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:	** Name des betreuenden Hochschullehrers **
Betreuer:	** Name des Betreuers **

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

1 Introduction **1**

1.1 Overview . . . 1

1.2 Related Work . . . 1

1.3 Contributions . . . 1

2 Concepts **3**

2.1 Question and Answer Site . . . 3

2.2 Rating System . . . 4

2.2.1 The time a user spends on a website . . . 4

2.2.2 The time the cursor is in motion . . . 5

2.2.3 The number of mouse clicks . . . 5

2.2.4 The time a user scrolls . . . 5

2.2.5 Total Rating Concept . . . 6

2.3 Tagging . . . 9

2.3.1 Levenshtein Distance . . . 9

2.3.2 Optimized Levenshtein distance algorithm . . . 10

2.4 Recommendation . . . 16

2.4.1 Different types of recommendation systems . . . 16

2.4.2 Collaborative recommendation system . . . 18

2.4.3 The recommendation process . . . 22

2.4.4 Different Recommendation approaches . . . 23

2.5 Concept Relationships . . . 24

3 Implementation **25**

3.1 Technologies . . . 25

3.1.1 Programming Language . . . 25

3.1.2 Databases . . . 26

3.1.3 Network Communication . . . 27

3.1.4 Testing . . . 28

3.2 Architecture . . . 28

3.3 Services . . . 29

3.3.1 RecommendationService . . . 29

3.3.2 ItemBasedService . . . 30

3.3.3 SVDBasedService . . . 31

3.3.4 TaggerService . . . 32

3.3.5 RatingService . . . 33

3.3.6 WebService 33

4 Evaluation 37

4.1 Rating 37

4.2 Tagging 39

4.3 Recommendation 40

5 Future Work 47

5.1 Rating System 47

5.2 Tagging Service 47

5.3 Recommendation System 47

List of Figures

2.1 Time spent scrolling Claypool et al. [2001] 6
2.2 Time spent on a page Claypool et al. [2001] 6
2.3 A non deterministic levenshtein automata for the word *test* with degree 2 12
2.4 A deterministic levenshtein automata for the word *test* with degree 1 14
2.5 User Item Relationship Table 20
2.6 Singular Value Decomposition of the User Item Relationship Table . 23
2.7 Similar Users based on Singular Value Decomposition 24

3.1 The Graphical User Interface for The Recommendation Process . . . 34
3.2 The Service Architecture of the system 35

List of Tables

List of Algorithms

1 Recursive Levenshtein Distance Algorithm 10
2 Levenshtein Automata is in distance 15

3 Item Similarity Calculation 31

Abkürzungen

Abstract

1 Introduction

1.1 Overview

What is this thesis all about. Why should anyone add a recommendation system for questions and answers to a website. What components are included in the recommendation system.

1.2 Related Work

Current state of the art - recommendation systems. - rating systems. - tagging systems.

1.3 Contributions

What is the new concept?

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 connect the questions and answers with the recommendation system. 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 relationship of 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. Therefore a user can ask a question. 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 subject. This recommendation process should increase the user interaction with 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 is in use for the implemented system is askbot.com. It is build with the *django web framework* and actively maintained since 2011.
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* measured the time a user spends on a website as a first indicator that the user works with the 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 is 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. Additionally the timer is only active as long as the website is visible in the web browser. They compared the time a user spends on a website with the explicit rating from the user this correlated about 70% of the time. 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. Therefore other indicators where measured and included in *Claypool's* 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. This 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 timing the cursor motion but it is only 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 is not significantly different as though the user rated it with the highest or lowest rating. So, this method is not a good indicator for the users' interest.

2.2.4 The time a user scrolls

The last indicator they used was the time a user scrolls. This is an important activity indicator because a user 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* 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. With that data they found out that both the measurement of user scrolls by keys and by mouse are on their own poor indicators of interest 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 was used in this thesis as an activity indicator.

2.2.5 Total 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 Claypool et al. [2001]. These methods have an accuracy of about 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, the user stays on the website longer and the user has to scroll in order to see the whole content of the website.

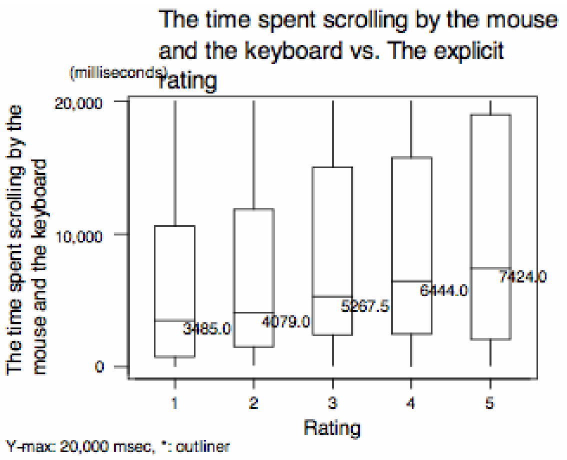


Figure 2.1: Time spent scrolling Claypool et al. [2001]

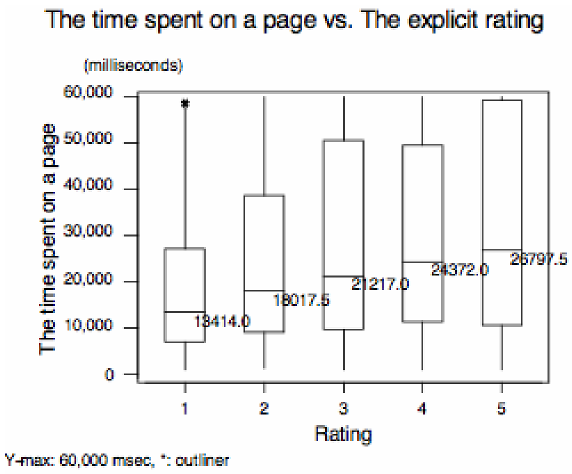


Figure 2.2: Time spent on a page Claypool et al. [2001]

In order to collect implicit user ratings by timing how long the website is in focus, the truncated mean time all users spend on the item page is used as a benchmark.

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.*

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

The percentages of the two rating processes result out of the two diagrams of figure 2.1 and figure 2.2.The following paragraph explains the rating of the item page in relation to the truncated mean time.

Add diagrams, maybe write the itemization as a text

- If the user spends less than 10% of the truncated mean time on the item page it is used as a rating of 1.
- If the user spends less than the truncated mean time minus 20% on the webpage it is used as a rating of 2.
- If the user spends plus minus 20% of the truncated mean time on the item page, it is used as a rating of 3.
- If the user spends longer than the truncated mean time plus 20% on the webpage 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. 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 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 40% of the truncated mean time on the item page it is used as a rating of 1.
- If the user scrolls between 40% and 80% of the truncated mean time on the webpage it is used as a rating of 2.
- If the user spends plus minus 20% of the truncated mean time on the item page, it is used as a rating of 3.
- If the user spends between the truncated mean time plus 20% and 60% on the webpage it is used as a rating of 4.

- If the user scrolls longer than the truncated mean time plus 60% 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. First of all 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 total rating.

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

If a user returns to an item site the new rating is added onto the previous rating with a maximum sum of 5. This rating calculation results from the observation that users tend to revisit websites they were interested in in the past and which might help them with their current research. It was considered to generally rate all websites that were revisited with the highest rating of 5 but this concept was abandoned because a user might be revisiting a formally uninteresting website unintentionally. As mentioned above Claypool et al. [2001] research group bases its results on a field experiment of over 80 people browsing over 2500 Web pages. The total rating concept is strongly build upon these results. It would have been desirable to conduct a field experiment for the concept of the total 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 *ZBW* provides the *STW Thesaurus for Economics*. The *STW Thesaurus for Economics* 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 Thesaurus for Economics* 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 Thesaurus for Economics*. So, the challenge for this tagging process is to find the correct words in the *STW Thesaurus for Economics* even though the words from the item might not correspond directly to the words in the *STW Thesaurus for Economics*. One possible solution for such a task is to reduce each word from the text to its stem.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 *Caumanns A Fast and Simple Stemming Algorithm for German Words*. For example a predefined rule might be '3+ies' → 'y'. So, the word *libraries* would be reduced to *library*. Thus it is important that the stemming algorithm has all necessary rules for each supported language.

definition stem

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 match in its original form. Due to 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 differences between the words from the text and the words from the basis for the tagging process. This difference 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 practical metric for such a task is the *levenshtein distance* which will be explained in the following section.

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. 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

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

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

```
procedure LevenshteinDistance($s : String, t : String$)
    $lenS \leftarrow length(s)$
    if lenS = 0 then
        return $lenT$
    end if
    if lenT = 0 then
        return $lenS$
    end if
    if s[lenS-1] = t[lenT-1] then        ▷ test if last characters of the strings match
        $cost \leftarrow 0$
    else
        $cost \leftarrow 1$
    end if
        ▷ The first recursive call represents a deletion, the second represents an
    insertion and the third represents a substitution or a correct letter
    return minimum of
      $LevenshteinDistance(s[0..lenS-1], t) + 1,$
      $LevenshteinDistance(s, t[0..lenT-1) + 1,$
      $LevenshteinDistance(s[0..lenS-1], t[0..lenT-1]) + cost)$
end procedure
```

The direct implementation of the *levenshtein distance algorithm* has a complexity of O(mn) with m size of the first word and n 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 Hopcroft et al. [2003].

Definition 2 (Non Deterministic Finite Automata)**.** *A non deterministic finite automata 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 exactly when* 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* $\delta : Q \times \Sigma \rightarrow Q$.

Definition 4 (Path). *A path for* $\mathcal{A}$ *is a series* $\pi = p_0 a_1 p_1 a_2 \ldots a_n p_n$, $(p_i, a_{i+1}, p_{i+1}) \in \Delta$ *and* $0 \leq i \leq n-1$. *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). *The function* $\beta(\pi)$ *maps a path* π *to a label* ω. $\mathcal{A} : p \overset{\omega}{\rightarrow} q$ *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* $\mathcal{A}$: $p \overset{\omega}{\rightarrow} q$. *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* 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 automata* is described by Baeza-Yates [1996]. The purpose of the *levenshtein automata* is to decide whether $d_L(W, V)$ is smaller than a specific $n \in \mathbb{N}$ with $V \in \Sigma^*$. So, it is possible to decide if a word from a question is similar to a word from the *STW Standard Thesaurus*, 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 automata* 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 $\sigma = \sigma_{correct}$
- $(i^j, \sigma, i^{(j+1)})$ if σ is inserted after state i^j and $(j+1) < n$
- $(i^j, \sigma, (i+1)^{(j+1)})$, if σ is substituted by $\sigma_{correct}$ 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.

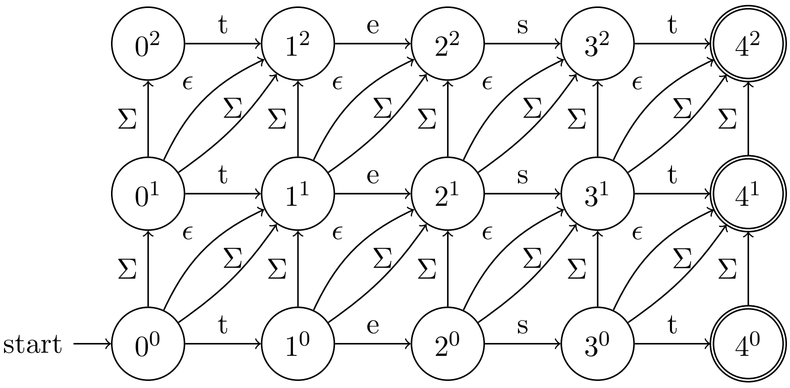


Figure 2.3: A non deterministic levenshtein automata for the word *test* with degree 2

The automata in example 2.3 is a non deterministic levenshtein automata for the word *test*. In the following description the *levenshtein automata* 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 automata. 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 automata can be in five different states namely 0^1, 1^2, 1^1, 2^2 and 1^0. 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 automata* the construction of an equivalent *deterministic levenshtein automata* 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 automata* is (Q', Σ, q_0', δ, F').

Deterministic Levenshtein automata example for the word test with max *levenshtein distance* 1.

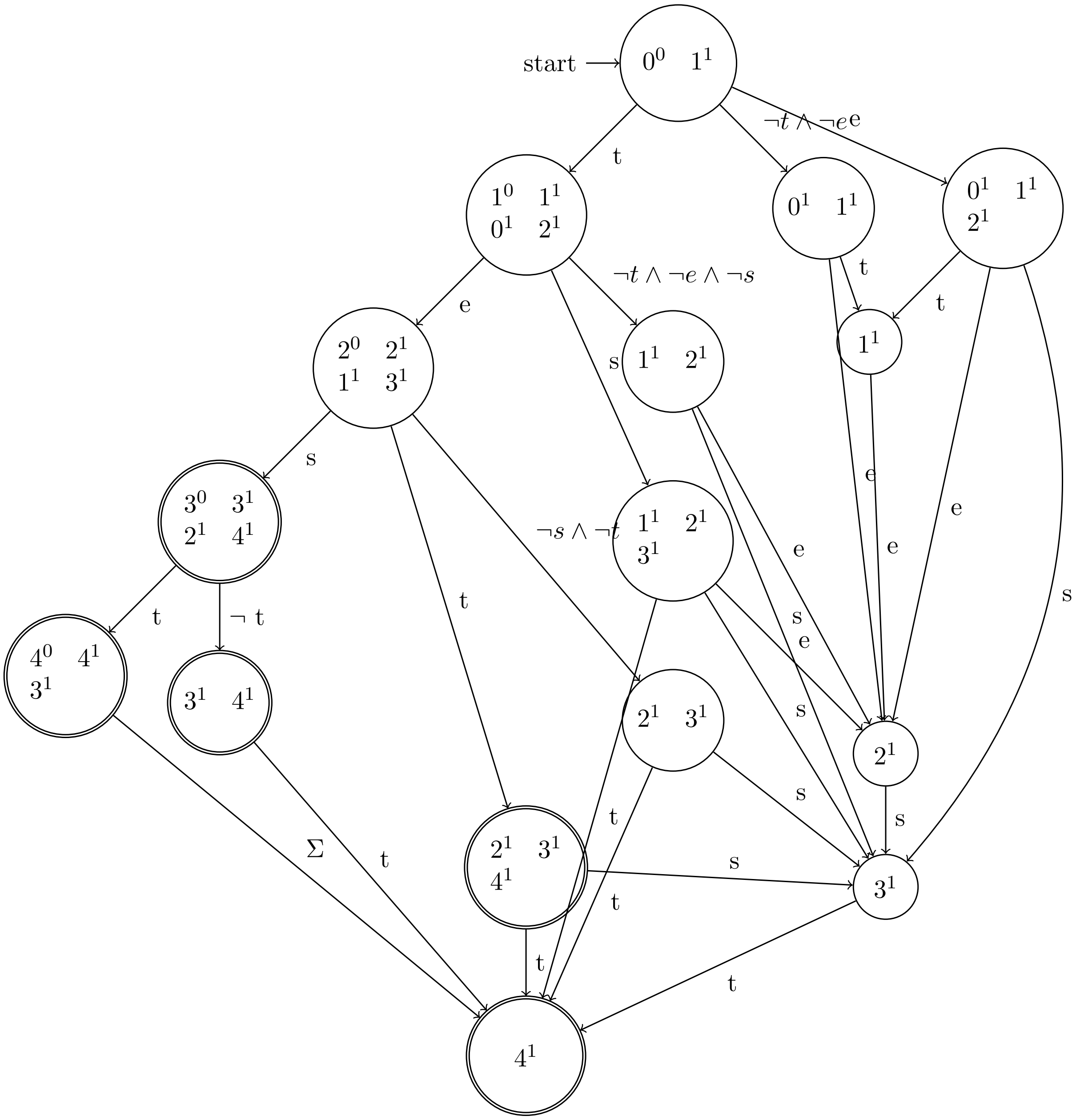


Figure 2.4: A deterministic levenshtein automata for the word *test* with degree 1

Let $(Q, \Sigma, q_0, \Delta, F)$ be a *non deterministic levenshtein automata* for the word *test* with maximum *levenshtein distance* 1 [2]. Then the *deterministic levenshtein au-*

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

tomata $DLA = (Q', \Sigma, q'_0, \delta, F')$ 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 automata with degree $n \in \mathbb{N}$ for a word $W \in \Sigma^*$ can decide in linear time if a word $V \in \Sigma^*$ has $d_L(W, V) \leq n$.

Algorithm 2 Levenshtein Automata is in distance

```
procedure IsInDistance($automata : DeterministicFiniteAutomata, term : String$)
    $i \leftarrow 0$
    $currentStates \leftarrow (0, 0)$                              ▷ add the initial state
    while currentStates.size > 0 AND i < term.length do
        $c \leftarrow term[i].toLowerCase$     ▷ c gets the current active lower cased letter
        $currentStates \leftarrow automate.nextState(currentStates, c)$
        $i \leftarrow i + 1$
    end while
    if currentStates includes a final state then
        return true
    else
        return false
    end if
end procedure
```

The maximum distance for a given word $W \in \Sigma^*$ depends on the length n of W. A word with length of $n \leq 3$ in the *Standard Thesaurus Economics* is usually an abbreviation and therefore is only used as a direct match. A word with length of $n > 3$ is used with a maximum distance of three due to the reason that most of the plural forms involve three changes (substitution, insertion and deletion) in the German language.

To get back to the original problem this paragraph explains the application of the *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 Thesaurus for Economics* into each automata. If a word from the *STW Thesaurus for Economics* ends in a final state it is used as a tag for the text. A possible optimization would be to calculate once all the automata's for the complete *STW Thesaurus for Economics* and to store it with an efficient data structure in a database. Currently the *STW Thesaurus for Economics* is stored in a triple store because it can easily be updated with the standard files from the website.

triple store definition

2.4 Recommendation

A recommendation system is a system that creates personalized item recommendations based on information about the items or the relationships 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 kind 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 relationship between the users and the items they are interested in might be needing some time. Although it needs to collect initial data it 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 the 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 system 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. So, 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 bought such as expensive digital goods or cars.

Hybrid approaches

All recommendation system approaches have advantages and disadvantages, therefore a combinations of the different approaches which are mentioned above could improve the user recommendations as long as the system has enough information about the items or users.

Conclusion

Knowledge based systems require too much domain knowledge that is not accessible for user generated questions and answers, therefore knowledge based systems are not feasible for the system. Collaborative recommendation systems have the *cold start problem*, so they 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 unapropriate 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. Moreover the content-based systems have the disadvantage that they do not make use of the information whether or not a user likes a document. Yet, they do not need to collect additional relationship information in order to recommend items. For these reasons the described recommendation system uses a collaborative filtering technique with pre filtered content based on the tags for each item.

2.4.2 Collaborative recommendation system

As mentioned above a collaborative recommendation system recommends items to users based on the interest of similar users. The following definitions are based on Jannach et al. [2011]

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

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

Definition 12 (Rating). *R is an $n \times m$ matrix with $n = |I|$ and $m = |U|$. Furthermore is $r_{n,m}$ a rating for item $n \in I$ and user $m \in U$ with $r_{n,m}$ 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 $i \in I$ where $r_{i,u}$ is empty, $\tilde{I_u}$ 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 $r_{i,v}$ 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.*

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 likes. If $i, \in \hat{I_u}$ and $u \in U$, then the prediction for i can be calculated with the similarity between i and all items $j \in \tilde{I_u}$. There are different possible ways to calculate the similarity between two items. However, the similarity calculation with the best performance is *adjusted cosine similarity* which is an optimized form of the *cosine similarity*[3]

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

$$cos(\overrightarrow{i}, \overrightarrow{j}) = \frac{\overrightarrow{i} \cdot \overrightarrow{j}}{||\overrightarrow{i}||_2 \cdot ||\overrightarrow{i}||_2} \tag{2.1}$$

If $\overrightarrow{i}, \overrightarrow{j}$ are rating vectors, the individual rating behaviour of a user needs to be taken into account to get better predictions with the similarities of $\overrightarrow{i}$ and $\overrightarrow{j}$. Due to the fact that different users have different average ratings. This results in the *adjusted cosine similarity.*

[3] see Sarwar et al. [2001]

Definition 16 (Adjusted Cosine Similarity)**.** *Let $i, j \in Items$ 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.*

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

Researches 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. [Sarwar et al. [2001]] The results of the cosine similarity are between -1 and 1 with 1 meaning identical, 0 meaning indifferent and -1 meaning opposite [Wikipedia [27 March 2013 at 10:24]]. Although the results vary between -1 and 1 those items that have a *cosine similarity* of ≤ 0 are too different and thus are not used for further calculations. The user item predictions can be generated with the similarities by using the following formula.

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* found out that the system only needs a subset of all items. According to *Sarwar et al* a sample of the 25 most similar items is needed to generate good recommendations [Sarwar et al. [2001]]. 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. So, more information about the user interest is available before the actual recommendations can be calculated. 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.

This example calculates the item based prediction for User1 and Item5

$$sim(Item5, Item1) = \frac{3 \cdot 3 + 5 \cdot 4 + 4 \cdot 3 + 1 \cdot 1}{\sqrt{3^2 + 5^2 + 4^2 + 1^2} \cdot \sqrt{3^2 + 4^2 + 3^2 + 1^2}} = 0.99$$

	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

sim(Item5, Item1) = 0.99
sim(Item5, Item2) = 0.74
sim(Item5, Item3) = 0.72
sim(Item5, Item4) = 0.94
With these similarities the following prediction can be calculated.

$$pred(User1, Item5) = \frac{0.99 \cdot 5 + 0.74 \cdot 3 + 0.72 \cdot 4 + 0.94 \cdot 4}{0.99 + 0.74 + 0.72 + 0.94} = 4.07$$

As a result User1 would most likely rate Item5 with 4.07.

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. This is done by decomposing a term by document matrix and using the item vectors of the decomposition to find documents based on queries that are decomposed into term vectors [Deerwester et al. [1990]].

Definition 18 (Vector).

Definition 19 (Matrix).

Definition 20 (Linear Dependent, Linear Independent).

Definition 21 (Rank of a Matrix).

Definition 22 (Matrix Multiplication).

Definition 23 (Orthogonal Matrix).

Definition 24 (Eigenvalue of a Matrix).

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

$$SVD(M) = U\Sigma V^t \tag{2.4}$$

U and V are orthogonal matrices with dimension $m \times r$ and $r \times n$. 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 [4] [Allaire and Kaber [2008]]. The columns of U are the corresponding eigenvectors to the eigenvalues of AA^t. On the other hand are the columns of the Matrix V the corresponding eigenvectors to the eigenvalues of A^tA [5] [Allaire and Kaber [2008]]. So, $AA^t \cdot u_i = \sigma_{i,i}^2 * u_i$.
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. 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 $A_k = U_k \times \Sigma_k \times V_k^t$ 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 interests by removing the last rows of matrix U.[6]
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 calculated 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. For this task the algorithm takes all similar users U_{sim} for $user_a$.
Afterwards it uses the top rated items from all $user_b \in U_{sim}$ that are not rated by $user_a$ $I'_{user_a,user_b}$. $\overline{r_a}$ denotes the average rating of user a. Finally for each item $i \in I'_{user_a,user_B}$ the prediction is calculated.

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

[4] Due to the fact that AA^t and A^tA share the same eigenvalues.

[5] The eigenvalues are the squares of $\sigma_{i,i}$

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

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* is used to calculate the following similarities.

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

- sim(User1, User2) = 0.222322
- sim(User1, User3) = 0.561072
- sim(User1, User4) = 0.151704
- sim(User1, User5) = 0.896770

The following itemization contains the average user ratings.

- $\overline{r_{User2}} = 2.4$
- $\overline{r_{User3}} = 3.8$
- $\overline{r_{User4}} = 3.2$
- $\overline{r_{User5}} = 2.8$

With these similarities and average ratings the following prediction 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.

2.4.3 The recommendation process

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. Therefore 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

$$
\begin{array}{cccccc}
 & \text{Item1} & \text{Item2} & \text{Item3} & \text{Item4} & \text{Item5} \\
\text{User1} & 5 & 3 & 4 & 4 & \\
\text{User2} & 3 & 1 & 2 & 3 & 3 \\
\text{User3} & 4 & 3 & 4 & 3 & 5 \\
\text{User4} & 3 & 3 & 1 & 5 & 4 \\
\text{User5} & 1 & 5 & 5 & 2 & 1
\end{array}
=
$$

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

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

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

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

in order to remove noise from the user item table and to improve the execution time of the recommendation process.The cosine similarity is calculated between the columns of the matrix U_k. 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.

2.4.4 Different Recommendation approaches

/todosvd variations The explained techniques that where used in the recommendation process can be used for other recommendation processes.

Only Item-Based Recommendations

The 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.

$$\begin{pmatrix} 0.475467 & 0.325691 \\ 0.342950 & -0.316949 \\ 0.535742 & -0.210593 \\ 0.455479 & -0.486113 \\ 0.402286 & 0.716108 \end{pmatrix}$$

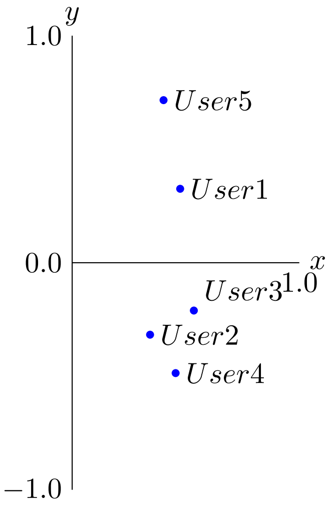


Figure 2.7: Similar Users based on Singular Value Decomposition

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 this svd-based recommendation system can be used as a standalone recommendation system. The runtime and performance of this system is compared with the recommendation system of this thesis in section 4.3.

2.5 Concept Relationships

write this

The last sections described the different concepts that are used in this thesis. This section emphasizes the relationship between the 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. Besides the items need a tag that describes the content of the item.

3 Implementation

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

3.1 Technologies

This system combines five different concepts namely the *question and answer system*, the *rating system*, the *tagging system*, the *rating system* and the *recommendation system*. For such a variety of techniques with different requirements there is the need for different technologies.

3.1.1 Programming Language

The requirements for the programming language are that it should be object oriented for good programming code encapsulation and that the software should work on a number of different operating systems due to the reason 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*. The different matrix calculations that one needs to have for the *singular value decomposition* must be implemented the way that they are able to calculate the results as fast as possible 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. A programming language that meets these requirements is the scala programming language.

Scala Programming Language

The scala programming language is an object oriented and functional programming language. Meaning that it is possible to write code with a functional programming paradigm together with an object oriented programming paradigm. Moreover 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 reason 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 therefore it 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. 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. Out of 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 section.

Performance auswertung google paper

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 in use by almost every web browser and thus 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 which is a web standard and therefore implemented by almost all web browsers the browser has to provide events for detecting scroll and focus events. On these events callback functions such as activity timers can be bound that are getting called if the event fires. 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*. This architectural choice was made in order to use the original *STW Thesaurus for Economics* 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 free to use relational database that is to the greatest possible extend conform with the SQL standard *ANSI-SQL 2008*. Thus the main functionality from this standard is available and works as expected.

4Store Triple Store

As mentioned at the beginning of the database section the triple store is used in order to be able to directly import the *resource description framework* (RDF) files from the *STW Thesaurus for Economics*. 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 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'
With these tools it is possible to create a graph that represents the data. 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 goes with the service oriented system architecture.

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 that is developed and maintained by *Twitter*.

Hypertext Transfer Protocol (HTTP)

HTTP is a stateless network protocol that is used amongst other things 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. There are five different http requests methods as of the standard HTTP 1.1 . . . The http protocol specifies a path that

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. Yielding 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 ore more server addresses together with the protocol and the connection limit. Thus it creates a client that can execute methods on a pool of 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. A *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 futures thus the functions will be executed if the server returns the value to the future. 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 in order to test the single recommendation techniques on a small and easy to edit test user-item table. The purpose of this application is to illustrate how the recommendation process works in the praxis to help understand the theory behind it.
The user interface is implemented with the *backbone.js* library that provides models, collections and views for creating model view controller websites on the client side.

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 not just opens 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. Figure 3.2 outlines the different services and the communication paths between those services.

3.3 Services

The individual parts of the system are separated into different services this is illustrated in figure 3.2. Each service inherits from the 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. These services return a standard http response that contains a json string as a return value.

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	-	
/calculateUserRecommendations	user id: Integer amount: Integer	List (user, rating)
/calculateUserPrediction	user id: Integer item id: Integer	List (user, rating)
Post Methods	Parameter: Type	
-	-	-

The */calculateSimilatities* method starts the item similarity calculation of the *ItemBasedService* Service. After the calculation finishes the *RecommendationService*

Service starts the svd based similar user calculation of the *SVDBasedService* Service. These similarity values are stored in the database by the individual service for future rating predictions. The */calculateUserRecommendations* method takes an user id, a number of predictions and a list of tags as parameters. This method forwards the parameters to the *SVDBasedService* Service and returns the list of item rating pairs. The */calculateUserPredictions* method takes an user id and an item id as parameters. If the item id is empty the prediction for all items are calculated if enough similar item ratings exist. The final prediction calculation is handled by the *SVDBasedService* Service.

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 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 an 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 not rated items for the specific user and returns it as a list.

Algorithm 3 Item Similarity Calculation

```
procedure ItemSimilarity
    items ← load all items
    seenTogether ← empty Set of item tupel
    for each $item_1$ in items do
        newItems ← empty Set of items
        for each user that rated $item_1$ do
            for each $item_2$ that is rated by user do
                if $item_2 \neq item_1$ and seenTogether does not contain $(item_2, item_1)$ then
                    add $(item_2, item_1)$ to seenTogether
                    add $item_2$ to newItems
                end if
            end for
        end for
        calculate similarities for $item_1$ and all items from newItems
    end for
end procedure
```

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. Where the columns of one matrix responds to the user interest of the original user-item 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 calculating 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	
-	-	-

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 calculates 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.

3.3.4 TaggerService

The *TaggerService* 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 automatas reads the complete *STW Standard Thesaurus Economics* 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

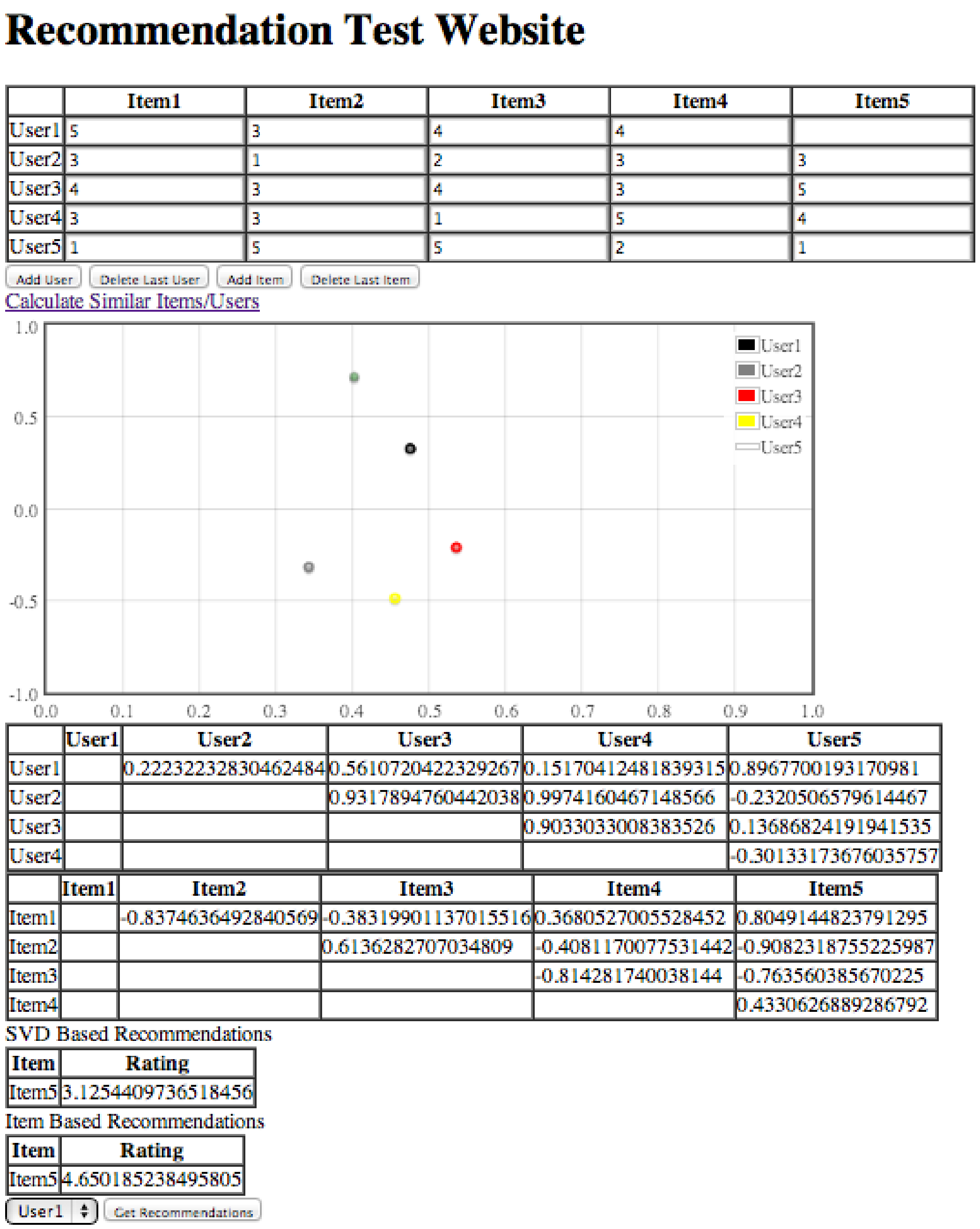
The *tegText* method takes a text as parameter. The single words of the text are extracted into a list. For each word in this list a deterministic levenshtein automata is generated. For this process the service creates a finite automata for each word with a specific degree. A degree states the maximum allowed levenshtein distance between the word for which the automata is generated and the word that the automata can read. This degree depends on the word size. For a word with more than five letters a degree of three is used otherwise a degree of one is used.The states are written as tupel of the form (c, d) where 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 automata 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 thru 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 automata is to mark all of the last states where the current position in the word is the size of the word as final states. This finite levenshtein automata is transformed into a deterministic finite levenshtein automata.

add transformation algorithm

3.3.5 RatingService

RatingService		
Standard Port	12000	
Get Methods	Parameter: Type	Return Value
-	-	-
Post Methods	Parameter: Type	Return Value
/rateItem	user id: Integer item id: Integer time spend: decimal time scroll: decimal	JavaScript if item is unknown

3.3.6 WebService



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

	User1	User2	User3	User4	User5
User1		0.2223232830462484	0.5610720422329267	0.15170412481839315	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	-0.38319901137015516	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

Figure 3.1: The Graphical User Interface for The Recommendation Process

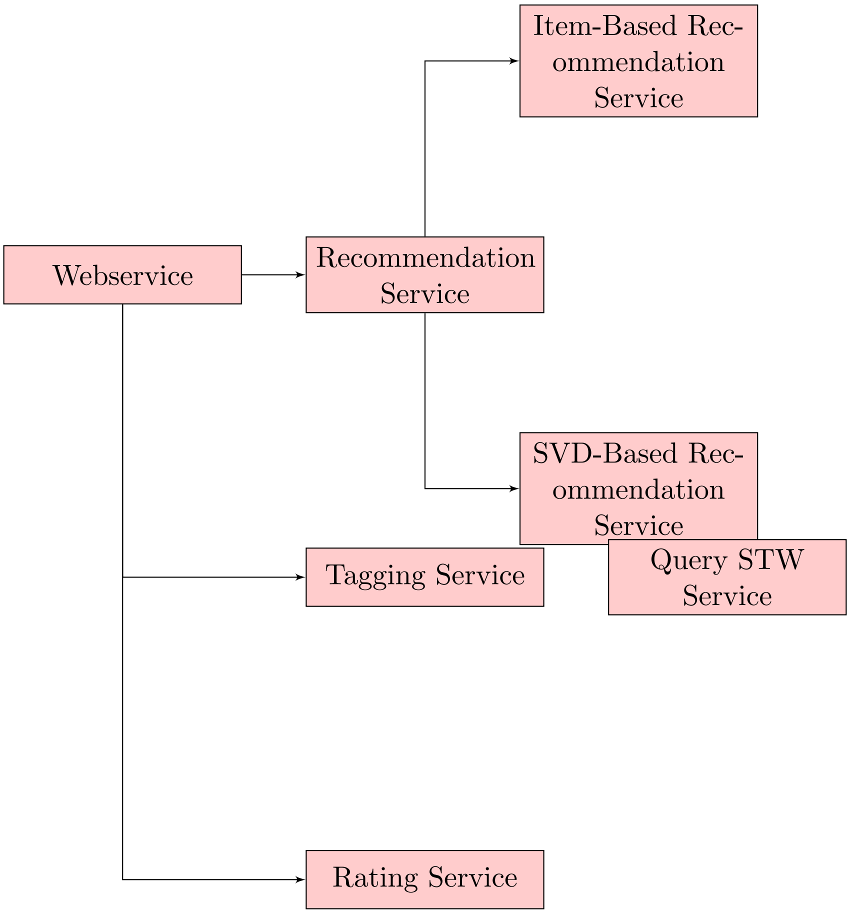


Figure 3.2: The Service Architecture of the system

4 Evaluation

This chapter is about . . .

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 about 70% to the user ratings are the time a user spend on a website and the time a user scrolls. Each of the 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 total rating by calculating the average of both ratings. This total 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 visits 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	Total 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 result 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 behavior. 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 behavior. The total rating is the implicit rating that the system uses as the rating of that user for the specific item. These total ratings are integer values and are therefore rounded. The values of the *Scroll Time* nearly match the average times that were found out 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 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 *Total Rating* which is introduced in this thesis correlate with the combined user interest of the *Scroll Rating* and the *Time Rating* and therefore represent the expected result. Moreover the total 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 automata* for every word in an item and read each word from the *STW Standard Thesaurus Economics* into the automata. If the word ends in a final state it is used as a tag for the item. Such a *levenshtein automata* 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 Wikipedia [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 Standard Thesaurus Economics.* 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 Standard Thesaurus Economics.* However, not all of the words are in the same form as they are found in the *STW Standard Thesaurus Economics.* Thus 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. Moreover, the time that the algorithm needed for creating the 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 Automata 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 automata 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 automata.

For the second evaluation ten words are used with 7 non matching words and three words from the *STW.* Two of the three words are modified therefore 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 the Levenshtein automata is 0.2843 seconds which is approximately three times as high as the Levenshtein automata 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 the 100 Levenshtein automata is 0.4751 and in total the computation needs 79.9616 seconds.

For the final tagging algorithm test a text with 1000 words is used. 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 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. However 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. Therefore 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

state that each recommendation technique used almost the same prediction calculations

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. First the evaluation of the performance of the different approaches by using the wall clock time. Second 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 of 1 to 5 and is afterwards presented with movies that might be interesting for him. To compare the accuracy of the recommendation systems the *Mean Absolute Error* (MAE) is used.

Definition 25 (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 of 1 to 5. Out of this realisation Herlocker et al. [2004] created the hypothesis that collaborative recommendation systems that are tuned to its optimum can not get much more accurate than a mean absolute error of 0.73. They explain this hypothesis with the reason that users tend to provide inconsistent ratings when they are asked to rate the same movie at different times and therefore 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 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. Thus 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 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 approximately represents 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 surprisingoutcome of the evaluation 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 behavior 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.

can I write this?

any reasons?

The prediction time for an 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 the 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 an 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 therefore should generate the poorest user similarities. The offline computation time of this system consist mostly of the same parts as the *SVD Concept*. Namely by calculating the average user rating for each user, creating the rating matrix from the database, calculating the singular value decomposition, creating the approximation of the matrix U that is part of the singular value decomposition, calculating the cosine similarity between the columns of the matrix U approximation and storing this 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 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 is has the second best mean absolute error of all testet systems. The offline calculation time consists of calculating the average user ratings, calculating the average item rating, creating the rating matrix from the database, calculating the singular value decomposition, creating the approximation of the matrix U which is part of the singular value decomposition, calculating the cosine similarity between the columns of the matrix U approximation and storing this user similarities in the database. These calculations are the offline calculation time processes for the *SVD Three* approach plus the calculation of the average item ratings. These average item ratings are needed to fill in 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 the finding of similar users that rated a specific item and the calculation of the predictions. Therefore it has almost the same prediction time as the other SVD based approaches that were tested in this evaluation. To sum 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 that the computation time.

The last two systems are added to the evaluation to compare the results of the more advanced recommendation systems with naive 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. Therefore it can be concluded that the interests of the individual users from the *MovieLens* database does not vary widely. However this technique has to calculate the average user ratings of all users which is very fast and needs to load only one value from the database to 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. However it does not vary widely from the MAE of the *SVD Average* technique and 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 teams of Sarwar et al. [2001] showed that this method can yield an MAE of up to 0.73 by improving the prediction calculation but 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 result generally 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 a specific use case reveal the same findings as this evaluation.

5 Future Work

5.1 Rating System

5.2 Tagging Service

Direct implementation of the deterministic levenshtein automata, table based approach. Save levenshtein automatas from all *STW Standard Thesaurus Economic* words with an appropriate datastructure.

5.3 Recommendation System

Bibliography

GrÃ©goire Allaire and Sidi Mahmoud Kaber. *Numerical Linear Algebra.* Springer, 2008.

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.

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.

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

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

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

Klaus Schulz and Stoyan Mihov. Fast string correction with levenshtein automata. *International Journal on Document Analysis and Recognition*, 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.

Wikipedia. Lorem ipsum — Wikipedia, the free encyclopedia, 2013. URL `http://en.wikipedia.org/wiki/Lorem_ipsum`. [Online; accessed 03-June-2013].

Wikipedia. Cosine similarity, 27 March 2013 at 10:24.