In [1]:
```
import sys
import json
from collections import defaultdict, Counter
from numpy import log
import copy
```

In [2]:
```
test_data=[]
filePath="test.txt"

with open(filePath, "r") as file:
    for x in file:
        x=x.rstrip()
        test_data.append(x.split("\t"))

print(len(test_data))
```

```
135115
```

In [3]:
```
## Reading the train data
train_data=[]
filePath="dev.txt"

with open(filePath, "r") as file:
    for x in file:
        x=x.rstrip()
        train_data.append(x.split("\t"))
```

In [4]:
```
## Reading the dev data
dev_data=[]
filePath="dev.txt"

with open(filePath, "r") as file:
    for x in file:
        x=x.rstrip()
        dev_data.append(x.split("\t"))

print(len(dev_data))
# print(dev_data)
```

```
137294
```

In [5]:
```
## Fetching the words list from the dev data

dev_words=list()
temp=[]
for i in dev_data:

    if len(i)<2:
        dev_words.append(temp)
        temp=[]
    else:
        temp.append(i[1])

print(len(dev_words))
# print(dev_words)


test_words=list()
temp2=[]
for i in test_data:

    if len(i)<2:
        test_words.append(temp2)
        temp2=[]
    else:
        temp2.append(i[1])

len(test_words)
```

```
5526
5461
```

Out[5]:

In [6]:
```
## Fetching the words and tag combination list from the dev data for accuracy

dev_words2=list()
temp=[]
for i in dev_data:
    if len(i)<2:
        dev_words2.append(temp)
        temp=[]
    else:
        temp.append(i[1]+'/'+i[2])

print(len(dev_words2))
# print(dev_words2)

## Fetching all the tags from dev_data in the list

dev_tags=list()
for each in dev_data:
```

```
    if len(each)>1:
        dev_tags.append(each[2])


print(len(dev_tags))
# print(dev_tags)
```

```
5526
131768
```

In [7]:
```
## Fetching all the 45 unique tags from the train data and storing it into a set

POS_total=set()
for each in train_data:
    if len(each)>1:
        POS_total.add(each[2])


print(len(POS_total))
# print(POS_total)
```

```
45
```

In [8]:
```
## Calculating the frequency of each words in the train data

train_count={}#defaultdict(lambda: defaultdict(int))
train_count2={}#defaultdict(lambda: defaultdict(int))

for lines in train_data:
        if len(lines)>1:
            if lines[1] not in train_count:
                train_count[lines[1]]=1
            else:
                train_count[lines[1]]+=1


print(len(train_count))
# print(train_count)
```

```
15081
```

In [9]:
```
## Calulating the frequency of unknown words that are below the threshold (3)

count=0
unknown_words={}
for word, val in train_count.items():
    if val >= 3:
        train_count2[word]=val
    else:
        unknown_words[word]=0
        count+=val
unk={"unk":count}
# print(count)
```

In [10]:
```
## Creating a sorted dictionary in which the unknown words count is at first and then the sorted words

train_count3={}
train_count3["unk"]=count
train_count2=dict(sorted(train_count2.items(),key=lambda x:-x[1]))
for k, v in train_count2.items():
    train_count3[k]=v
# print(train_count3)
```

In [11]:
```
## Creating a text file named vocab which shows word,index and occurences


with open('vocab.txt', 'w') as vocab:
    i=1
    for k,v in train_count3.items():
        vocab.write('%s\t%s\t%s\n' % (k, i, v))
        i+=1
```

In [12]:
```
## Creating a transition and emmision parameters in HMM

transition_matrix=defaultdict(lambda: defaultdict(int))
emmision_matrix=defaultdict(lambda: defaultdict(int))

for w in train_data:
    if len(w)>1:
        previous="start"
        term, pos = w[1],w[2]
        if (term.isdigit()):
            emmision_matrix[pos]['<digit>'] +=1
        if term in unknown_words.keys():
            emmision_matrix[pos]['<unknown>'] +=1

        emmision_matrix[pos][term] +=1
        transition_matrix[previous][pos] +=1

        previous=pos
```

```
        pos='fin'

        transition_matrix[previous][pos] += 1


transition_matrix['fin']={}

for previous in transition_matrix:
    for present in transition_matrix:
        transition_matrix[previous][present]=transition_matrix[previous].get(present,0)+1
```

In [13]:
```
emm_prob={}
trans_prob={}

#Calculating the Emission Probability
for pos in emmision_matrix:
    emm_prob[pos] = dict()
    Count_pos_emm = sum(emmision_matrix[pos].values())
    for term in emmision_matrix[pos]:
        emm_prob[pos][term] = emmision_matrix[pos][term]/Count_pos_emm


#Calculating the Transition Probability
for pos in transition_matrix:
    trans_prob[pos] = dict()
    Count_pos_trans = sum(transition_matrix[pos].values())
    for term in transition_matrix[pos]:
        trans_prob[pos][term] = transition_matrix[pos][term]/Count_pos_trans



transition={}
for i in trans_prob:
    for j in trans_prob[i]:
        transition[str(i)+','+str(j)]=trans_prob[i][j]


emission={}
for i in emm_prob:
    for j in emm_prob[i]:
        emission[str(i)+','+str(j)]=emm_prob[i][j]



hmm={"Emmision:":emission,"Transition:":transition}
```

In [14]:
```
get_ipython().system('pip install simplejson')
import json as simplejson
```

```
Collecting simplejson
  Downloading simplejson-3.18.4-cp39-cp39-macosx_10_9_x86_64.whl (75 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 75.9/75.9 kB 616.1 kB/s eta 0:00:00a 0:00:01
Installing collected packages: simplejson
Successfully installed simplejson-3.18.4
```

In [15]:
```
json_data = json.dumps(hmm)
jsonDataFile = open("hmm.json", "w")
jsonDataFile.write(simplejson.dumps(simplejson.loads(json_data), indent=4, sort_keys=True))
jsonDataFile.close()
```

In [16]:
```
## Greedy Algorithm running on dev_data for accuracy calculation


pred_pos =[]
pred_term=[]
curr_pos=''
arr=[]

hsh={}
i=0
for term in dev_data:
    #print(len(term))
    if term==".":
        i=0
        arr.append(".")
        i=i+1
        continue
    elif len(term)>1:
        t=term[1]
        if i >=1:
            for pos in POS_total:
                prob_t=trans_prob[curr_pos][pos]
                prob_e=emm_prob[pos].get(t,0.00000001)
                total_prob=prob_t*prob_e
                pred_pos.append([total_prob,pos])
            pred_pos.sort(key=lambda x: -x[0])
            curr_pos=pred_pos[0][1]
            pred_pos=[]

        else:
            for pos in POS_total:
                prob_t=trans_prob['start'][pos]
```

```
                prob_e=emm_prob[pos].get(t,0.00000001)
                total_prob=prob_t*prob_e
                pred_pos.append([total_prob,pos])
            pred_pos.sort(key=lambda x: -x[0])
            curr_pos=pred_pos[0][1]
            pred_pos=[]
        i=i+1
        arr.append(curr_pos)
```

In [17]:
```
from sklearn.metrics import accuracy_score
print(accuracy_score(dev_tags, arr))
```

```
0.9293151599781434
```