Sunil Murthy
April 7, 2017

Natural Language Processing

Programming Assignment 3 Report

In this assignment I created a Parts of speech tagger using Hidden markov model (Viterbi algorithm).

Development file:
The given training file *"berp-POS-train.txt"* contains around 15k sentences. Out of 15k only ~5k are unique sentences and rest of them are all duplicates. So this is a pretty closed domain with not much scope for improving the accuracy of the model.

For development testing purposes I randomly split the sentences into 80:20 ratio. And train the model on 80% of data and test it on 20%.

Tuning of HMM Model:
The training file and test file consists of only seen unigrams. But for the HMM model, we need to calculate tag transition probability and word likelihood probability as,

$$P(tag_i | tag_{i-1}) = \frac{C(tag_{i-1}, tag_i)}{C(tag_{i-1})} \qquad P(word | tag) = \frac{C(tag, word)}{C(tag)}$$

If the count of tag[i] and tag[j] is zero, then the probability becomes zero. In order to avoid such zero probabilities I smoothed the bigram probabilities, both tag-transition & likelihood using *add-k smoothing* as -

$$P(tag_i | tag_{i-1}) = \frac{C(tag_{i-1}, tag_i) + k}{C(tag_{i-1}) + kV} \qquad P(word | tag) = \frac{C(tag, word) + k}{C(tag) + kV}$$

Where *k* is constant and *V* is the vocabulary (i.e total unique tags).

Effect of add-k smoothing:
The below graph shows the effect of accuracy of the model over varying values of k smoothing. The accuracy in the second column is calculated using *evalPOSTagger.py*. And precision and custom accuracy are calculated using *POSstatus.py* using sklearn metrics module.

k	Accuracy (%)	Precision (%)	Custom Accuracy (%)
5	87.83	91.83	86.39
1	93.13	94.19	92.32
0	10.58	0	7.52
0.5	94.42	94.87	93.78
0.01	96.23	95.99	95.79
0.0000001	96.27	96.02	95.83

Table 1: Varying k with Accuracy, Precision and sklearn Accuracy.

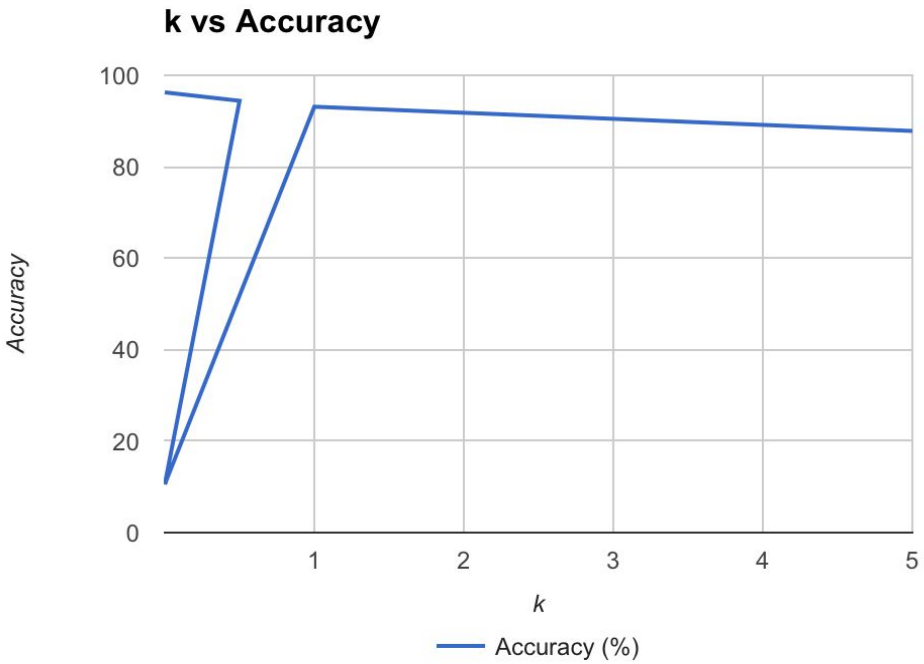


Figure-1: smoothing constant vs Accuracy

From the above graph it becomes clear that as, value of k increases the accuracy of the model drops. And accuracy increases as k becomes smaller and smaller but no exactly zero. When *k = 0 (without smoothing)*, the accuracy is terrible which is expected as there are many unseen bigram words/tags and their probabilities becomes zero.

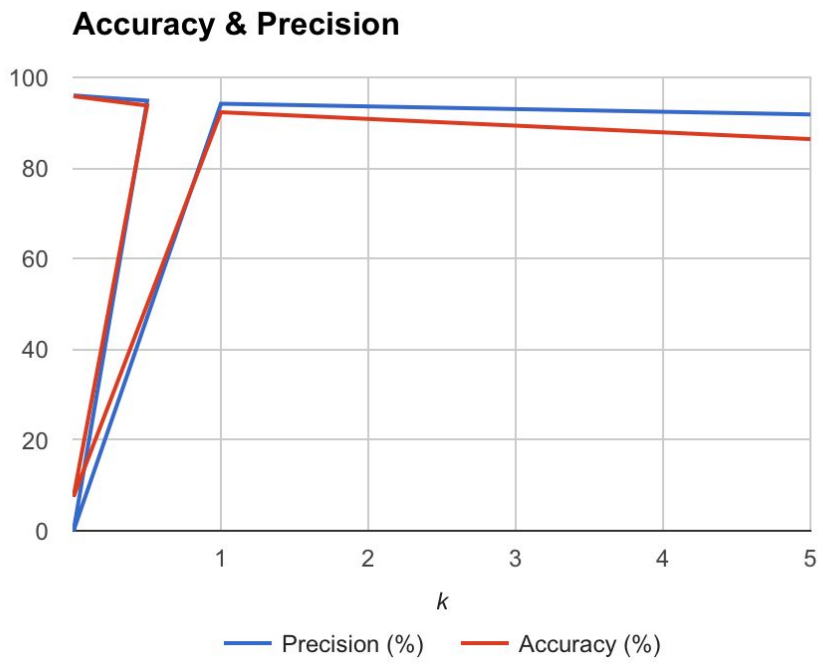


Figure-2: Accuracy vs Precision

From the above graph of accuracy & Precision it becomes clear that when k is really small, precision and accuracy are same, meaning that most (96%) of our predicted tags are accurate.

80/20 Split Validation:
To evaluate the model, I divided the training file sentences into 80:20 proportion. And used 80% for the training the model and 20% for testing. Only with varying k I was able to achieve an high accuracy of ~96% because the given language is very simple and contains repetitive sentences.

K-Fold Validation:
Apart from doing the 80:20 split validation, I also did the K-fold validation where K = 5 fold. With smoothing value k = 0.0000001. I still obtained the same accuracy of ~96%. The python script is called *"kfold.py"*.

To run the program: *python3 kfold.py <training file>*

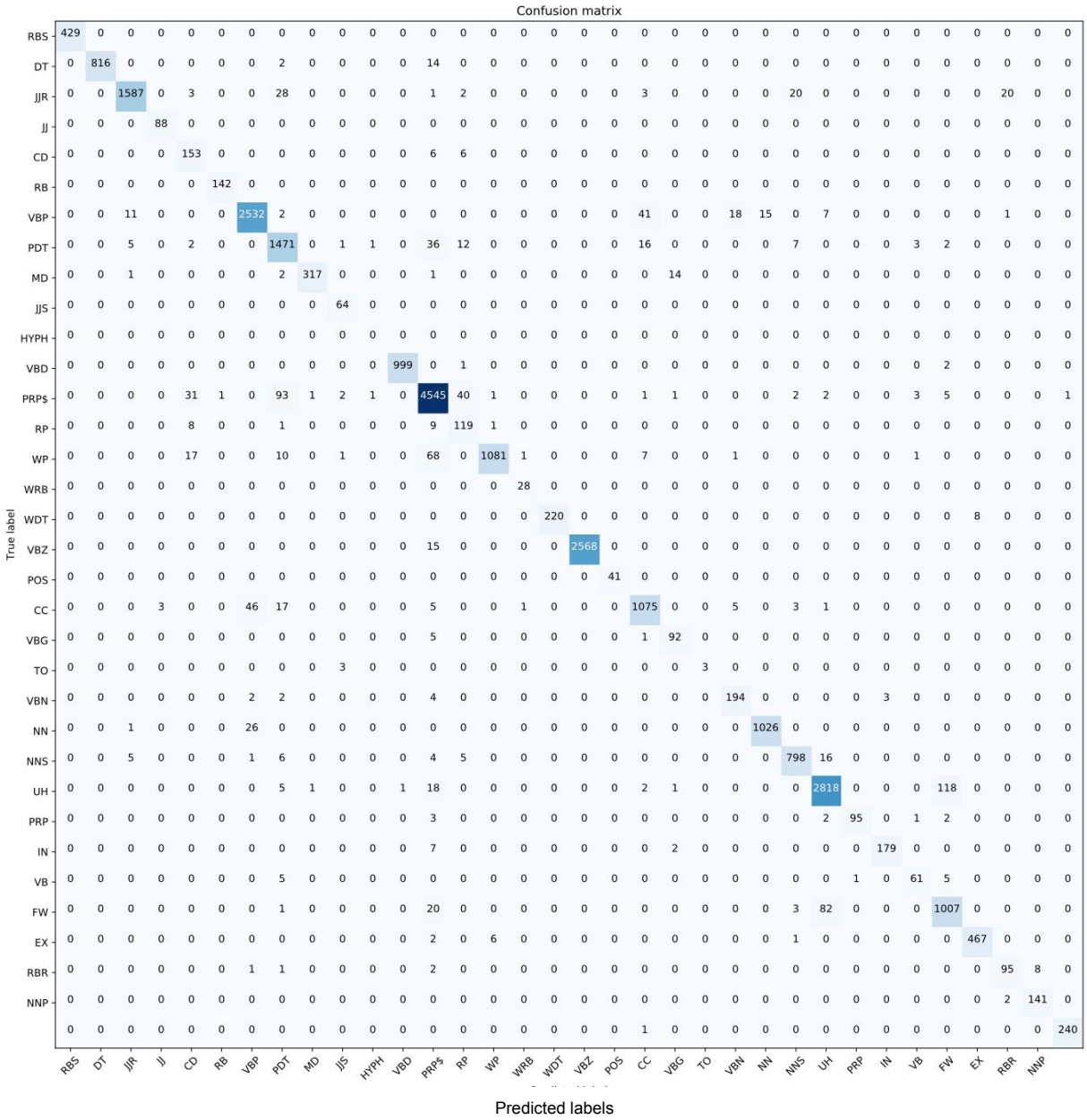
Confusion matrix:
I created a confusion matrix to help me understand the most commonly misclassified tags. The below figure shows a confusion matrix created from development test file.

From the confusion matrix, it's evident that the tag UH has been wrongly tagged as FW 118 times. Here's are some of the top mis-tagged words.

True label	Predicted label	Times

UH	FW	118
PRP$	PDT	93
FW	UH	82
CC	VBP	46

Table-2: Top misclassified tags, *k* = *.0000001*



To create the confusion matrix/ custom accuracy & precision:

$ python3 POSstats.py <gold-file> <output-file>

Also, *POSstats.py* has dependencies on ***numpy, sklearn & matplotlib***.