Unsupervised Learning of Edit Distance Weights for Retrieving Historical Spelling Variations

Andreas W. Hauser; Klaus U. Schulz
CIS
University of Munich
andy@splashground.de
schulz@cis.uni-muenchen.de

Abstract

While todays orthography is very strict and seldom changes, this has not always been true. In historical texts spelling of words often not only varies from todays but in some periods even varies from use to use in a single text. Information retrieval on historical corpora can deal with these variations using fuzzy matching techniques based on Levenshtein-Distance using stochastic weights. In particular by using the noisy channel model of (3) and the simple algorithm they give. The algorithm, they use for spell checking, adapted to the problem of information retrieval of historical words, with queries in modern spelling, uses stochastic weights, learned from training pairs of modern and historical spelling. Using these weights shows an improvement over standard Levenshtein-Distance in the F-Score. The preparation of the training pairs usually depends on manual work. To avoid this work we devised an unsupervised algorithm for obtaining the training pairs.

Keywords: approximate search, fuzzy matching, information retrieval, historical language, unsupervised, algorithm, noisy channel

1 Introduction

Information retrieval on corpora or databases containing historical language is difficult because of two spelling related problems. The first is that the spelling rules have changed over time and the second problem is that the rules were not as strict or not followed as strictly. Therefore a word might have different spellings at different times and even different spellings at one time. We will refer to these as variations in the rest of this text. These variations are sometimes regular but more often patterns are found, where either the underlying rules have not been identified or might not exist or are so very particular that they cannot be generalized.

German for example has been standardized at the beginning of the 20^{th} century and since has been updated in minor ways. Before that, from about the 17^{th} century on, there was a period where the written language had consolidated, so that a standardization was possible at all. Even earlier, from the 14^{th} century to the 17^{th} century, in the Early New High German period spelling was not standardized and even words in the same text or sentence had no need to be written exactly the same. See (2) for more on the history of the German language.

Common patterns of spelling variation in Early New High German are for example *i>y, t>th, e>, d$>t$, ^u> ^v, äu>eu*. Where the string before > is deleted and the string after it inserted. Deletions and insertions lack the respective string. ^ is used to mark beginning of string and $ for the end, as in *arbeiten* → *arbeyten* (engl. 'to work'). We find for example in a single text the following frequent spelling variations for modern German *Eulenspiegel*, the name of the main character, written as *Ulenspiegel Ulnspiegel* and less frequently *Ulenspieg Ulenspigel Ulnspeigel Ulnspiegl Ulnspigel Ulspiegel vlnspiegel*. A more elaborate overview can be found in (8).

Similar variations can be also be found in Middle English, English written from the 11^{th} to the 15^{th} century. Her an excerpt from an electronic version of an edition, (4), of "The Canterbury tales" from about 1400:

But nathelees, whil I have tyme and space,
Er that I ferther in this tale pace,
Me thynketh it acordaunt to resoun
To telle yow al the condicioun
Of ech of hem, so as it semed me,
And whiche they weren, and of what degree,
And eek in what array that they were inne;
And at a knyght than wol I first bigynne.

This fragment shows the general characteristics of these old texts where most of the words are close enough to todays but the spelling and some morphology is changed. As hinted by this example not all of the spelling is randomly changed. Some patterns are easy to see like *y* for *i* as in *knyght* and *bigynne*. But hard rules for all of the variations do not seem so easy to produce.

We found that less than 35% of the historical tokens equal the modern translation to the character, but estimate that around 80% differ only in spelling for the Munich Corpus. (6) give an overview of the variation rates by year encountered in historical German from about 1480 to 1880 with rates ranging from about 5% to about 50%.

Information retrieval on such texts, without handling the variations, results in poor recall because

of the high ratio of these variations. Approximate search techniques can help to improve the recall. Using stochastic weights for the edit distance costs provides better results than standard Levenshtein-Distance but they depend on the availability of training data, which often is not available. We show a simple learning algorithm that avoids the need for training data using only a lexicon and a corpus, showing better results then simple Levenshtein-Distance and worse than the one with training data.

The rest of this paper is structured in the following way. In Section 2 we introduce the stochastic substring to substring model we use for fuzzy matching and present a simple learning algorithm in Section 3, first based on trainings data, then a new one without the need for training data. Next we show our results for an Information Retrieval task with Early New High German in Section 4. After giving an overview of the applications in Section 5 that could benefit from these techniques, we draw a conclusion.

2 Advanced Fuzzy Matching

The roots of fuzzy matching are based on edit distances with equal weights for equal operations, which are limited to character to character operations, e.g. (5). Since all weights are equal problems arise when search terms are matched against lexicon and the lexicon is big or when the relevant terms in the lexicon are more than one edit operation away because of the number of false positives found this way.

In cases like historical variations where two or more edit operations are very common but certain edit operations are much more likely to happen, like an insertion of an h after a t or a y instead of an i, individuell weights can help to improve the retrieval quality.

Stochastic models, as in (1), which add distinct weights for each operation and characters involved based on their frequency in the training data, can be used. Algorithms, based on their work, use prepared pairs from words that are considered close. They learn that certain character transitions (edit operations) are cheaper, based on the frequencies these transitions have in the pairs. This makes it possible to model that *ein* and *eyn* are closer than *ein* and *ekn* by having lower weights for $i \rightarrow y$ than $i \rightarrow k$ transitions.

But while this improves, e.g. the F-Score, it is still insensitive to the context of the character transitions, thus giving equal weight to an $i \rightarrow y$ transitions any where in the strings. But it might make a big difference what is around the character, e.g. $ei \rightarrow ay$ might be close, especially it being a German diphthong, but $hi \rightarrow ty$ might be not. (3) introduced a noisy channel model that uses generic substring-to-substring transitions instead of character-to-character transitions, enabling us to catch exactly that.

With that model one can get the following transitions from the training pairs (*kein*, *kayn*) when one allows the substrings in the transitions to be three to three at maximum:

$k \rightarrow k$, $e \rightarrow a$, $i \rightarrow y$, $n \rightarrow n$
$ke \rightarrow ke$, $ei \rightarrow ay$, $in \rightarrow yn$,
$kei \rightarrow kay$, $ein \rightarrow ayn$

They also make their weights depend on the position in the string (start, end, none of both). For our results we modelled the start of the strings by prepending ^ and the end by appending *$*, loosing one character in our n-grams but simplifying the algorithm. Leading to the additional transitions for the above example:

$\hat{} \rightarrow \hat{}$, $\hat{}k \rightarrow \hat{}k$, $\hat{}ke \rightarrow \hat{}ka$,
$in\$ \rightarrow yn\$$, $n\$ \rightarrow n\$$, $\$ \rightarrow \$$

To account for this the ordinary Levenshtein matching algorithm must be modified to account for the weighted substring to substring edit operations. A straight forward way based on the matrix based Levenshtein implementations might use a lookup table, e.g. a hash map, for the weights. To support the substring to substring operations it needs to lookup the weights for all combinations of transitions possible at each cell in the matrix.

3 A simple unsupervised algorithm for learning edit distance weights

Supervised Algorithm (3) give a simple supervised learning algorithm, that works on pairs of words which are close. They use an ordinary Levenshtein-Distance to convert one word to the other recording the conversions, where conversions consist of a from part and a to part, where the from part is converted to the to part. That can be achieved by using a second matrix which records the sum of the conversions instead of the sum of the costs. Other techniques would be using Objects or Structs in the first matrix to record both and/or noting only the conversion to reach the current coordinate and a pointer to originating coordinate. Then the conversions and their combinations with surrounding conversions are counted. The weights are then computed by relating the count of the conversion to all other conversions with the same from part.

```
Map frequencies = ()
Map trainingPairs = (("ein", "eyn"),
                     ("zwei", "zwey"),
                   ("läuft", "leuft"),
                                 ...)

For each (word1, word2) in trainingPairs
  List conversions
  conversions = converter(word1, word2)
  recordConversions(conversions)
  computeWeights()

recordConversions(conversions):
  For i to conversions.size
    For j to maxSubstringLength
      For k to i + j
        from += conversion[k].from
        to   += conversion[k].to
      frequencies(from,to) += 1
      frequencies(from) += 1

computeWeights:
Map weights = ()
```

```
For each (from,to) in frequencies
  weights(from,to) = - Math.log(
                       frequencies(from,to) /
                       frequencies(from)))
```

Starting with a list of training pairs, modern spelling and historical spelling, a Levenshtein-Distance based *converter* function is used to transform each of the query words, the first element in the training pair, to the variation, the second element, recording the needed conversions. At this point the conversions are only character-to-character transformations. To obtain substring-to-substring transformations, combinations of the single character ones of up to the maximum length of the substrings, which was chosen, and the frequencies recorded.

Unsupervised Algorithm: The unsupervised algorithm tries to obtain the training pairs automatically, given a historical corpus and a modern lexicon, with an approximate search in the lexicon, instead of using prepared ones. It continuing very much like the supervised algorithm.

```
Map frequencies = {}
Map trainingPairs = ()
List lexicon = ("ein", "eins", "zwei",
               "läuft", "laufen", ... )
List corpus = ("eyn", "zwey", "leuft")
n = 1
maxCost = 1

For each word1 in corpus
  Map temp = ()
  For each word2 in lexicon
    distance = converter1(word1, word2)
    if distance < maxCost
      temp.add((word1, word2))
  If temp.size =< n
    For (word1, word2) in temp
      List conversions
      conversions = converter2(word2, word1)
      recordConversions(conversion)

computeWeights() # see supervised
```

First each word from the (historical) set or corpus is looked up in the lexicon. If more than zero matches within Levenshtein-Distance *maxCost* are found, but not more then n, these are used in reversed order as training pair. Using only those candidates that match only one word in the lexicon, guards against using pairs of unrelated words for learning that are within distance one.

We have chosen one in our experiments for *maxCost* and n.

4 Information Retrieval Experiment

In the information retrieval experiment a search engine for historical word forms from the Early New High German period is modeled. Modern German is used as query terms. Fuzzy matching is used to improve precision and recall because of the variations in spelling, we find in historical texts.

The groundtruth data for the evaluation and the training data for the supervised algorithm was obtained from the Munich Corpus[1] of Early New High German's database. The Munich Corpus is a collection of texts from the Early New High German period from the 14^{th} to the 17^{th} century. It includes a database containing, besides others, the following fields for each token: *historical token, modern translation, historical lexeme, modern lexeme.*

We compare the unsupervised algorithm for fuzzy matchingwith the supervised version, both described in Section 3, and the ordinary Levenshtein-Distance as described in (9).

4.1 Using the Supervised Algorithm

The Munich Corpus for Early New High German comes with a database containing modern German words mapped to corresponding Early New High German words. This includes mappings to words that are not in use anymore or have changed their meaning. Therefore it does not contain mappings between different spelling only and when using the mappings to obtain the weights for our distance function in the way described in Section 3. These will degrade the quality of the weights a little when using the mappings to learn the weights, unless the learning is constraint to mappings that stay in a certain edit distance. In the evaluation of the approximate search it will lead to a subset of the relevant words that are practically not found if a certain precision is to be retained. One probably would need to use a special thesaurus before using approximate search for these cases.

Because of the high variations in the historical language one modern spelling is usually mapped to more than one historical spelling. We therefore test the quality of our retrieval by using each modern spelling as query, ranking all the retrieved historical forms by their edit distance. Then we compute for each distance the precision and recall in respect to the groundtruth in the Munich Corpus' database.

4.2 Using the Unsupervised Algorithm

As described in Section 3 it is also possible to learn weights completely automatic. The unsupervised algorithm does not depend on training data but does depend on the availability of a, in this case modern German, lexicon.

The weights are learned by looking up the historical words, that are to be retrieved, in the lexicon. If only a single match is found, the pair is then used as training data.

We tested the following lexica for obtaining the weights automatically:

[1] The Munich Corpus of Early New High German is still unpublished but a demo of the database application is available at http://demo.fruehneuhochdeutsch.is.guad.de/. It will be published later this year by the Institut für Deutsche Philologie at the Ludwig-Maximilians University of Munich. We will refer to it as Munich Corpus.

Perfect Lexica The sets of modern entries are used as perfect lexica as they contain all the necessary query terms and not more. It contains the about 5000 New High German entries from the Munich Corpus' database.

Deutsches Wörterbuch (DWB) Keywords The Deutsches Wörterbuch, (7), is a diachronic dictionary for New High German, which was started in 1838 and finished in 1960. While it tries to cover all of the New High German language, its focus is from the 15^{th} to the 20^{th} century. It is a standard resource for German linguists and even though it includes archaic words out of use today, a linguist might very well use these words in a query, especially after having looked them up in the DWB. We used the about 300.000 keywords as lexicon for our experiments.

Standard System Lexicon (german+ngerman) A combined lexicon was obtained from the "german" and "ngerman" standard word lists on our Linux system, which are available online[2]. Like the DWB keywords lexicon it also contains about 300.000 words, but only modern words in use today.

For one experiment we used the fields *modern lexeme* and *historical lexeme* from the Munich Corpus's database, for the other the fields *modern _lexeme* and *historical token*. For both the set of of modern lexemes are used as query terms, the corresponding historical entries in the database were considered relevant for the query, all others irrelevant.

The improvements with substring to substring weights learned by the supervised and the unsupervised algorithm over the ordinary Levenshtein-Distance with standard weights of one can be seen in the figure on Page 5. Below it is a table showing the maximum in F-Score reached with each method and lexicon.

5 Applications

The application of advanced fuzzy matching in the historical language context is various.

Information Retrieval More and more older books are digitized, the older the more spelling variations are to be found, while most queries will be in todays spelling.

OCR / Typing post-processing Digitizing older books, the post-processing step needs to be able to handle the spelling variations of the original text, distinguishing between spelling variation and recognition error or typo.

Linguistic research Finding and describing the spelling changes is an ongoing linguistic research topic. While the improvements in information retrieval gives researchers better support in their daily work, the weights found can statistically support theories about the variations too.

[2] http://www.coding-zone.com/words-1.0.tar.gz

Automatic normalization The variations could be automatically normalized to or tagged with todays spelling such that contemporary readers can comfortably read the old texts or to enrich it with this information.

Morphology While morphology extraction of current language seems possible, with historical language the variations make the current techniques less usable. Using fuzzy matching techniques combined with these techniques make them usable again.

6 Conclusion

Information retrieval on written historical languages, like Early New High German or Middle English, is difficult, because spelling of words differs from todays. Variations in spelling of historical words, caused by less strict orthographical rules, makes it even more difficult. Fuzzy matching can be used to handle these variations.

We showed that the advanced fuzzy matching techniques, using weighted substring to substring edit operations, invented for spell checking, are much better suited in dealing with historical spelling variations than simpler techniques without weights, as our information retrieval experiment on the database of the Munich Corpus showed.

The simple unsupervised learning algorithm for obtaining the weights, we introduced, that can be used in the absence of training pairs raised the F-Score from 0.569 to 0.662 for the retrieval on historical lexemes and from 0.420 0.484 on historical tokens only using a modern lexicon. Using a perfect lexicon containing only the modern spellings from the training pairs achieved an F-Score of 0.674 for the lexemes and 0.511 for the tokens. The best results though were obtained with weights derived from training pairs, which gave an F-Score of 0.710.

A part of the relevant words could not be retrieved with fuzzy matching without giving up precision. These should probably be handled with a special thesaurus, as many of them are completely different words, not just spelling variations.

References

[1] L. R. Bahl and F. Jelinek. Decoding for channels with insertions, deletions and substitutions with applications to speech recognition. *IEEE Transformations on Information Theory*, 21(4):404–411, 1975.

[2] W. Besch, A. Betten, O. Reichmann, and S. Sonderegger, editors. *Sprachgeschichte*. 4 Volumes. Walter De Gruyter, second edition, 1998-2004.

[3] E. Brill and R. C. Moore. An improved error model for noisy channel spelling correction. In *ACL '00: Proceedings of the 38th Annual Meeting on Association for Computational Linguistics*, pages 286–293, Morristown, NJ, USA, 2000. Association for Computational Linguistics.

[4] G. Chaucer. *The works of Geoffrey Chaucer*. Houghton Mifflin, 2nd edition, 1958.

[5] F. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, 1964.

[6] A. Ernst-Gerlach and T. Pilz. Search methods for documents in non-standard spelling. Talk at the Workshop on Historical Text Mining, Lancaster, U.K., July 2006.

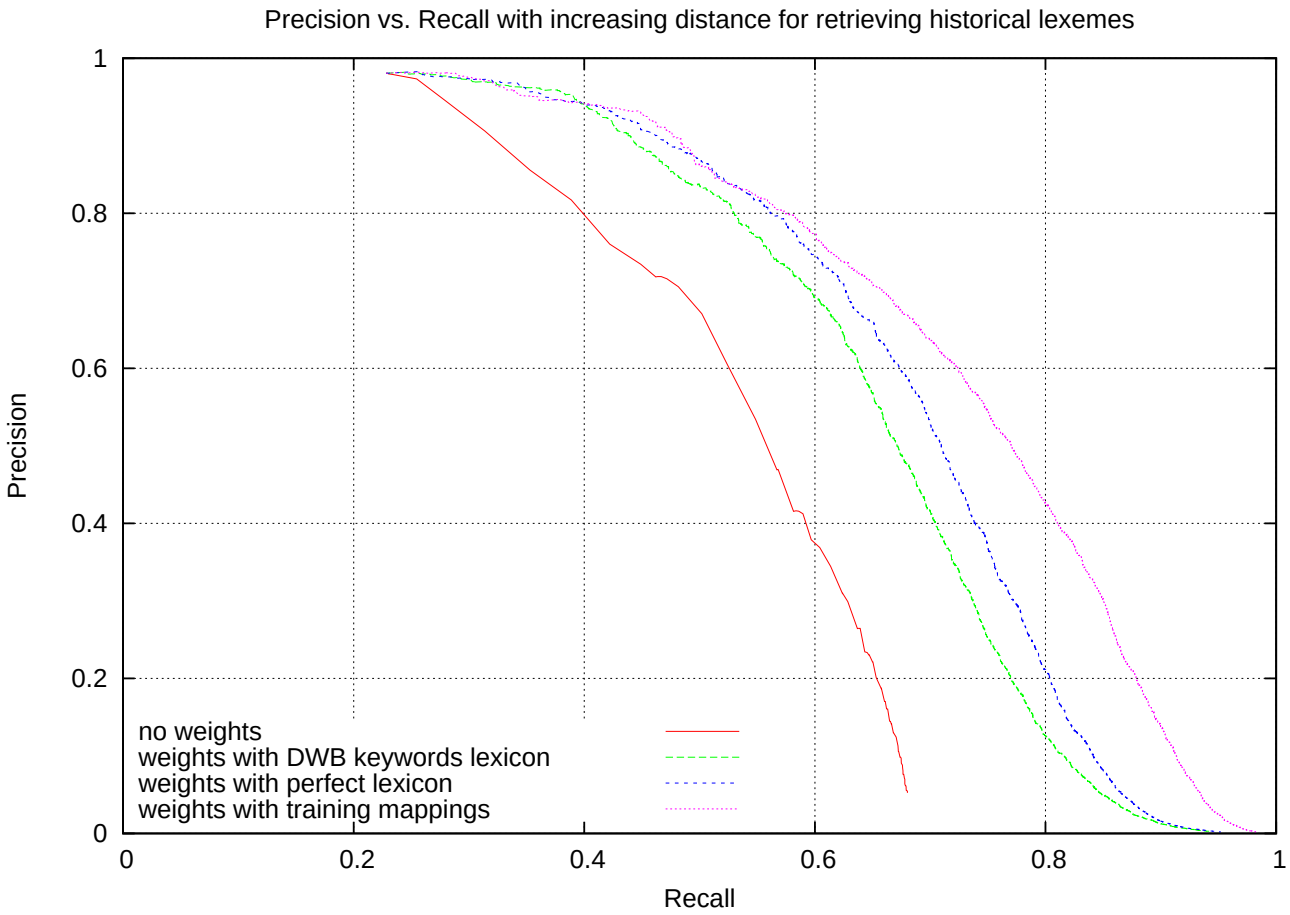

Fig. 1: *Improvements in information retrieval on the Munich Corpus's database, using weights from different sources for substring to substring edit operations to retrieve historical lexemes with modern lexemes as queries. From left to right fuzzy matching using standard Levenshtein weights, weights obtained with training mappings, weights obtained unsupervised using DWB keyword lexicon, weights obtained unsupervised with perfect lexicon.*

Table 1: *Information Retrieval F-Scores*

method / lexicon for weights	max. F-Score for retrieval of hist. lexemes	hist. tokens
non fuzzy string matching	0.337	0.201
standard Levenshtein-Distance	0.569	0.420
DWB keywords lexicon	0.648	0.479
german+ngerman lexicon	0.662	0.484
perfect lexicon	0.674	0.511
training pairs	0.712	0.603

Query Terms from modern lexemes, relevant are the respective **historical lexemes** in one case and **historical tokens** in the other. Weights are computed from pairs of words from the lexicon and historical forms.

[7] J. Grimm and W. Grimm. *Deutsches Wörterbuch. Elektronische Ausgabe der Erstbearbeitung.* Zweitausendeins, 2004.

[8] A. Hauser, M. Heller, E. Leiss, K. U. Schulz, and C. Wanzeck. Information access to historical documents from the early new high german period. In *Proceedings of IJCAI-07 Workshop on Analytics for Noisy Unstructured Text Data (AND-07)*, pages pp. 147 – 154, 2007.

[9] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Sov. Phys. Dokl.*, 1966.