-sgd-singular-value-decomposition-svd-algorithms-notes/):

![Singular Value Decomposition matrices](figure4.png)

However because of the before mentioned ranking of the \( s \)-values it is possible to pick the \( k \) highest ones and discard the others. This way the dimensions of all matrices are reduced by \( (r - k) \). This can then be used to reduce the original matrix as \( A_k = U_k \times S_k \times V_k^T \). The obtained matrix is then of rank \( k \) instead of \( r \) and consists of the factors that are most relevant to explaining the variance in the dataset as the highest singular values were picked. Researchers (Berry, Dumais, O’Brien, 1994) have also shown that this low-rank approximation of the original matrix is often better when used to make predictions as only the most relevant variances are kept. Finally it should be noted that the \( k \) nonzero eigenvalues that are kept, represent the interaction between ratings and products for every user (Sarwar, Karypis, Konstan, Riedl, 2002, p.2).

Another perspective on the eventual result of the singular value decomposition is given with the geometric interpretation. The singular vectors scaled by the different singular values can thus be interpreted as their coordinates in a \( k \)-dimensional space where every item is placed in the
centroid of its given rating. This way it leads to the mapping of the different items in the \( k \)-dimensional space, which can put items together even without knowing their rating. Then later this space can be used to calculate the vector of this item in the considered space and thus predicting the rating.

The value of \( k \) has to be chosen appropriately as this is one of the main drivers of the accuracy of the model. If singular values that still represent a big variance in the dataset are left out, an opportunity is missed to make the predictions more accurate. The real-valued feature vectors of size \( k \) of the items in the training set are then calculated within this space. These real-valued feature vectors are then used in some kind of learning algorithm (e.g. backpropagation (Nielsen, Robert, 1989)) to learn a function that predicts in which class an item will be or which scores will be accorded to a new item. One possible way of doing this is by creating artificial neural networks through backpropagation. Artificial neural networks have the characteristic of being very flexible, which is necessary if this SVD model has to be used for both classification and scoring.

Artificial neural networks are models which find a hidden function to explain the transformation from input to output. (Zhang, Patuwo, Hu, 1997) Different algorithms can be used to find this underlying hidden relationship, the goal of most is to find a non-linear function that explains the transformation from input to output. An approach used by Billsus and Pazzani (1998) used backpropagation, a back propagation of the squared errors through gradient descent, to find the artificial neural networks. Their approach is based on a feed-forward neural network with \( k \) inputs, 2 hidden units (which explain the transformation) and 1 output unit. For the hidden units, sigmoid functions were used to find the relationship between the input and output to then give a linear output. Their approach also wasn’t meant to predict ratings, but the difference in rating with the average rating as this yielded better performance. So implicitly defining the change in rating that was caused by the acquired feature vectors.

The hidden function used to find the transformation in this approach was the use of sigmoid functions. These kind of functions rely on the following underlying formula and is also called the ‘logistic transfer function’ (Zhang, Patuwa, Hu, 1998, p.13):

\[
f(x) = \frac{1}{1 + e^{-t}}
\]
So a function of this form is searched for by the learning algorithm to explain every transformation. In this case 2 hidden functions were used, which implies that 2 functions of the sigmoid-form are learned to explain the transformation. The sigmoid function is the most often used function in the context of the transfer function within artificial networks. It is characterized by its S-shaped form and is a particular form of a logistic function. In ‘Multilayer Feedforward Networks are Universal Approximators’ (Hornik, 1989) it is shown that this functions serves well for feedforward networks and is mentioned that if this does not perform well it’s not because of the underlying function, but the learning algorithms used to find the transfer formula’s.

So then finally once the dimensions have been reduced, the rating-patterns have been learned with the underlying hidden sigmoid functions and the linear output functions have been obtained, these functions can then be used to predict the ratings of new items. The characteristics of the new item are obtained as feature vectors and used in the linear functions to then obtain the change from the average rating caused by this feature vector.

Hybrid recommendations
In Hybrid recommendations both models, content-based and collaborative, are used to make predictions. However the way this is done can be very different. In general a distinction is made between 4 different approaches (Adomavicius, Tuzhilin, 2005, p.7):

1. Implementing both collaborative and content-based models and then combining their predictions.
2. Incorporating some content-based characteristics into a collaborative-based model.
3. Incorporating some collaborative-based elements into a content-based model.
4. Constructing a unified model, which incorporates both content-based and collaborative characteristics in 1 model.

Independent Combination of outputs
In this approach there are several different approaches. One approach is to use several independent recommender systems to produce a recommendation or rating prediction and then combining these outcomes by aggregating them. This aggregation can be of a linear form or can be done through voting, like for example majority voting. Examples of a linear approach can be
found in ‘Combining Content-based and Collaborative Filters in an Online Newspaper’ (Claypool, Gokhale, Miranda, Murnikov, Netes, Starin (1999) and an example of the voting-based approach can be found in ‘A Framework for Collaborative, Content-Based and Demographic Filtering’ (Pazzani, 1999). This kind of approach is easy as not a lot of effort has to be done to integrate the systems. However it should be noted that this kind of systems generally underperform, comparing to more advanced and integrated systems.

Another approach within this class of hybrid recommender is the use of independent recommender systems individually based on the current needs of the recommender system. This way the most optimal system can be chosen in certain situations. An example could be with the cold start problem of collaborative filtering. Here it could be useful to first use a content-based algorithm to provide ratings as long as not enough data entries were made. These kind of systems outperform collaborative filtering in this case, as CF has to first learn the patterns in the data before it is able to make accurate predictions. So to solve this cold start problem content-based algorithms can offer a temporary solution. Another application of this approach is creating a system which uses some kind of ‘quality’ metric that automatically determines which recommender system performs ‘best’ in the considered case. The DailyLearner system (Pazzani, 2001) is a recommender system that uses this approach to effectively decide upon the best approach in any given case. In this system the content-based recommender system is used first, to solve the cold start problem, until it fails. Once that occurs, the collaborative filtering will take its place and make predictions accordingly. The content-based system makes use of a k-NN clustering algorithm to cluster the different items together, however this is also influenced by the cold start problem.

Incorporation of content-based characteristics in a collaborative recommender system

In this approach a classic collaborative filtering algorithm is augmented with a content-based dimension. As an example Fab (Balabanovic, Shoham, 1997) is a collaborative-based recommender system which makes predictions mostly based on the collaborative filtering principles. However it does maintain user-profiles which are content-based. It calculates similarities between users with collaborative filtering, but the user-profiles themselves are created using content-based techniques. The inclusion of these characteristics helps the collaborative filtering as they can then be used to find similar users more accurately.
Another approach is the one of *filterbots* (Good, Schafer, Konstan, Borchers, Sarwar, Herlocker, Riedl, 1999). The principle of their approach relies on adding the content-based characterization as an extra user within the collaborative filtering framework. Different users are created based on the information they receive through content-based analysis and are then treated as users. The collaborative filtering technique then identifies the important information of these users to help in making predictions. For every user it analyses what information from which of these ‘*filterbots*’ is relevant for the user and then includes this in its recommendation algorithm. In their finding they have proven that this outperforms the general collaborative filtering approach, however which is not analysed is the impact this could have on the feasibility of the analysis. As datasets continue to grow in size, these created users, with information on all items and users, will surely make it more computationally heavy. Therefore this approach might not be feasible for large datasets, however can form a good extension for small to medium datasets.

**Incorporation of collaborative characteristics in a content-based recommender system**

In this approach certain collaborative characteristics are added to a traditional content-based recommender system. The most popular application of this kind of combination is the incorporation of a dimension reduction algorithm in a content-based recommender system. For example the Singular Value Decomposition could be able to reduce the dimensionality of a through content-based algorithm obtained, matrix. This then reduces the amount of noise and random variation in such a matrix thus performing some kind of data cleansing, before the application of other algorithms. This usually results in an improvement of prediction performance, comparing to the original content-based approach.

**A unified model**

In this approach both kind of models are not superior to one another or are not used separately from each other, but are combined into one unifying model. Most often this is done by developing a single rule-based classifier that incorporates both content-based and collaborative characteristics. In ‘*Probabilistic Latent Semantic Analysis*’ (Hoffman, 2013) a model is proposed which is called ‘*Probabilistic Latent Semantic Analysis*’. In this approach they combine Information Retrieval techniques with dimensionality reductions. They propose a probabilistic approach to documents, they define documents with the words that are used in the documents.
These are then represented as vectors of these underlying words, with their corresponding counts. After this they reduce the dimensionality with a kind of SVD approach for this. They decompose the original matrix into 128 factor decompositions, then probabilistic clustering is applied to then be able to classify new documents based on the probability predictions.

Another example of a unified model is the one proposed by Ansari, Essegaier and Kohli (1999). This model is based on Bayesian mixed-effects regression models, which rely on Markov chain Monte Carlo methods to estimate parameters and to predict rating. It uses profiles for both items and users to build a statistical model that includes both of their characteristics combined with other variables. The statistical model that predicts the ratings is given by (Adomavicius, Tuzhilin, 2005, p.8):

\[ r_{ij} = x_{ij} \mu + z_i \gamma_j + w_j \lambda_i + e_{ij} \]

With \( e_{ij} \) being the noise, which cannot be explained by the user-item interaction, \( \lambda_i \) being a not yet identified influence of user heterogeneity and \( \gamma_j \) being the influence of item heterogeneity. \( x_{ij} \) is a matrix containing both item and user characteristics, \( z_i \) is a vector containing the user characteristics and \( w_j \) is a vector contain the item characteristics. This model represents both the characteristics of both users and items (content-based) as well as the interaction between them (collaborative). Additionally some other terms are added to provide more accurate estimations, as these terms explain certain variance that couldn’t be explained by just analysing the interaction or the effect of the characteristics (Adomavicius, Tuzhilin, 2005, p.8).

Next to the incorporation of both content-based characteristics and collaborative characteristics, additional information can and should be added to achieve more accurate recommendations and solve some problems (like the new user or item problem). A possible solution for this is the incorporation of knowledge-based recommender systems. These systems provide additional information on a case of which is very little known. These are smart systems, meaning that they provide some kind of logical perspective on the case. The problem with this kind of recommender system is that it requires extra information on the case, which is not always available. However in recent developments web-crawling applications have become more popular in the use of
knowledge-based systems. These kind of applications make use of the abundance of information available on the internet to look up a certain case and link it with characteristics given on the internet. As an example a recommender system can be considered that sells desk equipment to businesses. If a new business accesses your website, a web-crawling application can then look up the business and find its main characteristics online; its location, its magnitude, the sector in which it’s active, ... These characteristics can then be used to link to other similar businesses, using for example a clustering algorithm, thus avoiding the lack of information for a new user. This can also be used for already existent clients of which not that much information is available, thus adding these sources of information can provide more accurate profiles of the existent clients. These profiles can then be used by other systems to more accurately predict ratings for other users as their features are more properly mapped out.

Furthermore in ‘Content-Boosted Collaborative Filtering for Improved Recommendations’ (Melville, Mooney, Nagarajan, 2002) is evaluated how these hybrid recommender systems perform in comparison with other systems. Their findings clearly show an increase in accuracy as more characteristics from other systems are added or incorporated. They are evaluated on 2 criteria: ‘Mean-Absolute-Error’ and Receiver Operator Characteristics. Their results are given by the following table (Melville, Mooney, Nagarajan, 2002, p.4):

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MAE</th>
<th>ROC-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pure content-based (CB) predictor</td>
<td>1.059</td>
<td>0.6376</td>
</tr>
<tr>
<td>Pure CF</td>
<td>1.002</td>
<td>0.6423</td>
</tr>
<tr>
<td>Naïve Hybrid</td>
<td>1.011</td>
<td>0.6121</td>
</tr>
<tr>
<td>Content-boosted CF</td>
<td>0.962</td>
<td>0.6717</td>
</tr>
</tbody>
</table>

**Figure 5: Comparison of CB, CF and Hybrid recommenders**

The first thing that can be noticed here is that the purely collaborative filtering approach outperforms the purely content-based predictor, which is also mentioned in other research. However the difference is so small that this is not conclusive evidence, as it depends purely on the used underlying models. Many other approaches to both collaborative filtering and content-based predictors are possible so these cannot be evaluated as easily. Nevertheless, comparing the ‘pure’ predictors with the content boosted collaborative filtering clearly shows an improvement in accuracy. The Naïve Hybrid however does not show this trend, for this many reasons are possible, one of which is the chosen parameters for the model. These are often not
optimized properly, thus resulting in very little or no improvement. Research in the latest year has been focusing a lot on the parameter optimization in recommender system algorithms, as this is crucial for its performance. Many applications of machine learning algorithms have been developed to optimize this using certain criteria, like the mean-squared error.

Recent developments

Recent research in the context of recommender systems is very widely spread. As mentioned in ‘Research Paper Recommender System Evaluation: A Quantitative Literature Survey’ (Beel, Langer, Genzmehr, Gipp, Breitnger, Nürnberger, 2013) it is mentioned that over 80 approaches exist for the recommendation problem for academic research papers, with more than 170 research articles published on them. The same can be said for the recommender systems for other applications, as the amount of different approaches to the same problem is very large so it is difficult to identify the most successful ones. They also mentioned that most developments have been made in the recent past as the field of recommender systems is now gaining a lot of traction. One possible reason for this is certainly the presence of certain competitions in this field, e.g. the Netflix Prize and the yearly KDD Cup. In most recent research many new approaches to the recommendation problem are suggested. Many trying to incorporate new sources of information in recommender system to improve its accuracy. An example of this can be found in ‘Combining similarity and sentiment in opinion mining for product recommendation’ (Dong, Mahony, Schaal, McCarthy, Smyth, 2016) in which they mine reviews given on products as a new source of information. In it they mine these reviews to gain information on the features of the different reviews as well as the sentiments linked to these reviews and features. They use a technique called ‘opinion pattern mining’ to analyse the different reviews and to identify the words which represent a sentiment and link these to certain features. Then similarities are calculated to find the general opinion, made by similar users. The recommender system they apply can be seen as a hybrid recommender that use collaborative recommender and content-based recommender separately to then combine the results after through a voting scheme.

Another example of a recently developed recommender system is the ‘Pairwise Interaction Tensor Factorization’ (PITF) (Rendle, Thieme, 2010) method. In this they incorporate elements from the current context into their recommender system. In their approach they define the current context with tags. They achieve a time-aware recommender system through approach the days as ‘structured tags’ in order to compare same dates with the year before. The social
context they define by tags that represent the current relevant topics which can be obtained through web-crawling websites using automatic information retrieval techniques. Then they perform the ‘Pairwise Interaction Tensor Factorization’ approach on the obtained tags, together with the already available information on its users and items. This approach is a particular case of matrix factorization in which they use a combination of the Tucker Decomposition and the Canonical Decomposition, which are both techniques to reduce the main ratings tensor into different lower-order tensors. Tensors are used in this case as their approach relies on pair-wise interactions, thus requiring a more flexible multi-dimensional representation of the ratings matrix. However the technique is quite similar to matrix factorization. By combining both decomposition techniques they produced a recommender system, which outperformed both techniques both in runtime and in accuracy. It should also be noted that for the optimization criterion used to optimize the decomposition they used the BPR-opt criterion. This is relevant as this was developed using the MyMediaLite framework and will be analysed more thoroughly in the analysis of the MyMediaLite framework. The BPR-opt criterion was first developed in the context of the KDD Cup (a recommender system competition) and showed great potential during the competition as they won the first place. This criterion has proven to improve performance of many original algorithms, thus proving the relevance of machine learning in the context of recommender systems. Many improvements to existing models can be made solely through using other optimization criteria, therefore it has been the subject of recent research.

Other recommender systems that incorporate context characteristics have also been developed, like LARS (Levandoski, Sarwat, Eldawy, Mokbel, 2012). LARS incorporates the location of its users when rating an item to improve accuracy of a normal collaborative filtering recommender system. It is based on a k-NN clustering algorithm, giving the top $k$ objects near the user. However also employing this to find more similar users based on their given location. They use Foursquare as a source for location-based ratings, as this is a widely used mobile application. From this they were able to train a model to combine with the MovieLens dataset, incorporating location-based ratings into a previously non-spatial dataset. This example shows that new developments in recommender systems are made not just because of the new needs of customers, but also because of the newly available technologies. Recent developments in technology, primarily the wide-spread use of smartphones, are sources of new information on users and can be used to improve recommendations or to provide new kinds of
recommendations. Also because of the adaptation of the smartphone by many users, much more information is available and comparing to a decade ago the amount of data has increased exponentially. Therefore an analysis will be made of the MyMediaLite recommender system library to evaluate whether this framework shows potential for companies active in the field of e-commerce, more precisely through websites.

**MyMediaLite**

MyMediaLite is an open-source library of recommender system algorithms (Gantner, Rendle, Freudenthaler, Thieme, 2011). It focuses on two scenarios in collaborative filtering: Rating Prediction and Item Prediction from positive-only implicit feedback. It contains state-of-the-art algorithms for both tasks and does not require deep knowledge of programming to use. It was written in C#, but since then many applications have been made to modify it through other languages, like java, Python, etc. Its applications so far have mostly been for research purposes and development of new recommender algorithms, nevertheless it could also be used quite easily by companies to implement a recommender system. Besides the available algorithms it also provides the possibility of using self-developed recommender algorithms and the possibility to evaluate a recommender system on certain criteria like ‘Root Mean Square Error (RMSE)’ and ‘Mean Average Error (MAE)’. The framework can be compared to a similar recommender system library called ‘Duine Recommender’ (Van Setten, 2005). This library provided a set of recommender system algorithms together with the possibility of combining these algorithms (hybridizer) to form a hybrid recommender system. However as it was last updated in 2009, it does not offer state-of-the-art recommender algorithms, unlike the MyMediaLite framework, which was updated December 31st 2015.

To use the functionalities of the MyMediaLite framework the library only has to be downloaded and can then be accessed with the command-line tool which is available in every operating system. Its flexibility and extensibility have also been proven in several cases (such as the KDD Cup). It is both capable of making accurate recommendations as well as producing feasible runtimes for the available algorithms. A deeper analysis of its characteristics and its potential will be made in the following part.
History
MyMediaLite was originally developed by Zeno Gantner, Steffen Rendle, Christoph Freudenthaler and Lars Schmidt-Thieme in 2010. It originates from the MyMedia project, a project funded by the European Commission in order to address “... the key social problem of information overload that has been called the "Crisis of Choice". “ (Voß, Newell, Marrow, 2009, p.1). It was an open-source software that was able to incorporate many different kinds of Audio and Video files as well as a multitude of recommender system algorithms into one system. It also provided a framework in which recommender system algorithms could be evaluated and in which metadata could be explored to get a better understanding of the underlying characteristics of a system’s users.

The MyMedia framework was applied to 4 different business cases. A real-time recommender system in the BBC, a recommender system for a video-on-demand service for the BT, a recommender system that incorporated social networking for the Microsoft MSN video service and an e-commerce application for Yoguie. In general is was created with the intention of creating a platform for multimedia, which offered tools to enrich the metadata automatically and provide information for recommender system to more efficiently share the content. It was defined as “a dynamic personalisation framework that links the different components required for effective recommendation with user recommendation and feedback.” (Marrow, Hanbidge, Rendle, Wartena, Freudenthaler, 2009). It was designed so that developers could use it to develop an extension to their already existent software, which is easily plugged into the code of their original software. They also attempted to make their framework flexible, i.e. choice between different recommender algorithms and compatibility with several file formats for the used content, the associated metadata and its automatic enrichment. They allowed for this by programming it in an object-oriented way as this offers most flexibility.

In its creation the creators made sure that the Application Programming Interface (API) was programmed in a very extensible way so that its users could easily import multimedia files as well as information files from different sources. Thus optimizing the open-source characteristics of the framework. This included compatibility with different operating systems, different file formats, different interfaces and possibilities to link the framework to external applications: Marrow, Hanbidge, Rendle, Wartena, Freudenthaler, 2009, p.2):
In its structure the core framework is linked to a relational database in which all data is stored. The code in itself was designed in such a way that both experienced and novice developers could work in it. The division in different classes made it so that for the novice developers they did not have to go into the classes of which they didn’t know a lot, only the parts that they wanted to change. It also offered a social network layer, which allowed for easy integration into social network features. As an example this could be used for advertisements in social media platforms, using the available data on a user to recommend the most relevant items in a side bar. Next to that it was also created in such a way that it could run at different runtimes, through parallelization, so that the activities could take place on different processors at the same time. This is important in the case of real-time recommender systems as these need to make recommendations instantly. It also allowed for processes that are separated from the real-time recommender systems to work on a different processor, so that its performance was not reduced by the real-time recommender algorithm.

The recommender interface of the framework consists of many possible recommender algorithms and relies on a relational representation of entities. Entities, like users, tags and items, are linked through relations as ‘has Tagged’ or ‘friendOf’. Each entity is given a unique ID-number within every type of entity. This kind of modelling allowed for flexible use of recommender algorithms and easy implementation as it also provided direct mapping to database tables.
Another previously mentioned feature of the framework is the metadata enrichment. It achieves this by analysing the available information on items, users and tags. However this information is most often available in text form, (like descriptions, reviews, synopses) which is difficult for recommender systems to work with. The MyMedia framework solves this by including keyword extracting modules in its software, however in the later MyMediaLite framework this was left out, as it was too computationally heavy.

In conclusion this framework can be seen as a one-stop shop for almost everything that had to do with recommender systems and the sharing of multimedia content and where it lacked some capabilities there was always the possibility of incorporating new elements. Eventually the project was finished and this framework was changed into the MyMediaLite framework. As the name suggests this was a lighter version of the original framework. They removed the most computational heavy parts of the system. They also reduced the overhead of software needed to use the framework by not making the database a requirement and by deleting some more complex recommenders and recommender flows.

Open Source
One of the main characteristics of MyMediaLite is its open source character. This is an important note as it has a lot of influence on the framework and its future development. First of all the most obvious benefit of this is that the software can be used freely by anyone with a computer. It is licensed under the GPU General Public License, which means that the creators of the software gave up their normal author rights. This license implies 4 essential freedoms: freedom to run the program for any purpose, freedom to study the software and change it, freedom to redistribute copies and the freedom to redistribute modified versions (GNU, 2014). The specific terms under which the framework was published is called GPLv3 and also allows for commercial applications. These characteristics offer a framework that is developed in cooperation of both its users and developers. As the MyMediaLite framework was designed in a flexible and extensible way this is crucial to motivate further development. Everything from the source code to the recommender algorithms is free to access and anyone with some knowledge of programming can modify it so that it better satisfies their needs. These modifications are then often openly shared so that other users can apply these modifications as well. It is because of this process that slowly a community is built around open source software, as mentioned in ‘Latent Social
Structure in Open Source Projects’ (Bird, Pattison, D’Souza, Filkov, Devanbu, 2008). In their research they studied how a community is slowly built around open source software as it is being developed. They observed that in open source software projects a natural modulation slowly forms around the software, dividing its different components in sub-communities. In these sub-communities people work together to improve the software and perform tasks.

The emergence of these kind of communities is another benefit of this kind of software, it should even be considered as more important than its free usability and share-ability as it ensures future development without many development costs for the original creators. In the case of MyMediaLite this is very important as the field of recommender systems is developing at a fast pace and inventing more and more approaches to solving the recommendation problem. By allowing external algorithms to be incorporated and by allowing the source code to be changed by its users, it provides the framework in which all of these new developments can be added in a rather easy way (because of the software structure). It also allows for easy incorporation in other software, which allows even small companies with less available capital to incorporate recommender systems into their e-commerce websites. Also because of the community that is slowly being built around the software, this requires less maintenance of the system as this is done by its users. This is very important for smaller companies as the maintenance costs of recommender systems are usually a big constraint for them. However it should be noted that it does require data to be processed to make recommendations and although this is sometimes shared in the community, every company should have their own data to make accurate recommendations. As the gathering of data is often a quite pricey activity (if it’s done properly) and thus still has some constraints for smaller companies, nevertheless will the open source framework of MyMediaLite reduce this constraint.

**File Format**

Another part of the MyMediaLite framework that should be considered is its compatibility with file formats. This is crucial as to what kind of systems can interact with the framework. As the framework requires information on different aspects: rating files, positive-only feedback files, attribute files, user and items lists and item recommendation files. MyMediaLite in general supports 2 main types of files for this information, SQL database files and simple text files.
Rating Files
The rating files can be in different formats, it supports both integer and non-integer ratings. It consists of the user ID, the item ID and the rating value. The different values can be separated by either a tab (.tsv), a whitespace or a comma (.csv), where tsv stands for “tab-separated values and csv stands for comma-separated values. It also allows for time and date stamping as this is often crucial for time-aware recommenders. The timestamps are given by a number, e.g. “978300760” and date stamps are written as “2005-12-04” for example. These are put behind the user ID, item ID and rating value and are separated by the same character that was used in the separation between the ID’s and the rating value. An example of rating data with a date and time stamp is given by (http://www.mymedialite.net/documentation/rating_files.html):

Table: Rating data with dates and times

<table>
<thead>
<tr>
<th>User ID</th>
<th>Item ID</th>
<th>Rating</th>
<th>Timestamp</th>
</tr>
</thead>
<tbody>
<tr>
<td>5951</td>
<td>50</td>
<td>5</td>
<td>2009-08-05 00:50:36</td>
</tr>
<tr>
<td>5951</td>
<td>223</td>
<td>5</td>
<td>2009-08-02 17:19:32</td>
</tr>
<tr>
<td>5951</td>
<td>260</td>
<td>5</td>
<td>2010-05-04 21:21:02</td>
</tr>
<tr>
<td>5951</td>
<td>293</td>
<td>5</td>
<td>2009-09-25 05:04:24</td>
</tr>
<tr>
<td>5951</td>
<td>356</td>
<td>4</td>
<td>2010-06-30 02:07:57</td>
</tr>
<tr>
<td>5951</td>
<td>364</td>
<td>3</td>
<td>2010-06-11 04:54:41</td>
</tr>
<tr>
<td>5951</td>
<td>457</td>
<td>3</td>
<td>2010-06-11 14:26:32</td>
</tr>
</tbody>
</table>

Figure 7: Rating data with date and time stamp

Positive-only feedback files
Positive-only feedback files represent a positive interaction of the user with an item. This can consist of an item being rated, an item being viewed or an item being bought depending on the used approach for the recommendation problem. The file format consists of a user ID first followed by the item ID for which positive feedback was given. These two ID’s can again be separated by either a tab (.tsv), a whitespace or a comma (.csv). An example of the comma-separated values file can be given by (http://www.mymedialite.net/documentation/implicit_feedback_files.html):
Comma-separated columns (.csv)

5951, 50
5951, 223
5951, 260
5951, 293
5951, 356
5951, 364
5951, 457

Attribute files
Attribute files are based on binary attributes, i.e. it can only have 2 values, for example 0 could usually indicate the absence of a certain characteristic or attribute and a 1 could indicate its presence. Again all 3 different separation are possible for this file format. The different entity-attribute relationship are represented by the entity ID, which can either be an item or a user, followed by the attribute ID. Thus if for example the number 51 could be considered as the item-ID of a Harry Potter movie and the attribute-ID 5 could be the characteristic of being a fantasy movie. If these are put behind each other in the file, this implies that the Harry Potter movie is a fantasy movie. Thus an attribute file is given by (http://www.mymedialite.net/documentation/attribute_files.html):

Tab-separated columns (.tsv)

51 5
51 22
51 26
51 29
51 35
51 36
51 45

User and Item lists
User and item lists are straight forward as these files contain the unique ID’s given to the different items and users. One ID per line is listed and no additional information is given in this
file. This file format is straightforward and an example is given by (http://www.mymedialite.net/documentation/entity_files.html):

**Example**

```
5
22
26
29
35
36
45
```

*Figure 10: user list*

**Item-recommendation files**

The item-recommendation files are usually the goal of a recommender system, however these files are also needed to train a model. When training a model predictions are made on the currently available information, however to improve the model it needs some kind of feedback on its accuracy. Another reason for this is that in hybridizing recommender systems the output of a certain model, like the k-NN clustering algorithm has to be used in another algorithm. It is because of this that the framework also has to be compatible with these files. These files consist of the user ID followed by the top items recommended to him, with the amount depending on what \( N \) is set on. Every recommended item is also linked to its accorded score, thus ranking it from the highest to the lowest scores within the top \( N \) items. The recommended items are put between brackets as follows (http://www.mymedialite.net/documentation/item_recommendation_files.html):

**Example**

```
0 [9:3.5, 7:3.4, 3:3.1]
```

*Figure 11: Item-recommendation file*

So in conclusion it can be said that the file formats that MyMediaLite requires are straightforward and easily obtained by either writing them manually or exporting them from SQL databases. It also provides routines to read data directly from SQL databases. One could also create small scripts of code to transform available information into the required format by replacing certain
characters or by information retrieval techniques as mentioned before in this research. The simplicity of the file formats also allows for small companies, which usually have less data to process, to do this manually if an investment in automated information retrieval is too expensive.

Application
MyMediaLite, as mentioned before, mainly focusses on 2 recommendation problems: rating prediction and item prediction from positive-only implicit feedback. For this 2 tools were created: the item-recommendation tool and the rating prediction tool. It provides easy access to these tools through command-line programs, which are available on all relevant operating systems. The package first has to be downloaded and can then be accessed through the command-line programs, thus avoiding the use of complicated code to access the framework. However it can also be used through different programming languages. Originally the framework was written in C# on the .NET platform. The .NET platform was originally created by Microsoft and can be defined as “An open language platform for enterprise and Web development.” (Meyer, 2001). MyMediaLite runs on this platform as it allows for web applications and open sharing of the source code. Because of its open character it is also able to run on all relevant operating systems (Microsoft Windows, iOS and Linux). Nevertheless the platform can also be run through other programming languages like Python, F#, Ruby and Clojure. For this the required coding is available with its download. A portal for usage through java has also been developed since the original release and here the open source character of the framework already shows its potential as this was created by one of its users, Chris Newell.

Next to the available item-recommendation and rating prediction algorithms the platform also provides additional features. One of which is a variety of evaluation measures to evaluate a recommender system algorithm. Additionally the framework also allows for comparisons against baseline methods, which can more precisely evaluate the performance of an applied algorithm. Another available feature is the capability of serialization. This is needed as model training often requires a different machine than the intended machine on which it will be run. Therefore it is necessary that the framework is able to store and reload its previously developed recommenders, which is one of the features of the MyMediaLite framework.
So far the MyMediaLite framework has mostly been used for research purposes, mostly concentrated within the academic research field within universities. However a few companies like Microsoft, AT+T, BBC and Zunnet Technologies have been using it for their research and development department. It has also been applied to a website, animerecs.com, in which it supports the search function through recommendation to find anime video’s more efficiently. So although it has not yet been used in actual commercial applications, its potential has been recognized by many institutions. A reason for this might be that companies do not trust the framework enough to deposit their valuable data into it. However what is not published is what recommender systems that were developed in the framework and were later on incorporated in other software applications.

It should also be noted that the MyMediaLite framework is capable of “...making between 1,400,000 (k =120) and 7,000,000 (k = 5) predictions per second.” (Gantner, Rendle, Freudenthaler, Thieme, 2011, p.1). It has also been shown that its runtime evolves linearly as more factors are added. This is an important notion as many recommenders’ runtime tends to increase exponentially as the amount of features increase.

In this research the focus will be on the item recommendation tool as this is the tool that would be used by companies to recommend items to its users. The rating prediction tool has been optimized for rating problems and takes into account the underlying characteristics in a different way. For example it will take into account the probability of a user rating an item, which is not relevant for the item recommendation problem. Furthermore the result of this rating prediction tool is the given ratings for every possible user-item pair. This however does not take into account the previously bought items by the users, so recommending items based on these predicted ratings could result in the users receiving constant recommendations on products that they already have. It is possible though to transform this data and remove these items, but the item recommendation tool already performs this automatically, so this approach is more ready for immediate implementation. As the potential of this framework is evaluated also for users with little knowledge of programming, this is left out of the scope as most unexperienced users wouldn’t know how to transform, aggregate or filter the resulting data.
BPR

As an example for the potential of the MyMediaLite framework an algorithm, which was developed in the MyMedia framework will be analysed. The algorithm has been applied to many models to improve its prediction accuracy and within the MyMediaLite framework many recommenders are available that utilize this algorithm, like matrix factorization. Its main purpose is to optimize the model parameters and it has been proven to perform very well in the context of the KDD Cup. Therefore it can also be considered as an example of the possibilities of the framework in the context of algorithm development.

The Bayesian Personalized Ranking (BPR) (Rendle, Freudenthaler, Gantner, Thieme, 2009) technique was first developed with the intention of creating a technique that was specifically created for the problem of ranking items as this is the most common interpretation of the recommendation problem. It makes use of the BPT-Opt criterion to optimize model parameters for the personalized ranking problem. This criterion is generic and can be applied to many recommender algorithms like matrix factorization. It is applied to different models through a machine learning technique called learnBPR, which is based on stochastic gradient descent with bootstrap sampling. The stochastic gradient descent algorithm has been mentioned before in this research, however the use of bootstrap sampling results in a different approach which will be analysed more deeply.

The BPR technique relies on implicit feedback as the driver for its recommendation. It differs from most other techniques in this context in the way that it treats negative feedback. Recommender systems that rely on implicit feedback usually have only 2 possible values for this feedback its presence or its absence. For example when a user buys an item this will be stored as a positive value, indicating positive implicit feedback and when the user hasn’t bought the item, it will be stored as a negative value, usually 0. By doing this however the model that is trained on this dataset will wrongly classify many cases as negative, while it was still possible that the user would buy the item later. So if the model is then fit to this data it will not be able to rank the items as it is only able to predict 0’s and not whether a user is still interested in an item. As mentioned in ‘BPR: Bayesian Personalized Ranking from Implicit Feedback’ (Rendle, Freudenthaler, Gantner, Thieme, 2009), these kind of learning methods only work when some form of regularization technique is applied to avoid overfitting the model. The way these
Methods turn their observations into data can be represented as follows (Rendle, Freudenthaler, Gantner, Thieme, 2009, p.3):

![Diagram showing common approach to missing values](image)

**Figure 12: Common approach to missing values**

With “?” being both the missing values and negative values.

BPR however approaches this in a different way. It makes use of item-pairs to check whether one item is preferred over the other and not scoring an item individually. In this way it is truly able to predict a ranking of different items. They achieve this by ranking all observed items higher than the non-observed ones as this implies that the user at least looked at it, showing initial interest. However for the items that have both been observed or not observed no preference can be deducted. Then for every user a mapping is made with the relative preferences of that user (Rendle, Freudenthaler, Gantner, Thieme, 2009, p.3):
For example here it can be observed that user 1 ($u_1$) has not seen item 1, both has seen item 2. So in its mapping item 2 is preferred over item 1. However User 1 has seen both item 2 and 3, so this results in a question mark as its relative preference is not known. Regarding the considered problem some assumptions can be made on the ordering of all items for a user $u$ (Rendle, Freudenthaler, Gantner, Thieme, 2009, p.2-3):

\[
\forall i, j \in I: i \neq j \rightarrow i >_u j \lor j >_u i \text{ (totality)}
\]

\[
\forall i, j \in I: i >_u j \land j >_u i \rightarrow i = j \text{ (antisymmetry)}
\]

\[
\forall i, j, k \in I: i >_u j \land j >_u k \rightarrow i >_u k \text{ (transitivity)}
\]

Thus fulfilling all of these assumptions leads to a completely ranked list of items for every user. $>_u$ is considered as the personal ranking for user $u$.

In the BPR technique they train the model on these observed preferences to then apply this model to new cases. Positive feedback is given for items that are preferred over another item, negative feedback is given for items that were not observed when compared with other observed items and then finally a missing value is given for the items that have both been observed or not
observed. The training set is thus given formally by (Rendle, Freudenthaler, Gantner, Thieme, 2009, p.3): $D_S : U \times I \times I$:

$$D_S := \{(u,i,j) | i \in I^+_u \land j \in I \setminus I^+_u \}$$

By doing this it is able to make a distinction between the missing values, while in other machine learning approaches these are all considered as negative feedback. This approach should thus be more accurate as it takes into account 3 possible outcomes and not just positive or negative.

To learn the optimal ranking of items the BPR optimization criterion is then used. This criterion is called ‘BPR-Opt’ (Rendle, Freudenthaler, Gantner, Thieme, 2009) and is formally given by (Rendle, Freudenthaler, Gantner, Thieme, 2009, p.4):

$$BPR - Opt := \sum_{(u,i,j) \in D_S} \ln \sigma (\hat{x}_{u,i,j}) - \ln p(\theta)$$

And can be rewritten as:

$$BPR - Opt := \sum_{(u,i,j) \in D_S} \ln \sigma (\hat{x}_{u,i,j}) - \lambda_\theta \|\theta\|^2$$

With $\sigma(\hat{x}_{u,i,j})$ being an arbitrary real-valued function of the natural logarithm of the logistic sigmoid form $\left(\frac{1}{1+e^{-x}}\right)$ that is found by the application of another statistical method, like matrix factorization. It represents the unique relationship of a user $u$, an item $i$ and an item $j$ as a vector of the model parameters $\theta$. The different relationships are then summed so that its optimization leads to the highest probability of item $i$ being ranked over item $j$: $p(i > u,i,j|\theta) := \sigma(\hat{x}_{u,i,j}(\theta))$ in every considered case. However than some kind of regularization has to be applied to make the optimization criteria less prune to overfitting. The $\lambda_\theta$ part of this regularization are the model specific regularizations and the $\|\theta\|^2$ part represents the underlying distributions of the data. Thus by subtracting this form the found relationships, its optimization then results in the best parameters for the underlying model (Rendle, Freudenthaler, Gantner, Thieme, 2009, p.4).

However before being able to optimize the ranking criterion, the underlying relationships first have to be found. For this it makes use of a machine learning technique based on stochastic gradient descent. They call this technique learnBPR and functions in the exact same way as the original stochastic gradient descent, except for the cases on which it is learned. It makes use of bootstrap sampling to learn the different relationships, which implies selecting cases randomly
and performing the stochastic gradient descent only on these cases. This is needed as the pair-
wise approach results in an exponential increase in different cases as the data set becomes
larger. Thus if this had not been applied the runtime of the model would increase exponentially.
However by applying the bootstrap sampling technique the runtime increases linearly (as the
amount of cases was chosen this way) with the enlargement of the dataset, without much
accuracy loss. It even proved to be more accurate than the original stochastic gradient descent.
The general procedure of this algorithm is given by (Rendle, Freudenthaler, Gantner, Thieme,
2009, p.5):

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{learnBPR.png}
\caption{learnBPR procedure}
\end{figure}

Where the 5th and the 6th step are the actual application of the stochastic gradient descent as it
learns the model more accurately with every considered case. With $\frac{\partial}{\partial \theta} \hat{x}_{uij}$ being the direction in
which the relationships will be modified as the algorithm learns from more cases. One thing that
should be noted here is that to optimize the model using the learnBPR approach only the
gradient of the model and its regulation factors are needed.

This machine learning technique has been compared to the normal stochastic gradient descent
in the ‘BPR: Bayesian Personalized Ranking from Implicit Feedback’ (Rendle, Freudenthaler,
Gantner, Thieme, 2009, p.6) paper. Here it was observed that the learnBPR technique led to
much faster convergence:
It should also be noted that the AUC or ‘Area Under the Curve’, which is often used for the evaluation of predictive algorithms is even higher for the learnBPR than for the original stochastic gradient descent. One reason for this could be that the original form of the stochastic gradient descent is too prone to overfitting as it takes into account every possible case and can thus include underlying distributions of the data which should not be taken into account.

As the BPR framework in general is always applied to another model, many possible applications are possible. However some have been compared to other recommender algorithms. In the comparison it’s clear that the BPR variants of the original models outperform the standard models and even the more complex variants of them, like the weighted risk matrix factorization. Nevertheless it should be noted that at the beginning the performance of the BPR models is lower as it performs bootstrap sampling and as such hasn’t learned enough yet from the training set. This comparison can be seen in the following graph (Rendle, Freudenthaler, Gantner, Thieme, 2009, p.9):
It can also be noted that as the number of dimensions increase the models get more accurate.
The main reason for this is the bootstrap sampling as this limits the cases from which the model is learned. So as the dimensions increase more samples are taken and result in an improved accuracy.

Recommender System library: Algorithms
Now an analysis will be made of the available recommender algorithms that are available within the MyMediaLite library. These will be linked to the before made classification of recommender systems and its underlying methods will be made clear.

Item recommendation tool
The item recommendation tool focusses on the problem of recommending items which are most likely to be liked by its users. Within the framework it mostly uses positive-only implicit feedback as feedback to make estimations of this. Positive-only feedback means that it will only take into account the implicit positive feedback, such as purchasing an item, clicking on an item and viewing an item. From this it will derive similarities between users and items to then make recommendations based on the found patterns.

The item-recommendation tool offers a multitude of algorithms as well as the possibility to alter many of its parameters, such as the learning rate, weighted or not, regularization, confidence levels and the number of iterations. The tool provides easy evaluation techniques as well as the possibility to export the predictions made by an algorithm to a new file. This file can then be used
by other software to recommend the predicted items to its users. The general item recommendation command is in the form of:

“item_recommendation --training-file=FILE --recommender=METHOD [OPTIONS]”

The “—training-file=FILE” part of the command is used to link the data used to train the model on. It also has the possibility to add “—test-file=FILE” to add a test set of the data to more properly train the model as this is usually needed for machine learning algorithms. In the “—recommender=METHOD [OPTIONS]” part of the command, the recommender algorithm that will be applied is picked. Additionally the options for the chosen recommender algorithm are defined. If no further options are defined in the command line, the algorithm will run with its standard set of parameters. An example of the standard parameters for user-kNN clustering is given by (http://www.mymedialite.net/documentation/item_prediction.html):

```
ItemKNN k=80 correlation=Cosine q=1 weighted=False alpha=0.5 (only for BidirectionalConditionalProbability) supports --online-evaluation
```

*Figure 17: Standard parameters for k-NN clustering*

The results of such an algorithm can then be used to make predictions for its users using the “—predict-items-number=N” command. In this it is possible to define how many items should be recommended to every user. The result can then be exported by using the “—prediction-file=FILE”, which exports the results to a new file labelling it with the name given in the “FILE” part of the command. So an example for an entire query can be given by:

“item_recommendation --training-file=u1.base --test-file=u1.test --recommender=userKNN --predict-items-number=5 --prediction-file=Result”

This command would first cluster all users together based on its standard parameters, then predict the 5 most likely items that a user would like and then the file is exported to the ‘Result-file’.

Now that the normal procedure for using the item recommendation tool has been analysed, an analysis of the available algorithms will be made.
Baseline Algorithms
Baseline algorithms are algorithms created for the evaluation of predictive algorithms. As the result of a recommender system is often evaluated by its accuracy it has to be placed in context to see how it relatively performs, otherwise the accuracy measure carries little meaning. The MyMediaLite offers 3 different baseline algorithms for this in its item-recommendation tool: “MostPopular by user”, “Random” and “Zero”.

The Random-algorithm calculates a baseline by using random recommending items randomly, without taking into account any of the users’ or items’ characteristics. A recommender algorithm should always have a higher accuracy than this baseline, if it doesn’t this would imply that it is better to recommend items randomly than using the recommender algorithm.

The Zero-algorithm is also a baseline-algorithm and is most often used for debugging. It predicts a score of 0 for every case and then makes recommendations based on the accorded scores. However since these are all 0’s this implies more of a random selection than a real prediction-based recommendation. This functions as a debugging algorithm because if this yields good results, this could be an indication that there are some underlying patterns that were missed. An example of this could be that the item lists are not ordered randomly, but follow a certain order. This can then be noticed by applying the zero-algorithm.

The “Most Popular” baseline is the third baseline algorithm available within the item recommendation tool. It makes predictions based on the overall most popular items, so no personalized recommendations will be made. A comparison against such an algorithm provides insight into the usefulness of another recommender system, because if the accuracy of the model is lower than this, the potential of the recommender system is non-existent.

As an example the zero-algorithm, the random algorithm and the most popular-algorithm will be tested on the Movielens dataset to show its performance:
Random:

In the zero-algorithm it can be observed that the AUC (Area Under the Curve) is observed at a level of 0.50688, which is about 50%. This implies that it is basically the same as randomly guessing whether a user will be interested in the item. This was to be expected as according a score of 0 to every item would result in randomly guessing whether a user would be interested.

Most Popular:

In the zero-algorithm it can be observed that the AUC (Area Under the Curve) is observed at a level of 0.50688, which is about 50%. This implies that it is basically the same as randomly guessing whether a user will be interested in the item. This was to be expected as according a score of 0 to every item would result in randomly guessing whether a user would be interested.

*Matrix Factorization algorithms*

**BPR-MF**

BPR matrix factorization is a form of matrix factorization in which the decomposition of the main matrix takes place through the BPR-Opt criterion. It applies the bootstrap sampling to learn the most optimal decompositions and optimizes for the before mentioned BPR-Opt criterion. The application of the BPR framework results in a Matrix Factorization model which is optimized for the ranking problem. As mentioned before in the analysis of the BPR approach only the gradient and the regularization factors are needed to optimize the model for ranking purposes. As the parameters for matrix factorization are given by: \( \theta = (W, H) \) as the matrix factorization attempts to decompose the target matrix into a product of matrices: \( \hat{X} := WH^T \). The prediction formula for a user \( u \) and an item \( i \) can then be given by (Rendle, Freudenthaler, Gantner, Thieme, 2009, p.6):

\[
\hat{x}_{ui} = \langle w_u, h_i \rangle = \sum_{f=1}^{k} w_{uf} \cdot h_{if}
\]

The application of the BPR-Opt criterion then results in the following derivatives for the gradient (Rendle, Freudenthaler, Gantner, Thieme, 2009, p.6):
\[
\begin{align*}
\frac{\partial}{\partial \theta} \hat{x}_{uij} &= \begin{cases} 
(h_{if} - h_{ij}) & \text{if } \theta = w_{uf}, \\
w_{uf} & \text{if } \theta = h_{ij}, \\
-w_{uf} & \text{if } \theta = h_{ij}, \\
0 & \text{else}
\end{cases}
\end{align*}
\]

With \( h_{if} \) and \( h_{ij} \) being the feature vectors for item \( i \) and item \( j \) respectively and \( w_{uf} \) being the feature vector of user \( u \). This gradient implies that no change will be made to the model parameters (factors) when its value is not equal to the value in either one of the item feature vectors (only 1 considered factor) or equal to the relative value of the user feature vector. If the considered parameter however is equal to one of these 3 cases and adjustment will be made to the considered factor of the parameter in the before mentioned direction. Then regularizations are added to avoid the model to overfit to the data. These are given by \( \lambda_W, \lambda_{H+}, \lambda_{H-} \) for the user’s features, positive updates on \( h_{ij} \) and negative updates on \( h_{ij} \) respectively. This way the decomposed matrices are optimized for the ranking problem (Rendle, Freudenthaler, Gantner, Thieme, 2009, p.6-7).

As an example the algorithm was tested on the MovieLens dataset and this resulted in the following accuracy:

| training_time | 00:00:05.186132 | AUC | 0.92228 | prec@5: 0.48081 | num items | 1682 | num lists | 459 | testing_time | 00:00:00.565597 |

Here the AUC is at a level of 0.92228, which is remarkably higher than the before tested baseline, which was to be expected as it does more than random guessing.

**Multicore BPR-MF**

The Multi-core BPR-Matrix Factorization is another available matrix factorization algorithm, which makes use of the same underlying principles as the normal BPR-MF. However this algorithm is able to divide its work over several cores, thus performing a parallelization of the model. This means that within the processor of a computer, which consists of multiple cores, the different computations needed for the model are divided over the different cores. This renders the algorithm less heavy for the platform on which it is run and this should be considered when other algorithms or software is running.

When tested on the MovieLens dataset however this resulted in a slightly higher accuracy:
The reason for this difference is most likely that in its integration in the source code this was done in a slightly different way. However what should be noted is that the training time went down from 6.0764026 seconds to 2.3325674, so it is about 3 times as fast as the original BPRMF algorithm. From this it can be concluded that the Multicore BPR-MF uses the processor on which it is ran more effectively, because of its parallelization.

**Weighted BPR-MF**

This algorithm is a variant of the before mentioned BPR-MF algorithms. It differs from the others in the fact that it’s weighted. These weights are calculated based on the overall popularity of a certain item or user. As items are sometimes more popular, this can cause a certain bias towards items, therefore weights are added to take this into account. As showed in the analysis of the matrix factorization concept this is often added to avoid the cold start problem. However in the context of the BPR framework it has to be considered that the usage of the bootstrap sampling has some influence on how the model is trained. More popular items appear more often in the item-pairs as these are observed more frequently. It is because of this that extra variables are added to avoid a biased model. The optimization criterion is then given by (Gantner, Drumond, Freudenthaler, Thieme, 2012):

$$\max_{W, H, b} \sum_{(u,i,j) \in D_S} w_u w_i w_j \ln \sigma (b_i - b_j + w_u h_i - h_j) - \lambda_u \|W||^2 - \lambda_i \|H||^2 - \lambda_b \|b||^2$$

With $b_i$ and $b_j$ being the bias for items $i$ and $j$ respectively and $w_u, w_i$ and $w_j$ being the weights of user $u$, item $i$ and item $j$ respectively. The model is then trained by the LearnWBPR-MF algorithm, a machine learning algorithm based on stochastic gradient ascent. However the learning algorithm will be left out of the scope of the research here. As an example the algorithm was run on the MovieLens dataset and resulted in the following:

What should be observed here is that the accuracy actually went down comparing to the other BPR-MF algorithms. This can have many reasons, however one might be that the other models actually overfits on the training set, while this algorithm prevents it from overfitting as it is more
regularized. Its prec@5 measure on the contrary is a lot lower than the previous algorithm, indicating that in the case of item recommendations this algorithm will perform worse and even comparing to the ‘most popular’-baseline this is lower. So it should be concluded that this algorithm shows very little potential in the considered context.

**Weighted Regularized Matrix Factorization (WR-MF)**

The weighted Regularized Matrix Factorization is another case of the matrix factorization approach to the recommendation problem. This algorithm is based on an adoption of the Singular Value Decomposition (SVD), which was analysed earlier in this research. It minimizes its optimization criterion for the square-loss loss function, which is calculated by subtracting the predicted rating from the actual rating and squaring it. The underlying optimization criterion used by the developer of MyMediaLite is given by (Gantner, Drumond, Freudenthaler, Thieme, 2012):

\[
\sum_{u \in U} \sum_{i \in I} c_{ui} ((w_u, h_i) - 1)^2 + \lambda \|W\|^2_f + \lambda \|H\|^2_f
\]

In contrast with the before analysed BPR-variants of the matrix factorization group, this algorithm minimizes its optimization criterion as it attempts to make as little errors as possible. The \(c_{ui}\) term represents the weight of every user-item combination. This is calculated based on additional information on the user-item pair for positive feedback and is set to 1 for all other cases. As this model is based on the approach in ‘Collaborative Filtering for Implicit Feedback Datasets’ (Hu, Koren, Volinsky, 2008), a possible approach for this weight is:

\[
c_{ui} = 1 + \alpha r_{ui}
\]

Where \(r_{ui}\) is the observed rating and \(\alpha\) the chosen confidence level. By doing this it accords more value to the positive feedback than it does for the negative feedback cases. When tested on the MovieLens dataset this resulted in the following:

As can be observed the accuracy (AUC) is at 0.9276, which is comparable to the previously tested Multicore BPR-MF algorithm as this produced an accuracy of 0.92823. So one could assume that both of these algorithms optimize the matrix factorization framework quite well, not taking the possibility of overfitting into account. Thus these 2 algorithms are most likely preferable when using it for the item recommendation problem. However a company should before applying a
certain model, understand its data and know what the underlying formulas of the used models are. By knowing this, it can more accurately decide which kind of matrix factorization algorithm will solve the recommendation problem appropriately.

**K-Nearest-Neighbor clustering algorithms (k-NN)**

Another class in algorithms that the item recommendation tool offers is the class of K-Nearest-Neighbor clustering algorithms. For this class it offers 4 variants: ItemAttributeKNN, ItemKNN, UserKNN and UserAttributeKNN. As mentioned before the algorithms use some kind of distance measure, like the cosine measure, to calculate the ‘nearest neighbors’ or the most similar items or users. Within the item recommendation tool it offers both content-based and collaborative filtering-based algorithms. UserAttributeKNN for example uses demographics of its users to provide recommendations, while UserKNN looks more at the implicit feedback given by users. As this application of the k-NN clustering algorithm is for the task of item recommendation it does not make use of aggregation function to predict a rating. Instead it calculates the distance from other items and picks the $k$ closest items.

**Item k-NN and user k-NN**

These algorithms work in the same way, as they both select the $k$ nearest neighbors based on their similarity measures. These similarity measures are calculated by the cosine measure, which was mentioned before in this research. This cosine measure is calculated on the implicit feedback given by its users on items. The difference between the item k-NN and the user k-NN algorithm is that the item k-NN will make recommendations based on the similarities between items that the user gave implicit feedback on before. The user k-NN algorithm however calculates the similarities between users, to then use the given feedback of its nearest neighbors to recommend items. Both of these algorithms were tested on the MovieLens dataset and resulted in the following:

**Item k-NN**

```
Training time 00:00:17.7284014 AUC 0.91738 prec@5 0.18998 num items 1682 num lists 459 testing time 00:00:21.7241720
```

**User k-NN**

```
Training time 00:01:06.5152301 AUC 0.92006 prec@5 0.11519 num items 1682 num lists 459 testing time 00:01:11.4326790
```
Here it can be observed that the accuracy of the user k-NN is slightly higher. However the prec@5 measure, which calculates the precision of the 5 nearest-neighbors, is a lot higher. Thus indicating that making predictions based on the similarities between users results in better recommendations. Comparing to the before analysed matrix factorization algorithms it should be noted that this yields about the same accuracy, thus offering a good alternative for the matrix factorization algorithms.

**Item-Attributes k-NN and user-attributes k-NN**

These variants of the k-NN clustering algorithm are a content-based and model-based approach to the recommendation problem. It clusters items together based on its attribute features, not taking into account the implicit feedback. This usually yields lower precision than the collaborative filtering based recommendations, however it can form a good baseline to compare new recommender systems with. To use this algorithm only the attribute files are required, as it does not take into account the given feedback. However within the MovieLens dataset, this information was not available. Thus it was not possible to test it on this dataset. However one possible application, if the data is available, is to cluster the users together based on their attributes and then only use the for example 10 nearest-neighbors to perform a user k-NN algorithm. Thus it would only take into account users that have similar features (e.g. demographics) and then calculate the similarities between their given ratings. Thus forming a hybrid recommender system.

So as it can be noted, the MyMediaLite library offers a few variants of the k-NN clustering algorithm next to its baseline algorithms and matrix factorization algorithms. These methods have been proven to perform at similar accuracy levels, so can both be considered when applying recommender algorithms to data. However to find the optimal model for recommending items, different algorithms should be tested as some might perform better in certain cases. For example in a case where the dataset is very sparse, thus lacking many ratings for items, a content-based approach is often better. Therefore the attribute k-NN algorithms should be considered, however this also requires information on the items itself.

**Sparse Linear Methods (SLIM)**

Another kind of algorithm that is provided in the item recommendation tool is the Sparse Linear Methods-algorithm. As analysed before this solves the recommendation problem through
solving an $l_1$-norm and an $l_2$-norm. It makes use of the observed ratings or purchases to define profiles for users, which are then used to accord a recommendation score to an item for a user. Based on these observations it creates a $W$ matrix, which contains aggregation coefficients to represent the relationship between a user and an item. It has been proven to yield good accuracy and fast runtime, because of its sparseness.

In the MyMediaLite framework 2 of these algorithms are included, namely: LeastSquareSLIM and BPR-SLIM.

**LeastSquare SLIM**
This variant of the SLIM-algorithm is the original form of the recommender algorithm and the calculation of the $W$-matrix happens through an optimization of the least-squares criterion. It learns the $W$-matrix through a machine learning technique called ‘coordinate descent algorithm with soft thresholding’ (Friendman, Hastien Tibshirani, 2010) to minimize the least-squares function with regularizations for the $l_1$-norm and $l_2$-norm. As it uses soft thresholding to calculate a sparse matrix $W$ it should have a fast runtime, because it is able to learn fast because of the low threshold and can disregard relationships that are not positive as this is one of the constraints of the model. However when tested on the MovieLens dataset it took the system 15 minutes to train the model. Which is very long comparing to the other algorithms: It should also be noted that the accuracy is considerably lower than all of the other algorithms. Therefore it is possible that this algorithm was not effectively coded or that something is wrong with the underlying model. It should also be noted that this is almost the same as the ‘Most Popular’ baseline, yet its prec@5 is considerably higher which shows that it does have some potential in the context of item recommendation since it is about 15% more likely to predict an item that the user will be interested in.

**BPR-SLIM**
The other offered variant of the SLIM-algorithms is the BPR-SLIM algorithm. This algorithm is the same in its underlying model. However instead of the least squared optimization criterion a BPR-variant of this is used to calculate the $W$-matrix. It is then also learned by the learnBPR algorithm. This combination of both the BPR-Opt and LearnBPR results in optimizing this model for ranking,
instead of providing a recommendation score for every single item. When tested on the MovieLens dataset this resulted in the following:

```
training_time 00:00:16.8462320 AUC 0.809042 prec@5 0.430844 num_items 1682 num_lists 459 testing_time 00:00:02.4327881
```

Here it is noticeable that the runtime decreased drastically, thus implying that the BPR optimization and learning technique is more efficient than the original model. However when comparing to the other algorithms, the BPR-SLIM does yield a slightly lower accuracy and precision (except for itemkNN).

**External Item Recommender**

Another feature of the item recommendation tool is the possibility to let an external recommender to be performed within the MyMediaLite framework. A reason for this could be the testing of a newly developed recommender algorithm, as the framework provides instant feedback on its runtime, accuracy and precision. However it is also possible that a user of the framework wants to include a state-of-the-art recommender algorithm, which has not been incorporated in the framework. This recommender-method is then able to link the data to this external recommender algorithm as a file and let it be performed within the framework as it adopts the file’s code. This again shows the extensibility of the framework, as it was designed to be both flexible and extensible, the incorporation of external recommenders is quite simple.

To include an external recommender system, the command should look like this:

```
item_recommendation --training-file=u1.base --test-file=u1.test --
recommender=ExternalItemRecommender --recommender-options="prediction_file=FileName"
```

Where the “FileName”-element links to the recommender system file.

**Summary**

So it can be concluded that the item recommendation tool provides all the necessary algorithms to apply recommender systems on a website, both for development and the actual application. The framework then just has to be linked within the code, which is available on the MyMediaLite-website for the implementation in C#.

Then the output file can be linked to the website so that the items linked to the given item ID’s can be recommended to the users of the website. As an example of this output file the BPR-MF algorithm will be performed and used to recommend 5 items:
Command:

"item_recommendation --training-file=u1.base --test-file=u1.test --recommender=BPRMF --predict-items-number=5 --prediction-file=BPRMF"

The file is then exported to the downloaded library of the MyMediaLite framework and the output for the first 5 users is given by:

1. [56:5.17769,421:5.04213,70:5.000381,171:4.91475,596:4.708562]

Where the first number is the user ID, followed by the 5 recommended items with their respective values. To then use this for actual recommendations these item ID’s will have to be linked to the item lists, however this is the only thing that still has to be done to actually apply the recommendations on a website. Thus we can conclude that the ease with which recommendations are made is rather remarkable.

The overview of the performance of all the algorithms within the item recommendation tool are given in the following table:

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AUC</th>
<th>prec@5</th>
<th>Training Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero</td>
<td>0.05688</td>
<td>0.04924</td>
<td>00:00:00</td>
</tr>
<tr>
<td>Random</td>
<td>0.50574</td>
<td>0.03617</td>
<td>00:00:00</td>
</tr>
<tr>
<td>Most Popular</td>
<td>0.85772</td>
<td>0.322</td>
<td>00:00:00:0085</td>
</tr>
<tr>
<td>BPR-MF</td>
<td>0.92228</td>
<td>0.48061</td>
<td>00:00:05.1861</td>
</tr>
<tr>
<td>MultiCore BPR-MF</td>
<td>0.92797</td>
<td>0.50763</td>
<td>00:00:02.2405</td>
</tr>
<tr>
<td>Weighted BPR-MF</td>
<td>0.87059</td>
<td>0.30806</td>
<td>00:00:03.7748</td>
</tr>
<tr>
<td>WR-MF</td>
<td>0.9276</td>
<td>0.55773</td>
<td>00:00:03.7163</td>
</tr>
<tr>
<td>Item kNN</td>
<td>0.91738</td>
<td>0.38998</td>
<td>00:00:17.7285</td>
</tr>
</tbody>
</table>
By comparing all of the algorithms it can be noted that many of the algorithms yield an accuracy (AUC) of more than 0.9, which indicates that these predictions are very accurate. However some (Weighted BPR-MF, LeastSquareSLIM and BPR-SLIM) are just under the 0.90 mark, which is usually considered as the limit for ‘excellent’ predictions. Additionally it should be noted that these underperforming algorithms’ accuracies are fairly close to the ‘Most Popular’-baseline, indicating that these are not much better than just recommending the most popular items to its users when it comes to the accuracy of their predictions. Nevertheless it should be noted that the BPR-SLIM does yield a prec@5 which is at a level 15% higher than the baseline, which can be seen as a big improvement over the baseline.

Another observation that can be made is that the BPR-based algorithms perform quite well on the prec@5 measure. As prec@5 measures the precision of the 5 highest recommended items, this is quite relevant in the context of recommending items to users. The BPR-variants yield a precision between 43% and 50%, which implies that about 50% of these recommended items are actually effective. Yet it is the WR-MF algorithm that yields the best prec@5, with a 55% precision on the top 5 items. So when recommending 5 items, it can be said that a bit less than half of the recommended items will trigger the interest of the user.

In general it can be concluded that the algorithms offered by the item recommendation tool within the MyMediaLite framework achieve decent to excellent accuracy and yield a very decent accuracy on its top 5 items. As this has been analysed for the purpose of recommending products on a website, these 5 top items are in most cases all the company needs as it will not be recommending a list of tens of items. Therefore this item recommendation framework should be sufficient for the application of a standard recommender system. Additionally it also offers the possibility of adding external recommender systems, so it is also compatible with new developments in the field of recommender systems.
Rating prediction tool
The rating prediction tool provides algorithms, which as the name indicates, will predict the rating of certain items. In the context of recommender systems this can be useful, mostly for companies that do not sell many items. As the item recommendation tool is very fit to provide recommendations in the context of webstores with a very large variety of items. The rating prediction tool is fit to provide expected ratings when only a few items are available. However both of the tools can be used in both cases, but one might consider this approach as more fit. The rating prediction is actually the same in the underlying principles as the item recommendation tool, the difference is that one extra step still has to be done to provide a ranking of the items. This is because the underlying principle in item recommendation is ranking items based on their predicted ratings, however some item recommendation techniques do not use the predicted ratings. For example the BPR-variants of the before mentioned algorithms do not predict ratings, but predict the relative preference of items.

As this research tries to explore the potential of MyMediaLite in the context of the item recommendation problem, the rating prediction tool will not be analysed deeper. The underlying algorithms of this tool have been optimized for the rating prediction problem and not the item recommendation problem and as such it has different variants of the available algorithms. This does not imply that the tool is useless in the context of recommender systems, but this is not the focus of this research. Nevertheless one possible application that could be useful is to test the recommendations given by the item recommendation algorithms within this tool to predict the rating that a user will give for an item. As false positives should be avoided in the context of recommendation, since this causes a drop in the user experience of a website, a verification like this could lower the amount of false positives. Another application of the rating prediction tool could be to calculate the predicted ratings first and then let an item recommendation algorithm learn from these predicted ratings in order to avoid the cold start problem. However using these predicted ratings to train a model will most likely end in a less accurate model than a normal recommendation model, as this optimizes the predicted ratings for the recommendation problem.
Additional features

Next to the available algorithms, the MyMediaLite also offers other features which allow for the applied model parameters to be optimized. This is crucial as models can be improved significantly by finding the proper parameters, as already shown in the BPR context.

Iterations

The iterations in a model define how many sets of training cases will be taken into account before stopping to learn from its training set. The lower the iterations are set, the faster the model will be learned. However what should be noted is that this affects the accuracy of the model, so the optimal amount of iterations should be found before applying a model to real world applications. For this the framework offers a tool:

With the “--find-iter=N”-function developers are able to analyse which amount of iterations results in the best performance. The function shows the performance of the model for every $N$-iterations so that it can be found at what point the performance starts going down. This is usually an indication that the model is overfitting to the data and should stop learning from the training data. As an example this function was applied to the BPR-SLIM algorithm and $N$ was set to 5:

By analysing this it can be observed that between 10 and 15 iterations the model’s performance is at its highest point. However the prec@5 measure does increase from iteration 10 to 15, thus after 15 iterations the model yields the best results. Then another function, “--max-iter=N”, can be used to stop iterating after that number of iterations.
**Recommender-options**

Another available function is the “--recommender-options= ... ”-function. With this function developers are able to change any parameter of the used recommender algorithm. This can be useful for testing and optimizing as this has the potential to improve the performance of the applied model. By then evaluating the model’s performance with every change in parameters, the optimal set of parameters can be found. As an example the BPR-SLIM algorithm was applied and the number of cases within every iteration step was changed from the standard 15 to 5. This resulted in the following:

Iteration-number = 15 (standard)

```
Training time: 00:00:15.2901872 AUC: 0.89168 prec@5: 0.42048 num items: 1602 num lists: 459 testing time: 00:00:02.845158
```

Iteration-number = 5

```
Training time: 00:00:05.3641076 AUC: 0.89414 prec@5: 0.47015 num items: 1602 num lists: 459 testing time: 00:00:02.845187
```

So as can be observed here the performance increased by changing the number of cases within every iteration from 15 to 5. The AUC increased from 0.89168 to 0.89414 and the prec@5 increased from 0.42048 to 0.47015. Thus it would be better to use the iteration-number 5.

This can be done for every parameter of the model and by doing so the optimal combination of parameters can be found. By finding the optimal combination of the parameters the model is then used to its full potential, which results in better recommendations for the users.

**Candidate items**

The last additional feature that will be analysed is the “--candidate-items=FILE”-function. This function is quite useful in the context of recommender systems for e-commerce websites as this function allows the user to define which items should be considered for recommendations. If for example an online web shop has too much stock of certain products, it can define these items in this function. By doing this it is able to calculate which of the items that the web shop wants to get rid of fits most to the needs of its customers.
Evaluation methods
To evaluate the performance of the used models 3 different methods are available within the framework. These were not used in the before given examples, as the MovieLens dataset had already been split into a training set and a test set (80-20). The first evaluation method is “--cross-validation=K”, this splits the dataset into a training set and a test set K-times to test the recommender algorithm. The division of this split is different every time so that it does not overfit on the data. The second evaluation method is a regular “--test-ratio=NUM”-function, where NUM indicates the percentage of the dataset that should be used as the test set. The third and final evaluation method is the “--num-test-users=N”-function. With this function the user has to pick a number for N and then N-amount of random users will be selected to test the model on.

Additionally the framework also provides the possibility for online evaluation through the “-online-evaluation”-function. In the context of recommender systems for e-commerce this is crucial as the model needs to be updated whilst being in use. The online-evaluation function works by treating a user of the website as a test case. It gives recommendations to the user and tests whether the made recommendations are accurate. Then feedback is given and the performance is evaluated, after which the user is incorporated within the training set of the model and an incremental update is performed on the model to improve its performance. As was analysed before in the relevance of recommender systems for e-commerce these incremental updates are crucial to the performance of recommender algorithms. Therefore this should be considered as a very important additional feature.

These testing-methods are then evaluated on 3 measures: AUC, prec@5 and runtime. The first measure, AUC is the most commonly known measure for accuracy. It calculates the ‘Area Under the Curve’, which is based on the ROC curve, which plots the true positive rate on the false positive rate. The AUC calculates the area under that curve, its value is between 0.5 and 1 where 0.5 is the same as randomly guessing and 1 is a perfectly accurate model. The benefit of this measure is that it takes into account the distribution of the underlying data. The second measure prec@5 calculates the precision of the top 5 recommended items. It is calculated by analysing the 5 highest recommended items and checks whether these were given positive feedback. As this evaluation is done for item recommendation, this measure indicates the performance of the 5 highest recommendations, thus should be considered as the most important measure. Finally
the runtime measure indicates how long it took for the model to be learned from the training set, this is valuable as this indicates the scalability of the model.

**Dataset**
The MovieLens dataset that was used to evaluate the MyMediaLite framework was obtained from the Grouplens website ([http://grouplens.org/datasets/movielens/](http://grouplens.org/datasets/movielens/)). The data was collected through the MovieLens website from the 19th of September, 1997 through the 22nd of April, 1998. It contains 100,000 ratings on a scale from 1 to 5 from 943 users on 1682 different movies. Every user that is contained within the dataset has at least rated 20 movies. The data that was used in the examples of the framework was an already split set of training data and testing data, with a division of 80% and 20% respectively.
Conclusion
In this research an analysis has been made of the usefulness of recommender systems for selling products through the internet, also known as e-commerce. Therefore a case study on the biggest e-grocer, Leshop, in Switzerland was analysed which showed that the application of a recommender system in a website has both a direct and an indirect effect on the revenue. The direct effect was shown to produce only a slight increase in sales, however still significant for companies with large turnovers. In contrast it was shown that the indirect effect tends to outperform the direct effect greatly. The indirect effect influences the revenue through general satisfaction with the company’s services as well as through repeat purchases of items that were originally recommended by a recommender system. This influence was then shown to be both durable and significant as it could yield up to an increase of 366% on the direct effect. Thus it could be concluded that the application of recommender systems within the context of online shop had a significant effect.

An analysis of the different classifications within the context of recommender systems has also been made with the addition of the analysis of a few very common recommender algorithms (k-NN clustering, Matrix Factorization and Sparse Linear Methods). In this analysis the variety of recommender algorithms was first indicated as well as the need to adapt the kind of applied recommender algorithm to the context in which it will be used, to make accurate and meaningful recommendations. The main reason for this was the lack of information in some cases, as well as the kind of information that was available From this the MyMediaLite framework was then proposed as an approach to implement recommender systems in an e-commerce website.

The MyMediaLite framework was then shown to be of a good design for the fast-developing field of recommender systems through its flexibility and extensibility. The open source character allows for constant updates on new developments in the field, free of charge for the companies that make use of the framework. Furthermore the license under which the framework was released also allows for commercial applications, thus proving potential for actual applications for the framework within companies.

The framework itself was then analysed on its scalability and proved to be more than adequate to process the information needed for the recommender problems, even with the possibility of parallelizing the different recommender algorithms. It also allows for implement-ready coding so that it could easily be incorporated in other software frameworks. However it also provides easy-
to-use command line tools in which the recommender library could be accessed without any need of deep knowledge of coding. These tools then proved to produce easy-to-export results, which could then automatically be applied to websites making use of the recommender system.

The recommender library for the item recommendation tool was then analysed more deeply on its available algorithms. These proved to produce accurate predictions as well as precision rates on its top 5 recommended items of up to 55%. Additionally other features of the item recommendation tool were analysed and proved to be well fit for the application of recommender systems in an online environment, such as the online-evaluation function. Furthermore it also offered the tools for proper recommender testing and the possibility of altering the parameters of the applied recommender algorithms through the command line tool, again without the need for deep knowledge of coding. Yet with some knowledge of coding recommender algorithms could be developed and then thoroughly tested within the framework. As an example the Bayesian Personalized Ranking framework was analysed and proved to produce good results in many of its applications.

So in general it can be concluded that the MyMediaLite framework does have potential in the context of e-commerce websites. It offers all the tools needed for the application of a recommender system on a website and can thus lead to improved revenues through both a direct and indirect effect. It should be noted however that the direct effect in the case of smaller companies will be of less value, but the indirect effect is substantial for both small and big companies. Additionally the MyMediaLite framework showed good scalability and potential for the development and testing of recommender algorithms and because of its free and open-source characteristics this can be used by anyone. Thus also showing potential for the modification of the recommender algorithms in order to better fit the needs of both the developers and the customers making use of this recommender system.
Bibliography

Articles


Gu, Q., Zhou, J., & Ding, C. Collaborative Filtering: Weighted Nonnegative Matrix Factorization Incorporating User and Item Graphs Proceedings of the 2010 SIAM International Conference on Data Mining (pp. 199-210).


Websites


Gantner, Z. (n.d.). MyMediaLite: BPRSLIM Class Reference. Retrieved August 08, 2016, from http://mymedialite.net/documentation/doxygen/class_my_media_lite_1_1_item_recommendation_1_1_b_p_r_s_l_i_m.html
Gantner, Z. (n.d.). MyMediaLite: BPRMF Class Reference. Retrieved August 08, 2016, from http://mymedialite.net/documentation/doxygen/class_my_media_lite_1_1_item_recommendation_1_1_b_p_r_m_f.html

Gantner, Z. (n.d.). MyMediaLite: ItemAttributeKNN Class Reference. Retrieved August 08, 2016, from http://www.mymedialite.net/documentation/doxygen/class_my_media_lite_1_1_item_recommendation_1_1_item_attribute_k_n_n.html

Gantner, Z. (n.d.). MyMediaLite: UserAttributeKNN Class Reference. Retrieved August 08, 2016, from http://mymedialite.net/documentation/doxygen/class_my_media_lite_1_1_item_recommendation_1_1_user_attribute_k_n_n.html

Gantner, Z. (n.d.). MyMediaLite: UserKNN Class Reference. Retrieved August 08, 2016, from http://mymedialite.net/documentation/doxygen/class_my_media_lite_1_1_item_recommendation_1_1_user_k_n_n.html

Gantner, Z. (n.d.). MyMediaLite: ItemKNN Class Reference. Retrieved August 08, 2016, from http://mymedialite.net/documentation/doxygen/class_my_media_lite_1_1_rating_prediction_1_1_item_k_n_n.html


Gantner, Z. (n.d.). MyMediaLite: WRMF Class Reference. Retrieved August 08, 2016, from http://www.mymedialite.net/documentation/doxygen/class_my_media_lite_1_1_item_recommendation_1_1_w_r_m_f.html

Gantner, Z. (n.d.). MyMediaLite: LeastSquareSLIM Class Reference. Retrieved August 08, 2016, from http://mymedialite.net/documentation/doxygen/class_my_media_lite_1_1_item_recommendation_1_1_least_square_s_l_i_m.html

Gantner, Z. (n.d.). MyMediaLite: MultiCoreBPRMF Class Reference. Retrieved August 08, 2016, from http://mymedialite.net/documentation/doxygen/class_my_media_lite_1_1_item_recommendation_1_1_multi_core_b_p_r_m_f.html

Gantner, Z. (n.d.). MyMediaLite: WeightedBPRMF Class Reference. Retrieved August 08, 2016, from http://mymedialite.net/documentation/doxygen/class_my_media_lite_1_1_item_recommendation_1_1_weighted_b_p_r_m_f.html

http://mymedialite.net/documentation/doxygen/class_my_media_lite_1_1_item_recommendation_1_1_least_square_s_i_l_i_m.html


Figures (in order of appearance)


83


