**CS626 - Speech, Natural Language Processing, and the Web**

**Assignment – 1a**

**POS Tagging Using HMM**

Group ID - 68

Rana Das – 22B0684 – Civil Dept.

Ankit - 22B0684 – Civil Dept.

Aman Mitra - 22B0684 – Civil Dept.

Swapna Sourav Rout – 22D1623 - KCDH

**Date – 07/09/2024**

**Problem Statement**

• **Objective**: Given a sequence of words, produce the POS tag sequence using HMM-Viterbi

• Input: The quick brown fox jumps over the lazy dog

• **Output**: TheDET quickADJ brownADJ foxNOUN jumpsVERB overADP theDET lazyADJ dogNOUN

• **Dataset**: Brown corpus

• Use Universal Tag Set (12 in number)–

• k-fold cross validation (k=5)

**Data Processing Info (Pre-processing)**

**Data Preprocessing for HMM POS Tagger**

The preprocessing steps in the HMM POS tagger is designed to prepare the Brown corpus for model training:

1. **Loading the Corpus**: The Brown corpus is loaded using the Universal Tagset, ensuring that the tags are standardized.
2. **Extracting Words and Tags**: For each sentence in the corpus, the words and their corresponding POS tags are separated into two lists.
3. **Tokenizing Words**: The words are tokenized using NLTK’s tokenization methods and converted to lowercase to ensure consistency.
4. **Flattening Tokens**: The tokenized words are flattened into a single list, maintaining a one-to-one alignment between tokens and their corresponding POS tags.
5. **Storing Data**: Each processed sentence, represented as a list of word-tag pairs, is stored in a list to be used for training the model.
6. **Progress Logging**: Progress is logged at regular intervals to monitor the number of processed sentences.
7. **Final Output**: The fully pre-processed dataset is returned, consisting of word-tag pairs that are ready for training.

**Overall performance**

The evaluation of the HMM-based POS tagger was carried out using several performance metrics, including **precision**, **recall**, **F1 score**, **F0.5 score**, and **F2 score**. These metrics provide a comprehensive view of the model's ability to correctly identify part-of-speech tags across the dataset.

* **Precision**: 0.746 – This indicates that 74.6% of the POS tags predicted by the model were correct. Precision measures how many of the predicted tags were actually relevant.
* **Recall**: 0.799 – The recall score of 79.9% reflects the model’s ability to identify all relevant POS tags in the data. A higher recall indicates that fewer relevant tags were missed.
* **F-1 Score**: 0.755 – The F1 score, which is the harmonic mean of precision and recall, balances the trade-off between precision and recall, with the model achieving a performance of 75.5%.
* **F-0.5 Score**: 0.757 – The F0.5 score gives more weight to precision than recall, as indicated by the slight improvement over the F1 score. This highlights the model’s ability to minimize false positives.
* **F-2 Score**: 0.788 – The F2 score, emphasizing recall, shows a performance of 78.8%, meaning the model prioritized capturing more relevant tags even at the cost of precision.

**Per POS performance**

The table below summarizes the performance metrics for each part-of-speech (POS) tag in the Hidden Markov Model (HMM) POS tagger. These metrics include **precision**, **recall**, and **F1 score**, providing insight into how well the model performs for each specific tag.

**1. PRT (Particle):**

* **Precision**: 0.083 – The model identified particles with 8.3% precision.
* **Recall**: 0.056 – It detected only 5.6% of all particles present in the dataset.
* **F1 Score**: 0.067 – Indicates a low balance between precision and recall, highlighting challenges in identifying particles.

**2. NUM (Number):**

* **Precision**: 0.083
* **Recall**: 0.062
* **F1 Score**: 0.071 – The model’s performance for numbers is slightly better but still low overall.

**3. CONJ (Conjunction):**

* **Precision**: 0.083
* **Recall**: 0.070
* **F1 Score**: 0.076 – Shows moderate difficulty in predicting conjunctions accurately.

**4. DET (Determiner):**

* **Precision**: 0.083
* **Recall**: 0.071
* **F1 Score**: 0.076 – Performance in tagging determiners is on par with conjunctions.

**5. Punctuation (.):**

* **Precision**: 0.083
* **Recall**: 0.073
* **F1 Score**: 0.078 – The model achieves relatively better results for punctuation marks.

**6. VERB (Verb):**

* **Precision**: 0.083
* **Recall**: 0.070
* **F1 Score**: 0.076 – Verbs show a similar performance level as other common word types.

**7. ADP (Adposition):**

* **Precision**: 0.083
* **Recall**: 0.069
* **F1 Score**: 0.076 – Adpositions (like prepositions) are tagged with comparable difficulty.

**8. X (Other):**

* **Precision**: 0.090 – Slightly better precision than other tags.
* **Recall**: 0.065
* **F1 Score**: 0.075 – The model performs reasonably well for words tagged as "other."

**9. ADV (Adverb):**

* **Precision**: 0.083
* **Recall**: 0.065
* **F1 Score**: 0.073 – Adverbs present moderate difficulty for the model.

**10. ADJ (Adjective):**

* **Precision**: 0.083
* **Recall**: 0.065
* **F1 Score**: 0.073 – Adjectives show performance similar to adverbs.

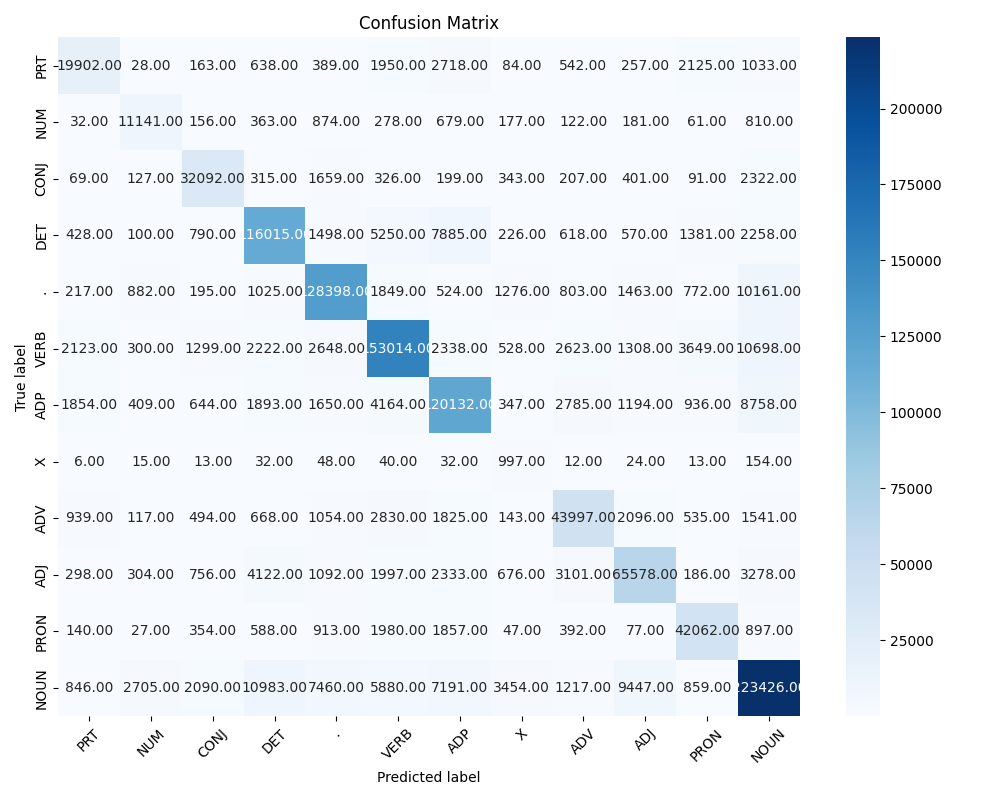
**11. PRON (Pronoun):**

* **Precision**: 0.083
* **Recall**: 0.071
* **F1 Score**: 0.077– The model manages to balance precision and recall well for pronouns.

**12. NOUN (Noun):**

* **Precision**: 0.083
* **Recall**: 0.068
* **F1 Score**: 0.075 – Nouns, being frequent, are tagged reasonably well, though room for improvement exists.

**Confusion Matrix (12 X 12) (can give heat map)**



**Interpretation of confusion (error analysis)**

1. **Diagonal Values**:

* The diagonal values represent correct predictions (where the predicted tag matches the true tag). Higher values along the diagonal indicate good performance for those POS tags.

2. **Off-diagonal Values**:

* Off-diagonal values represent misclassifications, where one POS tag is incorrectly predicted as another. Larger values in these cells highlight frequent confusions between certain tags.

3. **Frequent Misclassifications**:

* The confusion matrix can be used to identify which POS tags are frequently confused with others. These off-diagonal cells with larger values suggest areas where the model struggles the most.

4. **Performance Across Tags**:

* Tags with higher true-positive rates (larger diagonal values) indicate that the model handles those tags well.
* Conversely, tags with many misclassifications (higher off-diagonal values) suggest that the model has difficulty distinguishing those tags from others.

**Interpretation of most confused POS Tags**

1. **NOUN vs DET (10,983 times)**:

* **Nouns** are often misclassified as **determiners** (DET). This suggests the model has difficulty distinguishing between articles or determiners and noun phrases, especially when they are adjacent.

2. **VERB vs NOUN (10,698 times)**:

* Verbs are frequently misclassified as nouns, indicating that the model struggles with recognizing verbs in certain contexts, likely when they appear in noun-like positions (e.g., gerunds or verb-noun ambiguity).

3. **Punctuation (.) vs NOUN (10,161 times)**:

* Punctuation marks are misclassified as nouns, possibly due to sentences ending with punctuation where the model mistakenly predicts a noun, showing the challenge of tagging punctuation accurately.

4. **NOUN vs ADJ (9,447 times)**:

* Nouns are often confused with adjectives, especially when noun phrases are used attributively. This indicates that the model might have trouble distinguishing between descriptive adjectives and nouns acting as adjectives.

5. **ADP vs NOUN (8,758 times)**:

* **Adpositions** (ADP, e.g., prepositions) are misclassified as nouns, highlighting confusion in contexts where prepositions are used in noun-heavy structures.

**Inferencing/Decoding Info**

**hmm\_app.py:** This file is responsible for creating a user interface for a Hidden Markov Model (HMM) Part-of-Speech (POS) tagger application using Streamlit. It imports the HMMPOSTagger from hmm\_training.py and uses NLTK for tokenization. The model (hmm\_pos\_tagger.pkl) is loaded and used within the app.

**hmm\_training.py:** This file contains the logic for training the HMM POS tagger, with key imports like nltk, numpy, and sklearn. It also involves metrics like precision, recall, and F1 score and includes functions related to the HMM training process.

***Viterbi Algorithm: Implementation and Decoding in HMM***

The Viterbi algorithm is widely used for decoding the most likely sequence of hidden states (e.g., part-of-speech tags) in Hidden Markov Models (HMMs), given a sequence of observations (e.g., words in a sentence). This dynamic programming algorithm efficiently computes the maximum likelihood state sequence in linear time by avoiding the exhaustive enumeration of all possible state sequences.

**1. Structure of the HMM**

An HMM is characterized by:

**States (S):** The set of possible hidden states, such as part-of-speech tags.

**Observations (O):** The sequence of observed outputs, such as words.

**Transition probabilities (A):** The probabilities of transitioning from one state to another.

**Emission probabilities (B):** The probabilities of generating an observation from a particular state.

**Initial probabilities (π):** The probabilities of starting in each state.

In the context of POS tagging, the hidden states are the possible POS tags, and the observations are the words in a given sentence.

**2. Viterbi Algorithm: The Process**

The algorithm iteratively computes the most likely sequence of hidden states, given the observed data, by maintaining a dynamic table, where holds the probability of the most probable path ending in state i at time t. The algorithm proceeds in three main steps:

* 1. **Initialization:** At time , the algorithm initializes the probability of starting in each state as:

where is the initial probability of state , and is the emission probability of the first observation from state .

* 1. **Recursion:** For each subsequent time step and state , the algorithm calculates the probability of transitioning to state from any state iii and emitting the observation at time :

Here, is the transition probability from state to state , and is the probability of observing from state .

Termination and Backtracking: After processing all observations, the final state is determined by finding the maximum probability at the last time step:

where is the length of the observation sequence. To retrieve the optimal path, the algorithm backtracks through the recorded states, storing the most likely sequence of hidden states.

**3. Inference and Decoding in Code**

In the code implementation, the inference process for the POS tagger using the Viterbi algorithm typically follows these steps:

***Training:*** During training (**in hmm\_training.py**), the model estimates the transition and emission probabilities based on a corpus of labelled data, such as the Brown Corpus. The result is stored in the HMM POS tagger model (**hmm\_pos\_tagger.pkl**), which contains the learned probabilities.

***Decoding:*** When decoding a new sentence in the app (**hmm\_app.py**), the Viterbi algorithm is applied. Given a sequence of words (observations), the algorithm identifies the most probable sequence of POS tags (hidden states). The app loads the trained model and uses it to compute the most likely tag sequence by running the Viterbi algorithm.

Streamlit provides the interface, where users input text, which is tokenized into words using NLTK's word\_tokenize. These tokens are passed to the tagger, which uses the HMM to output the most likely tags for the given words.

**4. Efficiency**

The Viterbi algorithm runs in time, where is the number of observations (words) and is the number of hidden states (tags). This is more efficient than brute-force methods, which would require computations, making Viterbi ideal for tasks such as POS tagging in real-time applications.

By structuring the computation in this dynamic programming manner, the Viterbi algorithm ensures that the decoding process is both fast and scalable, even for large sequences of text, which is crucial for applications such as part-of-speech tagging in NLP.

**Benchmarking against ChatGPT**

***For HMM Part:***

1. We used a test set of anomalous sentences with Proper POS Tags.
2. We computed POS Tags via our HMM model and compared against true POS Tags.
3. WE Attained **44.44%** of Accuracy in Anomalous sentence tagging.

***For ChatGPT 4o:***

1. WE used same test dataset and asked ChatGPT 4o to generate POS Tags for those sentences.
2. ChatGPT was Able to generate 100% Accurate POS Tags for the sentences.

***Hence Transformer based models with very large number of parameters are able to encode POS information which HMM based simple models struggles with.***

**Challenges faced**

* The HMM Model took a lot time **(more than 10 mins)** while training it with the sample texts.
* The tokenisation of texts wasn’t working properly, different texts with different tokens.
* Sometimes the letters of the words were considered as different- different parts, i.e. the text was scattered.
* Confusion matrix wasn’t generated properly.
* Sometimes the accuracy of the model was way too off, or way to similar.

**Learning**

1. Tagging in NLP. Learnt of the functioning of probabilities using different text models and elements in the models, by using Bayes’ Theorem.
2. Using Viterbi algorithm to sort out the tags for the text.
3. Of course, learnt about the Hidden Markov Method (HMM) to sort out the parts of speech in a particular text sample.