Diplomarbeit

A Contribution to Rating and Recommendation Systems: Concepts, Development and Evaluation

Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Lehrstuhl Medieninformatik

angefertigt von: **Oliver Diestel**
betreuender Hochschullehrer: Prof. Dr. Klaus Tochtermann
Betreuer: Arne Martin Klemenz

Kiel, ** Datum der Abgabe **

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

....

** eigener Name **

Contents

List of Figures

List of Tables

List of Algorithms

Abkürzungen

Abstract

The focus of this thesis is the development of a recommendation system for user generated content. The purpose of this system is to utilize the unused content that is contributed by users. If this content is added in a meaningful way on existing websites, it can increase the benefit of the website for its users. Meaningful way means that the content matches the current context of the website and is interesting for the current user. This process involves three different types of research areas.

First, the generation of implicit user ratings by analyzing user interactions on a website. This part of the thesis involves the discussion of different user interactions on a website. The two interactions that correspond the most with explicit ratings are *the time a user spends on a website* and *the time a user scrolls on a website*. For the two times a scale is created that enables the transformation of user interactions into implicit ratings.

The second research area is approximate string matching which is the matching of strings even if the words are slightly different. This technology is used for categorizing the user generated content by its keywords. For this process the technique of the *Levenshtein automaton* is used and explained in more detail. Besides, a range is created that expresses how diverse words are allowed to be in order to be accepted as a match.

The last research area is the field of recommendation systems.The different types of recommendation systems are discussed in this part. Moreover, a new technique is introduced that combines two well known recommendation techniques to improve the rating predictions. This system is implemented with a service oriented architecture so that every component of the system can be exchanged with different alternatives. Furthermore services can be used to extend existing systems. This leads to a considerable amount of flexibility and expandability.

1 Introduction

Since the creation of the term *Web 2.0* the internet is full of user generated content and communities. More and more website operators are adding methods to their sites which have the goal to increase the user engagement by allowing users to actively contribute to the content of the website. However, as more and more content is produced most of the user contributed content will remain unseen. The value of a website is the benefit that it brings to its users. Therefore every content that includes valuable information must be presented to users that might be interested in it. This can improve the user interactions on the website and can help the users to learn from each other and thus to better understand the content. Though, the quality of the content that is produced by users may vary widely. Moreover, the context of the user generated content can be inappropriate to the current subject of the page. Thus there is a need for a system that is able to identify the interest of a user and the context of the user generated content. Such a system is developed in this thesis. This development includes the categorizing of the content by its keywords by matching the words of the content with words from a *thesaurus*. This process should work even if the words have spelling mistakes or are in a different form than the words of the thesaurus. Additionally a concept is developed and implemented that generates implicit user ratings for the content by analyzing the user interactions on the website of the content. Finally a recommendation system is introduced and evaluated that combines two different well known recommendation techniques for improving the prediction accuracy.

1.1 Related Work

This thesis combines different fields of research areas namely the field of recommender systems, the field of collecting implicit user ratings and the field of string matching techniques. Thus the following papers are related to one ore more of the research fields that are treated in this thesis. The team of Claypool et al. [2001] developed a web browser that collects user interactions on a website. This browser asks the user when he leaves a website for his rating for the site. They conducted a field experiment with 75 students that rated 1823 websites and analyzed the correlation between the explicit ratings and the implicit user interactions. Another paper in the field of implicit feedback is written by Joachims et al. [2005]. In this paper Joachims et al. [2005] analyze the behaviour of users with eye tracking technology before they click on links in a search result. Hereby they discover that users tend to click on links more frequently if they are relevant to their current interest. However the users tend to be biased by the trust in the search engines and tend to click

more often on the highest ranking links. Thus they found out that it is difficult to interpret clicks as absolute feedback. The paper *Personalized news recommendation based on click behavior* written by Liu et al. [2010] analyzed the click behavior of Google news users to find out how the interest of the users changed over time to predict the current and future interest depending on the clicks of similar users. They tested the developed system in production mode and found out that the traffic of the *Google News* website increased because of the improved recommendation system. The team of Sarwar et al. [2001] introduces their invention of the *item-based* recommendation algorithm. This algorithm calculates the item similarities based on item rating vectors and recommends items to users based on the similarity of the items they rated. Furthermore they discuss different calculation techniques and evaluate the performance of the algorithm. Linden et al. [2003] explains the product recommendation system of *amazon.com* in detail and describes their implementation of the item-based recommendation algorithm. Karatzoglou et al. [2010] describe a mathematical concept and an algorithm for representing and compressing the latent factors of the singular value decomposition with hashing. For each user and each item one vector has to be stored in the main memory while calculating the matrix decomposition. However usually only a small number of ratings exist for each item Karatzoglou et al. [2010] introduced a method of an approximate compressed representation of the latent factor matrices. Cacheda et al. [2011] describe the current state of the algorithm and evaluation techniques of collaborative filtering recommendation systems. They discuss the different metrics that are used for evaluating recommendation systems. Additionally they found that the best performing algorithms in their evaluation are all based on the singular value decomposition. The article *Matrix Factorization Techniques for Recommender Systems* by Koren et al. [2009] explains the strategies of recommender systems and how to apply matrix factorization methods in these systems which are according to Koren et al. [2009] the most accurate prediction techniques found out by many of the teams that were competing for the *Netflix prize*[1]. The paper of Herlocker et al. [2004] deals with the evaluation of recommender systems. They explain in detail the different type of testing datasets and the current trends in datasets and accuracy metrics. The following article treat the techniques of approximate string matching which is the matching of strings that allow errors. Navarro [2001] presents an overview on the different kind of concepts that enable approximate string matching. Schulz and Mihov [2002] explain the principle of a *Levenshtein automata* in detail and describe a concept on how to use this Levenshtein automata for an efficient way to correct strings with the help of a dictionary. The same authors wrote a paper about an *universal Levenshtein automata* - Mihov and Schulz [2004] - which is a specialized version of a *Levenshtein automata* that improves the time that is needed for constructing the automata.

---

[1]http://www.netflixprize.com/

1.2 Challenges

The focus of this work is the development of a recommendation system for user generated content. One challenge of such a system is that no trustworthy information about the quality or context of the user generated content is available. Hence, a concept must be created that treats the finding of keywords to represent the content. These keywords should be found by matching the content with the words from a thesaurus even if the words have spelling mistakes or are in a different form. Additionally the concept must include the generation of user ratings for this content to be able to predict how interesting the content is for the user. This rating generation should work by automatically analysing the user interactions on the website that presents the content. Therefore the user experience of the website is not disturbed by explicitly asking for the users opinion. Another challenge is to use the rating and keyword information to generate content recommendations for users based on their interest. This concept should be implemented into a system that can be used as an ad on for existing systems. Moreover each component should be usable as a standalone system to extend existing technologies.

1.3 Overview

In Chapter 2 the concepts of this thesis are presented. This chapter is divided into four sections. At first the concept of a *Question and Answer* website is explained which is used as a tool for collecting user generated content. The second section evaluates the differences between implicit and explicit ratings and the different sources of types of user interactions on websites. Besides it introduces a concept for generating implicit ratings with these user interactions. The next section examines the methods of string matching that allows a system to match words even if they are in a different form or have spelling mistakes. Additionally this section introduces the technique that is used in the developed system for finding the keywords in the user generated content that is collected by the Question and Answer website. In the last section different types of recommendation systems are considered and the concept of *collaborative filtering* is explained in more detail. Likewise the new concept of recommending user generated content based on a combination of *item* - and *singular value decompositon* - based methods is explained.

Chapter 3 starts with a section about the presentation of the technologies that are used for implementing the developed concept. Alongside, with a description of the service oriented software architecture and an insight into the implementation as well as the relationships of the single services.

Chapter 4 evaluates the implemented rating, tagging and recommendation concepts. This chapter is divided into three sections. The first section tests the implemented concept of the implicit ratings. For this process values that are collected by a field experiment conducted by Claypool et al. [2001] are used for evaluating the rating concept. These values are used for simulating user interactions on a website and the

results are compared with the analysis of Claypool et al. [2001]. The second section evaluates the implemented tagging concept. For this evaluation different sized texts were created. These texts have a percentage of 30% of words that are included in the used thesaurus and 70% that are not. However, into the 30% of the words that are included in the thesaurus are spelling mistakes inserted or the form is changed. Hence, the tagging implementation is tested for the amount of words that are found and the time that it needed for this process. The last section evaluates different types of recommendation systems that are composed with the techniques of the recommendation concept of this thesis. This evaluation uses a rating database with 100.000 real user ratings of the *MovieLens*[2] project for evaluating the speed and the quality of the predictions of the different recommendation systems.

---

[2]http://movielens.org/

2 Concepts

This chapter outlines the concepts that are integrated in the developed system. The first section deals with the approach of a *question and answer site* and explains the necessity of such a system. The second section is about the differences between implicit and explicit ratings and the different sources of implicit ratings. Moreover this section describes the implemented concept of collecting implicit ratings. The third section treats the automated tagging process that is used to find the keywords in the questions and answers. The fourth section is a discussion about the different types of recommendation systems. This leads to the analysis of the different techniques of *collective recommendations* and explains how these techniques are used in the final system. Finally the last section presents the relation between the individual concepts from the previous sections.

2.1 Question and Answer Site

A question and answer site is a website that has only one focus namely getting the right answer for a question. That means that a user can ask a question and that this question should be precise enough and include all information that another user needs to be able to write a correct answer to the question without asking for more details. So, it is not a discussion forum but a website in which every answer only refers to the original question. Then the user who started the question can mark one answer as correct. This question and answer site should make the information retrieval process on a website publicly accessible. Therefore all users are able to read all questions and answers and can support domain experts by answering questions. For each question on the site there is exactly one answer marked as correct, thus, it is possible to recommend the questions with the correct answers to users who visit a website with the same or similar subject. This recommendation process should increase the user interaction on the website. Moreover, it should help the user to understand the content which he is interested in. Such question and answer sites are widely spread on the internet and there exists a couple of open source implementations that can be used for free. Some of such open source implementations are askbot.com, discourse.org, lampcms.com and osqa.net. They all work and almost look the same. However, they use different technologies and some are more mature than others. The open source variant that has been decided to be used for the implemented system is askbot.com. It is build with the *django web framework* and actively maintained since 2011. Though, the system is implemented the way that every other question and answer site can be used as well.

Henceforth, for better understanding, the word *item* will be used instead of question and answer for an element that should be recommended.

2.2 Rating System

In order to be able to recommend items to users it is important to understand what items a user likes. For the purpose of such a task a rating scale ranging from 1, strongly disliked, to 5, strongly liked, is used. There are two possible forms for retrieving ratings namely implicit ratings and explicit ratings. The latter are those in which a user is explicitly asked for his opinion on a specific item. Implicit ratings are those in which a system generates a rating according to the actions of a user. Anyhow, according to the book *Recommender Systems An Introduction* [Jannach et al. [2011]] the explicit user ratings are usually not well accepted if a direct benefit is not visible to the user. Furthermore, the explicit ratings might disturb the user experience of a website. Adding the explicit ratings with visible benefits to an existing question and answer website would require too much customization because in case the question and answer website is replaced by a different open source version or an own implementation the explicit ratings with visible benefits have to be added again. Therefore it is a good approach to use implicit ratings to collect the interest of a user. The research group of Claypool et al. [2001] developed a web browser that monitors user interactions and afterwards asks a user how he would rate the page. Thereby it compares user interactions with explicit ratings. Claypool et al. [2001] build their results on a field experiment of over 80 people browsing over 2500 web pages. The user interactions they measured with their web browser are presented below.

2.2.1 The time a user spends on a website

The team of Claypool et al. [2001] measured the time a user spends on a website as a first indicator that the user is interested in the web site. The idea behind this method is that the more time the user takes to view the content of the website the more interesting the website seems to be for him. For this method a timer was used that measured the time a user spends on a website. The timer starts after the website is completely loaded and stops when the user leaves the website or closes the window. So, as soon as the internet user switches to another website the timer stops. They compared the time a user spends on a website with the explicit rating from the user. They found out that the time a user spends on a website correlates about 70% of the time with the explicit rating of the same user. So, they found out that the time a user spends on a website is a good indicator of interest. For this reason this method is used as one implicit rating indicator in this thesis. However, one has to keep in mind that the general time a website is open and visible needs to be relativized because one does not know whether the user is actually reading or interacting with the content of the current website. For all we know the user could have opened a website and afterwards left the computer. Therefore other indicators where measured and included in Claypool et al. [2001] research which will be clarified in the following sections.

2.2.2 The time the cursor is in motion

Whereas the general time a user spends on a website already gives a good indication on how interesting the website is it is also important to measure how far the user actively engages with the website. The use of methods that measure how active a user is on a website is important because these methods can indicate whether a user just opened a website and does not work with it or if he really engages with the content. The time the cursor is in motion was measured by timing how long the mouse cursor changes its position inside the active browser window. By comparing the explicit user ratings with the collected data they found out that the time a user moves the mouse cursor is proportional to the interest a user has in the website. However, due to the fact that some users use the mouse heavily whilst reading the content of the website or looking at interesting objects such as diagrams or pictures, other users tend to only use the mouse in order to click on objects. Consequently, it is not possible to tell how much a user is interested in the website by just timing the cursor motion but it is possible to tell which websites receive the least amount of interest. Thus, in order to reflect the users interest more properly other ways to measure the active time on the website must be taken into account.

2.2.3 The number of mouse clicks

Another method of collecting user interactions that might correlate with the explicit rating of a user is the number of mouse clicks on a website. The research team of Claypool et al. [2001] thought that a high number of mouse clicks would be a sign that the website has links to interesting websites or objects and as such would be valuable to the users. The collected data, however, showed that the number of mouse clicks on a website does not significantly differ even if the user rated it with the highest or lowest rating. So, this method does not seem to be a good indicator for the users' interest.

2.2.4 The time a user scrolls

The last indicator they used to determine how interesting a website is was the time a user scrolls. This is an important activity indicator because a user usually has to scroll in order to be able to see the whole content of a website. The web browser the team of Claypool et al. [2001] developed measured the time a user scrolls a website by keys and by mouse. They compared the data of the method of the field experiment with over 80 people with the explicit user ratings. Due to this data they found out that both the measurement of user scrolls by keys and by mouse are on their own poor indicators of interest whereas a combination of both methods is found to be a good indicator of interest. This is explainable by the fact that some users prefer to scroll by using the mouse while others prefer the keyboard for such a task. For this reason the time a user scrolls by mouse and by key is used in this thesis as an activity indicator.

2.2.5 Complete Rating Concept

As could be seen above the best sources of implicit ratings are the time a user spends on a web page and the amount a user scrolls on a web page according to the results of Claypool et al. [2001]. These methods agree approximately 70% to the explicit user ratings. This correlation between the implicit and explicit ratings can be explained by the fact that a user who is interested in an item takes a good look at an interesting website and reads the content that is presented on this site more carefully. This process takes time so that the user stays on the website longer and most of the time he/she has to scroll in order to see the whole content of the website.
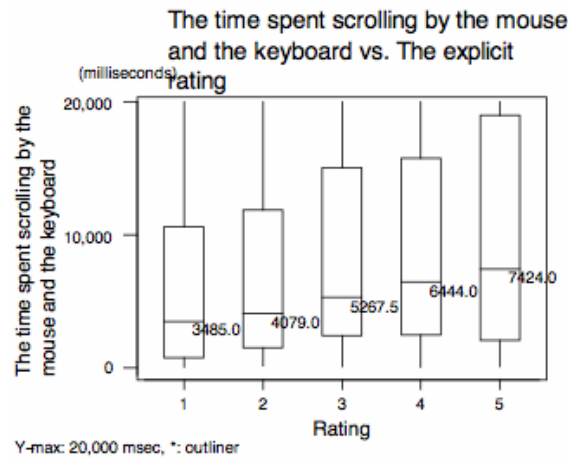

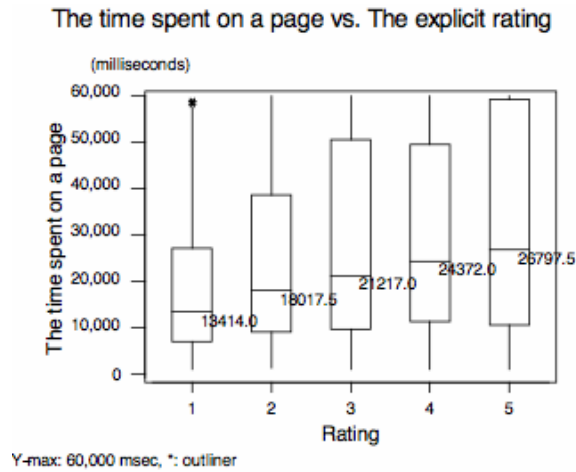Figure 2.1: Time spent scrolling source: Claypool et al. [2001]

Figure 2.2: Time spent on a page source: Claypool et al. [2001]

9

The two diagrams of figure 2.1 and figure 2.2 represent the time a user scrolls on a website and the time a user spends on a website. These diagrams are created by the team of Claypool et al. [2001]. The x-axis represents the explicit user ratings and the y-axis represents the time a user has scrolled on a website and the time that a user has spend on the website respectively. The bar for each explicit rating represents the time range of all users that explicitly rated with that rating. The line in each of the bars represents the average time of all user times of that rating. Interesting to note is that the average times for ratings in both diagrams have a steady rise which mirrors the statement of Claypool et al. [2001] that these times correspond with the rating of the users. However, the ranges of the ratings are very wide and thus for calculating the ratings from the times one should remove a percentage of the highest time that a user spends or scrolls as well as a percentage of the lowest times. This process should filter the abnormal user times and yield data that should reflect the user interest more explicit. This filtered data can now be used in order to calculate the average time of all users this process is called *Truncated Mean* [Wikipedia, The Free Encyclopedia [2013c]].

Definition 1 (Truncated Mean). *The truncated mean (tm) is calculated the same way as the average mean after discarding a specific percentage of the highest and lowest values. Let $n \in \mathbb{N}$, V be a set of sorted numbers and p be a percentage with $0 \leq p < 0.5$. Let $k = np$ be the trimmed value, $r = n - 2k$ be the remaining values, $v_j \in V$ and $j \in \mathbb{N}$. Then the following formula represents the truncated mean.*

$$tm = \sum_{j=k+1}^{n-k} v_j \cdot \frac{1}{r}$$

For this thesis the time differences between the average times of each rating are assumed as a range for implicit ratings. These ranges are expressed as percentages in dependence to the general average time of all users in the following two enumerations.

- If the user spends up to 76.5% of the truncated mean time on the item page it is used as a rating of 1.

- If the user spends between 76.5% and 94.3% of the truncated mean time on the website it is used as a rating of 2.

- If the user spends between 94.3% and 109.6% of the truncated mean time on the item page, it is used as a rating of 3.

- If the user spends longer than 109.6% of the truncated mean time on the website it is used as a rating of 4.

A rating of 5 can only be achieved if a user does a predefined user interaction on the web page such as clicking on a button that indicates interest. The reason behind this choice is that users tend to make breaks while surfing the internet. Thus, the user leaves the current browser window open but is not actively working on this

website. Therefore, it is possible that a user is on a break and the currently active browser window might not be interesting for him in that moment. Such a predefined user interaction can be used as an indicator that a user likes an item. A possible predefined user interaction could be an up-vote or the writing of a comment or an answer. A similar scale is chosen for determining the rating of a user by the time a user scrolls.

- If the user scrolls less than 69% of the truncated mean time on the item page it is used as a rating of 1.

- If the user scrolls between 69% and 86% of the truncated mean time on the webpage it is used as a rating of 2.

- If the user scrolls between 86% and 108% of the truncated mean time on the webpage it is used as a rating of 3.

- If the user scrolls between 108% and 129% of the the truncated mean time on the website it is used as a rating of 4.

- If the user scrolls longer than 129% of the truncated mean time on the website it is used as a rating of 5

Due to the fact that scrolling is an active process and the user has to be in front of the computer a rating of 5 is achievable in this method.

In these two enumerations it is noticeable that the largest percentage area is at the rating of one. This can be explained by the fact that the user regards the overall website before he starts to evaluate the content of the website. Therefore the area of the rating of one contains the time that the user needs to make himself familiar with the website as well as the time he needs to evaluate the content. For the calculation of the complete rating both implicit ratings - the time a user spends on a website and the time a user scrolls - are determined separately. Afterwards the average of both ratings is calculated and used as the complete rating.

$$completeRating = \frac{rating_{time} + rating_{scroll}}{2}$$

If a user returns to an item site the time that the user spends on the page and the time the user scrolls on the page is added to the previous times. The sum of both sessions is used to calculate the new rating.

As mentioned above the research group of Claypool et al. [2001] bases its results on a field experiment of over 80 people browsing over 2500 web pages. The complete rating concept is strongly build upon these results. It would have been desirable to conduct a field experiment for the newly developed concept of the complete rating but unfortunately due to the lack of time it was not possible.

2.3 Tagging

In order to recommend items with a matching subject on a website, the newly developed system needs to be able to filter the items based on their content. Therefore the keywords of the item content are used as tags in order to reflect the content of the items. In the context of economics the *Deutsche Zentralbibliothek für Wirtschaftswissenschaften Leibniz-Informationszentrum Wirtschaft* (ZBW) provides the *STW Thesaurus for Economics* (STW). The STW provides vocabulary on any economic subject and includes about 19000 terms and 6000 standardized keywords which can be used to match words from a text.[1] It, thus, provides a good basis for the tagging process. However, the STW only includes the basic form of the words while excluding most of the words with affixation, due to this it is not sufficient to check whether the words from the items are equal to the words from the STW. So, the challenge for this tagging process is to find the correct words in the STW even though the words from the item might not correspond directly to the words in the STW. One possible solution for such a task is to reduce each word from the text to its stem. According to Kortmann [2011] a stem is the lexical unit that remains when all inflectional affixes are taken away. Whereby, inflectional affixes are affixes that produce a new word form without changing the core meaning of the word. Such an inflectional affix is for example the adding of the plural *s* to a word. Such a task is called stemming and is usually done by using predefined rules on the words. A rule is usually a combination of the minimum number of letters in a word, plus the suffix that should be changed and the replacement for the suffix. More advanced algorithms might also use rules for prefix reduction and detecting irregular changes of the stem according to Jörg [1999]. For example a predefined rule might be '3+ies' → 'y'. So, the word *libraries* would be transformed to *library*. Thus, it is important that the stemming algorithm has all necessary rules for each supported language.

Another challenge for using the stemming algorithm is that all words in the basis for the tagging process must be in the stem form. Otherwise the algorithm might reduce a word to its stem that would actually match already in its original form. Additionally a stemming algorithm cannot find words if they have spelling mistakes. Because of these maintenance problems it was decided that the stemming algorithm is not suitable for the developed system. Another possible solution for the problem is to calculate the number of changes that have to be done in order to transform the word from the text to the word from the basis for the tagging process. This number can be used as an indicator whether the words are similar or not. If they are in a predefined difference range the words can be used as tags. This creates the need for a metric that indicates the difference or distance between two words. A well known metric for such a task is the *Levenshtein distance* which will be used in the system and will be explained in the following section.

---

[1]STW Thesaurus for Economics url: http://zbw.eu/stw/versions/latest/about

2.3.1 Levenshtein Distance

The Levenshtein distance calculates the minimum numbers of substitutions, insertions and deletions that are needed to change one word into another[Wikipedia, The Free Encyclopedia [2013a]]. The following example explains the Levenshtein distance for the words *library* and *libraries*. The algorithm has to perform two deletions and one substitution in order to create the word *library* out of the word *libraries*. If it creates the word *libraries* out of the word *library* it has to perform a substitution and two insertions.

$$library \leftrightarrow librari \leftrightarrow librarie \leftrightarrow libraries$$

Both processes result in a Levenshtein distance of three. So, the Levenshtein distance is zero if the words are equal and adds one to the result if it has to perform a substitution, an insertion or a deletion of a letter. For a more detailed description compare algorithm 1.

Algorithm 1 Recursive Levenshtein Distance Algorithm

1: **procedure** LEVENSHTEINDISTANCE($s : String, t : String$)
2: $\quad lenS \leftarrow length(s)$
3: $\quad$ **if** lenS = 0 **then**
4: $\quad\quad$ **return** $lenT$
5: $\quad$ **end if**
6: $\quad$ **if** lenT = 0 **then**
7: $\quad\quad$ **return** $lenS$
8: $\quad$ **end if**
9: $\quad$ **if** s[lenS-1] = t[lenT-1] **then** $\quad\quad \triangleright$ test if last characters of the strings match
10: $\quad\quad cost \leftarrow 0$
11: $\quad$ **else**
12: $\quad\quad cost \leftarrow 1$
13: $\quad$ **end if**
$\quad\quad\quad \triangleright$ The first recursive call represents a deletion, the second represents an insertion and the third represents a substitution or a correct letter
14: $\quad$ **return** minimum of
$\quad\quad LevenshteinDistance(s[0..lenS - 1], t) + 1,$
$\quad\quad LevenshteinDistance(s, t[0..lenT - 1]) + 1,$
$\quad\quad LevenshteinDistance(s[0..lenS - 1], t[0..lenT - 1]) + cost)$
15: **end procedure**

The direct implementation of the Levenshtein distance algorithm has a complexity of O(mn) with m being the size of the first word and n being the size of the second word. Therefore it is a good utility to better understand the Levenshtein distance in general, but it is not feasible for a software that should work in production mode.

2.3.2 Optimized Levenshtein distance algorithm

The following definitions are based on the book by Hopcroft et al. [2003].

Definition 2 (Non Deterministic Finite Automata)**.** *A non deterministic finite automaton is a 5-tupel of the form* $\mathcal{A} = (Q, \Sigma, q_0, \Delta, F)$.

- *Q is a finite set of the states*

- *Σ is a finite set of input symbols*

- *$q_0 \in Q$ is the initial state*

- *$F \subset Q$ is a subset that contains the final states*

- *Δ is a relation of the form $\Delta \subset Q \times \Sigma \times Q$*

$\mathcal{A}$ is called finite if and only if Q is finite. Furthermore Σ^* is the set of words over Σ and ϵ is the empty word.

Definition 3 (Deterministic Finite Automata)**.** *$\mathcal{A}$ is deterministic if for all $p \in Q$ and all $a \in \Sigma$ exists exactly one state $q \in Q$ with $(p, a, q) \in \Delta$. In this case Δ is written as a function $\delta : Q \times \Sigma \to Q$.*

Definition 4 (Path)**.** *A path for $\mathcal{A}$ is a series $\pi = p_0 a_1 p_1 a_2 \ldots a_n p_n$ with $(p_i, a_{i+1}, p_{i+1}) \in \Delta$, $0 \leq i \leq n - 1$ and $n \in \mathbb{N}$. The function $\beta(\pi)$ maps a path π to a word $\omega \in \Sigma^*$. The length of π is n and the label for $\beta(\pi)$ is $a_1 a_2 a_3 \ldots a_n$.*

Definition 5 (Path shortwriting)**.** *$\mathcal{A} : p \xrightarrow{\omega} q$ with $\omega \in \Sigma^*$ states that a path π for $\mathcal{A}$ from p to q with label $\beta(\pi) = \omega$ exists.*

Definition 6 (Automata accepts a word)**.** *$\mathcal{A}$ accepts $\omega \in \Sigma^*$, if and only if $p \in I, q \in F$ exists with $\mathcal{A}: p \xrightarrow{\omega} q$. For $\mathcal{A}$ let $\mathcal{L}(\mathcal{A}) = \{\omega \in \Sigma^* \mid \mathcal{A} \text{ accepts } \omega\}$ be the language that $\mathcal{A}$ accepts.*

The following definitions are based on Schulz and Mihov [2002]

Definition 7 (Formal Levenshtein Distance)**.** *The Levenshtein distance between two words $V, W \in \Sigma^*$ is the minimal number of edit operations (substitutions, deletions or insertions) that are needed to transform V into W. $d_L(V, W)$ denotes the Levenshtein distance between V and W.*

Definition 8. *$\mathcal{L}_{Lev}(n, W)$, $n \in \mathbb{N}$ and $W \in \Sigma^*$ is the set that denotes all words $V \in \Sigma^*$ such that $d_L(W, V) \leq n$.*

Definition 9 (Degree Levenshtein Automata)**.** *Let $W \in \Sigma^*$ and $n \in \mathbb{N}$. A finite state automaton $\mathcal{A}$ is a Levenshtein automaton of degree n for W if and only if $\mathcal{L}((A)) = \mathcal{L}_{Lev}(n, W)$.*

An optimized version of the Levenshtein distance algorithm that uses a Levenshtein automaton is described by Baeza-Yates [1996]. The purpose of the Levenshtein automaton is to decide whether $d_L(W, V)$ with $W, V \in \Sigma^*$ is smaller than a specific $n \in \mathbb{N}$. So, it is possible to decide if a word from a question is similar to a word from the STW, similar in the meaning it has a Levenshtein distance smaller than n. Therefore Σ contains the alphabet {a, ..., z, A, ..., Z}. The states in Q of the Levenshtein automaton denote the current position in the original word W, written as i and the current Levenshtein distance between ω and W, written as j, with ω prefix of V. The label of such a state is i^j. Thus the initial state $q_0 \in Q$ is 0^0. The final states $f \in F$ are all states where i equals |W| and j is smaller than n.

Let $\omega_{correct}$ be the correct letter after state i^j. The relation $\Delta : (Q \times \Sigma \times Q)$ has the following elements:

- $(i^j, \sigma, (i+1)^j)$ if $\sigma = \omega_{correct}$

- $(i^j, \sigma, i^{(j+1)})$ if σ is inserted after state i^j and $(j+1) < n$

- $(i^j, \sigma, (i+1)^{(j+1)})$, if σ is substituted by $\omega_{correct}$ and $(j+1) < n$

- $(i^j, \sigma, (i+1)^{(j+1)})$, if $\omega_{correct}$ is deleted

The elements are not exclusive therefore all of these cases can be possible after reading only one letter.
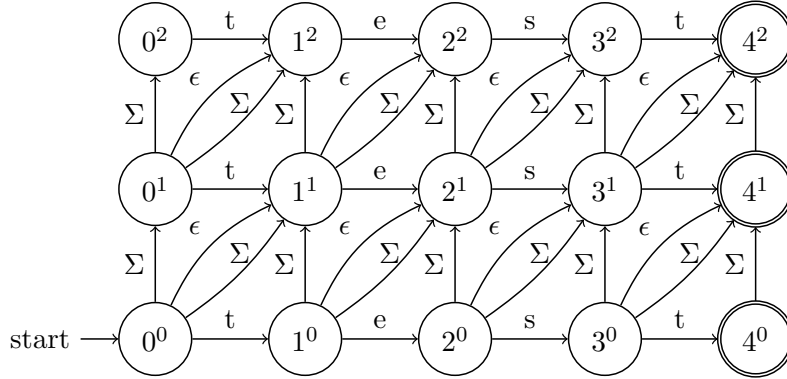
Figure 2.3: A non deterministic Levenshtein automaton for the word *test* with degree 2

The automaton in example 2.3 is a non deterministic Levenshtein automaton for the word *test*. In the following description the Levenshtein automaton represents the word $W \in \Sigma^*$ and $\sigma \in \Sigma$ is the currently read letter. Σ indicates that any element from Σ is accepted on this path. The initial state 0^0 is in the bottom left corner. If σ is a correct letter it follows the horizontal path in the automaton. A vertical path is an insertion of a letter $l \in \Sigma$ into the word W which is possible for any σ. A diagonal path can be a deletion of a letter $l \in W$ with the empty word ϵ or a

substitution of σ for any σ. Therefore after reading the letter t in the initial state 0^0 the automaton can be in seven different states namely 0^1, 1^2, 1^1, 2^2, 1^0, 2^1 and 3^2. These seven different states can be reached because the t is included in the alphabet Σ and therefore the vertical transition for inserting a letter as well as the diagonal transition for substituting a letter can be chosen. Additionally the ϵ transition which represents the deletion of a letter can be used at all times. Finally the horizontal path can be used because t is the correct letter for the current state. Moreover from each destination of a transition the ϵ transitions can be used as well. Yielding the seven different sates that were mentioned above. Evaluating a non deterministic Levenshtein automata is computational complex due to the fact that there can be a large number of active states at the same time. Thus it is necessary to convert a non deterministic automata to a deterministic automata before using it to find tags. The process of generating a deterministic automata with a non deterministic automata is called *powerset construction.*

Powerset Construction

The process of creating a deterministic Levenshtein automata out of an non deterministic Levenshtein automata is based on the powerset construction from the book *Introduction to Automata Theory, Languages, and Computation* by Hopcroft et al. [2003]. Given a non deterministic Levenshtein automaton the construction of an equivalent deterministic Levenshtein automaton is described below.

- Create q_0' as a set with original q_0 and all states that are reachable with an ϵ path.

- $Q' \subseteq 2^Q$, thus all $q \in Q'$ are subsets of Q.

- $\delta(R, a) = \{q \in Q \mid \exists r \in R$ with (r, a, q) $\in \Delta$ or (r, ϵ, p) and (p, a, q) $\in \Delta\}, R \subseteq Q$.

- F' includes all $q \in Q'$ that include $f \in F$

Therefore the new deterministic Levenshtein automaton is (Q', Σ, q_0', δ, F').
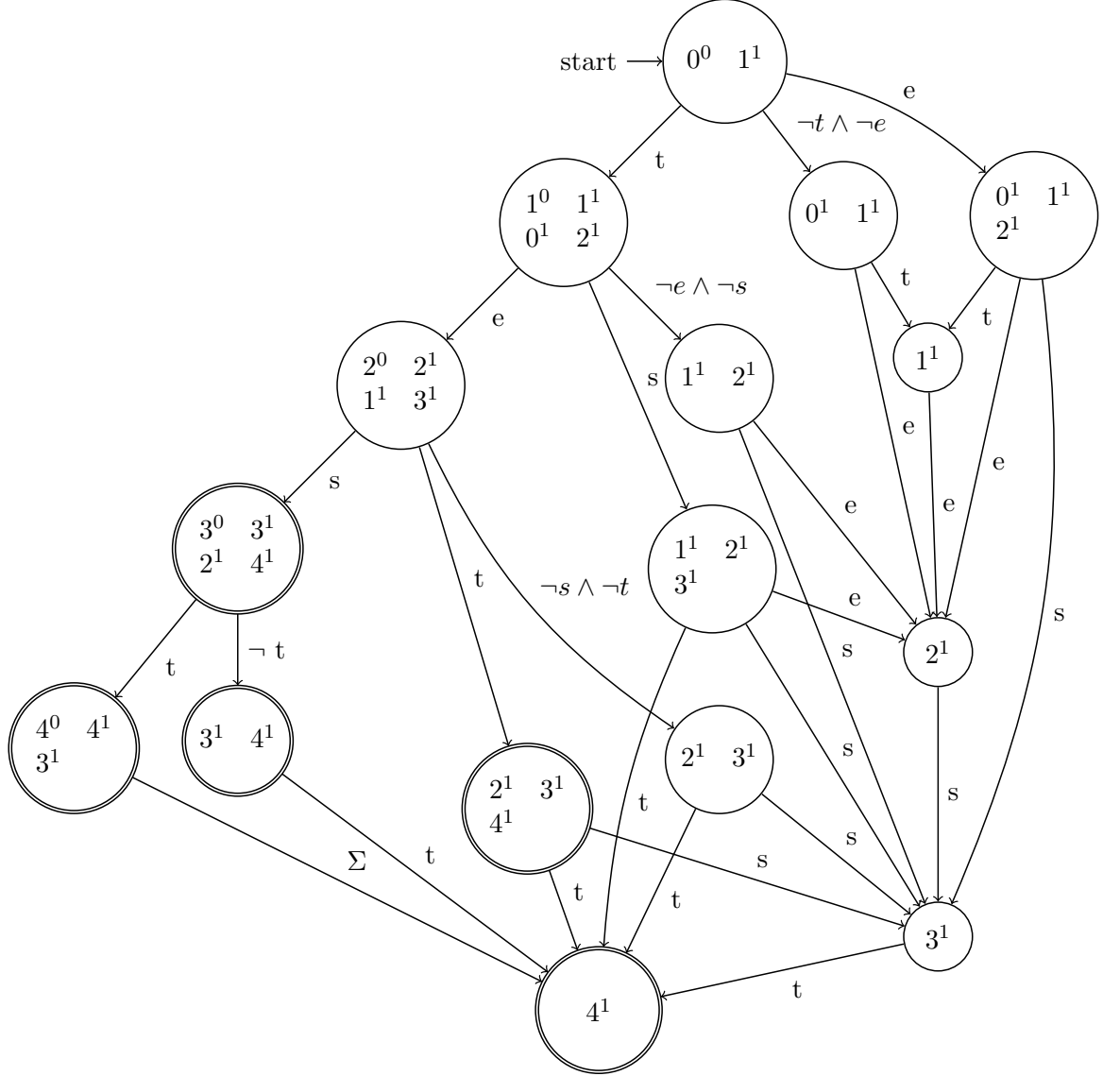
Figure 2.4: A deterministic Levenshtein automaton for the word *test* with degree 1

Let $(Q, \Sigma, q_0, \Delta, F)$ be a non deterministic Levenshtein automaton for the word *test* with maximum Levenshtein distance 1^2. Then the deterministic Levenshtein automaton $DLA = (Q', \Sigma, q'_0, \delta, F')$ from figure 2.4 is the output from the powerset construction. Consequently the DLA can only have one active state at a time and accepts all words V from Σ^* with $d_L('test', V) \leq 1$. This deterministic finite automaton with degree $n \in \mathbb{N}$ for a word $W \in \Sigma^*$ can decide in linear time if a word $V \in \Sigma^*$ has $d_L(W, V) \leq n$ [Schulz and Mihov [2002]].

---

[2]This is the automaton from figure 2.3 without the first row

---

Algorithm 2 Levenshtein Automaton is in distance

1: **procedure** ISINDISTANCE(*automata* : *DeterministicFiniteAutomata*, *term* : *String*)
2:      $i \leftarrow 0$
3:      $currentStates \leftarrow (0, 0)$ ▷ add the initial state
4:      **while** currentStates.size > 0 AND i < term.length **do**
5:          $c \leftarrow term[i].toLowerCase$ ▷ c gets the current active lower cased letter
6:          $currentStates \leftarrow automate.nextState(currentStates, c)$
7:          $i \leftarrow i + 1$
8:      **end while**
9:      **if** currentStates includes a final state **then**
10:          **return** true
11:      **else**
12:          **return** false
13:      **end if**
14: **end procedure**

With this concept for each word in an item a deterministic Levenshtein automaton can be generated. With the Levenshtein automaton it is decidable if a word is in a specific distance. However, it is not a good approach to use the same distance for all words if for example the standard distance would be three then each word that has three or less letters would match with every other word of the same size yielding a lot of false matches. Therefore there is a need for creating ranges for the degree of a Levenshtein automaton in dependence to the length of the word which is introduced in this thesis. The most occurring change of a noun is the plural form. The book *German Noun Plural reconsidered* by Thießen [2009] describes different forms of plural creation in the German language. According to Thießen [2009] the maximum number of letters that are appended as a suffix during the plural creation process for nouns is two. These suffixes are *-er* and *en*. However there are word forms that change the vowel into an umlaut during the plural creation process. Therefore the maximum Levenshtein distance between a noun and its plural form is three.

The maximum distance for a given word $W \in \Sigma^*$ depends on the length n of W. A word with a length of $n \leq 3$ in the STW is usually an abbreviation and therefore is only used as a direct match. A word with a length of $3 < n \leq 5$ is used with a maximum distance of one. Therefore all noun plural forms that are created with the appending of one suffix are matched as well as spelling mistakes with a distance of one but the cases with a distance of more than one character are not matched. This choice is made because a distance of more than one would yield too many false matches for such short words. A word with a length of $6 \leq n$ is used with a maximum distance of three. Thus, all plural forms that are created for a word with five or more letters are matched. If the stem of a word has five letters a plural form would add at least one additional letter and therefore the word would have at least six letters. This would create a deterministic Levenshtein automaton with a degree

of three for this word which would match all plural forms.

To get back to the original problem this paragraph explains the application of a Levenshtein distance automata to find the right tags for a text. The overall tagging concept is to calculate for every word from the text an automata and read all words from the STW into each automata. If a word from the STW ends in a final state it is used as a tag for the text. A possible optimization would be to calculate all the automata once for the complete STW and to store it with an efficient data structure in a database. Currently the STW is stored in a triple store because it can easily be updated with the standard files from the website. A triple store is a special database that stores data in the form of triples. The triple store technology is described in more detail in subsection 3.1.2.

2.4 Recommendation

A recommendation system is a system that creates personalized item recommendations based on information about the items or the relation between items and users. These recommendations can be created by different types of recommendation systems that exploit different information about the users or items.

2.4.1 Different types of recommendation systems

The book *Recommender Systems An Introduction* by Jannach et al. [2011] distinguishes between the following four different kinds of recommendation systems.

Collaborative recommendation

Collaborative recommendations are recommendations based on similar interests of users. So, if user A and user B are interested in similar items and user A shows interest in item I that is unknown to user B than item I might be interesting for user B as well. Due to the fact that this technique filters all items based on implicit collaboration of the users it is also known as *collaborative filtering*. For using the collaborative filtering approach no other information are needed than the relationship between users and items. Depending on the number of items in the system and the activity of the users of the system the process of collecting information about the relation between the users and the items they are interested in might be needing some time. Although it needs some time to collect initial data in the beginning it generally is a good approach if the items that need to be recommended are unknown or additional information for the items would be hard to maintain.

Content-based recommendation

In content-based recommendations the items are usually documents that should be recommended based on the content. Thus, the content of the document is described by tags. These tags can have an indicator that expresses the importance of the tag. Therefore the documents can be filtered by the tags of the documents and ranked by the importance of those tags. These tags can be maintained explicitly by the users of the systems or by tagging algorithms. One advantage of content-based recommendation systems is that they do not require a large user base to achieve good recommendations for users. A second advantage is that new items can be recommended to users immediately after the content has a description. If the content is user generated the recommendation system can only recommend documents that fit the current context. It has no information whether a user likes the content or not which can lead to poor recommendations. All in all, content-based systems are a good choice if the system has to provide recommendations for quality documents. As long as the documents have a description for the system.

Knowledge-based recommendation

The basic idea behind knowledge-based recommendation systems is a system that has enough information about the items and the needs of the user and as such can recommend items based on matching the needs of the user and the features of the items. Moreover the system has to use individual user requirements in order to create personalized recommendations. For example if a user would like to buy a new jacket for a summer camping trip in England the knowledge based system has to use the specific context in order to recommend a thin light rain jacket with the right size and price for the user. These information are usually manually provided by the user and the maintainer of the system. Furthermore not only the system needs to correctly interpret the information but the user needs to have the domain knowledge in order to provide the system with the correct information. Consequently, knowledge-based recommendation is a good approach if the system has to recommend items that are not frequently requested and no user history is available. It is especially suited for applications in which items are not frequently rated such as expensive digital goods or cars.

Conclusion

All of the described recommendation approaches have advantages and usually disadvantages. Knowledge based systems require too much domain knowledge that is not accessible for user generated questions and answers and therefore knowledge based systems are not feasible for the system. On the other hand collaborative recommendation systems need an initial amount of user item relationship data in order to present good recommendations. Additionally, they have no information about the subject of the items and this might lead to a situation were a collaborative filtering system might recommend inappropriate items to the current context. Nonetheless collaborative filtering is the only approach that takes the interest of similar users into account. As such it is the only technique that is able to recommend items that are approved by similar users. Besides the content-based systems have the disadvantage that they do not make use of the information whether or not a user is interested in a document. Thus they can only recommend items based on the context and are not able to recommend items that were interesting to other users in the past. Therefore it is possible that a content-based system recommends user generated content of poor quality which is not desirable. Due to the fact that the user generated content is not moderated and therefore the system has no information about the quality of the content the implemented system uses a collaborative filtering technique with pre filtered content based on the tags for each item. This approach should yield the best possible recommendations for a user of the system.

2.4.2 Collaborative recommendation system

The following definitions are based on Jannach et al. [2011]

Definition 10 (User). $U = \{u_1, u_2, u_3, \ldots, u_n\}$ *is a set with u_i being users from the recommendation system in which $n, i \in \mathbb{N}$ and $i \leq n$.*

Definition 11 (Item). $I = \{i_1, i_2, i_3, \ldots, i_n\}$ *is a set with i_j being items from the recommendation system in which $n, j \in \mathbb{N}$ and $j \leq n$.*

Definition 12 (Rating). *R is an $n \times m$ matrix with $n = |I|$ and $m = |U|$. Furthermore $r_{n,m}$ is a rating for item $n \in I$ and user $m \in U$ with $r_{n,m}$ being an entry in R and $r_{n,m} \in \{1, 2, 3, 4, 5\}$. If a user $k \in U$ has not rated an item $l \in I$ the entry $r_{l,k}$ remains empty. $\hat{I}_u$ is a set with all items $i \in I$ in which $r_{i,u}$ is empty and $\tilde{I}_u$ is a set with all items i and $r_{i,u}$ is not empty.*

Definition 13 (Prediction). *A prediction is a rating $r_{i,u}$ for item $i \in I$ and user $u \in U$ with $r_{i,v}$ being an empty entry in R.*

Definition 14 (Recommendation). *Let $n \in \mathbb{N}$ and $u \in U$. A recommendation is a set of n predictions for a user u ordered by the values of the predictions.*

As mentioned above a collaborative recommendation system recommends items to users based on the interest of similar users. All of the collaborative recommendation techniques can be divided into two different stages of calculations. The first calculation stage is called *offline calculation* in this thesis these include all calculations that are used to calculate the similarity between items or users. The second calculation stage is called *prediction calculation* in this thesis this stage includes all calculations that are used to calculate the rating predictions for a user-item pair. The book *Recommender Systems An Introduction* by Jannach et al. [2011] differentiates between a number of different collaborative filtering techniques which are outlined below.

User Based Nearest Neighbour Recommendations

The user based nearest neighbour recommendation technique is one of the earliest collaborative recommendation methods. It recommends items based on the similarity between users. Therefore the similarity between each user is computed and the prediction for a user-item pair is calculated by using the most similar users that rated the item and multiplying the similarity of the users with the rating the result is normalized by the sum of all similarities of the users that were used.

Item Based Recommendations

The item based recommendation method calculates rating predictions based on the similarity between the items. Thus, it is similar to the user based nearest neighbour recommendation technique but it uses the similarity between the items instead of the similarity between the users to calculate the rating predictions.

Matrix Factorization Recommendations

The matrix factorization recommendation technique can be used to derive a set of latent factors from the user-item rating matrix. These latent factors can be used as vectors which can characterize users or items. In the movie domain such a latent factor can correspond to obvious aspects of a movie such as the genre of the movie but it can also be uninterpretable. The vectors of latent factors can be used for calculating the similarities between users. Finally, these user similarities can be used for calculating the prediction for a user-item pair in the same manner that is used for the user based nearest neighbour recommendation technique.

Probabilistic Recommendations

The probabilistic recommendation technique creates groups of users and calculates the probability of group membership for each user. If a prediction has to be calculated for an user $u \in U$ and an item $i \in I$ the probability that the group in which user u belongs is used to decide whether u likes item i or not. However, the number of user groups and the maximum size of each group have to be evaluated for each use case depending on the prediction quality.

Thesis Recommendation Approach

The recommendation approach that is developed in this thesis is a combination of item based and matrix factorization recommendations. In productive recommendation systems the user-item rating table is usually very sparse [Jannach et al. [2011]]. That means that most users only rate a low number of items. Thus, the rating matrix that is used for calculating the matrix factorization has a lot of empty entries. The idea behind the approach in this thesis is to predict the missing entries of the rating matrix with the item based recommendation approach. Therefore, the matrix factorization can be calculated with a matrix that provides more rating data. Hence, the latent factors that are derived by the matrix factorization should be more distinctive and should result in better user similarities. For the part of the matrix factorization this thesis uses the *singular value decomposition* which is a well known matrix factorization technique. The following pages explain the item based recommendation technique and the recommendation technique based on the singular value decomposition in more detail.

Item-Based Recommendation

The concept of item-based recommendation was introduced by Sarwar et al. [2001]. The idea behind this concept is that recommendations can be calculated based on similar items of the items that a user rated. If $i \in \hat{I}_u$ and $u \in U$, then the prediction for i can be calculated using the similarity between i and all items $j \in \tilde{I}_u$. Researches made by Sarwar et al. [2001] show that the *adjusted cosine similarity* has the best performance with attention to prediction accuracy compared to all other similarity

measures for item based algorithms. The adjusted cosine similarity is a special form of the *cosine similarity* which is explained in the following definition which is based on Jannach et al. [2011].

Definition 15 (Cosine Similarity)*. One has $i, j \in \mathbb{N}^n$, $n \in \mathbb{N}$ so that*

$$cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{|| \vec{i} ||_2 \cdot || \vec{i} ||_2} = \frac{\sum_{i=1}^{n} \vec{a_i} \vec{b_i}}{\sqrt{\sum_{i=1}^{n} \vec{a_i}^2} \sqrt{\sum_{i=1}^{n} \vec{b_i}^2}} \tag{2.1}$$

. In general the results of the cosine similarity vary between -1 and 1. However, the values of the vectors in the use case of the recommendation system are in the range of 1 to 5. Therefore the vectors do not include negative values and the cosine similarity in the domain of the recommendation systems vary between 0 and 1. According to Jannach et al. [2011] results of the cosine similarity near to 1 mean strong similarity between the items. If $\vec{i}$ and $\vec{j}$ are rating vectors, the individual rating behaviour of a user needs to be taken into account to improve the similarities between $\vec{i}$ and $\vec{j}$. Due to the fact that some users tend to rate items in general higher or lower than other users. Yielding different average ratings for different users. This results in the adjusted cosine similarity.

Definition 16 (Adjusted Cosine Similarity)*. Let $i, j \in I$(Items) and $\overline{r_u}$ be the average rating from user u. The cosine similarity can be transformed to the adjusted cosine similarity by subtracting the average user rating from the current rating. Let $\tilde{U}$ be all users that rated item i and item j.*

$$sim(i, j) = \frac{\sum_{u \in \tilde{U}} (r_{u,i} - \overline{r_u})(r_{u,j} - \overline{r_u})}{\sqrt{\sum_{u \in \tilde{U}} (r_{u,i} - \overline{r_u})^2} \sqrt{\sum_{u \in \tilde{U}} (r_{u,j} - \overline{r_u})^2}} \tag{2.2}$$

The results of the adjusted cosine similarity are between -1 and 1 with 1 meaning identical, 0 meaning indifferent and -1 meaning opposite [Jannach et al. [2011]]. The user item predictions can be generated with the similarities by using the following formula which is based on Jannach et al. [2011].

Definition 17 (Item Based Prediction)**.**

$$pred(u, p) = \frac{\sum_{i \in \tilde{I}_u} sim(i, p) \cdot r_{u,i}}{\sum_{i \in \tilde{I}_u} sim(i, p)} \tag{2.3}$$

The item similarity calculations can be calculated offline on a regular basis - every day or weekly - depending on the activeness of the users and the amount of item changes. So, the item based technique is a good choice for websites that need fast scalable recommendations because only the item predictions are calculated on time. Furthermore the team of Sarwar et al. [2001] found out that the system only needs a subset of all items. According to Sarwar et al. [2001] a sample of the 25 most similar items is needed to generate good recommendations. Thus the on time

calculation needs to find the top 25 most similar users in the similarity table and has to calculate the recommendations with these users instead of using all similar users. Moreover, it is a tested concept for instance amazon.com uses item based recommendations for their product recommendations [Linden et al. [2003]]. To get back to the overall concept, the item based predictions are used in order to decrease the rating sparsity of the user-item table. Consequently, more information about the user's interest is available before the actual user similarities are calculated with the singular value decomposition which is explained in the following section. In real world recommendation systems the user item ratings tend to be very sparse because most users only rate few items [Jannach et al. [2011]].

Example for Item-Based Recommendations

The table in figure 2.5 represents a user item relationship.

	Item1	Item2	Item3	Item4	Item5
User1	5	3	4	4	?
User2	3	1	2	3	3
User3	4	3	4	3	5
User4	3	3	1	5	4
User5	1	5	5	2	1

Figure 2.5: User Item Relationship Table

This example calculates the item based prediction for User1 and Item5. For the adjusted cosine formula the average ratings for all users are needed.

$$\overline{r}_{User1} = \frac{\sum\limits_{i \in \tilde{I}_{User1}} r_{User1,i}}{\sum\limits_{i \in \tilde{I}_{User1}} 1} = \frac{5+3+4+4}{4} = 4.0$$

The following average ratings are calculated analog.
$\overline{r}_{User2} = 2.4$
$\overline{r}_{User3} = 3.8$
$\overline{r}_{User4} = 3.2$
$\overline{r}_{User5} = 2.8$
The following formula calculates the adjusted cosine similarity from definition 16 between Item5 and Item1.

$$sim(Item5, Item1) =$$
$$\frac{(3-2.4) \cdot (3-2.4) + (5-3.8) \cdot (4-3.8) + (4-3.2) \cdot (3-3.2) + (1-2.8) \cdot (1-2.8)}{\sqrt{(3-2.4)^2 + (5-3.8)^2 + (4-3.2)^2 + (1-2.8)^2} \cdot \sqrt{(3-2.4)^2 + (4-3.8)^2 + (3-3.2)^2 + (1-2.8)^2}} =$$
$$0.8040$$

Similar to this calculation one computes the degree of similarity between Item5 and Item2, Item3 as well as Item4. The results are shown below.

sim(Item5, Item2) = -0.9082

sim(Item5, Item3) = -0.7635

sim(Item5, Item4) = 0.4330

With these similarities the following prediction can be calculated using the item based prediction formula of definition 17.

$$pred(User1, Item5) = \frac{0.8040 \cdot 5 + (-0.9082) \cdot 3 + (-0.7635) \cdot 4 + 0.4330 \cdot 4}{0.8040 + (-0.9082) + (-0.7635) + 0.4330} = 4.6501$$

As a result User1 would most likely rate Item5 with 4.6501.

Singular Value Decomposition

The singular value decomposition is a matrix factorization technique that can be used to find latent factors in the rating patterns. With these factors it is possible to find similar users and items. The singular value decomposition was found in the late 1800 to early 1900 [Stewart [1993]]. Besides one of the main areas of application of the singular value decomposition today is information retrieval. The basic idea behind the singular value decomposition based information retrieval is to match user queries to documents [Deerwester et al. [1990]].

Definition 18 (Linear Dependent, Linear Independent)**.** *Let $v_1, v_2, \ldots, v_n$ be a list of vectors in $\mathbb{R}^k$ whereby $k \in \mathbb{N}$. This list of vectors is called linear independent if the only possible solution of $a_1, a_2, a_3, \ldots, a_m \in \mathbb{R}$ that makes $a_1 v_1 + a_2 v_2 + \cdots + a_n v_n$ equal to 0 is $a_1 = \cdots = a_n = 0$. A list of vectors is called linear dependent if they are not linear independent. In other words if at least one element of $a_1, a_2, a_3, \ldots, a_m$ is not null such that $a_1 v_1 + \cdots + a_n v_n$*

Definition 19 (Rank of a Matrix)**.** *$https://en.wikipedia.org/wiki/Rank_(linear_algebra)$ The rank of a matrix A is the maximum number of linearly independent column vectors of A. Due to the fact that the rank of the linearly independent column vectors is always equal to the number of linearly independent row vectors it is also correct that the rank of matrix A is the maximum number of linearly independent row vectors of A.*

Definition 20 (Unit Vector)**.** *A unit vector is a vector with length 1. Let $v \in \mathbb{R}^k$, $k \in \mathbb{N}$ and v is a unit vector then the following equation becomes true.*

$$||v|| = \sqrt{\sum_{i=1}^{k} v_i^2} = 1$$

Definition 21 (Orthogonal Vector)**.** *Two vectors are orthogonal if the dot product of those vectors is 0. Let $v, w \in \mathbb{R}^k$, $k \in \mathbb{N}$ and v, w are orthogonal to each other*

then the following equation becomes true.

$$v \cdot w = \sum_{i=1}^{k} v_i \cdot w_i = 0$$

Definition 22 (Orthogonal Matrix). *An orthogonal matrix $A \in \mathbb{R}^k$ whereby $k \in \mathbb{N}$ is a quadratic, real matrix whose column and row vectors are pairwise orthonormal together. Furthermore is the inverse of the matrix equal to the transpose of the matrix.*

$$A^{-1} = A^t$$

Additionally the product of an orthogonal matrix with its transpose results the identity matrix.

$$AA^t = I$$

Definition 23 (Eigenvalue of a Matrix). *An eigenvector of a square matrix A is a non-zero vector v that, when multiplied by A, yields the original vector multiplied by a single number λ; that is:*

$$Av = \lambda v$$

The number λ is called the eigenvalue of A corresponding to v.

The singular value decomposition of an $m \times n$ matrix M with rank r is a factorization of the form:

$$SVD(M) = U\Sigma V^t$$

U and V are orthogonal matrices with dimension $m \times r$ and $r \times n$ respectively. Furthermore Σ is a rectangular diagonal matrix with dimension $r \times r$. The diagonal values $\sigma_{i,i} \in \Sigma$ with $i \in \mathbb{N}$ have by convention the property $\sigma_{i,i} \geq \sigma_{i+1,i+1} > 0$. The $\sigma_{i,i}$ are the none negative square roots of the eigenvalues of AA^t and A^tA [Allaire and Kaber [2008]][3]. The columns of U are the corresponding eigenvectors to the eigenvalues of AA^t. On the other hand the columns of the Matrix V are the corresponding eigenvectors to the eigenvalues of A^tA [Allaire and Kaber [2008]][4]. This results in: $AA^t \cdot u_i = \sigma_{i,i}^2 \cdot u_i$ and $A^tA \cdot v_i = \sigma_{i,i}^2 \cdot v_i$.

To sum this up, the matrix U corresponds to the columns of the matrix A and the matrix V corresponds to the rows of the matrix A[Sarwar et al. [2002]]. It is possible to obtain an optimal rank k approximation from matrix A with $k \leq r$ by setting all $\sigma_{i,i}$ with $i > k$ to zero [Stewart [1998]]. Thus $A_k = U_k \Sigma_k V_k^t$ yields the optimal rank k approximation. As a result, all column vectors u_i with $i > k$, of matrix U will be multiplied by a zero vector from matrix Σ_k. Due to the fact that matrix U represents the columns of matrix A and that the last rows of matrix U will be removed in order to generate the optimal rank k approximation of matrix A, it is possible to generate a good approximation of the user's interests by removing the last rows of matrix

---

[3]AA^t and A^tA share the same eigenvalues.
[4]The eigenvalues are the squares of $\sigma_{i,i}$

U^5. The two dimensional matrix U_2 is created by removing all rows of matrix U up to the first two rows. This matrix U_2 can be seen as a value table that can be represented by a graph. Therefore, it enables a graphical presentation of the user similarities. Furthermore, the user similarities can be computed by using the cosine similarity between the columns of U_k.

After calculating the similarities for each user, the recommendations for $user_a \in U$ are calculated. The following prediction calculation process is based on Jannach et al. [2011]. The prediction calculation algorithm takes all similar users U_{sim} for $user_a$ and uses the top rated items from all $user_b \in U_{sim}$ that are not rated by $user_a$. This is denoted by $I'_{user_a, user_b}$. $\overline{r_a}$ which constitutes the average rating of user a. Finally for each item $i \in I'_{user_a, user_b}$ the prediction is calculated.

$$pred_{a,i} = \overline{r_a} + \frac{\sum_{b \in U_{sim}} (r_{i,b} - \overline{r_b}) * similarity_{a,b}}{\sum_{b \in U_{sim}} similarity_{a,b}} \qquad (2.4)$$

Example for a singular value based recommendation

	Item1	Item2	Item3	Item4	Item5
User1	5	3	4	4	?
User2	3	1	2	3	3
User3	4	3	4	3	5
User4	3	3	1	5	4
User5	1	5	5	2	1

The following example calculates the rating prediction for *User1* and *Item5*. A singular value decomposition for a user item relationship table is created in figure 2.6. The first two rows of matrix U are used to create a value table for a graph that represents the user similarities. This is displayed in figure 2.7. The cosine similarity from definition 15 is used to calculate the following similarities.

$$sim(User1, User2) = \frac{0.475467 \cdot 0.342950 + 0.325691 \cdot -0.316949}{\sqrt{0.475467^2 + 0.325691^2} \cdot \sqrt{0.342950^2 + (-0.316949)^2}} = 0.222322$$

In an analogous manner the other similarities are achieved:

- sim(User1, User2) = 0.222322

- sim(User1, User3) = 0.561072

- sim(User1, User4) = 0.151704

- sim(User1, User5) = 0.896770

The following equation calculates the average rating for User1.

$$\overline{r}_{User1} = \frac{\sum\limits_{i \in \tilde{I}_{User1}} r_{User1,i}}{\sum\limits_{i \in \tilde{I}_{User1}} 1} = \frac{5 + 3 + 4 + 4}{4} = 4.0$$

---

[5] A user from the system is represented by a column in the matrix A

The following average ratings are calculated analog.

$\bar{r}_{User2} = 2.4$

$\bar{r}_{User3} = 3.8$

$\bar{r}_{User4} = 3.2$

$\bar{r}_{User5} = 2.8$

With these similarities and average ratings the following prediction which is defined in equation 2.4 can be calculated.

$$pred(User1, Item5) =$$

$$4 + \frac{(3 - 2.4) \cdot 0.222322 + (5 - 3.8) \cdot 0.561072 + (4 - 3.2) \cdot 0.151704 + (1 - 2.8) \cdot 0.896770}{0.222322 + 0.561072 + 0.151704 + 0.896770} = 3.6254$$

As a result User1 would most likely rate Item5 with 3.6254.

	Item1	Item2	Item3	Item4	Item5
User1	5	3	4	4	
User2	3	1	2	3	3
User3	4	3	4	3	5
User4	3	3	1	5	4
User5	1	5	5	2	1

$$=$$

$$\begin{pmatrix} 0.475467 & 0.325691 & 0.798636 & -0.053513 & -0.164835 \\ 0.342950 & -0.316949 & 0.082532 & -0.250224 & 0.844099 \\ 0.535742 & -0.210593 & -0.362530 & -0.589189 & -0.435955 \\ 0.455479 & -0.486113 & -0.054206 & 0.729352 & -0.146077 \\ 0.402286 & 0.716108 & -0.470107 & 0.235423 & 0.221199 \end{pmatrix} \cdot$$

$$\begin{pmatrix} 15.627834 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 5.104899 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 3.541202 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 2.426008 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.534013 \end{pmatrix} \cdot$$

$$\begin{pmatrix} 0.46825 & 0.432206 & 0.46056 & 0.487586 & 0.379564 \\ -0.177671 & 0.42127 & 0.572180 & -0.250389 & -0.633148 \\ 0.609379 & -0.316926 & -0.139853 & 0.322858 & -0.635938 \\ -0.392214 & 0.489216 & -0.480126 & 0.571026 & -0.224148 \\ -0.473276 & -0.544015 & 0.458702 & 0.518898 & -0.019831 \end{pmatrix}$$

Figure 2.6: Singular Value Decomposition of the User Item Relationship Table

$$
\begin{pmatrix} User1 \\ User2 \\ User3 \\ User4 \\ User5 \end{pmatrix} = \begin{pmatrix} 0.475467 & 0.325691 \\ 0.342950 & -0.316949 \\ 0.535742 & -0.210593 \\ 0.455479 & -0.486113 \\ 0.402286 & 0.716108 \end{pmatrix}
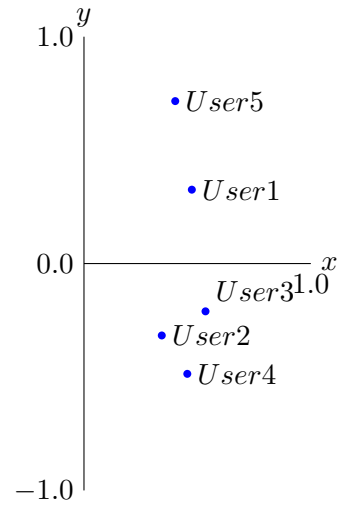$$

Figure 2.7: Similar Users based on Singular Value Decomposition

2.4.3 The recommendation process

The last section describes the concept of the item based and SVD based recommendations. This section explains the combination of these two concepts to create a new recommendation system that is evaluated in this thesis. The sparsity of the user item table is decreased by using item based predictions for user-item pairs that are not rated. For this process the system uses the similarities between rated and not rated user-item pairs. For each user (u) the rating for a not rated item (i) is predicted by using the similarity of i and items that are rated by u. This improved user item table is decomposed into three matrices U, Σ and V^t. The matrix U represents the user interests and is reduced by k rows in order to improve the execution time of the recommendation process. The cosine similarity is calculated between the columns of the matrix U_k. Consequently, the column vectors have less entries due to the approximation and the cosine similarity calculation is faster than with the original matrix U. With these similarity information the recommendations are generated by finding the top rated items of the most similar users that are unknown for the current user. If the current user of the system is not registered, the system uses the default user to calculate the recommendations. This default user is the first user of the system who should be generated by the system administrator.

2.4.4 Different Recommendation approaches

The explained techniques that are used in the recommendation process can also be used for other recommendation processes which are presented below.

Only Item-Based Recommendations

This recommendation system concept uses item-based recommendations to decrease the sparsity of the rating table. Thus the item-based method calculates similarities between items and with these similarities the empty user-item ratings are predicted if enough similar items are found. However, the item similarities can be used to directly generate recommendations for users. Yielding a functional recommendation system which will be compared with the main recommendation system in section 4.3.

Only SVD-Based Recommendations

The SVD-based recommendation system is part of the recommendation concept that is introduced in this thesis. This SVD-based recommendation system calculates the user-item recommendations after the item-based system decreased the sparsity of the rating table. Consequently, the SVD-based recommendation system can be used as a standalone recommendation system. However, a rating matrix that is used by the singular value decomposition must not have empty entries in the matrix otherwise the singular value decomposition can not be calculated for this matrix. Hence the empty entries of the rating matrix must be filled with values before calculating the

singular value decomposition. For this process two different approaches are tested in the evaluation process. The first one fills the empty entries with the average value of the 1 to 5 rating scale namely the 3. This approach is called *SVD Three* in the evaluation chapter. The second one uses the average item ratings for the missing entries. This approach is called *SVD Average* in the evaluation chapter. The runtime and performance of these systems is compared with the recommendation system of this thesis in section 4.3.

2.5 Concept Relations

The last sections describe the different concepts that are used in this thesis. This section emphasizes on the relations between these different ideas. The question and answer system enables users of a website to create questions and answers. These questions and answers are the *items* that are used as recommendations on sites that have the same subject. For this process the items need to have a rating that indicates the interest that a user has for this item. These ratings are generated by the implicit rating concept that is introduced in this thesis. Additionally, the ratings are used by the recommendation system concept for calculating the rating predictions. Besides the items need a tag that describes the content of the item for filtering the recommendations before they are used on the website. These tags are generated with the explained tagging concept. The final outcome is a question and answer recommendation list that is ordered by the predicted interest for a user. Moreover, the recommendation list matches the current subject of the website.

3 Implementation

This chapter begins with a discussion about the choices of technology for the different system parts. Afterwards the service oriented software architecture and the different interfaces to those services are presented and explained.

3.1 Technologies

The developed system combines five different concepts namely the question and answer system, the rating system, the tagging system and the rating system as well as the recommendation system. For such a variety of techniques with different requirements there is the need for different technologies which are outlined in the following paragraphs.

3.1.1 Programming Language

First of all one requirement for the programming language is that it should be object oriented for good programming code encapsulation. Besides the software should work on a number of different operating systems due to the fact that the development takes place on a mac operating system and the final software has to run on a linux server. Moreover it needs to generate fast and reliable software programs as well as it should bring libraries for fast linear algebra calculations. The linear algebra libraries are needed for the singular value decomposition. For the singular value decomposition different matrix calculations are necessary. Those must be implemented the way that they are able to calculate the results as fast as possible because otherwise the whole recommendation process would be too slow to be used on a productive system. Such a fast execution time can only be achieved by using different tricks that utilize specific programming language features. This implementation process would be too time consuming for a six month thesis. Fortunately a programming language that meets these requirements is the Scala programming language.

Scala Programming Language

The first version of the Scala programming language was released in the year 2003 the current version is 2.10.2 at time of writing. According to Wikipedia, The Free Encyclopedia [2013b] many large companies such as Twitter, Siemens, SAP, eBay, Tumblr, IBM and Amazon.com are using the programming language in production mode. So, Scala has been developed for 10 years and has reached a maturity where it is

safe to use it as a programming language for a system that should run in production mode.

The Scala programming language is an object oriented and functional programming language. Meaning that it is possible to write source code with a functional programming paradigm together with an object oriented programming paradigm. In addition every value in Scala is an object and every function is a value. The advantage of a functional programming paradigm is that it enables a more mathematical programming approach due to the fact that functions can be passed as parameters and can be used as return values. This is helpful for designing complex algorithms. Additionally, Scala offers the functionality to mark all operations on collections as parallel. Consequently the language supports developers to write programs that can utilize multicore computer systems. The Scala programming language compiles to the Java byte code and therefore runs in the *Java runtime environment*. Thus, it is possible to use applications that are developed with the Scala programming language on any system that is able to run the Java runtime environment which is almost any operating system. Likewise an application that is developed with the Scala programming language can use any Java library. Moreover Hundt [2011] analyzed four different programming languages namely C++, Go, Java and Scala. He implemented the same algorithm in each of these programming languages and analyzed the outcome on their individual performance, code size, binary size and complexity. The Scala implementation of this algorithm has the fastest execution time and the lowest number of lines of code of all non optimized implementations. The only implementation that achieves a shorter execution time is a highly optimized C++ version that utilizes optimized non standard C++ data structures. For these reasons the Scala programming language is used as the server side programming language. However the part of the rating systems that collects the user activities needs to run on the client side inside the web browser. This is currently only possible with the JavaScript programming language. For that reason JavaScript is used as a second programming language for the development of the system which is explained in more detail in the next subsection.

JavaScript

As mentioned in the last section the JavaScript programming language has to be used to create client side browser applications which is needed for the part of the rating system that collects the user activities. JavaScript is a scripting language. As a consequence, it can be interpreted and executed without being compiled first. The language is supported by almost every web browser and therefore is used by web pages to interact with the user and to dynamically load and change the website. As of the *W3C Document Object Model Events* specification the browser has to provide events for detecting scroll and focus events. The W3C Document Object Model Events specification is a web standard and therefore implemented by almost all web browsers. Callback functions such as activity timers can be attached to the callback functions of the scroll and focus events. These callback functions are called if the

event is activated. This enables the rating system to detect the user activities that it needs to calculate the user ratings. Therefore all technical requirements for the programming language are fulfilled with the use of the Scala programming language in combination with the JavaScript programming language.

3.1.2 Databases

The system uses two different databases one for the main system and a second database for the *STW Thesaurus for Economics*(STW). This architectural choice was made in order to use the original STW database files from the website. The first database that will be discussed is the *PostgreSQL* database.

PostgreSQL

The main database is a PostgreSQL database that is in use by the *askbot question and answer system* as well as the implemented system. The PostgreSQL database is an open source relational database which is free to use. Moreover it is to the greatest possible extend conform with the SQL standard *ANSI-SQL 2008*. Thereby the main functionality from this standard is available and works as expected.

4Store Triple Store

As mentioned at the beginning of this database section the triple store is used in order to be able to directly import the *resource description framework* (RDF) files from the STW. A triple store is a database that is optimized for storing and retrieving triples. Such a triple is a data entity composed of subject-predicate-object. Whereas the subjects and the predicates are resources the object might be a value or a resource as well. The subject is described by the predicate with the help of the object. For example the subject can be an identifier for a person the predicate can be a resource for a name and the object is the name itself.
ID:5462, name, 'John Doe'
The triples can be retrieved via a query-language which is called *SPARQL*. The triple store that is used in the implemented system is *4store* which is a free to use triple store developed and maintained by *Garlik*. The 4store can be used as a service that uses the HTTP protocol for querying the database. This fits the service oriented system architecture that is used in this thesis. Furthermore, an important aspect for the use case of the triple store in this thesis is the retrieving of ordered lists of objects. These order function is needed for dividing all words that are stored in the triple store into parts of 1000 words per request by using the ordered by, limit and offset command. This is used for processing 1000 words at once for the tagging algorithm. The triple store performance benchmark of Mironov et al. [2010] tests five triple store implementations for the average response time of different queries. This benchmark shows that the 4Store is the fastest triple store for the task of responding to requests which uses *ordered by* and that it has in comparison with the other triple store implementations an overall good performance result.

3.1.3 Network Communication

The implemented system uses the *Hypertext Transfer Protocol (HTTP)* as a network communication protocol. The network communication for this system is handled by the *finagle* open source framework[1] that is developed and maintained by *Twitter*.

Hypertext Transfer Protocol (HTTP)

HTTP is a stateless network protocol that is used by web browsers to access content in the world wide web. This protocol specifies how clients request data and how servers respond to those requests. The HTTP protocol is used in combination with the Uniform Resource Locator (URL). This URL is a combination of a host, a port and a path in the following form: http://host[:port][path]
The host specifies the server address that provides the requested service while the port is an optional number for further specifications. The path describes a specific resource that the client can access on the server. The implemented system uses the port for selecting the service. Consequently each service is accessible with a specific port. The path is used for accessing the methods that the service provides. For example http://localhost:10000/recommendation/5/3/marketing/unternehmen calls the recommendation method on the *WebService* that is running on the current computer. This method generates three item recommendations for the user with id 5 that are tagged with *Marketing* and *Unternehmen*. The implemented system uses four different HTTP request methods that are defined by the HTTP 1.1 standard[2]. These requests are the HTTP Get request, the HTTP Post request, the HTTP Put request and the HTTP Delete request. The HTTP Get request can access resources but cannot provide additional data other than the data that it specifies in the URL. The HTTP Post request can also access resources and can specify additional data to the path in the body of the request that should be added to the resource. The HTTP Put request is used for requesting that the specified data is stored or updated on the specific URL. The HTTP Delete request is used for requesting that the specific resource on the URL is deleted.

Finagle Framework

The *finagle* framework is an extensible remote procedure call system with the purpose of creating high performance and concurrent servers. On the one hand it provides interfaces for creating services that can be bound to a combination of a network address and a protocol for the network communication. As such a server that waits for a connection on the specific network address in order to execute the implemented service. On the other hand it provides interfaces for creating clients that must specify one or more server addresses together with the protocol and the connection limit. Thus, it creates a client that can execute methods on a pool of

---

[1]http://twitter.github.io/finagle/
[2]http://www.w3.org/Protocols/

servers and handles simple load balancing with the connection limit on its own. Furthermore it uses its own data structure *Future* as asynchronous return values. Future is a data structure that is returned as a value before the server finishes its execution. Consequently the client gets the future as a return value and can continue with its operation until it needs the result from the server. In that case the Future can either provide the value from the server directly or it has to block until the server finishes the operation. However the client and the server should be build in such a way that the server is able to return a value before the client actually needs the return value. As an additional feature the framework offers the possibility to bind functions on Future objects. Thus the functions will be executed if the server returns the value to the Future object. This yields a non blocking service architecture.

3.1.4 Testing

The implemented application uses two different test systems. One of the systems is a testing framework called *Spec2* that can run predefined tests in order to check on a regular basis if the system works as expected. The second system is a graphical user interface build with *backbone.js* that enables a user to create a user item table in order to test the recommendation process.

Spec2 Test Framework

As mentioned before Spec2 is a library for writing application specifications that are executable. This works by defining test classes for each class that should be tested. These test classes must have the same package structure as the original implementations. Moreover it is necessary to define a test inside the test class that checks if a function works as expected. The developed system uses the Spec2 test framework to check if all core components such as services function as anticipated.

Graphical User Interface

The graphical user interface is useful for analyzing the recommendation techniques. The purpose of this application is to illustrate how the recommendation process works in the praxis. The figure 3.1 displays the recommendation test website. On the top of the page the user can define the user ratings for the items. Depending on these entries the user and item similarities are calculated which is displayed by the diagram and the two tables underneath the diagram. Additionally the recommendation test website can calculate rating predictions for the users that have empty rating entries in the table.

The user interface is implemented with the *backbone.js* framework that provides models, collections and views for creating model view controller websites on the client side. The advantage of such a framework is that it supports the development of creating client side applications that communicate with the database. Therefore the framework provides objects that represent the data from the database and sends request to the server if data is modified added or deleted. The server has to provide

Recommendation Test Website

	Item1	Item2	Item3	Item4	Item5
User1	5	3	4	4	
User2	3	1	2	3	3
User3	4	3	4	3	5
User4	3	3	1	5	4
User5	1	5	5	2	1

[Add User] [Delete Last User] [Add Item] [Delete Last Item]
Calculate Similar Items/Users

	User1	User2	User3	User4	User5
User1		0.2223223283046248	0.5610720422329267	0.1517041248183931	0.8967700193170981
User2			0.9317894760442038	0.9974160467148566	-0.2320506579614467
User3				0.9033033008383526	0.13686824191941535
User4					-0.30133173676035757

	Item1	Item2	Item3	Item4	Item5
Item1		-0.8374636492840569	-0.3831990113701551	0.3680527005528452	0.8049144823791295
Item2			0.6136282707034809	-0.4081170077531442	-0.9082318755225987
Item3				-0.814281740038144	-0.763560385670225
Item4					0.4330626889286792

SVD Based Recommendations

Item	Rating
Item5	3.1254409736518456

Item Based Recommendations

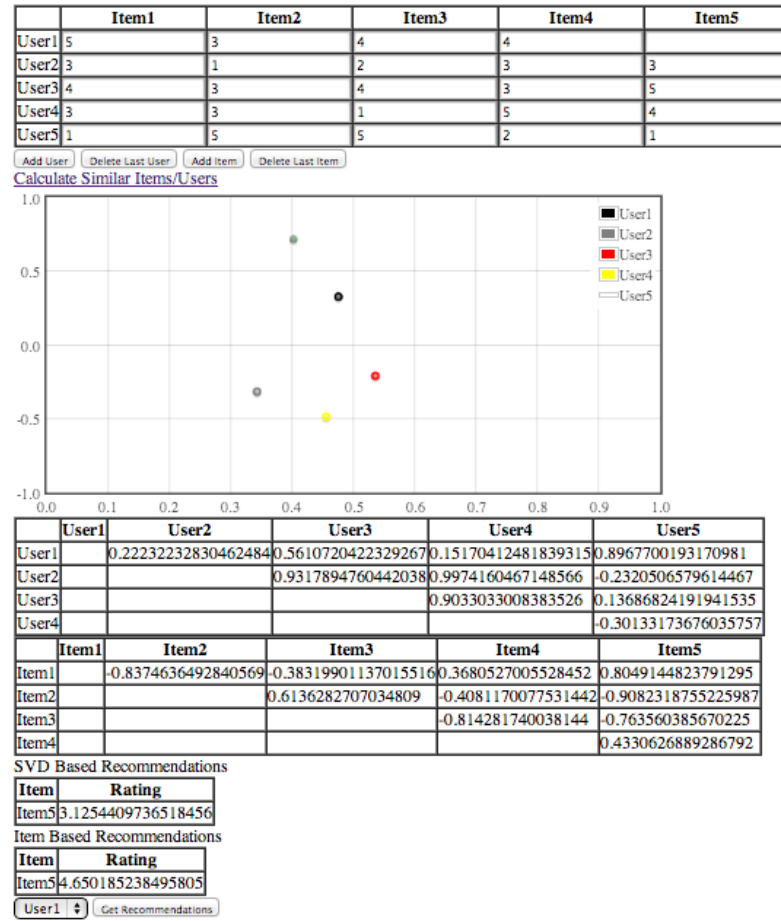| Item	Rating
Item5	4.650185238495805

[User1 ▼] [Get Recommendations]

Figure 3.1: The Graphical User Interface for The Recommendation Process

interfaces for the backbone.js framework that are based on HTTP Get, Post, Put and Delete methods for reading, updating, deleting and storing data in the database.

3.2 Architecture

The system uses a service oriented architecture. Thus, the main components of the system namely the tagging process, the rating system and the different recommendation techniques are encapsulated in different services. One advantage of that architectural choice is that the system can be spread on different servers. This does not only open the possibility to run services that have high hardware requirements on different servers but also enables the choice to duplicate services. The duplication of services can improve service downtimes and can decrease service respond times

if the service is under high load. A disadvantage of a service oriented architecture is that it is more complex to debug system errors because each service must be debugged on its own as well as the network communication between those services. However one requirement for this system is that the single system components should be usable separately. With a service oriented architecture each service can be used individually over a network communication. The network communication protocol that is necessary for communicating with the services is the HTTP protocol which is explained in subsection 3.1.3. The next sections describe the single services and the methods that they provide.

3.3 Services

The individual parts of the system are separated into different services. Each service inherits from the implemented HttpServer. This HttpServer handles the decomposing of the http request, provides simple HTTP routing and creates the http responses for the network communication. Thus the services implement the service specific logic and provide methods for accessing this logic. These methods are accessible for the router that is inherited by the HttpServer and provide a HTTP path for each method in form of a string. Consequently the services are accessible on their specific port with HTTP Post and HTTP Get requests that correspond to the HTTP path.

3.3.1 RecommendationService

The recommendation service controls the item based service and the svd based service. Therefore the single recommendation techniques can be regulated by editing the recommendation service. This opens the possibility to use different recommendation approaches without much programming effort.

RecommendationService		
Standard Port	13000	
Get Methods	Parameter: Type	Return Value
/calculateSimilarities	-	
/calculateUserPrediction	user id: Integer item id: Integer	List (user, rating)
Post Methods	Parameter: Type	
/generateRecommendations	user id: Integer amount: Integer	List (user, rating)

The /calculateSimilatities method starts the item similarity calculation of the ItemBasedService. After the calculation finishes the RecommendationService starts the svd based similar user calculation of the SVDBasedService. These similarity values are stored in the database by the individual service for future rating predictions. The /calculateUserPredictions method takes a user id and an item id as parameters. If the item id is empty, the prediction for all items are calculated as long as enough

similar item ratings exist. The final prediction calculation is handled by the SVD-BasedService. The /generateRecommendations method takes a user id, an amount of predictions that should be returned and a list of tags as parameters. This method forwards the parameters to the SVDBasedService and returns the list of item rating pairs to the client that send the request.

3.3.2 ItemBasedService

The ItemBasedService is a service that is responsible for calculating the item similarities and to predict user ratings to decrease the sparsity of the user-item table. It is the implemented item based recommendation concept and therefore can be used as a part of the complete recommendation concept that is introduced in this thesis. But at the same time it is a full functional recommendation system on its own. In the evaluation chapter (section 4.3) the ItemBasedService is used as a standalone recommendation system and the results are compared with the complete recommendation concept.

ItemBasedService		
Standard Port	13100	
Get Methods	Parameter: Type	Return Value
/calculateSimilarItems	-	-
/calculateUserPrediction	user id: Integer item id: Integer	List (user, rating)
Post Methods	Parameter: Type	
-	-	-

The /calculateSimilarItems method calculates the similarity between all items. The idea behind the similarity calculation is that the algorithm only considers to calculate the item similarity of items if they are rated together with other items by the same user. The algorithm 3 is based on the work of Linden et al. [2003]. Let N be the number of items in the system and M be the number of users who rated that item. Then as Linden et al. [2003] stated the worst case runtime of algorithm 3 is $O(N^2M)$ but due to the reason that most rating tables are very sparse the runtime on a productive system is closer to $O(NM)$. The /calculateUserPrediction method takes a user id and an item id as parameters. It calculates the rating prediction for the user-item pair and returns it as a list with one element. If the item id is empty it calculates the user predictions for all unrated items for the specific user and returns it as a list.

3.3.3 SVDBasedService

The SVDBasedService implements the concept of the singular value decomposition based recommendation system. This concept states that a user-item matrix can be decomposed into three matrices. One of these matrices has the property that the columns of this matrix respond to the user interest of the original user-item

---

Algorithm 3 Item Similarity Calculation

1: **procedure** ITEMSIMILARITY
2:     items ← load all items
3:     seenTogether ← empty Set of item tupel
4:     **for** each $item_1$ in items **do**
5:         newItems ← empty Set of items
6:         **for** each user that rated $item_1$ **do**
7:             **for** each $item_2$ that is rated by user **do**
8:                 **if** $item_2 \neq item_1$ and seenTogether does not contain $(item_2, item_1)$ **then**
9:                     add $(item_2, item_1)$ to seenTogether
10:                     add $item_2$ to newItems
11:                 **end if**
12:             **end for**
13:         **end for**
14:         calculate similarities for $item_1$ and all items from newItems
15:     **end for**
16: **end procedure**

---

matrix. With these columns the similarity between the users is calculated and with the user similarities the recommendations are generated. The whole concept of the singular value decomposition based recommendation is explained in more detail in subsection 2.4.2. The SVDBasedService service is responsible for computing the recommendations after the item based service has decreased the sparsity of the rating table. However this service can be used as a stand-alone recommendation system which is evaluated and compared with the complete recommendation system in section 4.3.

SVDBasedService		
Standard Port	13200	
Get Methods	Parameter: Type	Return Value
/calculateSimilarUsers	-	-
/calculateUserPrediction	user id: Integer item id: Integer	List (user, rating)
Post Methods	Parameter: Type	Return Value
/generateRecommendations	user id: Integer amount: Integer tags: List[String]	List (itemTitle, url, rating)

The /calculateSimilarUsers method executes the calculation of the similar user values. This process is split into four steps. At first the user-item matrix is created out of the rating table. After that the method computes the singular value decomposition of the matrix. For this task the *Apache commons-math* library is used.

This library is a general math library with strong linear algebra algorithms developed and maintained by *The Apache Software Foundation*. The third step is the approximation of the U matrix that is one of the resulting matrices from the singular value decomposition. Finally the similar users are generated by calculating the cosine similarity between all columns of the U matrix. This method does not have a return value. All similar user values are stored in the database for future calculations. The /calculateUserPrediction method calculates a prediction for the specified user-item pair. If no item is specified the method calculates a prediction for all items that are not rated by the specified user. For this process the system generates a list of the most similar users U_{sim}. With these similar users it generates a list of items that these users rated but are unknown to the current user. For each item that is in the unknown item list the system calculates the prediction for the user-item pair as described in subsection 2.4.2. These predictions are returned as a list of user-item tupel. The /generateRecommendations method calculates the item recommendations for the current user. This user is the first parameter that is passed to the function. If the user is unknown because he is not logged in, a value of zero is passed. This zero user is the default user that has ratings for the most popular items. The second parameter specifies the amount of recommendations that the system should return. The third parameter that is passed to the service inside the request body as a post parameter is a JSON string that represents a list of tags. The recommendations that the system generates correlate with these tags. The whole recommendation process is split into four steps. At first the tags are converted into the corresponding descriptor from the *STW Standard Thesaurus Economics*(STW). Afterwards the system generates a list of the most similar users for the current user. For each of these similar users a list is generated that contains the top rated items from this user that are unknown to the current user. Besides, the top rated items match the tags that are specified at the beginning of the request. Finally, for each of these items the rating prediction for the current user is calculated. The items with the highest rating predictions are used as recommendations.

3.3.4 TaggingService

The TaggingService is the implementation of the tagging concept. This concept describes the process of finding the right tags for a text. For this process the system creates a deterministic Levenshtein automata for each word in a text. Every one of these deterministic Levenshtein automata reads the complete (STW). If a word from the STW ends in a final state the word is used as a tag for the text. More details about the tagging concept can be found in section 2.3.

TaggingService		
Standard Port	11000	
Get Methods	Parameter: Type	Return Value
-	-	-
Post Methods	Parameter: Type	
/tagText	text: String	list of tags
/descriptorsForText	text: String	list of STW descriptors
/tagsForDescriptors	descriptors: List[String]	list of tags
/descriptorsForTags	tags: List[String]	list of STW descriptors

The /tagText method takes a text as parameter. The single words of the text are extracted into a list. For each word in this list a deterministic Levenshtein automaton is generated. For this process the service creates a finite automaton for each word with a specific degree. A degree states the maximum allowed Levenshtein distance between the word for which the automaton is generated and the word that the automaton can read. This degree depends on the word size. For a word with more than five letters a degree of three is used. For a word with between three and five letters a degree of 1 is used. Words with three or less letters are only used for direct matches. The states are written as tupel of the form (c, d). Thereby c denotes the current position in the word and d denotes the current Levenshtein distance between the prefix of the original word and the word that the automaton reads. For each letter in a word W with a length of n four transitions are created if the Levenshtein distance of the current state is smaller than n. These four transitions represent reading a correct letter, deleting a letter, inserting a letter and substituting a letter. The last states that are reached through the last letter do not get any transitions by this process. Consequently the next step is to insert the missing insertion transitions for these last states where the current position in the word is the size of the word. These insertions are needed because the word that is read might be longer than the original word and therefore letters must be added to the original word up to the maximum degree. The last step for creating this finite automaton is to mark all of the last states in which the current position in the word is the size of the word as final states. This finite Levenshtein automaton is transformed into a deterministic finite Levenshtein automaton. This transformation process replaces every deletion transition which represents an ϵ transition by one or more new transitions. These new transitions start at the beginning of the ϵ transition and end at a destination of a transition that starts at the end of the ϵ transition. This operation has to be performed due to the fact that ϵ transitions are not allowed in deterministic automata. Furthermore the substitution and insertion transitions are transformed in the way that they only accept labels that are not used by any other transition from the current source. This process yields a deterministic Levenshtein automaton. Finally, all words from the STW are loaded and read into the deterministic Levenshtein automaton. Each word that ends in a final state is added to the list of tags that is returned to the caller. The /descriptorsForText method generates a list of labels for a text. This method is implemented the same way as the /tagText method. The only difference

is that it converts the words which end in a final state to their corresponding STW descriptors before returning them. The /tagsForDescriptors method takes a list of STW descriptors as a parameter and returns the corresponding list of tags. For this process each preferred label is read from the triple store by its descriptors. The triple store contains the STW. The /descriptorsForTags method takes a list of tags as a parameter and returns a list of STW descriptors. This method loads the descriptors from the triple store that corresponds to the tags that were passed as a parameter.

3.3.5 QuerySTWService

The QuerySTWService is the interface between the services and the triple store. It provides methods for sending data to the triple store and processes the data before responding to the requests. This service encapsulates the triple store communication. Hence, it enables an integration into existing systems that use different triple store technologies by replacing only the provided methods of this service.

QuerySTWService		
Standard Port	11500	
Get Methods	Parameter: Type	Return Value
/descriptor	preferred label: String	descriptor
Post Methods	Parameter: Type	Return Value
/runQuery	query: String	-
/descriptorsForTags	tags: List[String]	list of descriptors

The /descriptor method takes a word from the STW as a parameter and returns the descriptor of the preferred label. Another method that the service provides is the /runQuery method. This method takes a query that should be executed on the triple store as a parameter and returns the response from the triple store. The last method that this service provides is the /descriptorsForTags method. This method takes a list of tags as a parameter and returns the corresponding list of descriptors.

3.3.6 RatingService

The RatingService is responsible for calculating and creating the user ratings for the items. This contains the creation process of new users and items. Besides the service starts the asynchronous tagging process if a new item is created and adds the tags to the items if it finishes.

RatingService		
Standard Port	12000	
Get Methods	Parameter: Type	Return Value
-	-	-
Post Methods	Parameter: Type	Return Value
/currentItem	user id: Integer item id: Integer item title: String item text: String item url: String	-
/calculateRating	user id: Integer item id: Integer time spend: Double time scroll: Double user interaction: Boolean	-

The /currentItem method is called every time a user visits an item-page by a JavaScript code that is included in the item-page. This JavaScript code sends the item id, the item title, the item text and the user id to the WebService which forwards these data to the rating service. The /currentItem method creates a new item if the current item does not exist. In addition, it sends the item text to the TaggingService to get the tags for the item if the item does not exist. Moreover, it creates a new user if the current user is unknown to the system. The same JavaScript code is responsible for sending every two seconds a request to the WebService. This request includes the current time that the user has stayed on the page (CT), the time a user has scrolled on the page (ST), the user id and the item id. Then the WebService forwards the data to the /calculateRating method. The design choice of sending the data periodically every two seconds to the server has been made for the reason that sending the complete data set if the user leaves the website is not reliable enough and can lead to the loss of these rating information. After the /calculateRating method receives the CT and ST times it adds these times to the previously received times or creates a new table entry with the times for that user. With these new times the method calculates the new rating for that item and stores it in the database.

3.3.7 WebService

The WebService is the interface for external clients which are usually web browser that visit item pages or the developed recommendation test page. This service forwards external enquiries and processes data before it sends the data back to the client. Furthermore it provides an interface for accessing the database objects for the *backbone.js* application that represents the *recommendation test website* as well as an interface for accessing static files like JavaScript and HTML files.

WebService		
Standard Port	10000	
Get Methods	Parameter: Type	Return Value
/	-	index.html
/tagger	-	tagger.html
/data	type: String id: Integer	model
/file	path: String	document
/calculate	type: String id: Integer	result
/recommendation	user id: Integer amount: Integer tags: List[String]	JavaScript code
Post Methods	Parameter: Type	Return Value
/data	model: Object	-
/rating	*	-

The / and the /tagger methods are both returning specific static HTML files from the file system. These methods should be deactivated for the production mode. The / method returns the index.html file which is the main page for the recommendation test website. Additionally the /tagger method returns the tagger.html website which is a website for testing the TaggingService. The /calculate method forwards the calculation requests that can be specified by a provided type parameter to the RecommendationService. For the provided type parameter the client can use *similarities*, *recommendations* and *recommendationsItemBased*. If a client sends a request for similarities to the WebService, the service starts the calculations for similar users and items on the RecommendationService. A user id is not necessary for this request. By sending the *recommendations* type together with a user id to the /calculate method the service sends a request to the RecommendationService for calculating all predictions for the items that the user has not yet rated. The *recommendationsItemBased* type has the same effect as the *recommendations* type. The only difference is that the predictions are calculated with the item based approach instead of the standard SVD based approach. The /recommendation method is usually called by an HTML website that expects a JavaScript code that is embedded into the page. The method takes a user id, the amount of recommendations and a list of tags as a parameter. These data is forwarded to the recommendation system. However, the returned list of item recommendations is transformed into a HTML link for each recommendation. These HTML links are embedded into a JavaScript code that is returned by the service. This JavaScript code inserts all item links into a specified HTML Div container. There are three /data methods that are interfaces for reading, creating, updating and deleting entries in the database. These interfaces are necessary for the backbone.js framework that is used for building the recommendation test website. The first method that is called via a post request

updates existing entries in the database or creates new entries for the provided parameter. The second method which is called via a get request reads data from the database. The type that is provided as a parameter represents the database table from where the data should be read. If an id is provided as a parameter, the service returns the specific entry. Otherwise the service returns all entries from the database table. The third method is accessible via a http delete method and must provide a type and an id. The specific entry with the corresponding id from the database table that is specified by the type is deleted by the service. The /rating method forwards all requests to the RatingService.

3.4 Service Relations

This section explains the relations between the services. The whole system can be separated into three use cases. The first use case is the situation when a user visits an item page. This situation is illustrated by a sequence diagram in figure 3.2. When a user visits a website that recommends items to users it belongs to the second use case. The sequence diagram figure 3.2 describes this situation. The third use case is the situation in which the system starts the calculation of the user and item similarities. It is presented in the sequence diagram in figure 3.4.

3.4.1 User Visits Item Page

The use case that occurs if a user visits an item page is illustrated in the sequence diagram in figure 3.3. The web browser that executes the JavaScript code that is embedded in each item page sends the current information about the item and the user to the WebService. However, the web browser does not expect a response. The WebService checks if the data is complete and forwards the request to the RatingService. The sequence diagram from figure 3.3 represents the situation when the item and the user are unknown to the system. Therefore the RatingService creates a new item and a new user. Besides, the RatingService sends the text of the item together with its title to the TaggingService. The TaggingService creates a Levenshtein distance automaton for each word in the text and requests all words that are included in the STW from the QuerySTWService. The words from the STW are tested with the Levenshtein automata. Each word that ends in a final state is returned in a list of tags. Finally, the returned tags are added to the newly created item.

Additionally, the web browser sends periodically the time that the user spends on the website together with the time that the user scrolls on the website to the WebService. This data is needed to calculate the implicit user rating for the item. The WebService forwards the request to the RatingService which updates the times in the database and calculates the new rating.

3.4.2 User Visits Recommendation Website

This subsection describes the network communication between the services if a user visits a website that recommends items. The sequence diagram in figure 3.3 represents the network communication between the involved services. The website that has to display the recommendations communicates with the system by including a JavaScript file from the system. The path for this JavaScript file includes optional parameters namely the user id, the amount of recommendations and tags. If the current user is not registered, the system uses a standard user for generating recommendations. The tags are used for filtering the items so that the recommended items match the subject of the website. The web browser requests the JavaScript file from the WebService, which is basically the interface for external clients for the internal services. Thus, the WebService forwards the request to the RecommendationService. This RecommendationService requests the recommendation calculation on the currently activated recommendation system. The latter is the SVDBasedService in the sequence diagram. Another option would be the ItemBasedService. The SVD-BasedService passes the tags to the TaggingService which translates them to the corresponding descriptors by communicating with the triple store via the QueryST-WService. The latter returns the descriptors to the TaggingService which passes the returned descriptor list on to the SVDBasedService. This SVDBasedService generates the item list which includes the items with the highest rating prediction for the current user that match the subject of the tags. This item list is returned to the RecommendationService. The RecommendationService hands the item list to the WebService. The WebService creates html links for each of the items in the list and embeds these links into the requested JavaScript file. The web browser executes the JavaScript file and, thus, inserts the recommended item links into the website.

3.4.3 System Calculates Similarities

The system requests the calculation of new similar user and similar item value from the RecommendationService. The RecommendationService calculates the average user ratings and the average item ratings. Additionally, the service requests the calculation of the user similarities by the SVDBasedService and the calculation of the item similarities by the ItemBasedService. Each of these services stores the calculated values in the database. This should occur periodically every 24 hours to once a week depending on the frequency that the item catalog changes and the frequency that new user ratings are generated.
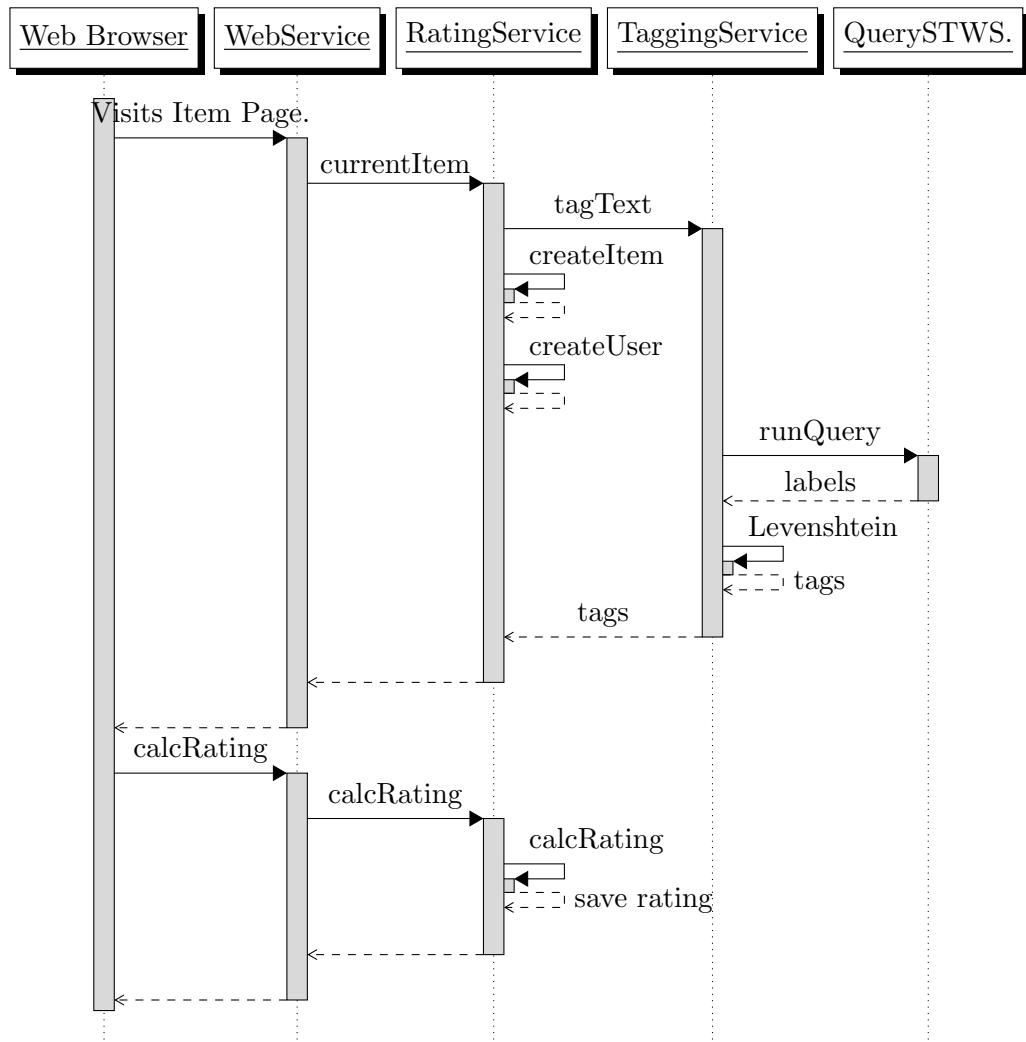
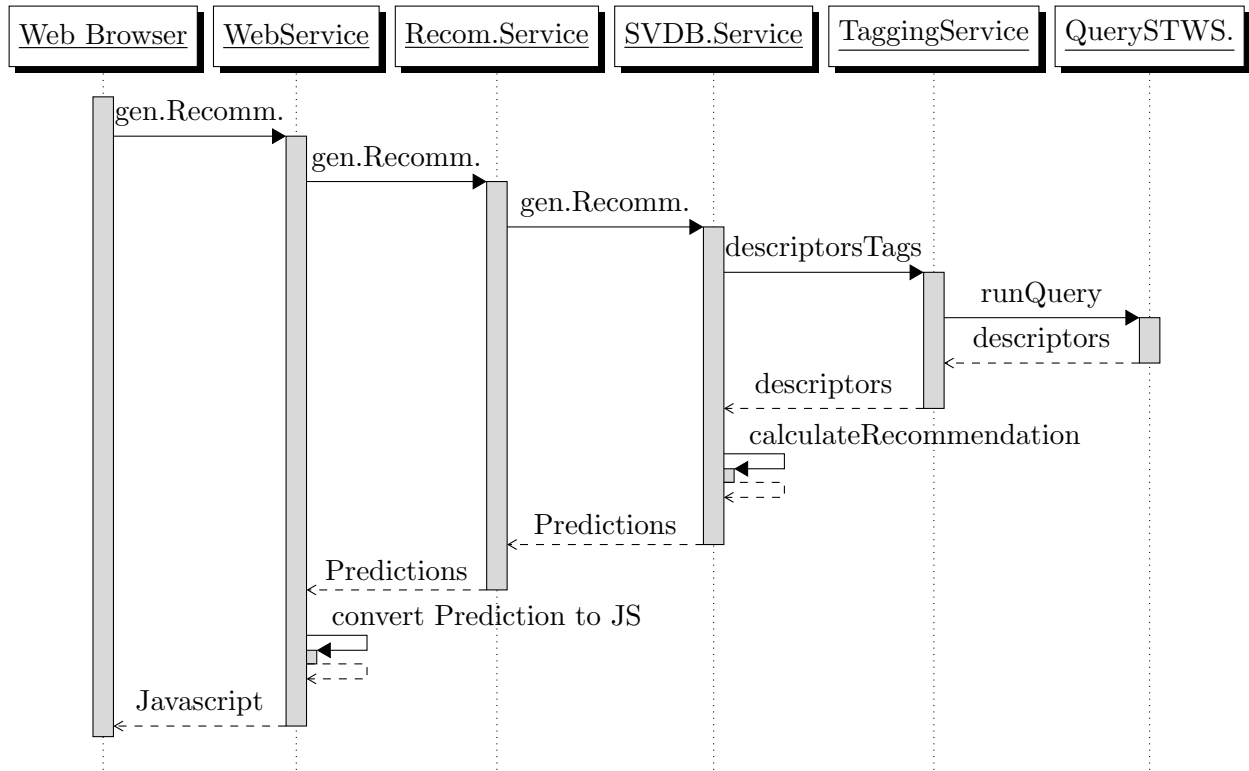Figure 3.2: User Visits Item Page Sequence Diagram

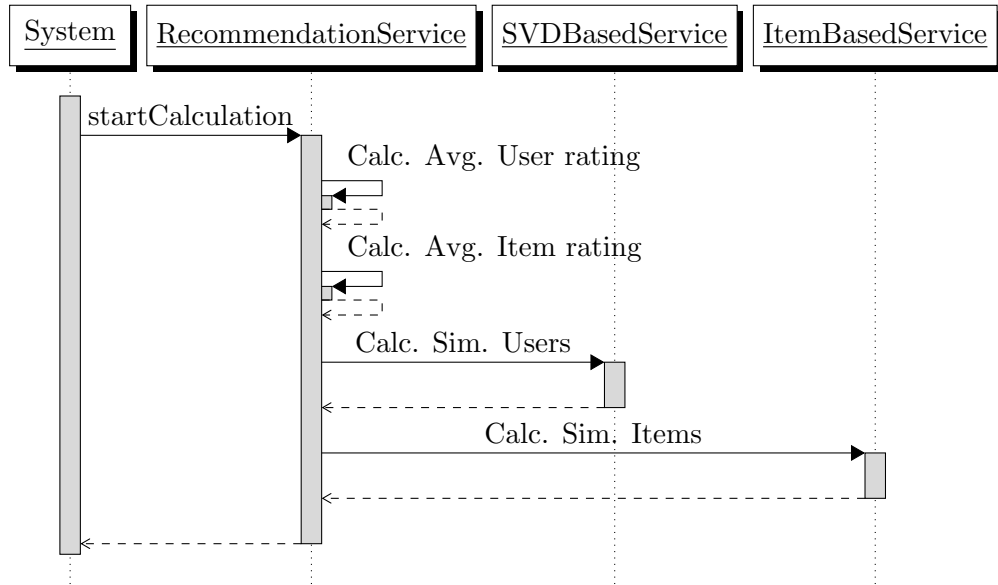Figure 3.3: Embedded Recommendations Sequence Diagram

Figure 3.4: System Calculates Similarities Sequence Diagram

4 Evaluation

In this chapter the different concepts that are introduced in this thesis are evaluated. The first section deals with the rating concept and evaluates the quality of the ratings that are generated for the individual user interaction on a web page. The second section evaluates the time and the quality of the tagging concept by applying the tagging algorithm on texts with different word counts. Moreover the tags that are returned by the algorithm are compared with the tags that should have been returned by the algorithm. Finally different recommendation techniques are compared by the run time and the quality of their predictions.

4.1 Rating

The evaluation for the rating system refers to the rating concept from subsection 2.2.5. This concept is build upon the researches from Claypool et al. [2001] who evaluated different user interactions and compared those with explicit user ratings. The two user interactions that performed the best with an accuracy of approximately 70% to the user ratings are the time a user spends on a website and the time a user scrolls. Each of these measured times of the user interactions are split into segments depending on the truncated mean time of all user times, yielding a scale that enables the system to calculate ratings based on the user interactions. Based on this scale the ratings are calculated separately and are combined to the complete rating by calculating the average of both ratings. This complete rating is evaluated with the research data from Claypool et al. [2001] in this section. For the rating evaluation an item test website is created. This test website is a standard html website with a high of 2000px. Consequently, there is the possibility to scroll on a standard screen resolution. Additionally, the item test website has a truncated mean time for the time a user spends on the website and another truncated mean time for the time a user scrolls on the website. For these truncated mean times the average times from all user ratings of the field test from Claypool et al. [2001] are used. These average times are 5.339 seconds for the time a user scrolls and 20.763 seconds for the time a user spends on a website. Finally, users who visit the website are simulated. Those users are interacting with the website according to the times of Claypool et al. [2001] field experiment. Therefore, the ratings that the simulated users generate can be compared to the data from the team of Claypool et al. [2001].

Rating Algorithm Evaluation				
Scroll Time	Scroll Rating	Time Spend	Time Rating	Complete Rating
3.559	1	14.012	1	1
3.249	1	21.603	3	2
3.355	1	25.497	4	3
5.077	3	13.414	1	2
5.342	3	21.217	3	3
5.287	3	26.797	4	4
7.374	5	12.113	1	3
7.504	5	20.917	3	4
7.431	5	24.797	4	5

The table represents the results of the rating algorithm evaluation. The *Scroll Time* represents the time that a user scrolls on the page and the *Scroll Rating* represents the rating that results out of that scrolling behaviour. Additionally, the *Time Spend* represents the time that a user spends on the web page and the *Time Rating* represents the rating that the system generates for that behaviour. The complete rating is the implicit rating that the system uses as the rating of that user for the specific item. These complete ratings are integer values and are therefore rounded. The values of the Scroll Time nearly match the average times that were found as rating indicators in the field experiment from Claypool et al. [2001]. These average times that a user scrolls on a web page and rates the site with an explicit rating are 3.485 seconds for an explicit rating of 1, 4.079 seconds for an explicit rating of 2, 5.267 seconds for an explicit rating of 3, 6.444 seconds for an explicit rating of 4 and 7427 seconds for an explicit rating of 5. All generated Scroll Ratings reflect the expected ratings from the field test of Claypool et al. [2001]. The same is true for the generated Time Ratings. The Time Spend values nearly match the average times for the specific ratings from the Claypool et al. [2001] field experiment. These average times are 13.414 for a rating of 1, 18.017 for a rating of 2, 21.217 for a rating of 3, 24.372 for a rating of 4 and 26.797 for a rating of 5. However, the rating of five could not be achieved by spending time on the web page because this rating is only attainable by performing a pre defined user interaction such as clicking on a specific button. All other Time Ratings reflect the expected ratings from the field experiment of Claypool et al. [2001]. The *Complete Rating* which is introduced in this thesis correlates with the combined user interest of the Scroll Rating and the Time Rating and therefore represents the expected result. Moreover, the complete rating is derived from the data of Claypool et al. [2001] and it would have been desirable to conduct a field experiment but unfortunately due to the lack of time this was not possible and should be considered for future works.

4.2 Tagging

This section evaluates the tagging process that was introduced in subsection 2.3.2. The concept behind the tagging process is to create a *Levenshtein automaton* for every word in an item and read each word from the *STW Standard Thesaurus Economics*(STW) into the automaton. If the word ends in a final state it is used as a tag for the item. Such a Levenshtein automaton is a deterministic finite automata that can decide in linear time if a word has a Levenshtein distance smaller than a desired $n \in \mathbb{N}$. Four different texts are created for the evaluation process. In each of these four texts 70% of the words are from a placeholder text that is called *Lorem ipsum*. The Lorem ipsum text is according to ipsum Wikipedia, The Free Encyclopedia [2013] typically based on a Latin text from Cicero with words altered, added and removed. Thus it is not a proper Latin text any more. These words are chosen based on the fact that they do not match with the words from the STW. Consequently, they do not create false matches during the evaluation process. The other 30% of the words in the evaluation texts are from the STW. However, not all of the words are in the same form as they are found in the STW. Therefore, the tagging algorithm has to match the words from the item with the correct words from the STW. After each test the matched words are compared to the words that should have matched. Besides, the time that the algorithm needed for creating all Levenshtein automata and the total time in seconds is evaluated. The computer that is used for testing this tagging algorithm is a *Macbook Air* with a 1.86 *GHz Intel Core 2 Duo* precessor and 4 GB DDR3 memory.

Tagging Algorithm Evaluation					
Words	STW	Non Matching	Matched Words	Levenshtein Time	Total Time
1	1	0	2	0.099	14.481
10	3	7	3	0.2843	20.348
100	30	70	54	0.4751	79.9616
1000	300	700	470	2.5606	777.4643

At first the word *tests* is used with the tagging algorithm. This evaluation results two matches from the STW namely *test* once for the English and once for the German language. The time that is needed for creating the Levenshtein automaton is approximately a tenth of a second. For this complicated generation process this is a good performance. However the total computation time is 14.481 seconds which involves the loading of the 31.682 words from the STW and the reading of all of these words into the Levenshtein automaton.

For the second evaluation ten words are used with seven non matching words and three words from the STW. Two of the three words are modified. Hence, they are not in the same form as they appear in the STW. All three of these words are matched by the tagging algorithm. The creation time of all Levenshtein automata is 0.2843 seconds which is approximately three times as high as the Levenshtein automaton

computation time for the one word creation. The total computation time of 20.348 seconds is 1.5 times higher than the total computation time for the one word tagging process.

The 100 words evaluation consist of 30 words from the STW with spelling mistakes and different forms. The other 70 words are non matching words from the Lorem ipsum placeholder text. The algorithm returns 54 words as matches from the STW. These 54 words include all 30 words that were included in the original 100 words plus 24 words that are similar spelled or use the same words in English and German. The computation time for all 100 Levenshtein automata is 0.4751 and in total the computation needs 79.9616 seconds.

The last tagging algorithm evaluation uses a text with 1000 words. This text contains 300 words from the STW with approximately 150 words in different forms or with spelling mistakes. The other 700 words are words from the Lorem ipsum placeholder text and, thus, are non matching. The algorithm returns 470 words whereby almost all of the original words were included. The words that did not match are the words that had more than three changes resulting from changing the singular form to its plural form and adding spelling mistakes. The computation time for the Levenshtein automata is five times higher than the corresponding time for the 100 word test. Anyhow, the total computation time is nearly ten times higher than the total time for the 100 word test.

A conclusion of this evaluation is that the largest part of the total time is the reading of the words from the tagging base into the Levenshtein automata and not the time that the algorithm needs to create these automata. Consequently, the main focus in future works should not be to improve the performance of the creation process of the Levenshtein automata but to decrease the number of words in the tagging base.

4.3 Recommendation

This section evaluates the different recommendation approaches that are explained in subsection 2.4.4. These different approaches are the item-based recommendation, the different singular valued decomposition (SVD) based recommendations, and the approach that is described in this thesis that uses item-based predictions to decrease the sparsity of the rating table and the SVD-based approach for recommendations. The evaluation process of the recommendation system contains two parts. Firstly, the evaluation of the performance of the different approaches by using the wall clock time. Secondly, the comparison of the accuracy of the different concepts. For this task the *MovieLens* database is used. The MovieLens database includes 100.000 ratings from 943 users for 1682 movies. This data is collected with the movielens.org website that is part of the *GroupLens* research project from the University of Minnesota. The website is a recommendation website for movies. Each user of the website can rate movies on a scale from 1 to 5 and is afterwards presented with movies that might be interesting for him/her. To compare the accuracy of the recommendation systems the *Mean Absolute Error* (MAE) is used.

Definition 24 (Mean Absolute Error). *Let $p_{i,j}$ be the predicted rating for user i and item j and let $q_{i,j}$ be the real rating for user i and item j. Then the following formula calculates the MAE.*

$$MAE = \frac{\sum_{u \in U} \sum_{i \in testset_u} |rec(u,i) - r_{u,i}|}{\sum_{u \in U} |testset_u|}$$

This metric is according to Sarwar et al. [2001] most commonly used for evaluating recommendation systems. The MAE is a statistical metric that compares the calculated user-item ratings from the recommendation system with real user-item ratings. For this process the recommendation system is trained with a percentage p of a test rating database and afterwards the remaining 1-p percent are used to check if the recommendations are correct. A ratio of 80% training data and 20% prediction data is used.

The team of Herlocker et al. [2004] noticed that many collaborative recommendation systems in the movie domain reach a mean absolute error of up to 0.73 on a rating scale from 1 to 5. Out of this realisation Herlocker et al. [2004] created the hypothesis that collaborative recommendation systems which are tuned to its optimum cannot get much more accurate than a mean absolute error of 0.73. They explain this hypothesis by reasoning that users tend to provide inconsistent ratings when they are asked to rate the same movie at different times. Hence, an algorithm cannot be more accurate than the users rating variance for the same item.

Recommendation Comparison 80.000 ratings 20.000 predictions			
Technology	MAE	Offline Computation Time	Prediction Time
Item Based	0.83187	3187.69	4208.49
SVD Concept	0.77937	32471.93	4315.31
SVD Three	0.79545	469.82	4395.75
SVD Average	0.78622	507.90	4226.31
Three	1.03333	0.00	0.49
Average	0.82714	112.13	249.18

The recommendation comparison table represents the evaluation of the recommendation systems. The computer that is used for testing the different recommendation techniques is a *Macbook Air* with a 1.86 *GHz Intel Core 2 Duo* precessor and 4 GB DDR3 memory. Consequently, all computation times that are measured in this evaluation are based on this hardware system and can be faster on a productive server. However, the computation times can be used to compare the speed of the different recommendation systems. The evaluation contains three different metrics the mean absolute error, the offline computation time and the prediction time. The mean absolute error is, as explained before, a statistical metric that declares the average distance between the predicted user-item rating and the real user-item rating. The offline computation time represents the time that the system needs to generate the data that is needed for the prediction of the items. Hence, this computation must

not be calculated for every recommendation but every 24 hours to once a week depending on the frequency that new items are added to the system and the activeness of the users in order to ensure the correctness of the user and item similarities. The prediction time is the time that the system needs to calculate the predictions for all 20.000 ratings. Therefore, this time divided by 20.000 represents approximately the time that the system needs to calculate one prediction for the recommendations that are presented to the user on a website.

The *Item Based* technology is the implementation of the item based recommendation approach that is part of the overall recommendation concept of this thesis. It has a mean absolute error of 0.83187 which is the second poorest mean absolute error of all tested technologies. This is a surprising outcome of the evaluation because it was expected that this technology would be more similar to the mean absolute error of the SVD based technologies. The offline computation time in this implementation consists of finding items that were rated by the same users and calculating the adjusted cosine similarity between those items. The adjusted cosine similarity is a special form of the cosine similarity. The cosine similarity for two items is calculated by applying the dot product between the rating vectors of the items and dividing the result by the Euclidean length of the rating vectors. The adjusted cosine similarity uses the same computation but normalizes the ratings in the rating vectors before calculating the cosine similarity. This normalization consists of subtracting the average user rating from the rating to remove the individual rating behaviour of the user. This process is explained in more detail in subsection 2.4.2. The calculation of the adjusted cosine similarity between all items takes the largest part of the computation time. At last the item similarities are stored in the database.

The prediction time for a user-item pair involves the finding of similar items that the user has rated and calculating the sum of the product of the similarity and the rating and dividing the result by the sum of the similarities. So, all in all the implemented Item Based technique has the second slowest offline calculation time and with an average distance of 0.83 between the predicted rating and the actual rating one of the poorest mean absolute errors of the implemented methods.

The *SVD Concept* technology represents the recommendation system that is introduced in this thesis. Therefore, it uses item based rating predictions to fill empty rating entries in the user-item rating matrix with predictions. The mean absolute error of this technology is the best of all implementations that were tested. In terms of the other SVD based implementations this can be explained by the fact that the rating matrix has the most accurate entries of all. However, the SVD Concept has a very high offline computation time. The offline computation time of this system contains the following calculations. At first the average user rating is calculated which is needed to remove the individual rating behaviour from the user ratings before calculating the predictions. Thereafter the item based cosine similarity is calculated which contains the same parts that where explained for the offline computation time of the Item Based technology. After that for each user the rating prediction of the missing user-item entries in the rating table is calculated which is responsible for the largest part of the total computation time. The computation time for each user is

approximately 30 seconds and the test database consists of 943 users. This can be explained by the fact that each user has only rated a fraction of all items and therefore many predictions must be calculated. Afterwards the rating matrix is generated from the rating table. The entries of the matrix represent a rating for a user-item pair. The previously generated rating predictions are used for empty entries in the rating matrix. For this rating matrix a singular value decomposition is generated. This process decomposes the rating matrix into three matrices namely U, V and Σ. The matrix U has the property that each column of the matrix represents the interest of a user. More details to the singular value decomposition and the features of the three matrices are described in subsection 2.4.2. In addition, the matrix U is reduced to the two dimensional approximation to decrease the following computation time. This computation time is the calculation of the cosine similarity between the columns of the two dimensional approximation of the matrix U. These similarities are used to created the final rating predictions for the item recommendations when a user visits a website. Finally, the user similarities are stored in the database.

The prediction time consists of finding similar users that rated a specific item and calculating the prediction with the similarity of the users and their ratings. Overall this technique has the best mean absolute error but it has by far the poorest offline computation time.

The *SVD Three* technology represents a singular value based recommendation system that uses a rating of three to fill in the missing user-item ratings in the rating matrix. The rating of three is used because it represents the middle of the rating scale of 1 to 5 and therefore it should have the least distance between all ratings if the ratings are uniformly distributed. The mean absolute error of this implementation is 0.79545 which is the poorest of the SVD based approaches. This is expected because it generates the singular value decomposition with a rating matrix that uses static values for the missing entries. This matrix should have the highest difference to the matrix that would represent the real user interest and hence should generate the poorest user similarities. The offline computation time of this system consist mostly of the same parts as the SVD Concept. These parts are calculating the average user rating for each user, creating the rating matrix from the database and calculating the singular value decomposition as well as creating the approximation of the matrix U that is part of the singular value decomposition. Besides the offlise calculation includes the calculation of the cosine similarity between the columns of the matrix U approximation and the storing of these user similarities in the database. The calculations that are created during the prediction time are the same as the calculations that were used in the SVD Concept approach namely the finding of similar users that rated a specific item and the calculation of the predictions. This system is evaluated in order to compare the impact of the values that are used to fill in the missing ratings in the rating matrix. This evaluation demonstrates that the more accurate these values are the better is the quality of the predictions. However, the difference of the mean absolute error of this approach and the other two SVD based approaches is smaller than expected. Therefore, further developments that should improve the prediction accuracy should be concentrated on the prediction

calculation and not on the improvement of the similarity calculations. Altogether this technique has the fastest offline calculation time of all tested more advanced recommendation systems and a good mean absolute error.

The *SVD Average* technology is the implementation of the SVD based approach but it uses the average item ratings to fill in the missing entries of the rating matrix. Therefore, it is a solution without complex calculations to create dynamic rating predictions for the missing entries in the rating matrix that mirror the overall user interest. The mean absolute error is 0.78622 and, thus, it has the second best mean absolute error of all tested systems. The offline calculation time consists of calculating the average user ratings and calculating the average item rating as well as all other calculations that are needed in the SVD Three approach. The singular value decomposition works in the same manner as the one in the SVD Three approach with one difference it uses the average ratings of the items to fill the missing entries in the rating matrix. The prediction time of 4226.31 seconds includes the same calculations that were used for the other two SVD based approaches which are the finding of similar users that rated a specific item and the calculation of the predictions. Hence, it has almost the same prediction time as the other SVD based approaches that were tested in this evaluation. To sum this up this technique has the second best mean absolute error and moreover it has one of the fastest offline computation times of the tested more advance recommendation systems. Thus, this system has a fast computation time and a very good mean absolute error and for this reasons it should be used for further developments if the quality of the recommendations is more important than the computation time.

The last two systems are added to the evaluation to compare the results of the more advanced recommendation systems with less advanced approaches. The *Three* technique uses a rating of three for all predictions. The three was chosen because it represents the middle of the five point rating scale. Therefore, it has the smallest mean absolute error of 1.2 of all static ratings if the user ratings are evenly distributed. This technique does not require any offline computations and the mean absolute error calculation consists of subtracting the real rating from three. This approach yields a MAE of 1.03333. Consequently, the ratings are less common on the outer boundaries of the rating scale which is a common user rating behaviour. The MAE of 1.03333 can be seen as a lower boundary. If any of the recommendation techniques is below this MAE, it is an indicator that the recommendation technique does not work properly.

The *Average* technique uses the average item rating as a rating prediction. Thus, it is the least complex method which generates dynamical rating predictions that represent the interest of all users. The difference from its MAE of 0.82714 to the lower boundary of 1.03333 is 0.20619 which is a good result. Additionally, it has nearly the same mean absolute error as the item based approach. Hence, it can be concluded that the interests of the individual users from the *MovieLens* database does not vary widely. This technique only has to calculate the average user ratings of all users for the offline computation time which is very fast. Besides it only needs to load one value from the database for the prediction calculation instead of calculating

a prediction. As a result the Average technique is a good choice if the user interest of the system does not vary widely and if the system needs a fast solution.

In conclusion the evaluation of the recommendation techniques revealed that the SVD based approach that was introduced in this thesis has the best mean absolute error of the implemented approaches. Anyway, it does not vary widely from the MAE of the SVD Average technique whereby the offline computation time of the SVD Concept approach is by far the slowest. The item based approach has a slow computation time and the MAE is nearly the same as the Average approach. The research team of Sarwar et al. [2001] showed that this method can yield a MAE of up to 0.73 by improving the prediction calculation. However these improvements should have the same effect for the SVD based approaches and therefore the SVD based techniques should be the better choice for collaborative recommendation systems. The evaluation of the SVD Three technique indicates that SVD based recommendation systems generally result in good predictions. Moreover, it showed that improvements for the missing entries for the rating matrix yield improvements of the predictions but that these optimizations only improve the MAE by approximately 0.2 points. Therefore, future works with the goal of optimizing collaborative recommendations should concentrate on the improvement of the prediction calculations. Finally, the evaluation revealed that using the average user rating as a rating prediction can yield good and fast predictions if the user interest does not vary widely. Consequently, this can be used for calculating rating predictions as part of recommendation systems if the use case of the system does not require highly optimized recommendations. This evaluation should be repeated when the system is in productive usage and has collected the same amount of data that was used for this evaluation to check if the data for the specific use case of recommending questions and answers reveal the same findings as this evaluation.

5 Conclusion

The focus of this thesis is the development of a recommendation system for user generated content. These recommendations should match the user interest and the context of the website on which they are included. For this it is important to analyze the user generated content quality and to be able to identify the subject of the content automatically. Hence, user interactions that can occur on a website are evaluated to be able to tell if a user might be interested in the content. Based on the research of Claypool et al. [2001] the time a user spends on a website and the time a user scrolls on a website are chosen for generating implicit user ratings. These two user interactions correspond about 70% of the time with the explicit ratings of a user. The times of each of these interactions are used separately for creating a scale that expresses user ratings by these times. The average of these two ratings is then used as the complete rating. This concept is evaluated with simulated user interactions. The result of this simulation displays that the two separate ratings agree with the findings of Claypool et al. [2001]. Besides the complete ratings are as expected. However for this part of the thesis a field test in the future would be a good idea to adjust the formula according to the use case. For the categorization of the user generated content the words of the content are matched with words from a thesaurus. For this task different error allowing string matching methods are evaluated. The chosen concept is the *Levenshtein automata* which is a special automata for approximate string matching. Based on the work of Schulz and Mihov [2002] the algorithm for a deterministic Levenshtein automata is integrated into the overall concept. Besides, a concept for finding the maximum *Levenshtein distance* for a word depending on its length is introduced in this thesis. This is important because a word with a small word size can not be handled the same way as a word with a long size. The smaller word need a smaller distance otherwise it would generate too many false matches. This word match algorithm is used to find the keywords of a text and to use them as tags. This enables the system to presort the content by the keywords before recommending it on a website. The tagging process is evaluated by using texts with a different amount of words. Each of those texts has 30% words that should match with the words in the thesaurus. However most of the words are in a different form or spelling mistakes are inserted. So, the evaluation process checks how many words are found and how long does the algorithm need. In three of four tests more words were returned than the algorithm should have matched. This is mostly due to the reason that most words in the thesaurus are included in the German and English language and that some of those words are quite similar. Consequently the algorithm returns the English and the German word. Other false matches occurred because the thesaurus included words that were too similar and

thus the algorithm returned both of these words. Therefore a future optimization is to reduce the size of the tagging base which would not just reduce the occurrence of false matches but would also decrease the time that the algorithm needs to find the matches. Another possible future work is to decide whether the text is in the German or English language and to use only the words from the corresponding language as a tagging base. In addition, different types of recommender systems are discussed. The conclusion of this discussion is that for the use case of recommending user generated content a collaborative recommendation system would be the best approach. Two existing collaborative recommendation techniques are explained namely the item based technique and a recommendation technique based on singular value decomposition. These techniques are combined into a new recommendation concept which is introduced in this thesis. This concept uses the item based approach for decreasing the scarcity of the rating matrix which is used for the calculation of the singular value decomposition. This recommendation system uses the ratings that are generated with the user interactions on the website and the tags that are generated with the help of the Levenshtein automata to recommend individual content to users that is interesting for the current user and matches the current subject of the website. The evaluation of the recommendation system showed that the new recommendation concept creates better prediction than the two techniques on their own. Though, the calculation time increased drastically and the prediction was only slightly better than a prediction that uses average ratings for decreasing the sparsity of the rating matrix for the singular value decomposition. Consequently future works with the goal of improving the recommendation system should use the singular value decomposition that uses average ratings and concentrate on the prediction calculation for improvements. Moreover the evaluation showed that the method of using only average ratings for prediction calculation can create good rating predictions. Thus, systems that do not have a need for using highly optimized recommendations could use this method for creating recommendations.

5.1 Future Work

The implemented system is fully functional and can be used for the purpose it is intended to be. However each of the single parts might have the possibility of improvements. The following subsections are suggestions on how the system might be enhanced in the future.

5.1.1 Rating System

The evaluation of the rating system denotes that the implicit ratings should approximately mirror the real user interest. However due to the shortness of time it was not possible to conduct a field experiment to test the assumptions. Such a field experiment would be desirable and might lead to further improvements.

5.1.2 Tagging Service

The evaluation of the tagging service suggests that the process can be improved by decreasing the tagging base. This might be done by automatically detecting the language of the content and only using the words of that language for generating tags.

5.1.3 Recommendation System

The evaluation of the recommendation techniques yield that the best recommendation approach of the implemented recommendation systems for a productive system is the singular value decomposition with the use of average ratings for the empty entries in the rating matrix. Furthermore the evaluation suggests to concentrate future improvements on the calculation of the rating predictions instead of the calculation of the user similarities. Thus, it might improve the prediction calculation by weighting the ratings for the calculation based on the difference to the average rating. This should on the one hand reduce the importance of ratings on which most user agree upon and on the other hand should increase the importance of the items that are more diverse. Consequently the predictions should be more individual for the single user.

Bibliography

G. Allaire and S.M. Kaber. *Numerical Linear Algebra.* Texts in applied mathematics. Springer-Verlag New York, 2008. ISBN 9780387689180. URL `http://books.google.de/books?id=PlUWqQzY-4EC`.

Ricardo Baeza-Yates. A unified view to string matching algorithms. In *IN PROC. THEORY AND PRACTICE OF INFORMATICS (SOFSEM'96), LNCS 1175,* pages 1–15. Springer Verlag, 1996.

Fidel Cacheda, Víctor Carneiro, Diego Fernández, and Vreixo Formoso. Comparison of collaborative filtering algorithms: Limitations of current techniques and proposals for scalable, high-performance recommender systems. *ACM Trans. Web,* 5(1):2:1–2:33, February 2011. ISSN 1559-1131. doi: 10.1145/1921591.1921593. URL `http://doi.acm.org/10.1145/1921591.1921593`.

Mark Claypool, Phong Le, Makoto Waseda, and David Brown. Implicit interest indicators. In *ACM Intelligent User Interfaces Conference (IUI),* 2001.

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *JOURNAL OF THE AMERICAN SOCIETY FOR INFORMATION SCIENCE.,* 1990.

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

John Hopcroft, Rajeev Motwani, and Jeffrey Ullman. *Introduction to automata theory, languages, and computation.* Pearson/Addison-Wesley, 2003.

Robert Hundt. Loop recognition in c++/java/go/scala. In *Proceedings of Scala Days 2011,* 2011. URL `https://days2011.scala-lang.org/sites/days2011/files/ws3-1-Hundt.pdf`.

Lorem ipsum Wikipedia, The Free Encyclopedia. Lorem ipsum, 2013. URL `http://en.wikipedia.org/wiki/Lorem_ipsum`. [Online; accessed 03-June-2013 at 15:34].

Dietmar Jannach, Markus Zanker, Alexander Felfernig, and Gerhard Friedrich. *Recommender Systems An Introduction.* Cambridge University Press, 2011.

Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings*

of the 28th annual international ACM SIGIR conference on Research and development in information retrieval, SIGIR '05, pages 154–161, New York, NY, USA, 2005. ACM. ISBN 1-59593-034-5. doi: 10.1145/1076034.1076063. URL `http://doi.acm.org/10.1145/1076034.1076063`.

Caumanns Jörg. A fast and simple stemming algorithm for german words, 1999.

Alexandros Karatzoglou, Alex Smola, and Markus Weimer. Collaborative filtering on a budget. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9, pages 389–396, 2010. URL `http://cs.markusweimer.com/2010/05/12/collaborative-filtering-on-a-budget/`.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, August 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.263. URL `http://dx.doi.org/10.1109/MC.2009.263`.

Bernd Kortmann. *English linguistics - essentials.* Studium kompakt : Anglistik - Amerikanistik ; System use. Cornelsen, Berlin, english ed., 1. aufl., 5. druck edition, 2011. ISBN 978-3-464-31162-2. Nebensacht.: Linguistics - essentials.

Greg Linden, Brent Smith, and Jeremy York. Amazon.com recommendations item-to-item collaborative filtering. *IEEE INTERNET COMPUTING*, 2003.

Jiahui Liu, Peter Dolan, and Elin Rønby Pedersen. Personalized news recommendation based on click behavior. In *Proceedings of the 15th international conference on Intelligent user interfaces*, IUI '10, pages 31–40, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-515-4. doi: 10.1145/1719970.1719976. URL `http://doi.acm.org/10.1145/1719970.1719976`.

Stoyan Mihov and Klaus U. Schulz. Fast approximate search in large dictionaries, 2004.

Vladimir Mironov, Nirmala Seethappan, Ward Blondé, Erick Antezana, Bjorn Lindi, and Martin Kuiper. Benchmarking triple stores with biological data, 2010.

Gonzalo Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33(1):31–88, March 2001. ISSN 0360-0300. doi: 10.1145/375360.375365. URL `http://doi.acm.org/10.1145/375360.375365`.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms, 2001.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Incremental singular value decomposition algorithms for highly scalable recommender systems. In *Fifth International Conference on Computer and Information Science*, pages 27–28, 2002.

Klaus Schulz and Stoyan Mihov. Fast string correction with levenshtein-automata. *INTERNATIONAL JOURNAL OF DOCUMENT ANALYSIS AND RECOGNITION*, 5:67–85, 2002.

G. W. Stewart. On the early history of the singular value decomposition. *SIAM Review*, 1993.

Gilbert W. Stewart. *Matrix algorithms: Volume 1, Basic decompositions*. Society for Industrial and Applied Mathematics, 1998.

E. Thießen. *Die Pluralbildung Im Deutschen: Eine Untersuchung an Hand Der Optimalitätstheorie*. GRIN Verlag, 2009. ISBN 9783640439096. URL `http://books.google.de/books?id=S4FjEHuR2mkC`.

Levenshtein Distance Wikipedia, The Free Encyclopedia. Levenshtein distance, 2013a. URL `http://en.wikipedia.org/wiki/Levenshtein_distance`. [Online; accessed 11-June-2013 at 12:43].

Scala Programming Language Wikipedia, The Free Encyclopedia. Scala programming language distance, 2013b. URL `http://en.wikipedia.org/wiki/Scala_(programming_language)#Companies`. [Online; accessed 11-June-2013 at 17:53].

Truncated Mean Wikipedia, The Free Encyclopedia. Truncated mean, 2013c. URL `http://en.wikipedia.org/wiki/Truncated_mean`. [Online; accessed 10-June-2013 at 11:24].