- Name: Deep Salunkhe
- Roll No.:21102A0014

```
pip install hmmlearn
```

```
Collecting hmmlearnNote: you may need to restart the kernel to use
updated packages.

  Downloading hmmlearn-0.3.2-cp311-cp311-win_amd64.whl.metadata (3.0
kB)
Requirement already satisfied: numpy>=1.10 in d:\anaconda\lib\site-
packages (from hmmlearn) (1.26.4)
Requirement already satisfied: scikit-learn!=0.22.0,>=0.16 in d:\
anaconda\lib\site-packages (from hmmlearn) (1.2.2)
Requirement already satisfied: scipy>=0.19 in d:\anaconda\lib\site-
packages (from hmmlearn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in d:\anaconda\lib\site-
packages (from scikit-learn!=0.22.0,>=0.16->hmmlearn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in d:\anaconda\
lib\site-packages (from scikit-learn!=0.22.0,>=0.16->hmmlearn) (2.2.0)
Downloading hmmlearn-0.3.2-cp311-cp311-win_amd64.whl (125 kB)
   ---------------------------------------- 0.0/125.4 kB ? eta -:--:--
   --- ------------------------------------ 10.2/125.4 kB ? eta
-:--:--
   ---------------------------------------  122.9/125.4 kB 1.8 MB/s eta
0:00:01
   ---------------------------------------- 125.4/125.4 kB 1.5 MB/s eta
0:00:00
Installing collected packages: hmmlearn
Successfully installed hmmlearn-0.3.2
```

```
import nltk
from nltk import pos_tag
from nltk.probability import FreqDist

# Download required NLTK data
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

def calculate_matrices(sentence):
    # Tokenize and tag the sentence
    tokens = nltk.word_tokenize(sentence)
    tagged_sentence = pos_tag(tokens)

    # Calculate transition matrix
    transitions = FreqDist(((tag1, tag2) for ((word1, tag1), (word2,
tag2)) in nltk.bigrams(tagged_sentence)))
    unique_tags = set(tag for word, tag in tagged_sentence)
    transition_matrix = {tag1: {tag2: transitions[(tag1, tag2)] for
tag2 in unique_tags} for tag1 in unique_tags}
```

```
    # Calculate emission matrix
    emissions = FreqDist((tag, word) for word, tag in tagged_sentence)
    emission_matrix = {tag: {word: emissions[(tag, word)] for word, _
in tagged_sentence} for tag in unique_tags}

    return transition_matrix, emission_matrix

# Example usage
sentence = "Ram is standing on the bank of the river"
transition_matrix, emission_matrix = calculate_matrices(sentence)

print("Transition Matrix:")
print(transition_matrix)
print("\nEmission Matrix:")
print(emission_matrix)
```

```
Transition Matrix:
{'NNP': {'NNP': 0, 'VBG': 0, 'VBZ': 1, 'NN': 0, 'IN': 0, 'DT': 0},
'VBG': {'NNP': 0, 'VBG': 0, 'VBZ': 0, 'NN': 0, 'IN': 1, 'DT': 0},
'VBZ': {'NNP': 0, 'VBG': 1, 'VBZ': 0, 'NN': 0, 'IN': 0, 'DT': 0},
'NN': {'NNP': 0, 'VBG': 0, 'VBZ': 0, 'NN': 0, 'IN': 1, 'DT': 0}, 'IN':
{'NNP': 0, 'VBG': 0, 'VBZ': 0, 'NN': 0, 'IN': 0, 'DT': 2}, 'DT':
{'NNP': 0, 'VBG': 0, 'VBZ': 0, 'NN': 2, 'IN': 0, 'DT': 0}}

Emission Matrix:
{'NNP': {'Ram': 1, 'is': 0, 'standing': 0, 'on': 0, 'the': 0, 'bank':
0, 'of': 0, 'river': 0}, 'VBG': {'Ram': 0, 'is': 0, 'standing': 1,
'on': 0, 'the': 0, 'bank': 0, 'of': 0, 'river': 0}, 'VBZ': {'Ram': 0,
'is': 1, 'standing': 0, 'on': 0, 'the': 0, 'bank': 0, 'of': 0,
'river': 0}, 'NN': {'Ram': 0, 'is': 0, 'standing': 0, 'on': 0, 'the':
0, 'bank': 1, 'of': 0, 'river': 1}, 'IN': {'Ram': 0, 'is': 0,
'standing': 0, 'on': 1, 'the': 0, 'bank': 0, 'of': 1, 'river': 0},
'DT': {'Ram': 0, 'is': 0, 'standing': 0, 'on': 0, 'the': 2, 'bank': 0,
'of': 0, 'river': 0}}
```

```
[nltk_data] Downloading package punkt to C:\Users\Deep
[nltk_data]     Salunkhe\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Deep Salunkhe\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

```
def calculate_matrices_hardcoded(sentence, pos_tags):
    words = sentence.split()

    # Calculate transition matrix
    transition_matrix = {}
    for i in range(len(pos_tags) - 1):
        if pos_tags[i] not in transition_matrix:
```

```
            transition_matrix[pos_tags[i]] = {}
        if pos_tags[i+1] not in transition_matrix[pos_tags[i]]:
            transition_matrix[pos_tags[i]][pos_tags[i+1]] = 0
        transition_matrix[pos_tags[i]][pos_tags[i+1]] += 1

    # Calculate emission matrix
    emission_matrix = {}
    for i, tag in enumerate(pos_tags):
        if tag not in emission_matrix:
            emission_matrix[tag] = {}
        if words[i] not in emission_matrix[tag]:
            emission_matrix[tag][words[i]] = 0
        emission_matrix[tag][words[i]] += 1

    return transition_matrix, emission_matrix

# Example usage
sentence = "Ram is standing on the bank of the river"
pos_tags = ["NNP", "VBZ", "VBG", "IN", "DT", "NN", "IN", "DT", "NN"]

transition_matrix, emission_matrix =
calculate_matrices_hardcoded(sentence, pos_tags)

print("Transition Matrix:")
print(transition_matrix)
print("\nEmission Matrix:")
print(emission_matrix)
```

```
Transition Matrix:
{'NNP': {'VBZ': 1}, 'VBZ': {'VBG': 1}, 'VBG': {'IN': 1}, 'IN': {'DT':
2}, 'DT': {'NN': 2}, 'NN': {'IN': 1}}

Emission Matrix:
{'NNP': {'Ram': 1}, 'VBZ': {'is': 1}, 'VBG': {'standing': 1}, 'IN':
{'on': 1, 'of': 1}, 'DT': {'the': 2}, 'NN': {'bank': 1, 'river': 1}}
```