Name	Pranay Singhvi
UID No.	2021300126
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.
Theory	Hidden Markov Model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (i.e. hidden) states. HMM is a doubly stochastic process, where the observed data is assumed to be generated by a stochastic process. The model is defined by the following components: 1. A set of N states, where N is the number of states in the model. 2. A set of M observation symbols, where M is the number of distinct observation symbols per state. 3. A state transition probability matrix A, where A[i][j] is the probability of transitioning from state i to state j. 4. An observation probability matrix B, where B[i][j] is the probability of observing symbol j from state i. 5. An initial state distribution π, where π[i] is the probability of starting in state i. Given a sequence of observations, the model aims to find the most likely sequence of states that generated the observations. This is done using the Viterbi algorithm, which is a dynamic programming algorithm for finding the most likely sequence of hidden states.

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')
```

2. Loading the corpus and preprocessing

In [ ]:
```
# load csv
df = pd.read_csv('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-w 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 of that w in the sentence
vocab = {}
for index, row in df.iterrows():
    w = row['Word']
    pos = row['POS']
    tag = tag_mapping.get(row['POS'], 'MODAL')
    # if only string
    if type(w) == str:
        if w == ';' or w == ':' or w == '``' or w == ',' or w == '.':
            continue
        else:
            w = preprocess(w)
    else:
        w = str(w)
        continue
    w = preprocess(w)
    if w in vocab and tag not in vocab[w]:
        vocab[w].append(tag)
    else:
        if w not in vocab:
            vocab[w] = [tag]
```

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():
    w = row['Word']
    tag = tag_mapping.get(row['POS'], 'MODAL')
    if type(w) == str:
        w = preprocess(w)
    else:
        w = str(w)
        continue
    w = preprocess(w)
    emission_matrix[w][tag] += 1
```

In [ ]:
```
emi_tab = PrettyTable()
emi_tab.field_names = [""] + list(set(tag_mapping.values())) + ['MODAL']
for w in emission_matrix:
    total = sum(emission_matrix[w].values())
    prob = {tag: round(emission_matrix[w][tag] / total, 2) for tag in emi_tab.field_names[1:]}
    emi_tab.add_row([w] + list(prob.values()))
print("Emission Matrix:")
# print only first 10 rows
print(emi_tab[: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 [ ]:
```
trans_mat = defaultdict(lambda: defaultdict(int))
prev_tag = None
for index, row in df.iterrows():
    tag = tag_mapping.get(row['POS'], 'MODAL')
    if prev_tag is not None:
        trans_mat[prev_tag][tag] += 1
    prev_tag = tag
```

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

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

5. Predicting POS tags for a given sentence

In [ ]:
```
# Function to perform POS tagging
def predict_pos(sentence):
    pre_tags = []
    for w in sentence.split():
        w = preprocess(w)
        if w in vocab:
            pre_tag = max(vocab[w], key=lambda tag: emission_matrix[w][tag] if tag in emission_matrix[w] else 0)
        else:
            pre_tag = 'UNK'  # If w not in vocabulary, assign 'UNK' tag
        pre_tags.append(pre_tag)
    return pre_tags
```

In [ ]:
```
sample_sentence = "The sun dipped below the horizon, casting a warm, golden glow across the tranquil, rippling waters of the lake."
pre_tags = predict_pos(sample_sentence)
print("\nPredicted POS Tags:")
print(pre_tags)
```

```
Predicted POS Tags:
['MODAL', 'NOUN', 'VERB', 'MODAL', 'MODAL', 'NOUN', 'VERB', 'MODAL', 'ADJ', 'ADJ', 'UNK', 'MODAL', 'MODAL', 'UNK', 'UNK', 'NOUN', 'MODAL', 'MODAL', '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:

1. **Rule-Based Tagging:**
   - **Theory:** Rule-based tagging involves creating a set of linguistic rules and patterns to determine the part of speech for each word in a sentence. These rules are based on grammatical structures and word patterns.
   - **Example:** If a word ends with "-ing," it is likely a gerund (verb form used as a noun).
2. **Stochastic Tagging:**
   - **Theory:** Stochastic tagging involves using statistical models to assign probabilities to different parts of speech for a given word. These models learn from training data to estimate the likelihood of a word belonging to a specific part of speech.
   - **Example:** Hidden Markov Models and Maximum Entropy Models are examples of stochastic tagging methods.
3. **Transformation-Based Tagging:**
   - **Theory:** Transformation-based tagging is a type of rule-based tagging where rules are learned from training data through a process of error-driven transformation. The system starts with an initial tagging and improves it iteratively.
   - **Example:** Brill Tagger is a transformation-based tagging algorithm.
4. **Hidden Markov Models (HMM):**
   - **Theory:** HMMs are probabilistic models that represent a sequence of observable events (words) and a sequence of hidden states (parts of speech). The model estimates the probability of transitioning between states and emitting observations.
   - **Example:** In POS tagging, the hidden states are parts of speech, and the observations are words.
5. **Maximum Entropy Models:**

- **Theory:** Maximum Entropy Models aim to find the probability distribution that maximizes entropy (uncertainty) given a set of constraints. In POS tagging, these models learn the most likely part-of-speech tags based on observed features.
- **Example:** Conditional Maximum Entropy Models are commonly used for part-of-speech tagging tasks.

6. **Deep Learning Models:**

- **Theory:** Deep learning models, such as recurrent neural networks (RNNs) and long short-term memory networks (LSTMs), use neural networks with multiple layers to capture complex relationships and dependencies in sequential data like language.
- **Example:** Bidirectional LSTMs are effective for part-of-speech tagging as they consider both past and future context.

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

Ans: Hidden Markov Models (HMMs) and the Viterbi algorithm are applied in Part-of-Speech (POS) tagging to find the most probable sequence of POS tags from a given text. HMMs model the probabilistic relationships between POS tags and observed words. Trained on labeled datasets, HMMs estimate transition probabilities between tags and emission probabilities for observed words. The Viterbi algorithm efficiently determines the most likely sequence of hidden states (POS tags) given a sequence of observations (words). It involves initialization, recursion, backtracking, and termination steps. In POS tagging, each word in a sentence corresponds to an observation, and the Viterbi algorithm calculates the optimal sequence of POS tags by considering both transition and emission probabilities. This approach efficiently narrows down the search space, making it computationally feasible for large datasets. Ultimately, the combination of HMMs and the Viterbi algorithm provides a probabilistic framework for accurate and efficient POS tagging in natural language processing tasks.

Q3. Differentiate between Markov chain and Markov model?

Ans: A Markov chain and a Markov model are related concepts in probability theory and stochastic processes, but they have distinct characteristics:

Markov Chain:

1. **Definition:** A Markov chain is a mathematical model that describes a sequence of events where the probability of transitioning to any particular state depends solely on the current state and time elapsed, and not on the sequence of events leading to that state.
2. **Memoryless Property:** Markov chains possess the Markov property, indicating that the future state depends only on the current state and is independent of the past states.
3. **States and Transitions:** Markov chains consist of a set of states and probabilities of transitioning between those states.
4. **Example:** Consider a board game where the probability of moving to the next square depends only on the current square and not on how the game reached that point.

Markov Model:

1. **Definition:** A Markov model is a broader term that encompasses various mathematical models, including Markov chains. It is a general framework for modeling systems where a process evolves over time through a series of states, and transitions between states are probabilistic.
2. **Incorporating More Complexity:** While Markov chains specifically refer to a simple memoryless process, Markov models can include additional complexity, such as different types of states, continuous-time transitions, or multiple interacting processes.
3. **Application:** Markov models find applications in various fields, including economics, biology, and computer science, allowing for more flexibility and customization to specific scenarios.

In summary, a Markov chain is a specific type of Markov model that represents a simple, memoryless process with probabilistic state transitions. Markov models, in a broader sense, encompass a range of models that can incorporate additional features and complexities beyond the basic Markov chain structure.

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

Ans: To identify if a system follows a Markov process:

1. **Markov Property:**
   - Check if future behavior depends only on the current state, independent of past states.
2. **Memoryless Transitions:**
   - Confirm that transitioning to the next state depends solely on the current state, with no influence from past events.
3. **Transition Probabilities:**
   - Ensure that transition probabilities are fixed and not influenced by previous states.
4. **State Space:**
   - Define a discrete set of possible states for straightforward application of the Markov property.
5. **Conditional Independence:**
   - Verify that the probability of the next state, given the current state, is conditionally independent of past states.
6. **Stationary Transition Probabilities (Optional):**
   - Check if transition probabilities remain constant over time.
7. **Transition Diagrams/Matrices:**
   - Use visual aids like transition diagrams or matrices to represent and analyze state transitions.

If these conditions hold, the system can be considered a Markov process. Memorylessness and independence of past events in state transitions are key indicators.

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

Ans: Markov Chains are widely employed in text generation algorithms to model and generate sequences of words or characters based on the probability of transitioning from one state to another. Here's how they are used:

1. **Modeling Sequential Dependencies:**
   - Markov Chains capture sequential dependencies in a text corpus. Each state represents a word or a sequence of words, and transitions between states are determined by the likelihood of a word following another.
2. **Order of the Markov Chain:**
   - The "order" of a Markov Chain determines the number of previous states considered when predicting the next state. For text generation, a higher-order Markov Chain incorporates more context, providing a more sophisticated understanding of language structure.
3. **Transition Probabilities:**
   - Transition probabilities are estimated from a training dataset, representing how often specific sequences of words occur. These probabilities guide the generation process, determining the likelihood of transitioning to different words.
4. **Text Generation Process:**
   - To generate text, start with an initial state (word or sequence), and iteratively choose the next state based on the transition probabilities. This process is repeated until the desired length of text is generated.
5. **N-gram Models:**
   - Markov Chains are related to N-gram models, where the order of the Markov Chain corresponds to the size of the N-gram. For example, a bigram model is a second-order Markov Chain.
6. **Applications:**
   - Markov Chains find applications in various text generation tasks, including auto-completion suggestions, chatbot responses, and even creative writing or poetry generation.
7. **Limitations:**
   - While Markov Chains capture local dependencies well, they may struggle with long-range dependencies in language. Higher-order Markov Chains mitigate this to some extent but may face data sparsity issues.
8. **Adaptations:**
   - Some text generation algorithms combine Markov Chains with other techniques, such as neural networks, to capture more complex patterns and semantic understanding.

In essence, Markov Chains provide a simple yet effective way to model the probabilistic nature of language and generate coherent text based on learned sequential dependencies from training data.

6. Conclusion

In this experiment, we explored Hidden Markov Models (HMMs) for Part-of-Speech (POS) tagging. We implemented an HMM, calculated emission and transition matrices, and applied the model to identify POS tags in sentences.