CrOssInG

Comparing Of AngliciSmS In German

Department of Computational Linguistics
University of Heidelberg

Sebastian Spaar
Spaar@stud.uni-heidelberg.de

Dennis Ulmer
D.Ulmer@stud.uni-heidelberg.de

30.8.2014

Contents

1 Abstract

CrOssInG's main functionality is the creation of transformation matrices from one vector space to another. When using vector data provided by other programs such as word2vec[1], these vector spaces represent a language model of a given language.
CrOssInG is then able to map a word's vector from a German, for instance, to an English vectorspace.
In our software project, CrOssInG was used for the purpose to analyze the semantic change of anglicisms found in the German language regarding their English "originals".

2 Prerequisites

Two major resources were needed for CrOssinG:

1. A large corpus in German and English to create two vector space models using word2vec.
2. A German-English dictionary.

The dictionary tells CrOssInG a standard translation of a word. During the creation process of a transformation matrix, CrOssInG tries to map the word vector of a German word onto the translated word in the English vector space model.
Additionally, information on the use and extent of anglicisms in German and a list of German-English so-called "false-friend" were needed.

3 Extraction of a German-English language model

For an adequate representation of the German and English language, we decided on the broad Wikipedia corpus to use with word2vec.
Since Wikipedia's corpus is freely available as an XML dump[2], we only needed to extract

[1] https://code.google.com/p/word2vec/
[2] https://dumps.wikimedia.org/

the plain text data from it. For this task, we ran a slightly simplified version of Wikipedia Extractor[3] on both an English and a German Wikipedia dump.
After the extraction of plain words, two files of about 1 GB size each were left, containing all the articles' text found in the German/English Wikipedia.
After the extraction, word2vec could be used on the plain text corpus. For each corpus, word2vec provided a file containing a large amount of word vectors. These word vectors represent a language model found in that corpus, and - within the nature of vector data - can be used for further analysis and calculations within that vector space model.
Due to the big size of the corpus, its word vector file contained several thousands of words[4], and processing the vector file within Python was time-consuming. For this reason, we used Python's module cPickle to quickly save and load the vector space data throughout our tests.

4 Extraction of a German-English dictionary

In the case of our project, we used the dict.cc[5] German-English-dictionary as a source. To illustrate the process of extraction, here is a very small two-entry-excerpt from the dictionary:

⋮						⋮
Platin	{n}	⟨Pt⟩		platinum	⟨Pt⟩	noun
platituedenhaft		[alt.]	[geh.]	platitudinous		adj
⋮						⋮

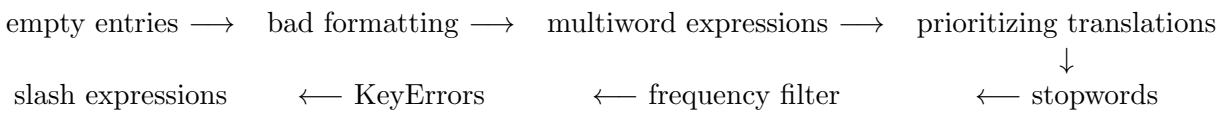
Even this quite simple example illustrates the complexity of the dict.cc-dictionary entry format. Many parts like abbreviations or comments are optional. The content is only used for nouns: It displays the gender or indicates plural form.
Hence, we can derive the following, general pattern for the entries:

German {Num./Gen.} Addition ⟨Abbr.⟩ / [com.] English ⟨Abbr.⟩ / [com.] word type

Of all these pieces of information, only the German and the English entry is needed for this task. Because the quality of the entries differs greatly, e.g. in terms of consistency

[3] http://medialab.di.unipi.it/wiki/Wikipedia_Extractor
[4] 95208 words for the German language model, and 68408 for the English model.
[5] http://www.dict.cc

of use of slashes, we also used different filters to filter out entries which could distort our result.
All of those are optional and and can often be used with different parameters:



- Empty entries: Filters out empty entries which are empty strings that are rarely creating in the dictionary-reading process.
- Bad formatting: Filters out hard to process entries with special characters.
- Multiword expressions: Filters out expression with more than one word.
- Prioritizing translations: Picks the n top translations for entries with more than one.
- Stop words: Filters out entries of German or English stop words.
- Frequency filter: Filters out entries of words with a too low frequency within the corpus.
- KeyError: Filters out entries which throw a KeyError.
- Slash expressions: Filters out entries with the "/"-characters, because the use of slash is very inconsistent.

Note that the recommended selection of filters and their parameters is set as default in the source code.

5 Creation of a transformation matrix

Given two vector space models for the German and English language, say V and W, each $v \in V$ and $w \in W$ is a word represented by the vector that was calculated from word2vec.
We know from the aforementioned that dictionary that for most $v \in V$, there is a $w \in W$ such that w is the translation of v, meaning that $translation(v) = w$.
It is our goal to find a transformation matrix T such that

$\forall v \in V : T \times v = translation(v) = w$, or in short terms: $Tv = w$.
We used the Python package scikit-learn[6] for the calculation of such matrix T. First, we looked for the intersection of all words from both the German and English language model and the words from the dictionary.
This way we operated on a subset of German words that were sure to have a translation within the English language model. Using scikit-learn's linear regression models, we then iterated over each German word and constructed a sample transformation matrix step by step:

$$\begin{array}{c} Hund \\ Katze \\ maus \end{array} \begin{pmatrix} 1 & 2 & 3 & \dots \\ 4 & 5 & 6 & \dots \\ 11 & 12 & 13 & \dots \end{pmatrix} \times T = \begin{pmatrix} 10 & 20 & 30 & \dots \\ \dots & & & \\ \dots & & & \end{pmatrix} \begin{array}{c} dog \\ cat \\ maus \end{array}$$

6 Evaluating different models

After we created different models using scikit-learn's *ridge*[7], *elastic net*[8] and *lasso*[9], we tried to find the best one of the created matrices. Therefore, we used a list of false friends (words in two languages which sound/are written the same or similarly, but have a different meaning), assuming that the most useful matrix would give us the lowest average overall similarity-value for these pairs. Therefore let S be a set of transformation matrices T with $T \in S$ and F a set of false-friend-tuples, where f is the German word and f' the English false friend with $(f, f') \in F$. The best matrix T^* is then to be found with the following formula:

$$T^* = \min_{T \in S} \frac{\sum_{(f,f') \in F} cosine\_similarity(Tf, f')}{|F|}; \qquad cosine\_similarity(a, b) = \frac{a \cdot b}{\|a\| \|b\|}$$

Note that there is no difference between finding the best matrix with the average similarity of all pairs or the total difference; the average value is just a bit more intuitive because it can be easily transformed into a percentage-value.

[6] http://scikit-learn.org/stable/
[7] http://scikit-learn.org/stable/modules/linear_model.html#ridge-regression
[8] http://scikit-learn.org/stable/modules/linear_model.html#elastic-net
[9] http://scikit-learn.org/stable/modules/linear_model.html#lasso

7 Results

Comparing different combinations of models and alpha-values, we found that a *elastic net* model with an alpha value of $\alpha = 0.1$ performed best using our method of evaluation. Calculating the similarity of all anglicism pairs (to be precise: The similarity of the German anglicism mapped into the English vector space and the English word's vector) gave us the following results (excerpt of the top ten best and worst):

	Highest similarity pairs	similarity	Lowest similarity pairs	similarity
1.	peeling - exfoliation	81.80%	city - city	46.83%
2.	body - onesie	77.97%	team - team	47.65%
3.	spray - spray	76.75%	campus - campus	48.57%
4.	aftershave - aftershave	76.59%	in - in	48.87%
5.	t-shirt - t-shirt	76.09%	golden_goal - golden_goal	49.27%
6.	shampoo - shampoo	76.05%	park - park	49.69%
7.	deodorant - deodorant	75.70%	australian_football - australian_football	49.70%
8.	deo - deodorant	75.70%	butler - butler	49.74%
9.	eyeliner - eyeliner	75.53%	posten - post	49.75%
10.	ou - egg	74.23%	festival - festival	49.79%

Whilst the top ten of highest similarities seem promising at first glance, many of the results of the lowest-similarity top ten seem to be beyond the usual amount of freak values and therefore let us question our approach.
We try to analyze some of the possible reasons for this in the next chapter, "conclusion".

8 Conclusion

With CrOssInG we provided a tool to use vector space models we created with word2vec from a German/English wikipedia corpus and other data from dict.cc and Wiktionary to map vectors of words into another language's vector space.

However, the results we received do not seem to make sense in all cases, so we tried to analyze our approach for possible weak spots:

We suspect that the outcome of word2vec might not be optimal for scikit-learn regarding the vectors: The pure amount of vectors as a result of the huge Wikipedia corpus we used could have led to a '"watered down" transformation matrix, which tried to map all vectors to their target vectors as best as possible by generalizing values too much, but hence failed to give good results for each individual vector pair. We base this idea on a much better performance of our system we observed on scaled-down datasets.
Another problem could be the evaluation of our matrices with false-friend-pairs:
We used this approach with the presumption that the similarity of false-friends-pairs had to be lower than the similarity of average dictionary entries, although this lacks evidence. It probably would have been better to use some German-English word pairs with a very high and certain similarity and to use them to find the matrix that maximizes their overall average similarity.
It would be interesting for future projects to improve the system by adjusting the transition between word2vec and scikit-learn and rethinking the evaluation process of the transformation matrices. Besides, this work could also lead to new insights by applying it onto other languages or measuring semantic developments over a larger timespan.