# Spring 2018 CS 5740: Final Exam

GITHUB GROUP: NERDS https://github.com/cornell-cs5740-18sp/final-exam-nerds

Disheng Zheng
dz336

Xiaohang Lu
xl672

## 1 Introduction

**Task**: In this assignment, we will implement several generative models for part-of-speech (POS) tagging. A POS tagger can process a sentence or document of words, and classifies each word to its lexical category. For example a sequence of words, "We have no useful information", the correct POS tags are: *We* is *PRP* a personal pronoun, *have* is *VBP* a verb (non-3rd person singular present), *no* is *DT* a determiner, *useful* is *JJ* a adjective, *information* is *NN* a noun.
**Data**: The data we used contains almost a million words of text from the WSJ. The words that made up each documents are written in column format and every document starts with *-DOCSTART-*. There are 696475 words (28837 sentences) in training data, 243021 words (10078 sentences) in dev. data, 236582 words (9818 sentences) in test data. Our task is to generate POS tags that are as accurate as the ground truth.
**Evaluation**: We evaluate our models by measuring their accuracies on the development dataset. The accuracy measures how well each model predicts the correct POS tag. A baseline accuracy is calculated as setting each word with its most frequent tag and unknown word as noun.

## 2 Approach: Trellis (HMM)

A trellis is a directed acyclic graph where each node represents a state and each edge represents transition between two states in the Hidden Markov Model (HMM). HMM is a generative model that was made up by two sets of probability matrices, one models the transition probability between hidden states and one models the emission probability from each state. In our POS tagging context, the hidden states are the 48 (after add $< start >$ and $< stop >$ )POS tags and the emissions are the possible words given each POS tag. In bi-gram model the HMM is defined by a 48 by 48 transition probability matrix and 48 by $N$ (vocabulary size) emission probability matrix. HMM learns the POS tags by maximizing the likelihood of the generated word sequence given all the possible transition and emission possibilities. It can be summarized by the following notation:
Give transition probability $q$ between states $y_i$ and $y_{i-1}$ as $q(y_i|y_{i-1})$, emission probability $e$ of word $x_i$ for a certain state as $e(x_i|y_i)$, HMM tries to learn the maximum likelihood of $p(x_1...x_n, y_1...y_n) = q(STOP|y_n)\prod_{i=1}^{n} q(y_i|y_{i-1})e(x_i|y_i)$ by searching for the set of states $y_1...y_n$ that maximize the "score" of $log(p(x_1...x_n, y_1...y_n))$.
For example, consider a sequence of word (Thank, you), where "Thank" is a *Verb* and "you" is a *Pronoun*, and the true POS tags are (*V*,*PR*). The trellis will look like the following figure with transition and emission probability matrices as shown in the figure 1. By calculating all the likelihoods of possible POS tags given by $q(y_2|y_1 = N)e(x_2 = you|y_2)$, the HMM trellis found the maximum is achieved by $p("Thank","you", V, PR)$.
There are three main methods to find the most likely sequences: greedy search, beam search and Viterbi.

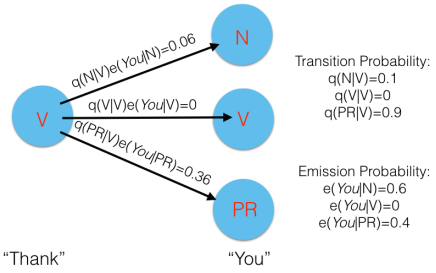


Figure 1: Example Trellis

**2.1 Greedy Search** The greedy search algorithm maximizes the likelihood between each neighbor states. It only searches for the most likely transition between current state and the next one, thus it only considers one possible POS tagging for a given sequence of words. That is, it picks the highest probability POS tag between $i-1$ and $i$ th token such that $argmax_k q(y_i = k|y_{i-1})e(x_i|y_i = k)$. The advantage of greedy search is its speed. Because it only considers the highest possibility between each state, it has a complexity of $O(N * K)$ where $N$ is the length of the document and K is all possible tags. However, the tags greedy algorithm found are almost always suboptimal because it doesn't find global optimal tags.
**2.2 Beam Search** The beam search is an extension of greedy approach. In contrast to greedy search it finds the top k possible transitions between $i-1$ and $i$ th token. Therefore, for every token, it keeps only top k possible transitions rather than 1 which is the case of greedy search. When hyperparameter $k$ is set to 1, it is equal to greedy search. Although beam search expands the capacity of greedy search and works empirically well, it still doesn't ensure to reach a global optimal.
**2.3 Viterbi Algorithm** The Viterbi algorithm is a dynamic programming approach to find the global optimal POS tags. Suppose the max "score" of POS tags for a sequence of $i$ words is denoted as $\pi(i, y_i)$, Viterbi algorithm searches for the previous max "score" sequence $\pi(i-1, y_{i-1})$ which is the global optimal POS tag for sequence of length $i-1$. Iteratively at each length, we find its global optimal sequence, i.e $\pi(0, y_0)$ stands for the global optimal for start of the sequence, $\pi(1, y_1)$ stands for the global optimal for the first token... The Viterbi algorithm states that we can achieve the global optimal by finding all the subsequence global optimal $y's$ and stitches all $y_0, y_1, ..., y_n$ together. The implementation of Viterbi comprises two parts, one part is to calculate the max score of $\pi(i, y_i)$ and the other part is to store the back-pointer $bp(i-1, y_{i-1})$ that refers to the previous global optimal $\pi(i-1, y_{i-1})$. The complexity of Viterbi algorithm is $O(N * K^2)$ where where $N$ is the length of the document and $K$ is all possible tags. It's $K$ times slower than the greedy search but it ensures to find global optimal POS tags.

## 3 Experimental Setup

**3.1 Data inspection**: Inspection of data helps understand the properties of data and is important for data processing. We no-

tice that our tags and words are of sparsity. Before introducing start and stop tag, there are 46 tags. Their distribution are shown in Figure 2. Most common tag is *NN* (13.9% of all tags). The top 5 are *NN, IN, NNP, DT, JJ* and they compose 49.0% of all the tags . The last 5 are *SYM, LS, UH, #, WP$* and their proportion is only 0.05% . There are 37504 unique words in training data, where 17592 words only appear once in the training data and 23033 words (61.4%) appear twice or less than twice.


Figure 2: Distribution of tag

The sparsity indicates a need of smoothing methods and handling of infrequent words. We inspect the words in dev. data. There are 8851 (3.6%) unknown words (not in the training) and rare words 6363 (2.7%) (1<= occurrence in training <=2) in dev. data and their error rate is high (Section 4.2). Though, the relative proportion of unknown/rare words are small in the whole data, high error rate of these words will increase the error rate of common words behind them, due to the nature of sequence model, which leads to low overall accuracy (Section 4.2). Therefore, it's of great importance to handle unknown/rare words.

**3.2 Sentence generation**: The POS tagging is sentence-based. We regard [".", "!", "?"] as stop punctuation and then based on this, we divide the data into sentence. In n-gram mode, (n-1) $< start >$ are added at the beginning of sentence and one $< stop >$ is added at the end of sentence.

**3.3 Unknown/Rare words inferring**: Here, we come up three methods to handle unknown and rare words. Rare/unknown words in dev. and test data are very likely also infrequent in training data, using rare words in the lexicon is a better approximation for unknown/rare words in dev. and test data than using frequent words in training (Thorsten ANLP 2000). After the following handling, the unknown/rare/common words in dev. data changes from 8851/6683/227487 to 0/46/242975, and that in test data changes from 8233/6363/221986 to 0/12/236570.

**3.3.1 Chunking**: There are many unknown/rare words in our train, dev. that share certain properties and can be segmented into spans. Based on our observation, we extracted the properties and divided them into the following spans.

**1. Numbers:** Numbers vary a lot but they have similar properties. E.g. there are 18445 numbers in training set (18380 of their tags are *CD*) and there are 3357 of them are rare words; there are 6648 numbers in dev. set (6633 of their tags are *CD*) and there are 1728 of them are unknown/rare words. Number like 15 are common in training (305 occurrence) while number like 852,000 is unknown.

**2. All capitals:** Words with all capitals and length greater than 1 tend to be organizations (e.g. CDC). E.g. there're 7158 this type words in training (6145 of their tags are *NNP*) and 887 of them are rare words; there're 2426 in dev. set (2093 of them tags are *NNP*) and there're 786 of them are rare/unknown words.

**3. All capitals expect the last character being 's':** These words, (e.g. CEOs), tend to be the plural form of a noun when word length is larger than 2 (to prevent words like 'As'). E.g. there are 148 this type words in training set (104 of their tags are *NNS*) and 26 of them are rare words; there are 23 in dev. set (all are *NNS*) and there are 2 of them are rare/unknown words. Though the absolute number is small, they have strong correlation within the group.

**4. Digitals plus low-case alpha:** Words that consist both digitals and alpha (expect the ones with forms being number+ 's' (e.g. '1990s'), which are mainly *CD* or *NNS*),these words tend to be adjective (e.g. 54-year-old). E.g. there are 825 this type words in training set (747 of their tags are *JJ*) and 455 of them are rare words; there are 243 in dev. set (212 of their tags are *JJ*) and 156 of them are rare/unknown words.

**5. Digitals plus 's':** The ones with form being number+ 's' (e.g. '1990s') are mainly about certain years, they are mainly *CD* or *NNS*. E.g. there are 183 this type words in training set (85 of their tags are *CD* and 90 are *NNS*) and there are 37 of them are rare words; there are 58 this type words in dev. set (18 of their tags are *CD* and 37 are *NNS*) and 11 of them are rare/unknown words.

**6. Hyphen and last alpha low-case:** The words like 'dealer-to-dealer','Jersey-based', which are alpha connected with low-case alpha by '-' tend to be adjective and their forms vary. E.g. there're 6851 of this type in training set (4733 of their are *JJ*) and 3125 of them are rare words; there're 2488 in dev. set (1744 of their with tags being *JJ*) and 1314 of them are rare/unknown words.

**7. Hyphen with alpha with upper-case:** The words like 'Dana-Farber', 'Anti-Deficiency', which are alpha with upper-case connected with alpha with upper-case by '-' tend to be *NNP* but their forms vary. E.g. there're 699 of this type in training set (630 of their are *NNP*) and 264 of them are rare words; there're 214 in dev. set (180 of their with tags being *NNP*) and 118 are rare/unknown words.

**8. Alpha with initial character upper-case:** The words with only initial character upper-case and no hyphen and no digit tend to be *NNP*, (e.g. Rudolph). E.g. there are 73683 of this type in training set (59368 of their are *NNP*) and 7829 of them are rare words; there are 25164 in dev. set (20372 of their with tags being *NNP*) and 4966 of them are rare/unknown words.

The unknown and rare words will be replaced by span name. To prevent that making span name the same as one of the word in previous vocabulary, we make these span name as *number*, *upper*, *upper_s*, *alpha_number*, *number_s*, *hyphen_alpha*, *hyphen_Alpha*, *Alpha* respectively.

**3.3.2 Suffix**: The suffix is a strong predictor for word tags. In our training data, e.g. the words ended with 'able' have 89% chance to be *JJ* (e.g. available) and the rest (9.7%) are mainly *NN,NNP* (e.g. cable, Cable). Suffix are mainly for real alpha words while the previous chunking are for special words with number, hyphen, all upper-case et al., so they hardly overlap. If they do, chunking processing has priority. The length of informative suffix varies, e.g. ness (length 4, 92.1% *NN* ), ful (length 3, 90.5% *JJ*), ez (length 2,100% *NNP*). To solve this, we build a list of dictionary based on rare words in training data to collect the suffix of different length (2,3,4) with information of their dominant tag's proportion (e.g. dominant tag of suffix 'tive' is

*JJ* since it is the most common tag). The suffix that has more than 5 occurrences and its dominant tag's proportion is above 0.3 is regarded as informative suffix and will be collected. E.g., part of dictionary for the informative suffix with length 4 is : ('aker': 0.69, 'tive': 0.86, 'rity': 0.92, 'usly': 1.0,'kman': 0.86). For a given rare/unknown word, if its suffixes (length 2,3,4) are in these dictionaries (e.g. evening has suffixes ng, ing, ning and they're all in our dictionary), we compare these suffixes' dominant tag's proportion (ng:0.69, ing:0.71, ning:0.66), and choose the suffix with largest dominant tag's proportion (ing in evening case) as the word's new representation. To prevent that making suffix name the same as the word in previous vocabulary, we make these suffix name as *suffix* (e.g. *ing*).

**3.3.3 Unknown token**: If the rare/unknown words are not processed by chunking, and their suffixes are not in the informative suffixes dictionaries, they will be replaced by the word '*unknown*' in the data.

**3.4 Smoothing**: Smoothing is the process of flattering probability distribution so that all word sequences can occur with some probability. This often involves redistributing weight from high probability regions to zero probability regions. Since we face some difficulties when sparse training data causes poor probabilities estimates. Unseen words have emission probabilities of zero. Proper smoothing the models are usually more accurate than the original models.

**3.4.1 Add-k**: The general form of Add-k smoothing is:

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

For Bi-gram transition smoothing, $x_i$ is our current state, $x_{i-1}$ is the previous state. And V is the size of all states. For Bi-gram emission smoothing, $x_i$ is our current word, $x_{i-1}$ is the current state. And V is the size of all vocabulary.

To choose k for tri-gram transition, we have validated k = 0.02, 0.05, 0.1, 0.5, 1, 2. And the respective accuracies are 0.9548, 0.9547, 0.9546, 0.9542 0.953, 0.950, 0.952, Hence we have chosen k to be 0.02 for transition. To choose k for tri-gram emission, we have validated k = 0, 0.1, 0.2, 0.5, 1. And the respective accuracies are 0.952, 0.951, 0.950, 0.948, 0.947. Hence we have chosen k to be 0, so not to use smoothing on emission.

**3.4.2 Linear interpolation**: The general form of tri-gram Linear interpolation smoothing is:

$$P_\lambda(w|u,v) = \lambda_3 P_{MLE}(w|u,v) + \lambda_2 P_{MLE}(w|v) + \lambda_1 P_{MLE}(w)$$

For tri-gram transition smoothing, w is our current state, u is the current - 1 (previous) state, and v is the current -2 state.

To choose $\lambda$ for bi-gram, we have validated $\lambda$ = (0.65, 0.35), (0.7, 0.3), (0.8, 0.2), (0.9, 0.1). The respective accuracies are 0.9182, 0.9157,0.9092, 0.9067. Hence, we have chosen $\lambda$ = (0.65, 0.35) To compare Add-k and linear interpolation, for the same parameters, add-k has accuracy: 0.95139, linear interpolation has accuracy: 0.9182. Hence we choose to use add-k 0.02 for transition.

**3.5 HHM**: For HHM, bi-gram and tri-gram are experimented. Greedy search, beam search with different width, and Viterbi algorithm were experimented. Details are described in Section 4.

**3.6 Final model**:

**Parameters**: Unknown/rare words handling as described in Section 3.3. Tri-gram, beam search (k=4). Smoothing method is add-k (k=0.02).

**Results**: 0.9592 in Dev. set and 0.9610 in test set.

# 4 Results and Analysis

**4.1 Baseline**: A baseline accuracy is calculated as setting word to its most frequent tag and unknown word as noun.

**Results**: The dev. data baseline accuracy is **91.0%** with only **13.9%** accuracy on 8851 unseen words.

**Analysis**: By generating the confusion matrix of the unknown words, we found that most errors are due to misclassification of *NNP* as *NN*. By assigning unknown words as *NN*, in fact, the accuracy for unknown is the proportion of *NN* in unknown data. The error is misclassification of all other tags into *NN*.

**4.2 Unknown/rare words handling**

**Table 1** Results with different unknown handling

| | accuracy | |
|---|---|---|
| **method** | **overall** | **unknown** |
| baseline | 91.0% | 13.9% |
| no handling | 59.6% | 4.5% |
| add unknown | 93.7% | 56.9% |
| suffix & unknown | 95.1% | 72.6% |
| best handling | 95.6% | 80.5% |

**Parameters**: Bi-gram, beam search (width=4), smoothing by add-k (k=0.5). No handling: Use the original words as the input for the HHM model. Add unknown: set the rare (occurrence =1,2) and unknown words (occurrence =0) all as *unknown*. Suffix & unknown: take last three characters as suffix if it is in length-3 suffix list which created by for rare words (occurrence =2) in training data and set the rest as *unknown*.

**Results**: No handling of unknown/rare words results in worst accuracy (59.6%) in dev. data, which is even much worse than the baseline. By assigning all unknown/rare words as *unknown* largely helps the tagging accuracy (56.9% on unknown words and 93.7% on overall words). If we assign rare words as their last three characters suffix and assign unknown words as *unknown*, we get better accuracy both on unknown and overall (72.6% and 95.1% respectively). And if we use comprehensive unknown/rare words handling as described in Section 3.3 where chunking, suffix and *unknown* are used, we get the best performance among all the handling (80.4% on unknown and 95.5% on overall).

**Analysis**: There is a strong correlation between unknown and overall accuracy. By comparing no handling methods and other handling methods, and considering the small proportion of rare/unknown words, it suggests that the error rate in no handling model is from the sequential affect of the low accuracy of unknown words. In other words, the low accuracy unknown words influence the prediction of common words behind it.

**4.3 Greedy, beam search and Viterbi**

**Table 2** Results with different search methods

| | accuracy | | | |
|---|---|---|---|---|
| **methods** | **overall** | **unknown** | **sub-rate** | **true-rate** |
| k=1 | 94.8% | 79.9% | 18.8% | 33.3% |
| k=2 | 95.4% | 81.0% | 4.49% | 37.8% |
| k=3 | 95.5% | 80.6% | 1.23% | 38.3% |
| k=4 | 95.6% | 80.5% | 0.71% | 38.3% |
| k=5 | 95.5% | 80.4% | 0.25% | 38.3% |
| k=10 | 95.6% | 80.3% | 0.029% | 38.3% |
| Viterbi | 95.6% | 80.7% | 0 | 38.2% |

Sub-rate: suboptimal sequence rate, true-rate: Correct sequence rate.

**Parameters**: Bi-gram, smoothing by add-k (k=0.5), unknown/rare words handling as described in Section 3.3. Different

beam widths and Viterbi are experimented here.

**Results**: The results that includes accurate, sub-optimal sequence rate and correct sequence rate are shown in above Table 2. Generally speaking, width of beam has small influence on the accuracy, compared to unknown words handling. Among them, width=1 shows relative worse performance on overall data than other width. As for suboptimal sequence rate, there is a obviously drop from width=1 (18.8%) to larger width (4.49% when width=2, 0.71% when width=4) (Viterbi has 0 sub-optimal rate). Once the width is larger than 2, the correct sequence rate is relative stable at 38.3%

**Analysis**: A larger beam width will have better approximation to the global optimal. This better approximation is shown in the table and especially when we compare results from width=1 to other larger width. However, in practice, when the accuracy is very near global optimal accuracy (Viterbi), more approximation does not necessary help the overall accuracy. This explains why the performance difference between width larger than 2 is small.

### 4.4 Gram size

**Table 3** Results with different gram size

| | accuracy | |
|---|---|---|
| **methods** | **overall** | **unknown** |
| bi-gram, k=4 | 95.6% | 80.5% |
| tri-gram, k=4 | 95.6% | 80.4% |
| four-gram, k=4 | 94.9% | 79.8% |
| bi-gram, Viterbi | 95.6% | 80.3% |
| tri-gram, Viterbi | 95.6% | 80.6% |

**Parameters**: smoothing by add-k (k=0.5 for bi-gram transition, 2 for tri-gram transition), unknown/rare words handling as described in Section 3.3. Beam widths 4 and Viterbi with different gram size are experimented here.

**Results**: The beam search (k=4) accuracy results indicate that increase the size of gram models doesn't show improve in both overall and unknown word tag accuracy. Also, we observed that bi-/tri-gram models performed closely well but not the four-gram model. We observed similar trend in Viterbi, where bi-gram and tri-gram have the same overall accuracy and tri-gram has a slight improvement in unknown word accuracy of 0.3%.

**Analysis**: By increasing the grams, it will increase the search space sparsity and possible noise for transition and emission probabilities. The bi-gram and tri-gram model showed very similar accuracy while it drops when we increase the gram size to four. When the width hyperparameter of beam search k is equal to 4, it performs the same as Viterbi which suggests the suboptimal rate is relatively low when k equals 4.

### 4.5 Error analysis

Here we analyze the error of our final model described in Section 3.6. We selected 17 most common tags and calculated the logged confusion matrix shown in the heatmap. We observed that the most common errors occurred in the predicted Adjectives where the prediction mis-classifies *NN* and *NNP* into *JJ*. Interestingly we saw that there are also some *VBN* and *VBG* were misclassified as *JJ* because most of *VBN* and *VBG* end with -ed and -ing and they often functions as Adjectives. For example, one of this type error occurred in common words *traded* and *listed* where they were predicted as *VBN* but actually are *JJ*. This type of error is hard to handle even for human taggers because sometimes the past tense of a verb can be used to function as adjectives. In our training data only 2/92 and 5/41 of them have tags *JJ*, others are *VBN, VBD*. The bias in training leads to this type of error. We also inspected the context of the word *traded* where it appears in the same context of *publicly traded* both in the training and dev. data. However, its true labels differs from *VBN* to *JJ*. We conclude that this labeling error is actually due to tag ambiguity (could be human error). In addition, we saw two clusters one with Noun-type, *NN*, *NNP*, *NNPS* and *NNS* and one with Verb-type, *VBP*, *VBN*, *VBG* and *VBD*. Because their differences are subtle, we saw majority of errors occurred within these two bigger groups. For example, we saw the word *Funds* were mistagged as *NNS* where its true tag is *NNPS*. It happens that in our training sample all six occurrence of *Funds* were tagged as *NNS* but we mistagged it in the dev. data where it has the tag of *NNPS*. We further inspect the context of *Funds*, we found it occurs twice in the same context *Closed End Bond Funds* in both training and dev. data. But the label differs where training gives *JJ,NN,NN,NNS* while development gives *NNP,NNP,NNP,NNPS*. This error is due to human tagging ambiguity as well as lack of training data. Our error analysis suggests that curating human taggings, adding more training data to reduce training bias will improve the tagging results. Our accuracy of 0.9610 is probably close to the upper bound given the human tagging error.

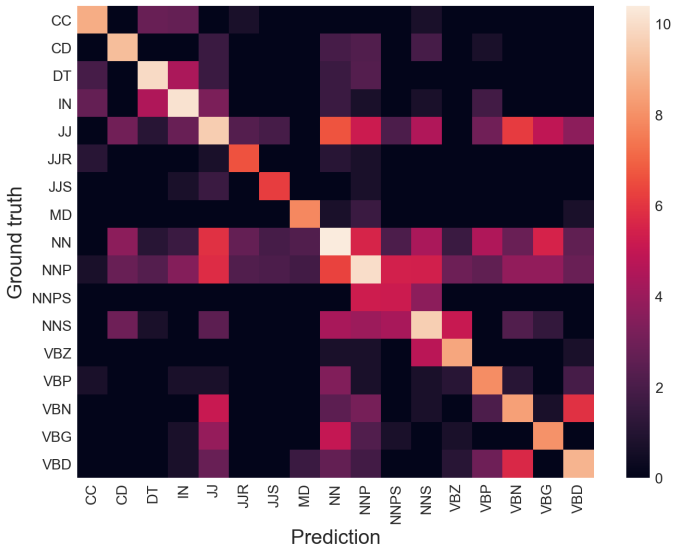


Figure 3: Confusion matrix of final model

## 5 Conclusion

To summarize, based on the HMM, we explored different unknown/rare words handling, search methods (greedy, beam, Viterbi) and different gram size. Failure of handling unknown/rare words results in very low accuracy due to its sequential influence on common words. Through careful feature engineering, we applied chunking to those sparse words based on their shared properties. We noticed that suffix is also very useful in predicting POS tags of the words. Adding unknown token also helps in improving the unknown word accuracy. Interestingly we didn't observe any substantial improvement by increase the gram size for both beam and Viterbi search, perhaps due to the limitation now is not the context information but possibly unknown words handling and smoothing. For the comparison between greedy, beam and viterbi, we observed that the greedy search showed the worst performance. By increasing the width k of beam search, the accuracy improved but started to saturate after k=4. Besides, beam search showed very similar accuracy with Viterbi algorithm. In the future, a better smoothing method can be implemented to improve our model.