NLP Assignment 4

Shreya Sharma (2015096)

POS Tagging using HMM

The folder contains 4 python files:

1. main.py
2. hmm_model.py
3. test.py
4. temp.py

- **main.py :** Reads the training_file and testing file. Calls the training and testing functions contained in hmm_model.py and test.py respectively.

- **hmm_model.py :** It counts the bigrams ({tag,tag} and {tag,word} for transition probability matrix and emission probability matrix respectively), counts the frequency of words at the starting position of the sentences for initial probability matrix, creates the vocabulary of words and set of tags found in the training data. All are in the form of dictionary except of word vocabulary which is a list.This is the training part of the model. All the structures are pickled.

- **test.py :** It implements the viterbi algorithm to test an unknown data.
  Smoothing has been used to calculate transition probabilities, emission probabilities and initial probabilities as well and instead of multiplication, I've taken log so as to avoid too small numbers multiplying and giving 0 as the result. The formulae used are as follows:

  Initial_Probability[tag] = log(C1(tag)+1) - log(Total_tags + T)

  Where C1(tag) = #times a tag occurs at the start of a sentence in the training data
  Total_tags = total tags that occur at the start of a sentence in the training data
  T = unique tags present in training data

  Emission_Probability[tag][word] = log(C2(tag,word) +1) - log(C3(tag)+V)

  Where C2(tag,word) = #times 'word' is tagged as 'tag'
  C3(tag) = #times tag occurs in the training data
  V = size of word vocabulary

  Transition_Probabilty[tagi][tagj] = log(C4(tagi,tagj) + 1) - log(C3(tagi)+T)

  Where C4(tagi,tagj) = #times the bigram (tagi,tagj) occurs in the training data in that order.
  T = size of the set of tags in the training data

Viterbi Algorithm:

```
Tag = list of set of tags in training data
word = sequence of words to be tested
For t = [0,…,T-1]
        V[0][t] = Initial_Probabilty[Tag[t]] + Emission_Probabilty[Tag[t]][word[0]]
For i = [1,..,N-1]
        For t = [0,…,T-1]
                V[i][t] = 0
                For t' = [0,…,T-1]
                        Temp = Transition_Probabilty[Tag[t_]][Tag[t]]
                        if(Temp > V[i][t])
                                V[i][t] = Temp
        V[i][t] = V[i][t]+Emission_Probabilty[Tag[t]][word[i]]

//backtrack
Path = []
Path <- Path + argmax,t(V[N-1][t])
For n = [N-2,…,0]
        Path <- Path + argmax,t(V[N-1][t]+Transition_Probability[Tag[t]][Tag[Path[N-2-n]]) for t = [0,…,T-1]
Path = reverse(Path)
Result = map Path values to Tag values to get the result
```

- **temp.py :** To get the training accuracy which is **93.653077538**

- **Sample Result:**

```
i           PRP
'd          MD
like        VB
to          TO
go          VB
to          IN
a           DT
fancy       JJ
restaurant          NN
.           .

i           PRP
'd          MD
like        VB
french      JJ
food        NN
.           .
```