Automatic Prediction of Cognate Orthography Using Support Vector Machines

Andrea Mulloni
Research Group in Computational Linguistics
HLSS, University of Wolverhampton
MB114 Stafford Street, Wolverhampton, WV1 1SB, United Kingdom
andrea2@wlv.ac.uk

Abstract

This paper describes an algorithm to automatically generate a list of cognates in a target language by means of Support Vector Machines. While Levenshtein distance was used to align the training file, no knowledge repository other than an initial list of cognates used for training purposes was input into the algorithm. Evaluation was set up in a cognate production scenario which mimed a real-life situation where no word lists were available in the target language, delivering the ideal environment to test the feasibility of a more ambitious project that will involve language portability. An overall improvement of 50.58% over the baseline showed promising horizons.

1 Introduction

Cognates are words that have similar spelling and meaning across different languages. They account for a considerable portion of technical lexicons, and they found application in several NLP domains. Some major applications fields include relevant areas such as bilingual terminology compilation and statistical machine translation.

So far algorithms for cognate recognition have been focussing predominantly on the detection of cognate words in a text, e.g. (Kondrak and Dorr 2004). Sometimes, though, the detection of cognates in free-flowing text is rather impractical: being able to predict the possible translation in the target language would optimize algorithms that make extensive use of the Web or very large corpora, since there would be no need to scan the whole data each time in order to find the correspondent item. The proposed approach aims to look at the same problem from a totally different perspective, that is to produce an information repository about the target language that could then be exploited in order to predict how the orthography of a "possible" cognate in the target language should look like. This is necessary when no plain word list is available in the target language or the list is incomplete. The proposed algorithm merges for the first time two otherwise well-known methods, adopting a specific tagger implementation which suggests new areas of application for this tool. Furthermore, once language portability will be in place, the cognate generation exercise will allow to reformulate the recognition exercise as well, which is indeed a more straightforward one. The algorithm described in this paper is based on the assumption that linguistic mappings show some kind of regularity and that they can be exploited in order to draw a net of implicit rules by means of a machine learning approach.

Section 2 deals with previous work done on the field of cognate recognition, while Section 3 describes in detail the algorithm used for this study. An evaluation scenario will be drawn in Section 4, while Section 5 will outline the directions we intend to take in the next months.

2 Previous Work

The identification of cognates is a quite challenging NLP task. The most renowned approach to cognate recognition is to use spelling similarities between the two words involved. The most important contribution to this methodology has been given by Levenshtein (1965), who calculated the changes needed in order to transform one word into another by applying four different edit operations – match, substitution, insertion and deletion – which became known under the name of edit distance (ED). A good case in point of a practical application of ED is represented by the studies in the field of lexicon acquisition from comparable corpora carried out by Koehn and Knight (2002) – who expand a list of English-German cognate words by applying well-established transformation rules (e.g. substitution of *k* or *z* by *c* and of *–tät* by *–ty*, as in German *Elektizität* – English *electricity*) – as well as those that focused on word alignment in parallel corpora (e.g. Melamed (2001) and Simard et al. (1999)). Furthermore, Laviosa (2001) showed that cognates can be extremely helpful in translation studies, too.

Among others, ED was extensively used also by Mann and Yarowsky (2001), who try to induce translation lexicons between cross-family languages via third languages. Lexicons are then expanded to intra-family languages by means of cognate pairs and cognate distance. Related techniques include a method developed by Danielsson and Mühlenbock (2000), who associate two words by calculating the number of matching consonants, allowing for one mismatched character. A quite interesting spin-off was analysed by Kondrak (2004), who first highlighted the importance of genetic cognates by comparing the phonetic similarity of lexemes with the semantic similarity of the glosses.

A general overview of the most important statistical techniques currently used for cognate detection purposes was delivered by Inkpen et al. (2005), who addressed the problem of automatic classification of word pairs as cognates or false friends and analysed the impact of applying different features through machine learning techniques. In her paper, she also proposed a method to automatically distinguish between cognates and false friends, while examining the performance of seven different machine learning classifiers.

Further applications of ED include Mulloni and Pekar (2006), who designed an algorithm based on normalized edit distance aiming to automatically extract translation rules, for then applying them to the original cognate list in order to expand it, and Brew and McKelvie (1996), who used approximate string matching in order to align sentences and extract lexicographically interesting word-word pairs from multilingual corpora.

Finally, it is worth mentioning that the work done on automatic named entity transliteration often crosses paths with the research on cognate recognition. One good pointer leads to Kashani et al. (2006), who used a three-phase algorithm based on HMM to solve the transliteration problem between Arabic and English.

All the methodologies described above showed good potential, each one in its own way. This paper aims to merge some successful ideas together, as well as providing an independent and flexible framework that could be applied to different scenarios.

3 Proposed Approach

When approaching the algorithm design phase, we were faced with two major decisions: firstly, we had to decide which kind of machine learning (ML) approach should be used to gather the necessary information, secondly we needed to determine how to exploit the knowledge base gathered in the most appropriate and productive way. As it turned out, the whole work ended up to revolve around the intuition that a simple tagger could lead to quite interesting results, if only we could scale down from sentence level to word level, that is to produce a tag for single letters instead of whole words. In other words, we wanted to exploit the analogy between PoS tagging and cognate prediction: given a sequence of symbols – i.e. source language unigrams – and tags aligned with them – i.e. target language n-grams –, we aim to predict tags for more symbols. Thereby the context provided by the neighbors of a symbol and the previous tags are used as evidence to decide its tag. After an extensive evaluation of the major ML-based taggers available, we decided to opt for SVMTool, a generator of sequential taggers based on Support Vector Machines developed by Gimenez and Marquez (2004). In fact, various experiments carried out on similar software showed that SVMTool was the most suitable one for the type of data being examined, mainly because of its flexible approach to our input file. Also, SVMTool allows to define context by providing an adjustable sliding window for the extraction of features. Once the model was trained, we went on to create the most orthographically probable cognate in the target language. The following sections exemplify the cognate creation algorithm, the learning step and the exploitation of the information gathered.

3.1 Cognate Creation Algorithm

Figure 1 shows the cognate creation algorithm in detail.

```
Input: C1, a list of English-German cognate pairs
    {L1,L2}; C2, a test file of cognates in L1
Output: AL, a list of artificially constructed
        cognates in the target language
1     for c in C1 do:
2         determine the edit operations to arrive
          from L1 to L2
3         use the edit operations to produce a
          formatted training file for the SVM tagger
4     end
5     Learn orthographic mappings between L1
      and L2 (L1 unigram = instance, L2 n-gram =
      category)
6     Align all words of the test file vertically in a
      letter-by-letter fashion (unigram = instance)
7     Tag the test file with the SVM tagger
8     Group the tagger output into words and
      produce a list of cognate pairs
```

Figure 1. The cognate creation algorithm.

Determination of the Edit Operations

The algorithm takes as input two distinct cognate lists, one for training and one for testing purposes. It is important to note that the input languages need to share the same alphabet, since the algorithm is currently still depending on edit distance. Future developments will allow for language portability, which is already matter of study. The first sub-step (Figure 1, Line 2) deals with the determination of the edit operations and its association with the cognate pair, as shown in Figure 2. The four options provided by edit distance, as described by Levenshtein (1965), are Match, Substitution, Insertion and Deletion.

```
toilet/toilette
t    |o    |i    |l    |e    |t    |  |
t    |o    |i    |l    |e    |t    |t |e
MATCH|MATCH|MATCH|MATCH|MATCH|MATCH|INS|INS

tractor/traktor
t    |r    |a    |c    |t    |o    |r
t    |r    |a    |k    |t    |o    |r
MATCH|MATCH|MATCH|SUBST|MATCH|MATCH|MATCH

absolute/absolut
a    |b    |s    |o    |l    |u    |t    |e
a    |b    |s    |o    |l    |u    |t    |
MATCH|MATCH|MATCH|MATCH|MATCH|MATCH|MATCH|DEL
```

Figure 2. Edit operation association

Preparation of the Training File

This sub-step (Figure 1, Line 3) turned out to be the most challenging task, since we needed to produce the input file that offered the best layout possible for the machine learning module. We first tried to insert several empty slots between letters in the source language file, so that we could cope with maximally two subsequent insertions. While all words are in lower case, we identified the spaces with a capital X, which would have allowed us to subsequently discard it without running the risk to delete useful letters in the last step of the algorithm. The choice of manipulating the source language file was supported by the fact that we were aiming to limit the features of the ML module to 27 at most, that is the letters of the alphabet from "a" to "z" plus the upper case "X" meaning blank. Nonetheless, we soon realized that the space feature outweighed all other features and biased the output towards shorter words. Also, the input word was so interspersed that it did not allow the learning machine to recognize recurrent patterns. Further empirical activity showed that far better results could be achieved by sticking to the original letter sequence in the source word and allow for an indefinite number of feature to be learned. This was implemented by grouping letters on the basis of their edit operation relation to the source language. Figure 3 exemplifies a typical situation where insertions and deletions are catered for.

```
START START         START START
a a                 m m
b b                 a a
i i                 c k
o o                 r ro
g g                 o e
e e                 e e
n n                 c k
e e                 o o
t t                 n n
i i                 o o
c X                 m m
a X                 i is
l s                 c ch
l c                 . END
y h
. END
```

Figure 3. Layout of the training entries macroeconomic/makrooekonomisch and abiogenetically/abiogenetisch, showing insertions and deletions

As shown in Figure 3, German diacritics have been substituted by their extended version – i.e. "ö" as been rendered as "oe": this was due to the inability of SVMTool to cope with diacritics. Figure 3 also shows how insertions and deletions were treated. This design choice caused a non-foreseeable number of features to be learned by the ML module. While apparently a negative issue that could cause data to be too sparse to be relevant, we trusted our intuition that the feature growing graph would just flat out after an initial spike, that is the number of insertion edits would not produce an explosion of source/target n-gram equivalents, but only a short expansion to the original list of mapping pairings. This proved to be correct by the evaluation phase described below.

Learning Mappings Across Languages

Once the preliminary steps had been taken care of, the training file was passed on to SVMTlearn, the learning module of SVMTool. At this point the focus switches over to the tool itself, which learns regular patterns using Support Vector Machines and then uses the information gathered to tag any possible list of words (Figure 1, Line 5). The tool chooses automatically the best scoring tag, but – as a matter of fact – it calculates up to 10 possible alternatives for each letter and ranks them by probability scores: in the current paper the reported results were based on the best scoring "tag", but the algorithm can be easily modified in order to accommodate the outcome of the combination of all 10 scores. As it will be shown later in Section 4, this is potentially of great interest if we intend to work in a cognate creation scenario.

As far the last three steps of the algorithm are concerned, they are closely related to the practical implementation of our methodology, hence they will be described extensively in Section 4.

4 Evaluation

In order to evaluate the cognate creation algorithm, we decided to set up a specific evaluation scenario where possible cognates needed to be identified but no word list to choose from existed in the target language. Specifically, we were interested in producing the correct word in the target language, starting from a list of possible cognates in the source language. An alternative evaluation setting could have been based on a scenario which included a scrambling and matching routine, but after the good results showed by Mulloni and Pekar (2006), we thought that yet a different environment would have offered more insight into the field. Also, we wanted to evaluate the actual strength of our approach, in order to decide if future work should be heading this way.

4.1 Data

The method was evaluated on an English-German cognate list including 2105 entries. Since we wanted to keep as much data available for testing as possible, we decided to split the list in 80% training (1683 entries) and 20% (422 entries) testing.

4.2 Task Description

The list used for training/testing purposes included cognates only. Therefore, the optimal outcome would have been a word in the target language that perfectly matched the cognate of the corresponding source language word in the original file. The task was therefore a quite straightforward one: train the SVM tagger using the training data file and – starting from a list of words in the source language (English) – produce a word in the target language (German) that looked as close as possible to the original cognate word. Also, we counted all occurrences where no changes across languages took place – i.e. the target word was spelled in the very same way as the source word – and we set this number as a baseline for the assessment of our results.

Preparation of the Training and Test Files

The training file was formatted as described in Section 3.1. In addition to that, the training and test files featured a START/START delimiter at the beginning of the word and ./END delimiter at the end of it (Figure 1, Line 6).

Learning Parameters

Once formatting was done, the training file was passed on to SVMTlearn. Notably, SVMTool comes with a standard configuration: for the purpose of this exercise we decided to keep most of the standard default parameters, while tuning only the settings related to the definition of the feature set. Also, because of the choices made during the design of the training file – i.e. to stick to a strict linear layout in the *L1* word – we felt that a rather small context window of 5 with the core position set to 2 – that is, considering a context of 2 features before and 2 features after the feature currently examined – could offer a good trade-off between accuracy and acceptable working times. Altogether 185 features were learnt, which confirmed the intuition mentioned in Section 3.1. Furthermore, when considering the feature definition, we decided to stick to unigrams, bigrams and trigrams, even if up to five-grams were obviously possible. Notably, the configuration file pictured below shows how a Model 0 and a global left-right-left tagging option were applied. Both choices were made after an extensive empirical observation of several model/direction combinations. This file is highly configurable and offers a vast range of possible combinations. Future activities will concentrate to a greater extent on the experimentations of other possible configuration scenarios in order to find the tuning that performs best. Gimenez and Marquez (2004) offer a detailed description of the models and all available options, as well as a general introduction to the use of SVMtool, while Figure 4 shows the feature set used to learn mappings from a list of English/German cognate pairs.

```
#ambiguous-right [default]
A0k = w(-2) w(-1) w(0) w(1) w(2) w(-2,-1)
w(-1,0) w(0,1) w(1,2) w(-1,1) w(-2,2)
w(-2,1) w(-1,2) w(-2,0) w(0,2) w(-2,-1,0)
w(-2,-1,1) w(-2,-1,2) w(-2,0,1) w(-2,0,2)
w(-1,0,1) w(-1,0,2) w(-1,1,2) w(0,1,2) p(-2)
p(-1) p(0) p(1) p(2) p(-2,-1) p(-1,0) p(0,1)
p(1,2) p(-1,1) p(-2,2) p(-2,1) p(-1,2)
p(-2,0) p(0,2) p(-2,-1,0) p(-2,-1,1)
p(-2,-1,2) p(-2,0,1) p(-2,0,2) p(-1,0,1)
p(-1,0,2) p(-1,1,2) p(0,1,2) k(0) k(1) k(2)
m(0) m(1) m(2)
```

Figure 4. Feature set for known words (A0k). The same feature set is used for unknown words (A0u), as well.

Tagging of the Test File and Cognate Generation

Following the learning step, a tagging routine was invoked, which produced the best scoring output for every single line – i.e. letter or word boundary – of the test file, which now looked very similar to the file we used for training (Figure 1, Line 7). At this stage, we grouped test instances together to form words and associated each *L1* word with its newly generated counterpart in *L2* (Figure 1, Line 8).

4.3 Results

The generated words were then compared with the words included in the original cognate file.

When evaluating the results we decided to split the data into three classes, rather than two: "Yes" (correct), "No" (incorrect) and "Very Close". The reason why we chose to add an extra class was that when analysing the data we noticed that many important mappings were correctly detected, but the word was still not perfect because of minor orthographic discrepancies that the tagging module did get right in a different entry. In such cases we felt that more training data would have produced a stronger association score that could have eventually led to a correct output. Decisions were made by an annotator with a well-grounded knowledge of Support Vector Machines and their behaviour, which turned out to be quite useful when deciding which output should be classified as "Very Close". For fairness reasons, this extra class was added to the "No" class when delivering the final results. Examples of the "Very Close" class are reported in Table 1.

Original EN	**Original DE**	**Output DE**
majestically	majestatetisch	majestisch
setting	setzend	settend
machineries	maschinerien	machinerien
naked	nakkt	nackt
southwest	suedwestlich	suedwest

Table 1. Examples of the class "Very Close".

In Figure 5 we show the accuracy of the SVM-based cognate generation algorithm versus the baseline, adding the "Very Close" class to both the "Yes" class (correct) and the "No" class (incorrect).

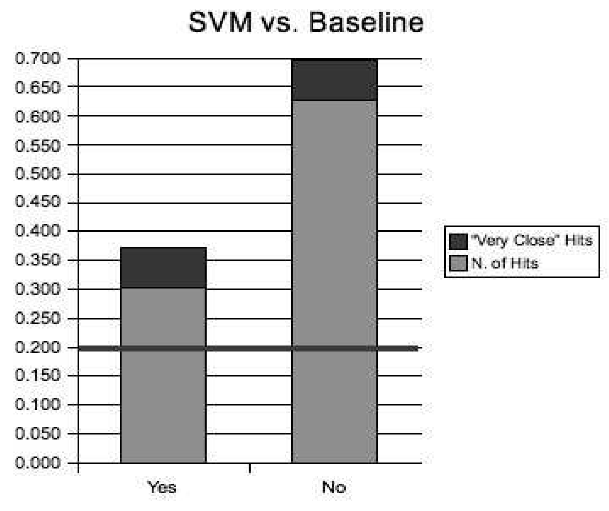

Figure 5. Accuracy of the SVM-based algorithm vs. the baseline (blue line).

The test file included a total of 422 entries, with 85 orthographically identical entries in *L1* and *L2* (baseline). The SVM-based algorithm managed to produce 128 correct cognates, making errors in 264 cases. The “Very Close” class was assigned to 30 entries. Figure 5 shows that 30.33% of the total entries were correctly identified, while an increase of 50.58% over the baseline was achieved.

5 Conclusions and Future Work

In this paper we proposed an algorithm for the automatic generation of cognates from two different languages sharing the same alphabet. An increase of 50.58% over the baseline and a 30.33% of overall accuracy were reported. Even if accuracy is rather poor, if we consider that no knowledge repository other than an initial list of cognates was available, we feel that the results are still quite encouraging.

As far as the learning module is concerned, future ameliorations will focus on the fine tuning of the features used by the classifier as well as on the choice of the model, while main research activities are still concerned with the development of a methodology allowing for language portability: as a matter of fact, n-gram co-occurrencies are currently being investigated as a possible alternative to Edit Distance.

References

Chris Brew and David McKelvie. 1996. Word-Pair Extraction for Lexicography. *Proceedings of the Second International Conference on New Methods in Language Processing,* 45-55.

Pernilla Danielsson and Katarina Muehlenbock. 2000. Small but Efficient: The Misconception of High-Frequency Words in Scandinavian Translation. *Proceedings of the 4th Conference of the Association for Machine Translation in the Americas on Envisioning Machine Translation in the Information Future*, 158-168.

Jesus Gimenez and Lluis Marquez. 2004. SVMTool: A General POS Tagger Generator Based on Support Vector Machines. *Proceedings of LREC '04*, 43-46.

Diana Inkpen, Oana Frunza and Grzegorz Kondrak. 2005. Automatic Identification of Cognates and False Friends in French and English. *Proceedings of the International Conference Recent Advances in Natural Language Processing*, 251-257.

Mehdi M. Kashani, Fred Popowich, and Fatiha Sadat. 2006. Automatic Translitteration of Proper Nouns from Arabic to English. *The Challenge of Arabic For NLP/MT*, 76-84.

Philipp Koehn and Kevin Knight. 2002. Estimating Word Translation Probabilities From Unrelated Monolingual Corpora Using the EM Algorithm. *Proceedings of the 17th AAAI conference*, 711-715.

Grzegorz Kondrak. 2004. Combining Evidence in Cognate Identification. *Proceedings of Canadian AI 2004: 17th Conference of the Canadian Society for Computational Studies of Intelligence*, 44-59.

Grzegorz Kondrak and Bonnie J. Dorr. 2004. Identification of confusable drug names. *Proceedings of COLING 2004: 20th International Conference on Computational LInguistics*, 952-958.

Sara Laviosa. 2001. *Corpus-based Translation Studies: Theory, Findings, Applications.* Rodopi, Amsterdam.

Vladimir I. Levenshtein. 1965. Binary codes capable of correcting deletions, insertions and reversals. *Doklady Akademii Nauk SSSR*, 163(4):845-848.

Gideon S. Mann and David Yarowsky. 2001. Multipath Translation Lexicon Induction via Bridge Languages. *Proceedings of NAACL 2001: 2nd Meeting of the North American Chapter of the Association for Computational Linguistics*, 151-158.

I. Dan Melamed. 1999. Bitext Maps and Alignment via Pattern Recognition. *Computational Linguistics*, 25(1):107-130.

I. Dan Melamed. 2001. *Empirical Methods for Exploiting Parallel Texts.* MIT Press, Cambridge, MA.

Andrea Mulloni and Viktor Pekar. 2006. Automatic Detection of Orthographic Cues for Cognate Recognition. *Proceedings of LREC '06, 2387-2390.*

Michel Simard, George F. Foster and Pierre Isabelle. 1992. Using Cognates to Align Sentences in Bilingual Corpora. *Proceedings of the 4th International Conference on Theoretical and Methodological Issues in Machine Translation*, Montreal, Canada, 67-81.