A Workbench for Comparing Collaborative- and Content-Based Algorithms for Recommendations

Master Thesis

Pat Kläy
from
Böisingen

University of Fribourg
March 2015

Prof. Dr. Andreas Meier, Information Systems, Department of Informatics, University of Fribourg, Supervisor
Dr. Luis Terán, Information Systems, Department of Informatics, University of Fribourg, Assistant
Abstract

Huge amount of information and easy access to it make recommender systems unavoidable. We use recommender systems everyday without realizing it and without knowing what exactly happens. In this thesis, we propose a workbench which aims to help people understand what recommender systems are and how they work. The goal is to have a functional workbench where different Collaborative Filtering and Content Based Filtering algorithms are implemented to let users play around with it. Providing real-world content as books and movies which one can rate and get personalized recommendations makes the workbench an interactive learning experience. Next to the personalized recommendations which can be compared between different recommender system algorithms, error metrics and runtime statistics demonstrate the differences between the algorithms on a numeric and absolute level.
# Contents

1 Introduction and Background .......................................................... 1
   1.1 Collaborative Filtering ......................................................... 2
       1.1.1 User-User .............................................................. 3
       1.1.2 Item-Item ............................................................ 4
   1.2 Content Based Filtering ....................................................... 5
   1.3 Evaluating Recommender Systems .......................................... 6
       1.3.1 User Centered Evaluation ........................................... 8

2 Aims and Objectives ........................................................................... 9
   2.1 Workbench ................................................................. 9
   2.2 Algorithms .............................................................. 11
   2.3 Datasets .............................................................. 11

3 Implementation .............................................................................. 13
   3.1 Basic Structure .......................................................... 14
       3.1.1 Content .............................................................. 15
       3.1.2 Explanation Pages ................................................ 16
       3.1.3 Statistics and Metrics ......................................... 17
       3.1.4 Registered Users ............................................... 21
   3.2 Datasets ............................................................................. 23
   3.3 Collaborative Filtering ........................................................ 26
   3.4 Content Based Filtering ..................................................... 30
       3.4.1 Create TF-IDF Vectors ........................................... 30
       3.4.2 Calculate Similarity ................................................ 31

4 Outlook ....................................................................................... 35

5 Conclusion .................................................................................... 37

A Explaining the File Structure ........................................................ 47
Introduction and Background

In today's information technologies, recommender systems are essential to handle the huge amount of data we have to deal with. We find recommender systems in search engines, dating sites, social networks, news sites and of course in nearly every web shop. They recommend us friends, articles, books, music, movies, jokes and even travel destinations [10][1]. Historically there are two basic types of recommender system [6]: Content Based Filtering and Collaborative Filtering. However, in the past few years, a lot of research was done in the area of recommender systems so today there are multiple approaches such as Hybrid Recommender Systems [2], Knowledge Based Recommender Systems [11] and Graph Based Recommender Systems [5]. In the following sections, Collaborative and Content Based Filtering are explained in more detail as they are important for this thesis.
CHAPTER 1. INTRODUCTION AND BACKGROUND

1.1 Collaborative Filtering

In Collaborative Filtering the idea is to learn form the past to predict the future. User profiles (such as purchase history, movies ratings or visited news sites) are analyzed to find items which fits the users needs best. The user profile can be generated in two ways:

- Explicit
  The user actively rates content as for example how much he liked a movie, news article or song

- Implicit
  Information about the users behavior is collected as for example how long he stayed on a specific page, how long he actually listened a song or simply that he bought a specific item

In state of the art recommender systems, a combination of both approaches is used to optimize the recommendations. Collaborative Filtering has the advantage that no knowledge about the items in the catalog is required at all. Not for the user profile generation and neither for the recommendations itself. It is sufficient to have the data and the algorithms to recommend items to a user. However there are also some challenges and drawbacks. Concerning the challenges there is the privacy issue. When collecting data explicitly the user might guess that this data may be used for recommendation purposes and he can therefore decide not to rate anything. But if the data is collected implicitly, the user cannot decide if he wants to ”contribute” to the recommender system. Often the modern recommender systems do not even require a user to be logged in but collect implicit information based on the IP address to generate a user profile. On these systems the privacy concerns are even bigger. To overcome this problem, research is done in how people trust recommender systems and how this trust can be enhanced [4].

Another challenge in Collaborative Filtering systems is the data sparsity. A system easily contains thousands of items and millions of users. A overlapping subset of user/item pairs might be hard to find and therefore it is difficult to make accurate recommendations. The recommender algorithm must be aware of this problem and try to handle it. But the major drawback in the Collaborative Filtering approach is the cold start problem. When a user newly signs up, the system cannot recommend him anything as there is no user profile yet. It needs time and actions from the user to finally recommend
1.1. COLLABORATIVE FILTERING

him items he could be interested in. That is why state of the art recommender systems often combine Collaborative and Content Based Filtering.

There are two types of Collaborative Filtering which are described in the two following sections.

1.1.1 User-User

User-User based Collaborative Filtering recommender algorithms analyze the similarity between the different users in the dataset. To find an item for the target user, the algorithm forms a neighborhood for it. Users close to the target user are similar, users far away from it are not. Then the algorithm can analyze the similar users good rated items and check which one the target user has not seen yet. If one or more items are found, they can be recommended to the target user.

In the following example - see Table 1.1 - we try to predict how much Bob might like Item 3. So the n most similar users to Bob are selected and their ratings of Item 3 evaluated. Based on these information, it is possible to predicts Bob’s rating for Item 3. For simplicity reasons, the example will use only one (close) User of Bobs neighborhood. Table 1.1 shows that User 1 is pretty similar to Bob. The Items Bob liked were rated as ”good” by User 1. The items which Bob did not like were rated as ”bad” by User 1. So these two users can be seen as similar. User 1 is therefore a good starting point to decide how much Bob might like Item 3. As User 1 liked Item 3, we can infer that Bob will also like Item 3. A predicted rating would be somewhere between 4 and 5 depending on the algorithm which is used for the prediction. As described earlier, normally more than one user of Bobs neighborhood is evaluated to refine the prediction. In this example we could for example have a look at Alice. Alice is also similar to Bob (unfortunately

<table>
<thead>
<tr>
<th>User</th>
<th>Item 1</th>
<th>Item 2</th>
<th>Item 3</th>
<th>Item 4</th>
<th>Item 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>?</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>User 1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>User 2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Bob</td>
<td>5</td>
<td>2</td>
<td>?</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>User 3</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 1.1: User-User Collaborative Filtering
we do not have that much common items between Bob and Alice, that is why we should not rely too much on this comparison) and Alice also liked Item 3. So the fact that Bob might like Item 3 can be secured.

1.1.2 Item-Item

Table 1.2: Item-Item Collaborative Filtering

<table>
<thead>
<tr>
<th>User</th>
<th>Item 1</th>
<th>Item 2</th>
<th>Item 3</th>
<th>Item 4</th>
<th>Item 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>?</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>User 1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>User 2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Bob</td>
<td>5</td>
<td>2</td>
<td>?</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>User 3</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Item-Item based Collaborative Filtering recommender algorithms are very similar to User-User based algorithms. Instead of analyzing the similarity between the different users in the dataset, the Item-Item algorithms analyze the similarity of the different items in the dataset. Apart from this difference Item-Item algorithms behave exactly as User-User algorithms. To check if a target item could be recommended to a user, a neighborhood for this target item is generated where close items are similar to the target item. Then those close items are analyzed if the user liked them or not. If yes, the target item can be recommended to the user. Or the other way around: An neighborhood for an item the user strongly liked is formed. Close / similar items can be directly recommended to the user as he might like them as well.

In the following example - see Table 1.2 - we try to predict how much Alice might like Item 1. So the n most similar items to Item 1 (which Alice has already rated) are selected and Alice’s ratings for these items are evaluated. Based on these information, it is possible to predict Alice’s rating for Item 1. For simplicity reasons the example will only use one (close) Item of Item 1’s neighborhood. In Table 1.2 we can see that Item 1 is similar to Item 5.

- The users which liked Item 1 (User 1 and Bob) also liked Item 5.
- User 2 did not like both of them.
1.2. CONTENT BASED FILTERING

- User 3 could not really decide whether or not he liked those two items. It is a neutral rating for both items.

Generally we can say that users who liked Item 1 also liked Item 5 and users who didn’t like Item 1 neither liked Item 5. So we can say that these two items are similar (Item 5 is a very close neighbor of Item 1). As Alice liked Item 5, we can assume that she will also like Item 1, a predicted rating would be somewhere between 4 and 5 depending on the algorithm which is used for predictions. Also for the Item-Item algorithms normally more than just one neighbor is analyzed. If we have a look at Item 3, we can see that it is also pretty similar to Item 1 and Alice also liked Item 3. Again, the fact that Alice might like Item 1 can be secured.

As you can see, Item-Item and User-User algorithms really have a lot in common. In fact, there is no difference at all if you invert the matrix of the User-Item ratings.

1.2 Content Based Filtering

In content based filtering the idea is to match (and recommend) similar items based on their content. The content information can either be extracted automatically or has to be added manually. Automatic extraction often is the case for news sites, search results etc. where text is the main content. Analyzing text can be done automatically and no further information has to be added (even though it is of course possible to enrich the text with tags or other information to improve the results). Content information which needs to be added manually can normally be found for books, movies and this kind of items where the algorithms do not have the possibility to analyze the content automatically. Taking the movies for example, one would add the genre, actors, director, studio and so on to identify the movie better. The algorithms will then work based on this information. Content Based Filtering algorithms can only be applied to items (there are no User-User algorithms) but they do not require a user profile to form a neighborhood for target items. Even in a completely empty and new system, the algorithms are able to identify similar items by just having a look at their content. Also system which do not collect any user data can therefore implement a recommender system. As an example we can think about a news site. If an anonymous user reads an article about politics in the middle east, the system can recommend the user similar articles (about politics, or something else in
CHAPTER 1. INTRODUCTION AND BACKGROUND

the middle east). This is of course not a personalized recommendation, but there are no privacy issues and no could start problem as there is with Collaborative Filtering algorithms. However, a user profile can lead to personalized recommendations (in a book store for example). If a user bought three books from J.R.R Tolkien, the system can recommend other books written by J.R.R Tolkien to the user. In such systems the cold start problem also plays its role, the impact however is not the same. If a new book is available in the store, it can be immediately be recommended to users who bought similar books. In systems which use Collaborative Filtering the book can only be recommended when users actually bought (and preferably rated) the book. The major drawback with Content Based Filtering algorithms is, that in most cases the content information has to be entered manually (as for example "this book is written by J.R.R Tolkien"), which is a very time consuming and expensive task.

1.3 Evaluating Recommender Systems

Deciding which recommender system to use in a given application is a very hard task. Stick to state of the art Content Based Filtering or Collaborative Filtering, User-User or Item-Item, combine them, try a new rather experimental approach? The possibilities are sheer endless. How to take the ”best” recommender system for the task it has to do. That is where you want to evaluate all those recommender system. Evaluating recommender algorithms is not an easy task for the very same reasons as described above: there are simply too many objective functions. Since recommender systems are widely used and there are a lot of them available, ”there needs to be something that determines which one to use”[6]. The first and easiest method to get an idea how the different recommender systems perform is to use evaluation metrics. A dedicated dataset set is split into a test and train set where the recommender algorithm uses the train set to predict the ratings from the test set. The difference between the test set and the algorithms output is then compared. There are two basic metrics types:

- Accuracy of prediction
  Evaluation metrics as Mean Absolute Error (MEA) or Root Mean Squared Error (RMSE) evaluate the algorithms in term of their prediction accuracy. For the formulas see page 18ff.
1.3. EVALUATING RECOMMENDER SYSTEMS

- Accuracy of rank

Evaluation metrics as Mean Reciprocal Rank (MRR) or Normalized Discounted Cumulative Gain (NDCG) evaluate the algorithms in terms of how accurate their predictions are in respect to the rank of the items (the prediction score is ignored, only the rank is important). For the formulas see page 18ff.

One must be careful when comparing recommender algorithms using error metrics. Given a recommender system A with MAE of 1 on ratings on a scale from 0 to 5 is still better than a recommender system B with MAE of 0.5 on ratings on a scale from 0 to 2. Even if B has the "much better" MAE value, A performs better when considering the relative error given by the rating scale. As A has an average error of 1 this is only an error of 20%. B has an average error of "only" 0.5, but relative it is an error of 25%. The same applies to RMSE, so evaluating algorithms with these metrics requires some caution to not wrongly judge them.

But accuracy is not everything. The best example is given in [7]. Consider the following recommender: In a supermarket you place a sign which says "Today you should buy bananas and bread". You examine how many people actually bought bananas or bread and you will find out that your recommender is highly accurate. Another shop just across the street however sells the same amount of bread and bananas. So the recommender is accurate but not really good as it did not increase the sales and did not make people reach the whole catalog of the supermarket. There is a metric called coverage to catch this. It is defined as "Coverage is the measure of the percentage of products for which the recommender can make a prediction"[7]. So this metric measures which parts of the catalog in the system can actually be predicted and recommended, and which parts cannot. The fraction then tells how good the recommender algorithm covers the catalog.

Another example, again taken from [7], which shows that pure accuracy alone does not make the perfect recommender. If a user bought and liked the books The Hobbit and Silmarillion from J.R.R Tolkien, the recommender system will recommend this user to buy The Lord of The Rings I, The Lord of The Rings II, The Lord of The Rings III, The Lord of The Rings Collection, The Lord of The Rings I Soft Cover ... Technically this top-n list is very accurate as the user will definitely like the The Lord of The Rings books. However it is recommending the user n times the same books. The diversity metric,
CHAPTER 1. INTRODUCTION AND BACKGROUND

defined as "Diversity is the measure of how different the recommended items are"[7], addresses exactly this problem and allows to increase the diversification of the top-n list. There is a process which uses content based approaches or clustering to measure the similarity of the items in the top-n list. An item which is similar to a previous item in the list will be replaced by an item which is not similar to any item in the top-n list. This way, the above mentioned problem that the recommender system recommends the same item multiple times can be reduced.

1.3.1 User Centered Evaluation

Having the matrix of all the possible recommender systems with all the different metrics may help to decide which recommender performs best, but in the end it is the user which decides if he accepts the systems recommendation. User are complex and user are different. Do they like to have perfect accurate results? Or do they accept results which are not perfectly accurate but more diverse? [7] All these question can only be answered with User Centered Evaluation. There is a big list of techniques for User Centered Evaluation. Inspecting application logs can give you a first impression if users accepted the recommendations or not. It is a cheap method which does not require a big effort. On the other hand you do not know why the user accepts or does not accept the recommendation. Other techniques as surveys or controlled lab experiments require a lot more effort and are quite expensive, but there you get a direct feedback from the user why exactly he did not accept (or did accept) the recommendation, if he trusts the system, how he likes it overall, if he prefers divers or accurate top-n lists and so on. Research shows that user tend to accept top-n lists which are not perfectly accurate but have a bigger diversity [12].
Aims and Objectives

The goal of this project is to create a workbench which helps students (and other interested persons) to understand what a recommender system is and how it could be implemented. Understanding recommender systems will also help to improve to trust these systems, use them and accept their recommendations. The more one knows about it, the less magic it seems to be. As described earlier recommender systems are nowadays spread widely that is why trusting those systems is relevant.

This workbench will also be used in the eBusiness course at the University of Fribourg\(^1\).

2.1 Workbench

The workbench finally will be a web application where one is able to explore the world of recommender systems. Obviously the workbench contains a lot of explanation pages.

\(^1\)http://recsys-wb.isproject.ch
CHAPTER 2. AIMS AND OBJECTIVES

which explain the different types of recommender systems (see chapter 1) and the different similarity algorithms. Besides that the workbench also contains pages which explain how to evaluate a recommender system and why this is necessary and useful. For interested readers there should also be a section with collected articles for each topic so one can even read more about recommender systems. These explanation pages must be accessible for everyone. Of course any registered user (much likely a student) but also anonymous users from all over the world can access these pages and learn about recommender systems. Only the papers must be handled differently. While a registered user can directly access it, an anonymous user will be redirected to the papers original page. The anonymous user then has the possibility to either buy the paper, or read it if he has access through his home university. This way we can make sure that no one accesses a paper to which he is not allowed to. Furthermore there will be different entities (for example movies) which can be explored. These entities are also open to the whole world, but a registered user will have the possibility to rate the entities and therefore get its personal recommendations. To achieve this, the workbench has to provide different recommender system algorithms (for detailed objectives see section 2.2). In order to be able to tell any difference between the algorithms, some basic statistics and evaluation metrics will be shown to the users (registered and anonymous). This gives the possibility to analyze and compare the different algorithms, get their advantages and weaknesses. To be able to do so, the workbench must provide a lot of data to overcome the cold start problem and show already from the beginning some statistics and evaluations (for detailed data objectives see section 2.3).

The most important difference between registered and anonymous user however is, that the registered user can interact with all the different recommender systems and similarity algorithms. As the registered user will be able to rate the entities provided by the workbench, he will also be able to get its personalized recommendations based on its ratings. The list of recommendations can then be compared to recommendations generated by another algorithm. A registered user will therefore not only be able to learn theoretically what a recommender system is, he will also be able to see and work with them. This offers another way to experience the topic. Getting different personalized recommendations shows best that there are (sometimes significant) differences between the similarity algorithms even if the input (the users own ratings) is exactly the same.

Last but not least, the workbench should be the place where students can complete
2.2 Algorithms

To generate the recommendations from the users ratings, the workbench needs - as already mentioned - to provide different recommender system algorithms. The main idea is, that there are algorithms for Collaborative as well as for Content Based Filtering. In order to be able to compare the algorithms, at least three algorithms must be provided. The idea is that we have for Collaborative Filtering

1. A User-User based algorithm
2. A Item-Item based algorithm
3. A fuzzy based algorithm

and for Content Based Filtering

1. A TF-IDF based algorithm
2. A Naive Bayes based algorithm
3. A fuzzy based algorithm

2.3 Datasets

As mentioned in section 2.1, the workbench contains entities like movies and books which can be rated. Therefore appropriate datasets must be present which already have a lot of ratings. We cannot use empty datasets as we need to provide statistics and analyses from the very beginning. Another point is, that also the first user which rates some content on the workbench should be able to get its recommendations. This is only possible if there is already some data in the back end (cold start problem in Collaborative Filtering).
The datasets will be split into 21 non overlapping subsets. There will be 10 groups working in the eBusiness course per semester, in order to not reuse the datasets in two consecutive semesters we need 20 subsets for the student groups. One dataset will be used (obviously) for the workbench itself. If users rate content they will rate it on this dedicated dataset. All these 21 subsets will be again split up into a train and test set for evaluation purposes. So the selected datasets for the entities must be big enough to fulfill these requirements.
3

Implementation

When one has to implement a web-application there are a lot of possible frameworks and programming languages available. J2EE (java), ruby-on-rails (ruby), django (phyton) or Zend (PHP) to name only some of them. Another possibility is to use a Content Management System (CMS) which already provide an empty web site where you simply have to add the content. We decided to go this way and use the Drupal CMS. The decision for a CMS was made by the fact that (as stated in section 2.1) we want to have real entities like books and movies which the user can explore and rate. So the application logic itself is not very complex but there will be a lot of content. It seemed to be better to use a CMS than start from scratch. Drupal was chosen because it is highly flexible and extensible as well as it provides already a lot of eShop and recommender modules out of the box. Another reason for using Drupal is a bachelor thesis project in the Information System research group at the University of Fribourg which implements a fuzzy based recommender system algorithm as Drupal module. If the workbench is implemented using Drupal, the integration of this algorithm can be done easily. Technically the
workbench is a Drupal module (called recsys_wb\(^1\)) which depends on the following Drupal modules:

- **Drupal Core Modules\(^2\)**
  The Drupal Core installation for the main functionalities of Drupal

- **Recommender API\(^3\)**
  The Recommender API allows to easily calculate recommendations based on a given dataset

- **MathJax\(^4\)**
  The MathJax tool allows to render \(\LaTeX\) in HTML code

- **Login as Other\(^5\)**
  The Login as Other module provides a block from which you can directly login as another user (without having to logout first)

- **Scheduler\(^6\)**
  The Scheduler module allows to schedule the publication date of new content on a specific date and time

The whole workbench is implemented based on these modules and their dependencies.

### 3.1 Basic Structure

As an anonymous user the workbench offers, as described in section 2.1, three different areas which are described in sections 3.1.1, 3.1.2 and 3.1.3. The registered users will additionally have access to their user profile where they can see their ratings and get personalized recommendations. The additional features for registered users are described in detail in section 3.1.4.

---

\(^1\) Source code available at https://github.com/patклаey/recsys_wb
\(^2\) https://www.drupal.org/project/drupal
\(^3\) https://www.drupal.org/project/recommender
\(^4\) https://www.drupal.org/project/mathjax
\(^5\) https://www.drupal.org/project/login_as_other
\(^6\) https://www.drupal.org/project/scheduler
3.1. BASIC STRUCTURE

3.1.1 Content

The workbench contains three different content types: Books, Movies and Stack Overflow questions (see section 3.2 for detailed information about the data itself). Each of these content types is modeled with a dedicated Drupal Content Type to match the data it has associated. On the live site, the data is pretty formatted using the Display Suite module\(^7\). Figure 3.1 shows how a book is displayed to an anonymous user with the help of the Display Suite module.

\[\text{Figure 3.1: A book displayed using the Display Suite module}\]

The Display Suite module is necessary as the book dataset has the an thumbnail field which is the URL to the actual image. So the module can be used to create a new Code

\(^7\text{https://www.drupal.org/project/ds}\)
CHAPTER 3. IMPLEMENTATION

Figure 3.3: A question displayed with ”Similar Questions” block on the right

Field for the content type book which fetches the thumbnail from the given URL to display it. For the movies (see Fig. 3.2) and questions (see Fig. 3.3) the Display Suite module was not needed as they do not have data that needs to be preprocessed. Both registered and anonymous users can browse and view all of those three content types.

The books and movies are used together with the CF algorithms (see section 3.3), that is why an anonymous user can only see the content. The questions are used together with the Content Based Filtering algorithm. As mentioned in section 1.2, Content Based Filtering methods do not require a user profile to work on, so even anonymous users see the block on the right side which indicates similar questions.

3.1.2 Explanation Pages

The explanation pages teach the visitors of the workbench the different recommender system methods (CF and CBF), the different similarity metrics for CF (for an example see Fig. 3.4) and how to evaluate recommender systems. These pages require the MathJax module to properly render \( \text{LaTeX} \) in the HTML code to nicely display all the different
3.1. BASIC STRUCTURE

3.1.1 Cosine Similarity

For the cosine similarity the products or users are seen as vectors in n-dimensional space. The similarity of two products or users (vectors) is given by the angle that they form:

\[
\text{Cosine}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| \times |\vec{y}|}
\]

The result will be between 0 and 1 (as normal for a cosine) where 0 indicates no similarity and 1 indicates absolute similarity.

Let’s have a look at a concrete example. Take the table below:

<table>
<thead>
<tr>
<th>User</th>
<th>Item 1</th>
<th>Item 2</th>
<th>Item 3</th>
<th>Item 4</th>
<th>Item 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>?</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>User 1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>User 2</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>Bob</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>User 3</td>
<td>3</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

The similarity between User 1 and Bob is:

\[
\text{Cosine}(\text{User}1, \text{Bob}) = \frac{4 \times 5 + 1 \times 2 + 2 \times 3 + 4 \times 4}{\sqrt{4^2 + 1^2 + 2^2 + 4^2} \times \sqrt{5^2 + 2^2 + 3^2 + 4^2}} = 0.98
\]

As we can see a value of 0.98 means that there is some huge similarity between User 1 and Bob.

To apply cosine similarity to items the process is exactly the same. You just compare the ratings of the given products instead of the ratings of the given users (just invert the matrix).

Figure 3.4: Explaining user-user based cosine similarity with an example

From these explanation pages there is a link to a special page which recommends the user different papers to read more about the above mentioned topics. Interested users can therefore read and learn more. The papers are directly available for registered users, anonymous users are redirected to the papers page for permission checks.

3.1.3 Statistics and Metrics

To visualize the differences between the recommender algorithms and similarity metrics, the workbench provides several error metrics and runtime statistics. These metrics and statistics are displayed on dedicated pages on the workbench. Again registered and anonymous users have the possibility to access those pages, select which algorithms they want to compare and directly see their advantages and weaknesses. The workbench provides the error metrics listed below. From Equation 3.1 to Equation 3.4 the following
definitions apply:  
\( T \) represents the whole test set, \( \text{pred}(r, t) \) is the prediction of the algorithm \( r \) for test-item \( t \) and \( \text{rating}(t) \in T \) is the actual rating of the test-item \( t \). A test-item \( t \) is unique as it represents the user-item combination, therefore \( \text{rating}(t) = \text{rating}(u, i) \) which is the rating of user \( u \) on item \( i \).

- **Mean Absolute Error (MAE)**
  This metric determines, by how much the average prediction is wrong. If the user rated the item as 3.5 but the algorithm predicts 4, it is wrong by 0.5. All these errors are averaged which gives the MAE.

\[
M E A(r) = \frac{\sum_{t=0}^{T} |\text{pred}(r, t) - \text{rating}(t)|}{|T|} \tag{3.1}
\]

Equation 3.1: Mean Absolute Error

- **Rooted Mean Squared Error (RMSE)**
  This metric is quite similar to the MAE. It also determines by how much the average prediction is wrong but it penalizes big errors over small ones.

\[
RM SE(r) = \sqrt{\frac{\sum_{t=0}^{T} (\text{pred}(r, t) - \text{rating}(t))^2}{|T|}} \tag{3.2}
\]

Equation 3.2: Rooted Mean Squared Error

- **Mean Reciprocal Rank (MRR)**
  The Mean Reciprocal Rank metric determines how good the recommender algorithm is in putting good stuff first [7].

\[
M R R(r) = \frac{1}{j} \tag{3.3}
\]

Equation 3.3: Mean Reciprocal Rank
Where \( j \) is the rank of the first item considered as good in the list of predictions produced by the recommender algorithm \( r \).

- Normalized Discounted Cumulative Gain (NDCG)
  The Normalized Discounted Normalized Discounted Cumulative Gain metric determines how correct the algorithms predictions are in respect of the rank the items appear in. It compares the ranked list of the recommender algorithm with an perfectly sorted list of the items [7]. The result of this comparison will be smaller or equal to 1 where a perfectly sorted list will get 1. So the closer to 1 the result is, the better the algorithm sorts the items.

\[
NDCG(r) = \frac{DCG(r)}{DCG(L_{\text{perfect}})}
\]

Equation 3.4: Normalized Discounted Cumulative Gain

Where

\[
DCG(r) = \sum_{t=0}^{\lfloor T \rfloor} \text{rating}(t) \times \text{discount}(t)
\]

\[
discount(t) = \frac{1}{\max(1, \log_2(\text{rank}(t)))}
\]

and \( L_{\text{perfect}} \) is a perfectly sorted item list by users ratings.

Figure 3.5 shows how the error metrics are displayed in the workbench. As one can see, there is a link for the algorithms itself and for the error metrics which takes the user to the corresponding explanation page.

As there were no Durpal modules or other code available which implements the needed metrics, they were implemented in pure PHP. Listing 1 shows the code for the MAE implementation.

```
function meanAbsoluteError( $results ) {
    $total_diff = 0;
    $total_count = 0;
    foreach ($results AS $result) {
        $diff = $result['rating'] - $result['score'];
        $total_diff += $diff;
        $total_count += 1;
    }
    return $total_diff / $total_count;
}
```
CHAPTER 3. IMPLEMENTATION

Figure 3.5: Error metrics for item-item based Collaborative Filtering algorithms

```php
$total_diff += abs($diff);
$total_count++;
}

if ($total_count == 0 )
    return 0;

return $total_diff / $total_count;
```

Listing 1: MEA implemented in PHP

The runtime statistics on the other hand include the number of users, number of items, number of similarity records calculated, number of predictions calculated and the calculation duration. These metrics are provided by the Recommender API Drupal module (except calculation time this statistic is provided by the Async Command Drupal module which is a dependency of the Recommender API module). The user has the possibility, as it is the case for the error metrics, to compare two or more algorithms at a time. Again the link to the explanation page of the algorithm is provided. For the runtime statistics there is also the option to display the history of a given algorithm. For example after the user rated a few items and started the recommender algorithm (see

---

9https://www.drupal.org/project/async_command
3.1. BASIC STRUCTURE

Chapter 3.1.4, one can compare the different number of users and how it influences the number of predictions and similarity records as well as the run duration. Figure 3.6 shows the history of the runtime statistics for item based Collaborative Filtering with cosine similarity.

![Figure 3.6: Runtime statistics with history](image)

### 3.1.4 Registered Users

The benefit of being a registered user on this workbench is simply that one cannot simply read about recommender systems but really interact with and experience them. As already mentioned earlier, a registered user can browse the content and rate it. The Fivestar Drupal module\(^\text{10}\) could not be used as this module depends on the Voting API module which does not comply with the rating format of our datasets. That is why the voting was implemented using simple HTML and CSS.

As Fig. 3.7 illustrates, there is a small block on the right side of the content which allows the user to rate from \(\frac{1}{2}\) to 5 stars in \(\frac{1}{2}\) star steps. Based on these ratings, the user can get its personalized recommendations. He simply has to start the calculations (he needs to select which algorithm to run) and can then get his recommendations. When retrieving these recommendations, the user can choose from which algorithm he wants

\(^{10}\text{https://www.drupal.org/project/fivestar}\)
CHAPTER 3. IMPLEMENTATION

Figure 3.7: A user can rate content scaled $\frac{1}{2}$ to 5 stars ($\frac{1}{2}$ star steps) to get the recommendation and whether he is interested in a top-n list or if he wants to get everything which has a prediction score higher than a given threshold (see Fig. 3.8).

Figure 3.8: Select which type of recommendations the user wants to get

For a really good feeling what it is about and that there are differences in all these different algorithms, a registered user can directly compare its personalized recommendations. He just has to select an algorithm and type (top-n or score) and retrieve its recommendations. After that he can select a second algorithm and the results will be displayed next to each other. In Fig. 3.9 we can see the comparison between two user based CF algorithms on movies. One used the cosine similarity method and the other one used the euclidean similarity method.
3.2. **DATASETS**

One can clearly see the obvious differences between those two algorithms. Even if the base data is exactly the same, the outcome is quite different. Not only there are different movies recommended (only 3 movies appear in both top 5 lists) also the prediction score is different.

As a last thing, the registered users have access to the project space in which they can sing up to the project of their choice. Mentioned in section 2.1 the workbench should also be the place where students can accomplish tasks and improve recommender systems.

### 3.2 Datasets

As mentioned in section 3.1.1 there are three different content types: Books, Movies and Questions. The movies dataset is provided by the MovieLens project\(^{11}\) and is freely available. Also freely available is the book dataset which is based on Cai-Nicolas Ziegler et al\(^{12}\). The questions dataset, also freely available, is the dump of the stack exchange data\(^{12}\) where the stack overflow questions were extracted. Movies and Books are used for Collaborative Filtering and therefore contain a dataset with the content (books and movies) and a dataset with the ratings. Questions are used for Content Based Filtering

---

\(^{11}\)[https://movielens.org/](https://movielens.org/)

\(^{12}\)[https://archive.org/details/stackexchange](https://archive.org/details/stackexchange) **Dump date:** 2014-09-26
and therefore just contain the content dataset. The main idea to provide these datasets to the registered users is simply that they are able to implement their own solution for a given problem. As these registered users will be mainly students, it was decided to create 21 datasets which all contain a subset of every dataset. It is 21 because there will be 10 groups working in the class and the datasets should not be the same in two consecutive semesters. One distinct dataset is needed as demo dataset, this is the one which is used for the workbench itself. Each of these 21 datasets contain a subset of the movie ratings, a subset of the book ratings and a subset of the questions. These subsets are partly overlapping. Again, each of these subsets is divided into three different sizes: small, medium and large. But this time, the subsets are fully overlapping (the small dataset is part of the medium and large). Finally the movie and book rating datasets (all sizes) are split into a train and test set. To summarize each of these 21 datasets contains:

- Movie ratings
  - Small test
  - Small train
  - Medium test
  - Medium train
  - Large test
  - Large train

- Book Ratings
  - Small test
  - Small train
  - Medium test
  - Medium train
  - Large test
  - Large train

- Questions
3.2. *DATASETS*

- Small
- Medium
- Large

The decision to split them into a small, medium and large dataset each was simply the huge amount of data which is available. The large movie rating train dataset for example still contains over 1.9 million ratings. Playing around with your new recommender system would be very annoying as it takes hours to calculate recommendations with this amount of data. So at the beginning the students can play around with the small and medium datasets, evaluate their algorithm, improve it and evaluate it again. As soon as they are happy with their solution, they can test it on the large dataset to see its performance and to get a better evaluation. The questions dataset does not have any ground truth and can therefore not be split into train and test set. To check if your algorithm works and gives the expected output it is still handy to have a smaller dataset to work on and then move on to a bigger one. The movie rating datasets are SQL-Dump files where the rows are user ID, movie ID, rating (0.5 to 5; step size 0.5) and timestamp. The book rating datasets are also SQL-Dump files where the rows are user ID, book ID, ISBN and rating (1 to 10; step size 1). To have a meaningful evaluation, the book ratings for the demo dataset were adjusted to 0.5 to 5 (step size 0.5) to match the movie rating scale. The question datasets are simply CSV files where the columns are question ID, title, body, score, creation date, view count, answer count, accepted answer ID and tags.

A book and movie content dataset is also provided but they are not part of each of the 21 datasets. Both content datasets are CSV files where a movie has the following attributes: ID, title, year of publication and genre (multivalued, comma separated). Next to the movies there is also a tag data file which stores the user defined tags which were associated to movies. This information is not used for the implementation of the workbench and the demo algorithms but it might be useful for the students to enhance their algorithms. A book on the other hand has the attributes ID, ISBN, title, author, year of publication, publisher, thumbnail url and image url. Next to the book dataset, there are also the user information stored as CSV file. Additional to the user id, there is the users origin and age. Again this information is not used on the workbench but might help to improve a recommender system.
CHAPTER 3. IMPLEMENTATION

3.3 Collaborative Filtering

The Collaborative Filtering algorithms were implemented using the Drupal Recommender API module. This module allows to simply configure the desired Collaborative Filtering Recommender System. All one needs to do is to indicate the database tables in which the algorithm can find the user ratings and choose the type and similarity method to use (for example item-item with cosine similarity). The configuration of all the supported algorithms is done as soon as the recsys_wb is enabled. The modules hook_enable()\(^{13}\) function is used. The whole configuration is stored in a PHP array which then can be passed to the recommender_app_register(ARRAY) function provided by the Recommender API module. Adding a item-item based Collaborative Filtering algorithm with cosine similarity to your module is thus as easy as Listing 2 shows.

```php
function recsys_wb_enable() {
    $apps = array(
        'book_rec_i2i_cosine' => array(
            'title' => t("I2I book recommender (cosine)")
        ),
        'params' => array(
            'algorithm' => 'item2item',
            'table' => '{Book_Rating_demo_train}',
            'fields' => array('UserID','BookID','Rating'),
            'similarity' => 'cosine',
            'preference' => 'score'
        ),
    );
    recommender_app_register($apps);
}
```

Listing 2: Adding a Collaborative Filtering algorithm to the recsys_wb Drupal module

As one can see, the $apps array can hold multiple algorithm definitions. Each algorithm definition is an array which contains a title and a set of parameters:

- **algorithm**
  
  The algorithm type, either 'item2item' or 'user2user'. According to the document-

\(^{13}\)See [https://api.drupal.org/api/drupal/modules!system!system.api.php/function/hook_enable/7](https://api.drupal.org/api/drupal/modules!system!system.api.php/function/hook_enable/7) for more information
3.3. **COLLABORATIVE FILTERING**

There would also be 'item2item\_increment' and 'svd', but unfortunately these two types have a bug in version 7.x-4.0-alpha14.

- **table**
  Indicates the database table in which the user ratings can be found

- **fields**
  An array of column names which indicate the user, item and preference (in this order).

- **similarity**
  Indicates the similarity method to use. Possible values are 'auto' (which is the default one), 'cityblock', 'euclidean', 'loglikelihood', 'pearson', 'spearman', 'tanimoto', and 'cosine'

- **preference**
  Indicates if the preference indicated in the fields parameter is 'score' (numeric) or 'boolean'

Given this easy "implementation", the workbench provides the following Collaborative Filtering algorithms for both books and movies:

- **item2item and user2user with cosine similarity**
  For the cosine similarity the items or users are seen as vectors in n-dimensional space. The similarity of two items or users (vectors) is given by the angle they form. 0 means the vectors are orthogonal and therefore there is absolutely no similarity between these items or users. 1 means the vectors are identical so the items or users are completely similar.

  \[
  \text{Cosine}(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{|\vec{x}| \times |\vec{y}|}
  \]

  Equation 3.5: Cosine similarity

- **item2item and user2user with euclidean similarity**
  For the euclidean similarity the items or users are seen as points in n-dimensional space.
space. The similarity of two items or users (points) is given by their distance. The smaller the distance, the bigger the similarity between the two users or items.

\[
Euclidean(X, Y) = \sqrt{\sum_{i=1}^{n} (X_i - Y_i)^2}
\]  

Equation 3.6: Euclidean similarity

- item2item and user2user with pearson similarity
  The pearson method calculates the correlation between the set of ratings of the users or items with the formula described by Equation 3.7, where \( \bar{X} \) and \( \bar{Y} \) is the average of \( X \) and \( Y \) respectively. The result will be between -1 and 1 where 1 means strong positive correlation (complete similarity), 0 means no correlation (no similarity) and -1 means strong negative correlation (complete dissimilarity).

\[
Pearson(X, Y) = \frac{\sum_{i=1}^{n} (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n} (X_i - \bar{X})^2} \sqrt{\sum_{i=1}^{n} (Y_i - \bar{Y})^2}}
\]  

Equation 3.7: Pearson similarity

- item2item and user2user with cityblock similarity
  The cityblock similarity is very similar to the euclidean similarity, but considers the absolute distance between the two points. This means, the distance is measured by going ”around” the blocks and not ”through ” them. The smaller the distance, the bigger the similarity between the items or users. Figure 3.10 graphically shows the cityblock and euclidean distance.
3.3. COLLABORATIVE FILTERING

\[ \text{Cityblock}(X, Y) = \sum_{i=1}^{n} |X_i - Y_i| \]  (3.8)

Equation 3.8: Cityblock similarity

- user2user with spearman similarity

The spearman similarity is very similar to the pearson similarity. It also measures the correlation between two variable sets, but does not assume that their relationship must be linear. In Equation 3.9 \( x_i \) defines rank of the \( i^{th} \) item in the set \( X \) and \( y_i \) defines the rank of the \( i^{th} \) item in the set \( Y \).

\[ \text{Spearman}(X, Y) = 1 - \frac{6 \sum_{i=1}^{n} (x_i - y_i)^2}{n(n^2 - 1)} \]  (3.9)

Equation 3.9: Spearman similarity

The remaining similarities (loglikelihood and tanimoto) were not used as they are rather designed for boolean similarity measurement and not for numeric.

As mentioned in section 3.1.4, registered users can get their personalized recommendations based on the above mentioned recommendation algorithms. After they rated
content (either movies or books) they can go to a dedicated page which triggers a new algorithm run. The Recommender API module uses the Async Command module to calculate the recommendations asynchronously. When the users select an recommender algorithm to run, he actually just schedules the execution of this algorithm. Then a shell script is triggered which tells the Async Command module to execute all scheduled commands. This way the registered users can run recommendation algorithms which then provide personalized recommendations for the user.

### 3.4 Content Based Filtering

For Content Based Filtering, there were no Drupal modules available which out of the box provided what we required for the workbench. Also the Recommender API module does not have a functionality for content based recommendations. So we decided to extend the recsys_wb module to allow Content Based Filtering on the stack overflow questions. The idea was to schedule the execution of the algorithm using the Async Command module (same behavior as the Recommender API). The Content Based Filtering algorithm itself is written in Java using the Apache Mahout\(^\text{14}\), Hadoop\(^\text{15}\) and Lucene\(^\text{16}\) libraries. There are two steps necessary to get the non-personalized recommendations for similar questions which are described in the following chapters.

#### 3.4.1 Create TF-IDF Vectors

First the documents (in this case the stack overflow questions) must be analyzed and so called TF-IDF vectors have to be created. TF-IDF stands for Term Frequency Inverse Document Frequency and measures how important a given word for the current document is in respect to all documents in the set. The formula to calculate the TF-IDF for a word \(w\) in document \(d\) is given in Equation 3.10.

\[
TF-IDF(w, d) = TF(w, d) \times IDF(w)
\]  

Equation 3.10: Term Frequency Inverse Document Frequency

\(^{14}\text{http://mahout.apache.org/}\)  
\(^{15}\text{http://hadoop.apache.org/}\)  
\(^{16}\text{http://lucene.apache.org/}\)
3.4. CONTENT BASED FILTERING

\[
TF(w, d) = \frac{freq(w, d)}{\max(freq(i, d), i \in d)} \quad (3.11)
\]

Equation 3.11: Term Frequency

\[
IDF(w) = \log \frac{N}{n(w)} \quad (3.12)
\]

Equation 3.12: Inverse Document Frequency

As one can see, the TF-IDF value is simply the product of the Term Frequency and the Inverse Document Frequency. From Equation 3.11 we can see, that the Term Frequency is the fraction of the terms occurrences in the document \((freq(w, d))\) and the term that occurs the most in the document \((\max(freq(i, d), i \in d))\). So the Term Frequency simply defines, how important the term is for the current document. The Inverse Document Frequency on the other hand defines how important the term is for the whole document set. We can see from Equation 3.12 that the Inverse Document Frequency is simply calculated as the logarithm of the fraction of the number of documents in the set \((N)\) and the number of documents in which the term \(w\) appears at least once \((n(w))\).

Each document is then represented as TF-IDF vector (each word has its TF-IDF value in the vector). As the process (namely the IDF part) requires all documents in the set to be present, the first step is to download all documents form the database and save them locally to the hard drive. Then the Hadoop, Mahout and Lucene libraries are used to calculate the TF-IDF vectors for each document and save them also locally on the hard drive. The last step is to upload these TF-IDF vectors to the database to be able to clean up the temporary used hard drive space. Subsequent steps will use the data from the database. As long as there is no document added to the set, the TF-IDF values do not change and therefore do not have to be recalculated.

3.4.2 Calculate Similarity

As mentioned above, the second step uses the TF-IDF vectors stored in the database to calculate their similarity. Calculating this vector similarity can easily be done using the similarity methods described in section 3.3. Unfortunately the Recommender API
module cannot be used as the data format does not match what the module expects. This is why, currently only the cosine similarity method (c.f. Equation 3.5) is implemented to compare the different TF-IDF vectors. In order to be able to compare two TF-IDF vectors, some preprocessing needs to be done. As not all document have the same size and contain different words, the TF-IDF vectors will also have different sizes and contain values of different words. Because of that, the vectors are enriched before they are compared. If a word occurs in one vector (document) but not in the other, it is added to with a TF-IDF value of zero. Zero because if a word does not occur $TF(w,d) = 0$ and therefore $TF-IDF(w,d) = 0$ (c.f. Equation 3.11 and 3.10). After this preprocessing step, the TF-IDF vectors have the same size, contain exactly the same words and can therefore be compared. Listing 3 shows the simple preprocessing in which the vectors are enriched with the missing words from the other vector. After this preprocessing step, the similarity algorithm function is called. Listing 4 shows the implementation of the cosine similarity function (c.f. Equation 3.5) which is called on line 20 in Listing 3.

```java
private double calculateSimilarity(Map<Integer, Double> documentVector1, 
                                   Map<Integer, Double> documentVector2) {
    Map<Integer, Double> map1 = 
        new TreeMap<Integer, Double>(documentVector1);
    Map<Integer, Double> map2 = 
        new TreeMap<Integer, Double>(documentVector2);

    for (Integer mapKey : map1.keySet()) {
        if ( ! map2.containsKey(mapKey) )
            map2.put(mapKey, (double) 0);
    }

    for (Integer mapKey : map2.keySet()) {
        if ( ! map1.containsKey(mapKey) )
            map1.put(mapKey, (double) 0);
    }

    List<Double> features1 = new ArrayList<Double>(map1.values());
    List<Double> features2 = new ArrayList<Double>(map2.values());
    return this.similarityAlgorithm.execute(features1, features2);
}
```

Listing 3: Enrich TF-IDF vectors to same size and vocabulary and calculate similarity
### 3.4. CONTENT BASED FILTERING

```java
public class CosineSimilarity implements SimilarityAlgorithm {

    /* (non-Javadoc)
     * @see ch.isproject.recsysWb.similarity.SimilarityType
     * #execute(java.util.List, java.util.List)
     */
    @Override
    public double execute(List<Double> featureVector0,
                          List<Double> featureVector1) {
        double dotProduct = 0.0;
        double magnitude0 = 0.0;
        double magnitude1 = 0.0;
        double cosineSimilarity = 0.0;

        for (int i = 0; i < featureVector0.size(); i++) {
            dotProduct += featureVector0.get(i) * featureVector1.get(i);
            magnitude0 += Math.pow(featureVector0.get(i), 2);
            magnitude1 += Math.pow(featureVector1.get(i), 2);
        }

        magnitude0 = Math.sqrt(magnitude0);
        magnitude1 = Math.sqrt(magnitude1);

        if (magnitude0 != 0.0 && magnitude1 != 0.0) {
            cosineSimilarity = dotProduct / (magnitude0 * magnitude1);
        } else {
            return 0.0;
        }

        return cosineSimilarity;
    }
}
```

Listing 4: Cosine similarity implementation

The similarity between the documents (vectors) is also written to the database from which the Drupal recsys_wb module can read the values and recommend similar questions to the user.
As mentioned above, the TF-IDF values and vectors do not change as long as there is no document added to the set. So obviously the similarity between the documents does not change as well. This is why the users are not able to trigger a run for Content Based Filtering. Only administrators can execute the TF-IDF vector generation and then the Content Based similarity calculation.
Outlook

Given the current state of the implementation, the workbench could be extended in different areas. The datasets for the Collaborative Filtering part are nice, books and movies with the ratings of users is pretty much the standard in recommender system context. Exactly the point that it is the standard shows that it could be enhanced. There are interesting datasets available which offer a complete different approach. The entree dataset\(^1\) for example has a bunch of restaurants available in different cities in the US. The data on which recommendations are performed are not ratings, but users which browsed the restaurants and navigated from one to another by expressing their preferences. If a user is at a very expensive Chinese restaurant, he can express his preference “find me something similar but cheaper”. The user is then redirected to another Chinese restaurant which is cheaper. Of course the restaurants in the dataset have the necessary information attached to make these kind of navigation choices. The browsing history of the users and the fact that the data also provides the information if the user finally accepted the

\(^{1}\)http://archive.ics.uci.edu/ml/datasets/Entree+Chicago+Recommendation+Data
websites recommendation could be used for Collaborative Filtering algorithms: find user(s) with similar taste and recommend restaurants which he (they) finally accepted. Another improvement to the datasets could be done on the Content Based Filtering data. The stack overflow dataset is very good because it is big enough to fulfill the requirements. However looking for similar questions is only useful if the question one is looking at was not answered. Additional Content Based Filtering algorithms could be added which take this fact into account. Generally the Content Based Filtering dataset could be extended with for example a News datasets for which the Content Based Filtering algorithms match perfectly.

Not also the datasets but also the logical content of the workbench itself could still be improved. The statistics could be extended to not only provide error metrics but also different metrics as coverage and diversity (see section 1.3). Also the error metrics for the Content Based Filtering algorithm are missing. As currently there is no ground truth for the stack overflow dataset, this is another point that could be done: enriching the currently available data with additional and new information to make the whole workbench better.

Last but not least, more algorithms could be added. For Content Based Filtering there is only the cosine similarity implemented to compare the TF-IDF vectors, so more similarity algorithms could be added. But not only the similarity methods can added. New approaches described in recommender system research papers could be added as for example recommendations based on graph databases [8] or machine learning algorithms such as SVM [9] or recurrent neural network [3].
Conclusion

Given the aims and objectives, the workbench fulfills them pretty good. The workbench can be used as starting point to learn about and experience recommender systems. There are enough different algorithms provided, especially for Collaborative Filtering, to clearly see and experience the differences between them. Getting different recommendations based on the same data impressively shows the user that recommender system is not equal to recommender system. The functionality to compare the different recommendations is a very nice feature which makes it easy to spot those differences. Unfortunately the workbench currently just contains state of the art recommender system algorithms. It would have been nice to include some other algorithms as described in chapter 4. Also the point that there is no fuzzy based recommender algorithm is present is a little bit unfortunate. However there we could not have much influence. As stated in section 2.2, the idea was to include a Drupal module which was, at the time the thesis started, in development. For some reason, this project was discontinued and could therefore not be included. At this point we decided to shift the objectives a little. As the implementation for Collaborative Filtering algorithms was very easy (see section 3.3), we just added
much more Collaborative Filtering algorithms which are provided by the Recommender API module. Not having the fuzzy based algorithm for Collaborative Filtering, we also decided to keep the Content Based Filtering as easy as possible. The main objectives for the workbench can also be met without the fuzzy based algorithms, it would just have been nice to have them provided by the workbench as well, to be able to compare them to the state of the art algorithms. Adding the fuzzy based algorithms to the workbench would definitely be something that should be concerned if the project is continued.

Implementing the workbench using Drupal was, in respect to the objectives, a good choice. Drupal itself provides a lot of modules which do a lot of work which had to be done anyway. User handling, the design and a lot of Collaborative Filtering algorithms are provided out of the box what made it finally easy to create the workbench. A far more challenging and time consuming task was it, to search, split and convert the datasets. The first problem was to find datasets which matched the criteria. There are a lot of datasets available for plain machine learning, but they do often not provide the necessary information needed by recommender systems such as user-item ratings or a ground truth (for evaluation). Once found datasets which provided all necessary information, they also had to be big enough to be split into 21 subsets. All this really restricted the possible candidates. The remaining datasets came in all different formats: SQL-Dump, CSV, XML, plain text and whatever one can think of. Working with Drupal made it likely to work with a relational database such as MySQL. So the datasets had to be converted to SQL format and be imported into the database. In those big datasets however, a lot of special characters from all over the world appeared. Getting the data in an importable state was a real pain. Changing encodings, filtering out not allowed characters, importing the dataset, failing. This process had to be repeated several times until the data finally was ready to use.
# List of Figures

3.1 A book displayed using the Display Suite module . . . . . . . . . . . . 15  
3.2 A movie displayed with its full information . . . . . . . . . . . . . . 15  
3.3 A question displayed with "Similar Questions" block on the right . . . 16  
3.4 Explaining user-user based cosine similarity with an example . . . . . 17  
3.5 Error metrics for item-item based Collaborative Filtering algorithms . . 20  
3.6 Runtime statistics with history . . . . . . . . . . . . . . . . . . . . . 21  
3.7 A user can rate content scaled $\frac{1}{2}$ to 5 stars ($\frac{1}{2}$ star steps) . . . . . . . 22  
3.8 Select which type of recommendations the user wants to get . . . . . 22  
3.9 Compare top 5 list on user based CF movie recommendations . . . . . 23  
3.10 Cityblock (red, blue and yellow lines) vs. euclidean (green line) distance 29
List of Tables

1.1 User-User Collaborative Filtering ........................................... 3
1.2 Item-Item Collaborative Filtering ........................................... 4
List of Equations

3.1 Mean Absolute Error .............................................. 18
3.2 Rooted Mean Squared Error ................................. 18
3.3 Mean Reciprocal Rank ........................................... 18
3.4 Normalized Discounted Cumulative Gain ................... 19
3.5 Cosine similarity .................................................. 27
3.6 Euclidean similarity ............................................. 28
3.7 Pearson similarity ................................................ 28
3.8 Cityblock similarity ............................................. 29
3.9 Spearman similarity ............................................. 29
3.10 Term Frequency Inverse Document Frequency .......... 30
3.11 Term Frequency .................................................. 31
3.12 Inverse Document Frequency ............................... 31
Bibliography


Explaining the File Structure

```
recsys_wb/
  ├── CB.php
  ├── CF.php
  │   └── config
  │       └── config.properties
  ├── css
  │   └── radio-button.css
  │       └── ...
  ├── datasets
  │   └── Demo_Dataset.zip
  ├── evaluation_explain.php
  ├── evaluation.php
  ├── forms
  │   └── forms_submit.php
  │       └── forms_view.php
  ├── img
  │   └── Manhattan_distance.png
  └── java
```
Listing 5: File structure

Listing 5 shows the file structure of the recsys_wb Drupal module. In this chapter, I’m going to explain, where and how the improvements mentioned in chapter 4 should be implemented.

- **Adding more statistics and metrics**
  The error metrics are implemented in the evaluation.php file located in the root directory. The functions for retrieving and displaying statistics are implemented in the statistics.php file which also is located in the root directory. Therefore these files are the starting point if one wants to add more functionality. All the functions implemented in those files are called from the recsys_wb.module file, according to the changes it might be necessary to adapt this file as well.

- **Adding more algorithms**

  - **Adding similarity methods for Content Based Filtering**
    The similarity methods for Content Based Filtering can be implemented in Java. A class which implements the SimilarityAlgorithm.java interface in the java/src/ch/isproject/recsysWb/similarity directory can be created. As the interface defines, the execute(List<Double> featureVector0, List<Double> featureVector1) has to be implemented which as a result returns a Double which indicates the similarity between the two feature vectors.
    Currently the similarity method to use is hard coded (as there is only one). So registration of the job using the Async Command module must be changed to indicate the similarity method to use. This change must be done on the PHP code. As currently the execution is a form which is submitted, both files inside the forms directory need to be adapted: The form itself needs to be extended in order to be able to select the similarity algorithm. This needs to be done in the forms_view.php file and

```
run-cf.sh
...  
statistics.php
util.php
```
the recsys_wb_run_content_recommender_form() method. Then the submission of this form needs to handle this new form field. This is done in the forms_submit.php file in the recsys_wb_run_content_recommender_form_submit() method (which is default Drupal behavior). As a next step, the constructor of the RunContentRecommender Java class (located in java/src/ch/isproject/recsysWb/contentRecommender) must be extended to read the similarity method parameter from the database and instantiate the correct similarity method. (The database is already prepared for this procedure.)

- Adding new algorithms

If one wants to add new algorithms (not Collaborative or Content Based Filtering), it is probably best to use also the Async Command module. Scheduling something for execution is very easy. As soon as the database entry for the scheduled execution is present, a shell script is called which executes the actual code. As it is a shell script, the algorithms can be written in any language one likes: Python, Java or even C. As Java (maybe other languages too) has a library to use the Async Command module (provided by the module itself), it is probably a good idea to use Java. An example for how to implement an algorithm in Java using the Async Command is given in the Content Based Filtering algorithm of the workbench (c.f. file java/src/ch/isproject/recsysWb/contentRecommender/ContentRecommenderApp.java in Listing 5). Scheduling an execution using the Async Command module can be seen in the file forms_submit.php in method recsys_wb_run_content_recommender_form_submit().

Generally the file structure is pretty easy and follows the Drupal module file structure. The dependencies of other Drupal modules as well as the version, name and description are in the recsys_wb.info file in the root directory. Database schemas for databases used by the module and configuration of the Collaborative Filtering algorithms (c.f. 3.3) are part of the recsys_wb.install file also located in the root directory. All the menu items and their page callback methods are implemented in the recsys_wb.module file. This files also contains the definition of the blocks, the dataset explanation and other small methods like text for the about page or the admin page.

The explanations can be found in the corresponding files in the root directories.
Collaborative Filtering explanations are in the CF.php file, Content Based Filtering explanations are in the CB.php file and evaluation explanations are (obviously) in the evaluation_explain.php file. The evaluation.php file contains the evaluation metrics implementation as for example the MAE (c.f. Listing 1). The files statistics.php and recommendations.php located in the root directory contain the code for retrieving and formatting the selected statistics and recommendations for the user. The util.php file, also located in the root directory, contains a bunch of methods which are used throughout the whole project.

The root directory also contains some directories which are pretty self explanatory: config, css, img, lib, log, papers and scripts. They all contain what the name already indicates. No surprise there. The other directories are used for the following purposes:

- **datasets**
  This directory contains all the datasets which are downloadable by a user. The module itself contains only the demo dataset. Other datasets can be added later as soon as the workbench is installed. There is one constraint: The dataset must be a .zip file in order to appear on the dataset page.

- **forms**
  The forms directory contains all the Drupal form views and the corresponding actions to take as soon as a form is submitted.

- **java**
  In this directory, the Java code for Content Based Filtering is located. It is important to know, that this code is not executed. So if one changes this code, this does not affect the workbench. The code that is being executed by the workbench is the compiled .jar file in the lib directory. This directory and code is only present to be transparent and let a possible interested person know how it was implemented.

- **mahout**
  This is the apache mahout library. As I am writing this appendix, I am asking myself why it is in the root and not in the lib directory. Probably this is a leftover because the mahout library was the first dependency of the workbench. It should actually be put in the lib directory.
• php-tail
  The php-tail script available at https://code.google.com/p/php-tail/. This script is used to track the execution progress of the recommender algorithms. I adapted the script a little bit to fit into the workbench and hide some potentially dangerous information.