CS5740 Assignment 3 Github Group: technologics

Zexi Liu
zl558@cornell.edu

Chaoping Deng
cd446@cornell.edu

Ying Zhu
yz2354@cornell.edu

1 Introduction

One of the popular topics in NLP is tagging each word in sentences with their syntax role. This task is called Part-Of-Speech Tagging. In this assignment, we built a POS Tagger that generate a sequence of POS tags for the correponding sentence. We train our POS Tagger with Penn TreeBank dataset and our POS Tagger made use of Penn TreeBank Tagset which contain 45 tags. Finally, we achieved test set accuracy 0.9616.

Section 2 introduces the main model we used, Hidden Markov Model (HMM), as well as different smoothing methods and different ways for inference we experimented with. Section 3 talks about details of pre-processing, unknown word handling and experiments implementations. Section 4 shows experiments results. Section 5 focus on error analysis and conclusions from experiments.

2 Approaches

The main model we used is HMM, which is a generative model. We also explored different smoothing methods like Add-k, Lineatr Interpolation and Kneser Ney. In the inference stage, we tried greedy approach, beam search and Viterbi algorithm.

2.1 HMM

We saw a sentence as a sequence of tokens, which is denoted as x_1, x_2 ... x_n. Denote the corresponding sequence of POS tags as y_1, y_2 ... y_n.

HMM is defined by four components: State (S), Observation (X), Transition Distribution (q) and Emission Distribution (e). HMM has two important assumption. 1) States are generated by Markov Model, which means the current state only depend on the previous state. 2) Observations are chosen independently, conditioned only on the state.

- States (S) are consist of POS tags. For example, if the HMM used bi-gram information, then S = {Penn TreeBank Tagset}, if the HMM used tri-gram information, then S = {Penn TreeBank Tagset} x {Penn TreeBank Tagset}.
- Observations (X) are the real words from the sentence. The size of X is exactlly the size of vocabulary V.
- Transition Distribution (q) model the conditional probability of current tag given previous tags information, $q(S_i|S_{i-1})$. For example, in bi-gram context, $q(S_i|S_{i-1}) = q(y_i|y_{i-1})$, in tri-gram context, $q(S_i|S_{i-1}) = q(y_i|y_{i-2}, y_{i-1})$.
- Emission Distribution (e) model the conditional probability of an observation given a tag, $e(x_i|y_i)$.

In essence, HMM is a generative method trying to model the joint probability,

$$P(x_1, x_2...x_n, y_1, y_2...y_n) = q(STOP|Y_n) \prod_{i=1}^{n} q(Y_i|Y_{i-1}) e(x_i|y_i) \quad (1)$$

Figure 1 is the graph representation of tri-gram HMM. Even though the states are tuples of tags, the first tag of current state must match the last tag of last state, only the last tag of current state is newly generated, and the observation is only generated from this new tag, that is how the transition probability and emission probability come.

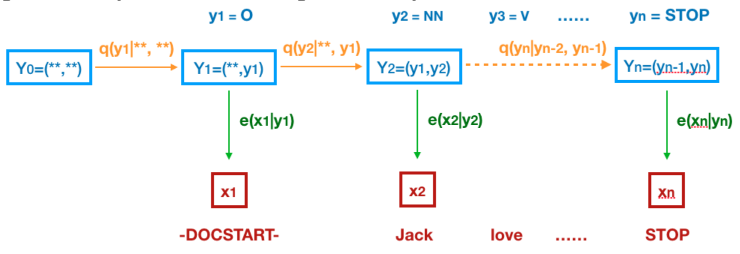

Figure 1: Tri-gram HMM Graph Representation

As for the learning problem, and the computations of transition and emission probabilities, we explain in next subsection, Learning and Smoothing Methods.

2.2 Learning and Smoothing Methods

To calculate Transition Distribution (q) and Emission Distribution (e), we used Maximum Likelihood Estimation. In essence, MLE is simply counting and doing division using the following formulars.

$$q_{MLE}(y) = \frac{c(w)}{c()} \quad (2)$$

$$q_{MLE}(y_n|y_1, y_2, ...y_{n-1}) = \frac{c(y_1, y_2, ...y_n)}{c(y_1, y_2, ...y_{n-1})} \quad (3)$$

$$e_{MLE}(x_i|y_i) = \frac{c(x_i, y_i)}{c(y_i)} \quad (4)$$

where $c(y_1, y_2, ...y_n)$ denotes the counting of sequence $y_1, y_2, ...y_n$ happened in training dataset, $c(x_i, y_i)$ denotes the counting of (x_i, y_i) pairs happened in training dataset, c() denote the count of entire corpus.

However, in reality, many counts are zero and the MLE method suffers from the sparsity issue. To make MLE work better, we smooth the counts using variant methods. Below we introduce Add-k and Linear Interpolation smoothing.

2.2.1 Add-k

One intuitive but naive way for smoothing is pretending we see each counted elements k more time than we do by simply adding k to all counts, that means, in MLE, we add k to the numerator.

To make the smoothed probability add up to 1, we add kV to the MLE denominator, where V is the vocabulary size.

$$P_{Add-k}(x_i|x_{i-1}) = \frac{c(x_{i-1}, x_i) + k}{c(x_{i-1}) + kV} \quad (5)$$

We can do some tricks to the k, like making it proportional to the uni-gram distribution, as known as Unigram Prior Smoothing. But in essence they are all the same. Add-k works poorly when there are many zeros. If re-construct the counts after smoothing, originally large number will be significantly dragged down.

2.2.2 Linear Interpolation

With the n of n grams increasing, the problem of sparsity become severer and severer. One natual thought is to make use of lower grams information when estimate higher grams probability.

$$P_\lambda(x_n|x_1, x_2, ..., x_{n-1}) = \sum_{i=2}^{n} \lambda_i P_{MLE}(x_n|x_{n-i+1}, ..., x_{n-1}) + \lambda_1 P_{MLE}(x_n) \quad (6)$$

where P_{MLE} is defined in equation (2) - (4). To make sure this is a well defined distribution, we must ensure $\lambda_i \geq 0$ and $\sum_{i=1}^{n} \lambda_i = 1$. We validate λs using Maximum Likelihood on the whole training data set. Linear Interpolation is the smoothing method that gave us 0.9616 accuracy.

2.3 Inference

This part focuses on the task of inferring tags given sequences of observation. We implemented three different methods, greedy, beam search and Viterbi algorithm. They differ in speed and optimality.

2.3.1 Greedy

Greedy takes the most intuitive but naive and short eye sight way to infer tags. For given observation and previous state, greedy method try out all the possible tags. For each possible tags, we calculate the transition and emission probability with the observation and previous state information. And then we picked out the tags with highest product of transition and emission probability and thrown away the rest. Using this newly generated so called "best" tag, the greedy algorithm continues doing the same procedure to generate the next tag. On each step, exactly one path is kept and all other possible paths are abandoned.

2.3.2 Beam Search

Greedy is a special form of Beam Search. Compared to greedy, who only keeps the highest scored tag on each step, Beam Search keeps top k highest scored tags, and abandons the rest. Therefore, on each step, there are exactly k paths are kept and all other possible paths are abandoned. And this is the only difference from greedy.

2.3.3 Viterbi Algorithm

Greedy and Beam Search only look at the local optimal, therefore, they often fall into sub-optimalities. Viterbi algorithm make use of dynamic programming and grantee the global optimal result.

It looks at the $\pi(i, y_i)$, which is the max score of a sequence of length i ending in tag y_i.

$$\pi(i, y_i) = max_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \pi(i-1, y_{i-1}) \quad (7)$$

Viterbi algorithm is a bottom up dynamic programming algorithm makes use of a data structure called trellis. In trellis, we not only compute the score $\pi(i, y_i)$, in order to re-construct the path, we also compute the back pointer $bp(i, y_i)$.

$$\text{bp(i, y}_i) = argmax_{y_{i-1}} e(x_i|y_i) q(y_i|y_{i-1}) \pi(i-1, y_{i-1}) \quad (8)$$

Even Viterbi algorithm manages to achieve global optimal, it also has an accpetable time and space complexity. The time is $O(nk^2)$, the space is O(nk), where n is the sentence length and k is the number of possible tags.

3 Experimental Setup

3.1 Pre-processing

The original data files are all in csv format with one index and one token or one tag per row. There is a special token named "-DOCSTART-" that separated documents. Each documents contained multiple sentences. We first regenerated the data file into txt format, where each line is no longer a token or a tag, but a document containing several sentences, the tokens including punctuation in a document were separated by a space. Each document must begin with "-DOCSTART-" and end with another special token "STOP" that we added to the end to each document. We did not change the token to lowercase, did not remove anything, since in POS tagging task, any pre-fix, suf-fix, uppercase, lowercase, punctuation could be hints to improve the performance.

However, we noticed the digits were shown in many forms and cause sparsity, to deal with this problem, we changed all the digits or any token containing digits to categorical tokens. The categoties with examples were:

oneD	9	DashLowerD	9-year
twoD	90	SlashD	11/9/89
threeD	900	SlashWordD	May/9/89
fourD	1990	CommaD	9,000
DashD	09-96	PeriodD	9.0
DashUpperD	A89-67	PeriodCommaD	9,000.0

Table 1: Digit Categories

We replaced all the token containing digits in the raw data with the above categories if the token matches one of the pattern, otherwise, we replaced the token with "otherDigit".

After the pre-processing, the training data looked like: "-DOCSTART- A shareholder filed suit , seeking to block Unitel Video Inc. 's proposed plan to be acquired by a new affiliate of closely held Kenmare Capital Corp. for $ twoD a share , or $ PeriodD million . The complaint alleges that the price is " unfair and grossly inadequate " . STOP" whose tags sequence was:

"O DT NN VBD NN , VBG TO VB NNP NNP NNP POS VBN NN TO VB VBN IN DT JJ NN IN RB JJ NNP NNP NNP IN $ CD DT NN , CC $ CD CD . DT NN VBZ IN DT NN VBZ " JJ CC RB JJ " . STOP", noted that the "twoD" after the first $ was originally 15, "PeriodD" after the second $ was originally 33.6.

3.2 Dealing With Unknown Words

	Threshold 2	threshold 3	threshold 5	Threshold 10
Accuracy	0.9504	0.9492	0.9472	0.9429

Table 2: Comparison of different unknown word threshold

We simulate the appearance of unknown words, which suppose to be the words that appear in dev set but not in the training sets, by categorizing words with frequency less than a certain threshold value (threshold value set to 1 has the best performance according to Table 2) from training set as unknown. We then process these unknown words using the word class implementation method. Our very first approach utilized a total number of 8 words classes as defined below:

From Table 3, we discovered that type 7 unknown words, which are words that contain only lower class letters, accounted for the most popular word class. Therefore, we investigated type 7 words with the help of suffix analysis. We believe suffixes can help us use context and etymological clues to make educated guesses about meaning of unfamiliar words, therefore deducting their tags. We therefore lever-

ID	Name	Meaning	Count
1	allCaps	Words with all capital letters	238
2	initCaps	Words starting with capital	3560
3	hasHypen_allCaps	Words containing only hyphen and capital letter	57
4	hasHypen	Words containing hyphen with more than one lowercase letter	329
6	specialCharacter	Word with special characters	1
7	Lowercase	Word with only lowercase letters	6878
8	UNK	None of the above	254

Table 3: Word Classes for Unknown/Low Frequency words

aged the most commonly used suffixes for English words and further break down the type 7 words to different subcategories, as shown in Table 4.

ID	Name	Meaning
7_1	Noun	(acy, ance, ence, dom, ism, ist, ity, ment, ness, ship, sion, tion, ee, tress, or, ette)
7_2	Verb	(ate, en, fy, ize, ise, ish, yze, yse)
7_3	Adjective	(able, ible, al, ial, ical, an, ian, ary, esque, ful, ic, ious, ish, ive, less, like, ous, ose, ant, ent, ile, zy, ty, sy, dy, ly, tory)
7_3_2	Adverb	(ward, wards, wise, istic, istical, ingly, edly, ishly, ively, lessly, iously, uously, ily)
7_4	PastTense	None of above and ends with "ed"
7_4_2	Comparative	Comparative and superlative words
7_5	Plural	None of above and ends with "s"
7_6	Verb form 2	None of above and ends with "e"
7_7	Verb or adjective	None of above and ends with "ing"
7_8	Noun	None of the above

Table 4: Further Word Classes Categories for Lowercase Unknown/Low Frequency words

Overall the suffix adjusted method resulted in significant improvement(0.01) for both our dev and test performances.

3.3 Implementation Details

- In our Linear Interpolation smoothing method, for trigrams transition probability, λ_3 is 0.85, λ_2 and λ_1 are both 0.075, for emission probability, λ_2 is 0.009 and λ_1 is 0.991.
- Our implementation use hash map heavily. All the counts are done using python dictionary, each layers in the trellis is also dictionary. There should be more efficient data structure that could finish this task, however, our speed and memory performances are not that bad, so we focused on experiments instead of code optimization.
- When we calculate the score in greedy, beam search and Viterbi, since the score is the production of probability, score could easily go very small even zero. Therefore, we use summation of log probability as scores.

4 Results and Analysis

4.1 Smoothing Methods

We found that add-k smoothing performance increase as k becomes smaller and linear-interpolation performs better than add-k smoothing method.

	Add-1	Add-5	Add-10	Add-200	Add-500	Linear- Interpolation
Accuracy	0.9287	0.8850	0.8508	0.6410	0.5954	0.9504

Table 5: Comparison of different smoothing methods

4.2 Smoothing Parameter

With other condition controlled, we experiment with different smoothing parameters in linear interpolation to see how accuracy changed.First, we experiment with the smoothing parameters lambda for transition probability. We found that with larger lambda, we will have better accuracy on the development set. But any lambdas larger than 0.85 will cause the drop in accuracy. The result is reasonable,by having larger lambda, we are decreasing the smoothing effect.

	0.10	0.30	0.50	0.70	0.80	0.85	0.90
Accuracy	0.9496	0.9500	0.9506	0.9505	0.9504	0.9504	0.9499

Table 6: Comparison of different transition probability smoothing

With other condition remain same, we experiment with different emission probability that we assigned for unknown words.The higher emission probability we assigned to unknown words, the lower accuracy we will obtain.

	0.005	0.01	0.02	0.03	0.05	0.1
Accuracy	0.9505	0.9505	0.9504	0.9503	0.9500	0.9480

Table 7: Comparison of different emission probability smoothing

4.3 N-grams

We experiment with multiple grams and found that the performance of 3 gram is superior than that of 2 gram model. Here is a short table to compare the performances of two types of gram models. We include the top 4 tag types that 3-gram model predicts a lot better than 2-gram model and top 3 tag types that 2-gram model outperforms 3-gram models. We discover that 3-gram model does well regarding Verb predictions, which generally follow the subjectverbobject (SVO) structure, whereas 2-gram model predicts better than 3 gram in cases for Determiner words such as a, the ,very. 3-gram model tends to add additional noise since DT words generally is followed by one word, for example, an apple.

	2 gram	3 gram	4 gram	5 gram
Accuracy	0.9486	0.9504	0.9485	0.9441

Table 8: Comparison of different n grams

4.4 Inference Methods

From Table 9 we see that Viterbi is obviously better than greedy and beam search. Beam search 2, 3 and 5 perform nearly the same, but they are not necessarily better than greedy. With the increasing of k, the performance is even slightly worse.

To validate the optimality of our Viterbi algorithm, we wrote a function to calculate the probability of a sequence of tags given the sequence of observations. For short sentences, we brute force the tags combinations with highest probability, the results comply to our Viterbi results. It worth mentioned that, even the "Ground Truth Tags" has lower probability than our Viterbi result, because the Viterbi only output the global optimal in mathematical sense, it doesn't necessary to be the optimal for human understanding.

After validating the optimality of Viterbi results, we compared our greedy result with Viterbi results to see the sub-optimalities of greedy method. Out of 10540 sentences separated by period, 4574 of the greedy results were not the same as Viterbi results, which mean nearly 43.4% of greedy results had sub-optimality. Most of the sub-optimality only happen at one word, for example, the sentence is "b - As of Thursday 's close .", the Viterbi results are "SYM : VBN IN NNP NNP NN NN" while the greedy results are "SYM : VBN IN JJ NNP NN NN", only the fourth word is sub-optimal.

Inference	Greedy	BS 2	BS 3	BS 5	Viterbi
Accuracy	0.9505	0.9499	0.9497	0.9493	0.9598

Table 9: Comparison of different inference methods

4.5 Final Model Performance

For our final model, we used 3-gram HMM with word-class unknown processing, linear-interpolation smoothing with parameter 0.85/0.009, Viterbi inference and achieved 0.9598 on dev set and 0.9616 on test set.

5 Error Analysis and Conclusion

5.1 Unknown Error Analysis

We achieved an accuracy around 80.35% on unknown words. The most popular mis-classifications are between nouns and adjectives.Among this type of misclassifications: The most mistake it makes (with count of 400 cases) are to categorize words with form of word-word, for example,[hen-minister , animal-rights, freight-transport] , which are supposed to be noun but are classified as adjective, and words like [private-school, law-enforcement, gas-station] could be categorized as nouns, for which cases they are actually subjectives.

The second type of mistake involves a single word that can be both noun and adjectives. These include words such as [Biotech, Hollow, Backup]

We have found that it is confusing even for humans to differentiate tags for this type of mistakes given only the syntaxical information. We think this scenario defines the stretch of prefix processing and additional information, such as position of the word in the sentence, could be helpful to a great extent.

Another popular mis-classification mistakes (around 200 cases) the machine makes for unknown words are differentiation between type NN and NNP tags. Fundamentally it is hard to differentiate a proper noun from an ordinary noun word just based on word forms. For example, it is hard to tell noun word [FOX, Jacuzzi,Texasness] from a standalone name referring to a special entity or person. We could potentially leverage NER algorithms to parse such entities as a preprocessing step for our POS task.

5.2 Development Set Error Analysis

Shown By observing the accuracy report of development set, we could find that the prediction accuracy of punctuation is relative higher than other tags. For those tags with low accuracy,one explanation for the poor performance is their low frequency in data.

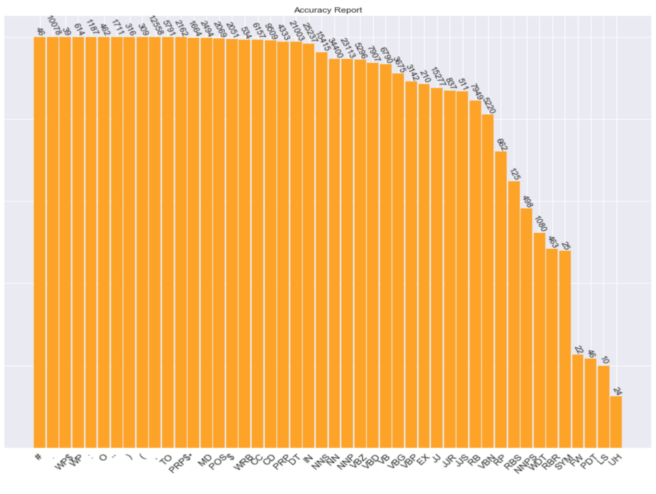

Figure 2: Accuracy Report

For those tags with high shown frequency and low accuracy, we take the error analysis one step further. We select $['NN','NNP','NNPS','NNS','JJ','RB','VBZ', 'VBD','VB','VBN','WDT','IN']$ as the representation of tags with high shown frequency and low accuracy and ploted the confusion matrix.

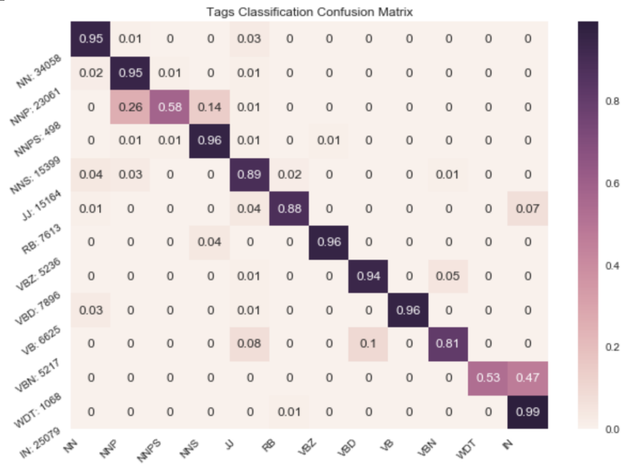

Figure 3: Confusion Matrix

After observing the confusion matrix, we observe that there are several pairs of tags that our model has difficulties to distinguish, including $['VBN','VBD'], ['IN','RB'], ['NN','JJ']$. The followings are some examples of mis-classification within these tag pairs.

Tag Pairs	Misclassified Words
VBN to VBD	'suffered','faced','imposed','warned','ended','valued','made','called'
RB to IN	'Besides', 'aboard', 'about', 'above', 'after', 'ago', 'along', 'around', 'as', 'because', 'before'
IN to RB	Once', 'So','about', 'ago', 'as', 'because', 'down', 'inside', 'off', 'once', 'out', 'so', 'though', 'up'
NN to JJ	'multiple', 'original', 'public','benchmark', 'total', 'average', 'marine-transport','standard', 'second', 'animal-rights'
JJ to NN	'public', 'chief', 'public', 'firm', 'scathing', 'biotech', 'chief','fellow', 'burdensome', 'perverse', 'non-duck', 'firm'

VBN is often used as participial adjectives,whereas VBD often follow a subjective. The evidence to classify VBN is the word afterward. For example, for the sentence 'This is an ended game', we classify 'ended' as VBN based on the noun shown after it. Since our model only pays attentions on the tags shown before the word 'ended', this could be one explanation for the mis-classification between VBN and VBD. This could also explain the difference between JJ and NN. For example, for the 'This is a race horse', we will easily classify 'race' as NN if we did not see the word 'horse' after it. The mis-classification between RB and IN could due to the large portion of common words within these two tags. There are 32.35% words in the RB tag also shown within the IN tag and there are 78.57 % words IN tag also shown in RB tag.