## Indian Institute of Information Technology Surat

****

# Lab Report on

# Natural Language Processing (CS 601) Practical

**Submitted by**

### [RAHUL KUMAR SINGH] (UI21CS44)

**Course Faculty**

### Mrs. Nidhi Desai

## Department of Computer Science and Engineering

## Indian Institute of Information Technology Surat

## Gujarat-394190, India

**Nov-2024**

## Lab No: 7

**Aim:**

To implement and log POS tagging using the Viterbi algorithm with HMM.

**Description:**

* Viterbi algorithm is used for POS tagging, estimating the most probable state sequence given observations.
* **Transmission Probabilities**: Probability of transitioning between POS tags, updated iteratively during training to maximize tag sequence likelihood.
* **Emission Probabilities**: Probability of observing a word given a POS tag, refined to fit training data.
* **Recursion Step**: Iterates over the sequence, calculating probabilities of each tag for each word based on previous tag states.
* **Backtracking**: Determines the most probable tag sequence by tracing back through the best state transitions.
* **Likelihood Calculation**: The likelihood of a word sequence is determined based on the final Viterbi probabilities, guiding model evaluation.
* **Transition Probabilities (backpointer)**: Backpointer tracks the most probable transitions between POS tags across time steps.

## Source Code:

**import numpy as np**

**from collections import defaultdict**

**import logging**

**for handler in logging.root.handlers[:]:**

**logging.root.removeHandler(handler)**

**logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s', handlers=[logging.StreamHandler()])**

**corpus = [**

**('dog', 'NOUN'),**

**('barks', 'VERB'),**

**('loudly', 'ADJ'),**

**('cat', 'NOUN'),**

**('meows', 'VERB'),**

**('quickly', 'ADJ'),**

**('dog', 'NOUN'),**

**('chases', 'VERB')**

**]**

**words = list(set(word for word, pos in corpus))**

**tags = list(set(pos for word, pos in corpus))**

**transition\_counts = defaultdict(lambda: defaultdict(int))**

**emission\_counts = defaultdict(lambda: defaultdict(int))**

**tag\_counts = defaultdict(int)**

**logging.info("Counting transitions and emissions...")**

**for i in range(1, len(corpus)):**

**word, tag = corpus[i]**

**prev\_word, prev\_tag = corpus[i - 1]**

**emission\_counts[tag][word] += 1**

**transition\_counts[prev\_tag][tag] += 1**

**tag\_counts[tag] += 1**

**tag\_counts[prev\_tag] += 1**

**transition\_probs = defaultdict(dict)**

**logging.info("Calculating transition probabilities...")**

**for prev\_tag in tags:**

**for tag in tags:**

**transition\_probs[prev\_tag][tag] = (transition\_counts[prev\_tag][tag] / tag\_counts[prev\_tag]) if tag\_counts[prev\_tag] > 0 else 0**

**emission\_probs = defaultdict(dict)**

**logging.info("Calculating emission probabilities...")**

**for tag in tags:**

**for word in words:**

**emission\_probs[tag][word] = (emission\_counts[tag][word] / tag\_counts[tag]) if tag\_counts[tag] > 0 else 0**

**print("Transition Probabilities:")**

**for prev\_tag in tags:**

**print(f"{prev\_tag}: {transition\_probs[prev\_tag]}")**

**print("\nEmission Probabilities:")**

**for tag in tags:**

**print(f"{tag}: {emission\_probs[tag]}")**

**observations = ['dog', 'barks', 'loudly']**

**states = tags**

**def viterbi(observations, states, transition\_probs, emission\_probs):**

**logging.info("Running Viterbi Algorithm...")**

**V = np.zeros((len(states), len(observations)))**

**backpointer = np.zeros((len(states), len(observations)), dtype=int)**

**logging.info(f"Initializing Viterbi for first word: {observations[0]}")**

**for s, state in enumerate(states):**

**V[s][0] = emission\_probs[state].get(observations[0], 0) \* 1**

**logging.info(f"V[0][{s}] (state: {state}) = {V[s][0]}")**

**for t in range(1, len(observations)):**

**logging.info(f"Processing word: {observations[t]}")**

**for s, state in enumerate(states):**

**max\_prob = -1**

**best\_state = None**

**for prev\_s, prev\_state in enumerate(states):**

**prob = V[prev\_s][t-1] \* transition\_probs[prev\_state].get(state, 0) \* emission\_probs[state].get(observations[t], 0)**

**if prob > max\_prob:**

**max\_prob = prob**

**best\_state = prev\_s**

**V[s][t] = max\_prob**

**backpointer[s][t] = best\_state**

**logging.info(f"V[{s}][{t}] (state: {state}) = {V[s][t]}, backpointer: {best\_state}")**

**best\_path\_prob = max(V[s][-1] for s in range(len(states)))**

**best\_last\_state = np.argmax(V[:, -1])**

**best\_path = [None] \* len(observations)**

**best\_path[-1] = states[best\_last\_state]**

**logging.info(f"Backtracking from last state: {best\_last\_state} (POS: {states[best\_last\_state]})")**

**for t in range(len(observations) - 2, -1, -1):**

**best\_path[t] = states[backpointer[best\_last\_state][t + 1]]**

**best\_last\_state = backpointer[best\_last\_state][t + 1]**

**logging.info(f"Backtracking to state: {best\_last\_state} (POS: {states[best\_last\_state]})")**

**return best\_path, best\_path\_prob**

**best\_tags, best\_prob = viterbi(observations, states, transition\_probs, emission\_probs)**

**print("\nBest POS Tags for Sentence:", observations)**

**print("POS Tags:", best\_tags)**

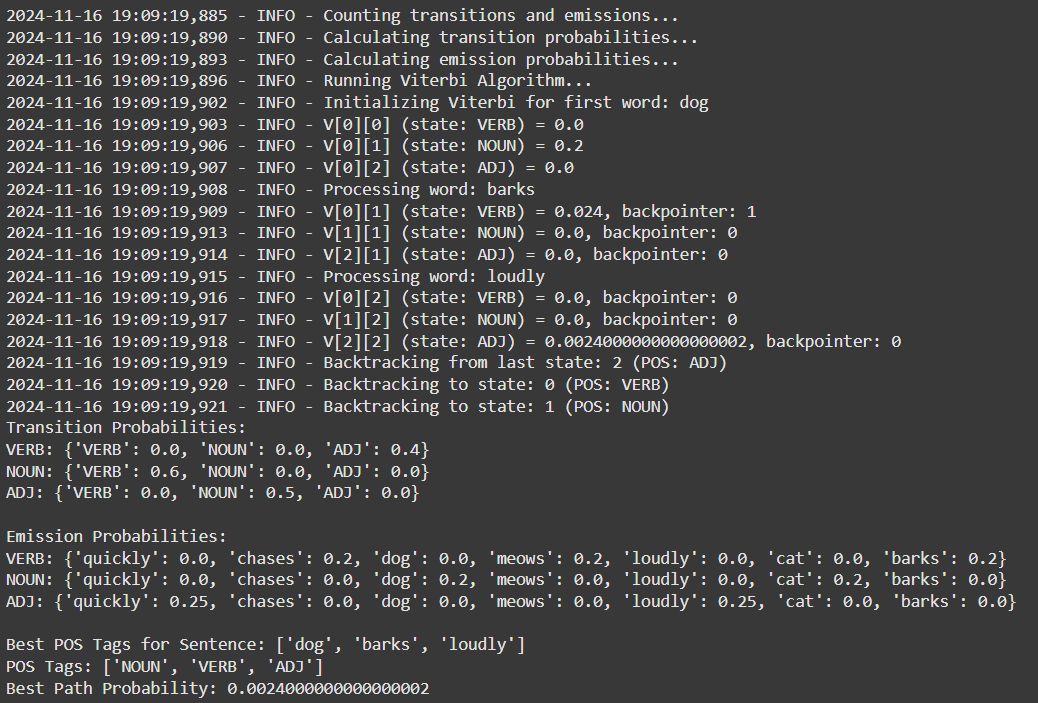
**print("Best Path Probability:", best\_prob)**

## Input:

corpus = [('dog', 'NOUN'), ('barks', 'VERB'), ('loudly', 'ADJ'), ('cat', 'NOUN'), ('meows', 'VERB'), ('quickly', 'ADJ'), ('dog', 'NOUN'), ('chases', 'VERB')]

observations = ['dog', 'barks', 'loudly']

## Output:

****

## Conclusion:

* The model applies the Viterbi algorithm to optimize POS tagging by estimating the most probable tag sequence.
* It efficiently handles word sequences through recursive probability calculations.
* The model calculates sequence likelihood, aiding in accurate POS tagging and evaluation.
* It encapsulates HMM-based POS tagging functionality, enabling potential extensions for tasks like sequence prediction and decoding.