Experiment 5

NAME	Shreya Shetty
UID	2019140059
CLASS	TE IT
BATCH	B
SUBJECT	NLP Lab

AIM: To calculate emission and transition matrix for tagging Parts of Speech using Hidden Markov Model.

THEORY:

POS tagging or part-of-speech tagging is the procedure of assigning a grammatical category like noun, verb, adjective etc. to a word. In this process both the lexical information and the context play an important role as the same lexical form can behave differently in a different context.

For example the word "Park" can have two different lexical categories based on the context.

- The boy is playing in the park. ('Park' is Noun)
- Park the car. ('Park' is Verb)

Assigning part of speech to words by hand is a common exercise one can find in an elementary grammar class. But here we wish to build an automated tool which can assign the appropriate part-of-speech tag to the words of a given sentence.

One can think of creating handcrafted rules by observing patterns in the language, but this would limit the system's performance to the quality and number of patterns identified by the rule crafter. Thus, this approach is not practically adopted for building POS Taggers.

Instead, a large corpus annotated with correct POS tags for each word is given to the computer and algorithms then learn the patterns automatically from the data and store them in form of a trained model. Later this model can be used to POS tag new sentences.

POS Tagging - Hidden Markov Model

A Hidden Markov Model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states.

In a regular Markov model, the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters. In a hidden Markov model, the state is not directly visible, but output, dependent on the state, is visible.

Hidden Markov Model has two important components

1. Transition Probabilities: The one-step transition probability is the probability of transitioning from one state to another in a single step.

2. Emission Probabilities: The output probabilities for an observation from state. Emission probabilities B = { bi,k = bi(ok) = P(ok | qi) }, where ok is an Observation. Informally, B is the probability that the output is ok given that the current state is qi

For POS tagging, it is assumed that POS are generated as a random process, and each process randomly generates a word.

Hence, the transition matrix denotes the transition probability from one POS to another and the emission matrix denotes the probability that a given word can have a particular POS. Words act as the observations. Some of the basic assumptions are:

Calculating the Probabilities

Consider the given toy corpus

```
EOS/eos
                                        They/pronoun
                                        cut/verb
                                        the/determiner
                                        paper/noun
                                        EOS/eos He/pronoun
                                        asked/verb
                                        for/preposition
                                        his/pronoun
                                        cut/noun.
                                        EOS/eos
                                        Put/verb
                                        the/determiner

                                        paper/noun
                                        in/preposition
                                        the/determiner
                                        cut/noun
                                        EOS/eos
```

Calculating Emission Probability Matrix

Count the no. of times a specific word occus with a specific POS tag in the corpus.
Here, say for **"cut"**

```
count(cut,verb)=1
          count(cut,noun)=2
          count(cut,determiner)=0
```

and so on zero for other tags too.

```
count(cut) = total count of cut = 3
```

Now, calculating the probability
Probability to be filled in the matrix cell at the intersection of cut and verb

```
P(cut/verb)=count(cut,verb)/count(cut)=1/3=0.33
```

Similarly,
Probability to be filled in the cell at he intersection of cut and determiner

```
P(cut/determiner)=count(cut,determiner)/count(cut)=0/3=0
```

Repeat the same for all the word-tag combination and fill the

Calculating Transition Probability Matrix

Count the no. of times a specific tag comes after other POS tags in the corpus.
Here, say for **"determiner"**

```
count(verb,determiner)=2
          count(preposition,determiner)=1
          count(determiner,determiner)=0
          count(eos,determiner)=0
          count(noun,determiner)=0
```

and so on zero for other tags too.

```
count(determiner) = total count of tag 'determiner' = 3
```

Now, calculating the probability Probability to be filled in the cell at he intersection of determiner(in the column) and verb(in the row)

```
P(determiner/verb)=count(verb,determiner)/count(determiner)=2/3=0.66
```

Similarly,
Probability to be filled in the cell at he intersection of determiner(in the column) and noun(in the row)

```
P(determiner/noun)=count(noun,determiner)/count(determiner)=0/3=0
```

Repeat the same for all the tags

Note: **EOS**/eos is a special marker which represents ***End Of Sentence***.

IDE USED: Jupyter Notebook

LIBRARIES USED:

Nltk:

The Natural Language Toolkit (NLTK) is a Python package for natural language processing. NLTK requires Python 3.7, 3.8, 3.9 or 3.10. It can be installed as :

pip install nltk

Pandas:

pandas is a Python package that provides fast, flexible, and expressive data structures designed to make working with "relational" or "labeled" data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. Additionally, it has the broader goal of becoming the most powerful and flexible open source data analysis / manipulation tool available in any language. It is already well on its way towards this goal. It can be installed as :

pip install pandas

Sklearn:

Scikit-learn (formerly scikits.learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support-vector machines,Naïve Bayes etc. It can be installed as:

pip install scikit-learn

Numpy:

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate. It can be installed as:

pip install numpy

PROCEDURE:

1. Firstly import the required libraries
2. Download Universal tagset and Treebank from corpus
3. Read the Treebank tagged sentences and print the starting 2 sentences
4. Split data into training and testing dataset
5. Create a list of train and test tagged words
6. Use set datatype to check how many unique tags are present in training data
7. Compute emission probability and transmission probability
8. Create a transmission matrix of tags and print it
9. Convert the above matrix in dataframe and print it

CODE:

```
# Importing libraries
import nltk
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

#download the treebank corpus from nltk
nltk.download('treebank')

#download the universal tagset from nltk
nltk.download('universal_tagset')

# reading the Treebank tagged sentences
nltk_data = list(nltk.corpus.treebank.tagged_sents(tagset='universal'))

#print the first two sentences along with tags
print(nltk_data[:2])
```

```python
#print each word with its respective tag for first two sentences
for sent in nltk_data[:2]:
  for tuple in sent:
    print(tuple)

# split data into training and validation set in the ratio 80:20
train_set,test_set =train_test_split(nltk_data,train_size=0.80,test_size=0.20,random_state = 101)


# create list of train and test tagged words
train_tagged_words = [ tup for sent in train_set for tup in sent ]
test_tagged_words = [ tup for sent in test_set for tup in sent ]
print(len(train_tagged_words))
print(len(test_tagged_words))

# check some of the tagged words.
train_tagged_words[:5]
```

```python
#use set datatype to check how many unique tags are present in training data
tags = {tag for word,tag in train_tagged_words}
print(len(tags))
print(tags)

# check total words in vocabulary
vocab = {word for word,tag in train_tagged_words}

# compute Emission Probability
def word_given_tag(word, tag, train_bag = train_tagged_words):
    tag_list = [pair for pair in train_bag if pair[1]==tag]
    count_tag = len(tag_list)#total number of times the passed tag occurred in train_bag
    w_given_tag_list = [pair[0] for pair in tag_list if pair[0]==word]
#now calculate the total number of times the passed word occurred as the passed tag.
    count_w_given_tag = len(w_given_tag_list)
    return (count_w_given_tag, count_tag)

# compute  Transition Probability
def t2_given_t1(t2, t1, train_bag = train_tagged_words):
    tags = [pair[1] for pair in train_bag]
    count_t1 = len([t for t in tags if t==t1])
    count_t2_t1 = 0
    for index in range(len(tags)-1):
        if tags[index]==t1 and tags[index+1] == t2:
            count_t2_t1 += 1
    return (count_t2_t1, count_t1)
```

```python
# creating t x t transition matrix of tags, t= no of tags
# Matrix(i, j) represents P(jth tag after the ith tag)

tags_matrix = np.zeros((len(tags), len(tags)), dtype='float32')
for i, t1 in enumerate(list(tags)):
    for j, t2 in enumerate(list(tags)):
        tags_matrix[i, j] = t2_given_t1(t2, t1)[0]/t2_given_t1(t2, t1)[1]

print(tags_matrix)
```

```python
# convert the matrix to a df for better readability
tags_df = pd.DataFrame(tags_matrix, columns = list(tags), index=list(tags))
display(tags_df)
```

OUTPUT:

```
[nltk_data] Downloading package treebank to
[nltk_data]     C:\Users\AIM\AppData\Roaming\nltk_data...
[nltk_data]   Package treebank is already up-to-date!
[nltk_data] Downloading package universal_tagset to
[nltk_data]     C:\Users\AIM\AppData\Roaming\nltk_data...
[nltk_data]   Package universal_tagset is already up-to-date!

[[('Pierre', 'NOUN'), ('Vinken', 'NOUN'), (',', '.'), ('61', 'NUM'), ('years', 'NOUN'), ('old', 'ADJ'), (',', '.'), ('will', 'VERB'), ('jo
in', 'VERB'), ('the', 'DET'), ('board', 'NOUN'), ('as', 'ADP'), ('a', 'DET'), ('nonexecutive', 'ADJ'), ('director', 'NOUN'), ('Nov.', 'NOU
N'), ('29', 'NUM'), ('.', '.')], [('Mr.', 'NOUN'), ('Vinken', 'NOUN'), ('is', 'VERB'), ('chairman', 'NOUN'), ('of', 'ADP'), ('Elsevier',
'NOUN'), ('N.V.', 'NOUN'), (',', '.'), ('the', 'DET'), ('Dutch', 'NOUN'), ('publishing', 'VERB'), ('group', 'NOUN'), ('.', '.')]]

('Pierre', 'NOUN')
('Vinken', 'NOUN')
(',', '.')
('61', 'NUM')
('years', 'NOUN')
('old', 'ADJ')
(',', '.')
('will', 'VERB')
('join', 'VERB')
('the', 'DET')
('board', 'NOUN')
('as', 'ADP')
('a', 'DET')
('nonexecutive', 'ADJ')
('director', 'NOUN')
('Nov.', 'NOUN')
('29', 'NUM')
('.', '.')
('Mr.', 'NOUN')
('Vinken', 'NOUN')
('is', 'VERB')
('chairman', 'NOUN')
('of', 'ADP')
('Elsevier', 'NOUN')
('N.V.', 'NOUN')
(',', '.')
('the', 'DET')
('Dutch', 'NOUN')
('publishing', 'VERB')
('group', 'NOUN')
('.', '.')

 80310
 20366
```

```
[('Drink', 'NOUN'),
 ('Carrier', 'NOUN'),
 ('Competes', 'VERB'),
 ('With', 'ADP'),
 ('Cartons', 'NOUN')]

12
{'ADJ', 'VERB', 'DET', 'PRON', '.', 'NUM', 'CONJ', 'PRT', 'X', 'NOUN', 'ADV', 'ADP'}
```

```
[[6.33009672e-02 1.14563107e-02 5.24271838e-03 1.94174761e-04
  6.60194159e-02 2.17475723e-02 1.68932043e-02 1.14563107e-02
  2.09708735e-02 6.96893215e-01 5.24271838e-03 8.05825219e-02]
 [6.63904250e-02 1.67955801e-01 1.33609578e-01 3.55432779e-02
  3.48066315e-02 2.28360966e-02 5.43278083e-03 3.06629837e-02
  2.15930015e-01 1.10589318e-01 8.38858187e-02 9.23572779e-02]
 [2.06410810e-01 4.02472317e-02 6.03708485e-03 3.30602261e-03
  1.73925534e-02 2.28546783e-02 4.31220367e-04 2.87480245e-04
  4.51343954e-02 6.35906279e-01 1.20741697e-02 9.91806854e-03]
 [7.06150308e-02 4.84738052e-01 9.56719834e-03 6.83371304e-03
  4.19134386e-02 6.83371304e-03 5.01138950e-03 1.41230067e-02
  8.83826911e-02 2.12756261e-01 3.69020514e-02 2.23234631e-02]
 [4.61323895e-02 8.96899477e-02 1.72191828e-01 6.87694475e-02
  9.23720598e-02 7.82104954e-02 6.00793920e-02 2.78940029e-03
  2.56410260e-02 2.18538776e-01 5.25694676e-02 9.29084867e-02]
 [3.53445187e-02 2.07068902e-02 3.57015361e-03 1.42806140e-03
  1.19243130e-01 1.84219927e-01 1.42806144e-02 2.60621198e-02
  2.02427700e-01 3.51660132e-01 3.57015361e-03 3.74866128e-02]
 [1.13611415e-01 1.50384188e-01 1.23490669e-01 6.03732169e-02
  3.51262353e-02 4.06147093e-02 5.48847427e-04 4.39077942e-03
  9.33040585e-03 3.49066973e-01 5.70801310e-02 5.59824370e-02]
 [8.29745606e-02 4.01174158e-01 1.01369865e-01 1.76125243e-02
  4.50097844e-02 5.67514673e-02 2.34833662e-03 1.17416831e-03
  1.21330721e-02 2.50489235e-01 9.39334650e-03 1.95694715e-02]
 [1.76821072e-02 2.06419379e-01 5.68902567e-02 5.41995019e-02
  1.60868734e-01 3.07514891e-03 1.03786280e-02 1.85085520e-01
  7.57255405e-02 6.16951771e-02 2.57543717e-02 1.42225638e-01]
 [1.25838192e-02 1.49133503e-01 1.31063312e-02 4.65906132e-03
  2.40094051e-01 9.14395228e-03 4.24540639e-02 4.39345129e-02
  2.88252197e-02 2.62344331e-01 1.68945398e-02 1.76826611e-01]
 [1.30721495e-01 3.39022487e-01 7.13731572e-02 1.20248254e-02
```

	ADJ	VERB	DET	PRON	.	NUM	CONJ	PRT	X	NOUN	ADV	ADP
ADJ	0.063301	0.011456	0.005243	0.000194	0.066019	0.021748	0.016893	0.011456	0.020971	0.696893	0.005243	0.080583
VERB	0.066390	0.167956	0.133610	0.035543	0.034807	0.022836	0.005433	0.030663	0.215930	0.110589	0.083886	0.092357
DET	0.206411	0.040247	0.006037	0.003306	0.017393	0.022855	0.000431	0.000287	0.045134	0.635906	0.012074	0.009918
PRON	0.070615	0.484738	0.009567	0.006834	0.041913	0.006834	0.005011	0.014123	0.088383	0.212756	0.036902	0.022323
.	0.046132	0.089690	0.172192	0.068769	0.092372	0.078210	0.060079	0.002789	0.025641	0.218539	0.052569	0.092908
NUM	0.035345	0.020707	0.003570	0.001428	0.119243	0.184220	0.014281	0.026062	0.202428	0.351660	0.003570	0.037487
CONJ	0.113611	0.150384	0.123491	0.060373	0.035126	0.040615	0.000549	0.004391	0.009330	0.349067	0.057080	0.055982
PRT	0.082975	0.401174	0.101370	0.017613	0.045010	0.056751	0.002348	0.001174	0.012133	0.250489	0.009393	0.019569
X	0.017682	0.206419	0.056890	0.054200	0.160869	0.003075	0.010379	0.185086	0.075726	0.061695	0.025754	0.142226
NOUN	0.012584	0.149134	0.013106	0.004659	0.240094	0.009144	0.042454	0.043935	0.028825	0.262344	0.016895	0.176827
ADV	0.130721	0.339022	0.071373	0.012025	0.139255	0.029868	0.006982	0.014740	0.022886	0.032196	0.081458	0.119472
ADP	0.107062	0.008479	0.320931	0.069603	0.038724	0.063275	0.001012	0.001266	0.034548	0.323589	0.014553	0.016958

REFERENCES:

1. https://analyticsindiamag.com/a-guide-to-hidden-markov-model-and-its-applications-in-nlp/
2. https://www.mygreatlearning.com/blog/pos-tagging/#:~:text=HMM%20(Hidden%20Markov%20Model)%20is,%2C%20partial%20discharges%2C%20and%20bioinformatics.
3. https://medium.com/hackernoon/building-a-bigram-hidden-markov-model-for-part-of-speech-tagging-1b784a87ab2c