STAT 927 Final Project

Part-Of-Speech Tagging Using Hidden Markov Chain Model

1 Introduction

Part-Of-Speech(POS) tagging is a significant research area in Natural Language Processing. In detail, it represents the process of assigning a part-of-speech marker to each word in an input text[1]. Word is ambiguous, for an individual word can have more than one possible tags. Therefore, one of the most challenging tasks of POS tagging is finding the correct tags for such ambiguous words.

Sentence	POS
a is a letter	NNP
buy *a* book	DT

Table 1: Example of word ambiguity

A good news is that, for some of ambiguous words, different tags are not equally likely. Let's take *a* as an example, the POS of *a* could be a Proper Noun (NNP) or Determiner (DT). However, in the emit matrix we calculated on our training data, the probability of DT is 20.1%, but NNP is only 0.00076%, which indicated that the DT is much more likely.

Due to this property, we can easily build a baseline model for POS tagging task: given an ambiguous word, choose the tag which is most frequent in the training corpus[1]. This most-frequent-tag baseline model performs quite good on our data set, it achieves an accuracy of 94.68% on our test set 1, and 91.67% on test set 2. However, this accuracy is still far away from that could have been achieved by more advanced models. Therefore, in this project, I try to use Hidden Markov Chain Model, which is introduced in the lecture to do POS tagging. The efforts I take to achieve this goal is as follows:

1. Extract emit and transition parameters from the training set.
2. Bulid the most-frequent-tag baseline model to tag the words in our test sets.
3. Build Viterbi and Forward algorithm from scratch in R studio to tag the words.
4. Analyze, evaluate and compare the predictions generated by each model.

2 Data Clarification

2.1 Data Source

Our training and test data set are all from Kaggle[2]. The original data set contains 25 columns, including word, previous word, lemma, word-shape and etc. However, as our task is predicting the POS tagging, we simply remain the current word and its corresponding POS. Here are some statistics of our data set.

Data	Size
Training set	1048554
Test set 1	94
Test set 2	444

Table 2: The sizes of our data sets

As it is quite time-consuming to use Viterbi and Forward algorithm to find the optimal tag sequence for given word sequence, I just limit the size of test set to an affordable range.

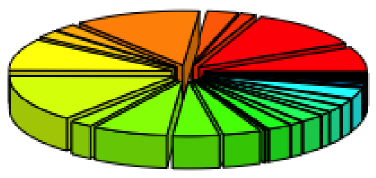
2.2 Tag Set

The tagset I use in this project is Penn Treebank tagset[3], one of the most authoritative English tagsets. It contains 36 alphabetical POS and 9 other pos for punctuations and special symbols. As some symbols do not appear in our data set, there are only 42 POS in total.

Tag	Description	Tag	Description	Tag	Description
CC	coordinating conjunction	PDT	predeterminer	VBP	verb non-3sg present
CD	cardinal number	POS	possessive ending	VBZ	verb 3sg pres
DT	determiner	PRP	personal pronoun	WDT	wh-determ.
EX	existential	PRP$	possess pronoun	WP	wh-pronoun
FW	foreign word	RB	adverb	WP$	wh-possess
IN	preposition/subordin-conj	RBR	comparative adverb	“	quote
JJ	adjective	RBS	superlatv. adverb	$	dollar sign
JJR	comparative adj	RP	particle	JJS	superlative adj
MD	modal	UH	interjection	TO	“to”
NN	sing or mass noun	VB	verb base form	NNS	noun, plural
VBD	verb past tense	NNP	proper noun	VBG	verb gerund
NNPS	proper noun, plu	VBN	verb past part	WRB	wh-adverb
RRB	close parenthesis	LRB	open parenthesis	,	comma
:	sent-mid punc	.	sent-end punc	;	sent-mid punc

Table 3: Part-of-Speech tags in our data set

The following pie chart shows the proportion of each tag in our data set. We can observe that, these tags are not equally frequent. Some tags, including NNP, VB, TO, NN and etc together, account for the majority of our data set. This imbalanced data naturally increases the difficulty of our POS tagging task.



3 Model

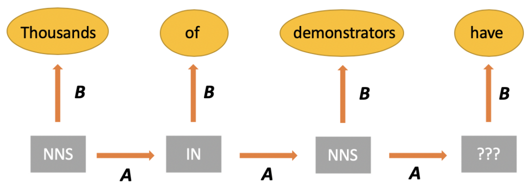
3.1 Hidden Markov Model

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 - the hidden states that cannot be observed. HMM assumes that there is another process Y, whose behavior "depends" on X.[4]

There are three basic problems of HMMs:

1. Given the observed sequence Y and model, compute the likelihood of observed sequence.
2. Given the observed sequence Y and model, find its corresponding optimal hidden state sequence X.
3. Given the observed sequence Y, find the optimal model.

If we apply HMMs to solve POS tagging task, then the observed sequence is a sequence of word. The hidden state sequence is a sequence of POS tags. And the problem we are going to solve is problem 2. In other word, we want to compute a probability distribution over possible sequences of tags and select the optimal one for the given word sequence.



The above graph briefly illustrates how our HMM tagger works. A HMM tagger contains two important components **A** and **B**), where **A** represents the transition matrix and **B** represents the emission matrix. After we understand the general idea of the HMM tagger, we can start to formulate our model.

First of all, I want to explicitly claim two assumptions for our model:

1. The current tag depends on and only on the previous tag. This is also the inherent assumption of first-order Markov chain.
$$Pr(t_i|t_{i-1}, t_{i-2}, ..., t_1) = Pr(t_i|t_{i-1})$$
2. The current word depends only on its corresponding tag.
$$Pr(w_i|t_i, t_{i-1}, t_{i-2}, ..., t_1, w_{i-1}, w_{i-2}, ..., w_1) = Pr(w_i|t_i)$$

Then, our goal is to select the most likely tag sequence t^n given the observed word sequence w^n, such that:

$$\hat{t^n} = arg\,max_{t^n} Pr(t^n|w^n) = arg\,max_{t^n} \frac{Pr(w^n|t^n)*Pr(t^n)}{Pr(w^n)} \propto arg\,max_{t^n} Pr(w^n|t^n) * Pr(t^n)$$

Now, we can plugin in our assumptions,

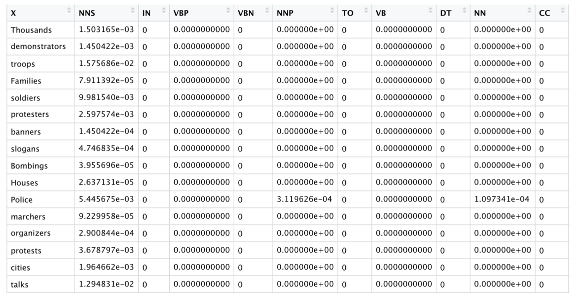
$$\hat{t^n} \propto arg\,max_{t^n} \prod_i^{i=n} Pr(w_i|t_i) * Pr(t_i|t_{i-1})$$

Next, we may notice $Pr(w_i|t_i)$ is the emission probability given by emission matrix $\mathbf{A}$ and $Pr(t_i|t_{i-1})$ is the transition probability given by transition matrix $\mathbf{B}$.

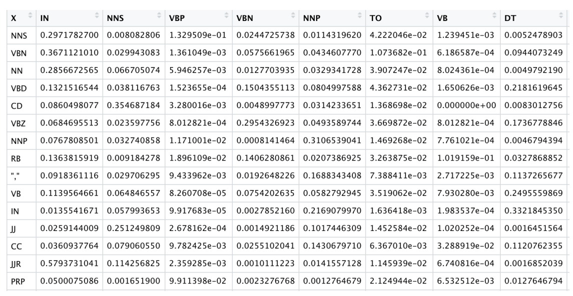
Finally, the problem comes - how do we get the emission and transition matrix? In this project, I use the MLE estimation calculated on our training corpus:

$$Pr(w_i|t_i) = \frac{Count(w_i,t_i)}{Count(t_i)} \qquad Pr(t_i|t_{i-1}) = \frac{Count(t_i,t_{i-1})}{Count(t_{i-1})}$$

Here is how our final emission and transition matrix (incomplete) look like:



X	NNS	IN	VBP	VBN	NNP	TO	VB	DT	NN	CC
Thousands	1.503165e-03	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
demonstrators	1.450422e-03	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
troops	1.575686e-02	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
Families	7.911392e-05	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
soldiers	9.981540e-03	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
protesters	2.597574e-03	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
banners	1.450422e-04	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
slogans	4.746835e-04	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
Bombings	3.955696e-05	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
Houses	2.637131e-05	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
Police	5.445675e-03	0	0.0000000000	0	3.119626e-04	0	0.0000000000	0	1.097341e-04	0
marchers	9.229958e-05	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
organizers	2.900844e-04	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
protests	3.678797e-03	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
cities	1.964662e-03	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0
talks	1.294831e-02	0	0.0000000000	0	0.000000e+00	0	0.0000000000	0	0.000000e+00	0



X	IN	NNS	VBP	VBN	NNP	TO	VB	DT
NNS	0.2971782700	0.008082806	1.329509e-01	0.0244725738	0.0114319620	4.222046e-02	1.239451e-03	0.0052478903
VBN	0.3671121010	0.029943083	1.361049e-03	0.0575661965	0.0434607770	1.073682e-01	6.186587e-04	0.0944073249
NN	0.2856672565	0.066705074	5.946257e-03	0.0127703935	0.0329341728	3.907247e-02	8.024361e-04	0.0049792190
VBD	0.1321516544	0.038116763	1.523655e-04	0.1504355113	0.0804997588	4.362731e-02	1.650626e-03	0.2181619645
CD	0.0860498077	0.354687184	3.280016e-03	0.0048997773	0.0314233651	1.368698e-02	0.000000e+00	0.0083012756
VBZ	0.0684695513	0.023597756	8.012821e-04	0.2954326923	0.0493589744	3.669872e-02	8.012821e-04	0.1736778846
NNP	0.0767808501	0.032740858	1.171001e-02	0.0008141464	0.3106539041	1.469268e-02	7.761021e-04	0.0046794394
RB	0.1363815919	0.009184278	1.896109e-02	0.1406280861	0.0207386925	3.263875e-02	1.019159e-01	0.0327868852
","	0.0918361116	0.029706295	9.433962e-03	0.0192648226	0.1688343408	7.388411e-03	2.717225e-03	0.1137265677
VB	0.1139564661	0.064846557	8.260708e-05	0.0754202635	0.0582792945	3.519062e-02	7.930280e-03	0.2495559869
IN	0.0135541671	0.057993653	9.917683e-05	0.0027852160	0.2169079970	1.636418e-03	1.983537e-04	0.3321845350
JJ	0.0259144009	0.251249809	2.678162e-04	0.0014921186	0.1017446309	1.452584e-02	1.020252e-04	0.0016451564
CC	0.0360937764	0.079060550	9.782425e-03	0.0255102041	0.1430679710	6.367010e-03	3.288919e-02	0.1120762355
JJR	0.5793731041	0.114256825	2.359285e-03	0.0010111223	0.0141557128	1.145939e-02	6.740816e-04	0.0016852039
PRP	0.0500075086	0.001651900	9.911398e-02	0.0023276768	0.0012764679	2.124944e-02	6.532512e-03	0.0127646794

3.2 Decoding Algorithms

I implement the Viterbi and Forward Algorithms in R from scratch. If you are interested in the details of these decoding algorithms, please see the R code. Here, I just introduce the general idea of them.

3.2.1 The Viterbi Algorithm

The Viterbi Algorithm is a Dynamic Programming Algorithm to find the $\mathbf{t^n}$ that returns the maximum $Pr(\mathbf{t^n}, \mathbf{w^n})$. To implement the Viterbi Algorithm, I first create a probability matrix with one row for each observed word in test set and one column for each possible tag in our tagset. Each cell (n, T) in this matrix represents the probability that the tag of the n^{th} word is T , given the most likely previous tag sequence $t_1, t_2, ..., t_{n-1}$, emission matrix and transition matrix. Formally,

$$\delta(n, T) = max_{t_1,t_2,...,t_{n-1}} Pr(t_1, t_2, ..., t_{n-1}, t_n = T, w_1, w_2, ..., w_n)$$

The whole process of calculating this matrix divides into three steps:

1. Initialization: In this step, I calculated π by calculating the frequency of each tag in our train corpus.

$$\delta(1, T) = Pr(w_1|t_1 = T) * \pi_T$$

2. Induction: In this step, P represents the "best" previous tag.

$$\delta(n, T) = max_P \ \delta(n-1, P) * Pr(t_n = T|t_{n-1} = P) * Pr(w_n|t_n = T)$$

3. Trace Back: In this step, T is the optimal tag we found for the last word and P is the optimal tag we found for the i^{th} word given the next tag $\hat{t_{i+1}}$.

$$\hat{t_n} = arg\,max_T \ \delta(n, T),$$
$$\hat{t_i} = arg\,max_P \ \delta(i, P) * Pr(\hat{t_{i+1}}|t_i = P)$$

3.2.2 The Forward Algorithm

The forward algorithm is quite similar with the Viterbi Algorithm. However, rather than choosing the tag that give us the highest probability in each iteration, forward algorithm just sums up the probability of each tag and sampling backwards with the posterior distribution. The whole process is as follows:

1. Initialization:

$$\alpha(1,T) = Pr(w_1|t_1 = T) * \pi_T$$

2. Induction: In this step, P represents all possible previous tags.

$$\alpha(n,T) = \sum_P \alpha(n-1,P) * Pr(t_n = T|t_{n-1} = P) * Pr(w_n|t_n = T)$$

3. Backwards Sampling: In this step, m is the total number of possible tags, M is our tagset.

$$t_n = T, \text{ w.p. } \frac{\alpha(n,T)}{\sum_{i=1}^{m} \alpha(n,M_i)}$$
$$t_{n-1} = P, \text{ w.p. } \frac{\alpha(n-1,P)*Pr(\hat{t_n}|t_{n-1}=P)}{\sum_{j=1}^{m} \alpha(n-1,M_j)*Pr(\hat{t_n}|t_{n-1}=M_j))}$$

4 Evaluation

4.1 Model Performance

The metric I used here to evaluate the model performance is accuracy, which is the percentage of tags correctly labeled by our model, compared with the true tags in test set.

Algorithm	Test set 1	Test set 2
Baseline	94.68%	91.67%
Viterbi Algorithm	97.87%	96.85%
Forward Algorithm	100%	97.07%

Table 4: The performances of various decoding algorithms

Based on the Table 4, we discover that, compared with baseline model, both Viterbi and Forward Algorithms greatly increase the accuracy. And Forward Algorithm outperforms Viterbi on both test sets. Especially, on test set 1, the accuracy of Forward Algorithm achieves 100%.

4.2 Error Analysis

By eyeballing the mistakes made by Viterbi and Forward Algorithms (shown in Table 6, the left table is for Viterbi), we find that in most of times, these two decoding algorithms make the same mistakes. For example, they both tends to predict "-" as ":", for in our emission matrix, the probability of $Pr("-"|":") = 0.83$, which is extremely high compared with the others.

However, they also have differences. Take *sought* as example, the emission probabilities of VBD and VBN are almost the same. However, the transition probability $Pr(VBN|NN)$ is less than $Pr(VBD|NN)$. Thus, the Viterbi tends to predict *sought* as VBD, which is not correct. However, the Forward Algorithm does not simply choose the tag with highest probability, but sample from the posterior distribution, which builds more uncertainty into the POS tagging process.

Word	True POS	Predicted POS
one	NN	CD
all	DT	RB
polling	VBG	NN
registered	JJ	VBN
expected	VBD	VBN
meeting	NN	VBG
offered	VBN	VBD
Protestant	NNP	JJ
witness	NN	VB
short	JJ	RB
sought	VBN	VBD
Protestants	NNPS	NNS
-	IN	:
as	RB	IN

Word	True POS	Predicted POS
all	DT	RB
-	:	IN
polling	VBG	NN
speaking	NN	VBG
expected	VBD	VBN
offered	VBN	VBD
Protestant	NNP	JJ
witness	NN	VB
short	JJ	RB
Protestants	NNPS	NNS
spying	VBG	NN
-	IN	:
as	RB	IN

Table 6: The mistakes made by Viterbi and Forward Algorithms

5 Discussion

5.1 Conclusion

Firstly, compared with the most-frequent-tag baseline model, the Hidden Markov Model we build takes the context information (previous tag) into consideration and performs quite good on our test corpus. The best accuracy we achieved is 97.07%, which is nearly 6% higher than that of the baseline model.

Moreover, I compare two different decode algorithms - Viterbi and Forward Algorithms. The model performance shows that, these two algorithms are similar with each other. However, as Forward Algorithm builds more uncertainty into POS tagging process, it outperforms Viterbi Algorithm in both test set.

5.2 Future Improvements

Although the performance of our model has achieved 97% accuracy, compared with the most state-of-art method, there is still a long way to go.

1. Unknown words. In this project, in order to simply the scenario, I just delete the unknown words in the test set, which means we can only find POS tags for the words that appear at least once in our train corpus. This is an ideal situation. In reality, we must find an elegant way to deal with unknown words.

2. Long-term dependency. In our bigram model, we simply assume that the current tag only depends on the current word and the previous tag. However, in reality, languages always have long-term dependency. Therefore, the models that can capture this long-term dependency, such as trigram, LSTM, Bi-LSTM may greatly improve our model performance.

3. More features. In POS tagging process, we can not only use POS, but also other possible features, such as word-shape, lemma, capitalization and so on. These additional information might increase our model performance.

References

[1] Jurafsky, Daniel & Martin, James. (2008). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.*

[2] Abhinav Walia: Annotated Corpus for Named Entity Recognition, `https://www.kaggle.com/abhinavwalia95/entity-annotated-corpus`

[3] Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. . *Building a large annotated corpus of English: The Penn treebank*. Computational Linguistics, 19(2), 313– 330.

[4] WIKIPEDIA: Hidden Markov model `https://en.wikipedia.org/wiki/Hidden_Markov_model`