**Natural Language Understanding (CSL7640)**
**Assignment 2**

**Submitted By:**

Pranjal Malik (M24CSA021)
Shivani Tiwari (M24CSA029)
Suvigya Sharma (M24CSA033)

**Submitted To:**
Dr. Asif Ekbal

**Named Entity Recognition (NER) using HMM-based Model**

1. **Introduction**

Named Entity Recognition (NER) is a crucial task in Natural Language Processing that involves identifying and classifying named entities in text. The goal is to label words in a sentence with predefined categories such as Person (PER), Location (LOC), Organization (ORG), and Miscellaneous (MISC).

In this project, we implement a Hidden Markov Model (HMM)-based NER system using a Twitter dataset. Two configurations are explored:

1. Bigram HMM Model: Transition probabilities depend only on the previous tag.
2. Trigram HMM Model: Transition probabilities depend on the previous two tags.

Additionally, an emission probability with context approach is explored, which conditions word emissions on the previous word tag.

**2. Dataset Preprocessing**

The dataset consists of Twitter texts annotated with named entity tags. The following preprocessing steps are applied:

- User mentions (@username) are replaced with @USER.
- URLs are replaced with URL.
- Repeated punctuation marks are normalized (e.g., !!! → !).
- Multiple spaces are collapsed into single spaces.
- Words are converted to lowercase for consistency.

The dataset is split into **training , validation, and testing sets**.

**3. Hidden Markov Model (HMM) Parameter Estimation**

To construct the HMM, the following parameters are estimated from the training data:

1. **Start Probability (π):** The probability of a tag occurring at the beginning of a sentence.
2. **Transition Probability (A):** The probability of transitioning from one tag to another.
3. **Emission Probability (B):** The probability of a word appearing given a specific tag.

**Model Configurations:**

- **Bigram HMM:** Uses P(tag_t | tag_(t-1)) for transitions.
- **Trigram HMM:** Uses P(tag_t | tag_(t-2), tag_(t-1)) for transitions.
- **Emission Probability with Context:** Uses P(word | tag, prev_tag).

**Laplace smoothing (k = 0.1)** is applied to avoid zero probabilities, and a **boost factor (0.75)** is used to enhance the probability of named entity tags.

### 4. Viterbi Algorithm for Decoding

The Viterbi Algorithm is a dynamic programming algorithm used to efficiently determine the most probable sequence of named entity tags for a given sentence. It maximizes the probability of the best possible tag sequence by systematically exploring all possible sequences while maintaining computational efficiency.

#### Steps of the Viterbi Algorithm

The algorithm consists of three main steps:

1. **Initialization:**
    - The probability of each tag for the first word is calculated using the start probability ($\pi$) and the emission probability (B).
    - These probabilities are stored in a dynamic programming table, along with the most probable previous state.
2. **Recursion:**
    - For each subsequent word, the probability of being in a given tag state is computed based on:
        - The **maximum probability path** leading to the current tag.
        - The **transition probability (A)** from the previous tag.
        - The **emission probability (B)** of the current word given the tag.
    - A backpointer table is maintained to track the most probable tag sequence.
3. **Termination and Backtracking:**
    - The tag with the highest probability at the last word is selected.
    - The best tag sequence is reconstructed by backtracking through the stored path.

### 5. Model Evaluation

The models are evaluated on validation data for tuning hyperparameters and on test data for final performance assessment.

#### 5.1 Accuracy Measurement

- **Overall Accuracy** = (Number of correctly predicted tags) / (Total number of tags)

### 5.2 Per-Class Performance Metrics

For each entity type, we compute:

- **Precision** = TP / (TP + FP)
- **Recall** = TP / (TP + FN)
- **F1-score** = 2 × (Precision × Recall) / (Precision + Recall)

## 6. Experimental Results and Observations

### 6.1 Test Data Performance

#### 6.1.1 Without Context

| Model | Accuracy | Macro Precision | Macro Recall | Macro F1-score | Micro Precision | Micro Recall | Micro F1-score |
|---|---|---|---|---|---|---|---|
| Bigram HMM | 90.87% | 0.3056 | 0.0970 | 0.1224 | 0.9087 | 0.9087 | 0.9087 |
| Trigram HMM | 81.94% | 0.1731 | 0.1262 | 0.1348 | 0.7814 | 0.7814 | 0.7814 |

#### 6.1.2 With Context

| Model | Accuracy | Macro Precision | Macro Recall | Macro F1-score | Micro Precision | Micro Recall | Micro F1-score |
|---|---|---|---|---|---|---|---|
| Bigram HMM | 90.38% | 0.0430 | 0.0476 | 0.0452 | 0.9038 | 0.9038 | 0.9038 |
| Trigram HMM | 89.02 | 0.0428 | 0.0475 | 0.0451 | 0.8902 | 0.8902 | 0.8902 |

## 7. Key Observations

1. Bigram HMM consistently achieves higher accuracy than Trigram HMM, both with and without context.
2. Without context, the Bigram model significantly outperforms the Trigram model, with a large gap in accuracy
3. With context, the gap between Bigram and Trigram models narrows, indicating that contextual emission probabilities improve the Trigram model's performance.
4. Macro Precision, Recall, and F1-score are much lower than Micro scores, suggesting an imbalance in the named entity categories.
5. The Trigram model benefits more from contextual emissions, increasing its accuracy.

## 8. Conclusion

This study successfully implemented an HMM-based Named Entity Recognition system and demonstrated that:

- The Bigram model performs better overall, but the Trigram model improves with context-aware emissions.
- Context-aware emission probabilities enhance entity recognition, improving classification accuracy.
- Laplace smoothing and entity-specific probability boosts mitigate sparsity issues.