CSCI 5832: Homework 2
Part of Speech Tagging

- By Payoj Jain

Overview

The aim of the assignment is to explore the statistical approach to part-of-speech tagging. The model has been trained using POS-tagged data from the Berkeley Restaurant corpus with over 15000 English sentences.

Below is the list of all the assumptions that have been taken into account while developing the model:

- New words may or may not occur in the test set.
- No new POS tags will appear in the test set.
- Words appearing in the train and test set are in English Language and thus, no text encoding and decoding is done in the code.
- While developing the model I have split the provided training data into *train set (90% of the provided training data) and development (dev) set (remaining 10%).* Also, I have randomized the training data while before splitting it into train set and dev set as lot sentences in the training data were repeated and to ensure that all possible type of sentences could be included in the training set

Baseline system

As asked in the requirements, I implemented baseline system by assigning the most frequent tag for a particular word in the training set.

Also, to deal with unknown/unseen words in the development/test set, I used rare words (with cutoff = 1) as a measure to assign tags to such words. The most frequent tag occurring for rare words in the train set has been assigned to the unknown/unseen words in the development/test set.

After implementing just these two things the system achieved an accuracy of around 93%.

```
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6
Evaluated  15233.0  tags.
Baseline Accuracy is:  0.9394735114553929

Process finished with exit code 0
```

Viterbi

To implement Viterbi there were certain calculations and implementations that needed to be done. These were:

1) ***Extract the required counts from the training data to generate the required probability estimates for the model.***

- List of all part-of-speech tags, their total unigram counts, their bigram counts, count a particular tag appeared at the end of a bigram type, count a particular tag appeared at the beginning of a bigram type, count of a particular tag starting a sentence, count of a particular tag ending a sentence, probabilities corresponding to these counts which are useful for smoothing, calculating emission probabilities of words and transmission probabilities of tags.

  Below figure shows an entryv ('.') in tag_dictionary. Tag_dictionary is a dictionary which has tags as its key and dictionaries of counts, transmission probabilities etc. as its values.

```
Tag Dictionary


{'.': {'Count_ending_sentence': 14279,
       'Count_in_1st_place': 0,
       'Count_in_2nd_place': 25,
       'Count_starting_sentence': 0,
       'Probability_after_smoothing': {'.': 0.0,
                                       ':': 0.011617100371747211,
                                       'CC': 0.0003241188062602011 6,
                                       'CD': 0.012197707161694151,
                                       'DT': 0.00928382010418101,
                                       'EX': 0.0006901247745924 03,
                                       'FW': 0.13816350783106832,
                                       'HYPH': 0.000358829355111883,
                                       'IN': 0.00545685906728242,
                                       'JJ': 0.09079678242003629,
                                       'JJR': 0.043391365158820715,
                                       'JJS': 0.06397053730171558,
                                       'LS': 0.004978757302177376,
                                       'MD': 7.081998420347137e-05,
                                       'NN': 0.3429439031991927,
                                       'NNP': 0.24628252788104088,
                                       'NNS': 0.49705622590431614,
                                       'PDT': 0.00030841859394019144,
                                       'POS': 0.11903546433495646,
                                       'PRP': 0.011200080357025754,
                                       'PRP$': 0.009976364220478222,
                                       'RB': 0.1368804213135068,
                                       'RBR': 0.006454351198665223,
                                       'RBS': 0.003485130111524163,
                                       'RP': 0.633786369549567,
                                       'TO': 5.993344989723411e-05,
                                       'UH': 0.1366092617378643,
                                       'VB': 0.02325618265234781,
                                       'VBD': 0.020234798562485827,
                                       'VBG': 0.02668599591759804,
                                       'VBN': 0.2601732607541158,
                                       'VBP': 0.01123719684237473,
                                       'VBZ': 0.030257251454718533,
                                       'WDT': 0.00126078417619415,
                                       'WP': 0.0005065596092331633,
                                       'WRB': 0.0026946253806886537},
       'Probability_continuation': 0.046468401486988845,
       'Probability_of_ending': 0.9975550122249389,
       'Probability_of_starting': 6.985679357317499e-05,
       'Total_count': 14279},
```

- List of words, their total counts, tags which they have been assigned in the train set, count of tags a particular word has been tagged to.

  Below is the figure which shows words as keys for word_dictionary and various dictionaries like counts, tags etc. as values.

```
Word Dictionary


{"'": {'Count_in_1st_place': 65,
       'Count_in_2nd_place': 65,
       'Count_starting_sentence': 0,
       'Tags': {'POS': {'Count': 65, 'Probability': 0.06103286384976526}},
       'Total_count': 65},
 "'d": {'Count_in_1st_place': 1125,
        'Count_in_2nd_place': 1125,
        'Count_starting_sentence': 0,
        'Tags': {'MD': {'Count': 1125, 'Probability': 0.2540076766764507}},
        'Total_count': 1125},
 "'ll": {'Count_in_1st_place': 81,
```

- List of all the bigram types and their total counts appeared in the train set.

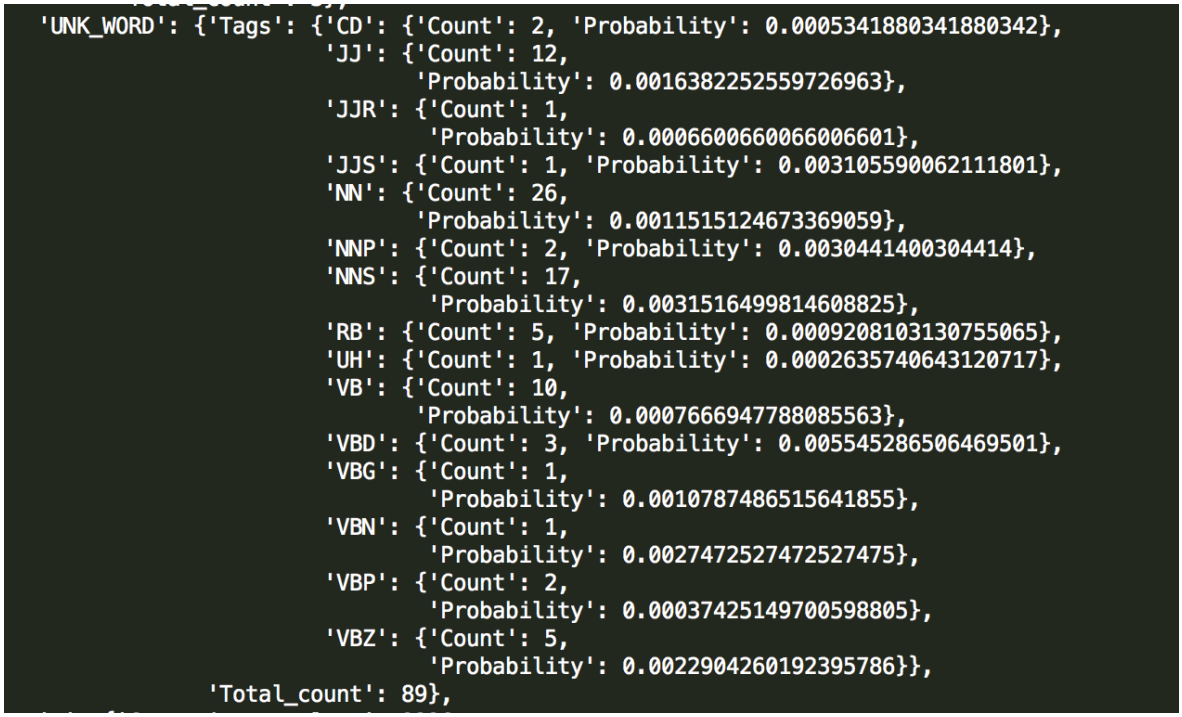
  Below figure shows bigram_types dictionary which has all the bigram types that appeared in the train set as its key and count of that bigram type in the train set as its value

```
Bigram Type


{(':', 'NN'): 3,
 ('CC', 'CC'): 9,
 ('CC', 'CD'): 206,
 ('CC', 'DT'): 174,
 ('CC', 'IN'): 91,
 ('CC', 'JJ'): 174,
 ('CC', 'JJR'): 70,
 ('CC', 'MD'): 22,
 ('CC', 'NN'): 199,
 ('CC', 'NNP'): 45,
 ('CC', 'NNS'): 141,
 ('CC', 'PRP'): 450,
```

2) ***Deal with unknown words:*** In the train set, words which rarely appear (less than or equal to cutoff) are considered as "UNKNOWN WORDS". Any new/unseen word appearing in test/dev set will be given emission probability based on the count of "UNKNOWN WORDS" from train set.

   Below is the figure which shows count and emission probability of 'UNK_WORD' given a part-of-speech tag



3) *Do some form of smoothing for the bigram tag model.*

- Two kinds of smoothing methods have been implemented to smooth transmission probabilities of bigram part-of-speech tags. Apparently, Kneser-ney performed better than Laplace smoothing for the given set of train and dev data.

i. Kneser-ney smoothing

```
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6
Evaluated  15233.0  tags.
Accuracy using Kneser-ney smoothing is:  0.9629094728549858

Process finished with exit code 0
```

ii. Laplace smoothing

```
/Library/Frameworks/Python.framework/Versions/3.6/bin/python3.6
Evaluated  15233.0  tags.
Accuracy using Laplace smoothing is:  0.9617934746931005

Process finished with exit code 0
```

I have also handled the probability of a tag starting or ending a sentence by Laplace smoothing. Since, in the train set almost all sentences were ended by '.' but there is a possibility that some other tag may end the sentence in dev/test set. Also, there were many tags which were never at the start of a sentence in the train set but this doesn't mean that the same tag won't appear at the beginning of a sentence in dev/test set.

Implementing Viterbi decoding and backtracking the path on Viterbi matrix to get the most probable tags for the given set of test/dev data shows result which looked like below.

Viterbi Matrix

	(0, 'tell')	(1, 'me')	(2, 'more')	(3, 'about')	(4, 'sushi')	(5, '-')	(6, 'ko')	(7, '.')
.	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-34.73613968618416
:	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
CC	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
CD	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
DT	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
EX	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
FW	-inf	-inf	-inf	-inf	-23.548946713914262	-inf	-inf	-inf
HYPH	-inf	-inf	-inf	-inf	-inf	-26.348477146513016	-inf	-inf
IN	-inf	-inf	-inf	-14.046991459067376	-inf	-inf	-inf	-inf
JJ	-inf	-inf	-inf	-inf	-21.275845094149048	-inf	-inf	-inf
JJR	-inf	-inf	-13.41797896014385	-inf	-inf	-inf	-inf	-inf
JJS	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
LS	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
MD	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
NN	-inf	-inf	-inf	-inf	-22.537846762776393	-inf	-33.66595129346839	-inf
NNP	-inf	-inf	-inf	-inf	-24.032762882407788	-inf	-inf	-inf
NNS	-inf	-inf	-inf	-inf	-23.410976343265812	-inf	-inf	-inf
PDT	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
POS	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf
PRP	-inf	-8.400751242032817	-inf	-inf	-inf	-inf	-inf	-inf
PRP$	-inf	-inf	-inf	-inf	-inf	-inf	-inf	-inf

Backtrace Matrix

	(0, 'tell')	(1, 'me')	(2, 'more')	(3, 'about')	(4, 'sushi')	(5, '-')	(6, 'ko')	(7, '.')
.	0	None	None	None	None	None	None	NN
:	0	None	None	None	None	None	None	None
CC	0	None	None	None	None	None	None	None
CD	0	None	None	None	None	None	None	None
DT	0	None	None	None	None	None	None	None
EX	0	None	None	None	None	None	None	None
FW	0	None	None	None	IN	None	None	None
HYPH	0	None	None	None	None	NN	None	None
IN	0	None	None	RBR	None	None	None	None
JJ	0	None	None	None	IN	None	None	None
JJR	0	None	PRP	None	None	None	None	None
JJS	0	None	None	None	None	None	None	None
LS	0	None	None	None	None	None	None	None
MD	0	None	None	None	None	None	None	None
NN	0	None	None	None	IN	None	HYPH	None
NNP	0	None	None	None	IN	None	None	None
NNS	0	None	None	None	IN	None	None	None
PDT	0	None	None	None	None	None	None	None
POS	0	None	None	None	None	None	None	None
PRP	0	VB	None	None	None	None	None	None
PRP$	0	None	None	None	None	None	None	None
RB	0	None	None	RBR	None	None	None	None
RBR	0	None	PRP	None	None	None	None	None
RBS	0	None	None	None	None	None	None	None
RP	0	None	None	None	None	None	None	None
TO	0	None	None	None	None	None	None	None
UH	0	None	None	None	None	None	None	None
VB	0	None	None	None	None	None	None	None
VBD	0	None	None	None	None	None	None	None
VBG	0	None	None	None	None	None	None	None
VBN	0	None	None	None	IN	None	None	None

Evaluation on unseen test-set

After implementing Viterbi, smoothing, handling unknown words the accuracy increased from 93.9% (baseline accuracy) to 96.291% (using Kneser-ney smoothing). An increase of 2.3% on the data when the provided training data was split in 90%:10% ratio of train set: dev set. Since the number of repetition of sentences in the training data was high enough there is a possibility that the dev set might contain sentences which already appeared in the train set. Also, the train and dev set contained the similar kind of sentences i.e. the structure of sentences were similar. For example, 'i want chinese food.' Is quite similar to 'i want Italian food' and 'i want indian food'. Hence, there is a possibility of overfitting. Doing randomisation on the training data may avoid the risk of overfitting. But, since train set size wasn't enough there is a possibility that the model is not train very well. Therefore, if an unseen test data is tested on this model, I believe that accuracy should be greater than baseline accuracy which is around 93.9% but should be less than 96.291%.

Thus, for an unseen data the model accuracy should be less that 96.291%.