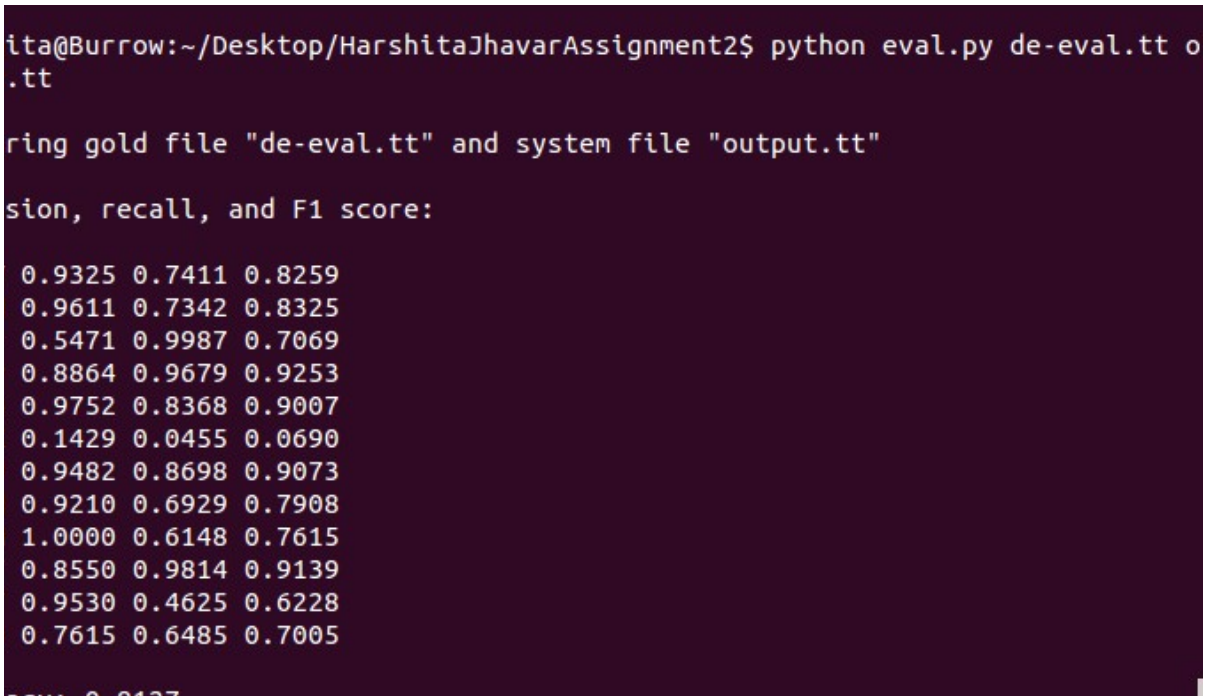
ASSIGNMENT 2

Submitted by – Harshita Jhavar, 255627, Msc in Computer Science, First Semester

These are the results obtained from my POS Tagger.



Instructions to run the POS tagger:

1. Make your current directory as my submission folder and run the command
   python viterbi.py
This will train the parser with the viterbi algorithm.

2. The output of this file will be generated in **output.tt** file.

3. Open the file output.tt, press Ctrl+H and replace “””” with “.
PS: I am not sure why this character was being treated differently.

4. After this, to evaluate the parser against the file eval.py, type the command
   python eval.py de-eval.tt output.tt

5. The results are as shown in the screenshot above.
The accuracy of the parser is 81.27%

Note: For the words which were unknown to the tagger, I have given it a tag of a dot - '.'

```
#To avoid integer division
from __future__ import division
from operator import itemgetter
import csv


###Reading the training data to train the parser line by line

with open("de-train.tt", "r") as f:
    training_string = f.read()
    training_line = training_string.split("\n")

## Calculating the number of rows in the training set
num_of_words_in_training = len(training_line)


training_line_words = ['']
training_line_words*= num_of_words_in_training

training_line_tags = ['']
training_line_tags*= num_of_words_in_training

## Reading the words and tags set separately
training_line_words = [element.split('\t')[0] for element in training_line]
training_line_tags = [element.split('\t')[-1] for element in training_line]

"""Nested dictionary to store the transition probabilities.Each tag A is a key of the outer di
ctionary.The inner dictionary is the corresponding value.
The inner dictionary's key is the tag B following A and the corresponding value is the number
of times B follows A
"""
dict2_tag_follow_tag_ = {}
dict2_word_tag = {}
"""Nested dictionary to store the emission probabilities.Each word W is a key of the outer dic
tionary. The inner dictionary is the corresponding value
The inner dictionary's key is the tag A of the word W and the corresponding value is the numbe
r of times A is a tag of W
"""
dict_word_tag_baseline = {}
#Dictionary with word as key and its most frequent tag as value

for i in range(num_of_words_in_training-1):
    outer_key = training_line_tags[i]
    inner_key = training_line_tags[i+1]
    dict2_tag_follow_tag_[outer_key]=dict2_tag_follow_tag_.get(outer_key,{})
    dict2_tag_follow_tag_[outer_key][inner_key] = dict2_tag_follow_tag_[outer_key].get(inner_k
ey,0)
    dict2_tag_follow_tag_[outer_key][inner_key]+=1

    outer_key = training_line_words[i]
    inner_key = training_line_tags[i]
    dict2_word_tag[outer_key]=dict2_word_tag.get(outer_key,{})
    dict2_word_tag[outer_key][inner_key] = dict2_word_tag[outer_key].get(inner_key,0)
    dict2_word_tag[outer_key][inner_key]+=1


"""The 1st token is indicated by being the 1st word of a senetence, that is the word after per
iod(.)
Adjusting for the fact that the first word of the document is not accounted for that way
"""

dict2_tag_follow_tag_['.'] = dict2_tag_follow_tag_.get('.',{})
dict2_tag_follow_tag_['.'][training_line_tags[0]] = dict2_tag_follow_tag_['.'].get(training_li
ne_tags[0],0)
dict2_tag_follow_tag_['.'][training_line_tags[0]]+=1


last_index = num_of_words_in_training-1

#Accounting for the last word-tag pair
outer_key = training_line_words[last_index]
inner_key = training_line_tags[last_index]
dict2_word_tag[outer_key]=dict2_word_tag.get(outer_key,{})
dict2_word_tag[outer_key][inner_key] = dict2_word_tag[outer_key].get(inner_key,0)
dict2_word_tag[outer_key][inner_key]+=1


"""Converting counts to probabilities in the two nested dictionaries
```

```
& also converting the nested dictionaries to outer dictionary with inner sorted lists
"""
for key in dict2_tag_follow_tag_:
    di = dict2_tag_follow_tag_[key]
    s = sum(di.values())
    for innkey in di:
        di[innkey] /= s
    di = di.items()
    di = sorted(di,key=lambda x: x[0])
    dict2_tag_follow_tag_[key] = di

for key in dict2_word_tag:
    di = dict2_word_tag[key]
    dict_word_tag_baseline[key] = max(di, key=di.get)
    s = sum(di.values())
    for innkey in di:
        di[innkey] /= s
    di = di.items()
    di = sorted(di,key=lambda x: x[0])
    dict2_word_tag[key] = di



###Testing with the given test file###

with open("de-test.t", "r", encoding="utf-8") as myfile:
    testing_str = myfile.read()
testing_line = testing_str.split()
num_words_testing = len(testing_line)

test_line_words = ['']
test_line_words*= num_words_testing

test_line_tags = ['']
test_line_tags*= num_words_testing

output_line = ['']
output_line*= num_words_testing


test_line_words = [element.split('\t')[0] for element in testing_line]
test_line_tags = [element.split('\t')[-1] for element in testing_line]

for i in range(num_words_testing):
    temp_li = testing_line[i].split('\t')
```

```
    if i==0:    #Accounting for the 1st word in the test document for the Viterbi
        di_transition_probs = dict2_tag_follow_tag_['.']
    else:
        di_transition_probs = dict2_tag_follow_tag_[output_line[i-1]]

    di_emission_probs = dict2_word_tag.get(test_line_words[i],'')

    #If unknown word  - tag = '.'
    if di_emission_probs=='':
        output_line[i]='.'

    else:
        max_prod_prob = 0
        counter_trans = 0
        counter_emis =0
        prod_prob = 0
        while counter_trans < len(di_transition_probs) and counter_emis < len(di_emission_prob
s):
            tag_tr = di_transition_probs[counter_trans][0]
            tag_em = di_emission_probs[counter_emis][0]
            if tag_tr < tag_em:
                counter_trans+=1
            elif tag_tr > tag_em:
                counter_emis+=1
            else:
                prod_prob = di_transition_probs[counter_trans][1] * di_emission_probs[counter_
emis][1]
                if prod_prob > max_prod_prob:
                    max_prod_prob = prod_prob
```

```
                    output_line[i] = tag_tr

                counter_trans+=1
                counter_emis+=1
```

```
    if output_line[i]=='': #In case there are no matching entries between the transition tags
and emission tags, we choose the most frequent emission tag
        output_line[i] = max(di_emission_probs,key=itemgetter(1))[0]


zip(test_line_words,output_line)
import csv
with open('output.tt', 'w', encoding='utf-8') as f:

        writer = csv.writer(f, delimiter='\t')
        writer.writerows(zip(test_line_words,output_line))


f.close()
```