Diplomarbeit

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

Christian-Albrechts-Universität zu Kiel
Institut für Informatik
Lehrstuhl Technologie der Informationssysteme

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

List of Figures

List of Tables

List of Algorithms

Abkürzungen

Abstract

1. Introduction

1.1. Related Work

1.2. Contributions

1.3. Overview

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. 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.

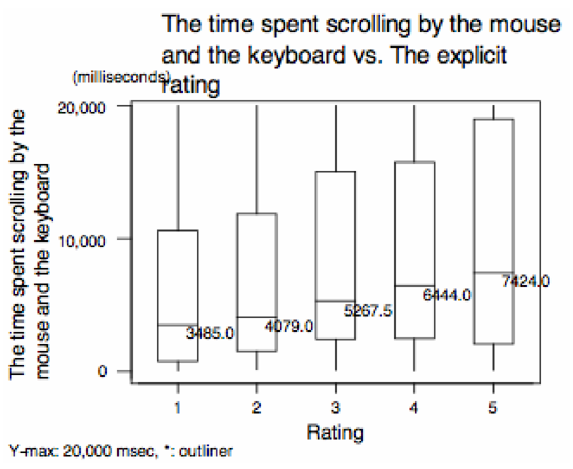


Figure 2.1.: Time spent scrolling Claypool et al. [2001]

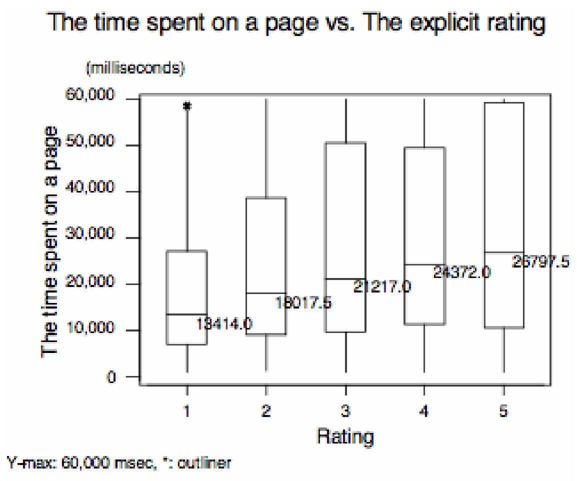


Figure 2.2.: Time spent on a page Claypool et al. [2001]

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. 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 word *library* out of the word *libraries*. If it creates the word *libraries* out of the word *library* it has to perform a substitution and two insertions.

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

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

Algorithm 1 Recursive Levenshtein Distance Algorithm

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

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

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

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

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

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

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

Definition 9 (Degree Levenshtein Automata)**.** *Let $W \in \Sigma^*$ and $n \in \mathbb{N}$. A finite state automaton 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.

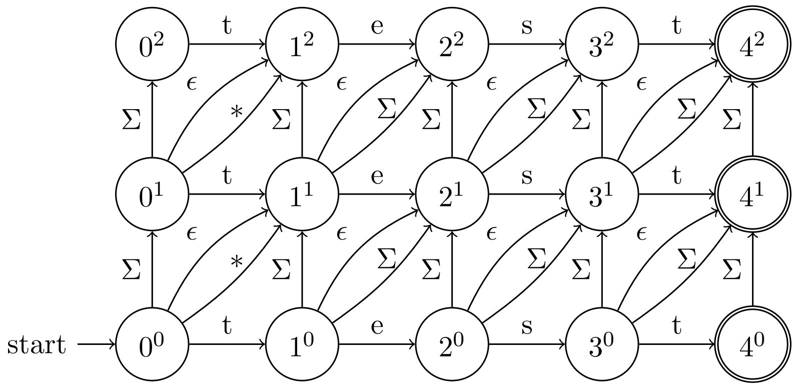


Figure 2.3.: A non deterministic levenshtein automata for the word *test* with degree 2

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 $Win\Sigma^*$ 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) $\in \Delta \vee$ (r, ϵ, q) $\in \Delta \vee$ (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.

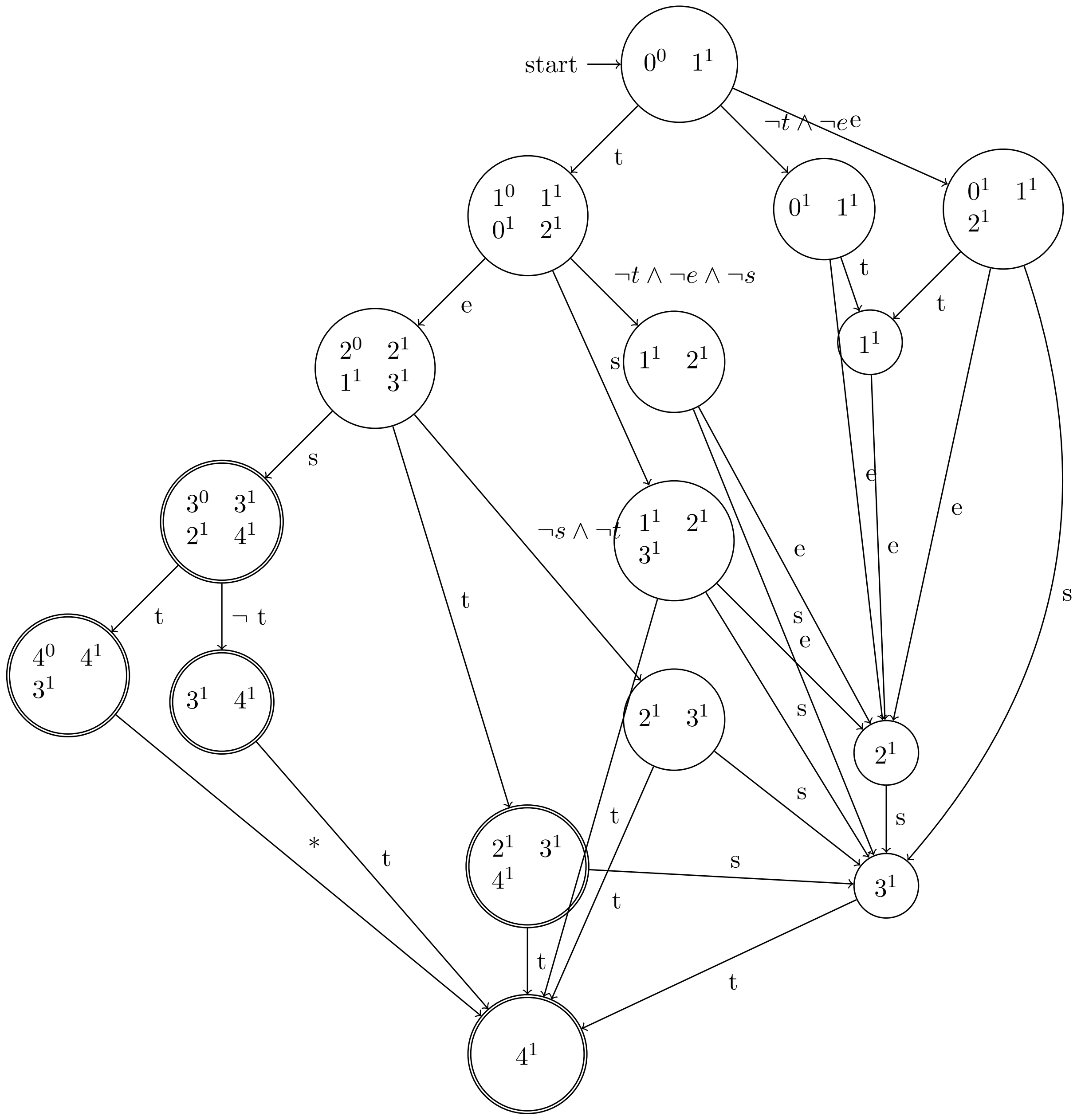


Figure 2.4.: A deterministic levenshtein automata for the word *test* with degree 1

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

[1]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

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

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.

try to find source

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

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 that is the reason why the 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 only able to recommend items that are approved by similar users. Moreover the content-based systems have the disadvantage that they do not make use 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). *$U = \{u_1, u_2, u_3, \ldots, u_n\}$ is a set with u_i users from the recommendation system. With $n, i \in \mathbb{N}$ and $i \leq n$.*

Definition 11 (Item). *$I = \{i_1, i_2, i_3, \ldots, i_n\}$ 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 $i, \in \hat{I_u}$, $u \in U$, 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*[2]

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.*

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.*

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

[2]see Sarwar et al. [2001]

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)**.**

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

Example
The table in figure 2.5 represents a user item relationship.

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

Figure 2.5.: User Item Relationship Table

This example calculates the item based prediction for User1 and Item5

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

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.
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]]. 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]].

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 [3] [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

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

the eigenvalues of A^tA [4] [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.[5]
The two dimensional matrix U_2 is created by removing all rows of matrix U up to the first two rows. This matrix U_2 can be seen as a value table that can be represented by a graph. Therefore, it enables a graphical presentation of the user similarities. Furthermore the user similarities can be 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]The eigenvalues are the squares of $\sigma_{i,i}$
[5]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 items that are not rated. This improved user item table is decomposed into three matrices U, Σ and V^t. The matrix U represents the user interests and is reduced by k rows in order to 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.

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

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

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

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

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

2.4.4. Different Recommendation approaches

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

Only Item-Based Recommendations

Only SVD-Based Recommendations

Use Item Similarities of the V Matrix

2.5. Summary

Add tags to all items, filter items by tag before using them for recommendations. Calculate the average ratings, calculate the item similarities, calculate user predictions, calculate svd, calculate U_k, calculate user similarities. Create on time recommendations.

write this

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

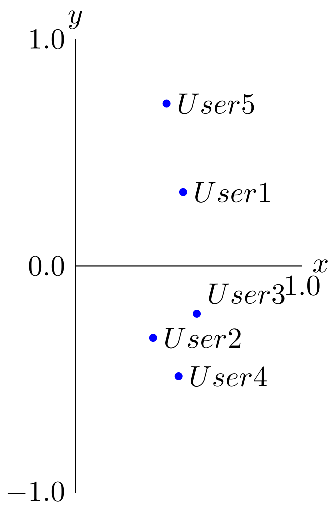
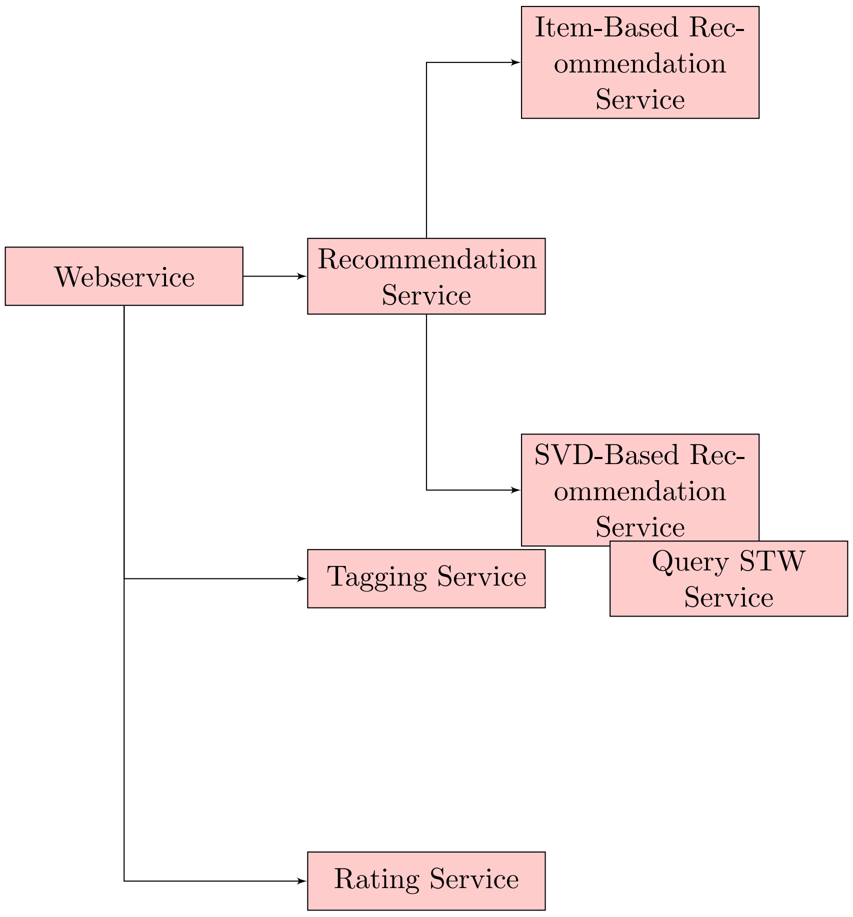


Figure 2.7.: Similar Users based on Singular Value Decomposition

3. Implementation

3.1. Technologies

3.2. Architecture



3.3. Services

3.3.1. ItemBasedService

3.3.2. SVDBasedService

3.3.3. RecommendationService

3.3.4. LevenshteinDistanceService

3.3.5. TaggerService

service architecture

4. Evaluation

4.1. Rating

4.2. Tagging

4.3. Comparison Of Different Approaches

4.3.1. Performance of the Recommendations

4.3.2. Accuracy of the Recommendations

Item-Based

Singular Value Decomposition

Hybrid Of Item-Based and Singular Value Decomposition

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

A. Erster Anhang

B. Zweiter Anhang

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.

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