[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

2.1. Question Answer System

2.2. Rating System

2.3. Tagging

The purpose of tagging is to find one or more keywords for a text that describe the content. For the context of economics the *STW Thesaurus for Economics* includes16000 labels 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. For example a word that is used in it's plural form in a text would return false on a test for equal and would therefore not be a candidate as a tag for the text. 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. This 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, an arrow and the replacement for the suffix. More advanced algorithms might also use rules for prefix reduction and detecting irregular changes of the stem. 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 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 tagging base as tags if they aren't too different from the original word in the text. 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*.

check number of labels and add a source

reason why I didn't use stemming add source for example A Fast and Simple Stemming Algorithm for German words.

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. Therefore it 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.

$$levDist_{a,b}(i,j) = \begin{cases} max(i,j) & \text{if min(i,j)} = 0 \\ min \begin{cases} levDist_{a,b}(i-1,j)+1 \\ levDist_{a,b}(i,j-1)+1 \\ levDist_{a,b}(i-1,j-1)+[a_i \neq b_i] \end{cases} & \text{else} \end{cases}$$

This can be directly translated into a recursive algorithm.

find better solution to move 'if'

However the direct implementation of the *levenshtein distance algorithm* has a complexity of O(something). 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.

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
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
```

2.3.2. Optimized Levenshtein distance algorithm

Definition 1 (Non Deterministic Finite Automata)**.** *A finite automata is a tupel of the form $FA = (Q, \Sigma, q_0, \Delta, F)$. Q is the set of the states. Σ is the set of the input alphabet. $q_0 \in Q$ is the initial state and $F \subset Q$ 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 2 (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 3 (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 4 (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 5 (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.*

Definition 6 (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.*

check if I have to add source to the definitions

Definition 7. *$\mathcal{L}_{Lev}(n, W)$, $n \in \mathbb{N}$ and $W \in \Sigma^*$ is the set that denotes all words $V \in \Sigma^*$ such that $d_L(W, V) \leq n$.*

Definition 8 (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 deterministic finite automata was introduced in *Fast string correction with Levenshtein automata by Schulz and Mihov*. The deterministic finite automata for a word $W \in \Sigma^*$ is a *levenshtein automata* with degree $n \in \mathbb{N}$ with n max $d_L(W, V)$.

W will be changed to V and this is the levenshtein distance

The purpose of the *levenshtein automata* is to decide whether $d_L(W, V)$ is smaller than a specific 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 (i) and the current *levenshtein distance* between ω and W (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. The $\Delta : (Q \times \Sigma \times Q)$ is mapped as follows:

right translation

$$(i^j, \sigma) \rightarrow \begin{cases} (i+1)^j & \text{if correct} \\ (i)^{(j+1)} & \text{if insertion} \\ (i+1)^{(j+1)} & \text{if substitution or deletion} \\ \textit{empty} & \text{if j is equal to n} \end{cases}$$

Non Deterministic Levenshtein automata for the word *test*

The automata in example 2.1 is a non deterministic levenshtein automata for

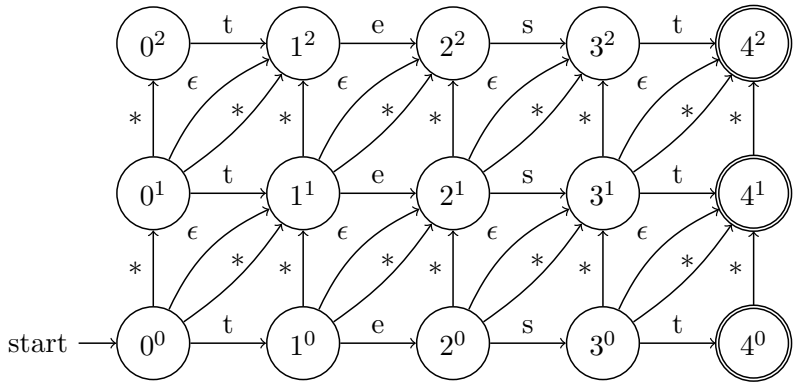

Figure 2.1.: A non deterministic levenshtein automata for the word *test* with degree 2

the word *test*. The initial state 0^0 is in the bottom left corner. If the automata

reads a correct letter it follows the horizontal path. The star (*) indicates that any element from Σ is accepted on this path. A vertical path is an insertion and on a diagonal path it can be a deletion with the empty word ϵ or a substitution of the letter indicated with *. Therefore after reading the letter 't' 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*.

Problems with non deterministic automatas. How to generate a deterministic automata. example

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').

finish automata

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

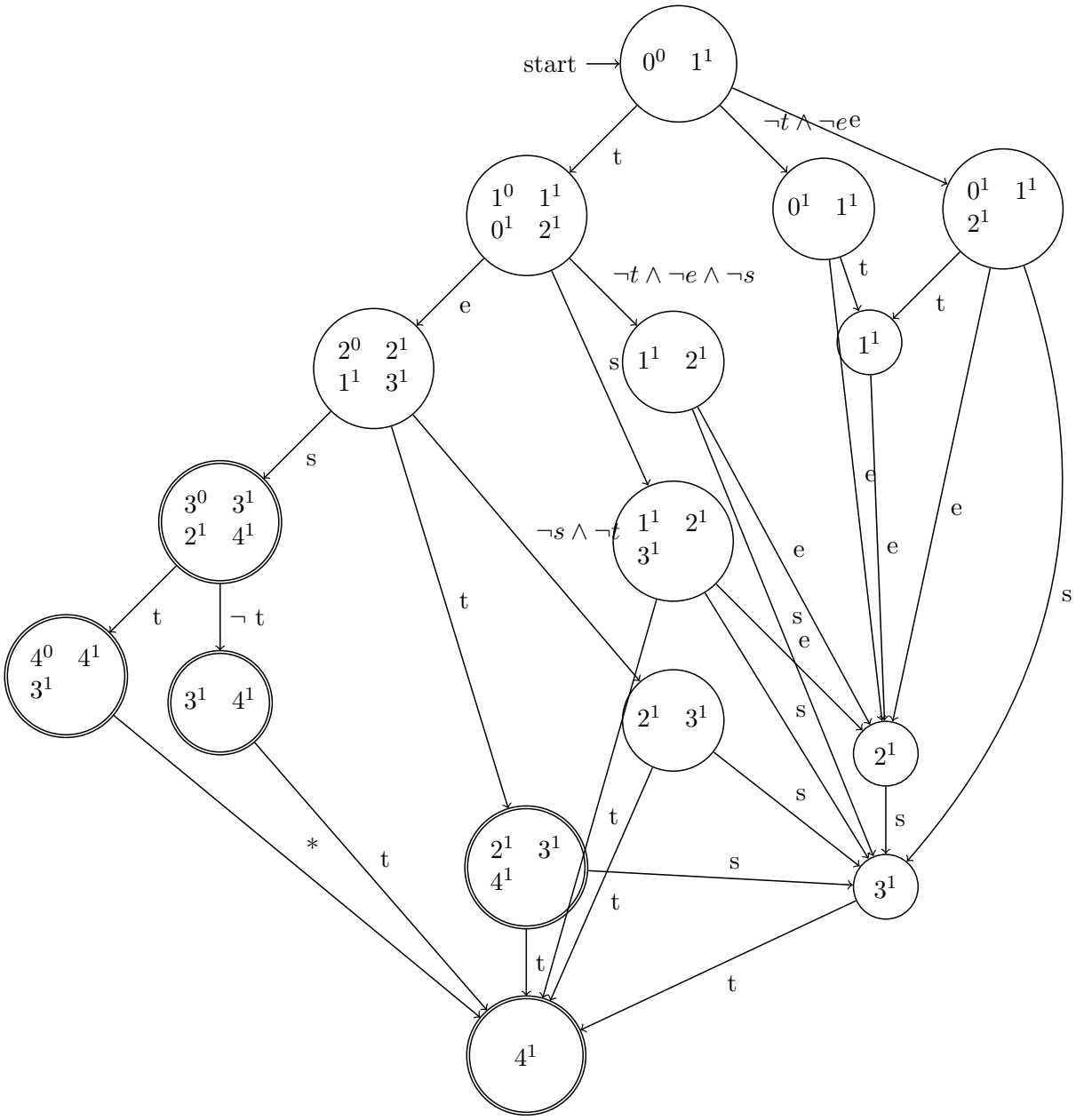

Figure 2.2.: A deterministic levenshtein automata for the word *test* with degree 1

Let $NDLA = (Q, \Sigma, q_0, \Delta, F)$ be the *non deterministic levenshtein automata* from figure 2.1 then the *deterministic levenshtein automata* $DLA = (Q', \Sigma, q_0', \delta, F')$ from figure 2.2 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.

The generation process of the levenshtein automata

The current system implementation calculates for every word from the text an automata and reads all words from the *STW Thesaurus for Economics*. 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 due to the fact that it can easily be updated with the standard files from the website.

describe word size maximum correction count correlation.

2.4. Recommendation

A recommendation system recommends items to a user.

more general stuff

Definition 9 (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 10 (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 11 (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 12 (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 13 (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.*

2.4.1. 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 we can calculate a prediction for i 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 14 (Cosinus Similarity)**.** *With* $i, j \in \mathbb{N}^n$, $n \in \mathbb{N}$.

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

Different rating behaviours

If $\overrightarrow{i}, \overrightarrow{j}$ are rating vectors, the individual rating behaviour of a user needs to be taken into account to get the correct similarity of $\overrightarrow{i}, \overrightarrow{j}$. This results into the *adjusted cosine similarity*.

Definition 15 (Adjusted Cosine Similarity)**.**

$$sim(i, j) = \frac{\sum_{u \in U} (R_{u,i} - \overline{R_u})(R_{u,j} - \overline{R_u})}{test} \tag{2.2}$$

add algorithm

With the similarities we can calculate the predictions with the following algorithm.

example

Example: So, item based is a good choice for websites that need fast scalable recommendations, because all the complex calculations can be computed offline. Furthermore the system only needs a subset of all items because a sample of the 25 most similar items are needed to generate good recommendations see Sarwar et al page 8. Moreover it is a tested concept, amazon.com uses item based recommendations for their product recommendations. The described systemuses item based predictions to decrease the scarcity of the rating matrix.

how to call the software implementation?

Definition 16 (Scarcity)**.** *Scarcity is . . . It should be minimized in order to . . .*

write this

Therefore all the calculations that are done for the item based algorithm can be computed offline.

2.4.2. Collaborative Filtering

Collaborative filtering is the task of recommending items based on the interest of similar users.

2.4.3. Singular Value Decomposition

write this

Definition 17 (Singular Value Decomposition)**.** . . .

The singular value decomposition is a mathematical concept created by . . . . The interesting part in hinsight of recommendation system is that with the singular value decomposition it is possible to create the best matrix approximation of the original matrix. Therefore the rows of the original matrix that represent the users can be approximated by a single two dimensional matrix. Thus it is easily possible to calculate the similarity between users. Forexample with cosinus similarity. The calculation of the similar users can be created offline and theirefore fits with the item based recommendation technique. So it is possible to create a pipeline architecture that starts with the item based technique to decrease the scarcity of the user item matrix and if this has finished can calculate the similarity between users. All this information can be stored in a database and can be used to calculate user recommendations if they are needed.

quote proof

3. Implementation

3.1. Technologies

3.2. Architecture

4. Evaluation

4.1. Comparison Of Different Approaches

4.1.1. Performance of the Recommendations

4.1.2. 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