Name : Bodhisatya Ghosh
Class : CSE DS
UID : 2021700026
Subject : NLP
Experiment number : 5

Aim: Print emission & transition matrix
Calculate POS tags for a given sentence

Questions

1. Ways for tagging parts of speech:

   a. Rule-based tagging: Utilizes predefined grammatical rules to assign parts of speech based on word patterns and context.

   b. Dictionary-based tagging: Matches words against a pre-built dictionary that includes information about the part of speech of each word.

   c. Probabilistic tagging: Uses statistical models to determine the likelihood of a word belonging to a specific part of speech based on training data.

   d. Machine learning-based tagging: Employs machine learning algorithms, such as Hidden Markov Models (HMMs) or Conditional Random Fields (CRFs), trained on annotated corpora to predict parts of speech.

2. Finding the most probable sequence of POS tags:

   a. Hidden Markov Models (HMMs): Use the Viterbi algorithm to find the most probable sequence of POS tags based on the emission and transition probabilities.

   b. Conditional Random Fields (CRFs): Optimize a conditional probability model that considers the entire sequence, taking into account both local and global context.

3. Markov chain vs. Markov model:

- Markov Chain: A mathematical model representing a sequence of events where the probability of transitioning to any particular state depends solely on the current state. It has discrete states and transition probabilities.

- Markov Model: A broader term that encompasses various mathematical models, including Markov Chains. Markov Models can refer to systems with both discrete and continuous states, and they may have additional parameters beyond transition probabilities.

4. Identifying whether a system follows a Markov Process:

- If a system exhibits the Markov property, meaning the future state depends only on the present state and not on the sequence of events leading to the present state, it can be considered a Markov Process. This property can be assessed by analyzing the conditional probability distribution of future states given the current state.

5. Use of Markov Chains in text generation algorithms:

- Markov Chains can model the transition probabilities between words or characters in a text. By analyzing a training corpus, the probabilities of transitioning from one word to another can be learned.

- In text generation, a Markov Chain can be used to predict the next word or sequence of words based on the current state (previous words). This allows for the generation of coherent and contextually relevant text, making Markov Chains a simple yet effective tool for text generation algorithms.

In [ ]:
```python
import numpy as np

def train_hmm(tagged_corpus):
    states = set()
    emissions = set()
    transitions = {}

    for sentence in tagged_corpus:
        prev_state = None
        for word, tag in sentence:
            states.add(tag)
            emissions.add(word)

            if prev_state is not None:
                if (prev_state, tag) in transitions:
                    transitions[(prev_state, tag)] += 1
                else:
                    transitions[(prev_state, tag)] = 1
            prev_state = tag

    # transition counts to get probabilities
    transition_matrix = np.zeros((len(states), len(states)))
    for (i, state1) in enumerate(states):
        for (j, state2) in enumerate(states):
            transition_matrix[i, j] = transitions.get((state1, state2), 0)

    transition_matrix /= transition_matrix.sum(axis=1, keepdims=True)

    # emission probabilities
    emission_matrix = np.zeros((len(states), len(emissions)))
    for sentence in tagged_corpus:
        for word, tag in sentence:
            emission_matrix[list(states).index(tag), list(emissions).index(word)] += 1

    emission_matrix /= emission_matrix.sum(axis=1, keepdims=True)

    return list(states), list(emissions), transition_matrix, emission_matrix
```

In [ ]:
```
def pos_tag(sentence, states, emissions, transition_matrix, emission_matrix):
    tags = []

    for word in sentence:
        if word not in emissions:
            #Using the first tag as default tag
            tags.append(states[0])
        else:
            word_index = emissions.index(word)
            state_index = np.argmax(emission_matrix[:, word_index])
            tags.append(states[state_index])

    return list(zip(sentence, tags))
```

In [ ]:
```
tagged_corpus = [
    [("The", "DT"), ("cat", "NN"), ("is", "VBZ"), ("on", "IN"), ("the", "DT"), ("mat", "NN")],
    [("A", "DT"), ("quick", "JJ"), ("brown", "JJ"), ("fox", "NN"), ("jumps", "VBZ"), ("over", "IN"), ("the", "DT"), ("lazy", "
    [("She", "PRP"), ("sells", "VBZ"), ("seashells", "NNS"), ("by", "IN"), ("the", "DT"), ("seashore", "NN")],
    [("Peter", "NNP"), ("Piper", "NNP"), ("picked", "VBD"), ("a", "DT"), ("peck", "NN"), ("of", "IN"), ("pickled", "VBN"), ("p
    [("How", "WRB"), ("much", "JJ"), ("wood", "NN"), ("would", "MD"), ("a", "DT"), ("woodchuck", "NN"), ("chuck", "VB"), ("if"
]

states, emissions, transition_matrix, emission_matrix = train_hmm(tagged_corpus)

# Print the emission matrix
print("Emission Matrix:")
print("          " + " ".join(emissions))
for i, row in enumerate(emission_matrix):
    print(f"{states[i]:<4} {row}")

# Print the transition matrix
print("\nTransition Matrix:")
print("          " + " ".join(states))
for i, row in enumerate(transition_matrix):
    print(f"{states[i]:<4} {row}")

example_sentence = "The quick brown fox jumps over the lazy dog"
example_sentence = example_sentence.split()

result = pos_tag(example_sentence, states, emissions, transition_matrix, emission_matrix)


print("\nPOS Tagging Result:")
print(result)
```

```
Emission Matrix:
          peppers sells of A She is pickled The seashore brown a could picked the much mat by wood jumps seashells fox cat wood
chuck would over on if dog quick peck Piper chuck How lazy Peter
IN   [0.  0.  0.2 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.2 0.
 0.  0.  0.  0.  0.  0.  0.2 0.2 0.2 0.  0.  0.  0.  0.  0.  0.  0. ]
VBZ  [0.         0.33333333 0.         0.         0.         0.33333333
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.33333333 0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.        ]
NNP  [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.5 0.  0.  0.  0.5]
VB   [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
VBD  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
NN   [0.  0.  0.  0.  0.  0.  0.  0.  0.1 0.  0.  0.  0.  0.  0.  0.1 0.  0.2
 0.  0.  0.1 0.1 0.2 0.  0.  0.  0.  0.1 0.  0.1 0.  0.  0.  0.  0. ]
NNS  [0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.
 0.  0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
MD   [0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.5 0.  0.  0.  0.  0.  0.
 0.  0.  0.  0.  0.  0.5 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0. ]
DT   [0.    0.    0.    0.125 0.    0.    0.    0.125 0.    0.    0.375 0.
 0.    0.375 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.
 0.    0.    0.    0.    0.    0.    0.    0.    0.    0.    0.   ]
PRP  [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
JJ   [0.   0.   0.   0.   0.   0.   0.   0.   0.   0.25 0.   0.   0.   0.
 0.25 0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.   0.
 0.25 0.   0.   0.   0.   0.25 0.  ]
VBN  [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
WRB  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]

Transition Matrix:
          IN VBZ NNP VB VBD NN NNS MD DT PRP JJ VBN WRB
IN   [0.  0.  0.  0.  0.  0.  0.  0.  0.8 0.  0.  0.2 0. ]
VBZ  [0.66666667 0.         0.         0.         0.         0.
 0.33333333 0.         0.         0.         0.         0.
 0.        ]
NNP  [0.  0.  0.5 0.  0.5 0.  0.  0.  0.  0.  0.  0.  0. ]
VB   [0.5 0.  0.  0.  0.  0.5 0.  0.  0.  0.  0.  0.  0. ]
VBD  [0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
NN   [0.16666667 0.33333333 0.         0.16666667 0.         0.
 0.         0.33333333 0.         0.         0.         0.
 0.        ]
NNS  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
MD   [0.  0.  0.  0.5 0.  0.  0.  0.  0.5 0.  0.  0.  0. ]
DT   [0.   0.   0.   0.   0.   0.75 0.   0.   0.   0.   0.25 0.   0.  ]
PRP  [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
JJ   [0.   0.   0.   0.   0.   0.75 0.   0.   0.   0.   0.25 0.   0.  ]
VBN  [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
WRB  [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]

POS Tagging Result:
[('The', 'DT'), ('quick', 'JJ'), ('brown', 'JJ'), ('fox', 'NN'), ('jumps', 'VBZ'), ('over', 'IN'), ('the', 'DT'), ('lazy', 'J
J'), ('dog', 'NN')]
```

```
In [ ]:
```