3 Part of Speech Tagging

In this chapter we will look at a statistical method which can be used for adding part of speech tags (POS tags) to a corpus. The method is based on a message model from information theory called the noisy channel model. This model can also be used for other purposes than POS tagging. We will start the chapter with an explanation of two other information theory concepts: Zipf's law and text entropy.

In this chapter we will use the book [Cha93]: Statistical Language Learning by Eugene Charniak and the overhead sheets collection [LS93]: Statistical Methods in Natural Language Processing by Mark Liberman and Yves Schabes.

3.1 Information Theory

Information theory is a quantitative theory about the distribution of information elements in some information structure. The theory was developed for describing messages in radio transmissions. We will use it for describing texts. We will regard texts as messages and words (sometimes characters) as information elements.

When you look at the frequency of words in a text you will see that words have different frequency. Some words appear many times and others only a few times. The frequency distribution is best described with ZIPF'S LAW. This law states how the frequency distribution will be of the words in a text when they are put in a list ordered in decreasing frequency:

$$f(word)*rank(word) \approx constant$$

(published in 1949 by George K. Zipf). So the product of the frequency of a word and its rank on a frequency list will approximately be constant. When you look at figure 1 which represents Zipf's law you will see that there will be many words with a low frequency and only a few with a high frequency. Example data from the Swedish Press65 corpus confirms this: the corpus contains more than 30,000 words with frequency one but there is only one word which has a frequency that is larger than 30,000.

Low frequency words are a problem for a statistical language model: their frequencies are not very accurate. One may try to get rid of the problem by increasing the size of the corpus that was used for building the model. Zipf's law tells us that this will not get rid of low frequency words. When make our corpus 10 times as big the graph representing the distribution of the words will move up. The frequency of the low frequency words will increase but we will get even more words with low frequencies in the larger corpus (see figure 1).

Now for something completely different. Suppose we compare a corpus of a million words with a list of one million zeroes. The corpus contains may different elements but the list contains only one element. We can describe the list with the phrase "a million zeroes" and everybody will know what we are talking about. For describing the corpus we would probably need a million words.

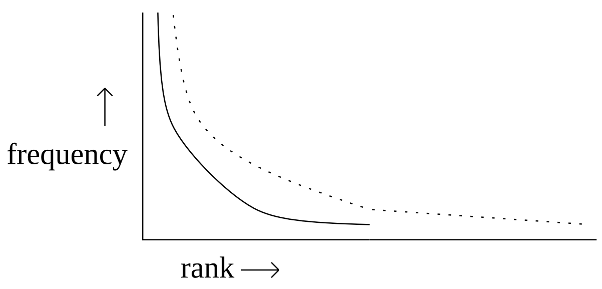

Figure 1: Graphs representing Zipf's law: word frequency multiplied with word rank in a frequency list is constant. The lower line represents the general case; the dotted line represents the distribution for a larger corpus. Note that the number of low frequency words is larger in the large corpus.

The differences between the two structures can be described as follows: the corpus contains more information than the number list.

We would like to represent the concept "information content" in a number. Information theory offers us a measure for doing that. The measure that is used for describing the information content of a message is called ENTROPY. The notation for entropy is H(W) where W is some stochastic variable. Entropy is defined as ([Cha93] page 29):

$$\mathrm{H(W)} = - \sum_w (P(w) * log_2(P(w))$$

So H(W) is the negation of the sum over all possible values w of W of the product of P(w), the probability of a value of W (a word), and its $\log_2$ value. The unity of H(W) is bits. An example:

> A stochastic variable can take the values a, b and c with probabilities P(a)=0.5, P(b)=0.25 and P(c)=0.25. This means that its entropy is equal to -(0.5*$\log_2$(0.5) + 0.25*$\log_2$(0.25) + 0.25*$\log_2$(0.25)) = -(-0.5+-0.5+-0.5) = 1.5 bits.

This is the computation of one instance of the stochastic variable (one word). The entropy of a sequence of stochastic variables, in our case a text, is the sum of the entropy of the words in the text. Note that this sum need not necessarily be equal to N times the entropy of a single word. Word probability depends on word context which causes words in context to have a different probability than words in isolation. An example: the Swedish preposition i might be the second most probable word but it is very unlikely that one will find the word when the previous word was some other preposition.

The entropy concept has many applications. Some examples: Speech messages with a large entropy will be more difficult to recognize than speech messages with a small entropy. Information streams with a small entropy are easier to compress than streams with a large entropy (example: text files and binary files). By the way, the entropy value of some information stream gives the average number of bits that is necessary for encoding the elements in the stream. And finally, the notion of cross entropy can be used for evaluating statistical models. Models with a low cross entropy are better than models with a large cross entropy. More on entropy and cross entropy can be found in [Cha93].

3.2 The noisy channel model

The noisy channel model is an approximation of a transmission channel and its influence on a transmitted message. We will use this model for part of speech tagging. However in this section the noisy channel model will be introduced as a method for correcting OCR input. We will start with an analogy with radio transmissions.

When a radio message in sent from point A to point B the observed message at point B will be slightly different from the original message. This difference in due to disturbances in the signal during transmission also called noise. The addition of noise to the message during the transmission can be simulated with the noisy channel model. This model contains three parts: the initial message (m), the observed message (o) and the channel model which is a collection of conditional probabilities P(o|m).

A person at point B who receives a radio message will know that the message contains some errors and will want to reconstruct the original message. We will try to define the reconstruction of the original message as a statistical question. We will assume that there are several possible original messages and we want to find the one that has the largest probability of being the original message for some specific observation. In other words we want to solve the equation [LS93]:

$$Max_m P(m|o)$$

This equation gives as a result the maximum value $P(m|o)$ can have for a any message m. We are interested in the message for which the probability obtains this maximal value. Getting the probability $P(m|o)$ is problematic. We can estimate the probability of some observation given some message $P(o|m)$ by sending the message through the channel many times and counting the number of times we the specific observation occurs. Doing the same for the probability of some message given some observation $P(m|o)$ is infeasible.

We can obtain the probability $P(m|o)$ by performing some arithmetic. In [FPPA91] section 13.4 we saw that the probability of two events A and B happening together will be equal the probability of A multiplied with the probability of B given that A happened. And the probability of B and A happening is equal the probability of B multiplied with the probability of A given that B happened. We can write down these equations and derive a formula for P(B|A) [LS93]:

1. $P(A\&B) = P(A) * P(B|A)$
2. $P(B\&A) = P(B) * P(A|B)$
3. $P(A) * P(B|A) = P(B) * P(A|B)$, derived from 1 and 2 since $P(A\&B) = P(B\&A)$
4. $P(B|A) = \frac{P(B)*P(A|B)}{P(A)}$, derived from 3

Equation 4 is called BAYES' RULE. We can use it for obtaining a a method for computing $P(m|o)$ given $P(o|m)$:

$$P(m|o) = \frac{P(m)*P(o|m)}{P(o)}$$

We will apply this formula for computing different $P(m|o)$ values for one particular observation o. We are only interested in how large these values are when compared which each other. This means that we may ignore the division by $P(o)$ which will be a constant since we work with one observation. So we end up with the following problem which we want to solve [LS93]:

> Find the message m for which $P(m) * P(o|m)$ has a maximal value because this will be the same message for which $P(m|o)$ has a maximal value.

We are left with two probabilities. The conditional probability $P(o|m)$ will be called the channel model. It gives the probability of an observation given a message. The other probability P(m) gives the probability of a message. We will call this one the language model [LS93].

Now let's look at an example. We have some channel which can be used for transmitting characters which are encoded in ASCII[1]. The channel has a problem: bit six of each character representation has a random value. In 50% of the transmission cases bit six will be a zero and in 50% it will be a one independent of which character is transmitted.

For the message ee this noisy channel will might generate the observations ee, eg, ge and gg. All observations are equally probable: 25%. In general this channel can generate 2^n different observations for a message of n characters and each of these observation will have probability 2^{-n}. The channel model can be defined as: $P(o_1...o_n|m_1...m_n) = 2^{-n}$.

We can use different language models. The most simple one is called the unigram model. This model assumes that the occurrences of the characters are independent of each other. In that particular case the probability of a message containing the characters $m_1...m_n$ will be equal to the product of all character probabilities $P(m_1) * ... * P(m_n)$. Since the characters are assumed to occur independently of each other we can maximize the probability of the message by maximizing the probabilities of the characters.

> An example: suppose we have the observation "maj". The first character is an m. The original character might have been an m or an o. According the the first file in the Press65 corpus the o is more common than the m so we change the m in an o. The second character is an a and could have been an a or a c. Since the a is more common than the c we leave the a there. Finally we have a j which could have been a j or an h. The latter is more common so we change the j in an h. So the most probable message for the observation "maj" is "oah".

We can use this statistical computation for creating a correction method. This method takes as input an observation and returns a message that is most likely to be the original message given some channel model and some language model. A correction method that uses an unigram language model will be called a unigram corrector. [LS93] reports that a unigram corrector can improve a text that was corrupted by the sixth bit noisy channel model from 50% correct characters to 83%.

The results of the correction method can be improved by using more elaborate language models, for example bigram or trigram models. These models include context of characters in their probability estimation. This approach will result better language models. For example: the probability of the character u in Swedish is 1.5%. However if we know that the previous character was q then

[1]Some ASCII codes for characters: e=01100101, g=01100111, i=0101001 and k=0101011. With the random sixth bit problem e might become g and vice versa while i might become k and vice versa. Note that the bit count started with 0.

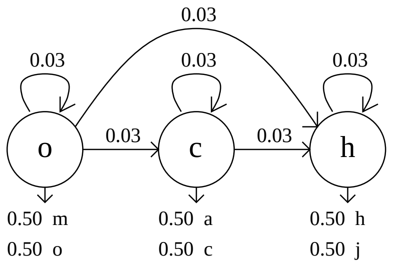

Figure 2: A part of an HMM that models the sixth bit flip problem and processes the message "och". Each character in the message has been represented by a state. Each state can output two characters with equal probability (0.50). Each state is equally likely to follow any other state (0.03).

the probability that we find a u increases to 88.2% (source: first file Press 65 corpus). By using context characters bigram and trigram correction methods have a better chance to guess to correct characters.

[LS93] reports that the 50% correct text could be improved to 90.5% correct by a bigram corrector and even to 96.8% by a trigram corrector. Working with sequences that are larger than three characters will be problematic because the frequencies of larger structures will be small and it will be difficult to build a good language model with them. Furthermore the number of different sequences will increase when we look at larger sequences and this can result in computational problems.

3.3 Hidden Markov Models

The application of bigram and trigram language models in a correction method will cause computational problems. In a unigram model the character occurrences were independent. Therefore we could retrieve the most probable message by computing the most probable character at each position in the message. This means that we have to perform n computations for a message containing n characters.

We cannot apply the same strategy when we are using bigram and trigram models since the occurrences of the characters have become dependent. The best sequence of some length might be less probable than some other sequence of the same length when considered in a larger context. This means that the only way to get the most probable message is to compute the probabilities of all possible messages and compare these. This is infeasible because of the large number of possible messages. For example, in the sixth bit flip channel model there are two possible characters at each position and this means that an observation of 100 characters would require $2^{100} = 1.3 * 10^{30}$ messages to be checked.

Fortunately there exists a method for computing the most probable message in a reasonable amount of time. This method is called the VITERBI ALGORITHM [TKS??]. This algorithm is part of a statistical model for sequences that is called a HIDDEN MARKOV MODEL. This model looks like a finite state automaton: it contains states that can output characters with transition links between the states. The difference with finite automata is that in Hidden Markov Models probabilities have been attached to the interstate links and the character productions by each state.

A Hidden Markov Model (HMM) can be used for message correction by representing the characters in the message by states and by using the output characters of the HMM as observation. In figure 2 you will find an HMM which models the transmission of the message "och" through a channel that contains the sixth bit flip problem. Any character is the message can be changed into some other character in the observation with a probability of 50%. The probability of each observation is equal to the product of the character probabilities multiplied with the product of the probabilities of the links used in the observation generation. Example: the probability of the sequence "maj" in the HMM part shown in figure 2 is equal to P(m|o) * P(o→c) * P(a|c) * P(c→h) * P(j|h) = 0.50*0.03*0.50*0.03*0.50 = 1.125%

The Viterbi algorithm makes use of the following assumption:

> For the computation of the most probable sequence of n characters in a message we only need the most probable messages of n-1 characters for any possible final character and the probabilities of these messages.

If we have an alphabet containing 26 characters, the computation of the most probable messages of 100 characters would progress as follows: First we compute the probabilities of all message of one character (26 computations). Then we use these results for computing the most probable messages of two characters for any final character (26*26 computations). We continue like this until we have the most probable messages of 99 characters for any final character. We can use these messages for computing the most probable message of 100 characters (26*26 computations). The result is that we have performed approximately 100*26*26 computations which is a lot less than 26^{100} (the number of possible messages of 100 characters).

The Viterbi algorithm ables us to compute the most probable sequence much faster than by computing and comparing the probabilities of all sequences. It will use the observation to restrict the number of relevant sequences and cut down the number of computations even further. Hidden Markov Models contain two more interesting algorithms which we will not discuss in great detail. The forward-backward procedure is used for computing the probability of some observation in an effective way. The Baum-Welch algorithm is the HMM training algorithm; it can be used for estimating the internal HMM probabilities based on data.

3.4 Application to POS Tagging

Now let's assume the following message transmission situation: Somewhere someone is sending a message which only contain part of speech tokens. The message passes through a noisy channel and by miracle it has changed into a comprehensive text when it reaches the other end of the channel. Thus we have a text as observation and a sequence of corresponding part of speech tags as message. Our task is to find back the tags given the text. This is the same task as in part of speech tagging. This is how the noisy channel can be used as a model for a POS tagger.

In the application of the noisy channel model in POS tagging we need to maximize the function P(m|o) = P(tags|words). If we have a tagged corpus we can use it for estimating the probability of tag sequences giving word sequences. This means that we do not have to use the Bayes rule here. The computation of the probability P(tags|words) is usually split in several computations given some assumption. Examples:

1. $P(t_1...t_n|w_1...w_n) = P(t_1|w_1) * ... * P(t_n|w_n)$ (unigram assumption)
2. $P(t_1...t_n|w_1...w_n) = P(t_1|w_1) * P(t_2|t_1, w_2) * ... * P(t_n|t_{n-1}, w_n)$ (bigram assumption)
3. $P(t_1...t_n|w_1...w_n) = P(t_1|w_1) * P(t_2|t_1, w_2) * P(t_3|t_1, t_2, w_3) * ... * P(t_n|t_{n-1}, t_{n-2}, w_n)$ (trigram assumption)

With the unigram assumption we are assuming that the tag of a word will only be dependent of the current word. The result of this is that we will choose the most frequent tag for each word. This will lead to approximately 90% correct tags (no reference available). The bigram and the trigram assumption are a little bit more advanced. They assume that the current tag is not only dependent on the current word but also on the previous tag (bigram) or both the previous and the next-to-previous tag (trigram). We can also make the tags dependent on previous words instead of previous tags but this will make it harder to collect the necessary conditional probabilities.

The POS tagging approach sketched here is a one-strategy approach. In reality many other components can be added to the tagging process like lexical and morphological analysis [MW96]. The statistical tag computation part is important but the tagging results can benefit from these other components.

References

[Cha93] Eugene Charniak. *Statistical Language Learning.* MIT Press, 1993.

[FPPA91] David Freedman, Robert Pisani, Roger Purves, and Ani Adhikari. *Statistics.* W.W. Norton & Company, 1991. ISBN 0-393-96043-9.

[LS93] Mark Liberman and Yves Schabes. *Statistical Methods in Natural Language Processing.* handout supplied at the EACL93 conference, Utrecht, The Netherlands, 1993.

[MW96] Tony McEnery and Andrew Wilson. *Corpus Linguistics.* Edinburgh University Press, 1996.

[TKS??] Erik F. Tjong Kim Sang. *Machine Learning of Phonotactic Structure.* PhD thesis University of Groningen. To be published in this millenium.