Name	Hatim Sawai
UID No.	2021300108
Experiment No.	5

Experiment 5

Aim	To calculate emission and transition matrix for tagging Parts of Speech using Hidden Markov Model. Find POS tag of given sentence using HMM.

1. Installation of NLTK and downloading the required corpus

In [ ]:
```
import re
import nltk
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')
```

```
C:\Users\shabb\AppData\Local\Temp\ipykernel_2288\4190258577.py:5: DeprecationWarnin
g:
Pyarrow will become a required dependency of pandas in the next major release of pan
das (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better inte
roperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
```

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 [ ]:
```
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 [ ]:
```
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 [ ]:
```
# 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:
+---------------+-----+-----+------+------+-------+
|               | ADJ | ADV | VERB | NOUN | MODAL |
+---------------+-----+-----+------+------+-------+
|   thousands   | 0.0 | 0.0 | 0.0  | 1.0  |  0.0  |
|       of      | 0.0 | 0.0 | 0.0  | 0.0  |  1.0  |
| demonstrators | 0.0 | 0.0 | 0.0  | 1.0  |  0.0  |
|      have     | 0.0 | 0.0 | 1.0  | 0.0  |  0.0  |
|    marched    | 0.0 | 0.0 | 1.0  | 0.0  |  0.0  |
|    through    | 0.0 | 0.0 | 0.0  | 0.0  |  1.0  |
|     london    | 0.0 | 0.0 | 0.0  | 1.0  |  0.0  |
|       to      | 0.0 | 0.0 | 0.0  | 0.0  |  1.0  |
|    protest    | 0.0 | 0.0 | 0.48 | 0.52 |  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:
+-------+------+------+------+------+-------+
|       | ADJ  | ADV  | VERB | NOUN | MODAL |
+-------+------+------+------+------+-------+
|  NOUN | 0.01 | 0.01 | 0.55 | 0.18 |  0.25 |
| MODAL | 0.02 | 0.14 | 0.31 | 0.13 |  0.41 |
|  VERB | 0.05 | 0.07 | 0.54 | 0.18 |  0.16 |
|  ADJ  | 0.0  | 0.09 | 0.13 | 0.01 |  0.77 |
|  ADV  | 0.05 | 0.1  | 0.39 | 0.41 |  0.04 |
+-------+------+------+------+------+-------+
```

5. Predicting POS tags for a given sentence

In [ ]:
```
# Function to perform POS tagging
def predict_pos(sentence):
    predicted_tags = []
    for word in sentence.split():
        word = preprocess(word)
        if word in vocab:
            predicted_tag = max(vocab[word], key=lambda tag: emission_matrix[word][
```

```
        else:
            predicted_tag = 'UNK'  # If word not in vocabulary, assign 'UNK' tag
        predicted_tags.append(predicted_tag)
    return predicted_tags
```

In [ ]:
```
sample_sentence = "The quick brown fox jumps over the lazy dog."
predicted_tags = predict_pos(sample_sentence)
print("\nPredicted POS Tags:")
print(predicted_tags)
```

```
Predicted POS Tags:
['MODAL', 'ADJ', 'NOUN', 'NOUN', 'NOUN', 'MODAL', 'MODAL', 'ADJ', 'NOUN']
```

6. Curiosity Questions

Q1. List a few ways for tagging parts of speech?

Ans: There are several ways to tag parts of speech. Some of them are: - Rule-based tagging
- Stochastic tagging
- Transformation-based tagging
- Hidden Markov Models
- Maximum Entropy Models
- Deep Learning Models

Q2. How do you find the most probable sequence of POS tags from a sequence of text?

Ans: The most probable sequence of POS tags from a sequence of text can be found using Hidden Markov Models. The Viterbi algorithm is used to find the most probable sequence of POS tags from a sequence of text.

Q3. Differentiate between Markov chain and Markov model?

Ans: Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. Markov model is a model that assumes that the future state depends only on the current state and not on the sequence of events that preceded it. Markov chain is a special case of Markov model.

Q4. How you can identify whether a system follows a Markov Process?

Ans: A system follows a Markov Process if it satisfies the Markov property. The Markov property states that the future state of the system depends only on the current state and not on the sequence of events that preceded it. If the system satisfies the Markov property, then it follows a Markov Process.

Q5. Explain the use of Markov Chains in text generation algorithms.

Ans: Markov Chains are used in text generation algorithms to generate text that is similar to the input text. Markov Chains are used to model the probability of the next word in a sequence of words given the current word. This probability is used to generate the next word in the sequence. Markov Chains are used to generate text that is similar to the input

text by sampling the next word from the probability distribution of the next word given the current word.

6. Conclusion

In this experiment we learned about Hidden Markov Models and how to implement a Hidden Markov Model for tagging Parts of Speech. We also learned how to calculate the emission and transition matrix for tagging Parts of Speech using Hidden Markov Model. We also learned how to find the POS tag of a given sentence using Hidden Markov Model.