Name: Mayank Baheti

Topic: Training our own Part-Of-Speech Tagger

Part-Of-Speech tagging (or POS tagging, for short) is one of the main components of almost any NLP analysis. The task of POS-tagging simply implies labelling words with their appropriate Part-Of-Speech (Noun, Verb, Adjective, Adverb, Pronoun, …).

Data Description:

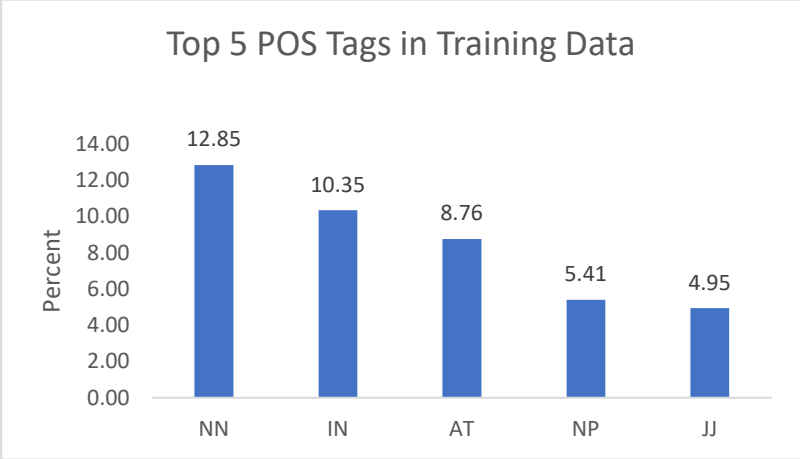
We have been given three files as below:

1. **train.txt**: tagged data on which model has to be trained.
2. **test.txt**: untagged data on which final model must be fitted.
3. **test.tag**: tagged data for the same text we had in test.txt file so that we can evaluate how our model has performed.

Output: test.out, which has tokens and the predicted tags, separated by '/'.

Primary Data Analysis:

Below are the top 5 most occurring **Part-of-speech tags**:



- ✓ Test.txt has 898 tokens for which we have to do the predictions for, but the validation set (test.tag) has only 892 tags as target, which will cause an issue will calculating the accuracy score. I have mentined in the approached how I have fitted the model on this data and the calculated the accuracy score.

Problem Statement:

Given the training data, we need to train the model and use it to tag the new data on its POS category.

Approach:

1. We have been given tagged training data which we have read in python using NLTK's function **TaggedCorpusReader** (Reader for simple part-of-speech tagged corpora. Paragraphs are assumed to be split using blank lines. By default, ``'/'`` is used as the separator. I.e., words should have the form: word1/tag1 word2/tag2 word3/tag3 ... Part of speech tags are case-normalized to upper case.)

2. The data is split into train and test datasets. So that we can train the model on training dataset and test it on testing dataset. Also along with the train.txt we have 'test.tag' file which we will use as our validation dataset.

3. Before starting training a classifier, we must agree first on what features to use. Most obvious choices are: the word itself, the word before and the word after. So for this we have made a custom function named 'features' which returns a dictionary of multiple keys which are our features for each word. Along with that we have used a Small helper function to **strip the tags from our tagged corpus** and feed it to our classifier.

4. Our classifier should accept features for a single word, but our corpus is composed of sentences. We'll need to do some transformations, for this 'transform_to_dataset' function which calls all our above custom-made function to make data ready to be used in a classifier model.

5. We have opted for RandomForestClassifier, which is used in Sklearn pipeline along with DictVectorizer as Vectorizer (Transforms lists of feature-value mappings to vectors).

6. While training the model we have just used 25K records as it was causing memory allocation issues. And later we have fitted this trained model on the test data which gave us the following accuracy measure:

   Precision: 0.8506806148727546,
   Recall: 0.8506806148727546,
   F1-Score: 0.8506806148727547.

7. We had a file of untagged data which we had to use as our validation data set, on reading and applying all the transformation, we found that it had 898 tokens, and the 'test.tag' file had 892 tags associated for the each token. So we just used the tokens from this tagged data to be used to fit the model and do the scoring against this 892 actual tagged POS. And the results are as follows:

   Precision: 0.8901345291479821,
   Recall: 0.8901345291479821,
   F1-Score: 0.8901345291479821.

Future Scope: We had 165816 words, in training data, but because for memory issues we were able to train only on 25K token's features. Given the infrastructure we could have trained on complete data and get better results.