Language and Speech Processing

Nicholas Piël - 0104612

Bart J. Buter - 9910522

Sanne Korzec - 0252700

{npiel,bjbuter,skorzec}@science.uva.nl

1 Introduction

Currently human-computer interaction is limited by the use of mouse and keyboard. It would be an improvement when humans could communicate through language and speech with intelligent devices in their environment. On the side of these intelligent computational devices, the task of understanding language is far from trivial. The language and speech processing community has set itself the task to move towards understanding language and speech by computational devices.

A commonly used model for communication is the noisy channel model, which describes communication in terms of a sender, receiver and a message which needs to be passed through a noisy communication channel. Algorithms try to decode and reproduce the original message. Our task of understanding the semantics of a sentence can be helped by using the syntactic structure of the sentence, to narrow the search and exclude certain ambiguities.

In this paper, we will describe a decoder that returns the syntactic structure of a given sentence. This is decoder is also known as a part-of-speech(POS) tagger. We will report on research relevant to POS-taggers. Such as the influence of the choice of n-gram language model and the influence of different types of smoothing.

In section 2 we describe the theory which forms the basis of our POS-tagger. Section 3 is used to describe our research methodology. Section 4 shows the results to our experiments performed with the POS-tagger and in section 5 we discuss our findings.

2 Theoretical Model

In this section we will describe the theory which is the basis for POS-tagger. We will describe the lexical and language model, the noisy channel model, the Viterbi algorithm, smoothing and related theories and concepts. These theories are based on [2, 3, 4].

2.1 Lexical and Language model for Part-of-speech tags.

Consider the sentence: "I took a walk.", walk is a noun, however walk could also be a verb, as in: "I walk.". When we use a model which makes use of POS-tags, we can disambiguate between the different uses of "walk". This disambiguation is essential when trying to understand natural language. We will describe a model that uses POS-tags and describe its usage to create a POS-tagger. The task of a POS-Tagger is to generate a POS-tag sequence corresponding to a given sentence .

The model can be split into two parts: the language model which describes the probabilities of a tag sequence i.e. $P(t_1, t_2, ..., t_m) = P(t_1^m)$, and a lexical model describing the probabilities of a word sequence w_1^m given a tag sequence t_1^m i.e. $P(w_1^m|t_1^m)$. Below we will describe these lexical and language and models.

2.1.1 POS-tagger as noisy channel decoder

The model we use is based on the notion of a noisy channel. In a noisy channel, a message M is sent through a channel, on the other side of the channel the message M' is received. The received message M' is not necessarily the sent message M. It is the task of the decoder to try to make an estimate $\hat{M}$ of the original message M from the received message M'.

For a POS tagger we can use the noisy channel model as follows: the original message being sent is a sequence of POS tags t_1^m, the received message is a sequence of words w_1^m. The POS tagger is the decoder which creates an estimate $t_1^{\bar{m}}$ of the original tag sequence.

More formally, given a powerset operator $*$, a vocabulary V of words, a set of POS tags T, word sequence w_1^m, where $w_1^m \in V^*$ and tag sequence t_1^m where $t_1^m \in T^*$, find the tag sequence $t_1^{\bar{m}} \in T^*$ that is the most likely tag sequence corresponding to the received message w_1^m. Thus the task of the POS-tagger is

$$t_1^{\bar{m}} = \arg\max_{t_1^m} P(t_1^m|w_1^m).$$

2.1.2 The Language and Lexical Model

The noisy channel model indicates that there are two processes at work to ultimately produce the received message M'. Namely the process that produces the message M and the process of transmitting M through the channel $M \rightarrow M'$.

These two processes are modeled in the language and lexical model. We assume that the tag sequence t_1^m is generated from a probabilistic process, the related model is the language model $P(t_1^m)$. The channel is also assumed to be a probabilistic process which is modeled by the lexical model $P(w_1^m|t_1^m)$.

This gives us a way of expressing the probabilities needed for the POS-tagger, using Bayes' rule we get:

$$P(t_1^m|w_1^m) = \frac{P(w_1^m|t_1^m)P(t_1^m)}{P(w_1^m)}.$$

However, we are only interested in $t_1^{\bar{m}} = \arg\max_{t_1^m} P(t_1^m|w_1^m)$, w_1^m is given, making the term $P(w_1^m)$ equal for all tag sequences. Therefore we can leave this out of the equation without changing any results. We write:

$$P(t_1^m|w_1^m) \propto P(w_1^m|t_1^m)P(t_1^m).$$

thus,

$$\begin{aligned} t_1^{\bar{m}} &= \arg\max_{t_1^m} P(t_1^m|w_1^m) \\ &= \arg\max_{t_1^m} P(w_1^m|t_1^m)P(t_1^m) \end{aligned} \quad (1)$$

This means that we can calculate the most likely tag sequence $t_1^{\bar{m}}$ once we have a lexical model $P(w_1^m|t_1^m)$ and a language model $P(t_1^m)$.

2.2 Estimating the language and lexical models

In the previous section we discussed the language and lexical model, the POS-tagger and their relationships. They all work with probabilities of sequences, these probabilities need to be obtained from somewhere. Corpora, sentences POS-tagged by linguists, can be used to estimate the needed probabilities. We will describe the assumptions made in our model to be able to estimate sequence probabilities from corpora.

2.2.1 Markov assumption

The sets V^* and T^* have infinite many sequences. making it impossible to obtain the probabilities of all sequences. However, they can be approximated when we apply the Markov assumption. This can be stated as: "The conditional probability of observing an item in a sentence is not dependent on all items in the sequence, but only on a fixed number n of preceding items." We will call this an nth-order Markov assumption, which uses sub sequences of $n+1$ words, thus an $n+1$-gram or N-gram model.

Let m bet the length of a sentence, for our language model this results in the approximation

$$\begin{aligned} P(t_1^m) &= \prod_{k=1}^{m} P(t_k|t_1^{k-1}) \\ &\approx P(t_k|t_{k-n}^{k-1}). \end{aligned}$$

We will call t_{k-n}^{k-1} the context for tag t_k.

For our lexical model we make the assumption that words are only dependent on their corresponding tag.

$$\begin{aligned} P(w_1^m|t_1^m) &= \prod_{k=1}^{m} P(w_k|w_1^{k-1}, t_1^k) \\ &\approx \prod_{k=1}^{m} P(w_k|t_k) \end{aligned}$$

2.2.2 Obtaining the models from corpora

We can estimate the probabilities for our models by using counts of N-gram occurrences in a corpus. Let Z be the total number of all sequences of length N in the corpus, and let T be the set of all tags in the corpus, and let k denote a location in a sequence. The ˆ- symbol indicates an estimate. We can estimate using:

$$\hat{P}(t_{k-N+1}^{k-1}, t_k) = \frac{Count(t_{k-N+1}^{k-1}, t_k)}{Z}.$$

Because all $t \in T$ partition the event-space we can express the chance for a sequence of length $n-1$ in terms of sequences of length n.

$$\hat{P}(t_{k-N+1}^{k-1}) = \frac{\sum_{w \in W} Count(t_{k-N+1}^{k-1}, t)}{Z}$$

With the definition of conditional probabilities and some mathematical manipulations we get:

$$\begin{aligned} \hat{P}(t_k|t_{k-N+1}^{k-1}) &= \frac{\hat{P}(t_{k-N+1}^{k-1}, t_k)}{\hat{P}(t_{k-N+1}^{k-1})} \\ &= \frac{Count(t_{k-N+1}^{k-1}, t_k)}{\sum_{t \in W} Count(t_{k-N+1}^{k-1}, t)}. \end{aligned}$$

The probabilities for the lexical model can be estimated in a similar fashion to the sequence of tags. We use the fact that $w \in V$ is a partition of the event space, therefore we get:

$$\begin{aligned} \hat{P}(w_k|t_k) &= \frac{Count(< w_k, t_k >)}{Count(< t_k >)} \\ &= \frac{Count(< w_k, t_k >)}{\sum_{w \in V} Count(< w, t_k >)}. \end{aligned}$$

2.3 The Viterbi algorithm

In previous sections we have described our language and lexical models, our POS-tagger and how to estimate the models from corpora. Looking at formula 1 it seems we have all ingredients needed for our POS-tagger.

A practical problem which does arise is the exponential explosion of possible tag sequences given a sentence. If every word in a sentence with z words has only 2 possible tags the number of possible tag sequences becomes 2^z.

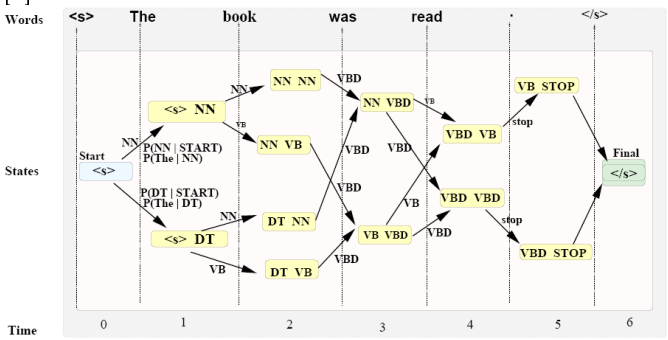
The Viterbi algorithm is an algorithm that can find the most likely POS-tag sequence for a sentence. We will describe the Viterbi algorithm and related topics in this section.

2.3.1 Trellis-graph

A trellis is a graph with nodes grouped by time, it has two terminal nodes, namely a start and a stop node. All non terminal nodes are connected to at least one node in an earlier time group and at least one node in a later time group. Thus through every non terminal node there is a path from the start to stop node.

A trellis can be used to represent a tag sequence corresponding to a sentence. The start and end nodes of the trellis correspond with the start and end nodes of the sentence. The time groups correspond with words in the sentence. The nodes in a time group correspond with contexts of POS-tags, that can produce the word for that time group.

Figure 1: Trellis for a 2nd order Markov Language Model [2]



Edges from nodes in one time group to the next correspond with POS-tags given the context of the outgoing node, these tags will be part of the context in the ingoing node. The edges can also be seen to correspond with the probabilities in our language model $P(t_k|t_{k-N+1}^{k-1})$, in this context also referred to as transition probabilities. A trellis is displayed in figure 1.

We can augment the trellis to also include our lexical model $P(w_k|t_k)$. In this context the probabilities for the lexical model are often referred to as emission probabilities. Because there is an unseen process - generating a POS-tag sequence - which emits observable events, the word sequence. Every edge in the graph emits a word with probability $P(w_k|t_k)$, but every word in the same time group emits the same word. Thus every edge emits a single word. Therefore we augment the original trellis by also storing the corresponding probabilities for the lexical model on the edges.

The described trellis can be seen from viewpoints as a Stochastic Finite State Transducer, the transition and emission processes are defined as a State Machine. A Hidden Markov Model, is another view, where the basis is a statistical model, not a machine where the nodes are called states, and edges transitions. The trellis is the last viewpoint where we are using a graph to describe and store the probabilities associated with the underlying processes.

2.3.2 The Viterbi algorithm

The Viterbi algorithm is a dynamic programming (DP) algorithm which calculates the most likely POS-tag sequence. This is done by calculating the maximal probabilities for every node in a trellis. The algorithm calculates the probabilities of the nodes at time=ts by only using the probabilities it stored at the nodes of time=$ts-1$. And the transition and emission probabilities stored in the trellis at the edges between nodes of $ts-1$ and ts, using the following formula:

$$\gamma_{ts}(s_i) \quad = \quad \max_{s_j} \gamma_{ts-1}(s_j) \times P(s_i|s_j) \times P(w|s_i),$$

where $\gamma_{ts}(s_i)$ is the maximum probability for the sequence of hidden states ending with $s_j, \quad s_i$ at time $ts-1$ and ts respectively, these probabilities can be stored at the nodes in the trellis. $P(w|s_i)$ and $P(s_i|s_j)$ corresponds with probabilities from the lexical and language model, stored on the edge between s_j and s_i in the trellis.

Algorithm 1 Viterbi Algorithm

$\Gamma =$`Viterbi(trellis,` w_1^k`)`
$\gamma_0(s_0) = 1$
`for` $ts <$ `max(time`$\in$`trellis)`
 `for state`$\in$`timegroup(t)`$\in$`trellis`
  $\gamma_{ts}(s) = \max_{s_j} \gamma_{ts-1}(s_j) \times P(s|s_j) \times P(w_{ts}|s_i)$
`return sequence_contributors(`$\max_s \gamma_{\max(\texttt{time})}(s)$`)`.

Shown in algorithm 1 is the Viterbi algorithm. It is easy to store the arg max whenever the algorithm chooses a maxvalue. These stored arguments could be used to return a sequence with for every time group a tag, that contributed to the most likely sequence of POS-tags. In equation 1 we described the formula for task of our POS-tagger. The Viterbi algorithm calculates exactly the wanted likelihoods, and returns the most likely POS-tag sequence.

2.4 Smoothing

Even though we can find the most likely POS-tag sequence given a sequence of words, we can still improve on the tagger. This has to do with the fact that models estimated from a corpus can only roughly approximate natural language.

Most problematic are unseen occurrences or events. Since our estimations are based on counts, when we get a word or a POS-tag sequence we have not seen before it receives probability 0. This will lead to the tagger concluding that an event unseen during training is impossible. But the event exists because it is being presented during tagging, therefore it should get a reasonable probability.

To get a reasonable probability we will have to smooth our estimated models. There are many approaches to apply smoothing on the language model, we have implemented Good Turing and Katz' Backoff.

2.4.1 Lexical model smoothing

For every word in a sentence presented to the tagger, it locates all tags and their respective probabilities $P(w|t)$, it has observed with this word during training. When we come across an unknown word, we have chosen the solution to use expert knowledge to assign the word a reasonable POS-tag.

This expert knowledge has been put into a simple decision rules. With rules such as: if the unknown word starts with a capital, assign it the tag; proper noun (NNP). These rules were derived after a study of unknown words and their tags. The words to which no linguistic rule applies are tagged as NN, because this is the largest group to which no rule applies, see also figure 2.

2.4.2 Language model, Good-Turing smoothing

In the language model we represent a POS-tag sequence as t_1^m. We want to know the probability of an unseen t_k given the sequence of up to $k-1$, t_{k-N+1}^{k-1}. When using the Good-Turing method [6], we want to reserve some portion of the

Algorithm 2 Tag of unknown word, decision rules

```
if contains_number(word)          : tag = CD
elseif starts_with_capital(word): tag = NNP
elseif contains_dash(word)        : tag = JJ
elseif ends_with_s(word)          : tag = NNS
elseif ends_with_ing(word)        : tag = VBG
elseif ends_with_ed(word)         : tag = VBD
else                              : tag = NN
```

total probability and reuse this when we encounter unseen events.

The probability of event t_{k-N+1}^{k-1} can be written as $P_{GT}(t_{k-N+1}^{k-1}) = \frac{r^*}{N}$, where r^* is the smoothed count of event t_{k-N+1}^{k-1} and can be formally written as $r^* = (r+1) * \frac{n_{r+1}}{n_r}$. r is the actual count or frequency of event t_{k-N+1}^{k-1} and n_r is the number of events with frequency r.

However, when choosing an implementation there are some problems: Good-Turing estimation uses values of neighboring events in such a way that if n_{r+1} or $r+1$ is zero, the current smoothed event $P_{GT}(t_1^{k-1})$ becomes zero. This makes it difficult to smooth the entire range of events.

There are several methods to overcome this difficulty, we have chosen to smooth the events with the lowest frequencies as they are closer to zero events than events with higher frequency. When a neighboring event had zero probability, we used the real value for the subsequent event and the reserved probability for the zero event.

2.4.3 Language model, Katz' backoff smoothing

We also implemented Katz' backoff [5]. This method behaves in such a way that when no t_k for an event t_{k-N+1}^{k-1}or event t_{k-N+1}^{k-1}does not exist, we use the $n-1$-gram language model instead of n, when this does not exist we use $n-2$ and so on. When $n-x$ becomes 1, we have no lower n^{th}-model to rely on. In this case we switch to the Good-Turing estimate again.

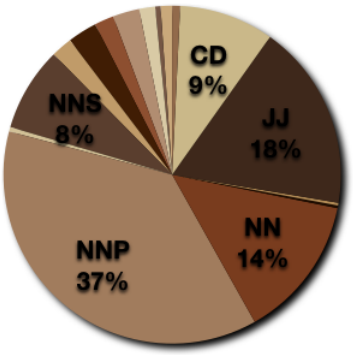
3 Research Methodology

We implemented a POS-tagger based on the models and principles described in section 2. We used a corpus with sentences from an English financial newspaper, this corpus was divided in a training set with sentences and POS-tags, a test set with only sentences and a gold set with the true POS-tags for the sentences in the test set. The test set included 2245 distinct sentences with an average length of about 20 words per sentence, while the training set had 39832 distinct sentences with an average of about 23 words per sentence.

Research on the influence of the length of n-grams and the different smoothing techniques was conducted by testing the accuracy of the POS-tagger on the test set with different combinations of n-gram and smoothing settings.

To obtain data on unknown words, we trained the POS-tagger on our corpus and observed the unknown words. The words were grouped by the POS-tag that they should be associated with. The largest groups were analyzed by linguists on common features within the group.

Figure 2: POS-tag distribution of unknown words



4 Results

4.1 Baseline results

The baseline POS-tagger consists of a simple tagger without any smoothing on the lexical or language model. The results can be seen in table 1 under standard. The accuracy for a bigram is around 84%, we can see that its performance will not increase with larger n-grams. We assume that this is being caused by the sparseness of the training set as increasing n-grams requires a larger corpus a lack of smoothing will penalize this.

4.2 Lexical model smoothing

The frequency of unknown words per POS-tags are displayed in figure 2.

The figure clearly shows that nouns make up more than half of the unknown words, with the largest group being proper nouns (NNP). This high frequency of unknown NNP's can be explained by the notion that there can be infinitely many names of people or organizations. The same notion applies to the cardinal numbers(CD), since there are infinitely many numbers, the training set will only contain a small fraction of these. Therefore many of them will be unknown to the POS-tagger after training. However, both classes are good examples of classes where linguistic knowledge can make a difference. Since numbers are numerical and proper names start with a capital, they can be tagged correctly without the need to have seen all instances of this class during training. These results are incorporated in the POS-tagger by the rules shown in algorithm 2.

We can see from the results in table 1 under +LexE that this expert knowledge increases the baseline result with about 4% for each n-gram.

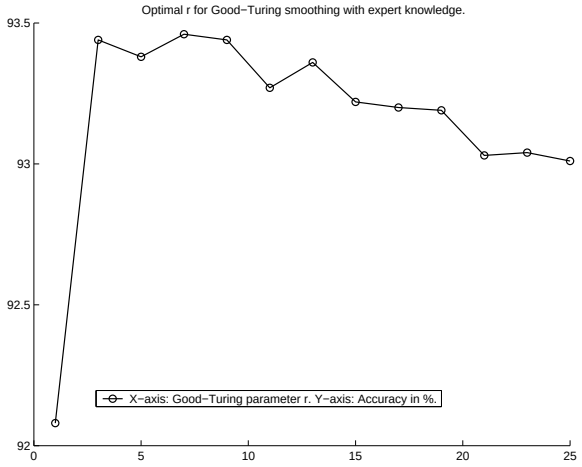
4.3 Language model smoothing

As discussed in subsection 4.1 we can see that there is a big drop in performance for increasing n-grams. The be-

Table 1: Accuracy results for n-gram POS-tagger with different smoothing parameters.

n-gram	standard	+LexE	+LexE+GT	All
2	84.25%	89.28%	90.86%	90.86%
3	83.85%	87.54%	**93.36%**	92.92%
4	77.40%	80.57%	93.29%	92.47%

Figure 3: Good Turing cutoff parameter r



havior was assumed to be caused by the lack of smoothing. When we add Good-Turing smoothing to our model, we can see in table 1 under +LexE+GT that the results improve dramatically for large n-grams. The accuracy of the POS-tagger jumps from 80.57% to 93.29% for 4-grams. However, increasing the n-gram still does not mean increasing the accuracy as the best performance can be obtained by using a 3-gram.

In order to try and improve on this we tried an additional different smoothing method, namely the Katz' backoff. This method is mainly targeted to handle sparse data that will occur with larger n-grams. Because when it observes an n-gram which does not exist in its language model it will fallback to a lower n-gram and use the observed probabilities on the training data for this lower n-gram.

In table 1 under All we can see that using Katz' backoff does not improve results. A reason for this could be that the lower n-grams do not correctly represent the possibility or the meaning of the higher n-grams. This is known behavior for this algorithm [2].

Since a POS-tagger with trigrams and Good-Turing smoothing was our best performing tagger, we have looked at the parameter for Good-Turing. This parameter gives the cutoff parameter r, for occurances with frequency $\geq r$ no smoothing is attempted. It turned out that $r = 7$ was the optimal parameter. Figure 3 shows the behavior for different values of r.

5 Conclusion

From the results displayed in table 1 it can be seen that a POS-tagger based on 3-grams with the Lexical expert rules for unknown-words, together with Good Turing smoothing is the superior method. The main contribution is due to the usage of Good-Turing smoothing. Some of the accuracy of the method can be contributed to the lexical expert rules for unknown words.

However there are some issues to our implementation that need to be addressed: First of all, we assume that using sequences of length N in a sentence represent enough information to estimate the correct probabilities. One can argue that the relation between 'did' and 'decide' in the following sentence is lost when N is chosen too small: 'Did the referee, after considering all remarks, decide incorrectly?'

On the other hand, when N is chosen too large, we need an extremely large corpus if we want to avoid an explosion of zero occurrences. Smoothing, used in whatever form, is always an estimation of the actual probability and therefore if the smoothed occurrences become larger, the performance will decrease.

When using the Katz' backoff smoothing algorithm for larger N-grams, there will be more zero occurrences. Therefore the algorithm will value smaller n-grams as more important than their own N^{th} probabilities. This will probably be a poor representation of the actual probabilities.

Another issue are the lexical rules. Even though the chosen rules work very well for the current corpus, it can be postulated that they might over-fit since we are only using one corpus. The same rules will most likely perform much worse on another domain. This leads to the question whether this is actually a problem or a feature. When dealing with NLP problems, many fields have discovered that models only work for the domains they are trained on. This is also mathematically proven by the No Free Lunch theorem [1].

References

[1] No Free Lunch Theorem. http://www.no-free-lunch.org

[2] Sima'an. Language and speech Processing course 2007. http://staff.science.uva.nl/%7Esimaan/D-LangAndSpeech03/LangANDSpeech07.html

[3] Manning and Schutze. "Foundations of Statistical Natural Language Processing". The MIT Press, 1999.

[4] Jurafsky and Martin. "An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition". Prentice-Hall, 2000.

[5] Joshua Goodman and Stanley Chen. "An empirical study of smoothing techniques for language modeling". Technical report TR-10-98, Harvard University, 1998.

[6] William A. Gale and Geoffrey Sampson. "Good-Turing Frequency estimation Without Tears". Journal of Quantitative Linguistics, vol 2, 1995.