

# Generating Search Term Variants for Text Collections with Historic Spellings

Andrea Ernst-Gerlach and Norbert Fuhr

University of Duisburg-Essen  
ernst@is.informatik.uni-duisburg.de  
fuhr@uni-duisburg.de

**Abstract.** In this paper, we describe a new approach for retrieval in texts with non-standard spelling, which is important for historic texts in English or German. For this purpose, we present a new algorithm for generating search term variants in ancient orthography. By applying a spell checker on a corpus of historic texts, we generate a list of candidate terms for which the contemporary spellings have to be assigned manually. Then our algorithm produces a set of probabilistic rules. These probabilities can be considered for ranking in the retrieval stage. An experimental comparison shows that our approach outperforms competing methods.

## 1 Introduction

In 2005, we have seen a number of initiatives addressing the problem of digitising books and making them available on the Internet, following earlier less ambitious projects like e. g. project Gutenberg<sup>1</sup>. The US search engine Google proclaimed an effort to digitise 15 million books. Some months later, the Open Content Alliance<sup>2</sup> was formed by several companies, research institutes and universities from the US. Since these initiatives are focusing on books in English only — and mainly as a reaction to the Google digitisation initiative — the European Union plans to create a European digital library in order to preserve the culture of the European countries. This library should include texts from the traditional European libraries and make them available on the Internet. So far, 19 European libraries have signed the corresponding manifest [6]. There have been already several digitisation projects in the past, but only a small fraction of the library content is digitised so far. With a European digital library project, the collections that are available in the Internet could be growing exponentially.

In contrast to countries with institutions defining spelling standards (e.g. Spain, France), English<sup>3</sup> and German [11] spelling was not stable over several centuries. English spelling was more or less fixed around 1800. In contrast, German spelling was not

<sup>1</sup> <http://www.gutenberg.org/>

<sup>2</sup> <http://www.opencontentalliance.org/>

<sup>3</sup> [http://en.wikipedia.org/wiki/English\\_spelling](http://en.wikipedia.org/wiki/English_spelling), access 20 January 2005 11:05

standardised until 1901/1902. Before that date, there was the rule 'write as you speak' (phonological principle of spelling) [8]. Because of the various dialects and the variations over time, German spelling before 1900 was highly time- and region-dependent. But even for languages like French where the orthography has been standardised, early spelling variants are occurring [2]. Furthermore, the predominant part of the 6,000 contemporary spoken languages never became official languages and thus, they have never been standardised at all [15].

The non-standard spelling produces problems when searching in the historic parts of digital libraries. Most users will enter search terms in their contemporary language which differs from the historic language used in the documents. In order to solve this problem, our project deals with the research and development of a search engine where the user can formulate queries in contemporary language for searching in documents with an old spelling that is possibly unknown to the user. For this purpose, we are developing transformation rules for generating historic spellings from a given word.

More specifically, our project aims at the following goals:

- The development of time- and location-specific rule sets. The revision of rules from the text basis and from statistical analyses should be possible.
- The development of new distance measures for spelling variants on the basis of a modified Levenshtein similarity measure.
- Application of the search engine in other German digitisation projects (e.g. the Nietzsche project [1]).

The search engine under development is based on the probabilistic information retrieval engine PIRE [9] and will create a platform that supports the interactive, iterative development of new rules.

The paper has the following structure. In Section 2, we give a brief survey over related work. Section 3 discusses approaches for the search in text collections with non-standard spelling, and outlines our work. The core of our approach is presented in Section 4, where we specify the generation of rules for transforming words into their ancient spellings. Our approach is evaluated in Section 5, and the last section concludes the paper and gives an outlook on future work.

## 2 Related Work

Rayson et.al. [14] describe a project for dealing with historic spellings of English. They developed a variant detector for English texts from the 16th-19th century. A major difference to our work consists in the fact that German is a highly inflected language, in contrast to English. Thus approaches developed for English can hardly be applied for German.

Previous digitisation projects for the German language (e.g. the Bayrische Staatsbibliothek<sup>4</sup>) employed standard search engines. Some of them are thesaurus-based (in combination with manual indexing), but they do not offer specific support for searching in historic texts. Other approaches use dictionaries for this purpose. However, this

---

<sup>4</sup> <http://www.bsb-muenchen.de/mdz/>

approach covers only the words contained in the dictionary. Furthermore, the time and effort for the manual construction of the word entries is rather high.

We want to overcome this disadvantage with a rule-based approach, in order to be able to cover the complete vocabulary (and thus increase recall). On the other hand, the rules to be developed should be sufficiently precise, for distinguishing between spelling variants of the search term and other words.

The topic addressed in this paper is related to problem of approximate name matching [10]. There names with an incorrect spelling have to be found in a list of names. However, the major difference between the two problems consists in the fact that names usually differ only in their spelling, but not in their pronunciation. In contrast, words from historic texts may also differ in their pronunciation, mainly due to regional dialects [15]. These differences can also have effects on the spelling (see section 1).

The problem studied here is also somewhat related to cross-language information retrieval [12], since in both cases mappings between words are considered. However, our problem can be solved by means of mappings at the grapheme level, while only dictionary-based approaches are suitable for cross-language information retrieval.

### 3 Searching in text collections with non-standard spelling

There are two possible approaches for searching in texts with spelling variants:

1. Stemming at indexing time: This standard information retrieval method requires the set of stemming rules to be known at indexing time. In the case of spelling variants, also rules for the standardisation of the different spellings are necessary. However, the German language is a highly inflected language. Rule-based stemming for contemporary German requires rather complex rule sets, so it would be very difficult to find the inflection rules which map the ancient spelling onto the associated contemporary radical. In our case, no rule set for spelling standardisation exists (and, due to the time- and region-dependence, will probably never become available).
2. Generation of search term variants at retrieval time: For this query expansion again rules for inflections and derivations of words as well as for handling spelling variants are required, but this time in the opposite direction, i.e. we need a mapping

search term  $\rightarrow$  contemporary inflections (or derivations)  $\rightarrow$  spelling variants

This approach is more flexible, as new rule sets can easily be adopted.

Indeed, the first approach would be a lot faster than the second one, because the time-sensitive processing of the transformation happens once when a new collection gets indexed. However, we assume that rule sets for spelling variations will not be fixed for quite a long time and so only the second approach gives us the necessary flexibility.

In the latter approach, we first have to deal with morphological variations, before we can start constructing the rules for spelling variants. For this purpose, we are using a contemporary dictionary<sup>5</sup> containing the full word forms [13]. Thus, when the user enters a search term (in its basic word form), the dictionary yields all inflected forms.

---

<sup>5</sup> <http://wortschatz.uni-leipzig.de/>

This way, we can focus on the second mapping, i.e. the generation of spelling variants of the inflected forms.

By comparing the inflected forms of the dictionary with the word list of our corpus (or using a spell checker), we are getting a list of candidate words in non-standard spelling (some words also may not be contained in the dictionary, though). On the other hand, this method will not be able to detect homographs (ancient spelling that matches a different contemporary word); this issue will be addressed at a later stage of our project. This way, we get a list of candidate words from our corpus. Then, we have to check manually if the words are really in a non-standard spelling, and have to assign the equivalent words in the contemporary standard spelling. After that, we can focus on the second step — the building of new rules.

Even though the studied language is German we also found examples for English [14] where our approach could be employed. E. g. always — alwaies (y → ie), sudden — suddain (e → ai), and publicly — publikely (c → ke).

In the following we list some example rules developed manually for the 19th century German (Table 1).

| Contemp. spelling | 19th century | rules                    |     |
|-------------------|--------------|--------------------------|-----|
| wiedergaben       | widergaben   | wieder → wider<br>ie → i | (1) |
| akzeptieren       | acceptieren  | kz → cc<br>k → c ∧ z → c | (2) |
| überall           | ueberall     | ü → ue                   | (3) |
| seht              | sehet        | t → et                   | (4) |

**Table 1.** Example rules for German

The first example shows two rules at different levels of specialisation. The first rule transforms a prefix whereas the second one only transforms an allograph (see section 5.2). The next example also offers two possibilities. In this case the transformation can consist of one rule or the concatenation of two rules. However it becomes apparent that the precision of the first rule would be much higher than that of the second rule. The third case shows a very common rule for umlauts. The last example contains a very general rule, but it could reach a higher precision if the rule is connected with context information (in this case the end of the word). So not only the transformation itself is important, but also the associated position.

Even though our approach requires a substantial manual effort at the beginning, we expect that only little additional work is required later when the collection is growing continually — due to the fact that we are working at the grapheme level.

## 4 Generation of transformation rules

As described above, our rule generation method starts with a training sample of historic texts, on which we run a spell checker for contemporary German<sup>6</sup>. For all words marked as incorrect spelling, the contemporary word form has to be assigned manually; furthermore, we determine *cf* the number of occurrences of each historic word form. Thus, we have a set  $H$  of triplets  $(\mathbf{a}, \mathbf{h}, cf)$  (contemporary word form  $a$ , full word form  $h$ , collection frequency  $cf$ ).

In the following, we denote a character string  $\mathbf{a}$  also as a sequence  $a_0 \dots a_n$ . Furthermore, if  $n = 0$ , then  $a_0 \dots a_n$  denotes the empty string  $\varepsilon$ , and  $a|b$  denotes the concatenation of strings (for convenience, we don't distinguish between characters and strings of length 1 here). The definitions apply accordingly for a character string  $\mathbf{h}$ . Furthermore, we assume that each word form has a leading and a trailing blank. For lists we use the Prolog-like notation  $[l_1, \dots, l_k]$ , and we also use the notation  $[h|t]$  for splitting a list into head  $h$  and tail  $t$ .

For generating transformation rules, we use the set  $H$  containing the contemporary words and their historic spellings. First, we compare the two words and determine so-called 'rule cores', i.e. the necessary transformations. In a second step, we generate rule candidates that also consider context information from the word  $\mathbf{a}$ . Finally, in the third step, we select the useful rules by pruning the candidate set. For describing the different steps, we are specifying the functions involved — the corresponding algorithms offer various possibilities for optimisations, which are still under development (a straightforward implementation could be achieved by using Prolog or a function-oriented programming language).

### 4.1 Generate rule cores

For the rule cores, we determine the necessary transformations and also identify the corresponding contexts. First, we define a function  $rcg1()$ , which creates a mixed list of transformations and contexts when being called with two words and an empty string as initial value. For example, for  $\mathbf{a} = \text{'unnüt z'}$  and  $\mathbf{b} = \text{'unnuts'}$ ,  $rcg(\mathbf{a}, \mathbf{b}, \text{''})$  would yield  $p = [\text{'unn'}, (\text{'ü'}, \text{'u'}), \text{'t'}, (\text{'z'}, \text{'s'})]$ .

$$rcg1(a_0 \dots a_n, h_1 \dots h_m, p) = \begin{cases} [p] & , \text{ if } n = m = 0 \\ rcg1(a_2 \dots a_n, h_2 \dots h_m, p|a_1) & , \text{ if } a_1 = h_1 \\ [p, (a_1 \dots a_j, h_1 \dots h_l) | rcg1(a_j \dots a_n, h_l \dots h_m, \varepsilon)] & , \text{ if } a_1 \neq h_1 \\ \quad \text{so that } a_{j+1} = h_{l+1} \text{ and } j + m \text{ is minimum} & \end{cases}$$

<sup>6</sup> We are not using the contemporary dictionary for this purpose, since it contains a large number of spelling errors, due to the fact that it was built automatically from large volumes of Web pages.

Given such a list  $L$ , we now generate the rule cores, i.e. the transformations with the left and right contexts, by means of the following recursive function:

$$rcg(L) = \begin{cases} \emptyset & , \text{ if } |L| = 1 \\ rcg(L') \cup \{(l,t,r)\} \text{ with } L = [l,t,r|R] \text{ and } L' = [r|R] & , \text{ otherwise} \end{cases}$$

For our example from above, we would get the following 2-element set of rule cores:  $\{('unn', ('ü', 'u'), 't'), ('t', ('z', 's'), ' ')\}$ .

## 4.2 Generate rule candidates

For each rule core, we want to generate a set of rule candidates, by successively adding left and right context to the left-hand side of a transformation rule. Besides considering the exact characters occurring in the context, we also regard abstractions to the two character classes vowels and consonants denoted by  $V$  and  $C$ , respectively. In addition,  $B$  denotes a blank. Thus, the left context of a transformation has the general syntax  $B?[C/V]^*['a'..'z']^*{}^7$ , and the right context follows the grammar  $['a'..'z']^*[C/V]^*B?$ .

For each element  $(l, (s,d), r)$  from our list of rule cores, we call the function  $rg(l, r, (\epsilon, s, \epsilon, d))$  which is defined as follows:

$$\begin{aligned} rg(l, r, t) &= rgcl(l, r, t) \cup rgcr(l, r, t) \cup rge(l, r, t) \\ rge(l, r, t) &= rgl(l, r, t) \cup rgr(l, r, t) \cup \{r\} \\ \\ rgcl(l_1 \dots l_n, r_1 \dots r_m, (f, s, b, d)) &= \begin{cases} rg(l_1 \dots l_{n-1}, r_1 \dots r_m, (l_n|f, s, b, d)) & , \text{ if } n > 0 \\ \emptyset & , \text{ otherwise} \end{cases} \\ rgcr(l_1 \dots l_n, r_1 \dots r_m, (f, s, b, d)) &= \begin{cases} rg(l_1 \dots l_n, r_2 \dots r_m, (f, s, b|r_1, d)) & , \text{ if } m > 0 \\ \emptyset & , \text{ otherwise} \end{cases} \\ rgl(l_1 \dots l_n, r_1 \dots r_m, (f, s, b, d)) &= \begin{cases} rge(l_1 \dots l_{n-1}, r_1 \dots r_m, (cc(l_n)|f, s, b, d)) & , \text{ if } n > 0 \\ \emptyset & , \text{ otherwise} \end{cases} \\ rgr(l_1 \dots l_n, r_1 \dots r_m, (f, s, b, d)) &= \begin{cases} rge(l_1 \dots l_n, r_2 \dots r_m, (|f, s, b|cc(r_1), d)) & , \text{ if } m > 0 \\ \emptyset & , \text{ otherwise} \end{cases} \end{aligned}$$

Here  $rgcl()$  and  $rgcr()$  generate rules containing literal characters, whereas  $rge()$  produces rules with the generalisations mentioned above.  $rgl()$  generalises the next character of the left context, and  $rgr()$  the next character on the right. In practical applications, we further restrict the number of candidate rules generated by these functions by defining a maximum length for the left and right context to be considered.

<sup>7</sup> According to the notation of regular expressions,  $?$  denotes an occurrence once or not at all and  $*$  denotes an occurrence zero or more times

For our example from above, the following candidate rules are generated (among others:  $(\epsilon, \ddot{u}, \epsilon, u)$ ,  $(n, \ddot{u}, \epsilon, u)$ ,  $(\epsilon, \ddot{u}, t, u)$ ,  $(n, \ddot{u}, t, u)$   $(C, \ddot{u}, \epsilon, u)$ ,  $(\epsilon, \ddot{u}, C, u)$ ,  $(C, \ddot{u}, C, u)$ ).

Given these candidate rules for a contemporary word  $\mathbf{a}$  and its historic form  $\mathbf{h}$ , we generate a tuple  $(\mathbf{a}, \mathbf{h}, cf, (f, s, b, d))$  for each candidate rule; in addition,  $cf$  denotes the collection frequency of  $\mathbf{h}$ . The set of these tuples for all our triplets from  $H$  forms the set of training instances  $E$ .

### 4.3 Rule set pruning

The generation of the final transformation rules can be regarded as a classification task, where we have to distinguish between 'correct' and 'incorrect' rules. The set  $E$  of instances  $e_i = (\mathbf{a}_i, \mathbf{h}_i, cf_i, (f_i, s_i, b_i, d_i)) \in E$  with the rule candidates contains the positive examples. The negative examples consists of the words  $\mathbf{a}$  in  $E$  where a rule can be applied, but has not been generated<sup>8</sup>.

For deriving a good set of transformation rules, we have developed an extension of the PRISM algorithm developed in data mining [4]. PRISM assumes that we have a set of instances to be classified into a set of classes (in our case: correct/incorrect rules). Instances are described by a fixed set of attributes with values from a nominal scale. For each class  $C$ , the algorithm tries to find a set of high-precision rules for identifying the instances belonging to  $C$ .

```

For each class C
  Initialise E to the instance set
  While E contains instances in C
    Create a rule R with an empty left-hand side that predicts C
    Until R is perfect (or there are no more attributes to use) do
      For each attribute A not mentioned in R, and each value v,
        Consider adding the condition A=v to the left-hand side of R
        Select A and v to maximise the accuracy p/t
        (break ties by choosing the condition with the largest p)
      Add A=v to R
    Remove the instances covered by R from E

```

Unfortunately, we cannot apply PRISM directly, for two major reasons:

1. We are not aiming at perfect rules, since this would result in rules which are specific for each contemporary word form — and thus, hardly any words not seen before would be covered by these rules (i.e. we would get a high precision, but very low recall).
2. Instead of a set of attributes with fixed sets of nominal values, we have a possibly infinite set of left-hand sides of rules. For this reason, we are generating rule candidates from examples, whereas PRISM generates rules independent from examples and then tests their quality. In PRISM, adding a condition to a rule results in a more specific rule. In our case, we also have generalisation/specialisation relationships between rule antecedents, which we can exploit for directing the search.

<sup>8</sup> This is only an approximation, but a proper set of negative examples would require a huge manual effort.

Based on these considerations, we have developed the rule pruning algorithm that takes the candidate set  $E$  and outputs a final set  $F$  of rules. (Similar algorithms have been proposed for text categorisation, see e. g. [5], but they do not consider the specialisation hierarchy on rule conditions.) As additional parameters, two cutoff values have to be specified for this algorithm:  $q_{\min}$  denotes the minimum number (of tokens) of correct applications of a rule, and  $p_{\min}$  is the minimum precision of rules to be considered.

Let us assume that we have a Boolean function  $match(r, \mathbf{a})$  which tests if the word  $\mathbf{a}$  satisfies the right-hand side of rule  $r$ . Based on this function, the most expensive step in our algorithm is the search for all word forms where  $match()$  yields true. In order to speed up this process, we first sort the instances by the rule itself; thus, we have to perform the search only once for each rule (and not once per instance). Due to the regular structure of our rules, we can use an access structure like a PAT array [7] for determining all matching word forms.

In the following  $q_i$  denotes the number of positive occurrences of rule  $r$  and  $p_i$  denotes the precision of rule  $r$ .

For each training instance  $e_i$ , let

$$\begin{aligned} E_i &= \{r | r \in E \wedge r = (\mathbf{a}, \mathbf{h}, cf, (f_i, s_i, b_i, d_i))\} \\ m_i &= \sum_{e_j \in E \wedge match(e_j, \mathbf{a}_j)} cf_j \\ q_i &= \sum_{e_j \in E_i} cf_j \\ p_i &= \frac{q_i}{m_i} \end{aligned}$$

Remove all instances  $e_i$  from  $E$  where  $p_i < p_{\min} \vee q_i < q_{\min}$ .

Let  $F = \emptyset$ .

while  $E \neq \emptyset$  do

1. from the instances with the highest  $p$  values, select those with the highest  $q$  values and among those, choose one for which there is no instance in  $E$  with a more general rule. Let  $e_i$  denote this instance.
2.  $F = F \cup (e_i, p_i)$ .
3. remove all instances from  $E$  where  $e_i$  applies: let

$$\begin{aligned} D &= \{e_j | e_j \in E \wedge e_j = (\mathbf{a}_j, \mathbf{h}_j, cf_j, (f_j, s_j, b_j, d_j)) \wedge \\ &\quad \exists e_k \in E \wedge e_k = (\mathbf{a}_j, \mathbf{h}_j, cf_k, (f_i, s_i, b_i, d_k))\} \end{aligned}$$

Then set  $E := E - D$ .

od.

#### 4.4 Rule application

Given the set of probabilistic rules as described above, they can be applied in our search engine. For a contemporary word  $\mathbf{a}$ , we want to generate all historic spellings. Thus, for

any element  $(r_i, p_i) \in F$ , if  $match(r_i, \mathbf{a})$ ,  $r_i$  is applied to  $\mathbf{a}$ , thus yielding the word  $\mathbf{a}_i$ . This way we are generating a set of historic spellings for a single word  $\mathbf{a}$ , by application of single rules. Obviously, these word forms are not all equally precise. Therefore, these words should be assigned weights which reflect the precision of the rules they resulted from [17].

Our retrieval engine considers the precision of rules in the following way:

For a spelling variant  $w$  generated from a search term  $t$ , we interpret  $p$  as the probability  $p = P(t \rightarrow w)$  that  $t$  implies  $w$ . Since our search engine is based on retrieval as uncertain inference, these probabilities can be easily incorporated into the retrieval process (e.g. in the simple case of binary indexing and single-term queries,  $p$  would be the weight of a document containing  $w$ ).

## 5 Evaluation

For evaluation, we compared our new approach with two other methods developed before. Here we first describe the alternative methods, and then we present the experimental results.

### 5.1 Manually-built rules

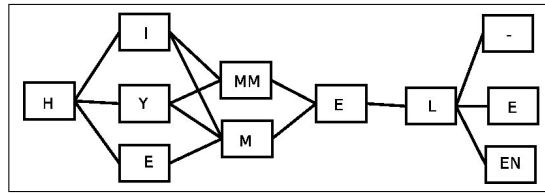
In [11], the manual construction of two rule sets for 19th century German texts is described. For the first rule set, each rule can only be applied at most once at a specific position in a word (e. g.  $\ddot{a} \rightarrow e$ ). In contrast, the rules from the second rule set (e. g.  $aa \rightarrow a$ ) can be applied an arbitrary number of times. Due to the fact that spelling variants often occur because of differences in pronunciation, a part of the rules have been developed by comparing pronunciations. Other rules have been developed by literature research. For each word in question, these rules are applied onto the contemporary and the historic word form. If the results of the transformations are equal, this means that a historic spelling variant has been found.

### 5.2 Variant graph

The approach presented in [2] is similar to phonetic name matching. For each grapheme, the appropriate allographs are generated. For example, the grapheme  $f$  has the allographs  $\{u, v, f, ff, pf\}$ . Given the set of allographs for each grapheme, a variant graph can be build for each input word in contemporary spelling. For each grapheme in the input word this graph contains the corresponding allographs; each path in the graph produces one variant spelling. As an example, Figure 1 shows the variant graph for the word ‘Himmel’ (sky).

### 5.3 Experiments

For a comparative evaluation of the different approaches, we used documents from the Nietzsche collection and other smaller collections containing texts from the 19th century. The manually build rule set had been developed only based on the Nietzsche collection while the rule set for the other two approaches had been developed for the whole



**Fig. 1.** The variant graph for 'himmel'

test collection. Our small collection contains 64290 word tokens, with 11326 different words (types). After feeding the types into the spell checker, followed by manual checking of the marked words, we were left with 717 different words in historic spellings.

Since our new approach requires a training sample, we split the available data such that two thirds of the instances were used as training sample, and the remaining third as test sample.

The recall and precision values are based on collection frequency of the retrieved full word forms. The results of applying the three approaches to the test sample are shown in Table 2.

| Approach        | Precision | Recall |
|-----------------|-----------|--------|
| Manual rules    | 0.53      | 0.09   |
| Variant graph   | 0.48      | 0.69   |
| Automatic rules | 0.45      | 0.88   |

**Table 2.** Recall and precision figures of the three approaches

The manually developed rules from [11] perform poorly, as they achieve a precision of 0.53, but only a recall of 0.09. In addition to the low quality, a major disadvantage of the manual rules is the intellectual effort for their development. Thus, this approach does not seem to be suitable for supporting retrieval of texts in historic spelling, because we did not even get an expected high precision for the available rules.

With the variant graph, we reach a similar precision, but a recall level of 0.69. Roughly speaking, this approach misses one out of three words in historic spelling; this is not a satisfying result - even though only a fraction of all words occur in non-standard spelling. Another drawback of the variant graph method is the large number of spelling variants generated (e. g. 18 in Figure 1), which increases retrieval times substantially.

The automatic generated rules reach a precision of 0.45 and a recall of 0.88. In comparison to the other two approaches, precision is slightly inferior. On the other hand, the recall is 28 % better as that of the variant graph method.

With the automatically generated rules, it is also simple to look at the precision values for single rules. These precision values are based on the collection frequency of full word forms and the false positives the rules are used for. Table 3 shows some frequently used rules with their corresponding precision values.

| Rules  | Context | Frequency | Precision | Examples                   |
|--------|---------|-----------|-----------|----------------------------|
| t → th |         | 116       | 0.67      | Einteilung - Eintheilung   |
| ä → ae | post: C | 42        | 0.98      | Ämter - Aemter             |
| s → ß  |         | 35        | 0.62      | aus - auß                  |
| k → c  |         | 32        | 0.8       | Kollegien - Collegien      |
| ü → ue |         | 20        | 0.69      | Übertragung - Uebertragung |
| ä → ai |         | 18        | 1.0       | souverän - souverain       |

**Table 3.** Frequently used rules

| $p_{\min}$ | 0.0  | 0.1  | 0.2  | 0.3  | 0.4  | 0.5  | 0.6  | 0.7  | 0.8  | 0.9  | 1.0  |
|------------|------|------|------|------|------|------|------|------|------|------|------|
| Recall     | 0.88 | 0.87 | 0.87 | 0.87 | 0.86 | 0.73 | 0.73 | 0.69 | 0.65 | 0.63 | 0.29 |
| Precision  | 0.45 | 0.65 | 0.65 | 0.75 | 0.77 | 0.93 | 0.94 | 0.96 | 0.98 | 0.99 | 1.00 |

**Table 4.** Recall and Precision for different threshold values  $p_{\min}$

We generated a first rule set with the parameters  $p_{\min} = 0.0$  and  $q_{\min} = 1$ . By applying higher thresholds values  $p_{\min}$  in our rule pruning algorithm, we can increase the precision on the test sample. Table 4 shows the corresponding results. A reasonable threshold seems to be 0.4, because recall decreases only slightly from 0.88 to 0.86, while precision decreases from 0.45 to 0.77 and is consequently superior to the precision values of the other two approaches.

Overall, these results the implementation of our new approach show that automatically generated rules outperform previous methods, and that we are on the right track to achieve a high quality in the generation of historic spellings.

## 6 Conclusion and Future Work

In this paper, we have discussed the problem of retrieval in historic texts with non-standard spelling. We have shown that due to the large variations in historic spellings, the standard stemming approach cannot be applied. Instead, historic variants of the search terms have to be generated. By using a contemporary dictionary containing the inflected word forms, we first map the search term onto its inflected forms, and then generate the corresponding historic variants. For this purpose, we have described a machine learning method for generating appropriate transformation rules. Since the rules have probabilistic weights, these weights can be considered in retrieval for weighting the documents matched by search terms generated through weighted rules.

So far, we have only a first version of our algorithm. As an obvious extension of the basic algorithm, we should consider better parameter estimation methods for estimating the precision of rules with a small number of positive examples. The manual analysis of the errors on the test set has shown that a large number of errors is caused by a few words. Thus, these words should be considered as exceptions in the rules — so the rule generation algorithm should be modified accordingly.

For an application of our approach to large corpora, we are also working on the development of an interactive tool for rule generation. Instead of assessing a large number of 'misspelled' words before starting rule generation, the interactive tool would start with a few assessed examples only, generate rule candidates and then ask the user for judging about (a representative sample of) other words where these rules apply.

So far, our method has been applied to German texts only. However, by exchanging the linguistic resources (i.e. the dictionary containing the full word forms) it can be easily applied to other languages.

The module for generating historic variants of search terms will be integrated in a probabilistic engine for the retrieval of historic texts. Only through retrieval experiments with historic collections, we can assess the ultimate quality of the new method presented here.

## 7 Acknowledgements

This work is supported by the DFG (project RSNSR).

## References

- [1] Biella, D., Dyllong, E., Kaiser, H., Luther, W., and Mittmann, T.: Edition électronique de la réception de Nietzsche des années 1865 à 1945." In: Proc. ICHIM03, Paris, 2003.
- [2] Biella, D., Dyllong, E., H., Luther, W., and Pilz, T.: An On-line Literature Research System with Rule-Based Search, In: Proc. of the 4th European Conference on e-Learning (ECEL2005). Amsterdam, 2005.
- [3] Camps, R., Daudé, J.: Improving the efficacy of approximate personal name matching. In: Proc. 8th International Conference on Applications of Natural Language to Information Systems (NLDB 03). 2003. <http://www.lsi.upc.es/dept/techreps/ps/R03-9.ps.gz>.
- [4] Cendrowska, J.: PRISM: An algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4), pp. 349-370. 1987.
- [5] Cohen, W., W., Singer, Y.: Context-sensitive learning methods for text categorization. In *ACM Trans. Inf. Syst.* Vol. 17, No. 2, pp 141-173. ACM Press, New York, NY, USA. 1999.
- [6] De Roux, E.: 19 bibliothèques en Europe signent un manifeste pour contrer le projet de Google. *Le Monde*, Paris, 28.04.2005.
- [7] Frakes, W., B., Baeza-Yates, R., A.: *Information Retrieval: Data Structures & Algorithms* Context-sensitive learning methods for text categorization. Prentice-Hall, 1992. DBLP, <http://dblp.uni-trier.de>.
- [8] Keller, R.: *Die Deutsche Sprache und ihre historische Entwicklung*. Hamburg: Helmut Buske Verlage, 1986.
- [9] Nottelmann, H.: PIRE: An extensible IR engine based on probabilistic Datalog. *ECIR 2005*.
- [10] Pfeifer, U., Poersch, T., and Fuhr, N.: Retrieval Effectiveness of Proper Name Search Methods. *Information Processing and Management* Vol. 32, No. 6, pp. 667-669. 1996.
- [11] Pilz, Th.: *Unschärfe Suche in Textdatenbanken mit nichtstandardisierter Rechtschreibung am Beispiel von Frakturtexten zur Nietzsche-Rezeption*. Staatsexamensarbeit. Universität Duisburg-Essen, 2003.
- [12] Peters, C. (Hrsg.): *Cross-Language Information Retrieval and Evaluation*, Vol. 2069 von *Lecture Notes in Computer Science*, Heidelberg et al. Springer. 2001.

- [13] Quasthoff, U.: Projekt Der Deutsche Wortschatz. In Heyer, G., Wolff, Ch. (eds.) (1998). Linguistisch und neue Medien. In: Proc. from the GLDV-Tagung, 17.-19. März 1997 at Leipzig, Deutscher Universitätsverlag, pp. 93-99, 1998.
- [14] Rayson, P., Archer, D., Smith, N.: VARD versus Word. A comparison of the UCREL variant detector and modern spell checkers on English historical corpora. In proceedings of the Corpus Linguistics 2005 conference. Birmingham, UK. In: Proc. from the Corpus Linguistics Conference Series on-line e-journal, Vol. 1, No. 1., 2005.
- [15] Strunk, J.: Information Retrieval for Languages that lack a fixed orthography. 2003. <http://www.linguistics.ruhr-uni-bochum.de/~strunk/LSreport.pdf>.
- [16] Witten, I., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Morgan Kaufmann Publishers, San Francisco, CA, 2000.
- [17] Zobel, J. and Dart, P.: Phonetic String Matching: Lessons from Information Retrieval. 1996. In Frei, H.-P.; Harman, D.; Schäuble, P.; Wilkinson, R. (eds.): Proc. 19th Inter. Conf. on Research and Development in Information Retrieval (SIGIR), New York, 1996, pp. 166-172.