Name	Hatim Sawai
UID No.	2021300108
Experiment No.	6

Experiment 6

Aim	To calculate emission and transition matrix and find Find POS tags of words in a sentence using Viterbi decoding

1. Installation of NLTK and downloading the required corpus

In [ ]:
```
import re
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from prettytable import PrettyTable
from collections import defaultdict
warnings.filterwarnings('ignore')
```

2. Loading the corpus and preprocessing

In [ ]:
```
# load csv
df = pd.read_csv('../dataset/exp5.csv', encoding='isO-8859-1')
df1 = df[df['Sentence #'].notna()]
print("There are",df1['Sentence #'].iloc[-1].split()[-1],"sentences in the dataset"
df.drop(['Sentence #', 'Tag'], axis=1, inplace=True)
df.head()
```

```
There are 47959 sentences in the dataset
```

Out[ ]:

	Word	POS
0	Thousands	NNS
1	of	IN
2	demonstrators	NNS
3	have	VBP
4	marched	VBN

In [ ]:
```
# print all unique values in POS column
print("Unique values in POS column:",df['POS'].unique())
```

```
Unique values in POS column: ['NNS' 'IN' 'VBP' 'VBN' 'NNP' 'TO' 'VB' 'DT' 'NN' 'CC'
'JJ' '.' 'VBD' 'WP'
 '``' 'CD' 'PRP' 'VBZ' 'POS' 'VBG' 'RB' ',' 'WRB' 'PRP$' 'MD' 'WDT' 'JJR'
 ':' 'JJS' 'WP$' 'RP' 'PDT' 'NNPS' 'EX' 'RBS' 'LRB' 'RRB' '$' 'RBR' ';'
 'UH' 'FW']
```

In [ ]:
```python
def preprocess(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text) # remove punctuation
    text = text.replace("\n", " ") # remove \n
    text = re.sub(r'\W', ' ', text)  # Remove non-word characters
    text = re.sub(r'\s+', ' ', text).strip()  # Remove extra whitespaces
    text = re.sub(r'\d', '', text)  # Remove digits
    return text
```

In [ ]:
```python
tag_mapping = {
    'NN': 'NOUN',
    'NNS': 'NOUN',
    'NNP': 'NOUN',
    'NNPS': 'NOUN',
    'VB': 'VERB',
    'VBD': 'VERB',
    'VBG': 'VERB',
    'VBN': 'VERB',
    'VBP': 'VERB',
    'VBZ': 'VERB',
    'JJ': 'ADJ',
    'JJR': 'ADJ',
    'JJS': 'ADJ',
    'RB': 'ADV',
    'RBR': 'ADV',
    'RBS': 'ADV',
}
```

3. Building Vocabulary

In [ ]:
```python
# convert the dataframe to a dictionary, make value field as list of all the tags o
vocab = {}
for index, row in df.iterrows():
    word = row['Word']
    pos = row['POS']
    tag = tag_mapping.get(row['POS'], 'MODAL')
    # if only string
    if type(word) == str:
        if word == ';' or word == ':' or word == '``' or word == ',' or word == '.'
            continue
        else:
            word = preprocess(word)
    else:
        word = str(word)
        continue
    word = preprocess(word)
    if word in vocab and tag not in vocab[word]:
        vocab[word].append(tag)
```

```
    else:
        if word not in vocab:
            vocab[word] = [tag]

print(vocab)
```

4. Calculating Emission & Transition Probabilities

In [ ]:
```
emission_matrix = defaultdict(lambda: defaultdict(int))
    # calculate the emission probability and store it in the emission matrix
for index, row in df.iterrows():
    word = row['Word']
    tag = tag_mapping.get(row['POS'], 'MODAL')
    if type(word) == str:
        word = preprocess(word)
    else:
        word = str(word)
        continue
    word = preprocess(word)
    emission_matrix[word][tag] += 1
```

In [ ]:
```
emission_table = PrettyTable()
emission_table.field_names = [""] + list(set(tag_mapping.values())) + ['MODAL']
for word in emission_matrix:
    total = sum(emission_matrix[word].values())
    prob = {tag: round(emission_matrix[word][tag] / total, 2) for tag in emission_t
    emission_table.add_row([word] + list(prob.values()))
print("Emission Matrix:")
# print only first 10 rows
print(emission_table[:10])
```

```
Emission Matrix:
+---------------+------+-----+------+-----+-------+
|               | VERB | ADV | NOUN | ADJ | MODAL |
+---------------+------+-----+------+-----+-------+
|   thousands   | 0.0  | 0.0 | 1.0  | 0.0 |  0.0  |
|      of       | 0.0  | 0.0 | 0.0  | 0.0 |  1.0  |
| demonstrators | 0.0  | 0.0 | 1.0  | 0.0 |  0.0  |
|     have      | 1.0  | 0.0 | 0.0  | 0.0 |  0.0  |
|    marched    | 1.0  | 0.0 | 0.0  | 0.0 |  0.0  |
|    through    | 0.0  | 0.0 | 0.0  | 0.0 |  1.0  |
|     london    | 0.0  | 0.0 | 1.0  | 0.0 |  0.0  |
|       to      | 0.0  | 0.0 | 0.0  | 0.0 |  1.0  |
|    protest    | 0.48 | 0.0 | 0.52 | 0.0 |  0.0  |
|      the      | 0.0  | 0.0 | 0.0  | 0.0 |  1.0  |
+---------------+------+-----+------+-----+-------+
```

In [ ]:
```
transition_matrix = defaultdict(lambda: defaultdict(int))
previous_tag = None
for index, row in df.iterrows():
    tag = tag_mapping.get(row['POS'], 'MODAL')
    if previous_tag is not None:
```

```
        transition_matrix[previous_tag][tag] += 1
    previous_tag = tag
```

In [ ]:
```
print("\nTransition Matrix:")
trans_table = PrettyTable()
trans_table.field_names = [""] + list(set(tag_mapping.values())) + ['MODAL']
for tag in transition_matrix:
    total = sum(transition_matrix[tag].values())
    prob = {}
    for tg in set(tag_mapping.values()) | {'MODAL'}:
        prob[tg] = round(transition_matrix[tag][tg] / total, 2)
    trans_table.add_row([tag] + list(prob.values()))
print(trans_table)
```

```
Transition Matrix:
+-------+------+------+------+------+-------+
|       | VERB | ADV  | NOUN | ADJ  | MODAL |
+-------+------+------+------+------+-------+
|  NOUN | 0.01 | 0.18 | 0.25 | 0.55 |  0.01 |
| MODAL | 0.02 | 0.13 | 0.41 | 0.31 |  0.14 |
|  VERB | 0.05 | 0.18 | 0.16 | 0.54 |  0.07 |
|  ADJ  | 0.0  | 0.01 | 0.77 | 0.13 |  0.09 |
|  ADV  | 0.05 | 0.41 | 0.04 | 0.39 |  0.1  |
+-------+------+------+------+------+-------+
```

5. Predicting POS tags using Viterbi Algorithm

In [ ]:
```
# Viterbi Algorithm
def viterbi(words, emission_matrix, transition_matrix):
    tags = list(set(tag_mapping.values())) + ['MODAL']
    pi = np.zeros((len(words), len(tags)))
    bp = np.zeros((len(words), len(tags)), dtype=int)
    for i, word in enumerate(words):
        for j, tag in enumerate(tags):
            if i == 0:
                pi[i][j] = 1
            else:
                max_prob = -1
                max_prob_index = -1
                for k, prev_tag in enumerate(tags):
                    if emission_matrix[word][tag] == 0:
                        emission_matrix[word][tag] = 0.0001
                    if transition_matrix[prev_tag][tag] == 0:
                        transition_matrix[prev_tag][tag] = 0.0001
                    prob = pi[i-1][k] * emission_matrix[word][tag] * transition_mat
                    if prob > max_prob:
                        max_prob = prob
                        max_prob_index = k
                pi[i][j] = max_prob
                bp[i][j] = max_prob_index
    max_prob = -1
    max_prob_index = -1
    for j, tag in enumerate(tags):
        if pi[-1][j] > max_prob:
            max_prob = pi[-1][j]
```

```
            max_prob_index = j
    predicted_tags = [tags[max_prob_index]]
    for i in range(len(words)-1, 0, -1):
        max_prob_index = bp[i][max_prob_index]
        predicted_tags.append(tags[max_prob_index])
    return list(reversed(predicted_tags))
```

In [ ]:
```
sample_sentence = "The quick brown fox jumps over the lazy dog."
predicted_tags = viterbi(sample_sentence.split(), emission_matrix, transition_matri
print("\nPredicted tags for the sample sentence:")
print(predicted_tags)
```

```
Predicted tags for the sample sentence:
['MODAL', 'ADJ', 'NOUN', 'NOUN', 'NOUN', 'MODAL', 'MODAL', 'ADJ', 'NOUN']
```

6. Curiosity Questions

Q1. What are different algorithms that can be used to find POS tags using emission and transition probabilities?

Ans: Different algorithms that can be used to find POS tags using emission and transition probabilities are:
- Viterbi Algorithm
- Forward-Backward Algorithm
- Baum-Welch Algorithm

Q2. What is Viterbi Algorithm?

Ans: Viterbi Algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states—called the Viterbi path—that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models. Viterbi decoding efficiently determines the most probable path from the exponentially many possibilities. It finds the highest probability given for a word against all our tags by looking through our transmission and emission probabilities, multiplying the probabilities, and then finding the max probability.

Q3. State advantages & disadvantages of Viterbi Algorithm?

Ans: Advantages of Viterbi Algorithm are:
1. It is a dynamic programming algorithm and is efficient.
2. Possible to reconstruct lost data.

Disadvantages of Viterbi Algorithm are:
1. Computation becomes complex for large number of states.
2. More bandwidth needed for redundant information.

6. Conclusion

In this experiment we learned how to calculate the emission and transition matrix for tagging Parts of Speech. We also learned how to find the POS tags of a given sentence using Viterbi decoding.