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

Jan-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)))
```

```python
    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:

```
2024-11-16 19:09:19,885 - INFO - Counting transitions and emissions...
2024-11-16 19:09:19,890 - INFO - Calculating transition probabilities...
2024-11-16 19:09:19,893 - INFO - Calculating emission probabilities...
2024-11-16 19:09:19,896 - INFO - Running Viterbi Algorithm...
2024-11-16 19:09:19,902 - INFO - Initializing Viterbi for first word: dog
2024-11-16 19:09:19,903 - INFO - V[0][0] (state: VERB) = 0.0
2024-11-16 19:09:19,906 - INFO - V[0][1] (state: NOUN) = 0.2
2024-11-16 19:09:19,907 - INFO - V[0][2] (state: ADJ) = 0.0
2024-11-16 19:09:19,908 - INFO - Processing word: barks
2024-11-16 19:09:19,909 - INFO - V[0][1] (state: VERB) = 0.024, backpointer: 1
2024-11-16 19:09:19,913 - INFO - V[1][1] (state: NOUN) = 0.0, backpointer: 0
2024-11-16 19:09:19,914 - INFO - V[2][1] (state: ADJ) = 0.0, backpointer: 0
2024-11-16 19:09:19,915 - INFO - Processing word: loudly
2024-11-16 19:09:19,916 - INFO - V[0][2] (state: VERB) = 0.0, backpointer: 0
2024-11-16 19:09:19,917 - INFO - V[1][2] (state: NOUN) = 0.0, backpointer: 0
2024-11-16 19:09:19,918 - INFO - V[2][2] (state: ADJ) = 0.0024000000000000002, backpointer: 0
2024-11-16 19:09:19,919 - INFO - Backtracking from last state: 2 (POS: ADJ)
2024-11-16 19:09:19,920 - INFO - Backtracking to state: 0 (POS: VERB)
2024-11-16 19:09:19,921 - INFO - Backtracking to state: 1 (POS: NOUN)
Transition Probabilities:
VERB: {'VERB': 0.0, 'NOUN': 0.0, 'ADJ': 0.4}
NOUN: {'VERB': 0.6, 'NOUN': 0.0, 'ADJ': 0.0}
ADJ: {'VERB': 0.0, 'NOUN': 0.5, 'ADJ': 0.0}

Emission Probabilities:
VERB: {'quickly': 0.0, 'chases': 0.2, 'dog': 0.0, 'meows': 0.2, 'loudly': 0.0, 'cat': 0.0, 'barks': 0.2}
NOUN: {'quickly': 0.0, 'chases': 0.0, 'dog': 0.2, 'meows': 0.0, 'loudly': 0.0, 'cat': 0.2, 'barks': 0.0}
ADJ: {'quickly': 0.25, 'chases': 0.0, 'dog': 0.0, 'meows': 0.0, 'loudly': 0.25, 'cat': 0.0, 'barks': 0.0}

Best POS Tags for Sentence: ['dog', 'barks', 'loudly']
POS Tags: ['NOUN', 'VERB', 'ADJ']
Best Path Probability: 0.0024000000000000002
```

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.