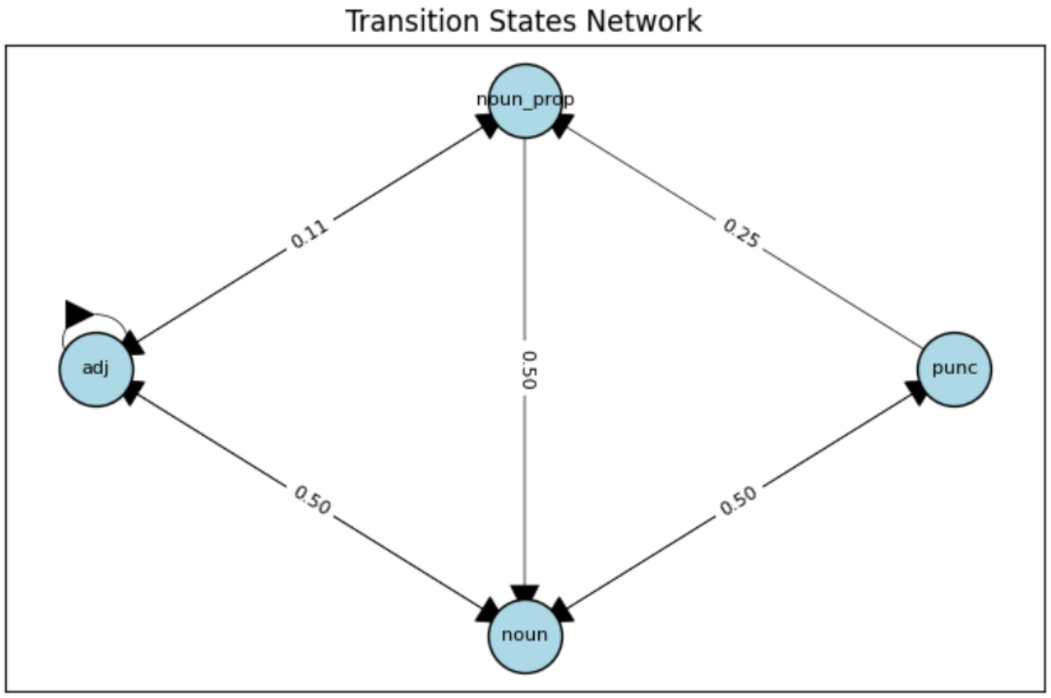
Arabic Part Of Speech Tagging with Network-X in Articles



Contents

Table Of Figures

1.0 Introduction

First we want to know what is Arabic Part-of-Speech (POS) tagging!

Arabic Part-of-Speech (POS) tagging is the process of labeling each word in an Arabic sentence with its corresponding grammatical category, such as noun, verb, adjective, etc. Due to the rich morphological structure of Arabic, POS tagging is more complex compared to languages like English. Arabic has various forms of a word depending on gender, number, case, and tense, which makes it challenging for traditional methods specifically in Arabic Language

Part of Speech (POS) tagging is important because it helps in understanding the grammatical structure of a sentence. This understanding is crucial for various natural language processing tasks, including:

- Syntactic Parsing: Identifying the relationships between words in a sentence.
- Machine Translation: Improving translation accuracy by understanding word functions..
- Sentiment Analysis: Determining the sentiment by analyzing verbs and adjectives

In this project, we aim to develop a pipeline for Part of Speech (POS) tagging that can be visualized as a network graph. The primary objective is to preprocess Arabic text articles, optimize the representation of each word,
and optionally explore deep learning methods to compare their effectiveness.

2.0 Data Description

2.1 Pipe Line Dataset

I used a corpus of Arabic text articles that is scraped from Youm7 website. The corpus contains 341 sentences of Arabic sentences. Additionally, to further test our pipeline, we generated 5 new sentences using ChatGPT, based on the content of these articles. This provided a comprehensive dataset to evaluate the performance and robustness of our POS tagging pipeline.

2.1.1 Pipe Line Dataset Statistics

- **Total Articles:** One article(contains 341 Arabic sentences) .
- **Total Sentences:** 341 sentences + 5 sentences for testing pipeline.
- **Total Tags:**341 list of tags for each sentence (created using Camel Tool Pos Tagger).

2.1.2 Data Preprocessing

We perform several preprocessing steps to normalize the text before passing it to our algorithm for graph creation.
The preprocessing steps are as follows:

1. **Remove Tashkeel (Diacritics)**: Stripping diacritical marks to simplify the text.
2. **Remove Tatweel**: Eliminating elongated characters.
3. **Normalize Alef**: Standardizing different forms of the letter Alef.
4. **Normalize Hamza**: Unifying the representation of Hamza.
5. **Normalize Ya**: Standardizing different forms of the letter Ya.
6. **Normalize Ta Marbutah**: Converting Ta Marbutah to its standard form.
7. **Remove Emojis**: Eliminating any emojis present in the text.

After normalization, we split the articles into sentences. These sentences are then tokenized using the Camel Tools Tokenizer. Each token is assigned a POS tag using the Camel Tools Arabic Tagger model. This ensures that the text is uniformly processed and ready for the subsequent steps in our pipeline.

2.2 Arabic POS-Tagger Model Data Source

The data was collected from Kaggle Dataset for the purpose of sentiment analysis in order to produce a score for a given company. The data has 40K+ reviews in Arabic for sentiment analysis each labelled with a rating and its associated company name.

2.2.1 Dataset Statistics

- **Training Set Size:** 30,733 sentences
- **Validation Set Size:** 5,762 sentences
- **Test Set Size:** 1,922 sentences

2.2.2 Dataset Features

- Index column (integer)
- Review (text)
- Rating (-1,0,1)
- Company (text)

2.2.3 Companies Reviews counts in Data sets

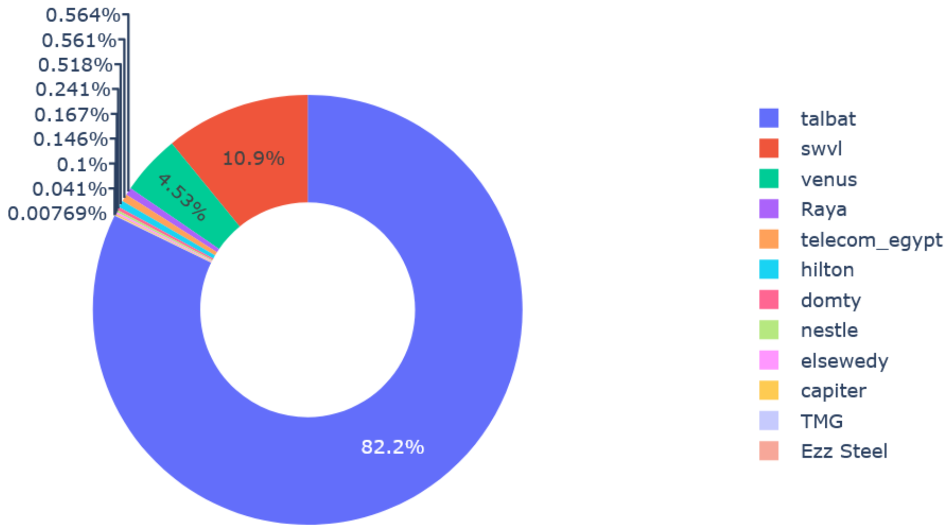


Figure 1 Arabic companies reviews number distribution

2.2.4 Data Preprocessing

The dataset initially contained **1,042** duplicated reviews, which were removed to ensure data integrity. We also checked for null reviews, but none were found. Additionally, we removed any samples with more than 60 tokens. This decision was based on the distribution of the dataset, which showed that the majority of examples contained between **2** and **60** tokens. The following graph illustrates this distribution, highlighting the prevalence of examples within this token range.

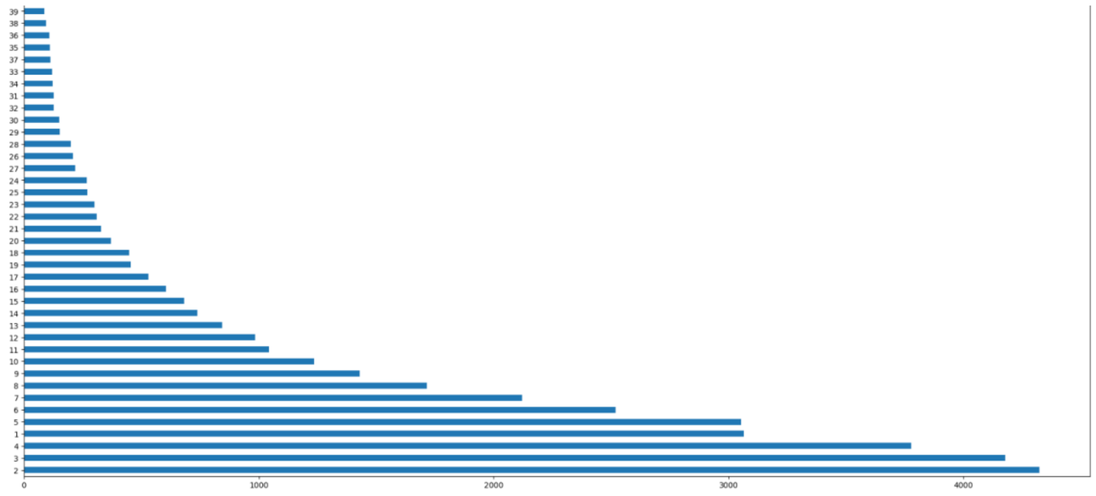
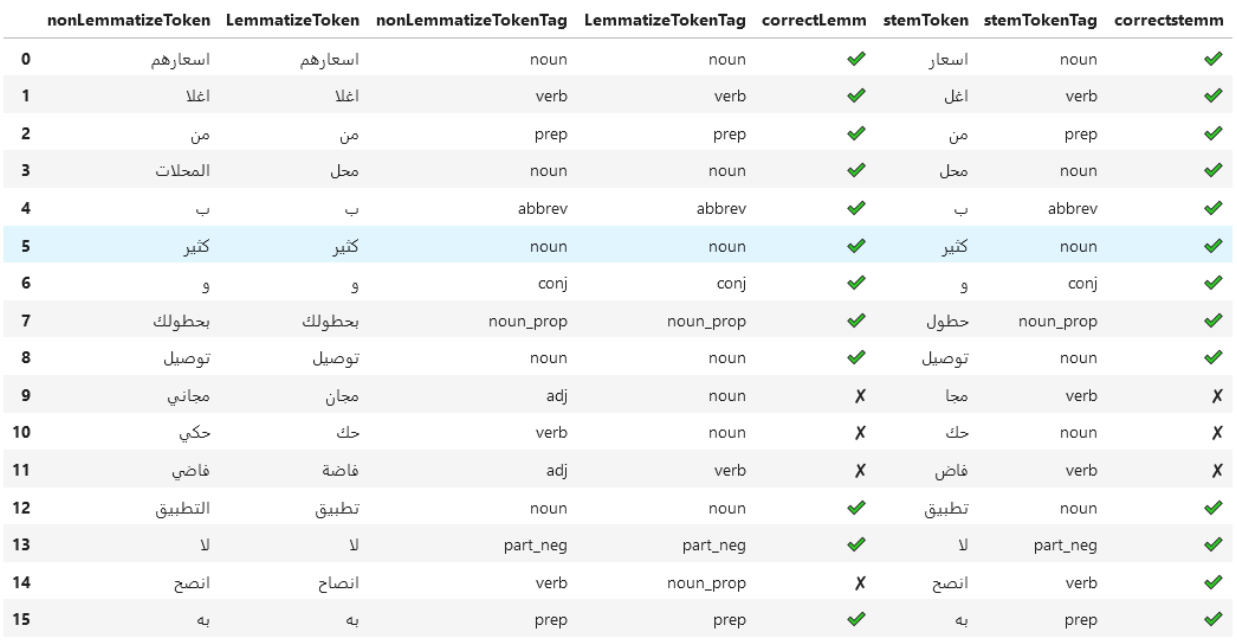


Figure 2 Distribution for tokens lengths and there counts in Data

Then we normalized the revies as the following:

1. **Remove Tashkeel (Diacritics)**: Stripping diacritical marks to simplify the text.
2. **Remove Tatweel**: Eliminating elongated characters.
3. **Normalize Alef**: Standardizing different forms of the letter Alef.
4. **Normalize Hamza**: Unifying the representation of Hamza.
5. **Normalize Ya**: Standardizing different forms of the letter Ya.
6. **Normalize Ta Marbuta**: Converting Ta Marbuta to its standard form.
7. **Remove Emojis**: Eliminating any emojis present in the text.

We initially considered applying lemmatization and stemming to the dataset. However, after testing these techniques on sample data, we observed significant changes in the POS tags for certain tokens. For example, the word "**حكي**" (**VERB**) was transformed to "**حك**" (**NOUN**) after lemmatization/stemming. In Arabic, "**حكي**" is a verb, and altering it to "**حك**" changed its meaning and grammatical role, leading to inaccurate POS tagging.



	nonLemmatizeToken	LemmatizeToken	nonLemmatizeTokenTag	LemmatizeTokenTag	correctLemm	stemToken	stemTokenTag	correctstemm
0	اسعارهم	اسعارهم	noun	noun	✓	اسعار	noun	✓
1	اغلا	اغلا	verb	verb	✓	اغل	verb	✓
2	من	من	prep	prep	✓	من	prep	✓
3	المحلات	محل	noun	noun	✓	محل	noun	✓
4	ب	ب	abbrev	abbrev	✓	ب	abbrev	✓
5	كتير	كتير	noun	noun	✓	كتير	noun	✓
6	و	و	conj	conj	✓	و	conj	✓
7	بحطولك	بحطولك	noun_prop	noun_prop	✓	حطول	noun_prop	✓
8	توصيل	توصيل	noun	noun	✓	توصيل	noun	✓
9	مجاني	مجان	adj	noun	✗	مجا	verb	✗
10	حكي	حك	verb	noun	✗	حك	noun	✗
11	فاضي	فاضة	adj	verb	✗	فاض	verb	✗
12	التطبيق	تطبيق	noun	noun	✓	تطبيق	noun	✓
13	لا	لا	part_neg	part_neg	✓	لا	part_neg	✓
14	انصح	انصاح	verb	noun_prop	✗	انصح	verb	✓
15	به	به	prep	prep	✓	به	prep	✓

Figure 3 Checking Lemmatization/stemming Example tags

Due to this issue, we decided not to apply lemmatization or stemming, as it negatively impacted the accuracy of our POS tags. This decision ensures that the words retain their original forms and meanings, preserving the integrity of the POS tagging process.

2.2.5 Generate POS-Tags

I used the Camel Tools Arabic Tagger model to assign POS tags to each review in our dataset. After generating tags for the entire dataset, I found that the classes were imbalanced. And to address this issue, we decided to use the **F1 score Measure** to evaluate the performance of each model during training. The **F1 score** is particularly useful in this context as it put into consideration both precision and recall, So it provide a more comprehensive evaluation of model performance for imbalanced dataset. Below is the distribution of tag counts in the dataset, highlighting the imbalance among the classes.

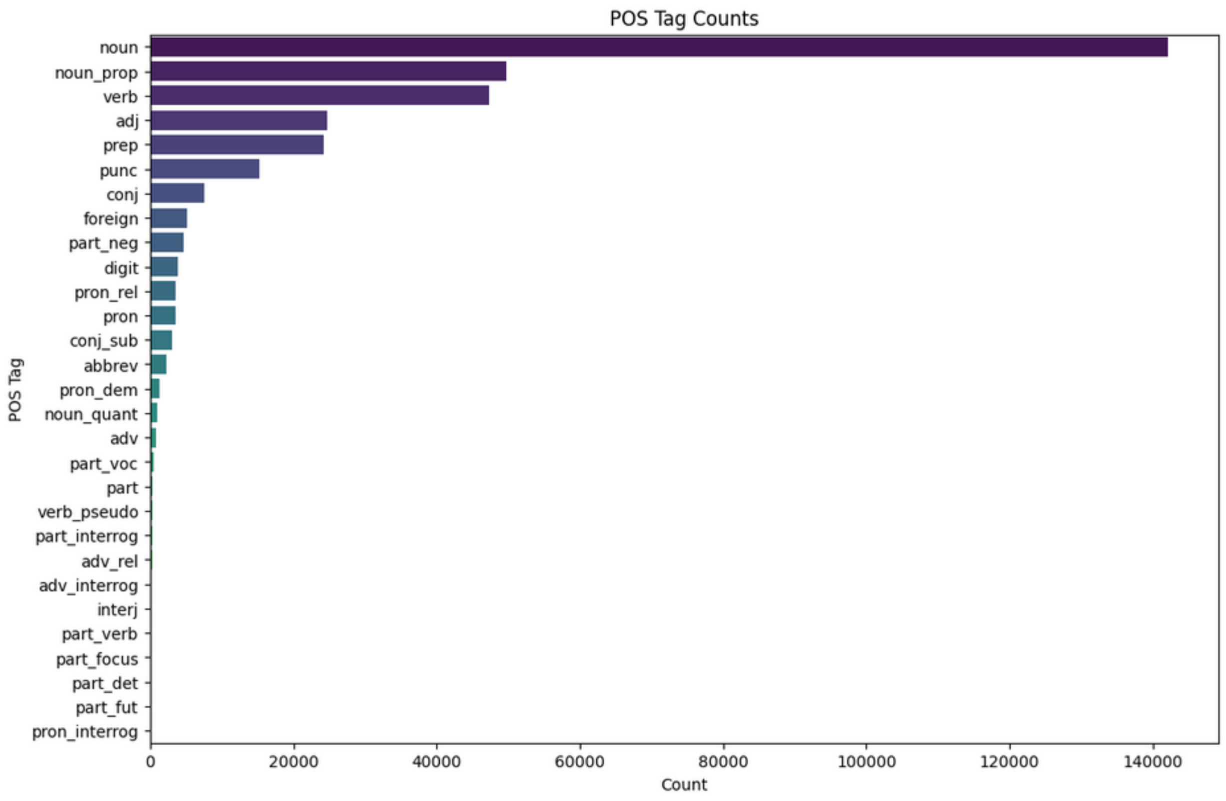


Figure 4 Pos-Tags Counts in Dataset

3.0 POS Tagging using Hidden Markov Models (HMM) & Viterbi algorithm

3.1 Hidden Markov Model (HMM)

HMM is a statistical model used to represent systems that are assumed to follow a Markov process with hidden states. It models a process where an observable sequence of data depends on underlying hidden states. In an HMM, there are two components:

- Hidden States: These are the actual states of the system that are not directly observable.
- Observations: The visible outputs or data that depend on the hidden states.

HMM is widely used in sequence modeling, such as speech recognition, part-of-speech tagging, and biological sequence analysis

3.1.1 Why Use HMM for POS Tagging?

In POS tagging, the words in a sentence can be considered as observable states (or observations) given to us in the data. Their corresponding POS tags, however, are hidden states. Hidden Markov Models (HMM) are used to estimate these hidden states (POS tags) based on the observable states (words). This allows us to effectively determine the POS tags for each word in a sentence using the probabilistic framework provided by HMM.

3.1.2 Hidden Markov Model components

Q: Set of Possible Tags

The set of possible tags represents all the different POS tags that can occur in the text.

A: Transition Probabilities

The A matrix contains the tag transition probabilities P(ti|ti−1)P(t_i | t_{i-1})P(ti|ti−1), which represent the likelihood of a tag occurring given the previous tag.

This probability tells us how often a noun follows a verb in the given text corpus.

O: Sequence of Observations

The sequence of observations refers to the words in a sentence that we are analyzing.

B: Emission Probabilities

The B matrix contains the emission probabilities P(wi|ti)P(w_i | t_i)P(wi|ti), which represent the probability of a given word occurring given a specific tag.

This probability tells us how often the word "Playing" is tagged as a verb in the corpus.

Note

All these counts (i.e., Count()\text{Count}()Count()) are derived from the corpus used for training the model.

3.1.3 Algorithm for Populating A, B, and π Matrices

3.1.3.1. Initialization

- **Define** the number of tags (`num_tags`) and words (`num_words`).
- **Create** the following matrices and vectors initialized to zero:
    - **A Matrix** (`num_tags x num_tags`): Transition count matrix.
    - **B Matrix** (`num_tags x num_words`): Emission count matrix.
    - **π Vector** (`num_tags`): Initial probabilities vector.
- **Define** dictionaries to map tags and words to their indices:
    - `tag2idx`: Maps each tag to a unique index.
    - `word2idx`: Maps each word to a unique index.

3.1.3.2. Iterate Through Sentences

- For each sentence in the corpus:
    - **Extract** the list of (word, tag) pairs.
- **Process Each Word-Tag Pair**:
        - **If the word is the first in the sentence**:
            - Increment the initial probability count for the tag.
        - **If the word is not the first in the sentence**:
            - Update the transition count from the previous tag to the current tag.
        - **Update the emission count** for the current tag and word.

3.1.3.3. Convert Counts to Probabilities

- **Normalize the A Matrix**:
    - Convert transition counts to probabilities by dividing each row by its sum.
- **Normalize the B Matrix**:
    - Convert emission counts to probabilities by dividing each row by its sum.

3.1.3 Transition state matrix

The transition matrix below represents the probabilities of transitioning from one part-of-speech (POS) tag to another based on the tag sequences found in the example sentences. This matrix is a crucial component in modeling tag sequences, such as in Hidden Markov Models (HMMs), which estimate the likelihood of tag sequences occurring given the observed words.

example:

"التكنولوجيا الحديثة مفيدة جداً. الأجهزة الجديدة مذهلة للغاية. البرامج التعليمية رائعة جداً. الإنترنت سريع جداً ومفيد للغاية. الهواتف الذكية مفيدة جداً."

Tags Founds: noun,adj,noun_prop,punc.

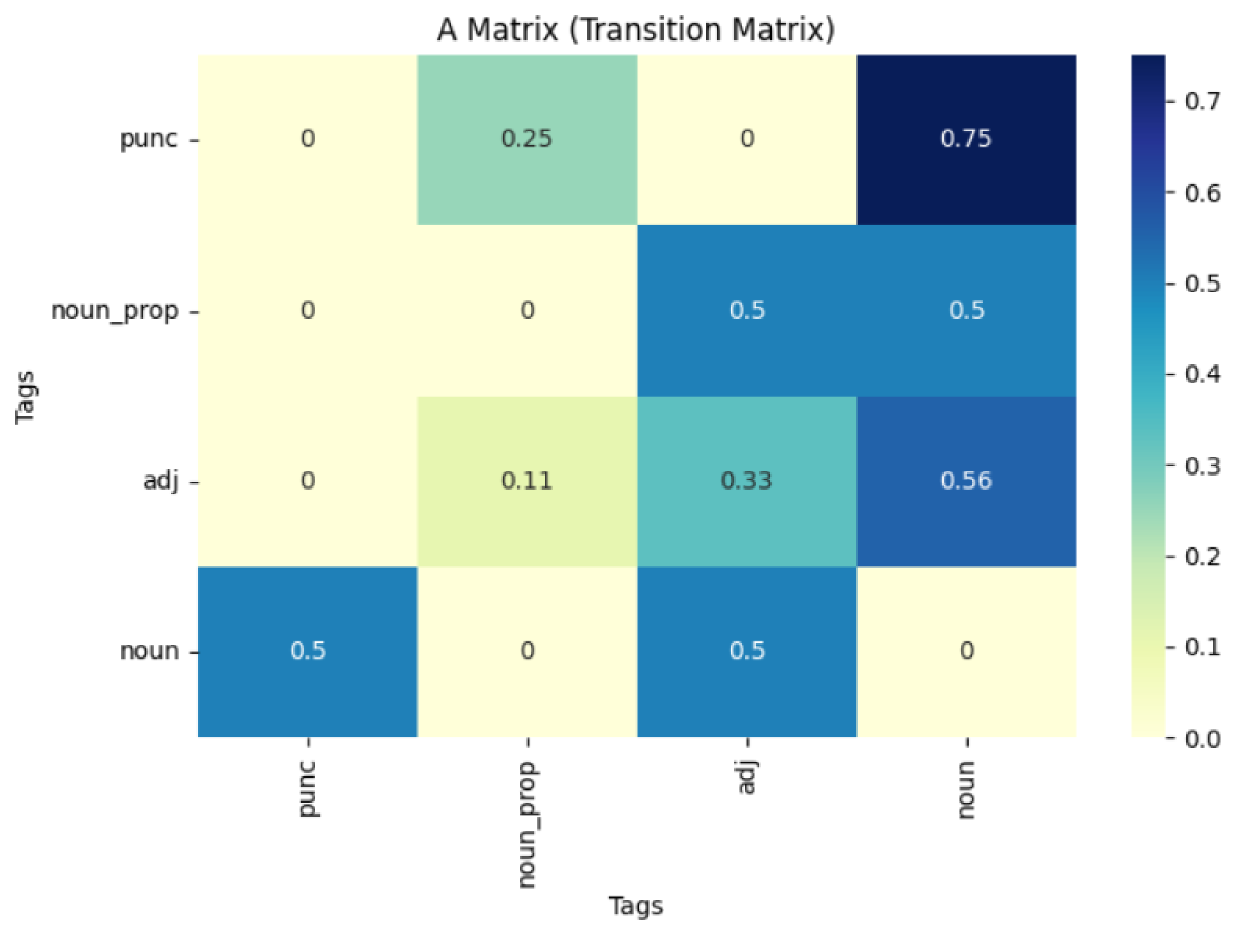


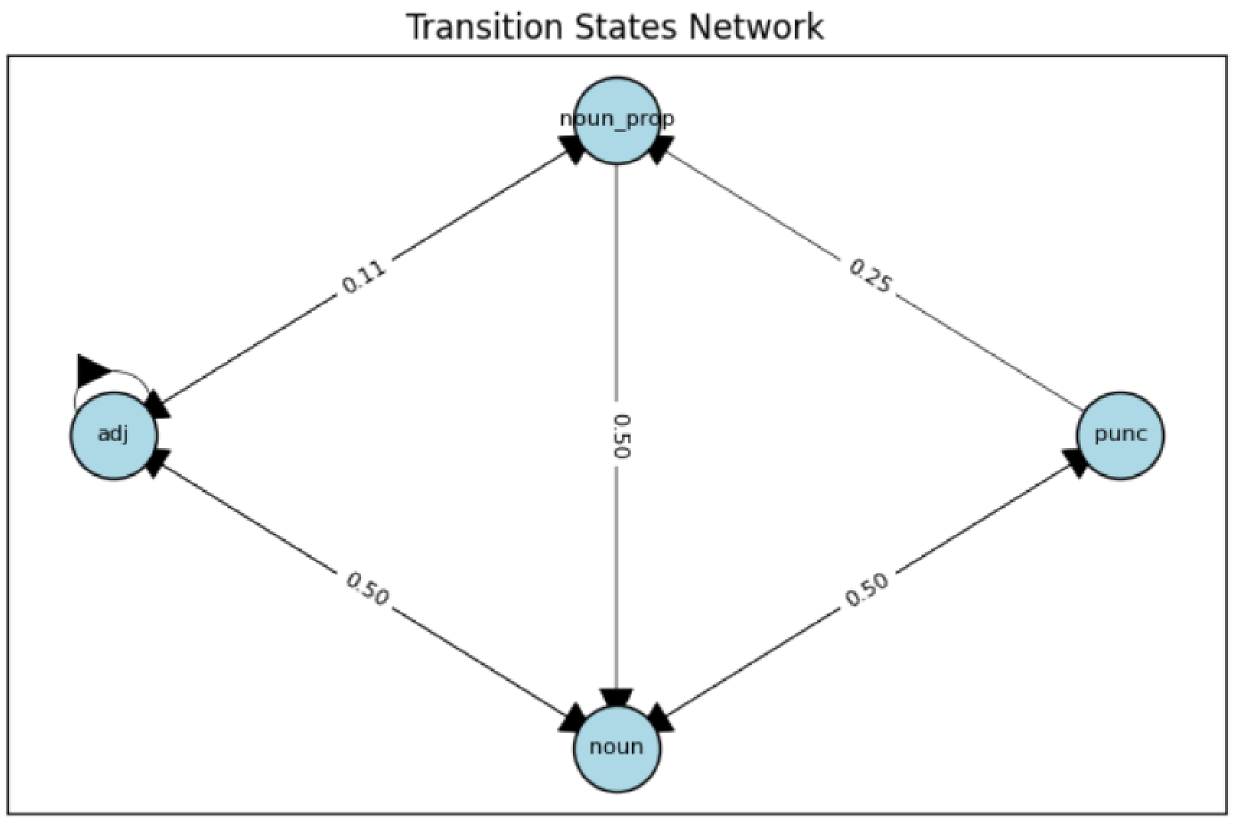
Figure 5 Transition State Matrix

3.1.4 Transition state diagram using NetworkX for Tag

The following transition state diagram illustrates the probabilities associated with each part-of-speech (POS) tag transitioning to another tag. This diagram is derived from the transition probability matrix A and provides a visual representation of the likelihood of one tag following another.

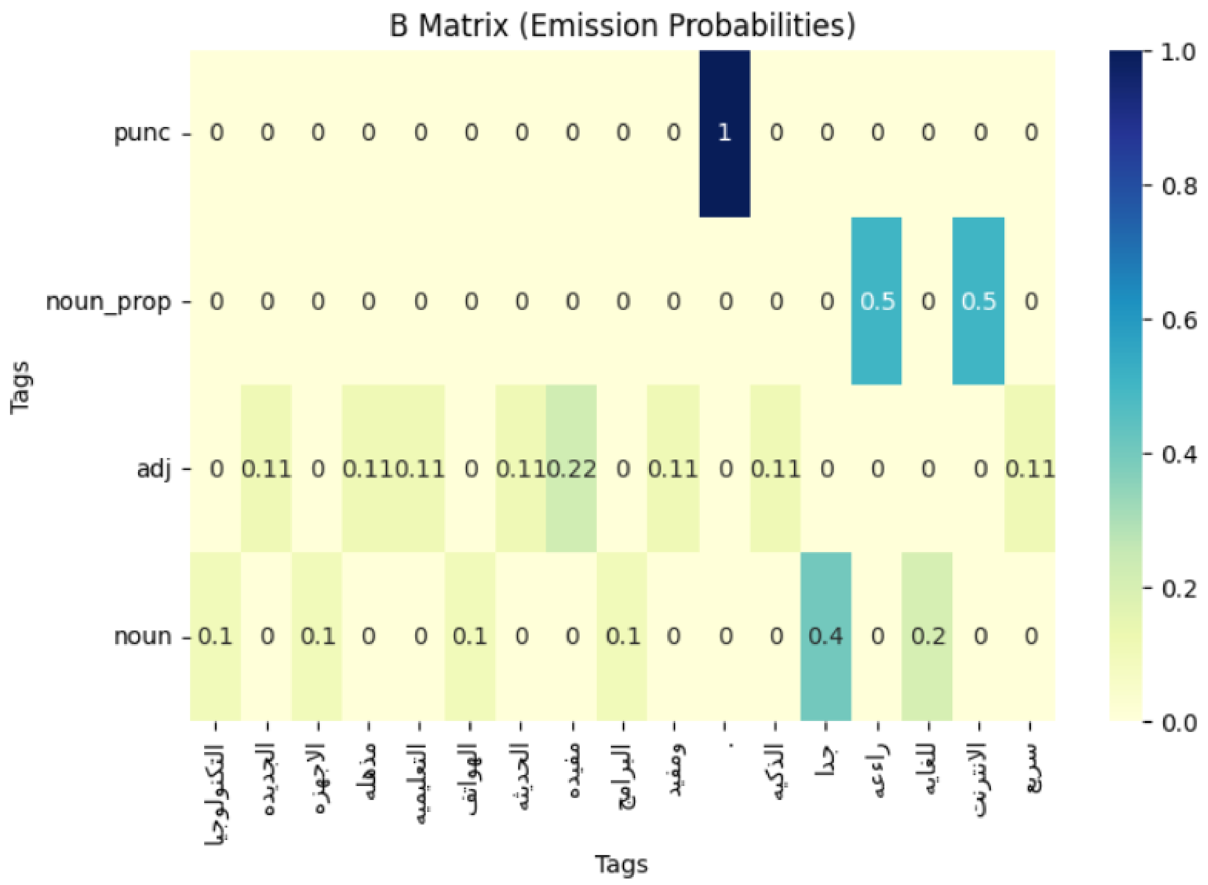
Diagram Overview

- **Nodes**: Represent the POS tags. In this diagram, we focus on four tags.
- **Edges**: Indicate the transition probabilities between tags. Each edge is labeled with the probability of transitioning from one tag to another, as specified in the matrix A.



3.1.5 Emission Probabilities Matrix

The emission matrix BBB captures the probabilities of words (observations) being associated with specific part-of-speech (POS) tags. It plays a crucial role in models like Hidden Markov Models (HMMs), where it helps determine the likelihood of a particular word being tagged with a specific POS label.



3.2 VITERBI ALGORITHM

The Viterbi algorithm is a dynamic programming approach used for decoding the most likely sequence of hidden states (such as part-of-speech (POS) tags) in Hidden Markov Models (HMMs). It is particularly valuable in natural language processing (NLP) for tasks such as POS tagging, where the goal is to determine the most probable sequence of tags for a given sequence of words.

3.2.1 How the Viterbi Algorithm Works

The Viterbi algorithm operates on the principle of maximizing the probability of a sequence of tags given an observed sequence of words. It does this by using a dynamic programming table to keep track of the highest probability of any tag sequence up to each word position in the sequence.

Steps of the Viterbi Algorithm:

1. **Initialization:**
    - Initialize the probability for the starting tag (often using the initial state probabilities π).
    - Compute the probabilities of the first word being tagged with each possible tag.
2. **Recursion:**
    - For each subsequent word, calculate the probability of each tag given all possible preceding tags. This involves:
        - Multiplying the transition probability of moving from a previous tag to the current tag.
        - Multiplying this by the emission probability of the current word given the current tag.
    - Store the maximum probability and the corresponding tag for each tag in a table.
3. **Termination:**
    - Identify the most probable tag sequence by backtracking through the table from the last word to the first.

3.2.2 Why Use the Viterbi Algorithm?

1. **Efficiency:**
   - The Viterbi algorithm efficiently handles the computational complexity of finding the most probable sequence of tags by breaking down the problem into manageable sub-problems. It avoids the exponential time complexity of a naive approach by storing intermediate results.
2. **Optimal Solution:**
   - By using dynamic programming, the algorithm ensures that the solution is optimal. It systematically builds up the solution by considering all possible tag sequences, ensuring that the final sequence is the most probable one.
3. **Handling Large Sequences:**
   - The algorithm is well-suited for sequences of varying lengths, making it adaptable to different sizes of text data in NLP tasks.
4. **Utilization of Transition and Emission Probabilities:**
   - It effectively incorporates both transition probabilities (likelihood of one tag following another) and emission probabilities (likelihood of a word given a tag) to determine the best sequence of tags.

3.3.3 Interpretation of lattice:

Each cell of the lattice is represented by **V_t(j)** (‘t’ represent column & j represents the row, called Viterbi path probability) representing the probability that the HMM is in ***state j(present POS Tag)*** after seeing the ***first t observations(past words for which lattice values has been calculated)*** and passing through the most **probable state sequence(previous POS Tag)** *q*_1…..*q*_*t*−1. Here is an example that shows lattice of the following example.

Article:

" التكنولوجيا الحديثة مفيدة جداً. الأجهزة الجديدة مذهلة للغاية. البرامج التعليمية رائعة جداً. الإنترنت سريع جداً ومفيد للغاية. الهواتف الذكية مفيدة جداً."

Example: الأجهزة الجديدة مذهلة جداً

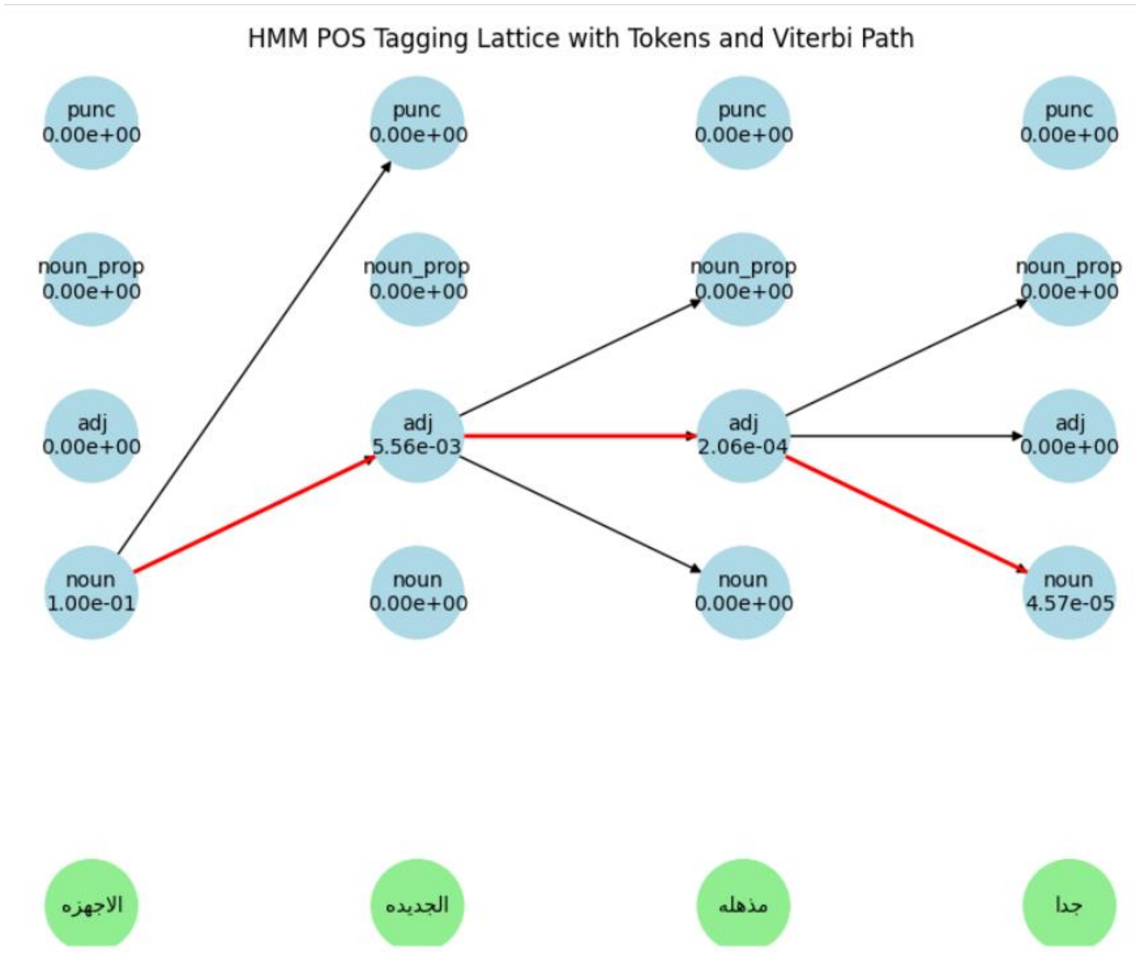


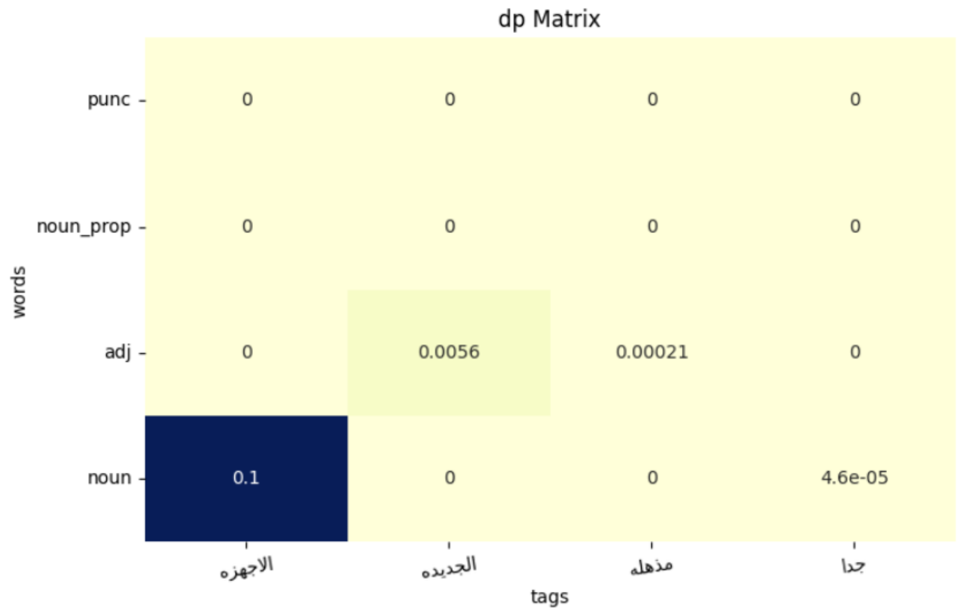
Figure 6 POS Tagging Lattice

3.3.4 Dp&Bp matrix after applying algorithm

Article:

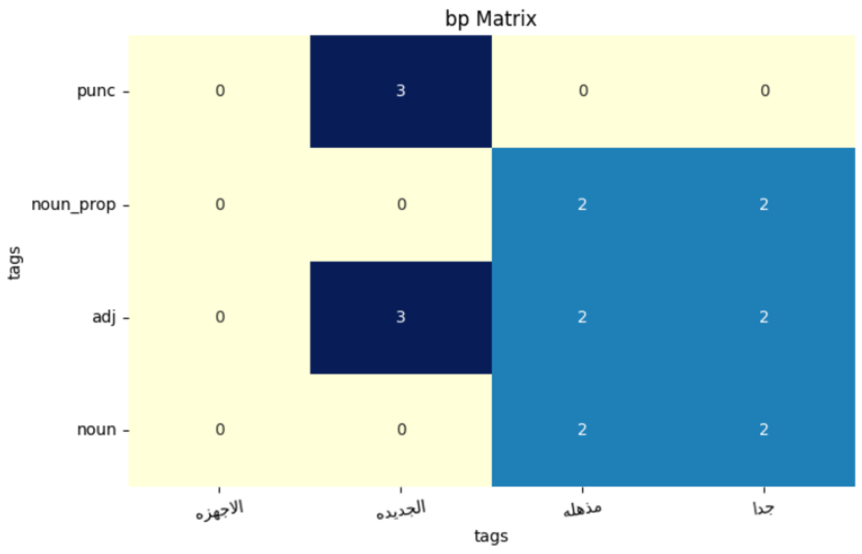
" التكنولوجيا الحديثة مفيدة جداً. الأجهزة الجديدة مذهلة للغاية. البرامج التعليمية رائعة جداً. الإنترنت سريع جداً ومفيد للغاية. الهواتف الذكية مفيدة جداً."

Example: الأجهزة الجديدة مذهلة جداً



	الاجهزه	الجديده	مذهله	جدا
punc	0	0	0	0
noun_prop	0	0	0	0
adj	0	0.0056	0.00021	0
noun	0.1	0	0	4.6e-05

Figure 7 Dynamic Programming Matrix



	الاجهزه	الجديده	مذهله	جدا
punc	0	3	0	0
noun_prop	0	0	2	2
adj	0	3	2	2
noun	0	0	2	2

Figure 8 Back Pointer Matrix

4.0 Training My Own POS-Tagging model

4.1 Goal

To evaluate the effectiveness of different deep learning models in POS tagging and to integrate them into our pipeline, I conducted a series of experiments using an Arabic companies' reviews dataset.

4.2 Models

1. **LSTM (Long Short-Term Memory):**
    - LSTM networks are a type of recurrent neural network (RNN) capable of learning long-term dependencies. They are well-suited for sequence prediction problems and have shown great promise in NLP tasks.
2. **Bi-LSTM (Bidirectional Long Short-Term Memory):**
    - Bi-LSTM networks extend the standard LSTM by processing the input sequence in both forward and backward directions. This allows the model to capture context from both past and future states, improving its accuracy in POS tagging.
3. **GRU (Gated Recurrent Unit):**
    - GRU networks are a simplified variant of LSTMs, designed to address the vanishing gradient problem in standard RNNs. GRUs have fewer parameters and can be more efficient while still delivering competitive performance in sequence modeling tasks.
4. **Bi-GRU (Bidirectional Gated Recurrent Unit):**
    - Bi-GRU networks, similar to Bi-LSTM, process the input sequence in both directions, providing the model with a comprehensive understanding of the context surrounding each word. This bidirectional processing enhances the model's ability to accurately tag POS in complex sentences.

4.3 Embedding Model: AraVec

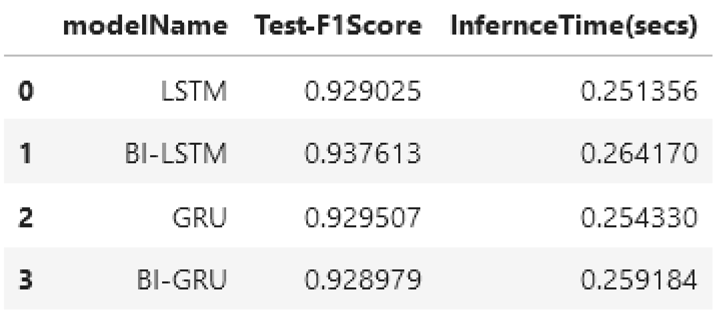
For the embeddings, I utilized the AraVec model, specifically the "**full_grams_cbow_300_twitter.mdl**" variant. AraVec is a pre-trained Arabic word embedding model that captures semantic information from a large corpus of Arabic text. The "**full_grams_cbow_300_twitter.mdl**" is trained using the Continuous Bag of Words (CBOW) approach with 300 dimensions on Twitter data.

Using this model provided rich contextual representations of words, which enhanced the performance of the POS tagging models. The embeddings facilitated better generalization and understanding of the Arabic language nuances present in the review dataset.

4.4 Results

I trained the four aforementioned models (LSTM, Bi-LSTM, GRU, Bi-GRU) over 25 epochs, each with 64 hidden states. To optimize the training process, I employed gradient accumulation steps, accumulating gradients over 4 steps, which expedited the training process.

To evaluate the performance of these models, I compared them using the F1 score and measured their inference times on the test dataset. The results are summarized in the following table:



	modelName	Test-F1Score	InfernceTime(secs)
0	LSTM	0.929025	0.251356
1	BI-LSTM	0.937613	0.264170
2	GRU	0.929507	0.254330
3	BI-GRU	0.928979	0.259184

Figure 9 Result-1

5.0 Overall Conclusion

The project successfully demonstrated the application of POS tagging on Arabic text using both traditional and advanced techniques. Deep learning models, especially Bi-LSTM and Bi-GRU, provided superior performance compared to traditional methods.

6.0 Tools and Resources

6.1 Tools

1. **Python**: The primary programming language used for developing the entire pipeline.
2. **Jupyter Notebook**: For interactive development, visualization, and documentation of the code.
3. **Camel Tools**: Specifically used for Arabic text preprocessing, tokenization, and POS tagging.
4. **AraVec**: Pre-trained Arabic word embeddings model ("full_grams_cbow_300_twitter.mdl") used to generate rich word representations.
5. **Pytorch**: Deep learning libraries used to build, train, and evaluate LSTM, Bi-LSTM, GRU, and Bi-GRU models.
6. **scikit-learn**: For various machine learning utilities, including the calculation of performance metrics such as the F1 score.
7. **Matplotlib/Seaborn**: Libraries for data visualization to create plots for analysis and comparison of model performance.
8. **Beautiful Soup**: Used for web scraping reviews from social media platforms like Twitter and Facebook.
9. **NetworkX**: For creating and visualizing the transition state diagrams of the POS tags.

6.2 Resources

1. **Youm7 Website**: Source of the large corpus of Arabic text articles used in the project.
2. **AraVec**: Pre-trained word embeddings specifically designed for Arabic language processing.
3. **Arabic Companies Reviews Dataset (Kaggle)**: Scraped reviews used for training and evaluating the deep learning models.

6.3 External Libraries

1. **NumPy**: For numerical computations and handling matrix operations.
2. **pandas**: For data manipulation and analysis, particularly for handling the datasets.
3. **tqdm**: For displaying progress bars during data preprocessing and model training.
4. **NLTK**: For natural language processing tasks that complement the functionalities provided by Camel Tools and spaCy.

7.0 Biggest Challenge Faced

The biggest challenge in carrying out this project was effectively implementing the Hidden Markov Model (HMM) and the Viterbi algorithm for POS tagging. Specific challenges included:

- **Complexity of HMM**: Understanding and implementing the HMM for POS tagging was challenging due to the complexity of calculating transition and emission probabilities accurately from the training data.
- **Viterbi Algorithm**: Applying the Viterbi algorithm for decoding the sequence of POS tags required careful handling of the probabilistic computations and ensuring numerical stability.
- **Sparse Data**: The transition and emission matrices often contained many zero probabilities due to the sparsity of certain tag combinations and word-tag pairs in the training data, complicating the model's performance.
- **Integration with Deep Learning**: Combining traditional HMM methods with modern deep learning approaches required careful integration and testing to ensure that the models complemented each other effectively.

8.0 Lessons Learned

This project provided valuable insights and learning experiences in implementing and understanding HMM and the Viterbi algorithm:

- **Deep Understanding of HMM**: I gained a thorough understanding of the Hidden Markov Model, including how to construct transition and emission matrices and the significance of initial state probabilities (π matrix).
- **Effective Use of the Viterbi Algorithm**: Implementing the Viterbi algorithm for decoding POS tags from HMMs taught me how to efficiently compute the most probable sequence of hidden states, which is crucial for accurate POS tagging.
- **Handling Sparse Data**: I learned techniques to handle sparsity in transition and emission matrices, such as smoothing methods, which are essential for improving the model's performance.
- **Model Evaluation and Optimization**: The process of evaluating the HMM-based POS tagger using metrics like the F1 score provided insights into the strengths and limitations of HMMs compared to deep learning models.
- **Hybrid Approaches**: Integrating HMMs with deep learning models highlighted the potential benefits of hybrid approaches, combining the strengths of probabilistic models and neural networks for improved performance.