CS5012 - Language and Computation
Practical 1 POS tagging with HMMs

180008901

This assignment is to implement POS taggers using three algorithms; Viterbi algorithm, beam search algorithm and forward-backwards algorithm. The Brown corpus with the universal tagset is used as training data. Witten-Bell smoothing is used to avoid zero probability which is likely to occur in NLP problems. I used bin size of the smoothing parameter = 100000 as an estimated number of words in current used English according to the Oxford dictionary.

To measure the accuracy, I separated the corpus into two sets of data; training data (54473 sentences) and testing data (2867 sentences). The gold standard tags are used to compare and estimate %accuracy of the model.

To validate the correctness of the program, `assert` command is used to detect unexpected probability, especially, in the forward-backwards approach; according to these equations

- $P(w_1^n) = backward[q_0, 0] = farward[q_F, n+1]$
- $\sum_{t_i} backward[t_i, i] forward[[t_i, i] = P(w_1^n)$

Running

```
python P1_POS <algorithm> <do_accuracy_test> <test_sentence>
```

<algorithm> : select one of these options; viterbi, beam, fwbw
<do_accuracy_test> : select one of these options; yes, no
<test_sentence> : (optional) testing sentence

Example
python P1_POS.py viterbi yes "I like cats"

Discussion

The accuracy is measured using the following formula;

$$\%\ accuracy = \frac{number\ of\ correct\ tags}{number\ of\ incorrect\ tags + number\ of\ correct\ tags} \cdot 100\%$$

In this implementation, the execution time is defined as the time it used to count and build up the probability tables -- training time and the time it used to predict tags for testing data(all 2867 sentences) -- predicting time. The % accuracy and execution time are shown in table 1-2. All measurement could be reproduced by re-running the code.

I found that forward-backwards probability is the best in term of accuracy but it took more time to give a prediction. This longer time might cause by its mathematical complexity which requires to compute probabilities in two directions.

In contrast, the beam search has slightly lower accuracy but it has major benefits in term of predicting time. Its parameter K plays a key role to balance its accuracy and computational time. The larger K parameter, the more accurate it becomes but the longer time it needs.

	Viterbi	Beam search (K=1)	Beam search (K=2)	Beam search (K=2)	Forward-backward
%accuracy	95.86	94.42	95.81	95.86	**95.88**

Table 1 shows % accuracy of each algorithm

	Training Time (s)	Predicting time (s)
Viterbi	10.29	13.67
Beam search (K=1)	10.08	2.13
Beam search (K=2)	10.12	3.20
Beam search (K=3)	10.39	4.59
Forward-backward	10.19	27.76

Table 2 shows % accuracy by tag and overall accuracy

Extension

Closer Analysis

As shown in table 3, the %accuracy for each tag was measured. It pointed out that the majority mistake comes from the tag “X”; other. Without considering tag “X”, the models could achieve up to 95.99% and 96.01% in Viterbi algorithm and forward-backwards algorithm respectively. One reason that could explain this problem is lacking example words which I found that there are only 51 in 2867 sentences that has tag X.

In addition, I found that the models have a more accurate prediction of tags from closed class for example; punctuation marks, pronoun, and conjunction. This could be explained by the characteristic of closed-class tags which usually be a small set of words and fixed.

Another observation suggests that the models' accuracy could be improved by considering grammatical information. Prefix and suffix are very informative in English for example, P(-tion| NOUN) should have a high probability for noun so having -tion might be a good identifier for tag NOUN. Phrasal verbs are also interesting because they usually do not occur as an adjacent word so using transmission probability from far neighbours (the higher-order HMM) might provide a more accurate model.

Tag	Viterbi	Beam search (K=1)	Beam search (K=2)	Beam search (K=2)	Forward-backward
ADV	**89.31**	85.06	89.16	89.31	89.24
NOUN	95.57	**96.11**	95.94	95.66	95.81
ADP	95.29	91.12	**95.37**	95.29	95.07
PRON	**99.04**	95.52	98.98	**99.04**	98.90
DET	97.25	97.12	97.15	97.27	**97.31**
.	99.98	**100.00**	99.98	99.98	99.99
PRT	**89.05**	81.94	88.93	**89.05**	88.88
VERB	96.03	95.46	**98.99**	96.03	96.14
X	**35.35**	12.12	21.21	21.21	34.34
NUM	97.88	**98.59**	97.88	97.88	97.88
CONJ	99.64	99.58	**99.70**	99.64	99.64
ADJ	**89.63**	85.81	88.41	89.66	89.49

Overall	95.86	94.42	95.81	95.86	**95.88**

Table 3 shows % accuracy by tag and overall accuracy

Different corpora

I experimented further with different corpora and different categories. It can be run by

```
python P1_ExtensionDiffCorpus.py <corpus>
```

<corpus> : select one of these options; brown, treebank, nps_chat, conll2000

From table 4, I found that the %accuracy, in general, has no significant differences between brown, treebank and CoNLL-2000 corpora because they usually include all general topics which result in the similarly distributed probability. A small accuracy drop in Treebank and CoNLL-2000 might be because of its data size.

The interesting result is the model from NPS Chat Corpus which is a dataset gathered from online chat services. The result was only 90% accuracy. I believed that because of the ambiguity and grammatical flexibility of the resource. Working with spoken language might be necessary to use more data to cope with its complicated behaviour. Moreover, the semantics of the data is also likely to depend on dialogue so analysis sentence as an individual could not distil the original intention of the sentence.

Lastly, I found that working with the dataset in different categories gave a different result. As you can see from Table 5, the model from news data gains 97.17% for noun but only 80.79% for adjective. A possible explanation for these might be because it has a different writing style. News tends to be informative and emotionless so it uses more noun but not many adjectives. In contrast, fiction uses more verb and adverb so it gains better accuracy in those tags. However, the models are likely to have the same error rate for closed-class tags such as DET because they have fixed grammatical function in the sentence so it will not have a vital difference across the categories.

Corpus	Data size	%Accuracy		
		Viterbi	Beam search K=1	Forward-backward
Brown	57340	95.86	94.42	95.88
Treebank	3914	94.36	93.85	94.48
CoNLL-2000	10948	95.68	95.17	95.74

NPS Chat Corpus	10567	90.28	90.76	90.57

Table 4 shows % accuracy from different corpora

Corpus/Category	%Accuracy of the forward-backwards model					
	NOUN	VERB	ADV	ADJ	DET	Overall
Brown	95.81	96.14	89.24	89.49	97.31	95.88
Brown/news	97.17	91.23	85.33	80.78	97.99	94.42
Brown/fiction	93.16	92.14	85.45	77.49	98.02	94.03
Brown/romance	91.85	91.11	74.59	76.30	98.92	92.71

Table 5 shows % accuracy from different categories

Beam widths

As I discussed earlier, table 1 clearly shows that parameter K is a key factor that balance tradeoff between accuracy and time usage. It is important when the model has too many hidden states (in this problem is the size of tagset) because the time complexity of the model depends on a number of these hidden states. In term of accuracy, beam search with lower parameter K is likely to have lower overall accuracy but it is randomly good in certain tags which is not a significant difference.

Different smoothing techniques

Lastly, I tried three smoothing techniques as shown in table 6. I used Viterbi model with no smoothing as the control group. The result can be seen by running the following command.

```
python P1_ExtensionDiffSmoothing.py
```

Table 6 clearly shows that smoothing is an important component to gain better results. Because some words in the target sentence might never be observed in dataset resulting in zero probability which will propagate and change all values in the Vertabi table into zero. It resulted in 49.57% accuracy.

In Laplace smoothing, the idea is to assume that the prior probabilities of all the events are equal. All events are upgraded with the same constant (add +1). Because of this adding, it generally gives a huge bias toward a rare word and unseen words which results in poor accuracy (92.7%).

Lastly, Good-Turing smoothing, due to unknown circumstance, the NLTK library prompted up that “UserWarning: SimpleGoodTuring did not find a proper best fit line for smoothing probabilities of occurrences. The probability estimates are likely to be unreliable”. So I choose not to use it. However, I believe that it will give a better result because it fairly adjusts the distributed probability. The major disadvantage might be that it requires difficult computation; for counting all frequency of the frequency of word and regression process so it needs more computational resources and time.

	No Smoothing	Laplace Smoothing	Witten-Bell Smoothing	Good-Turing Smoothing
%accuracy	49.57	92.70	95.86	* no result *

Table 6 shows % accuracy of smoothing techniques