2020

TnT – A Statistical Part-Of-Speech Tagger

GROUP MEMBERS

FAISAL RIAZ - F2019313020 | TALHA RASOOL - F2019313039 | WASEEM ABBAS - F2019313001

Contents

Review of "TnT – A Statistical Part-Of-Speech Tagger"

Abstract:

This paper presents evaluation of statistical part of speech tagging method using second order Hidden Markov Model. One of the fundamental tasks in natural language processing is part of speech (POS) tagging. A POS tagger is a piece of software that reads text in some language and assigns a part of speech tag to each one of the words. Predicting the part of speech (POS) tag of an unknown word in a sentence is a significant challenge. This is particularly difficult where POS tags serve as an input to training sophisticated literature summarization techniques, such as those based on Hidden Markov Models (HMM). Different approaches have been taken to deal with the POS tagger challenge, but with one exception – the ***TnT, A Statistical part of speech Tagger*** the technique we are reviewing presented in this paper. Previously publications on POS tagging have omitted details of the suffix analysis used for handling unknown words. In this Paper author used a simple technique based on probabilistic model as described with inclusion of other techniques of smoothing, handling of unknown words, capitalization and Beam search.

Introduction

We would like to briefly explain the Parts of speech tagging and HMM before formerly introducing this paper.

Parts-Of-Speech Tagging

In corpus linguistics, part-of-speech tagging (POS tagging or PoS tagging or POST), also called grammatical tagging or word-category disambiguation, is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech, based on both its definition and its context — i.e., its relationship with adjacent and related words

in a phrase, sentence, or paragraph. A simplified form of this is commonly taught to school-age children, in the identification of words as nouns, verbs, adjectives, adverbs, etc. (Wikipedia, n.d.)

Identifying part of speech tags is much more complicated than simply mapping words to their part of speech tags. This is because POS tagging is not something that is generic. It is quite possible for a single word to have a different part of speech tag in different sentences based on different contexts. That is why it is impossible to have a generic mapping for POS tags. Let's have a look at the following example

"They refuse to permit us to obtain the refuse permit".

The word refuse is being used twice in this sentence and has two different meanings here. refUSE (/rəˈfyo͞oz/)is a verb meaning "deny," while REFuse(/ˈrefˌyo͞os/) is a noun meaning "trash" (that is, they are not homophones). Thus, we need to know which word is being used in order to pronounce the text correctly. (For this reason, text-to-speech systems usually perform POS-tagging.)

As you can see, it is not possible to manually find out different part-of-speech tags for a given corpus. New types of contexts and new words keep coming up in dictionaries in various languages, and manual POS tagging is not scalable in itself. That is why we rely on machine-based POS tagging.

Word Sense Disambiguation

Words often occur in different senses as different parts of speech. For example:

"She saw a bear."

"Your efforts will bear fruit."

The word bear in the above sentences has completely different senses, but more importantly one is a noun and other is a verb. Rudimentary word sense disambiguation is possible if you can tag words with their POS tags. Word-sense disambiguation (WSD) is identifying which sense of a word (that is, which meaning) is used in a sentence, when

the word has multiple meanings. As we can clearly see, there are multiple interpretations possible for the given sentence. Different interpretations yield different kinds of part of speech tags for the words. This information, if available to us, can help us find out the exact version / interpretation of the sentence and then we can proceed from there. It is very important to know what specific meaning is being conveyed by the given sentence whenever it's appearing. This is word sense disambiguation, as we are trying to find out THE sequence. (Godayal, n.d.)

Hidden Markov Model (HMM)

is a statistical Markov model in which the system being modeled is assumed to be a Markov process – call it X – with unobservable ("hidden") states. HMM assumes that there is another process Y whose behavior "depends" on X. The goal is to learn about X by observing Y. (Wikipedia, n.d.)

HMMs for Part of Speech Tagging

We know that to model any problem using a Hidden Markov Model we need a set of observations and a set of possible states. The states in an HMM are hidden. In the part of speech tagging problem, the observations are the words themselves in the given sequence. As for the states, which are hidden, these would be the POS tags for the words.

The transition probabilities would be somewhat like P (VP | NP) that is, what is the probability of the current word having a tag of Verb Phrase given that the previous tag was a Noun Phrase.

Emission probabilities would be P(john | NP) or P(will | VP) that is, what is the probability that the word is, say, John given that the tag is a Noun Phrase. Note that this is just an informal modeling of the problem to provide a very basic understanding of how the Part of Speech tagging problem can be modeled using an HMM. (Godayal, n.d.)

Second Order Hidden Markov Model

The second-order HMM is an extension to the first-order HMM which contains additional information about previous activity that leads to better accuracy results. Second-order HMM relies on one observation in a current state and the transition probability function based on two previous states.

Trigrams n Tags

Hidden Markov Models (HMM) have been used in Part-Of Speech (POS) tagging of text for 30 years. Since this paper was presented 20 years ago so author compared the different techniques presented at that time, he compared different models rule based and stochastic models to compare his results.

Among the statistical approaches used at that time, the Maximum Entropy framework (Ratnaparkhi, 1997) had a very strong position. Author used the results performed by an independent comparison of 7 taggers (Daelemans, 1999)

Author had been quite successful in proving that Markov models combined with a good smoothing technique and with handling of unknown words. This tagger, TnT, not only yielded the highest accuracy, it also was the fastest both in training and tagging.

This paper describes the models and techniques used by TnT together with the implementation. We are surprised how simple the underlying model is. The result of the tagger comparison seems to support the "the simplest is the best". However, in this paper Author had clarified a number of details that were omitted in major previous publications concerning tagging with Markov models. As two examples, (Rabiner, 1989) and (Eugene Charniak, 1993) gave good overviews of the techniques and equations used for Markov models and part-of speech tagging, but they were not very explicit in the details.

Working of TnT POS

If a sentence of length T, contains words/tokens w_1, w_2…,w_T and POS tags for them are t_1, t_2….. ,t_T then according to the topology of the HMM the joint probability of this combination will be:

$$\underset{t_1 \ldots t_T}{\operatorname{argmax}} \left[\prod_{i=1}^{T} P(t_i|t_{i-1}, t_{i-2}) P(w_i|t_i) \right] P(t_{T+1}|t_T) \quad (1)$$

The first term is $P(t_i \mid t_i\text{-}1, t_i\text{-}2)$, and suggests that each word tag depends on 2 previous tags. This is known as a 3-gram HMM and The second term is $P(w_i \mid t_i)$, which determines the word probability distribution given a POS tag, and refer to it from now on as the word conditional probability. This conditional probability shows that the probability of one observation (word) only depends on its current state (tag), not on previous or subsequent states. Third term $P(t_{T+1} \mid t_T)$ is the probability of that tag appearing at a particular position i.e at the beginning or the end of the sequence. tags $t-1$, $t0$, and t_{T+1} are beginning-of-sequence and end-of-sequence markers. Using these additional markers even if the input token contains punctuation marks, those can be stemmed. This improves the tagging results. TnT would add these markers if not specified in input sequence, if it encounters one of the punctuation marks.

To estimate the transition and output probabilities, needed first to process the training corpus. Frequencies for each token that exist in the training corpus is calculated. Subsequently, for each word in lexicon maximum likelihood estimation is determined as $\hat{P}$

$$\text{Unigrams:} \quad \hat{P}(t_3) = \frac{f(t_3)}{N} \tag{2}$$

$$\text{Bigrams:} \quad \hat{P}(t_3|t_2) = \frac{f(t_2, t_3)}{f(t_2)} \tag{3}$$

$$\text{Trigrams:} \quad \hat{P}(t_3|t_1, t_2) = \frac{f(t_1, t_2, t_3)}{f(t_1, t_2)} \tag{4}$$

$$\text{Lexical:} \quad \hat{P}(w_3|t_3) = \frac{f(w_3, t_3)}{f(t_3)} \tag{5}$$

Where t1, t2, t3 in the tagset and w3 in the training corpus and N is the total number of tokens in the training corpus. Eq 2 calculates the MLE of a tag t_3 in question in a corpus of N tokens. Eq 3 Calculates the probability of a tag t_3 given that tag t_2 has occurred (Bigram). Eq 3 calculates the probability of a tag t_3 given that tag t_1 and t_2 has occurred (Trigram). Eq 5 calculates the probability of a token w_3 subjected to its occurrence with a particular tag t_3. Author has considered MLE / Probability to be 0 if in any of the above equations from 2 to 5 if numerator or denominator is 0.

Smoothing Technique

If you observe closely than Trigrams would run you into a problem. You cannot use the probabilities generated from the corpus directly. Why? because the output of Trigrams would have sparse data (a lot of trigrams with 0 probability). Practical implication of this result is that we cannot estimate the probabilities correctly. Here we face another problem what if a particular trigram never occurred in the corpus then its probability would be zero thus making the probability of a whole new sentence to be 0 if that trigram with 0 probability has occurred and is necessary in a new word sequence. How we are going to rank different sequence containing 0 probability.

Author has come up with a solution to that problem with a smoothing paradigm that produced the best results in TnT was linear interpolation[i] . Therefore, author calculated the probability of the trigrams as

$$P(t_3|t_1,t_2) = \lambda_1\hat{P}(t_3) + \lambda_2\hat{P}(t_3|t_2) + \lambda_3\hat{P}(t_3|t_1,t_2) \quad (6)$$

Where $\hat{P}$ is the MLE of the probabilities calculated previously and t1, t2, t3 in the tag set of the training corpora. Author has specified a condition here $\lambda_1 + \lambda_2 + \lambda_3 = 1$, since probability is between 0 and 1 always, so P is again here representing the probability distribution.

If we want to linearly interpolate values between two points for example then this interpolant is dependent on the starting values and end value. Applying the same concept here interpolant values should be context dependent. i.e. values of λs should be calculated for each trigram in corpora. Author has used context independent variant of linear interpolation. Contrary to the intuition of the Author the results he obtained were better than context dependent variant.

Since we have previously discussed the sparse data output for trigrams, for the same reason it is not feasible to calculate a set of λs for each trigram. In order to overcome this issue Author has again come up with a smart solution. Author grouped trigrams on the basis of the frequency of the trigrams and estimated and tied λs to each frequency group. This method has its own shortcomings that some of the trigram frequency groups yielded results as good as compared to results with context free linear interpolation and some frequency groups yielded worse results. This frequency grouping technique was not used before in any known research at that time.

Tested frequency groups were assigned set of λs according to following rule

- One set of λs for each frequency value
- Two classes on low and high frequency values on the two ends of the scale.
- Several frequency groupings in between them and several setting for partitioning classes.

Algorithm used by the author to estimate the set of λs is given below:

```
set λ1 = λ2 = λ3 = 0
foreach trigram t1, t2, t3 with f(t1, t2, t3) > 0
    depending on the maximum of the following three values:
        case (f(t1,t2,t3)−1)/(f(t1,t2)−1):  increment λ3 by f(t1, t2, t3)
        case (f(t2,t3)−1)/(f(t2)−1):  increment λ2 by f(t1, t2, t3)
        case (f(t3)−1)/(N−1):  increment λ1 by f(t1, t2, t3)
    end
end
normalize λ1, λ2, λ3
```

Figure 1: Algorithm for calculting the weights for context-independent linear interpolation $\lambda_1, \lambda_2, \lambda_3$ when the n-gram frequencies are known. N is the size of the corpus. If the denominator in one of the expressions is 0, we define the result of that expression to be 0.

Author has applied the deleted interpolation technique to determine the values of λ_1, λ_2 *and* λ_3 by removing each trigram one by one from the training corpus and estimate the best values of λs from all other n-grams in the corpus. Algorithm calculate the values of λs but subtracting 1 from each frequency value because if you train the model for each trigram and calculate its relative weight than you are over training the model , that's why author has subtracted 1 from each value here to make a bias -variance trade off. Author discussed that if you don't consider the margin for an unknow trigram than model would produce worse results. Since λ coefficients should not be greater than 1 so all coefficients/weights are normalized.

Handling of Unknown Words

Best method known to handle unknown words at that time worked best for inflected languages[ii] such as English and German was proposed by (Samuelsson, 1993). Tag

probabilities of unknown words are estimated according to the letters where word is ending. In English language suffix is the strong predictor of the class of the word.

In a study conducted for the words in Wall Street Journal which is a part of Penn Treebank, words (e.g. fashionable) ending with "able" are adjectives (JJ) 98% of the total words observed. Whereas, only 2% were nouns (e.g. cable, variable). Now we will explain the methodology adapted by the Author to estimate the probabilities of unknown words below:

Probability distribution of a particular suffix is generated from all the words in the training set that share the same suffix of some predefined maximum length.

"Suffix here is the final sequence of characters which is not necessarily a linguistically a meaningful suffix."

Let us consider an example of word "smoothing"

e.g: **smoothing**

g
ng
ing
hing
thing
othing
oothing
moothing
smoothing

This calculates the probability of a tag t given the last m letters li of an n letter word:

P(t | ln−m+1, . . . ln). The sequence of increasingly more general contexts omits more and more characters of the suffix, such that P(t | ln−m+2, . . . , ln), P(t | ln−m+3, . . . , ln), . . . , P(t) are used for smoothing. As we have shown in the above for the case of smoothing.

Given suffix length: $i = m$ to 0 (7)

$$P(l_{n-i+1}, \ldots l_n|t) \propto P(t|l_{n-i+1}, \ldots l_n)P(t)$$

Author has determined the maximum length of suffix i.e m =10, here probability for each letter for i = 10 to 0 is calculated for given that a tag t has occurred. Since Author is interested in determining the tag t for an unknown word so he has used Bayesian inversion here as shown above.

So for n length of an unknown word and suffix i = m to 0, tag t is calculated from training data and here $P(t) = \hat{P}(t)$,Where $\hat{P}(t)$ is the MLE / Probability we obtained from the frequencies in the lexicons.

Author has used recursive formula to determine the tags of the unknown words as shown below:

$$P(t|l_{n-i+1}, \ldots l_n) = \frac{\hat{P}(t|l_{n-i+1},\ldots l_n)+\theta_i P(t|l_{n-i},\ldots l_n)}{1+\theta_i} \quad (8)$$

Where the maximum likelihood estimate for suffix of length i is derived from corpus frequencies by following formula

$$\hat{P}(t|l_{n-i+1}, \ldots l_n) = \frac{f(t,l_{n-i+1},\ldots l_n)}{f(l_{n-i+1},\ldots l_n)} \quad (9)$$

RHS is the frequency of intersection of tag t with the n length of characters token from i = 10 to 0 divided by the frequency of the n length of character token from i = 10 to 0.

$$\theta_i = \frac{1}{s-1}\sum_{j=1}^{s}(\hat{P}(t_j) - \overline{P})^2$$
$$\overline{P} = \frac{1}{s}\sum_{j=1}^{s}\hat{P}(t_j) \qquad 10$$

Since Author has adopted this method from (Samuelsson, 1993) which used a theoretical motivated argumentation and used the standard deviation of the maximum likelihood probabilities $\hat{P}$ for the weights θ_i .

Author has identified a good values of m , where m depends on the token/word in question. Author has identified longest suffix with frequency > 1 but length not more than 10 .

Author used the context independent linear interpolation again to determine the value of θ_i which has been used previously to determine the coefficients λ_i.

Let us explore the equations labeled as eq 10, $\bar{P}$ is the average of probability calculated by the sum of the probabilities for each tag t in the set of tags.

Weights θ_i is calculated by difference of the square of MLE / probability for each tag t calculated initially as $\hat{P}$ for each tag t in the set of tags s and then dividing it with s -1 to cater the weight for unknown tokens. Author has determined that range for weights θ_i is in between 0.03 ….0.10. Since the unknown word may or may not be a capitalize. Author has used different set of suffix tries depending upon the capitalization to improve the tagging results.

Here is another problem pointing to the choice of tokens/words in lexicon. Should we use all words or fix words for suffix handling in the lexicons. Author has assumed here that unknown words are those which are appearing less frequent in the lexicons and using the suffixes of infrequent words but with a threshold of frequency less than or equal to 10. This choice turned out to be a good choice.

Capitalization

Capitalization plays a vital role in disambiguation process for different corpora and tag sets. Tags are not usually informative about capitalization, but the probability distributions of tag around capitalized words are different from those not capitalized. This effect becomes quite prominent for English and German language. In English only the starting word is capitalized for proper Nouns whereas, in German all the letters are capitalized for proper nouns. In order to cater the capitalize words following condition is defined by Author.

$$c_i = \begin{cases} 1 & \text{if } w_i \text{ is capitalized} \\ 0 & \text{otherwise} \end{cases}$$

Now these flags are added to the contextual probability distributions. Instead of

$$P(t_3|t_1, t_2) \quad (12)$$

He used

$$P(t_3, c_3|t_1, c_1, t_2, c_2) \quad (13)$$

This condition implies that in order to estimates the MLE / probability of the capitalized tokens we have to update equation (3) to (5) incorporating the flag C_i. Now this will double the size of the tag set and to use 2 different tag sets. One tag set for the not capitalized tokens and another tag set for capitalized tokens.

Beam Search

Let us comprehend the concept of Viterbi Algorithm before discussing Beam Search.

Viterbi Algorithm

Let us consider a simple example where we have to Tag only 4 tokens and number of Tags in tag set are 3. For each token we have 3 possible tags so total number of possible

sequences would be $3^4 = 81$. It would be difficult to find the best possible sequence with highest probability.

So to overcome this problem and find the best sequence with highest probability we use Viterbi Algorithm. "***Viterbi Algorithm***" for every possible state in HMM (Tags in our case here) record the highest probability path that ends in that state known as Viterbi Algorithm.

Viterbi algorithm is linear in the size of the input, it more accurate to say that the complexity of Viterbi

algorithm is proportional to N^2*T, where N is the size of the network and T is the length of the input. When N gets large, the full algorithm rapidly becomes impractical. For instance, consider a relatively simple trigram part-of-speech model with 40 different tags. Each state represents a pair of tags, so we have 40^2 (=1600 states). Thus, to fully explore every possible next state in the Viterbi algorithm we would need to look at 40^3 (64,000) possibilities for each term in the input. (James, n.d.)

Beam Search for Large Networks:

The simplest idea is to rank all the alternatives produced so far by their probabilities, and always just expand the most likely sequences (with highest probability). This is called a ***beam search***. We continue to do this at each time point until we have an interpretation that covers the entire output. This is a simple modification to the basic Viterbi Algorithm. A beam search prunes out the unpromising states at every step, leaving only the best to be used in future calculations. We can decide how many to keep using several different methods. We could, for instance, decide on a fixed number to keep at each step, say B nodes. Alternately, we could decide on some factor K and delete all states that have a probability less that m*K, where m is the maximum probability for a state at the current time. For instance, if K=.5, we would only keep hypotheses that have a score greater than half the maximum score at that time. (James, n.d.)

Author's approach for Beam Search

The processing time for Viterbi Algorithm can be reduced by introducing a beam search. Each state that receives a δ value smaller than the largest δ divided by some threshold value θ is excluded from further processing. Since we know Viterbi Algorithm is guaranteed to find the best sequence of states with highest probability will no longer hold after applying the beam search. Which means Beam Search may or may not produce best sequence of stats. Author has claimed here that if you carefully identify the best value θ then it can perform as good as Viterbi Algorithm. Author has used $\theta = 1000$ which doubled the speed of tagger. Author achieved tagging speed of tags between 30,000 and 60,000 tokens per second (including file I/O) on a Pentium 500 running Linux. This speed was highly dependent on the percentage of unknow words and average ambiguity rate.

Model Evaluation and Experiments Strategy

We have carefully studied and outline the strategy adopted by the Author as below:

- Accuracy of tagger was evaluated for ten iterations and then averaged out as the overall accuracy. In addition to that separate accuracies for known and unknown words were also measured.
- Tagger was trained over corpora of different sizes ranging from 1000 token to the entire corpus.
- Tagger assigned tags to tokens with its respective probability, Author uses these assignments to differentiate between reliable assignments and unreliable assignments of tags. Tags assignments are ranked on the quotient [iii]value greater than a threshold. Alternative tag assignment is calculated as $t_{alt} = p(t_{best})/ p(t_{alt})$.
- Author has used 90/10 Train, Test ratio.
- Tagger was trained with 10 Fold cross validation and results were averaged out for single outcome.

- Author has used contiguous test sets in all experiments, alternative to that is round robin presenting 10th sentence every time. He argued that contiguous test sets achieved higher accuracy when a completely unseen article was tagged.
- Accuracy is calculated by dividing the no of correctly assigned tags by total no of tokens in the corpus processed.

Corpus used for Evaluation

Author has used two corpora for all the outlined experiments and model evaluation. Which are

1- German Negra corpus
2- Penn Treebank

We will describe how author experimented with these two corpora in detail.

The German Negra Corpus

German Negra corpus consists of 20,000 sentences and approximately 355,000 tokens of newspaper texts. It was developed at the Saarland University. Author has only used the part of speech annotation in this evaluation.

Tagging accuracies achieved by TnT for Negra Corpus are highlighted in Table2.

Table 2: Part-of-speech tagging accuracy for the NEGRA corpus, averaged over 10 test runs, training and test set are disjoint. The table shows the percentage of unknown tokens, separate accuracies and standard deviations for known and unknown tokens, as well as the overall accuracy.

	percentage unknowns	known acc.	σ	unknown acc.	σ	overall acc.	σ
NEGRA corpus	11.9%	97.7%	0.23	89.0%	0.72	96.7%	0.29

Learning curve of the tagger is explained in Figure 3 shown below:

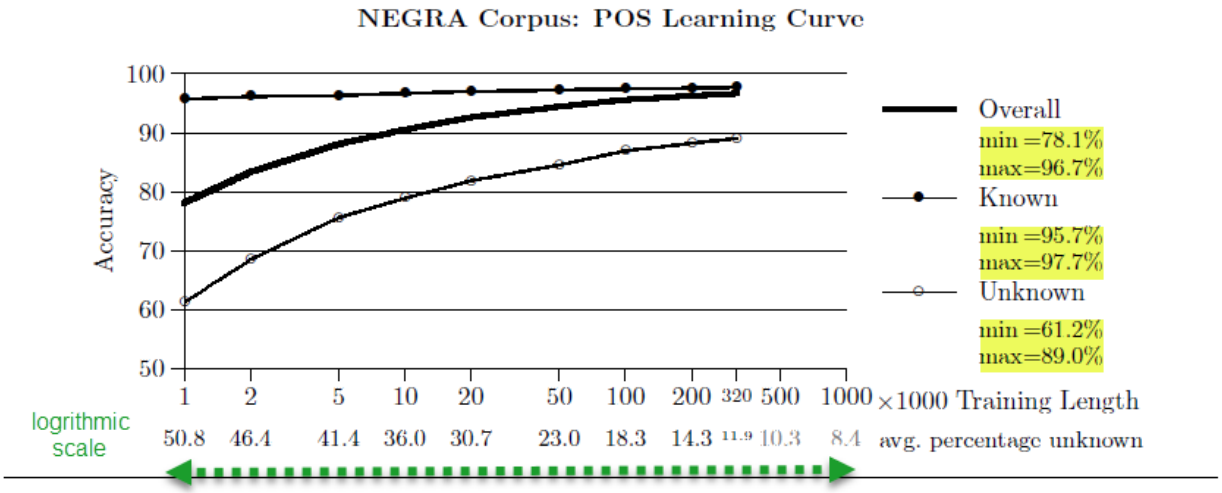


Figure 3: Learning curve for tagging the NEGRA corpus. The training sets of variable sizes as well as test sets of 30,000 tokens were randomly chosen. Training and test sets were disjoint, the procedure was repeated 10 times and results were averaged. Percentages of unknowns for 500k and 1000k training are determined from an untagged extension.

Figure 3 is explaining the learning curve based on the training data. It is quite obvious that accuracy is directly proportional to the amount of training data. Training length is the number of tokens and each training length was tested ten times and results were averaged out. Author observed that tagging accuracy is very high for known words, which considerably improved the chances to correctly tag unknown words. If tagger had encountered that token only once during the training.

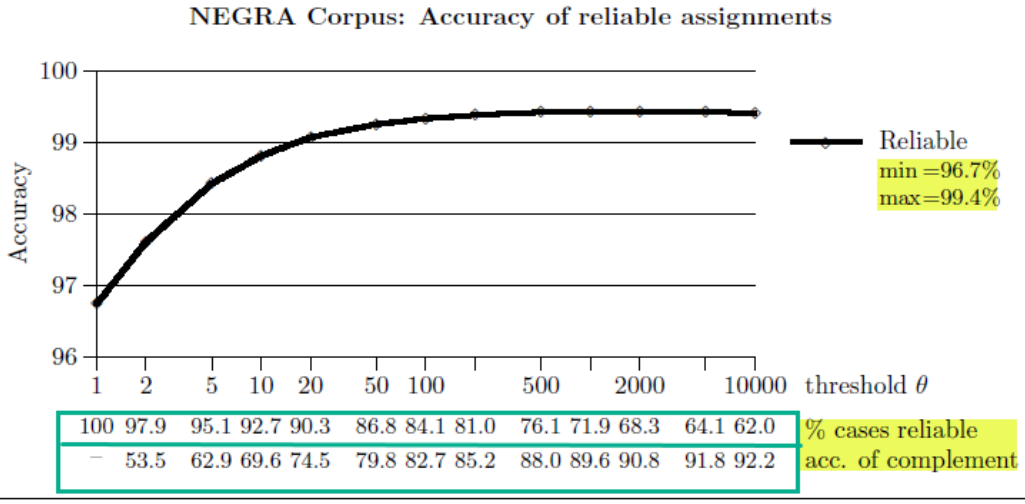


Figure 4: Tagging accuracy for the NEGRA corpus when separating reliable and unreliable assignments. The curve shows accuracies for reliable assignments. The numbers at the bottom line indicate the percentage of reliable assignments and the accuracy of the complement set (i.e., unreliable assignments).

Figure 4 explains the accuracy of tagger when assignments are separated by quotients larger and smaller than threshold θ . Reliable and unreliable assignments. As expected reliable assignments are much higher than unreliable assignments. This is quite useful for corpus annotation projects if the best tag is classified as unreliable.

The Penn Treebank

An important tag set for English is the 45-tag Penn Treebank tag set, which has been used to label many corpora. Author has used WSJ (Wall Street Journal) as contained in Penn Treebank. WSJ consists of approx. 50,000 sentences and 1.2 million tokens. Corpus's annotation consists of four parts. Author has only used Part of speech annotation.

Tagging accuracies achieved by TnT for WSJ are highlighted in Table5.

Table 5: Part-of-speech tagging accuracy for the Penn Treebank. The table shows the percentage of unknown tokens, separate accuracies and standard deviations for known and unknown tokens, as well as the overall accuracy.

	percentage unknowns	known acc.	σ	unknown acc.	σ	overall acc.	σ
Penn Treebank	2.9%	97.0%	0.15	85.5%	0.69	96.7%	0.15

Learning curve of the tagger with Penn Treebank is explained in Figure 6 shown below:

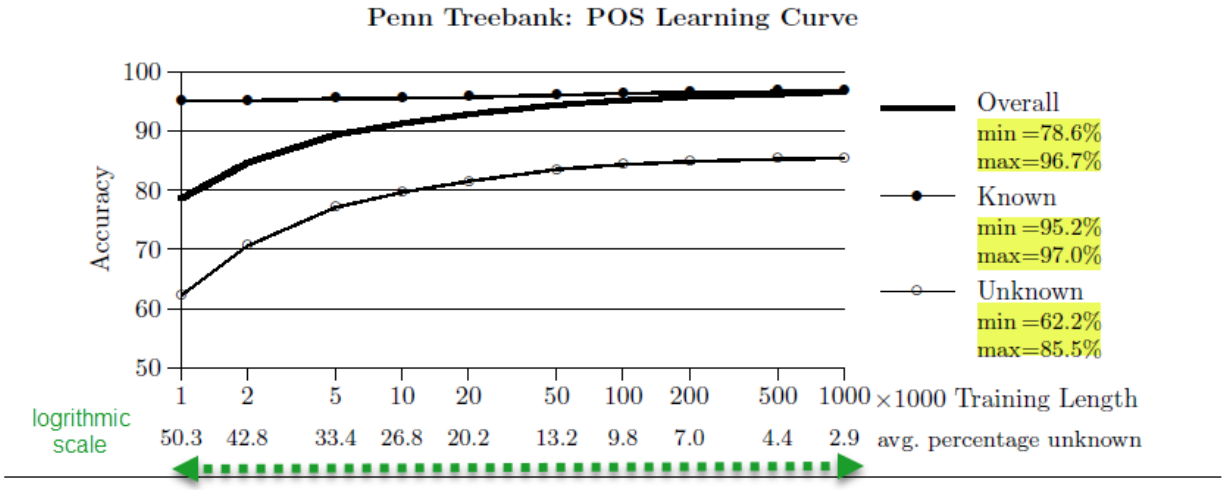


Figure 6: Learning curve for tagging the Penn Treebank. The training sets of variable sizes as well as test sets of 100,000 tokens were randomly chosen. Training and test sets were disjoint, the procedure was repeated 10 times and results were averaged.

Figure 6 is explaining the learning curve based on the training data. It is quite obvious that accuracy is directly proportional to the amount of training data. Training length is the

number of tokens and each training length was tested ten times and results were averaged out. Author observed that tagging accuracy is very high for known words, which considerably improved the chances to correctly tag unknown words. If tagger had encountered that token only once during the training.

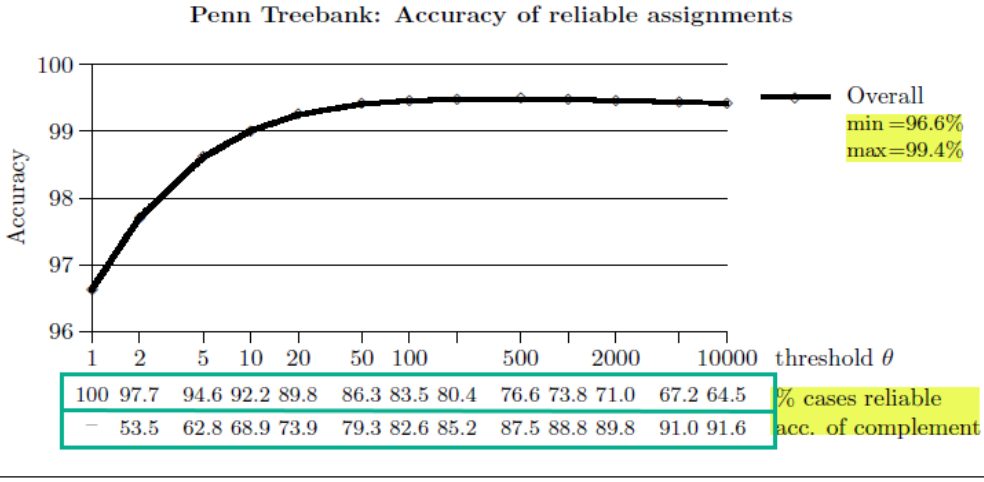


Figure 7: Tagging accuracy for the Penn Treebank when separating reliable and unreliable assignments. The curve shows accuracies for reliable assignments. The numbers at the bottom line indicate the percentage of reliable assignments and the accuracy of the complement set.

Figure 7 explains the accuracy of tagger when assignments are separated by quotients larger and smaller than threshold θ . Reliable and unreliable assignments. As expected, reliable assignments are much higher than unreliable assignments. This is quite useful for corpus annotation projects if the best tag is classified as unreliable.

Results and conclusion

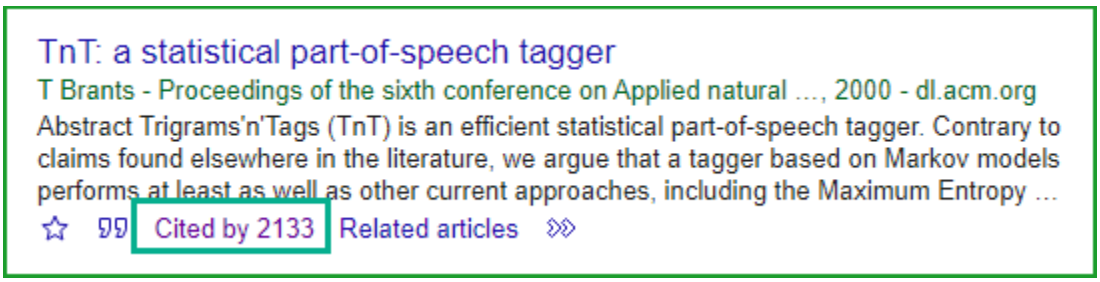
Average part of speech tagging achieved with TnT is between 96% to 97%. Accuracies slightly varies depending upon the language and tagset used . TnT achieved atleast equivalent or better results with state of the art tagger present at that time. (Ratnaparkhi, 1997) achieved the accurarcy of 96.6% using Maximum Entropy Method with Penn Treebank only for one iteration. Whereas , TnT achieved better and faster results with much simpler method moreover , these results were averaged over ten iterations.

Author achieved 8.7% higher accuracy for the German Newspaper for the lexicons that was known to model during the training. If now known than accuracy dropped from 97.7

% to 89%. It was observed that accuracy of the model was pretty high 95% to 96% for a very small training set of 1000 words. Author has used the stochastic taggers advantage to identify reliable and unreliable assignments. Author claimed to achieve accuracy of 99% for a subset during the processing.

Author achieved state of the art results with Hidden Markov model contrary to the claims found in the literature. Basic Algorithm presented earlier left several decisions to the implementors and left many questions unanswered. Author clearly explained and outlined the procedure for handling of start and end sequence, exact smoothing technique , how to determine the weights for context probabilities, detailed procedures for handling unknown words. Author not only achieved good results for German and English Corpus but model also performed well for other set of languages. Author achieved a generalized architecture that can be applied to large variety of languages.

Contributions of TnT Model



One the milestone tagger in the history of Natural language processing and paved path for many other researches. The biggest advantage of this model is that it can be applied to variety of languages and domain specific lexicons such as biomedical fields (Mario Fruzangohar, October 2013). Same model was used for the part of speech tagging for Persian language (Seraji, 2011). An HMM based part-of-speech tagger and statistical chunker for 3 Indian languages (GMR Sastry, 2007). Statistical Part of speech tagger for urdu (Anwar, Wang, Li, & Wang, 2007). This paper had quite and impact as you can see from the number of citations. Table in the following figure show the comparison of

tagging accuracy of different proposed model this model significantly performed better than others. (Vishaal Jatav, 2017)

System	Languages/Tags	Accuracy
[Béchet, 2000]	NP Evaluation	72.6
[Brants, 2000]	All Tags	96.7
[Choi, 2012]	All Tags	93.05
[Church, 1988]	All Tags	99.5
[Dandapat, 2007]	All Tags	88.41
[Das, 2011]	Multi-lingual All Tags	83.4
[Dredze, 2008]	Islandic All Tags	91.54
[Duong, 2013]	Multi-lingual All Tags	83.4
[Giménez, 2004]	Multi-lingual All Tags	96.46
[Hajič, 2001]	Czech All Tags	95.16
[Haulrich, 2009]	All Tags	96.37
[Hepple, 2000]	All Tags	97.35
[Huihsin, 2005]	Multi-lingual All Tags	93.7
[Kim, 2003]	All Tags	96.9
[Lee, 2000]	All Tags	97.93

Table 1. (Vishaal Jatav, 2017)

Issues

Accuracy discussed in this model are calculated on per token basis. Author has not explained anything about the sentence level accuracy. At first glance, current part-of-speech taggers work rapidly and reliably, with per-token accuracies of slightly over 97% [1–4]. Looked at more carefully, the story is not quite so rosy. This evaluation measure is easy both because it is measured per-token and because you get points for every punctuation mark and other tokens that are not ambiguous. It is perhaps more realistic to look at the rate of getting whole sentences right, since a single bad mistake in a sentence can greatly throw off the usefulness of a tagger to downstream tasks such as dependency parsing. Current good taggers have sentence accuracies around 55–57%, which is a much more modest score .Tables shown below is taken from (Manning, 2011) which shows the tagging accuracies and POS Tagger errors on WSJ developmental set.

Model	Feature Templates	# Feats	Sent. Acc.	Token Acc.	Unk. Acc.
3GRAMMEMM	See text	248,798	52.07%	96.92%	88.99%
NAACL 2003	See text and [1]	460,552	55.31%	97.15%	88.61%
Replication	See text and [1]	460,551	55.62%	97.18%	88.92%
Replication′	+rareFeatureThresh = 5	482,364	55.67%	97.19%	88.96%
5W	$+\langle t_0, w_{-2}\rangle, \langle t_0, w_2\rangle$	730,178	56.23%	97.20%	89.03%
5WSHAPES	$+\langle t_0, s_{-1}\rangle, \langle t_0, s_0\rangle, \langle t_0, s_{+1}\rangle$	731,661	56.52%	97.25%	89.81%
5WSHAPESDS	+ distributional similarity	737,955	56.79%	97.28%	90.46%

Tagging Accuracies on WSJ Development Set

Class	Frequency
1. Lexicon gap	4.5%
2. Unknown word	4.5%
3. Could plausibly get right	16.0%
4. Difficult linguistics	19.5%
5. Underspecified/unclear	12.0%
6. Inconsistent/no standard	28.0%
7. Gold standard wrong	15.5%

Frequency of different POS tagging error types

TnT Statistical Part of speech tagger did not perform well on morphological rich languages like Icelandic language. Solution was proposed by (Loftsson, 2007) by filling in the gaps in the lexicon using language morphological analyzer.

(Loftsson, 2007) described his linguistic rule-based tagger Ice Tagger, and compare its tagging accuracy to the TnT tagger, a state-of-threat statistical tagger, when tagging Icelandic, a morphologically complex language. Evaluation shows that the average tagging accuracy is 91.54% and 90.44%, obtained by Ice Tagger and TnT, respectively. When tag profile gaps in the lexicon, used by the TnT tagger, are filled with tags produced by our morphological analyzers Ice Morphy, TnT's tagging accuracy increases to 91.18%.

References

Anwar, W., Wang, X., Li, L., & Wang, X.-L. (2007). A Statistical part of speech tagger for urdu. *International Conference on Machine Learning and Cybernetics.* Hong Kong, China: IEEE.

Daelemans, J. Z. (1999). Evaluatie van part-of-speech taggers voor het corpus gesproken nederlands. CGN technical report, Katholieke Universiteit Brabant, Tilburg.

Eugene Charniak, C. H. (1993). Equations for part-of-speech tagging. *Eleventh National Conference on Artificial Intelligence,* (pp. 784–789). Menlo Park: AAAI Press/MIT Press.

GMR Sastry, S. C. (2007). An HMM based part-of-speech tagger and statistical chunker for 3 Indian languages.

Godayal, S. M. (n.d.). *https://www.freecodecamp.org/news/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24/*. Retrieved from freecodecamp.org.

James. (n.d.). *https://www.cs.rochester.edu/u/james/CSC248/Lec9.pdf*. Retrieved from https://www.cs.rochester.edu: https://www.cs.rochester.edu/u/james/CSC248/Lec9.pdf

Lawrence R, R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *In Proceedings of the IEEE,vol 77(2)* (pp. 257-285). IEEE.

Loftsson, H. (2007). Tagging icelandic text using a linguistic and and statistical tagger. (pp. 105-108). Association for Computational Linguistics.

Manning, C. (2011). Part-of-speech tagging from 97% to 100%: is it time for some linguistics? *International Conference on Intelligent Text Processing and Computational Linguistics*, (pp. 171-189). Berlin.

Mario Fruzangohar, T. A. (October 2013). Improved Part-of-Speech Prediction in Suffix Analysis. *PLoS ONE*.

Rabiner, L. R. (1989). A tutorial on Hidden Markov Models and selected applications in speech recognition. *IEEE, volume 77(2)*, 257–285.

Ratnaparkhi, A. (1997). A Ratnaparkhi - IRCS Technical Reports Series, 1997 - repository.upenn.edu.

Samuelsson, C. (1993). Morphological Tagging BasedEntirely on Bayesian Inference. *Proceedings of NODALIDA*, (pp. 225-238).

Seraji, M. (2011). A Statistical Part-of-Speech Tagger for Persian. *Proceedings of the 18th Nordic Conference of Computational Linguistics NODALIDA 2011 / [ed] Bolette Sandford Pedersen, Gunta Nešpore and Inguna Skadiņa*, 340-343.

Vishaal Jatav, R. T. (2017). Improving Part-of-Speech Tagging for NLP Pipelines.

Wikipedia. (n.d.). *https://en.wikipedia.org/wiki/Hidden_Markov_model*. Retrieved from wikipedia.org.

Wikipedia. (n.d.). *https://en.wikipedia.org/wiki/Part-of-speech_tagging*. Retrieved from wikipedia.org.

[i] In mathematics, linear interpolation is a method of curve fitting using linear polynomials to construct new data points within the range of a discrete set of known data points.

[ii] In linguistic morphology, inflection (or inflexion) is a process of word formation,[1] in which a word is modified to express different grammatical categories such as tense, case, voice, aspect, person, number, gender, mood, animacy, and definiteness.[2] The inflection of verbs is called conjugation, and one can refer to the inflection of nouns, adjectives, adverbs, pronouns, determiners, participles, prepositions and postpositions, numerals, articles etc., as declension. An inflection expresses grammatical categories with affixation (such as prefix, suffix, infix, circumfix, and transfix), apophony (as Indo-European ablaut), or other modifications.

[iii] By definition, this quotient is ∞ if there is one possible tag for a given token.