PROJECT REPORT
CSCE 5290 NATURAL LANGUAGE PROCESSING
UNIVERSITY OF NORTH TEXAS, FALL 2016

Deep Analysis of POS tagger using HMM

Anvesh Athmakuri
anveshathmakuri@my.unt.edu
Student Id – 11116468

INTRODUCTION:

This project is to implement POS tagger by using Hidden Markov Model and comparing accuracy with different tagsets. In this POS tagger was implemented by using trigrams. By applying Good Tuning and Morphology rules for unknown words the accuracy was improved.

Programming Language:

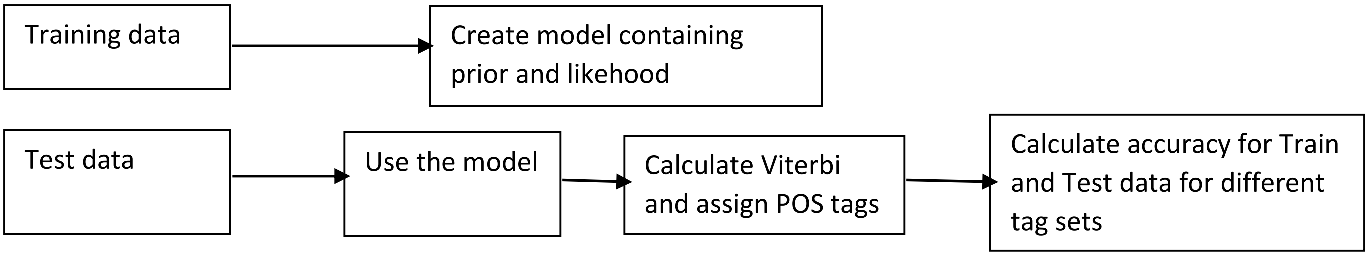
Python 2.7

Corpora:

Training and Test data are from HMM POS tagging assignment

- hw3_train
- hw3_heldout

Block Diagram:


Training data
Create model containing prior and likehood
Test data
Use the model
Calculate Viterbi and assign POS tags
Calculate accuracy for Train and Test data for different tag sets

Implementation Approach:

Creating Model:

Here Model is created on training data. The main purpose of Model is to get prior probabilities and Likehood probabilities.

- **Prior probability:**

  In this we have taken trigrams. So prior probability is probaility of getting a tag T_i after the tag T_{i-2} and Tag T_{i-1}. Here we store these probability values by using data structure “dictionary of dictionary of dictionaries”

  $P(T_i / T_{i-1}T_{i-2}) = Count(T_iT_{i-1}T_{i-2}) + 1 / Count(T_{i-1} T_{i-2}) + | Tagset |$

- **Likehood probabilty:**

  It is probability that a word get a tag $T_{i.}$. Here we store these probability values by using data structure “dictionary of dictionaries”.

  $P(w/T_i) = Count(w/T_i) / Count(T_i)$

From Model we are sending 5 values.

1. Prior Probabilty of trigrams
2. Likehood probability
3. Tagset (Unique tags from training data)
4. Word Types (Unique words from training data)
5. Good tuned smooth probabilty value

Smoothing Technique:

Here I have used two smoothing techniques

1. **Laplace Smoothing**: To smooth Prior probabilities values.
2. **Good Tuning:** To smooth Likehood probability values of Unknown words.

Prediction Method:

Handling Unknown Words:

First we find whether word is unknown or not by using word types obtained from model. If the word is identified as Unknown word then we are sending this as a parameter to a method named “getPossbileTagForUnknownWord” which contains **“Morphology rules”.** It returns a tag based upon pattern of unknown word. If the word does not match any one of the specified patterns then we are assigning “NN” tag to that word.

Morphology Rules:

It consists of some regular expressions to match the specific patterns. Based on the pattern of Unnown word a Tag is being set to that word.

Assigning POS Tags :

In HMM we assign POS tags by using Viterbi and Back tracking approach. Viterbi is implemented by using two matrices Viterbi1 and Viterbi2. Viterbi1 matrix stores the maximum viterbi value for each word and a tag. Viterbi2 stores the location of previous word and thag with whom the present word and tag get maximized value.

Command To execute:

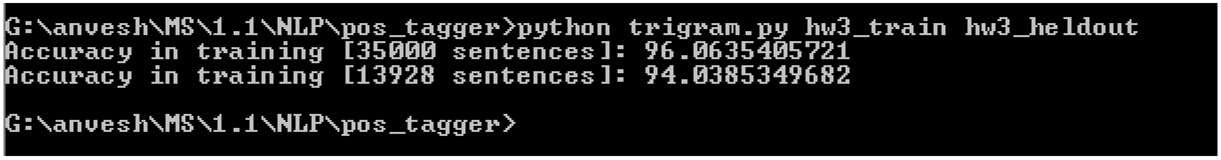
Python trigram.py hw3_train hw3_heldout

Different Tagsets:

Here I have done by using 4 different tagsets

1. **Penn Tree Bank Tagset:**
   It cotains all the tags specified in Penn tree bank

   Accuracy for using Penn Tree Bank tag set:

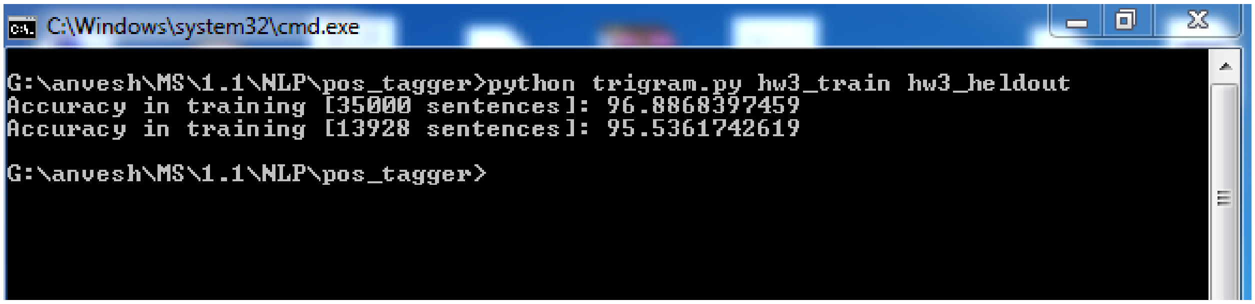


```
G:\anvesh\MS\1.1\NLP\pos_tagger>python trigram.py hw3_train hw3_heldout
Accuracy in training [35000 sentences]: 96.0635405721
Accuracy in training [13928 sentences]: 94.0385349682

G:\anvesh\MS\1.1\NLP\pos_tagger>
```

2. **Fine grained Tagset except Noun:**
   Here we changed all tags "NN","NNS","NNP",NNPS" to “NOUN”. Except this remaining all tags were kept as it is.

   Accuracy for using Fine grained Tagset except Noun:



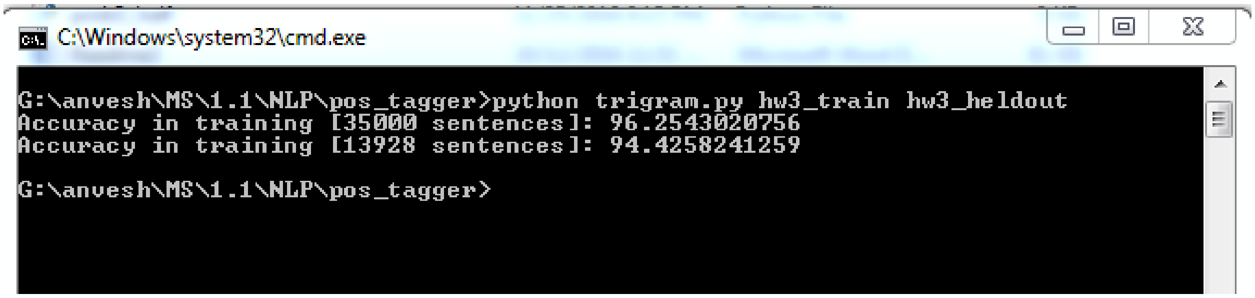
C:\Windows\system32\cmd.exe

```
G:\anvesh\MS\1.1\NLP\pos_tagger>python trigram.py hw3_train hw3_heldout
Accuracy in training [35000 sentences]: 96.8868397459
Accuracy in training [13928 sentences]: 95.5361742619

G:\anvesh\MS\1.1\NLP\pos_tagger>
```

3. **Fine grained Tagset except VERB:**
   Here we changed all tags "VB","VBD","VBG","VBN" ,"VBP","VBZ" to “VERB”.Except this remaining all tags were kept as it is.

   Accuracy for using Fine grained Tagset except VERB:

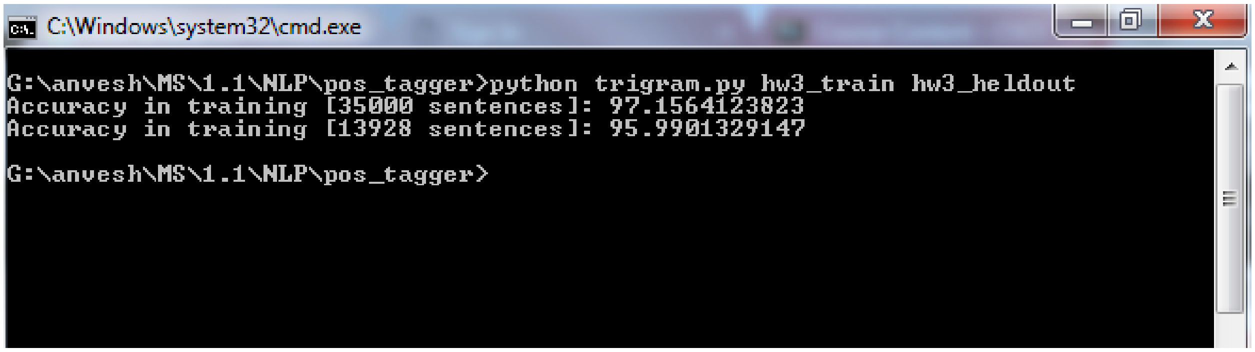


C:\Windows\system32\cmd.exe

```
G:\anvesh\MS\1.1\NLP\pos_tagger>python trigram.py hw3_train hw3_heldout
Accuracy in training [35000 sentences]: 96.2543020756
Accuracy in training [13928 sentences]: 94.4258241259

G:\anvesh\MS\1.1\NLP\pos_tagger>
```

4. **Fine grained Tagset except NOUN,VERB:**
   Here we changed all tags "NN","NNS","NNP",NNPS" to “NOUN” & "VB","VBD","VBG","VBN" ,"VBP","VBZ" to “VERB”.Except that remaining all tags were kept as it is.
   Accuracy for using Fine grained Tagset except NOUN &VERB:



```
G:\anvesh\MS\1.1\NLP\pos_tagger>python trigram.py hw3_train hw3_heldout
Accuracy in training [35000 sentences]: 97.1564123823
Accuracy in training [13928 sentences]: 95.9901329147

G:\anvesh\MS\1.1\NLP\pos_tagger>
```

Comparision between different tagsets:

Accuracy for using Fine grained Tagset except NOUN &VERB > Accuracy for using Fine grained Tagset except Noun > Accuracy for using Fine grained Tagset except VERB > Accuracy for using Penn Tree Bank tagset.

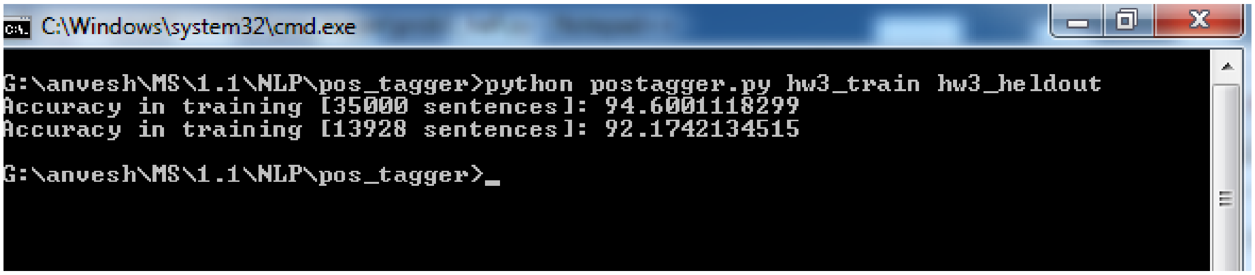
Reason:

Accuracy for Coarse grained tagset would be more than Fine grained Tagset why because there would be a chance of changing tag from one fine grained tag to other fine grained tag. For example the word “class” changed the tag from “NN” to “NNS”. Though it is a “NOUN” because of fine grained tagset it changed from “NNS” to “NN”. So because of thesekind of changes accuracy of Fine grained is less tha Coarse grained tagset. But we prefer to use Fine grained tag set because we get more information in fine grained tags so that it would be more helpful in later steps like Semantic Role labeling, Information extraction etc.For example it is very important to know whether the given noun is a proper noun or common noun in further NLP conncepts.

Accuracy of Training data would be more than Test data because there would be a lot of unseen words in test data whereas there would be no unseen word in Train data because model is created by using training data.

Accuracy by using trigrams would be more than Accuracy by using bigrams because the current tag not only depends on previous tag but also tags of Preceding two words.

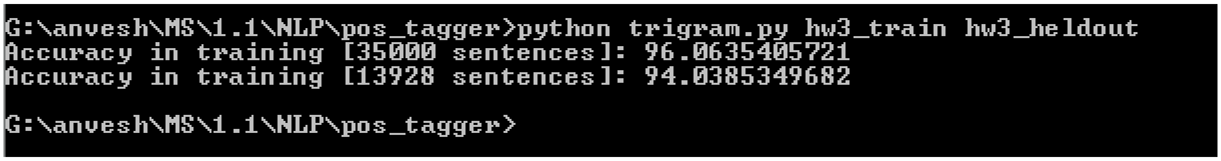
By using bigrams:



```
G:\anvesh\MS\1.1\NLP\pos_tagger>python postagger.py hw3_train hw3_heldout
Accuracy in training [35000 sentences]: 94.6001118299
Accuracy in training [13928 sentences]: 92.1742134515

G:\anvesh\MS\1.1\NLP\pos_tagger>_
```

By using Trigrams:



```
G:\anvesh\MS\1.1\NLP\pos_tagger>python trigram.py hw3_train hw3_heldout
Accuracy in training [35000 sentences]: 96.0635405721
Accuracy in training [13928 sentences]: 94.0385349682

G:\anvesh\MS\1.1\NLP\pos_tagger>
```

N-Gram	**Accuracy of Training Data**	**Accurcay of Testing Data**
Bigram	94.60%	92.17%
Trigram	96.06%	94.03%

Accuracy values of different tag sets:

Tagset	**Accuracy of Training Data**	**Accuracy of Test data**
Penn Tree bank tagset	96.06 %	94.03%
Fine grained Tagset except Noun	96.88%	95.53%
Fine grained Tagset except VERB	96.25%	94.42%
Fine grained Tagset except Noun & Verb	97.15%	95.99%

Conclusion:

Finally in this project we increased accuracy by using trigrams, Morphology rules and good tunned smoothing for unknown words. We also compared the accuracy of different tagsets. Though accuracy of Fine grained tagset is less than the coarse grained tagset but Fine grained is better because we get more information from fine grained tags than coarse grained tags which would be more helpful in further concepts of NLP.