St. Francis Institute of Technology

Department of Computer Engineering

Academic Year: 2021-2022 **Semester: VIII**

Subject: Natural Language Processing **Class/Branch/: BE/CMPNA**

Name :- Nithin Menezes **Roll Number: 56**

Aim : To implement the Part of Speech Tagging with Hidden Markov Models

```
lines = ['<s>  Marry Janne can see will <s> spo </s>']
outF = open("hmm.txt", "w")
for i in lines:
  outF.write(i+"\n")
outF.close()
```

```
ct = {}
bigram = {}
trigram = {}
cth={}
x=[]
with open("hmm.txt",encoding='utf-8') as f:
    for line in f:
        x.clear()
        l = line.strip()
        l.replace("\n","")
        xm=l.split(' ')
        for i in xm:
            if (i in cth):
                cth[i] += 1
            else:
                cth[i] = 1
            sp = i.split('/')
            x.append(sp[0])

        i = 0
        j = 1

        for m in x:
            if (m in ct):
                ct[m] += 1
            else:
                ct[m] = 1
```

```
        while(j<len(x)):

            bi = x[i]+' | ' +x[j]
            if (bi in bigram):
                bigram[bi] += 1
            else:
                bigram[bi] = 1
            i+=1
            j+=1


print("Transmission probabilities are:")
print("Bigram \t\tProbability(x(n)|x(n-1))")
for key, value in bigram.items():
    x = key.split(" | ")
    xm=x[0]
    den = ct[xm]
    prob = value/den
    print(key+"\t"+str(prob))

print("Emmision Probabilities are:")
print("Value \tProbability(x(n)|(n-1))")
for key, value in cth.items():
    x = key.split("/")
    xm=x[0]
    den = ct[xm]
    prob = value/den
    print(key+"\t"+str(prob))
```

```
Transmission probabilities are:
Bigram          Probability(x(n)|x(n-1))
<s> |   0.5
 | Marry        1.0
Marry | Janne   1.0
Janne | can     1.0
can | see       1.0
see | will      1.0
will | <s>      1.0
<s> | spo       0.5
spo | < 1.0
Emmision Probabilities are:
Value   Probability(x(n)|(n-1))
<s>     1.0
        1.0
Marry   1.0
Janne   1.0
can     1.0
see     1.0
will    1.0
spo     1.0
</s>    1.0
```

```
lines = ['<s> BOS' ,'the DT', 'students NN', 'pass VB', 'the DT', 'test NN', '<s> BOS', 'the
outF = open("corpus_eng.txt", "w")
for i in lines:
  outF.write(i+"\n")
outF.close()
line = 'the students studied for the test'
outF = open("corpus_test.txt", "w")
outF.write(line+"\n")
outF.close()

import os
import sys
import time
tags = ['NN', 'VB', 'DT', 'P','BOS']

def max_connect(x, y, viterbi_matrix, emission, transmission_matrix):
    max = -99999
    path = -1

    for k in range(len(tags)):
        val = viterbi_matrix[k][x-1] * transmission_matrix[k][y]
        if val * emission > max:
            max = val
            path = k
    return max, path

def main():

    start_time = time.time()

    # Path of training files
    filepath = 'corpus_eng.txt'
    exclude = ['END','</s>']
    wordtypes = []
    tagscount = []
    f = open(filepath, mode='r', encoding='utf-8')
    file_contents = f.readlines()


    for x in range(len(tags)):
        tagscount.append(0)
    for x in range( len(file_contents)):
        line = file_contents.pop(0).strip().split(' ')
        for i, word in enumerate(line):
            if i == 0:
                if word not in wordtypes and word not in exclude:
                    wordtypes.append(word)
            else:
                if word in tags and word not in exclude:
                    tagscount[tags.index(word)] += 1
    f.close()
```

```
emission_matrix = []
transmission_matrix = []
for x in range(len(tags)):
    emission_matrix.append([])
    for y in range(len(wordtypes)):
        emission_matrix[x].append(0)


for x in range(len(tags)):
    transmission_matrix.append([])
    for y in range(len(tags)):
        transmission_matrix[x].append(0)


f = open(filepath, mode='r', encoding='utf-8')
file_contents = f.readlines()

row_id = -1
for x in range(len(file_contents)):
    line = file_contents.pop(0).strip().split(' ')

    if line[0] not in exclude:
        col_id = wordtypes.index(line[0])
        prev_row_id = row_id
        row_id = tags.index(line[1])
        emission_matrix[row_id][col_id] += 1
        if prev_row_id != -1:
            transmission_matrix[prev_row_id][row_id] += 1
    else:
        row_id = -1
for x in range(len(tags)):
    for y in range(len(wordtypes)):
        if tagscount[x] != 0:
            emission_matrix[x][y] = float(emission_matrix[x][y]) / tagscount[x]

for x in range(len(tags)):
    for y in range(len(tags)):
        if tagscount[x] != 0:
            transmission_matrix[x][y] = float(transmission_matrix[x][y]) / tagscount[x]

wstring = ''
for i in wordtypes:
    wstring +='\t'+i
#print(wstring)
print(wstring)
for i in range(len(emission_matrix)):
    tstring=''
    for j in emission_matrix[i]:
        tstring +='\t'+str(round(j,4))
    print(tags[i]+tstring)
```

```
#print(emission_matrix)
print('transmission_matrix')
wstring=''
for i in tags:
    wstring +='\t'+i
    #print(wstring)
print(wstring)
for i in range(len(transmission_matrix)):
    tstring=''
    for j in transmission_matrix[i]:
        tstring +='\t'+str(round(j,4))
    print(tags[i]+tstring)


start_time = time.time()

# Open the testing file to read test sentences
testpath = 'corpus_test.txt'
file_test = open(testpath, mode='r', encoding='utf-8')
test_input = file_test.readlines()

# Declare variables for test words and pos tags
test_words = []
pos_tags = []

# Create an output file to write the output tags for each sentences


# For each line POS tags are computed
for j in range(len(test_input)):

    test_words = []
    pos_tags = []

    line = test_input.pop(0).strip().split(' ')

    for word in line:
        test_words.append(word)
        pos_tags.append(-1)

    viterbi_matrix = []
    viterbi_path = []

    # Initialize viterbi matrix of size |tags| * |no of words in test sentence|
    for x in range(len(tags)):
        viterbi_matrix.append([])
        viterbi_path.append([])
        for y in range(len(test_words)):
            viterbi_matrix[x].append(0)
```

```
                viterbi_matrix[x].append(0)
                viterbi_path[x].append(0)

            # Update viterbi matrix column wise
            for x in range(len(test_words)):
                for y in range(len(tags)):
                    if test_words[x] in wordtypes:
                        word_index = wordtypes.index(test_words[x])
                        tag_index = tags.index(tags[y])
                        emission = emission_matrix[tag_index][word_index]
                    else:
                        emission = 0.001

                    if x > 0:
                        max, viterbi_path[y][x] = max_connect(x, y, viterbi_matrix, emission, tra
                    else:
                        max = 1
                    viterbi_matrix[y][x] = emission * max

            maxval = -999999
            maxs = -1
            for x in range(len(tags)):
                if viterbi_matrix[x][len(test_words)-1] > maxval:
                    maxval = viterbi_matrix[x][len(test_words)-1]
                    maxs = x
            for x in range(len(test_words)-1, -1, -1):
                pos_tags[x] = maxs
                maxs = viterbi_path[maxs][x]
            print("\nOutput Tags: ")
            file_output = open("opcorpus.txt", mode =  'a', encoding = 'utf-8')
            for i, x in enumerate(pos_tags):
                file_output.write(test_words[i] + "_" + tags[x] + " ")
                print(test_words[i] + "_" + tags[x] + " ")
            file_output.write("\n")

        f.close()
        wstring=''
        file_output.close()
        print("\nEmission Matrix")
        for i in test_words:
                wstring +='\t'+i
            #print(wstring)
        print(wstring)
        for i in range(len(viterbi_matrix)):
                tstring=''
                for j in viterbi_matrix[i]:
                    tstring +='\t'+str(round(j,4))
                print(tags[i]+tstring)

        #print(viterbi_matrix)
        file_test.close()
```

```
if __name__ == "__main__":
    main()
```

```
        <s>     the     students        pass    test    studied for     teachers
NN      0.0     0.0     0.5     0.1667  0.1667  0.0     0.0     0.1667
VB      0.0     0.0     0.0     0.3333  0.3333  0.3333  0.0     0.0
DT      0.0     1.0     0.0     0.0     0.0     0.0     0.0     0.0
P       0.0     0.0     0.0     0.0     0.0     0.0     1.0     0.0
BOS     1.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
transmission_matrix
        NN      VB      DT      P       BOS
NN      0.0     0.5     0.0     0.0     0.3333
VB      0.3333  0.0     0.3333  0.3333  0.0
DT      1.0     0.0     0.0     0.0     0.0
P       0.0     0.0     1.0     0.0     0.0
BOS     0.3333  0.0     0.6667  0.0     0.0

Output Tags:
the_DT
students_NN
studied_VB
for_P
the_DT
test_NN

Emission Matrix
        the     students        studied for     the     test
NN      0.0     0.5     0.0     0.0     0.0     0.0046
VB      0.0     0.0     0.0833  0.0     0.0     0.0
DT      1.0     0.0     0.0     0.0     0.0278  0.0
P       0.0     0.0     0.0     0.0278  0.0     0.0
BOS     0.0     0.0     0.0     0.0     0.0     0.0
```