[Studien|Diplom|Bachelor|Master]arbeit

**** Titel der Arbeit ****

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

angefertigt von:	**** eigener Name ****
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

The basic approach behind the system is, that it should be easy to integrate the technology into an existing system and that each described concept should work as a stand alone technology.

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, all answers only refer to the question and the user that started the question can mark one answer as correct. This question and answer site should open the information retrieval process on a website for the public. 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, therefore it is possible to recommend the questions with the correct answers to users that visit a website with the same subject. This process should increase the user interaction with the website and moreover help the user to understand the content that he is interested in. Such question and answer sites are widely spread on the internet and there exist 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.
From here on for better understanding the word item will be used instead of question and answer for an element that should be recommended.

why did I choose askbot some screen-shots

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. Explicit ratings are those where a user is explicitly asked for his opinion on a specific item. Implicit ratings are those where a system generates a rating according to the actions of a user. However 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. Moreover the explicit ratings might disturb the user experience of a website. Adding the explicit ratings with visible benefits to an existing *question answer webpage* would require too much customization, with attention to the possibility of replacing the *question and answer webpage* with a different open source version or an own implementation. Therefore it is a good approach to use implicit ratings to gather the interest of a user. The research group of Claypool et al. [2001] developed a web browser that tract implicit ratings and asked a user how they would rate the page in order to compare implicit with explicit ratings. The implicit ratings that they measured with their web browser where:

2.2.1. The time a user spends on a website

The timer starts after the website is completely loaded and stops if the user leaves the website or closes the window. Furthermore the timer is only active if the website is in focus in the web browser. They found that the time a user spends on a website is a good indicator of interest.

2.2.2. The time the mouse is in motion

This was measured by timing when the mouse cursor changed its position inside the active browser window. They found that the time a user moves the mouse cursor is proportional to the interest that a user has in the website. However due to the fact that some users use the mouse heavily while they read the website or are looking at interesting objects, other users tend to only use the mouse in order to click on objects. Therefore it is not possible to tell how much a user is interested in the website, it is only possible to tell which websites receive the least amount of interest.

2.2.3. The number of mouse clicks

The 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 that the website therefore would be valuable to the users. However the collected data showed that the number of mouse clicks on a website aren't a good indicator for interest.

2.2.4. The time a user scrolls

The developed web browser measured the time a user scrolls a website by keys and by mouse. Though the team of Claypool et al. [2001] found that each method on its own is a poor indicator of interest, but both methods combined is 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.

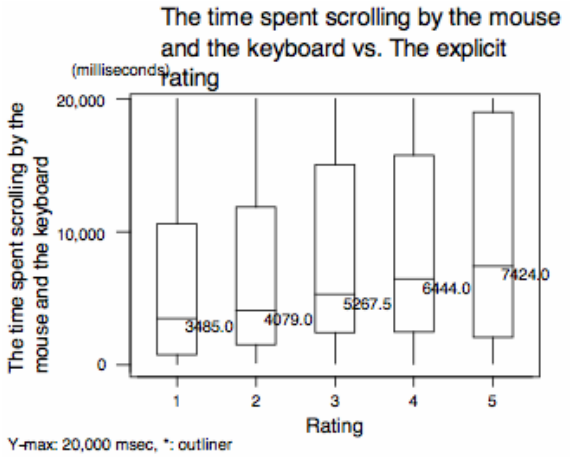


Figure 2.1.: Time spent scrolling Claypool et al. [2001]

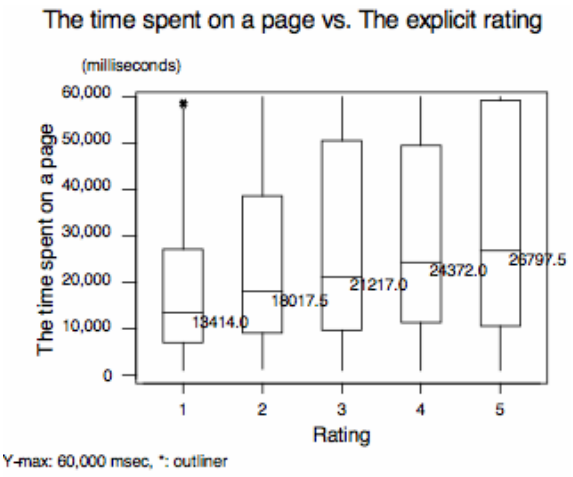


Figure 2.2.: Time spent on a page Claypool et al. [2001]

2.2.5. Concept

The best sources of implicit ratings that have an accuracy of about 70% to the explicit user 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]. This correlation between the implicit and explicit ratings can be explained by the fact that a user that is interested in an item takes a good look at the interesting website and reads the content that is presented on the site. This process takes time and the user has to scroll in order to see the whole content of the website. For the task of gathering implicit user ratings by the time that the website is in focus, the truncated mean time that all users spend on the item page is used as a benchmark. The percentages of the two rating processes result out of the two diagrams of figure 2.1 and figure 2.2.

improve this

Definition 1 (Truncated Mean)**.** *The truncated mean 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* $p \geq 0$ *and* $p < 0.5$*. With* $k = np$ *as the trimmed value,* $r = n - 2k$ *as the remaining values,* $v_j \in V$ *and* $j \in \mathbb{N}$*.*

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

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

A ranking 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 in front of the computer. Therefore it is possible, that a user is on a break and the currently active browser window might not be interesting for him. Such a predefined user interaction that can be used as an indicator that a user likes an item for the use case of a question and answer website can be an up vote or the writing of a comment or answer. A similar process 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 ranking of 1.
- If the user scrolls between 40% and 80% of the truncated mean time on the webpage it is used as a ranking of 2.

- If the user spends plus minus 20% of the truncated mean time on the item page, it is used as a ranking of 3.
- If the user spends between the truncated mean time plus 20% and 60% on the webpage it is used as a ranking of 4.
- If the user scrolls longer than the truncated mean time plus 60% it is used as a ranking 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 process. After calculating both ratings the average of both ratings is calculated and is used as the total rating.

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

The concept is build upon measurements of a field experiment of 75 students by Claypool et al. [2001], though due to the lack of time it wasn't possible to test this concept on a large user base. If a user returns on an item site the rating that the user action results is added onto the previous rating with a maximum sum of 5. This rating calculation results out of the observation that users tend to revisit websites that where interesting in the past and that might help the user by their current research. However the rating can't be set to the highest rating every time, because the user might visit an uninteresting website twice without realising it. Thus the subject of the website is interesting to the user but the content is not and therefore the low ratings of both visits result into a mediocre overall rating.

add this to evaluation

2.3. Tagging

In order to recommend the items with a matching subject on a website the system needs to be able to filter the items based on the content. The content can be described with the use of tags, these tags are the keywords of the content. For the context of economics the *ZBW* provides the *STW Thesaurus for Economics*. The *STW Thesaurus for Economics* providers vocabulary on any economic subject and it includes about 19000 terms and 6000 standardized keywords which can be used to match words from a text and provide therefore a good tagging base. However the *STW Thesaurus for Economics* only includes the basic form of the words. Therefore a normal test for equal is not sufficient enough due to the fact that it wouldn't match on plural forms and other affix word changes. So, the challenge for this tagging process is to find the correct words in the *STW Thesaurus for Economics* even if they are in a different form then the corresponding words from the text. 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.
Another challenge for using the stemming algorithm is that all words in the tagging base must be in the stem form. Otherwise the algorithm might reduce a word to its stem that would match in it's original form. Due to these maintenance problems the stemming algorithm is not suitable for this task. Another possible solution for the problem is to calculate the differences between the words from the text and the words from the tagging base and use the words from the text as tags if they are in a predefined difference range. 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*.

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. Example for a *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*. However if it creates the word *libraries* out of the word *library* it has to perform a substitution and two insertions. Both processes result a *levenshtein distance* of three. So, the *levenshtein distance* returns 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 compare algorithm 1.

Algorithm 1 Recursive Levenshtein Distance Algorithm

```
1: procedure LevenshteinDistance(s : String, t : String)
2:     lenS ← 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 ← 0
11:    else
12:        cost ← 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
```

However 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 finite automata is a 5-tupel of the form* $FA = (Q, \Sigma, q_0, \Delta, F)$.

- 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* $\Delta \subset Q \times \Sigma \times Q$

FA 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). *FA 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 FA 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). *$FA : p \xrightarrow{\omega} q$ with $\omega \in \Sigma^*$ states, that a path π for FA from p to q with label $\beta(\pi) = \omega$ exists.*

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

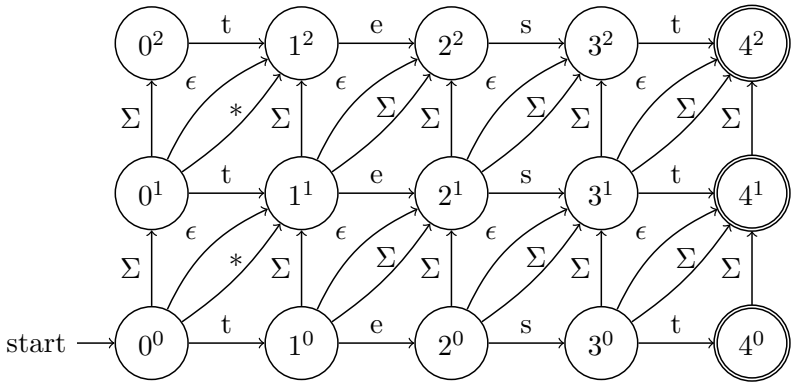


Figure 2.3.: A non deterministic levenshtein automata for the word *test* with degree 2

The elements are not exclusive, therefore all of these cases can be possible after reading only one letter.
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

Given a *non deterministic levenshtein automata* the construction of an equivalent *deterministic levenshtein automata*:

add source book Introduction to Automata Theory, Languages and Computation by Hopcroft, Motwani

- 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. finish automata

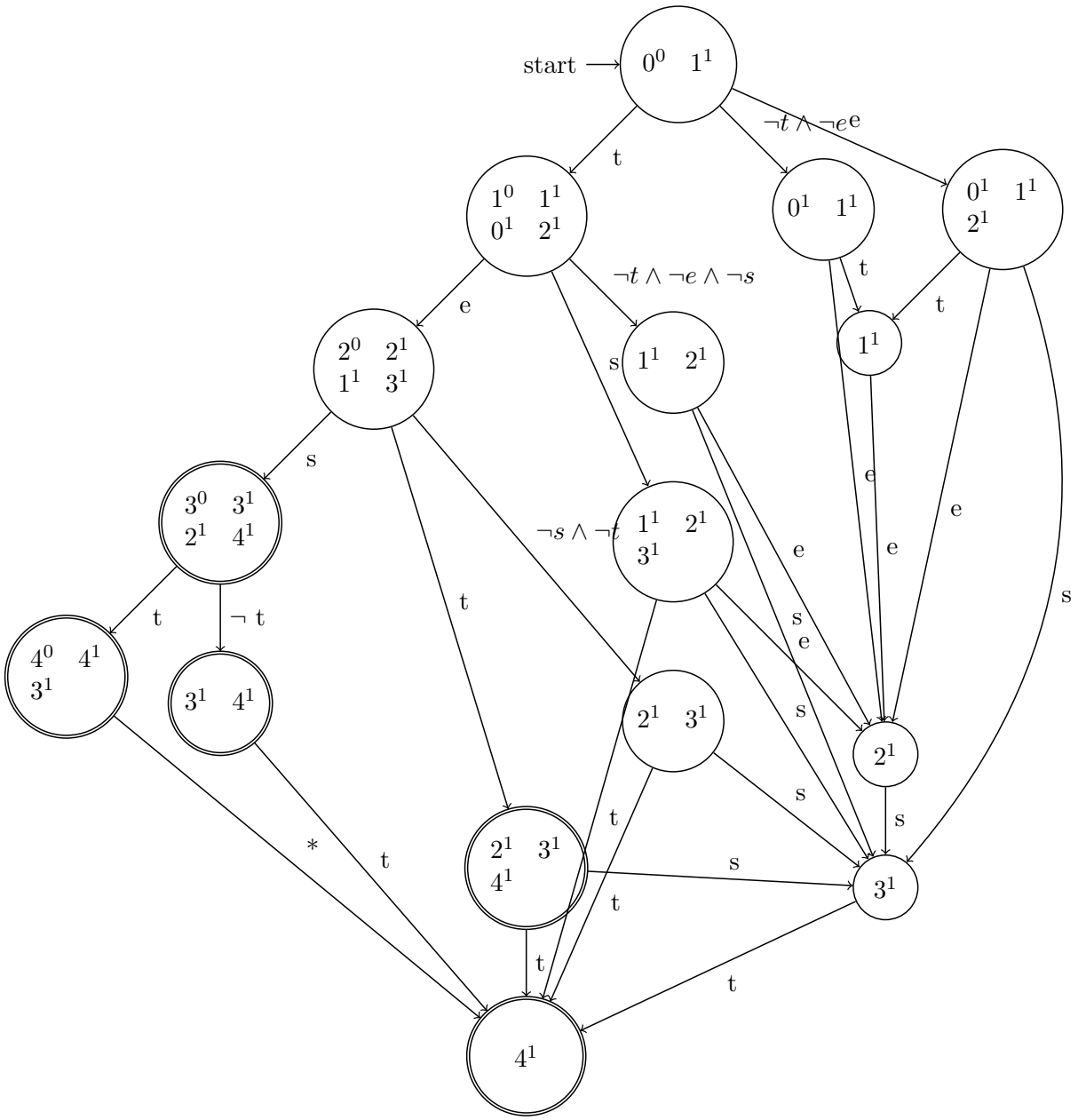


Figure 2.4.: A deterministic levenshtein automata for the word *test* with degree 1

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

[1] Therefore 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. Therefore 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 for a word $V \in \Sigma^*$ $d_L(W, V) \leq n$ source Lemma3.

Algorithm 2 Levenshtein Automata is in distance

```
1:  procedure isInDistance(automata : DeterministicFiniteAutomata, term : String)
2:      i ← 0
3:      currentStates ← (0, 0)                          ▷ add the initial state
4:      while currentStates.size > 0 AND i < term.length do
5:          c ← term[i].toLowerCase   ▷ c gets the current active lower cased letter
6:          currentStates ← automate.nextState(currentStates, c)
7:          i ← 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
```

describe word size maximum correction count correlation.

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 postfix endings have a maximum length of three in the German language.

try to find source

add more explanation how the levenshtein automata is integrated into the system

To get back to the original problem of finding 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.

2.4. Recommendation

A recommendation system creates personalized item recommendations for users. These recommendations can be created by different types of recommendation system, that exploit different information about the users or items.

more general stuff

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. However the relationship information between the users and the items might take some time to gather depending on the number of items in the system and the activity of the users of the system. Therefore it is a good approach if the items that needs 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 that can have an importance indicator. 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 it doesn't 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. However if the content is user generated the recommendation system can only recommend documents that fit the current context. It has no information if 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, that if a system has enough information about the items and the needs of the user, it 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, 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 aren't frequently requested and therefore no user history is available. Such as expensive digital goods, cars and so on.

Hybrid approaches

All recommendation system approaches have advantages and disadvantages, therefore it is a good approach to combine different techniques to improve the user recommendations, if the system has enough information about the items or users.

Conclusion

improve this

Knowledge based systems require too much domain knowledge that is not accessible for user generated questions and answers, therefore knowledge based systems aren't 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. Furthermore they have no information about the subject of the items therefore they might recommend unapropriate items to the current context. Nonetheless collaborative filtering is the only approach that takes the interest of similar users into account. Therefore it is able to only recommend items that are *approved*, in hindsight that similar users liked the items. Moreover the content-based systems have the disadvantage that they don't use the information whether or not a user likes a document. However they do not have the cold start problem.
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 techniques

explain probabilistic and slope one and why you didn't choose them

There are different types of collaborative recommendation techniques. 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 $k \in U$ hasn't rated an item $l \in I$ yet 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 concepts of item-based recommendations was introduced in 2001 by Sarwar et al *Item-Based Collaborative Filtering Recommendation Algorithms*. 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*. (see Sarwar et al)

Definition 15 (Cosinus 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 into the *adjusted cosine similarity*.

Definition 16 (Adjusted Cosine Similarity). *With $i, j \in$ Items and $\overline{r_u}$ 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 focus on prediction accuracy from all other similarity measures for item based algorithms . The results of the cosine similarity are between -1 and 1, with 1 meaning identical 0 meaning indifferent and -1 . However items that have a cosine similarity of ≤ 0 are too different and aren't used for further calculations. The user item predictions can be generated with the similarities by using the following formula.

source Sarwar et al 2001

find source

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

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

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 prediction can be calculated:

$$pred(User1, I5) = \frac{0.99 \cdot 5 + 0.74 \cdot 3 + 0.72 \cdot 4 + 0.94 \cdot 4}{0.99 + 0.74 + 0.72 + 0.94} = 4.07$$

Therefore User1 would most likely rate Item5 with 4.07.
The item similarity calculations can be calculated offline on a regular basis, every day or week 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 that the system only needs a subset of all items, a sample of the 25 most similar items are needed to generate good recommendations Sarwar et al. [2001]. Moreover it is a tested concept, amazon.com uses item based recommendations for their product recommendations Linden et al. [2003]. To get back to the overall concept item based predictions are used in order to decrease the scarcity of the rating matrix. So, more information about the user interest is available before the actual recommendations can be calculated.

write this

Definition 18 (Scarcity)**.** *Scarcity is . . . It should be minimized in order to . . .*

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 invented by , one of the main applications of the svd is information retrieval. The basic idea behind the svd based information retrieval is to match user queries to documents 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].

find source

anwendungsgebiete

Definition 19 (Vector)**.**

Definition 20 (Matrix)**.**

Definition 21 (Linear Dependent, Linear Independent)**.**

Definition 22 (Rank of a Matrix)**.**

Definition 23 (Matrix Multiplication)**.**

Definition 24 (Orthogonal Matrix)**.**

Definition 25 (Eigenvalue of a Matrix)**.**

add definition of orthogonal vector and orthogonal matrix and eigenvector / eigenvalue

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 is Σ 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 $\sigma_{i,i} \geq \sigma_{i+1,i+1} > 0$. Furthermore are the $\sigma_{i,i}$ the none negative square roots of the eigenvalues of AA^t and A^tA [2]. 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 [3]. So, $AA^t * u_i = \sigma_{i,i}^2 * u_i$.

add definition of orthogonal vector and orthogonal matrix and eigenvector / eigenvalue

To sum up, the matrix U corresponds to the columns of the matrix A and the matrix V corresponds to the rows of the matrix A. Furthermore 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.Thus $A_k = U \times \Sigma \times V^t$ yields the optimal rank k approximation. Therefore all column vectors u_i, with $i > k$, of matrix U will be multiplied by a zero vector from matrix Σ. 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

add source for this subsection: Incremental Singular Value Decomposition Algorithms for Highly Scalable Recommender Systems by Sarwar et al

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

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

source Numerical Lineal Algebra Allaire and Kaber Springer Verlag

source Numerical Linear Algebra

is this a reason why this spans

approximation of the user interests.[4] For example by removing all rows of matrix U up to the first two rows U_2 yields a two dimensional table that is drawable onto a graph. The user similarities can be calculated by using the cosine similarity between the columns of U_k.

The recommendations for $user_1$ are calculated by using the top rated items, that are unknown to $user_1$, of the most similar users compare algorithm.

improve this

how to calculate predictions with these tables

Example

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 compare figure 2.7. The angle between lines from the graph origin to the users are used to calculate the similarities.

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

$$=$$

$$\begin{pmatrix} 0.544178 & 0.0875457 & 0.303701 & -0.598262 & -0.496039 \\ 0.330582 & 0.270665 & 0.155794 & -0.281254 & 0.845033 \\ 0.517601 & 0.04377 & 0.429477 & 0.737011 & -0.0503875 \\ 0.438285 & 0.373564 & -0.800903 & 0.129864 & -0.100232 \\ 0.366854 & -0.881822 & -0.239996 & -0.0541261 & 0.165165 \end{pmatrix} \cdot$$

$$\begin{pmatrix} 16.499 & 0 & 0 & 0 & 0 \\ 0 & 4.93905 & 0 & 0 & 0 \\ 0 & 0 & 2.58239 & 0 & 0 \\ 0 & 0 & 0 & 1.20841 & 0 \\ 0 & 0 & 0 & 0 & 0.511218 \end{pmatrix} \cdot$$

$$\begin{pmatrix} 0.452438 & 0.336841 & 0.410894 & -0.456436 & -0.55197 \\ 0.403968 & -0.531238 & -0.483027 & 0.210155 & -0.526419 \\ 0.43523 & -0.601119 & 0.481498 & -0.122706 & 0.449817 \\ 0.463447 & 0.282982 & -0.586236 & -0.401113 & 0.447854 \\ 0.477391 & 0.403612 & 0.149459 & 0.756011 & 0.12371 \end{pmatrix}^T$$

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

2.4.3. Everything together

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.

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

$$\begin{pmatrix} 0.544178 & 0.0875457 \\ 0.330582 & 0.270665 \\ 0.517601 & 0.04377 \\ 0.438285 & 0.373564 \\ 0.366854 & -0.881822 \end{pmatrix}$$

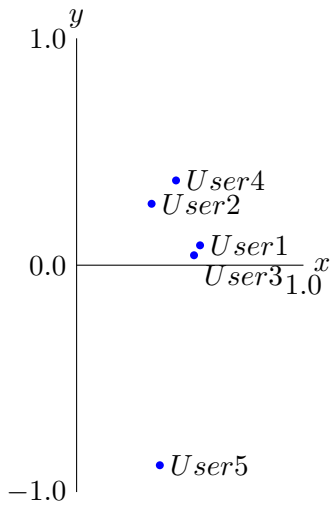
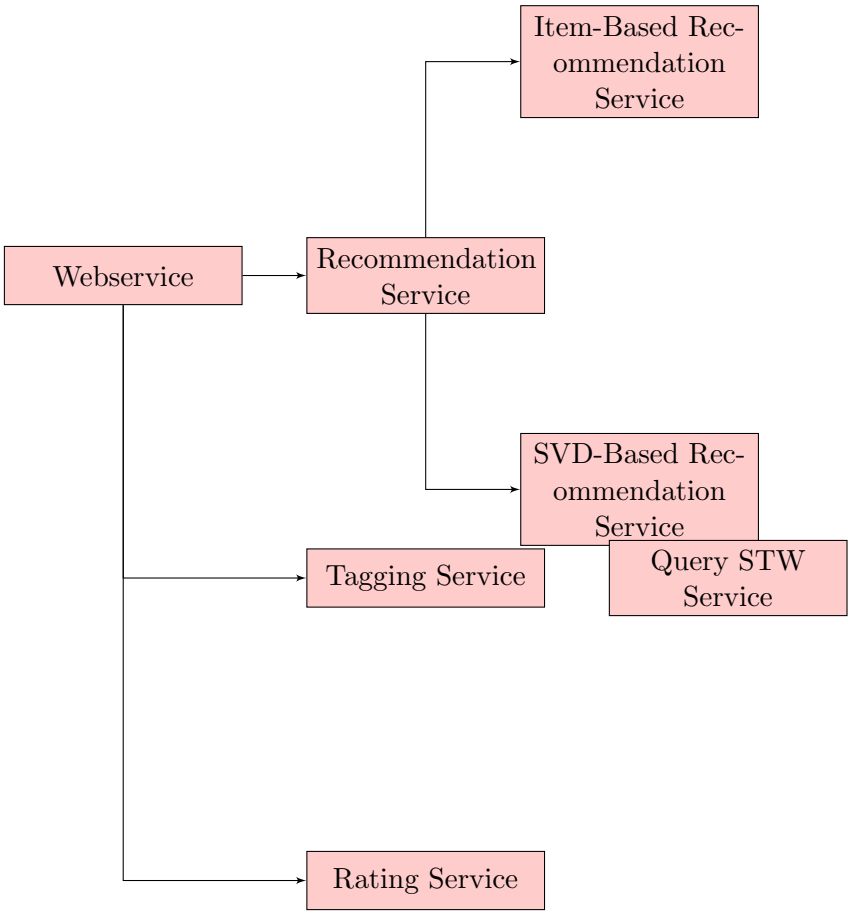


Figure 2.7.: Similar Users based on Singular Value Decomposition

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

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.