ASSESSMENT COVER SHEET

Assignment Details

Course: Applied Natural Language Processing________________

Semester/Academic Year: Sem 1 2023________________

Assignment title: Info Retrieval System________________

Assessment Criteria

Assessment Criteria are included in the Assignment Descriptions that are published on each course's website.

Plagiarism and Collusion

Plagiarism: using another person's ideas, designs, words, or works without appropriate acknowledgment.

Collusion: another person assisting in the production of an assessment submission without the express requirement, consent or knowledge of the assessor.

Consequences of Plagiarism and Collusion

The penalties associated with plagiarism and collusion are designed to impose sanctions on offenders that reflect the seriousness of the University's commitment to academic integrity. Penalties may include the requirement to revise and resubmit assessment work, receiving a result of zero for the assessment work, failing the course, expulsion, and/or receiving a financial penalty.

declaration

I declare that all material in this assessment is my own work except where there is clear acknowledgment and reference to the work of others. I have read the University Policy Statement on Plagiarism, Collusion, and Related Forms of Cheating:

mttp://www.adelaide.edu.au/policies/?230

I give permission for my assessment work to be reproduced and submitted to academic staff for the purposes of assessment and to be copied, submitted, and retained in a form suitable for electronic checking of plagiarism.

Yash Kasundra (A1838670)
27/04/2023

SIGNATURE AND DATE

Introduction

This project aims to create a powerful Python text-matching system that can successfully match queries based on their semantic similarities. Our system can match questions accurately and quickly by using various techniques, including TF-IDF, sentence embedding models, and matrix operations.

We have decided to use Python as our programming language to develop our text-matching system. Where applicable, we will use third-party packages for particular tasks including reading files, text preparation, word count statistics, and matrix operations. The following essential functions, however, will be carried out using custom code rather than third-party packages:

1. Term Frequency-Inverse Document Frequency (TF-IDF) calculation is a crucial part of our text-matching system. To accurately gauge the significance of words within questions, we shall implement the TF-IDF calculation from scratch.

2. Building an Inverted Index: We will build an Inverted Index to allow for effective retrieval and search. This data format will make it easier to find pertinent queries quickly using particular words or terms, which will speed up and improve the efficiency of the system as a whole.

3. Sentence Embedding Calculation from Word Embeddings: In order to fully understand the semantic meaning of a question, sentence embedding is essential. We will using a outside package called "Glove" for sentence embeddings based on word embeddings. This approach will help us in creating a better model than TF-IDF since Glove was trained on much larger data than ours.

4. Sentence Similarity Calculation: We will create our own algorithms to compare the distance or similarity between each sentence embedding in order to assess how similar the questions are. We can properly determine the semantic similarity of questions by comparing the embeddings, which enables efficient question matching.

We have used a variety of methods to improve the question-matching process throughout the project. With the help of TF-IDF, we have been able to appropriately weigh terms in our matching system by capturing their relative value. The Average Word-based Sentence Embedding Model and the Smooth Inverse Frequency-based Sentence Embedding Model are two further sentence embedding models that we have investigated. These models have demonstrated success in accurately representing sentences and capturing the semantic meaning of queries.

1. Exploratory Data Analysis:

To get insightful knowledge and a deeper comprehension of our dataset, we concentrate on performing exploratory data analysis (EDA) during this stage of our research. We can find patterns, trends, and anomalies with EDA, which opens the door to more thorough preprocessing and informed decision-making. Let's examine the EDA procedure we used for our project to develop a text matching system.

1. Handling Null and Duplicate Values:
   Our EDA journey began with identifying and addressing duplicate and null values. The precision and reliability of our study can be hampered by null values, thus we carefully checked our dataset to look for any missing data. When the null values were found, we made the decision to eliminate them so that our subsequent research would be built on accurate and complete data. Similar to how duplicate records can influence our results, we likewise addressed any instances of them. We guarantee that each question appears just once in our study by removing duplicates.

2. Label Column Distribution:
   Our label column's distribution must be understood to perform question matching task. We made a histogram to better understand how the labels were distributed. With the help of the histogram, which visualizes the frequency distribution of the label values, we can see clearly if the dataset is balanced or unbalanced. We can determine the prevalence of various question kinds using this data, which also directs our modeling choices.

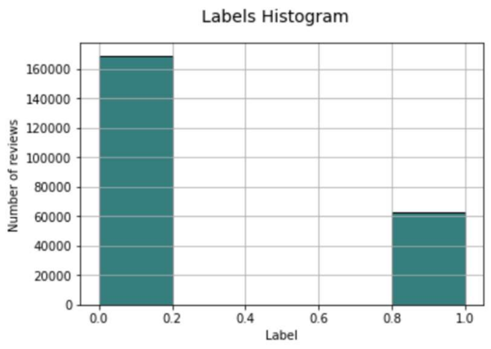


Fig-1: Histogram for checking distribution of Label’s

3. Word Cloud and N-gram Analysis:
   We used word cloud and n-gram analysis to better grasp the information contained in the Question1 and Question2 columns. With more common words appearing larger in word clouds, word frequency in a corpus is represented visually. We can see the most often occurring words in the questions by creating word clouds for each column, which helps us find recurrent themes or topics.

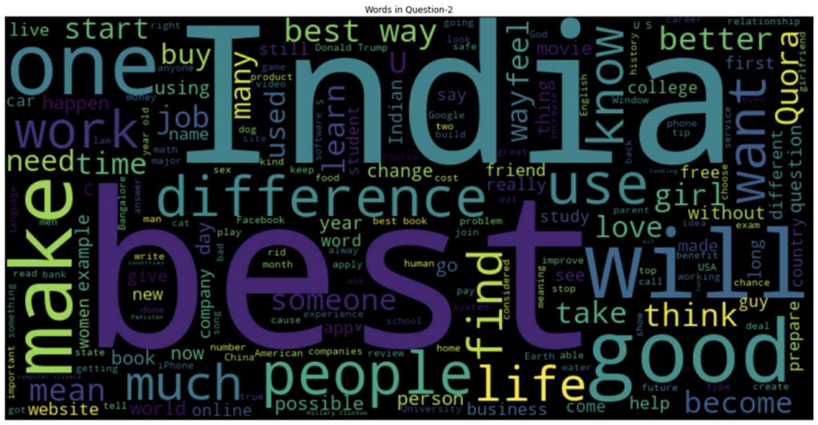


Fig-2: Wordcloud for question2 column

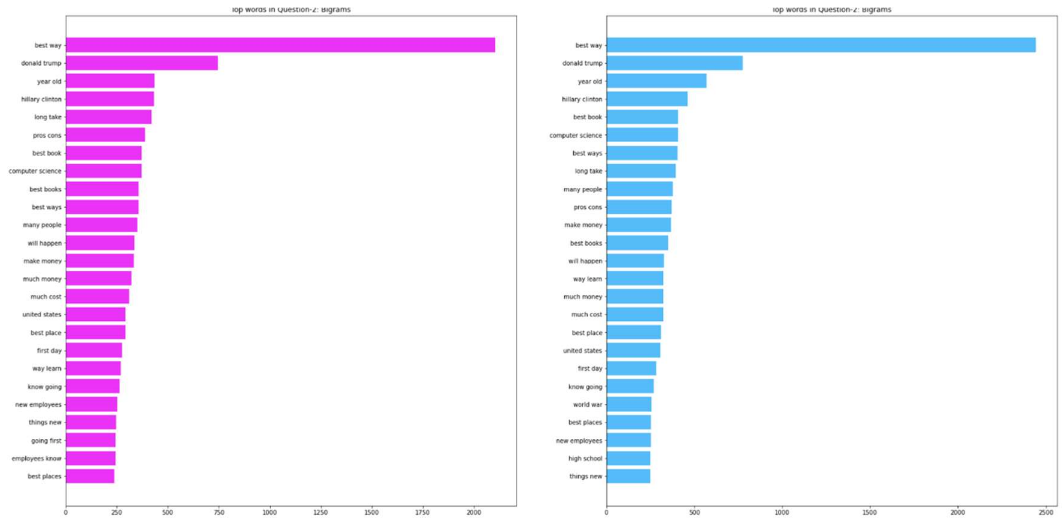


Fig-3: Bigram plot for Question1 and Question2 columns

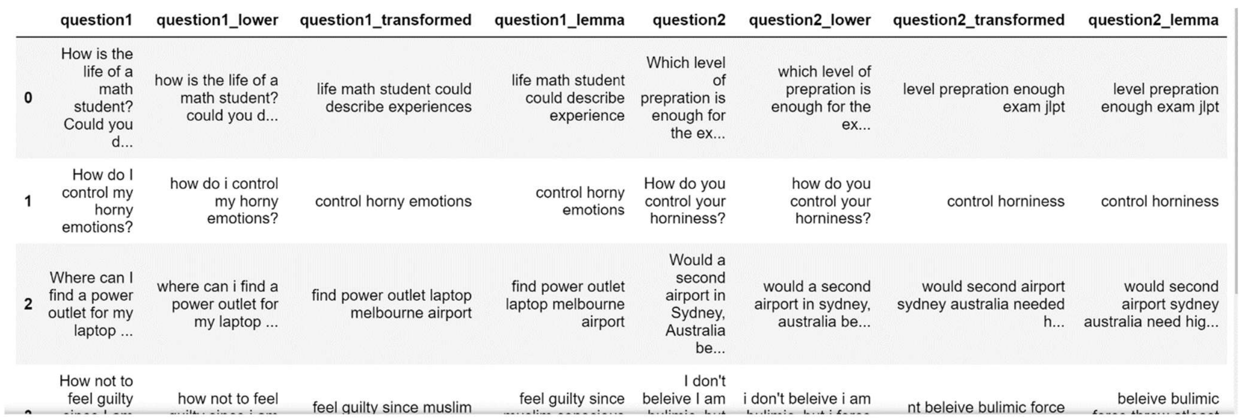
Additionally, we looked at the distribution of n-grams (unigrams, bigrams, and trigrams) in the text data using n-gram analysis. We can find repeating patterns and correlations between words by examining n-grams, which are sequences of n words. Plotting n-grams enables us to recognise frequently occuring phrases or word combinations that may be crucial in question matching.

These exploratory studies help us learn important things about our dataset, spot problems with the data's quality, and comprehend the nature of the questions we are trying to answer. These insights can help us make preprocessing decisions and lay a strong foundation for our text matching system's later stages.

2. Preprocessing

Preprocessing is essential to the quality and efficiency of our text-matching system for question matching in the field of natural language processing (NLP). In this project, we put a lot of effort into cleaning and transforming our textual data through a number of preprocessing stages, which will ultimately improve the precision and effectiveness of our system. Let's investigate the preparation procedures used on our dataset.:

- **Lower_casing:** Lowercase every word in the text to prevent having vocabulary words appear twice. This method can also help to make the analysis simpler by reducing the number of distinct words [2].
- **Stop Word Removal:** Stopwords are frequently used words in a language (such as "and," "the," and "is") that have little significance and can be safely discarded in our research. We take stopwords out of our question text during the preprocessing stage. By eliminating these terms, we lower noise and raise the importance of the words that remain, enabling more precise question matching [6,7]
- **Elimination of Special Characters and Punctuation:** Although commas, periods, question marks, and exclamation points are used for grammatical purposes, they are frequently not necessary for our study. We remove punctuation from our question content during preparation. This makes the text easier to read and makes sure that our system concentrates on the important semantic information and substance rather than unnecessary symbols [2].
- **Number and HTML tag Removal:** In rare circumstances, textual numbers may not support a question's meaning or aim. We eliminate numerals from our data during preparation to make it simpler and more uniform. HTML tags that are unrelated to our investigation may be present in our dataset if the text was taken from HTML sources. As a result, we eliminate these tags during the preprocessing stage [5].
- **Lemmatization:** Lemmatization is a process that converts words into lemmas, which are the basic or dictionary forms of words. We can combine several word forms, such as plurals, verb conjugations, and different tenses, into a single representative form by using lemmatization. By aligning comparable words, this stage lowers lexical variety, improves text coherence, and increases matching precision [12].



	question1	question1_lower	question1_transformed	question1_lemma	question2	question2_lower	question2_transformed	question2_lemma
0	How is the life of a math student? Could you d...	how is the life of a math student? could you d...	life math student could describe experiences	life math student could describe experience	Which level of prepration is enough for the ex...	which level of prepration is enough for the ex...	level prepration enough exam jlpt	level prepration enough exam jlpt
1	How do I control my horny emotions?	how do i control my horny emotions?	control horny emotions	control horny emotions	How do you control your horniness?	how do you control your horniness?	control horniness	control horniness
2	Where can I find a power outlet for my laptop ...	where can i find a power outlet for my laptop ...	find power outlet laptop melbourne airport	find power outlet laptop melbourne airport	Would a second airport in Sydney, Australia be...	would a second airport in sydney, australia be...	would second airport sydney australia needed h...	would second airport sydney australia need hig...
[illegible]	How not to feel guilty	how not to feel	feel guilty since muslim	feel guilty since	I don't beleive I am	i don't beleive i am	nt beleive bulimic force	beleive bulimic

Table-1: Preprocessed review

3. Model

In this project we used 3 models, let’s take a look at each of them.

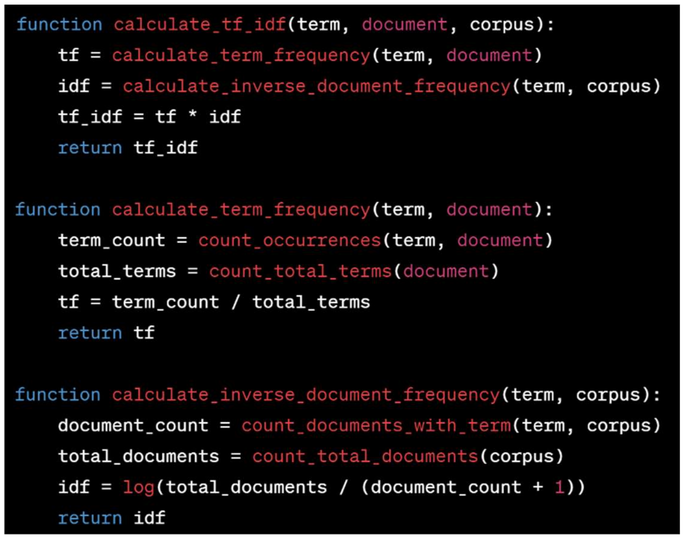
1. **TF-IDF (Term Frequency-Inverse Document Frequency):**
   A common numerical statistic that assesses the significance of a term in a document inside a collection or corpus is the TF-IDF. It consists of two parts: inverse document frequency (IDF) and term frequency (TF).

   TF calculates how frequently a term appears in a given document. Assuming that terms that appear more frequently in the paper are more pertinent, it gives them more weight. However, TF might not be able to fully express a term's significance throughout the entire corpus.

   On the other hand, IDF calculates a term's rarity throughout the entire corpus. Inferring that terms with fewer occurrences in the corpus have greater discriminative potential, it gives those terms heavier weights. The logarithm of the inverse fraction of documents containing the phrase is used to compute IDF.

   The TF and IDF values are multiplied to provide the TF-IDF score for a term in a document. This method gives phrases that frequently exist in one document but not in others more weight, suggesting that they are important in characterising the content of that text.

   Information retrieval, text mining, and document classification tasks frequently employ TF-IDF. It enables quick computations of content similarity and document ranking using keywords.



```
function calculate_tf_idf(term, document, corpus):
    tf = calculate_term_frequency(term, document)
    idf = calculate_inverse_document_frequency(term, corpus)
    tf_idf = tf * idf
    return tf_idf

function calculate_term_frequency(term, document):
    term_count = count_occurrences(term, document)
    total_terms = count_total_terms(document)
    tf = term_count / total_terms
    return tf

function calculate_inverse_document_frequency(term, corpus):
    document_count = count_documents_with_term(term, corpus)
    total_documents = count_total_documents(corpus)
    idf = log(total_documents / (document_count + 1))
    return idf
```

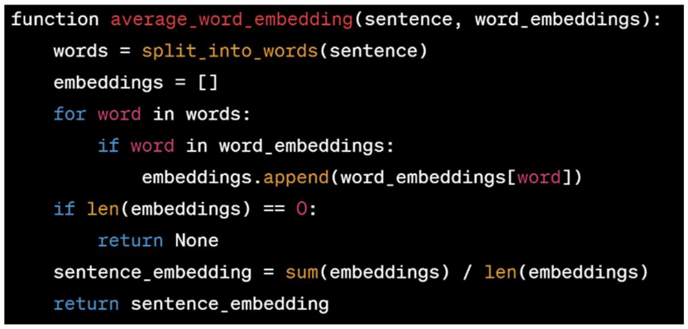
Fig-4: Pseudo code for TF-IDF

2. **Average Word-based Sentence Embedding Model:**
   By calculating the average of the word embeddings within the phrase, the average word-based sentence embedding model encodes sentences as fixed-length vectors. Dense vector representations called word embeddings capture the semantic meaning of words based on their environment.

   Using this method, each word in the phrase is first represented using pre-trained word embeddings (such as Word2Vec, GloVe, or FastText). You then average these word embeddings to create a single vector that represents the complete sentence.

This approach is simple to use and computationally effective. It can extract certain semantic data that is hidden in the sentence's words. It does not, however, take word order into account or capture more intricate sentence-level interactions.

In tasks like text classification, sentiment analysis, and clustering where a quick and easy representation of sentences is needed, average word-based sentence embeddings are frequently utilized.



```
function average_word_embedding(sentence, word_embeddings):
    words = split_into_words(sentence)
    embeddings = []
    for word in words:
        if word in word_embeddings:
            embeddings.append(word_embeddings[word])
    if len(embeddings) == 0:
        return None
    sentence_embedding = sum(embeddings) / len(embeddings)
    return sentence_embedding
```

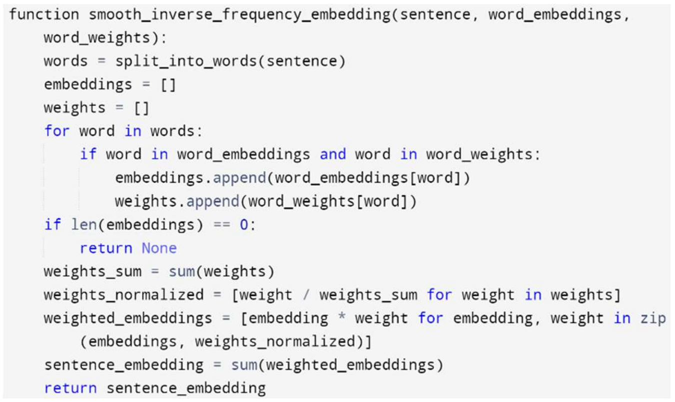
Fig-5: Pseudo code for Average word embedding

3. **Smooth Inverse Frequency-based Sentence Embedding Model:**
The term weighting system used in the Smooth Inverse Frequency (SIF)-based sentence embedding model improves the typical word-based approach by addressing the problem of common word dominance.

Each word's embedding in the SIF model is weighted inversely in relation to how frequently it appears in a sizable reference corpus. Low-frequency words like "the," "and," or "is" are given a lower weight because they commonly occur in sentences and may not be very informative.

The phrase embedding is calculated as the weighted average of the word embeddings after the inverse frequency weighting, much like the average word-based technique. This weighting system enables more informative terms to contribute more to the representation of the phrase while reducing the impact of common words.

By lessening the influence of common words and highlighting the contribution of more informative terms, the SIF-based model outperforms the typical word-based model. In tasks like text similarity, paraphrase identification, and document clustering, where it's important to understand a sentence's underlying meaning, this method can be helpful.



```
function smooth_inverse_frequency_embedding(sentence, word_embeddings,
    word_weights):
    words = split_into_words(sentence)
    embeddings = []
    weights = []
    for word in words:
        if word in word_embeddings and word in word_weights:
            embeddings.append(word_embeddings[word])
            weights.append(word_weights[word])
    if len(embeddings) == 0:
        return None
    weights_sum = sum(weights)
    weights_normalized = [weight / weights_sum for weight in weights]
    weighted_embeddings = [embedding * weight for embedding, weight in zip
        (embeddings, weights_normalized)]
    sentence_embedding = sum(weighted_embeddings)
    return sentence_embedding
```

Fig-6: Pseudo code for Smooth Inverse Frequency

Now let's see how to build sentence representation in all 3 models:

1. **TF-IDF:** To build sentence representations using TF-IDF, you follow these steps:
    - Preprocess the documents by tokenizing them into words.
    - Compute the TF-IDF score for each term in each document using the TF-IDF formula.
    - Represent each sentence as a vector where each element corresponds to the TF-IDF score of a term in the sentence.

2. **Average Word-based Sentence Embedding Model:** To build sentence representations using the average word-based model, you follow these steps:
    - Preprocess the sentences by tokenizing them into words.
    - Use pre-trained word embeddings (e.g., Word2Vec, GloVe) to represent each word as a dense vector.
    - Compute the average of the word embeddings within each sentence.
    - The resulting vector represents the sentence.

3. **Smooth Inverse Frequency-based Sentence Embedding Model:** To build sentence representations using the SIF-based model, you follow these steps:
    - Preprocess the sentences by tokenizing them into words.
    - Use pre-trained word embeddings to represent each word as a dense vector.
    - Calculate the inverse frequency weights for each word in a reference corpus.
    - Apply the inverse frequency weights to the word embeddings.
    - Compute the weighted average of the word embeddings within each sentence.
    - The resulting vector represents the sentence.

Comparison between TF-IDF and Sentence Embedding model :

Sentences are represented sparsely in TF-IDF-based systems, with each sentence being represented by a vector of TF-IDF scores for individual phrases. These systems work well for keyword-based matching and retrieval, but they do not take into account the order of words in sentences or the semantic links between words.

By representing sentences as dense vectors in a continuous vector space, sentence embedding-based systems, on the other hand, seek to capture the semantic content of sentences. These systems are helpful for tasks like text similarity, paraphrase identification, and clustering because they can capture more intricate sentence-level interactions.

Sentence embedding-based algorithms frequently do better at identifying semantic similarities between sentences than TF-IDF. However, pre-training on vast amounts of text data is necessary for creating high-quality sentence embeddings, and this can be computationally expensive and possibly resource-intensive.

Comparing different ways of creating sentence embeddings:

There are various methods to create sentence embeddings, and each approach has its own strengths and limitations.

a. Average Word-based Sentence Embeddings:

- Pros: Simple to implement, computationally efficient, and captures some semantic information encoded in the words.

- Cons: Ignores word order and fails to capture more complex sentence-level relationships.

b. Smooth Inverse Frequency-based Sentence Embeddings:

- Pros: Addresses the issue of common word dominance by weighting word embeddings inversely proportional to their frequency. Can provide improved representations compared to average word-based embeddings.
- Cons: Still does not capture word order or more sophisticated sentence-level structures.

Increasingly sophisticated techniques, like models based on transformers (like BERT and GPT), have become increasingly popular recently. These models have demonstrated higher performance in various natural language processing tasks by utilizing self-attention processes to capture contextual information. By taking into account the complete sentence and its context, they can provide contextualized sentence embeddings.

The ideal strategy for constructing sentence embeddings will depend on the specific task at hand and the resources that are accessible.

References:

[1] Jurafsky, D., & Martin, J. H. (2019). Speech and Language Processing (3rd ed.). Pearson. [Link: https://web.stanford.edu/~jurafsky/slp3/]

[2] Zhang, H., & Wallace, B. C. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. arXiv preprint arXiv:1510.03820. [Link: https://arxiv.org/abs/1510.03820]

[3] Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. Foundations and Trends in Information Retrieval, 2(1-2), 1-135. [Link: https://www.cs.uic.edu/~liub/FBS/SentimentAnalysis-and-OpinionMining.pdf]

[4] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781. [Link: https://arxiv.org/abs/1301.3781]

[5] Manning, C. D., & Schütze, H. (1999). Foundations of Statistical Natural Language Processing. MIT Press. [Link: https://nlp.stanford.edu/fsnlp/]

[6] Bird, S., Klein, E., & Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media, Inc. [Link: http://www.nltk.org/book/]

[7] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. arXiv preprint arXiv:1301.3781. [Link: https://arxiv.org/abs/1301.3781]

[8] Ranga, V. (2020) Naïve Bayes algorithm -implementation from scratch in python., Medium. Medium. [Link: https://medium.com/%40rangavamsi5/na%C3%AFve-bayes-algorithm-implementation-from-scratch-in-python-7b2cc39268b9]

[9] Stanina, I. (2021) Implementing naive Bayes algorithm from scratch - python., Medium. Towards Data Science. [Link: https://towardsdatascience.com/implementing-naive-bayes-algorithm-from-scratch-python-c6880cfc9c41]

[10] Byte-pair encoding tokenization - hugging face course (no date) Byte-Pair Encoding tokenization - Hugging Face Course. [Link: https://huggingface.co/course/chapter6/5?fw=pt]

[11] Saumyab (2022) Stemming vs lemmatization in NLP: Must-know differences, Analytics Vidhya. [Link: https://www.analyticsvidhya.com/blog/2022/06/stemming-vs-lemmatization-in-nlp-must-know-differences/#:~:text=Stemming%20is%20a%20process%20that,%27%20would%20return%20%27Car%27]