CLIN 27 Shared Task - Translating Historical Dutch Text

supervised by Rob van der Goot and Gertjan van Noord

Remko Boschker
master student of information science at the Rijks Universiteit Groningen
s1282603, r.boschker@student.rug.nl

Abstract

This report describes a system that uses a HMM, a bigram language model of modern Dutch and the Viterbi algorithm to perform the CLIN 27 shared task of translating seventeenth century Dutch into a form that is better processed by POS tagging tools developed for modern Dutch. The system does not perform well compared to systems using a character based model and machine learning technology. It came second to last with a Bleu score of 0.45. The top three teams from Helsinki, Bochum and Ljubljana had scores around 0.60.

1 Introduction

As more historical text is becoming available online and interest in using these texts for linguistic and historical research grows, the need for tools that assist this research increases as well. For many modern languages such tools have been developed. However these tools generally do not perform well on historical texts. The spelling can differ, words have appeared or disappeared, frequencies of use have changed and their grammatical composition as well.

Faced with these challenges tools can either be modified or retrained as if the historical texts were another language or the language can be somehow normalised or modernised to improve the performance of existing tools. The CLIN 27 shared task (Sang and others, 2017) chooses the latter approach and focusses on translating seventeenth century Dutch text for improved part-of-speech tagging performance. Please find an example sentence from Steven Blankaart's *Nederlandsche Herbarius* from 1637 below along with its modernisation.

> Voorts soude ik veele vreemde en gebruikelyke gewassen , hier tusschen gevoegt hebben , om dat voor de Leerlingen , en andere Weet-gierige de gewoonlyke Lands- en Stads-tuinen meest toegesloten syn .
>
> Voorts zou ik vele vreemde en gebruikelijke gewassen , hier tussen gevoegd hebben , om dat voor de leerlingen , en andere weetgierige de gewoonlijke lands- en stadstuinen meest toegesloten zijn .

The task consists of translating text word for word. This token alignment supposedly assures that any part-of-speech tags correspond directly to the historical token in the same place. A 1260 word translated text is provided as development data as well as the mostly aligned translations of the *Statenbijbel*. The translations of ten previously unseen texts each from a different decade of the seventeenth century are evaluated using the Bleu score and for one text the actual POS tagging performance is included.

In the following I report on my participation in the CLIN 27 shared task where I translate historical Dutch texts to improve the performance of existing language technology. I choose a classic stochastic approach using a HMM and bigram model, because there is little training data available and because I am interested to see how far I can get using this technique. The question is whether a bigram language model provides sufficient information to normalise a historical text for the purpose of processing it with technology developed for modern text.

2 Approach

Any approach to the task needs not only to handle differences in spelling and syntax, but also lan-

guage variety and change across the century and authors as well as names and idioms rarely used anymore, but that do have a modern equivalent. There is little training data which makes more brute force machine learning approaches harder to use. Sang et al. (2017) reports that some task participants use the Moses machine translation system and related technology in some form or other. Two teams use a lexicon and rewrite rules. Some teams use a language model on character level and some on word level. The performance resulting from these approaches is discussed later on.

Following the advise of my supervisors I implemented a system that represents sentences and their possible translations as a Hidden Markov Model. I use the Viterbi algorithm to find the most likely translation. I derive the probabilities in the model from a word level uni- and bigram language model of modern Dutch.

I base the possible translations on a maximum Lebenshtein distance of all words in a vocabulary to the original word. The maximum distance is the length of the word divided by three rounded up to the nearest integer. I preprocess the original word replacing an *y* with an *i* and the matching algorithm ignores capitalisation. If there is no word in the vocabulary that is within the maximum distance, the original is used as the translation. If a token does not contain a lower case letter (for instance punctuation) the token is used as the only possible translation.

The emission probability of a possible translation is the unigram count of the translation in a corpus of modern Dutch divided by its Lebenshtein distance to the original plus one. The transition probability from one possible translation of a word to a possible translation of the next word is modeled by their bigram count. I smooth over absent counts by adding 0.5 as the count. I select the possible translation sentence with the highest probability as correct.

I evaluated the possible translations for different vocabulary, unigram model, edit distance and preprocessing choices. Table 1 shows the percentage of the correct translation being in the set of possible translations, labelled *found*, the largest number of possible translations for a word in the text, labelled *maxSet* and the average number of possible translations, labelled *avgSet*. The last two values labelled *first* and *avgPos* show for the correct translation the percentage of it being ranked first and its rank on average. The edit distance can either be the word length divided by three rounded down, labelled *floor* or up, labelled *ceil*. Using *floor* words can be preprocessed, labelled $y \rightarrow i$, and lastly capitalisation can be ignored by the Lebenshtein distance matching algorithm, labelled *cap*.

The size of the sets of possible translations that need to be processed by the Viterbi algorithm impacts performance and memory requirement drastically. I selected the GNU Aspell dictionary[1] as the vocabulary and a uni- and bigram model of the SONAR corpus[2] generated by Rob, one of my supervisors. The Google ngram models[3] show a slightly higher coverage, but the average position is lower and it comes at a high performance cost.

When I received the test data, I noticed that the tokenisation had changed. Previously sentences were not marked by a new-line and paragraphs were marked by two. The new data contained a new version of the development data as well. On the original development data I had a Bleu score of 0.62, but on the same text in the test data I had a Bleu score of 0.42. The Bleu scoring application provided by the organiser depended on the newline and scores are influenced by sentence length.

As a last-minute fix I added the following rules. If the exact token is in the vocabulary, it is the only possible translation. If the token is in a lexicon generated from the development data, then the same translation is used as in the development data. I also added some translation rules for frequent words that were not being translated correctly such as ”der”, ”des” and ”soo”. Unfortunately there were a few errors in the generated lexicon which actually hurt the performance. They will be discussed later on. I find the last minute scuffle takes away from the initial idea to rely on a stochastic model without any training or data specific tweaking.

3 Implementation

I implemented the system in javascript on Node.js. I like working in javascript and wanted to learn about the state of natural language processing (nlp) in it. I did not find any reasonably mature nlp projects beyond projects oriented towards input helpers and spellcheckers on websites. Find-

[1] ftp://ftp.gnu.org/gnu/aspell/dict/0index.html
[2] http://lands.let.ru.nl/projects/SoNaR/
[3] https://catalog.ldc.upenn.edu/LDC2009T25

Table 1: evaluation of possible translations generation

	Aspell				Rob - SONAR				Google Ngrams	
	floor	ceil	$y \rightarrow i$	caps	floor	ceil	$y \rightarrow i$	caps	floor	caps
found	63%	69%	70%	74%	65%	71%	73%	74%	67%	76%
maxSet	353	1439	1439	1510	609	2103	2103	2868	2362	14669
avgSet	40	177	182	213	77	400	412	632	315	3337
first	-	-	-	-	24%	12%	12%	12%	21%	10%
avgPos	-	-	-	-	3	12	12	15	4	29

ing all the words in the vocabulary that are within some editing distance to the original word is the most time consuming operation in the program. I wanted to speed up this process by taking the intersection of a Levenshtein automaton with a linearly constructed minimal fsa of the vocabulary. I did some tests with an existing library[4], but it actually consumed much more memory than the size of the vocabulary in a text file on disc. I set out to develop a library in javascript[5] that could support this approach, but lack of time and experience forced me to fall back to a linear search.

I implemented the system using streams, but was having memory issues. Considering the maximum set size of candidate translations of about 1500 there can be at most approximately two million combinations of consecutive words in the original. Because the Viterbi algorithm only needs to keep track of the previous set of results this should not cause any problems. I mistakenly suspected the stream implementation holding onto references to data and rewrote the implementation using node.js native promises (es2017 async / await). The issue persisted and after some memory profiling and further study into node.js memory management I found that I could control when the garbage collector starts freeing unused memory. After some trial and error I found the right setting and could run the system on my 16 GB laptop.

The implementation path I took was not the most straight-forward or efficient. There are very good libraries for nlp and building finite-state automaton systems for several other languages as well as complete machine translation systems. Still I learned a lot and regret I was not able to contribute more to nlp in javascript. It would be very useful on the web as well make nlp more accessible to the many javascript developers. Nlp in js can certainly be done, but you need to be aware of performance issues.

[4]https://github.com/universal-automata/liblevenshtein

[5]https://github.com/informatietuin/automata

4 Results and Discussion

Table 2: Top 10 errors made in test data

Gold	Our	Count
de	De	330
ende	En	124
zo	dus	49
haar	het	45
mij	Mij	45
des	van_het	38
der	van_de	25
edele	E.	19
zij	hij	19
met	mit	19

As I mention in the approach section of this report I did some last minute tweaking to improve the Bleu score my system achieved on the new version of the development data that came with the test data. The translations I entered into the shared task achieved a score of 0.45. Erik Tjong Kim Sang, the organiser of the shared task, reported that he noticed that I had some strange errors for instance capitalising all tokens *de*. After the competition the gold data was released and I investigated. Table 2 shows the top ten errors. As it turns out the lexicon generated from the development data I used had multiple entries for *de, mij en zij*. The erroneously used incorrect entries account for a 0.04 drop in the score. The gold data left *des, der, ende* untranslated because these words are in a standard dictionary. However they are not modern Dutch and I find whether or not my translations of them were in error debatable. They account for another 0.02 point drop.

Table 3 shows the results of the other entries for the shared task. You can see that character based systems using some form of machine learning per-

Table 3: Scores from the different teams

Team	Bleu	POS	Approach
Ljubljana	0.61029	0.842	char, mgiza, moses, rules, Lev
Bochum	0.60665	0.824	char, mgiza, neural network
Helsinki	0.59896	0.856	char, NN and Moses, Lev
Amsterdam	0.56783	0.836	mgiza, rules
Leuven	0.53752	0.825	derived lexicon and spelling rules, Lev
Utrecht	0.46867	0.812	rules and lexicon from parallel text
Groningen	0.45061	0.793	Viterbi, ngram model, Lev, lexicon
Valencia	0.43011	0.759	mgiza, moses
baseline	0.33097	0.709	Lexicon from parallel text

form best. The best performing systems differ in the way they align, expand and refine the training corpus. Further details can be found in Sang at al. (2017). My system did not do very well coming second to last. Correcting for the errors mentioned puts my system closer to the rest at 0.51. It would have been nice to compare the performance of the pure HMM and ngram-model system, but I abandoned the purist approach and entered a somewhat hybrid system.

In future work it would be nice to revisit this pure approach. I did not evaluate the performance of the ranking of the candidate translations in detail and I expect some improvements can be made. The alignment of tokens, especially the splitting and merging of translations, can be handled more graciously by my system as well as by the shared task design itself. Lastly it would be interesting to evaluate this approach using a tri-gram model rather than a bigram one for transition probabilities. However sparsity of the bigram model was already a serious issue and this will be far greater for tri-grams. To revisit the question posed in the introduction if a bigram word model provides sufficient information to modernise a historical text for automated processing, it seems that character based models of a higher complexity than a HMM do a better job.

References

Erik Tjong Kim Sang et al. 2017. The clin27 shared task: Translating historical text to contemporary language for improving automatic linguistic annotation. submitted to volume 7 of the CLIN journal, available at https://ifarm.nl/clin2017st/paper/.