Name : Krishna Kant Verma (2211cs19)
Name : Gourab Chatterjee (2211cs08)
M.Tech CSE

NLP Assignment - 1

*Problem Statement: Part-of-Speech (PoS) tagging assigns grammatical categories to every token in a sentence. In this assignment, you have to develop a PoS *tagger *using 2nd order Hidden Markov Model (HMM).*
Dataset name: English Penn Treebank (PTB) corpus
Number of PoS tags: 36 (Alphabetical list of part-of-speech tags used in the Penn Treebank Project) Link to download the dataset:
https://drive.google.com/file/d/1R1BLcghCh4j9Kl8_CR7MxZ4Wj57RiTxn/view?usp=share_link

```
# All important libraries needed to run this Model
```

```
import pandas as pd
import numpy as np
import copy
import math
import matplotlib.pyplot as plt
from google.colab import files
```

```
# uploading dataset(formatted dataset)
```

```
file = files.upload()
```

Choose files NLPdataset.csv

- **NLPdataset.csv**(text/csv) - 772260 bytes, last modified: 08/02/2023 - 100% done

```
Saving NLPdataset.csv to NLPdataset (2).csv
```

```
# Loading dataset into dataframe to manipulate the data
```

```
dataFrame = pd.read_csv("/content/NLPdataset.csv")
```

```
# printing top 15 data of he dataset
```

```
dataFrame.head(15)
```

	sentence	part of speech
0	Pierre Vinken 61 years old will join the board...	NNP,NNP,CD,NNS,JJ,MD,VB,DT,NN,IN,DT,JJ,NN,NNP,CD
1	Mr Vinken is chairman of Elsevier NV the Dutch...	NNP,NNP,VBZ,NN,IN,NNP,NNP,DT,NNP,VBG,NN
2	Rudolph Agnew 55 years old and former chairman...	NNP,NNP,CD,NNS,JJ,CC,JJ,NN,IN,NNP,NNP,NNP,NNP,...
3	A form of asbestos once used to make Kent ciga...	DT,NN,IN,NN,RB,VBN,TO,VB,NNP,NN,NNS,VBZ,VBN,DT...
4	The asbestos fiber crocidolite is unusually re...	DT,NN,NN,NN,VBZ,RB,JJ,IN,PRP,VBZ,DT,NNS,IN,RB,...
5	Lorillard Inc the unit of New Yorkbased Loews ...	NNP,NNP,DT,NN,IN,JJ,JJ,NNP,NNP,WDT,VBZ,NNP,NNS...
6	Although preliminary findings were reported mo...	IN,JJ,NNS,VBD,VBN,RBR,IN,DT,NN,IN,DT,JJS,NNS,V...
7	A Lorillard spokewoman said This is an old story	DT,NNP,NN,VBD,DT,VBZ,DT,JJ,NN
8	We're talking about years ago before anyone he...	PRP,VBG,IN,NNS,IN,IN,NN,VBD,IN,NN,VBG,DT,JJ,NNS
9	There is no asbestos in our products now	EX,VBZ,DT,NN,IN,PRP$,NNS,RB

```
# Spiltting dataset into training and 80% and 20% dataset
```

```
start_index = int(len(dataFrame)*0.8)
train_data_x = dataFrame.iloc[:start_index,0]
test_data_x = dataFrame.iloc[start_index:,0]
train_data_y = dataFrame.iloc[:start_index,1]
test_data_y = dataFrame.iloc[start_index:,1]
```

```
# Prnting Training Dataset
```

```
train_data_x
```

```
0       Pierre Vinken 61 years old will join the board...
1       Mr Vinken is chairman of Elsevier NV the Dutch...
2       Rudolph Agnew 55 years old and former chairman...
3       A form of asbestos once used to make Kent ciga...
4       The asbestos fiber crocidolite is unusually re...
                              ...
3126    The Treasury said the refunding is contingent ...
3127    Until such action takes places the Treasury ha...
3128    Meanwhile Treasury bonds ended modestly higher...
3129    The benchmark 30year bond about 1/4 point or 2...
3130    The benchmark was priced at 102 22/32 to yield...
Name: sentence, Length: 3131, dtype: object
```

```
# Printing Training DataSet of Y
```

```
train_data_y
```

```
0        NNP,NNP,CD,NNS,JJ,MD,VB,DT,NN,IN,DT,JJ,NN,NNP,CD
1                 NNP,NNP,VBZ,NN,IN,NNP,NNP,DT,NNP,VBG,NN
2       NNP,NNP,CD,NNS,JJ,CC,JJ,NN,IN,NNP,NNP,NNP,NNP,...
3       DT,NN,IN,NN,RB,VBN,TO,VB,NNP,NN,NNS,VBZ,VBN,DT...
4       DT,NN,NN,NN,VBZ,RB,JJ,IN,PRP,VBZ,DT,NNS,IN,RB,...
                              ...
3126    DT,NNP,VBD,DT,NN,VBZ,NN,IN,JJ,CC,JJ,NN,IN,DT,N...
3127    IN,JJ,NN,VBZ,NNS,DT,NNP,VBZ,DT,NN,TO,VB,JJ,NN,...
3128                       RB,NNP,NNS,VBD,RB,JJR,IN,JJ,NN
3129            DT,NN,JJ,NN,IN,CD,NN,CC,CD,IN,DT,CD,NN,NN
3130    DT,NN,VBD,VBN,IN,CD,CD,TO,VB,CD,VBN,IN,CD,CD,T...
Name: part of speech, Length: 3131, dtype: object
```

`#Extracting Dataset into Distinct Parts of Speech Tags (Pos Tags)`

```
distinctTags = set()
i = 0
for i in train_data_y:
    for j in (i.split(",")):
        distinctTags.add(j)

posTags = {}
j = 0
for i in distinctTags:
    posTags[i] = j
    j = j + 1
```

```
# Printing Length of Disinct Pos Tags
```

```
len(posTags)
```

```
35
```

`#Creating Transition Matrix`

```
cntDistinctPosTags = len(posTags)
#declaring transition matrix
transitionCount = []
for i in range(0,cntDistinctPosTags):
```

```
    t = []
    for j in range(0,cntDistinctPosTags):
        t.append(float(0))
    transitionCount.append(t)

cntEachPosTags = [0]*cntDistinctPosTags
posTagsInitCnt = [0]*cntDistinctPosTags
count  = 0
for i in train_data_y:
    line = list(i.split(","))
    posTagsInitCnt[posTags[line[0]]]+=1
    cntEachPosTags[posTags[line[0]]]+=1
    length  = int(len(line))
    for j in range(1,length):
        cntEachPosTags[posTags[line[j]]]+=1
        transitionCount[posTags[line[j-1]]][posTags[line[j]]] += 1
```

```
# Finding initial state probability
```

```
initStateProb = [0]*cntDistinctPosTags
for i in range(cntDistinctPosTags):
    initStateProb[i] = (posTagsInitCnt[i] + 1)/cntEachPosTags[i]
```

```
# Finding Transition Probabilities
```

```
transitionProbability = copy.deepcopy(transitionCount)

for i in range(0,cntDistinctPosTags ):
    for j in range(0,cntDistinctPosTags ):
        transitionProbability[i][j] = ((transitionProbability[i][j] + 1)*(100))/(cntEachPosTags[j] + 1)

print(transitionProbability)
```

```
[[0.825643516270034, 4.62962962962963, 2.7636659215606585, 14.634146341463415, 0.546448087431694, 6.451612903225806, 1.7
```

#Extraction of distinct words from the dataset

```
distinctWords = set()
i = 0
for i in train_data_x:
    for j in (i.split(" ")):
        distinctWords.add(j)

words = {}
j = 0
for i in distinctWords:
    words[i] = j
    j = j + 1
print(words)
print(len(words))
```

```
{'': 0, 'quotas': 1, 'mouthup': 2, 'required': 3, '1457': 4, 'Einhorn': 5, '2735': 6, 'initialing': 7, 'Ray': 8, 'oils':
10844
```

#Initializing Emission Matrix

```
cntDistinctWords = len(words)

#declaring emission count matrix
emissionCount = []
for i in range(0,cntDistinctWords):
    t = []
    for j in range(0,cntDistinctPosTags):
        t.append(float(0))
    emissionCount.append(t)



for i in range(0,len(train_data_x)):
    line_x = train_data_x[i].split(" ")
    line_y = train_data_y[i].split(",")
```

```
    length = len(line_x)

    for j in range(0,length):
        emissionCount[words[line_x[j]]][posTags[line_y[j]]] += 1
```

#finding Emission Probabilities

```
emissionProbability = copy.deepcopy(emissionCount)

for i in range(0,cntDistinctPosTags):
    for j in range(0,cntDistinctPosTags ):
        emissionProbability[i][j] = ((emissionProbability[i][j] + 1)*(100))/(cntEachPosTags[j])

print(emissionProbability[0])
```

```
[10.787172011661808, 0.46511627906976744, 0.02032520325203252, 1.2345679012345678, 0.5494505494505495, 3.333333333333333
```

#Extracting Most Freq Words

```
for i in range(0,cntDistinctPosTags):
    if(cntEachPosTags[i]==max(cntEachPosTags)):
        mostFreqWords = i
        break
print(mostFreqWords)

for i in posTags:
    if(posTags[i]==mostFreqWords):
        mostFreqTags = i
print(mostFreqTags)
```

```
10
NN
```

Implementation of Viterbi Algorithm

```
def viterbiAlgorithm(transitionProbability,emissionProbability,initStateProb,cntEachPosTags,posTags,words,mostFreqTags,tes
    cntDistinctPosTags = len(posTags)


    #declaring matrix to calculate probability at each k
    probabiltyAndSequence = []
    currentSentence = testX.split(" ")
    currentLength = len(currentSentence)
    predictedSeqPosTags = []

    for i in range(0,cntDistinctPosTags):
        t = []
        for j in range(0,currentLength):
            t.append([0,"a","a"])
        probabiltyAndSequence.append(t)
    #end of matrix creation

    for j in posTags.keys():
        if currentSentence[0] in words.keys():
            probabiltyAndSequence[posTags[j]][0][0] = initStateProb[posTags[j]] * emissionProbability[words[currentSentenc
            probabiltyAndSequence[posTags[j]][0][1] = "start_index"
            probabiltyAndSequence[posTags[j]][0][2] = j
        else:
            probabiltyAndSequence[posTags[j]][0][0] = initStateProb[posTags[j]]
            probabiltyAndSequence[posTags[j]][0][1] = "start_index"
            probabiltyAndSequence[posTags[j]][0][2] = j


    for i in range(1,currentLength):
        if currentSentence[i] in words.keys():
            for j in range(0,cntDistinctPosTags): #every prev state
                previousProb = probabiltyAndSequence[j][i-1][0]
                prevPrevTag = probabiltyAndSequence[j][i-1][1]
                prevTag = probabiltyAndSequence[j][i-1][2]

                #print(currentSentence[i],i,prevTag)
                for k in posTags.keys():#every cur state
                    maxProb = probabiltyAndSequence[posTags[k]][i][0]
                    max_postag = probabiltyAndSequence[posTags[k]][i][1]

                    if(prevPrevTag=="start_index"):
                        p1 = initStateProb[posTags[prevTag]]
```

```
                    else:
                        p1 = transitionProbability[posTags[prevPrevTag]][posTags[prevTag]]

                    p2 = transitionProbability[posTags[prevTag]][posTags[k]]
                    p3 = emissionProbability[words[currentSentence[i]]][posTags[k]]

                    currProb = previousProb + math.log(p1 + 1) + math.log(p2 + 1) + math.log(p3 + 1)

                    if maxProb<=currProb:
                            maxProb = currProb
                            probabiltyAndSequence[posTags[k]][i][0] = maxProb
                            probabiltyAndSequence[posTags[k]][i][1] = prevTag
                            probabiltyAndSequence[posTags[k]][i][2] = k

        else:
            for j in range(0,cntDistinctPosTags): #every prev state
                previousProb = probabiltyAndSequence[j][i-1][0]
                prevPrevTag = probabiltyAndSequence[j][i-1][1]
                prevTag = probabiltyAndSequence[j][i-1][2]

                for k in posTags.keys():#every cur state
                    if (k==mostFreqTags):
                        maxProb = probabiltyAndSequence[posTags[k]][i][0]
                        max_postag = probabiltyAndSequence[posTags[k]][i][1]

                        if(prevPrevTag=="start_index"):
                            p1 = initStateProb[posTags[prevTag]]
                        else:
                            p1 = transitionProbability[posTags[prevPrevTag]][posTags[prevTag]]

                        p2 = transitionProbability[posTags[prevTag]][posTags[k]]

                        currProb = previousProb + math.log(p1 + 1) + math.log(p2 + 1)

                        if maxProb<=currProb:
                                maxProb = currProb
                                probabiltyAndSequence[posTags[k]][i][0] = maxProb
                                probabiltyAndSequence[posTags[k]][i][1] = prevTag
                                probabiltyAndSequence[posTags[k]][i][2] = k
                    else:
                        probabiltyAndSequence[posTags[k]][i][0] = float(0)
                        probabiltyAndSequence[posTags[k]][i][1] = prevTag
                        probabiltyAndSequence[posTags[k]][i][2] = k


            #print(probabiltyAndSequence[posTags[k]][i][1])

    maxiprob = probabiltyAndSequence[0][currentLength-1][0]
    prevpostag = probabiltyAndSequence[0][currentLength-1][1]
    maxipostag = probabiltyAndSequence[0][currentLength-1][2]

    for i in range(0,cntDistinctPosTags):
        if probabiltyAndSequence[i][currentLength-1][0]>=maxiprob:
            maxiprob = probabiltyAndSequence[i][currentLength-1][0]
            prevpostag = probabiltyAndSequence[i][currentLength-1][1]
            maxipostag = probabiltyAndSequence[i][currentLength-1][2]


    predictedSeqPosTags.append(maxipostag)

    for i in range(currentLength-2,-1,-1):
        predictedSeqPosTags.append(prevpostag)
        prevpostag = probabiltyAndSequence[posTags[prevpostag]][i][1]


    predictedSeqPosTags.reverse()

    #print(predictedSeqPosTags)
    return predictedSeqPosTags
```

`#Executing Viterbi algorithm to predict the POS tags for each word of a sentence`

```
predictedY = []
for i in test_data_x:
    predictedY.append(viterbiAlgorithm(transitionProbability,emissionProbability,initStateProb,cntEachPosTags,posTags,word
```

`Printing Results`

```
j = 0
for i in test_data_y:
    print(i)
    print(predictedY[j])
    j +=1
```

```
['IN', 'NNS', 'IN', 'NNP', 'IN', 'NN', 'DT', 'NN', 'NNS', 'VBP', 'SYM', 'SYM', 'NN', 'DT', 'NN', 'IN', 'SYM']
JJ,NNS,RB,MD,VB,IN,NNS,VBD,IN,JJS,CD,NNS,DT,NN,VBD,NNP,NNP,NN,IN,NNP,NNP,CC,NNP,NNP,NNP,IN,NNP,NNP
['JJ', 'NNS', 'IN', 'DT', 'NN', 'IN', 'NNS', 'VBD', 'IN', 'JJS', 'SYM', 'SYM', 'DT', 'NN', 'VBD', 'NN', 'IN', 'NN', 'I
PRP,VBZ,JJ,RB,WP,MD,VB,DT,VBG,NN
['PRP', 'VBZ', 'UH', 'SYM', 'SYM', 'MD', 'VB', 'DT', 'RBS', 'SYM']
IN,NNP,NN,NNS,VBD,IN,RB,DT,IN,DT,NN,IN,DT,NN,DT,NNP,NNP,VBZ,IN,RB,NN,IN,DT,NN,MD,RB,VB,IN,NN,IN,DT,CD,NN,NN,VBZ
['NNP', 'NNP', 'NNP', 'NN', 'NN', 'IN', 'DT', 'NN', 'IN', 'DT', 'NN', 'IN', 'DT', 'NN', 'DT', 'NNP', 'NNP', 'NNP', 'WD
DT,NNS,VBP,VBN,IN,NNS,IN,DT,NN,NN,NNS,MD,VB,NNS,TO,VB,VBG,NN,NNS,NNS,CC,RB,TO,VB,JJ,NN
['DT', 'NNS', 'VBP', 'VBN', 'IN', 'NNS', 'IN', 'DT', 'NN', 'IN', 'NNS', 'MD', 'VB', 'NNS', 'TO', 'VB', 'SYM', 'SYM', '
DT,NN,VBZ,VBG,DT,NN,WRB,IN,NN,EX,VBZ,NN,IN,NN,TO,VB,VBN,VBD,NNP,NNP,NNP,NNP,NN,NN
['DT', 'NN', 'VBZ', 'UH', 'DT', 'NN', 'WRB', 'IN', 'DT', 'NN', 'VBZ', 'VBN', 'IN', 'NN', 'TO', 'VB', 'SYM', 'SYM', 'SY
IN,JJ,IN,DT,NN,NN,NN,VBZ,TO,NNP,NNP,IN,NN,PRP,VBZ,JJ,IN,NNS,TO,VB,DT,NNP,NNP,IN,VBG,DT,JJ,NN,IN,NN,IN,NN,TO,DT,NN
['DT', 'JJ', 'IN', 'DT', 'NN', 'NN', 'NN', 'VBZ', 'TO', 'NNP', 'NNP', 'IN', 'NN', 'PRP', 'VBZ', 'VBN', 'IN', 'NN', 'TO
NNS,IN,DT,NNP,NNPS,CC,NNP,NNP,MD,RB,VB,NN,IN,NNP,NNP
['NN', 'IN', 'DT', 'JJ', 'NN', 'CC', 'SYM', 'NN', 'MD', 'RB', 'NN', 'NN', 'IN', 'NNP', 'NNP']
CD,NN,IN,NN,VBZ,RB,VBG,PRP$,NN,VBG,NN,IN,NNP,TO,NNP
['DT', 'NN', 'IN', 'NN', 'VBZ', 'UH', 'SYM', 'SYM', 'SYM', 'NN', 'NN', 'IN', 'NNP', 'TO', 'SYM']
CC,PRP,VBZ,JJ,IN,DT,NNP,NNP,MD,VB,IN,PRP$,NN,VBG,NN
['CC', 'PRP', 'VBZ', 'VBN', 'IN', 'DT', 'JJ', 'NNS', 'MD', 'VB', 'IN', 'PRP$', 'SYM', 'SYM', 'SYM']
DT,JJ,NNS,VBD,VBN,IN,DT,NN,IN,NNP,IN,IN,NNS
['DT', 'JJ', 'NNS', 'VBD', 'NN', 'IN', 'DT', 'NN', 'IN', 'NNP', 'NNP', 'IN', 'SYM']
DT,NNP,NNP,RB,VBZ,VBG,NNP,NNS,JJR,IN,DT,NN
['DT', 'JJ', 'NNP', 'NNP', 'NNP', 'NNP', 'NNP', 'SYM', 'SYM', 'IN', 'DT', 'NN']
CC,PRP$,NNS,RB,VBD,VBN,IN,DT,NN,IN,PRP$,JJ,NN,RB,RB,IN,NN,NNS
['CC', 'PRP$', 'SYM', 'SYM', 'SYM', 'NN', 'IN', 'DT', 'NN', 'IN', 'PRP$', 'SYM', 'SYM', 'SYM', 'SYM', 'SYM', 'SYM', 'S
IN,JJ,NN,NNS,NN
['IN', 'JJ', 'NN', 'NNS', 'NN']
NN,JJ,NN,NNS,NNS,VBD,IN,JJ,NN,CC,JJ,IN,DT,NN,VBD,IN,NN,NN
['IN', 'DT', 'NN', 'NNS', 'VBP', 'VBN', 'IN', 'DT', 'NN', 'CC', 'RB', 'IN', 'DT', 'NN', 'VBD', 'IN', 'DT', 'NN']
NNS,VBD,IN,DT,NN,IN,DT,JJ,JJ,JJ,NN,VBD,VBN,IN,DT,NN,VBG,DT,RB,JJ,JJ,NN
['NNP', 'VBD', 'IN', 'DT', 'NN', 'IN', 'DT', 'NN', 'IN', 'DT', 'NN', 'VBD', 'VBN', 'IN', 'DT', 'NN', 'NN', 'DT', 'NN',
NN,NN,IN,NNP,NN,VBD,IN,CD,NNS,DT,NN,RB,CD,NN,IN,DT,NNP,NNP,NNP,NNP
['DT', 'NN', 'IN', 'DT', 'NN', 'VBD', 'IN', 'NN', 'IN', 'DT', 'NN', 'IN', 'CD', 'SYM', 'IN', 'DT', 'NNP', 'NNP', 'NNP'
NNP,NNP,NNP,IN,NNP,NN,VBD,CD,NNS,TO,CD,DT,NN
['NNP', 'NNP', 'NN', 'IN', 'DT', 'NN', 'IN', 'CD', 'NNS', 'TO', 'VB', 'DT', 'NN']
NN,NNS,VBD,DT,NN,WDT,VBD,NNP
['JJ', 'NNS', 'VBD', 'DT', 'NN', 'IN', 'DT', 'NNP']
NNP,NNPS,NNP,NNS,VBD,RB,VBN,NN,CC,NN,VBD,IN,DT,NNS,NNS
['DT', 'NN', 'IN', 'DT', 'NN', 'IN', 'DT', 'NN', 'CC', 'SYM', 'SYM', 'IN', 'DT', 'JJ', 'NNS']
NNP,NN,NN,VBD,CD,DT,NN,TO,CD
['DT', 'NN', 'NN', 'VBD', 'UH', 'DT', 'NN', 'TO', 'NN']
NNP,NN,VBD,CD,NNS,DT,NN,TO,CD
['DT', 'NN', 'VBD', 'NN', 'IN', 'DT', 'NN', 'TO', 'NN']
NNP,NN,VBD,IN,CD,DT,NN,IN,CD
['DT', 'NN', 'VBD', 'RP', 'SYM', 'DT', 'NN', 'IN', 'NN']
JJ,NNS,NN,IN,JJ,RB,VBP,VBG,VBN,JJR,IN,NN,NN,NNS,IN,DT,NN,IN,NNS,VBP,JJR,NN,NN,VBG,TO,NNP,NNP,NN,NN,IN,NN,IN,NNPS,NNP,I
['IN', 'DT', 'NN', 'IN', 'JJ', 'NNS', 'VBP', 'VBG', 'NN', 'RBR', 'SYM', 'SYM', 'SYM', 'SYM', 'IN', 'DT', 'NN', 'IN', '
DT,JJ,NN,IN,JJ,NNS,VBD,DT,NN,IN,NN,CC,NN,IN,NNS,PRP,VBD
['DT', 'JJ', 'NN', 'IN', 'DT', 'NN', 'VBD', 'DT', 'NN', 'IN', 'NN', 'CC', 'NN', 'IN', 'NN', 'PRP', 'VBD']
NN,DT,NN,NN,VBD,RB,VBG,DT,JJR,JJ,NN,IN,JJ,NNS,NNS,PRP,VBD
['IN', 'DT', 'SYM', 'SYM', 'SYM', 'SYM', 'SYM', 'DT', 'JJR', 'SYM', 'SYM', 'SYM', 'SYM', 'SYM', 'SYM', 'PRP', 'VBD']
NN,CC,NN,WDT,VBP,JJR,IN,DT,JJ,NN,IN,NN,VBD,RB,JJR,PRP,VBD
['NN', 'CC', 'NN', 'WDT', 'VBP', 'SYM', 'IN', 'DT', 'JJ', 'NN', 'IN', 'NN', 'VBD', 'RB', 'NN', 'PRP', 'VBD']
NN,VBZ,RB,IN,NN,IN,RB,JJ,NNS,IN,NNS,IN,DT,NNP,NNP,PRP,VBD
['NN', 'VBZ', 'RB', 'IN', 'NN', 'IN', 'DT', 'JJ', 'NNS', 'IN', 'NN', 'IN', 'DT', 'NNP', 'NNP', 'NNP', 'VBD']
NN,DT,NNS,VBD,IN,CD,NNS,TO,DT,NN,IN,CD,NNS,VBG,TO,DT,NN,NN
['IN', 'DT', 'NNS', 'VBD', 'IN', 'NN', 'NNS', 'TO', 'DT', 'NN', 'IN', 'NN', 'IN', 'VBG', 'TO', 'DT', 'NN', 'NN']
NNP,NNS,NNS,RB,VBD,NNP,NNS,IN,JJ,NNS,VBD,TO,VB
```

#Finding Accuracy

```
k = 0
accuracy = []
for i in test_data_y:
    count = 0
    actualtags_y = i.split(",")
    currentLength = len(actualtags_y)
    for j in range(0,currentLength):
        if predictedY[k][j]==actualtags_y[j]:
            count +=1
    k +=1
    accuracy.append(count/currentLength)
print("Individual Accuracy: ")
print(accuracy)

print("OverAll Accuracy: ")
print(sum(accuracy)/len(accuracy))
```

```
Individual Accuracy:
[0.5789473684210527, 0.6666666666666666, 0.0, 0.75, 0.38461538461538464, 0.8064516129032258, 0.625, 0.75, 0.52, 0.666666
```

```
OverAll Accuracy:
0.6472258333275303
```

Repeating Same thing after mapping of 36 tags to 4 tags

#Transition Matrix for 4 tags

```
map36to4 = {'NNPS': "N",'VBG': "V",'VBD': "V",'PDT': "O", 'PRP': "O", 'PRP$': "O", 'WP': "O", 'JJR': "A", 'JJ': "A", 'SYM'
```

```
cntDistinctPosTags_4 = 4
newPosTags = {"N": 0,"V": 1,"A": 2,"O":3}
transitionCount_4 = []
for i in range(0,cntDistinctPosTags_4):
    t = []
    for j in range(0,cntDistinctPosTags_4):
        t.append(float(0))
    transitionCount_4.append(t)

cntEachPosTags_4 = [0]*cntDistinctPosTags_4
posTagsInitCnt_4 = [0]*cntDistinctPosTags_4
count  = 0
for i in train_data_y:
    line = list(i.split(","))
    posTagsInitCnt_4[newPosTags[map36to4[line[0]]]]+=1
    cntEachPosTags_4[newPosTags[map36to4[line[0]]]]+=1
    length  = int(len(line))
    for j in range(1,length):
        cntEachPosTags_4[newPosTags[map36to4[line[j]]]]+=1
        transitionCount_4[newPosTags[map36to4[line[j-1]]]][newPosTags[map36to4[line[j]]]] += 1
```

Counting Distinct Pos Tags for 4 tags

```
cntDistinctPosTags_4 = 4
initStateProb_4 = [0]*cntDistinctPosTags_4
for i in range(cntDistinctPosTags_4):
    initStateProb_4[i] = (posTagsInitCnt_4[i] + 1)/cntEachPosTags_4[i]
```

Printing Transition Count 4

```
print(transitionCount_4)
```

```
[[7287.0, 4056.0, 1200.0, 7970.0], [1451.0, 2124.0, 1516.0, 5308.0], [3836.0, 856.0, 981.0, 1674.0], [9220.0, 3647.0, 35
```

Printing Count of Each Pos Tags

```
cntEachPosTags_4
```

```
[22647, 10776, 7609, 23929]
```

```
posTagsInitCnt_4
```

```
[853, 93, 318, 1867]
```

#Calculating Transition probabilities for 4 POS tags

```
transitionProb_4 = copy.deepcopy(transitionCount_4)
cntDistinctPosTags_4 = 4
for i in range(0,cntDistinctPosTags_4 ):
    for j in range(0,cntDistinctPosTags_4 ):
        transitionProb_4[i][j] = (transitionProb_4[i][j] + 1)*100/(cntEachPosTags_4[j] + 1)

print(transitionProb_4)
```

```
[[32.17944189332391, 37.64498468961678, 15.78186596583443, 33.30965315503552], [6.4111621335217235, 19.7179177878816, 19
```

#Calculating Emission Count Matrix For 4 POS tags

```
cntDistinctWords = len(words)
cntDistinctPosTags_4 = 4
#declaring emission count matrix with 4 POS tags
emissionCount_4 = []
for i in range(0,cntDistinctWords):
    t = []
    for j in range(0,cntDistinctPosTags_4):
        t.append(float(0))
    emissionCount_4.append(t)



for i in range(0,len(train_data_x)):
    linex = train_data_x[i].split(" ")
    liney = train_data_y[i].split(",")
    length = len(linex)

    for j in range(0,length):
        emissionCount_4[words[linex[j]]][newPosTags[map36to4[liney[j-1]]]] += 1
```

`#Calculating Emission Probabilities For 4 POS tags`

```
emissionProb_4 = copy.deepcopy(emissionCount_4)
cntDistinctPosTags_4 = 4

for i in range(0,cntDistinctPosTags_4):
    for j in range(0,cntDistinctPosTags_4):
        emissionProb_4[i][j] = ((emissionProb_4[i][j] + 1)*(100))/(cntEachPosTags_4[j])

print(emissionProb_4[0])
```

```
[0.6888329580076832, 0.16703786191536749, 0.3811276120383756, 0.09193865184504157]
```

`#Getting Most Freq Wordss`

```
cntDistinctPosTags_4 = 4
for i in range(0,cntDistinctPosTags_4):
    if(cntEachPosTags_4[i]==max(cntEachPosTags_4)):
        mostFreqTagIndex_4 = i
        break
print(mostFreqTagIndex_4)

for i in newPosTags:
    if(newPosTags[i]==mostFreqTagIndex_4):
        mostFreqTag_4 = i
print(mostFreqTag_4)
```

```
3
O
```

`Testing Pos tagging with 4 pos Tag (viterbi algo)`

```
predictedY_4 = []
for i in test_data_x:
    predictedY_4.append(viterbiAlgorithm(transitionProb_4,emissionProb_4,initStateProb_4,cntEachPosTags_4,newPosTags,words
```

`Calculating Accuracy`

```
k = 0
accuracy_4 = []
print("Individual Accuracy: ")
for i in test_data_y:
    count = 0
    actualTags_4 = i.split(",")
    curlen = len(actualTags_4)
    for j in range(0,curlen):
        if predictedY_4[k][j]==map36to4[actualTags_4[j]]:
            count +=1
    k +=1
    accuracy_4.append(count*100/curlen)
print(accuracy_4)
print("Overall Accuracy: ")
print(sum(accuracy_4)/len(accuracy_4))
```

```
Individual Accuracy:
[21.05263157894737, 37.03703703703704, 50.0, 41.666666666666664, 30.76923076923077, 32.25806451612903, 31.25, 12.5, 24.0
```

```
Overall Accuracy:
28.40087994815522
```

ClassWise Accuracy

```
correctpredict = {"N": 0,"V": 0,"A": 0,"O":0}
totalprediction = {"N": 0,"V": 0,"A": 0,"O":0}
k = 0
accuracy_4 = []
for i in test_data_y:
    actualTags_4 = i.split(",")
    curlen = len(actualTags_4)
    for j in range(0,curlen):

        s = predictedY_4[k][j]
        t = map36to4[actualTags_4[j]]
        if s==t:
            correctpredict[t] +=1
        totalprediction[t]+=1
    k +=1
    accuracy_4.append(count/curlen)

print("Average accuracy = ",sum(accuracy_4)*100/len(accuracy_4))

print("\nClass wise prediction")
for i in correctpredict:
    print(i," = ",(correctpredict[i]*100)/totalprediction[i])
```

```
Average accuracy =  20.17781170796642

Class wise prediction
N  =  31.792207792207794
V  =  7.3161485974222895
A  =  1.0582010582010581
O  =  42.7536231884058
```

Comparison:

In part-of-speech (POS) tagging, the number of tags used can affect the performance of the model. A model with 36 tags is likely to have more detailed information about the different parts of speech in the text, but this increased complexity can also lead to overfitting and decreased performance on unseen data. On the other hand, a model with 4 tags may simplify the task and lead to more robust and generalizable results, especially if the data is limited.

The transition and emission probabilities are an important factor in the performance of a POS tagging model. The transition probability determines the likelihood of one tag following another, while the emission probability determines the likelihood of a word being associated with a particular tag. With fewer tags, the model is likely to have more accurate transition and emission probabilities, which can lead to better performance.

In conclusion, the choice between using a 36-tag or a 4-tag model depends on the specifics of the task, the data, and the model architecture. If the 4-tag model performs better, it may be due to its simplicity and more robust probabilistic assumptions.

In This Case 36 tags Give : 64% accuracy and 4 tags Give 28% 36 tags performs better than 4 tags

On Given Data Set Bigram HMM gives better accurancy in 36 tags than 4 tags

0s completed at 00:23