Paraphrase Identification for Plagiarism Detection using Word Embeddings

Plagiarism detection

Submitted in partial fulfilment of the requirements of
Edinburgh Napier University
for the Degree of
MSc Computing

School of Computing

April 2024

Authorship Declaration

I, *Bani Kaur Bhatia*, confirm that this dissertation and the work presented in it and my own achievement.

1. Where I have consulted the published work of others this is always clearly attributed;
2. Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;
3. I have acknowledged all main sources of help;
4. If my research follows on from previous work or is part of any larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;
5. I have read and understood the penalties associated with Academic Misconduct.
6. I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines.

Type name: *Bani Kaur Bhatia*
Date: 24th April 2024

Matriculation No: 40626682

General Data Protection Regulation Declaration

Under the General Data Protection Regulation (GDPR) (EU) 2016/679, the University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name under *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

Bani Kaur Bhatia

The University may not make this dissertation available to others.

Abstract

Current plagiarism checkers focus on catching copied text and missing instances where the meaning is stolen, but the wording is changed. Deep learning offers a promising solution through representing text in an embedding space – a technique that captures a word's meaning based on its context. This project explores text representation approaches such as TF-IDF, Word2Vec, GloVe, SBERT, and BERT to evaluate which one performs best at identifying paraphrases. We evaluated these embedding models on two publicly available datasets on short and plagiarized text based on several performance metrics such as accuracy, recall, precision, and F1-score. We compare all five embeddings using these datasets on two classification models, XGBClassifier & LightGBMClassifier. After extensive experiments and analysis methods, Word2Vec,BERT, SBERT & GloVe emerged as the most effective model, considering 100% accuracy, precision, recall and F1-score for small corpus like MRPC. Transformer-based models like SBERT & BERT performed best with 88% accuracy, precision, recall and F1-score in identifying the paraphrases on large corpus like Quora. Further, we evaluated the effect of the quantization technique on the embeddings of BERT and found that quantization does change the classification performance for Quora and remains the same for MRPC.Out of the two classifiers, XGBClassifier provided the best results for prediction. Coding is done in Python notebook for MRPC [1] and Quora [2] corpora.

[1] `https://github.com/banibhatia/msc_dissertation/blob/main/Embedding_dissertation_MRPC.ipynb`

[2] `https://github.com/banibhatia/msc_dissertation/blob/main/Embedding_dissertation_Quora_Results_Glove_SBERT.ipynb`

Contents

List of Figures

Acknowledgements

First of all, I would like to thank my supervisor, Professor Md Zia Ullah of Edinburgh Napier University, for coming up with the idea for the study and for all of his insight, counsel, encouragement, and support during this research. I am grateful to Prof. Yanchao Yu for serving as the internal examiner for my dissertation and for his insightful comments. In addition, I want to express my gratitude to my parents for providing me with financial assistance during my academic career. Lastly, I would like to thank Edinburgh Napier University for giving me this chance to present my work.

Chapter 1

Introduction

1.1 Background

Rewriting text while preserving its meaning is known as paraphrasing. Deep learning models are showing great potential in identifying similar meaning between documents using a technique called embedding. This technique represents the vocabulary of a document, capturing the meaning of individual words and the overall document context.

1.2 Aim and Objectives

The objective of this study is to create a system that measures semantic similarity between a given dataset and identifies paraphrase plagiarism using machine learning and statistics. The methodical investigation facilitates the process of selecting the mechanisms and hyperparameters most appropriate for a novel framework.

This research compares five models such as TF-IDF, Word2Vec, GloVe, SBERT, and BERT on two open-source corpora, namely Quora and MRPC available on HuggingFace and in turn, these models are evaluated in terms of accuracy, precision, recall, and F1-score metrics. Before applying the models, we split the data into test and train dataset, data prepossessing techniques, feature engineering are applied after careful experimentation on corpus.

1.3 Research Questions

Reviewing and contrasting the effectiveness of several embedding-based models (including some deep learning models) for semantic similarity detection

is one of the study's primary research goals. This effort will result in the following contributions:

1. A methodical overview of word embedding models based on corpora.
2. Explanation of the most effective models, pre-processing methods, and the types of corpora that are most suitable for them in order to detect plagiarism

1.4 Thesis Structure

The thesis is structured as a discussion of the following topics:-

- Literature review
- Proposed methodology
- Implementation
- Evaluation
- Conclusions

Chapter 2

Literature Review

While the notion of using deep learning models' semantic potential is not new—researchers have been doing so for years—there aren't many published papers that discuss its application to plagiarism detection. (Altheneyan and Menai, 2020)

Research is conducted in determining whether the word vectorization methodology is more effective for recognising paraphrases in sentences (Gangadharan et al., 2020). Word vectorization is a technique that associates each word with a vector in order to recover information from vast collections of textual data, such as texts or corpuses.Count Vectorizer, Hashing Vectorizer, TF-IDF Vectorizer, fastText, ELMo, GloVe, and BERT are the word vectorization approaches that are compared in this work. The primary idea behind vectors is to identify word and phrase similarity.This study used a dataset of questions that were both similar and distinct, and it used the vectorization techniques previously discussed to turn the dataset into vectors. The findings are compared after the cosine similarity utilising the vectors is applied to determine the document similarity. It's official: fastText is the most effective way to transform words to vectors and determine how similar they are.

Researchers have hypothesised that certain sorts of paraphrase serve as the paraphrasing processes behind plagiarism (Alvi et al., 2021) .The purpose of this study is to uncover two key forms of paraphrasing: synonym substitution and word reordering in pairs of paraphrased, plagiarised sentences. According to the suggested methodology, the best results in terms of F1 scores are obtained by using ConceptNet Numberbatch pretrained word embeddings and the Smith Waterman Algorithm for Plagiarism Detection.

The purpose of this study (Vrbanec and Meštrović, 2020)is to provide an overview of the performance of several kinds of corpus-based models, particularly deep learning (DL) models, with respect to the paraphrase detection

problem. We provide the findings from the evaluation of eight models (LF-IDF, Word2Vec, Doc2Vec, GloVe, FastText, ELMO, and USE) using three distinct publicly accessible corpora: the Microsoft Research Paraphrase Corpus, the Clough and Stevenson, and the Webis Crowd Paraphrase Corpus 2011. The USE model performs marginally better than the other models, according to the data, and TF-IDF unexpectedly performs far better than predicted. Sadly, it becomes evident to us that the findings of DL models alone are still insufficient to completely address the problem of paraphrase identification because of the challenging scenarios that still arise, where lexical patterns are less important than semantic level.

The research paper (Vrbanec and Meštrović, 2023) examined the performance of 60 methods/models on five paraphrase corpora, evaluating them using conventional evaluation metrics, and undertake a comprehensive series of experiments for assessing the semantic similarity of texts. The findings of the experiment show that DL models perform better than statistical techniques, with the BERT family model known as "distilroberta-base-paraphrase-v1" exhibiting the best performance. Four DL models (Word2Vec, Doc2Vec, FastText, and GloVe) and their nine submodels (total) trained on five corpora, 149 pre-trained language models; four in the first phase (USE, ELMo, BERT, and Laser) and (146 in the second) in paraphrase detection on five corpora, determine that "jfarray_Model_paraphrase-multilingual-mpnet-base-v2_50_Epochs" has the best overall performance.

The performances of five word-embedding-based deep learning models—TF-IDF, Word2Vec, Doc2Vec, FastText, and BERT—in the area of semantic similarity detection are compared and summarised in this paper using two publicly available corpora: Plagiarised Short Answers (PSA) and Quora Question Pairs (QQP). (Chawla et al., 2022).The best text preparation techniques, distance measurements, and thresholds for identifying semantic similarity/paraphrasing have been determined after a thorough evaluation of the literature and testing. The study shows that, among the five models, FastText is the most efficient in terms of evaluation criteria, including resource consumption, receiver operating characteristic (ROC) curve, accuracy, precision, recall, F1-score, and F1.

In the study (Chandra and Stefanus, 2020), the Quora question dataset is examined using Bag of Words (BoW) and Word Piece to chunk the text. Also, tree-boosting algorithms such as Catboost, and XGBoost are widely used. Moreover, a study on deep learning algorithms such as LSTM and BERT to see how effective to classify paraphrase identification is conducted. As a result, BERT gave the best result among other models.

Another research (Ilyas et al., 2021), explores new and modern plagiarism detection tasks especially, text-based plagiarism detection including

monolingual plagiarism detection. A four-stage novel framework is proposed for plagiarism detection. This framework uses Natural Language Processing (NLP). Firstly, Text pre-processing and shallow NLP techniques applied to text. Secondly, a pairwise comparison among source and suspicious documents is carried out using skip-gram and Dice coefficient metric. Thirdly, the Deep NLP technique, along with the word2vec process, is evaluated and with the help of the threshold point, candidate documents are selected. Lastly, the WEKA tool for the classification of documents is used. In conclusion, Word2vec results are close to simple Deep NLP methods, but word2vec also highlights those documents that other techniques may not highlight.

Word embeddings from trained models, such as BERT, are better at extracting contextual information (Bohra and Barwar, 2022). This paper shows that the pre-trained models provide contextualized word embeddings that play a key role in extracting semantic similarity of the content. The obtained results from the BERT model are compared with UGC granted 'Ouriginal' plagiarism detection system available for access at Jai Narain Vyas University. The BERT model gives the best results when used with large training and development datasets.

The advantage of using the fine-tuned sentence-BERT language model for paraphrased sentence classification over the other modern models is proved in the research paper "Using BERT model to Identify Sentences Paraphrase in the News Corpus" [1]. The BERT (Devlin et al., 2018) authors established the following concept to ascertain the semantic similarity between two independent phrases: a token [SEP] is inserted between the sentences, and the sequence is then processed by the model. The next phrase states that the [CLS] marker should be used as input for either simple classification or regression to determine whether the two sentences are related (Viji and Revathy, 2022). While the suggested approach for determining semantic similarity shows sufficient reliability, its working algorithm becomes problematic when the two subsequent tasks need to be completed:

1. Matching a pair of the most semantically similar sentences. It takes more than two days for a modern computer to handle even a little quantity of data, according to experts, as such processing requires over 50 million iterations. Using BERT for such jobs is inefficient and cumbersome due to the high time expenses.

2. Semantic search. The group of scientists (Reimers and Gurevych, 2019) claim that it will take more than 40 seconds to display the results of a single query for a modest sample of phrases since BERT must process

[1] `https://ceur-ws.org/Vol-3171/paper6.pdf` accessed on 19/04/2024

each unique pair. In this instance, BERT is inappropriate for the work at hand due to the intricacy of the calculations.

There is a need to use a new, more adaptable, and quicker method to address the aforementioned problems since it is extremely difficult to accomplish these two scenarios using the BERT language paradigm. Recently, N. Reimers and I. Gurevych (Reimers and Gurevych, 2019) created the Sentence-BERT model, a twin network that processes two compared sentences at once.

Chapter 3

Proposed Methodology

3.1 Introduction

In this research work, the effectiveness of five embedding models—TF-IDF, Word2Vec, Glove, BERT, and SBERT—in the job of detecting plagiarism is evaluated and compared using two publicly accessible corpora: Quora and the MRPC dataset on HuggingFace. Training a classification model is done after applying the proper preprocessing methods to both corpora. Once that is done, the models are assessed using the conventional metrics of F1-score, accuracy, precision, and recall.

3.2 Problem Definition

The problem of paraphrase detection is stated as follows:

Given two text sequences A and B, such that $A = < a_1, a_2, \cdot, a_{|A|} >$ and $B = < b_1, b_2, \cdot, b_{|B|} >$. The paraphrase label $l \in \{0, 1\}$ is defined between two sequences A and B. The label $l = 1$ if A and B are paraphrased, otherwise $l = 0$.

For example, 'Quora' dataset contains the set of questions that are paraphrased in the 3.1

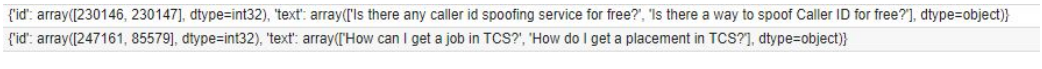


Figure 3.1: Paraphrased questions

It also contains a set of questions that are not paraphrased as shown in the 3.2

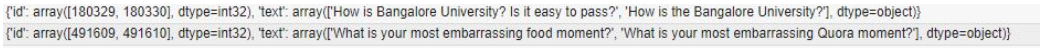


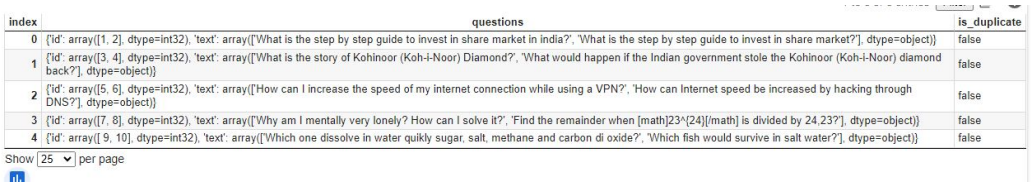
Figure 3.2: Questions not paraphrased

3.3 Corpora

The paper aims at comparing the five models based on their semantic similarity detection. There are two open source corpora used in the semantic similarity detection. They are described as below:

3.3.1 Quora

Quora dataset is available on HuggingFace library[1]. It is composed of question pairs, and the task is to determine if the questions are paraphrases of each other (have the same meaning) to find out plagiarism and paraphrasing to be able to combine similar/duplicate questions. The dataset contains genuine examples from the Web site with over 400,000 records.In the initial dataset there are columns named as 'index', 'question', and 'isduplicate'.



index	questions	is_duplicate
0	{'id': array([1, 2], dtype=int32), 'text': array(['What is the step by step guide to invest in share market in india?', 'What is the step by step guide to invest in share market?'], dtype=object)}	false
1	{'id': array([3, 4], dtype=int32), 'text': array(['What is the story of Kohinoor (Koh-i-Noor) Diamond?', 'What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?'], dtype=object)}	false
2	{'id': array([5, 6], dtype=int32), 'text': array(['How can I increase the speed of my internet connection while using a VPN?', 'How can Internet speed be increased by hacking through DNS?'], dtype=object)}	false
3	{'id': array([7, 8], dtype=int32), 'text': array(['Why am I mentally very lonely? How can I solve it?', 'Find the remainder when [math]23^{24}[/math] is divided by 24,23?'], dtype=object)}	false
4	{'id': array([9, 10], dtype=int32), 'text': array(['Which one dissolve in water quikly sugar, salt, methane and carbon di oxide?', 'Which fish would survive in salt water?'], dtype=object)}	false

Figure 3.3: Quora dataset

3.3.2 MRPC

MRPC dataset is available on HuggingFace [2]. It is composed of four columns namely 'sentence1', 'sentence2', 'label' and 'idx'. There are total 5800 records which is split into test and train dataset.

[1] `https://huggingface.co/datasets/quora` accessed on 02/02/2024

[2] `https://huggingface.co/datasets/nyu-mll/glue/viewer/mrpc` accessed on 07/04/2024

	sentence1	sentence2	label	idx
0	PCCW 's chief operating officer , Mike Butcher...	Current Chief Operating Officer Mike Butcher a...	1	0
1	The world 's two largest automakers said their...	Domestic sales at both GM and No. 2 Ford Motor...	1	1
2	According to the federal Centers for Disease C...	The Centers for Disease Control and Prevention...	1	2
3	A tropical storm rapidly developed in the Gulf...	A tropical storm rapidly developed in the Gulf...	0	3
4	The company didn 't detail the costs of the re...	But company officials expect the costs of the ...	0	4

Figure 3.4: MRPC dataset

3.4 Embedding Models

3.4.1 TF-IDF

TF-IDF (term frequency-inverse document frequency) is a famous technique for weighting terms within a document collection. It reflects the importance of a word to a specific document in relation to the entire collection.

Term Frequency (TF): This measures how often a word appears in a particular document. It is calculated by dividing the number of times a word appears in a document by the total number of words in that document (Kim and Gil, 2019). For instance, if the word “computer" shows up five times in a document with 100 words, the TF for “computer" in that document would be 8 / 100 = 0.08.

Inverse Document Frequency (IDF): This captures how often a word appears across a collection of documents. Words that show up in many documents will have a low IDF value, and words that appear only in a few documents will have a high IDF value. The IDF is calculated using the following formula:

$$IDF = \log \frac{N}{n\_w} \tag{3.1}$$

Where N is the total number of documents in the collection. n_w is the number of documents where the word appears. For example, if a word appears in 10 out of 1000 documents in a collection, then its IDF would be: $IDF = \log \frac{1000}{10} = 2$

By considering both TF and IDF, TF-IDF helps us to weigh the terms that are important for a specific document. Words that appear frequently within a document (high TF) but not very frequently across the document collection (high IDF) will have a high TF-IDF value, indicating their significance for that specific document.

3.4.2 Word2Vec

Word2Vec is a popular technique for representing words as numerical vectors. These vectors capture the semantic relationships between words, allowing us to perform tasks like finding synonyms or measuring word similarity. Here's a breakdown of Word2Vec based on the research paper "Efficient Estimation of Word Representations in Vector Space" by (Mikolov et al., 2013).

Word2Vec aims to learn word embeddings by predicting surrounding words based on a given word (Skip-gram) or vice versa (CBOW - Continuous Bag-of-Words) .

Skip-gram: This approach takes a word as input and tries to predict surrounding words within a specific window size in the sentence. Words appearing close together frequently are likely to be semantically similar, and the model learns to represent them with similar vectors .

CBOW: This method takes surrounding words as input and attempts to predict the central word. By analyzing the context words, the model learns to represent words with similar meanings by similar vectors .

Benefits of Word2Vec:

1. Semantic Relationships: Word2Vec captures semantic relationships between words. Words with similar meanings will have similar vector representations. This allows tasks like finding synonyms or identifying words with opposite meanings.

2. Numerical Representation: Words are converted into numerical vectors, enabling efficient processing by machine learning algorithms.

3.4.3 GloVe

GloVe (Global Vectors for Word Representation) is a popular word embedding technique that learns vector representations of words based on their statistical co-occurrence within a large text corpus (Pennington et al., 2014). Unlike Word2Vec, which focuses on predicting surrounding words based on a central word, GloVe leverages the statistical information from word co-occurrence matrices. These matrices capture how frequently words appear together within a specific window size in a text corpus.

Advantages of GloVe:

1. Leverages Statistical Information: GloVe utilizes co-occurrence statistics, which can capture broader semantic relationships compared to local context-based methods like Skip-gram in Word2Vec.

2. Handles Polysemy: GloVe can potentially handle words with multiple meanings (polysemy) by learning separate vector representations based on the word's context .

3. Scalability: Due to its focus on matrix factorization, GloVe can be more scalable for processing large text corpora compared to some neural network-based word embedding methods.

3.4.4 BERT

BERT, or Bidirectional Encoder Representations from Transformers, is a powerful pre-trained model for natural language processing (NLP) tasks. It leverages the Transformer architecture to learn contextual representations of words in a sentence.

Unlike traditional language models that process text sequentially (word by word), BERT utilizes a Transformer-based architecture to analyze words in both directions (bidirectionally) within a sentence. This allows the model to capture the context of each word based on its surrounding words, leading to a deeper understanding of the sentence's overall meaning (Devlin et al., 2018) .

Benefits of BERT:

1. Contextual Word Representations: BERT generates contextualized word embeddings, where the meaning of a word is derived from its surrounding context.

2. State-of-the-Art Performance: BERT has achieved state-of-the-art performance on a wide range of NLP tasks, including question answering, text summarization, and sentiment analysis.

3. Transfer Learning: The pre-trained BERT model can be fine-tuned for various NLP tasks by adding a task-specific output layer, allowing for efficient adaptation to new problems.

3.4.5 SBERT

Sentence-BERT is a powerful model for learning sentence embeddings. It builds upon the success of pre-trained models like BERT (Bidirectional Encoder Representations from Transformers) by fine-tuning them specifically for the task of sentence representation (Reimers and Gurevych, 2019).

Unlike BERT, which focuses on understanding individual words within a sentence, SBERT aims to capture the meaning of an entire sentence and

represent it as a fixed-length vector. This vector encodes the semantic information of the sentence, allowing tasks like sentence similarity analysis or information retrieval.

SBERT leverages pre-trained models like BERT, which have already learned rich representations of words and their contexts through massive amounts of text data. SBERT then fine-tunes these pre-trained models on sentence-level tasks to further refine the sentence embeddings.

Benefits of SBERT:

1. Sentence Embeddings: SBERT provides informative sentence embeddings that capture the overall meaning of a sentence.

2. Improved Performance: Compared to sentence embeddings derived directly from pre-trained models like BERT, SBERT often achieves better performance in sentence-level tasks.

3. Versatility: SBERT embeddings can be applied to various tasks involving sentence similarity analysis, information retrieval, or question answering.

3.5 Classification Models

3.5.1 XGBClassifier

XGBClassifier is a machine learning model from the XGBoost library, known for its effectiveness in various classification tasks. It's an ensemble method based on gradient boosting, which combines multiple weak decision trees into a robust and highly accurate model (Chen and Guestrin, 2016).

Advantages of XGBClassifier:

1. High Accuracy: XGBClassifier frequently achieves state-of-the-art performance on a wide range of classification problems. Flexibility: It can handle various data types, including numerical, categorical, and mixed datasets.

2. Interpretability: While ensemble methods can be less interpretable than simpler models, techniques like feature importance analysis can provide insights into how XGBClassifier makes predictions.

3. Regularization: As mentioned earlier, regularization helps prevent overfitting, leading to better generalization on new data.

3.5.2 LightGBMClassifier

LightGBM, short for Light Gradient Boosting Machine, is a powerful open-source machine learning framework known for its speed, efficiency, and accuracy in various tasks, particularly ranking, classification, and regression. It's based on the concept of gradient boosting, similar to XGBoost, but incorporates several optimizations for efficiency and scalability (Ke et al., 2017).

Advantages of LightGBM:

1. Speed and Efficiency: LightGBM is often praised for its training speed, making it a great choice for large datasets. The optimizations mentioned above contribute significantly to this advantage.

2. Accuracy: LightGBM delivers competitive accuracy on a wide range of machine learning problems.

3. Scalability: It scales efficiently across multiple cores or GPUs, making it suitable for distributed computing environments.

4. Memory Efficiency: The histogram-based approach and feature bundling techniques reduce memory usage, allowing LightGBM to handle larger datasets efficiently.

3.6 Quantization

Quantization[3] is an optimization technique that reduces the size and computational cost of a deep learning model by representing its weights and activation with lower precision (e.g., int8) compared to the standard float32 format. This allows for faster inference and deployment on resource-constrained devices. In this paper, quantization is applied on the BERT (RoBERTa) model and SBERT using PyTorch utilities.

There are two main approaches for PyTorch quantization of RoBERTa:

1. Post-training Static Quantization (PTQ): This involves quantizing a pre-trained RoBERTa model without modifying its training process. It's a simpler approach but might lead to a slight accuracy drop.

2. Quantization-aware Training (QAT): This integrates quantization operations into the training loop, allowing the model to adapt to the lower precision format. QAT can achieve higher accuracy compared to PTQ but requires modifying the training pipeline.

[3] `https://pytorch.org/docs/stable/quantization.html`

Chapter 4

Implementation

4.1 Introduction

Machine learning models are powerful tools, but their success hinges on the quality of the data they are trained on. Before unleashing the learning process, data needs to be carefully prepared and wrangled into a suitable format. This preparation journey involves several crucial steps: data splitting, data analysis, feature engineering, and data preprocessing. Let's delve into each of these steps to understand how they contribute to building a robust foundation for your machine learning model.

4.2 Data Splitting

Before text preprocessing following steps are implemented:-

1. Quora dataset is splitted into test and train dataset.

```
#splitting of data first
df_train,df_test = train_test_split(dframe,
                                    test_size=0.3,
                                    stratify=dframe['is_duplicate'],
                                    random_state=42)

df_train.reset_index(drop=True,inplace=True)
df_test.reset_index(drop=True,inplace=True)

print("Training data shape:",df_train.shape)
print("Test data shape:",df_test.shape)
```

```
Training data shape: (283003, 2)
Test data shape: (121287, 2)
```

Figure 4.1: Quora Dataset splitting

2. Corpora contains a 'Question' column which consists of two questions. The 'Question' column is split into two questions column 'text_question1' and 'text_question2' based on their index. This is done on both test and train datasets.

```
                                        text_question1  \
0       How do I gain healthy weight without eating junk?
1       What is unusual or different about the food an...
2       How can I make music player with sensor in and...
3                  How much can you charge for a website?
4                     How can I treat a swollen clitoris?
...                                                   ...
282998  What is best way for preparing civil services ...
282999              How do I improve my English speaking?
283000  Would you beat a man up if he talked and flirt...
283001  What is the difference between media and liter...
283002               What is corporate events management?

                                        text_question2  is_duplicate
0       What are the healthy ways of gaining weight an...         True
1       What is unusual or different about the food an...        False
2                How can I make music player for android?        False
3                    How much I can charge for a website?         True
4                      How do you treat a swollen tongue?        False
...                                                   ...          ...
282998          Can a graduate crack civil services exam?         True
282999             How I can enhance my English language?         True
283000  Would you beat a woman up if she talks to your...        False
283001  What are some of the main differences between ...        False
283002                            What is corporate event?        False

[283003 rows x 3 columns]
```

Figure 4.2: Quora questions splitting

Data splitting is skipped for the MRPC dataset as it already contains train and test datasets.

4.3 Data Analysis

The process of examining and condensing a dataset in order to get knowledge and comprehension of its primary features is known as data analysis. It involves techniques such as data visualization, feature engineering, and data cleaning to discover patterns and relationships within the data. Since it provides information for further modeling and analysis decisions, EDA is an essential stage in the data analysis process.

4.3.1 Data Visualization

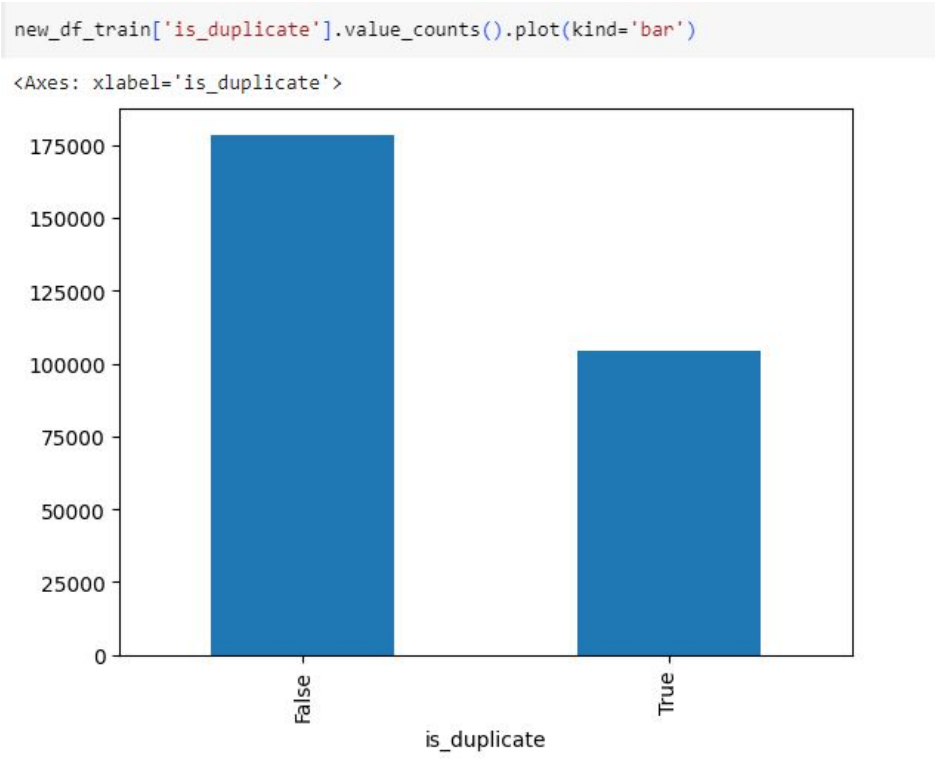


Figure 4.3: Train duplicate

```
new_df_test['is_duplicate'].value_counts().plot(kind='bar')
```

```
<Axes: xlabel='is_duplicate'>
```

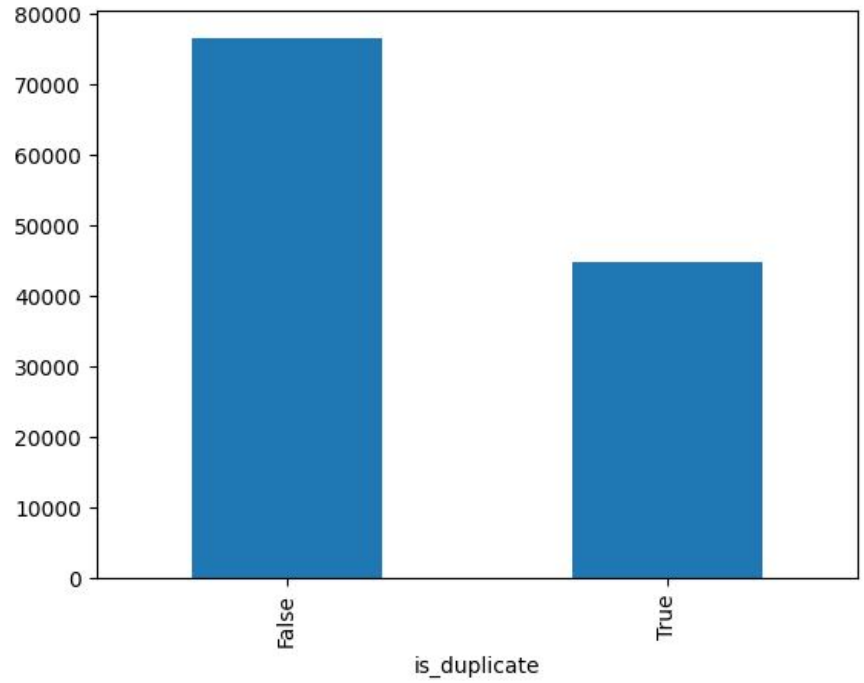


Figure 4.4: Test duplicate

Both the graphs shows that the test and train dataset have unbalanced data with respect to duplicate columns. When training the word embedding models, unbalanced duplicate data can skew the learning process. Models might focus excessively on patterns present in duplicate questions, potentially leading to overfitting. This implies that while the model could work well on training data, it might not generalize well to fresh, untested data. Techniques such as data preprocessing, sampling strategies, and model fine-tuning can help mitigate the impact of unbalanced duplicate data on model performance.

4.3.2 Feature Engineering

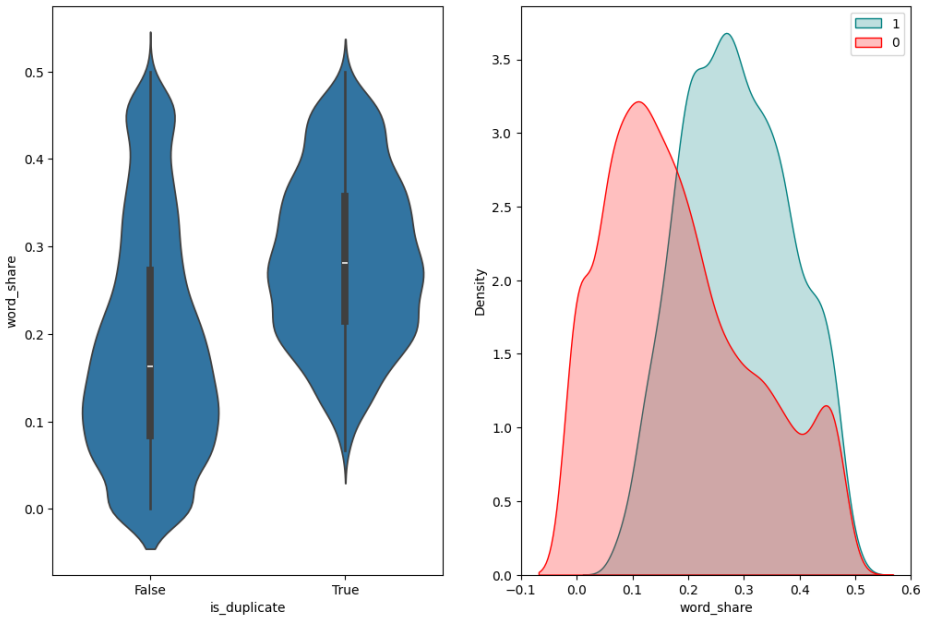


Figure 4.5: Train Word sharing

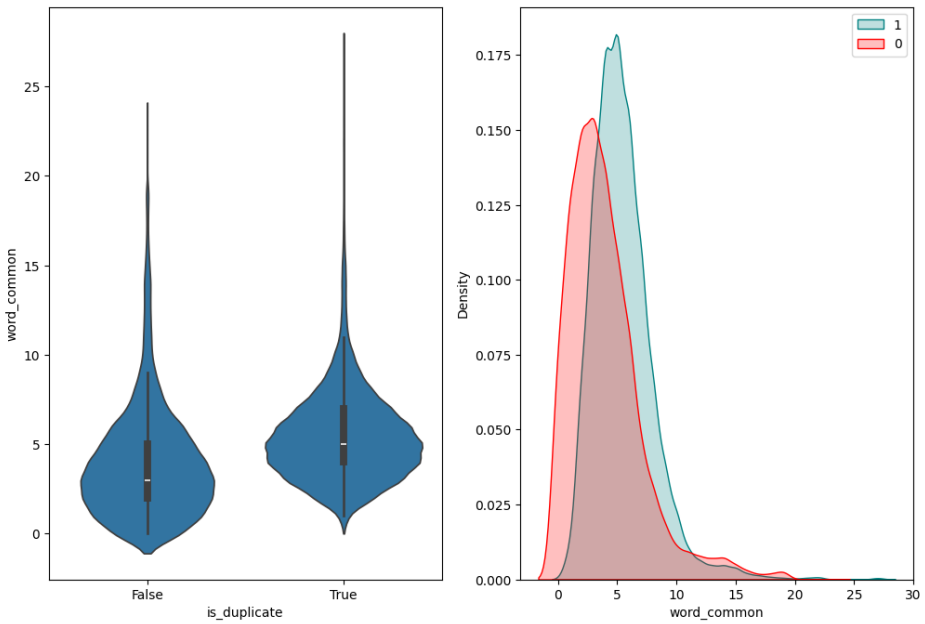


Figure 4.6: Test Word sharing

Both of the plots presented in the analysis suggest that there is significant overlap between the distributions of the 'word_Common' feature in similar and non-similar questions. This means that the 'word_Common' feature alone may not be a strong enough indicator to determine whether two questions are similar or not. It is important to take into account other features as well when building models for identifying duplicate questions, in order to achieve a high level of accuracy.

4.3.3 Data Preprocessing

To maximize the processing impact and remove noise, the test and train datasets must be preprocessed before being fed into the models. Here is a quick rundown of the main preprocessing steps:-

```
#Pre-processing of data
def preprocess_data(text):
    # conver to string
    text = str(text)
    # lowercase
    text = text.lower()
    # remove contractions
    text = contractions.fix(text)
    # remove hashtags
    text = re.sub(r'#(\w+)','',text)
    # remove special characters
    text = re.sub(r'[^\w ]+','',text)
    # remove links if any
    text = re.sub(r'https?://\S+|www\.\S+','',text)
    # remove non-ascii
    text = ''.join(word for word in text if ord(word) < 128)
    # remove punctuation
    text = text.translate(str.maketrans('','',string.punctuation))
    # remove digits
    text = re.sub(r'[\d]+','',text)
    # remove single letters
    text = ' '.join(word for word in text.split() if len(word) > 1)
    # remove multiple spaces
    text = ' '.join(text.split())

    return text
```

Figure 4.7: Data Preprocessing

4.4 Models Implemented

4.4.1 TF-IDF

1. Fit TF-IDF vectorizer of preprocessed train and test dataset.

```
# Fit Tfidf Vectorizer of train & test dataset

tfidf = TfidfVectorizer()
train_tfidf = tfidf.fit_transform(df_preprocess_train['text_question1']+' '+df_preprocess_train['text_question2'])

#no. of feature
test_tfidf = tfidf.transform(df_preprocess_test['text_question1']+' '+df_preprocess_test['text_question2'])

train_tfidf.shape,test_tfidf.shape
```

```
((283003, 82815), (121287, 82815))
```

Figure 4.8: TF-IDF Vectorizer

2. Then use XGBClassifier for prediction on preprocessed train and test dataset. Find accuracy and classification report for both datasets.

```
labels = df_preprocess_train['is_duplicate']
model_xgb = xgb.XGBClassifier()
model_xgb.fit(train_tfidf,labels)

predictions_train=model_xgb.predict(train_tfidf)
predictions_train = list(predictions_train)
print('Train Accuracy score:',accuracy_score(df_preprocess_train['is_duplicate'],predictions_train))
print("Classification Report:\n", classification_report(df_preprocess_train['is_duplicate'],predictions_train))

predictions_test = model_xgb.predict(test_tfidf)
predictions_test = list(predictions_test)


print('Test Accuracy score:',accuracy_score(df_preprocess_test['is_duplicate'],predictions_test))
print("Classification Report:\n", classification_report(df_preprocess_test['is_duplicate'],predictions_test))
```

Figure 4.9: TF-IDF XGBClassifier

3. Use LightGBMClassifier for prediction. Calculate the accuracy and classification report.

```
# Train LightGBM Classifier
model_lgb = lgb.LGBMClassifier()
model_lgb.fit(embeddings,labels)


predictions_test=model_xgb.predict(embeddings)
predictions_test = list(predictions_test)
print('Test Accuracy score:',accuracy_score(sampled_df['is_duplicate'],predictions_test))
print("Classification Report:\n", classification_report(sampled_df['is_duplicate'],predictions_test))
```

Figure 4.10: TF-IDF LightGBMClassifier

4.4.2 Word2Vec

1. Tokenize the sentence contained in preprocessed train dataset

```
# Assuming 'processed_questions' contains preprocessed text
X = df_preprocess_train[['text_question1', 'text_question2']]
y = df_preprocess_train['is_duplicate']

# Tokenize the sentences
tokenized_sentences_q1 = X['text_question1'].apply(lambda x: str(x).split())
tokenized_sentences_q2 = X['text_question2'].apply(lambda x: str(x).split())
```

Figure 4.11: Sentence tokenization on train dataset

2. Train Word2Vec model on tokenized sentences

```
# Train Word2Vec model
word2vec_model_q1 = Word2Vec(sentences=tokenized_sentences_q1, vector_size=100, window=5, min_count=1, workers=4)
word2vec_model_q2 = Word2Vec(sentences=tokenized_sentences_q2, vector_size=100, window=5, min_count=1, workers=4)
```

Figure 4.12: Train Word2Vec model

3. Create vector features of each sentence in the train dataset

```
# Function to get the vector representation of a sentence
def get_sentence_vector(sentence, model):
    vector = np.zeros(model.vector_size)
    count = 0
    for word in sentence:
        if word in model.wv:
            vector += model.wv[word]
            count += 1
    if count != 0:
        vector /= count
    return vector

# Create feature vectors for each question
X_q1 = np.array([get_sentence_vector(sentence, word2vec_model_q1) for sentence in tokenized_sentences_q1])
X_q2 = np.array([get_sentence_vector(sentence, word2vec_model_q2) for sentence in tokenized_sentences_q2])

# Concatenate the feature vectors
X_combined = np.concatenate((X_q1, X_q2), axis=1)
```

Figure 4.13: Feature vectors

4. Use XGBClassifier for prediction, then find accuracy and classification report

```
labels = df_preprocess_test['is_duplicate']
model_xgb = xgb.XGBClassifier()
model_xgb.fit(X_combined,labels)


predictions_test=model_xgb.predict(X_combined)
predictions_test = list(predictions_test)
print('Test Accuracy score:',accuracy_score(df_preprocess_test['is_duplicate'],predictions_test))
print("Classification Report:\n", classification_report(df_preprocess_test['is_duplicate'],predictions_test))
```

Figure 4.14: Word2Vec XGBClassifier

5. Use LightGBMClassifier for prediction. Calculate the accuracy and classification report.

```
# Train LightGBM Classifier
model_lgb = lgb.LGBMClassifier()
model_lgb.fit(X_combined, labels)

# Predictions and evaluation
predictions_train = model_lgb.predict(X_combined)
print('Train Accuracy score:', accuracy_score(labels, predictions_train))
print("Classification Report:\n", classification_report(labels, predictions_train))
```

Figure 4.15: Word2Vec LightGBMClassifier

Similar steps are followed for processed test dataset

4.4.3 SBERT

1. Load pre-trained model

```
#Load pre-trained model
model_name = 'paraphrase-distilroberta-base-v2'
model = SentenceTransformer(model_name)
```

Figure 4.16: Load SBERT model

2. Take sample data from train dataset initially and encode the questions using SBERT model. Then, use XGBClassifier for prediction and calculate accuracy along with classification report.

```
sample_size = 3000  # Adjust as needed
sampled_df = new_df_train.sample(n=sample_size, random_state=52)

# Encode questions using SBERT model
question1_embeddings = model.encode(sampled_df['text_question1'].tolist())
question2_embeddings = model.encode(sampled_df['text_question2'].tolist())


X_combined = np.concatenate((question1_embeddings, question2_embeddings), axis=1)

# Generate labels for the sampled data
labels = sampled_df['is_duplicate'].values


model_xgb = xgb.XGBClassifier()
model_xgb.fit(X_combined,labels)


predictions_train=model_xgb.predict(X_combined)
predictions_train = list(predictions_train)
print('Train Accuracy score:',accuracy_score(sampled_df['is_duplicate'],predictions_train))
print("Classification Report:\n", classification_report(sampled_df['is_duplicate'],predictions_train))
```

Figure 4.17: SBERT XGBClassifier

3. Use LightGBMClassifier for prediction. Calculate the accuracy and classification report.

```
# Train LightGBM Classifier
model_lgb = lgb.LGBMClassifier()
model_lgb.fit(X_combined, labels)

# Predictions and evaluation
predictions_train = model_lgb.predict(X_combined)
print('Train Accuracy score:', accuracy_score(labels, predictions_train))
print("Classification Report:\n", classification_report(labels, predictions_train))
```

Figure 4.18: SBERT LightGBMClassifier

Same steps are followed for test dataset.Preprocessing is not required for SBERT model.

4.4.4 GLoVe

1. Load Spacy for tokenization

```
nlp = spacy.load("en_core_web_md")
```

Figure 4.19: Load Spacy

2. Load GloVe embeddings 'GloVe.6B' with dimensions 100.This is one of the most commonly used variants of GloVe embeddings. It was trained on a corpus of Wikipedia 2014 and Gigaword 5 data, resulting in embeddings with 6 billion tokens and a vocabulary of 400,000 words.

```
def load_glove_embeddings(file_path):
    embeddings_index = {}
    with open(file_path, 'r', encoding='utf-8') as f:
        for line in f:
            values = line.split()
            word = values[0]
            vector = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = vector
    return embeddings_index

glove_embeddings = load_glove_embeddings('/content/drive/My Drive/glove.6B.100d.txt')
```

Figure 4.20: GloVe embeddings

3. Take sample data and embed questions in preprocessed train dataset

```
def embed_question(question, model):
    tokens = [token.vector for token in model(question)]
    return np.mean(tokens, axis=0)

sample_size = 1000  # Adjust as needed
sampled_df = df_preprocess_train.sample(n=sample_size, random_state=42)
# Function to calculate cosine similarity between two embeddings
def cosine_similarity(embedding1, embedding2):
    return np.dot(embedding1, embedding2) / (np.linalg.norm(embedding1) * np.linalg.norm(embedding2))

    # Preprocess the data and create embeddings for each question pair
embeddings = []
labels = []
for idx, row in sampled_df.iterrows():
    q1_embedding = embed_question(row['text_question1'], nlp)
    q2_embedding = embed_question(row['text_question2'], nlp)
    combined_embedding = np.abs(q1_embedding - q2_embedding)  # Combine embeddings
    embeddings.append(combined_embedding)
    labels.append(row['is_duplicate'])
```

Figure 4.21: Questions embedding

4. Use XGBClassifier for prediction. Calculate the accuracy and classification report.

```
model_xgb = xgb.XGBClassifier()
model_xgb.fit(embeddings,labels)


predictions_train=model_xgb.predict(embeddings)
predictions_train = list(predictions_train)
print('Train Accuracy score:',accuracy_score(sampled_df['is_duplicate'],predictions_train))
print("Classification Report:\n", classification_report(sampled_df['is_duplicate'],predictions_train))
```

Figure 4.22: GloVe XGBClassifier

5. Use LightGBMClassifier for prediction. Calculate the accuracy and classification report.

```
# Train LightGBM Classifier
model_lgb = lgb.LGBMClassifier()
model_lgb.fit(embeddings,labels)


predictions_test=model_xgb.predict(embeddings)
predictions_test = list(predictions_test)
print('Test Accuracy score:',accuracy_score(sampled_df['is_duplicate'],predictions_test))
print("Classification Report:\n", classification_report(sampled_df['is_duplicate'],predictions_test))
```

Figure 4.23: GloVe LightGBMClassifier

Same steps are followed for test dataset

4.4.5 BERT

1. Load BERT tokenizer

```
# Load the pre-trained RoBERTa tokenizer and model
tokenizer = RobertaTokenizer.from_pretrained("roberta-base")
roberta_model = RobertaModel.from_pretrained("roberta-base")
```

Figure 4.24: BERT tokenizer

2. Configure the model for quantization

```
# Configure the model for quantization
roberta_model = roberta_model.eval()  # Put the model in evaluation mode
roberta_model = torch.quantization.quantize_dynamic(roberta_model, {torch.nn.Linear}, dtype=torch.qint8)
```

Figure 4.25: Quantization

3. Take sample data from train dataset initially and tokenize questions using RoBERTa tokenizer.Encode the questions using RoBERTa model with reduced batch size.

```
# Sample size and load data
sample_size = 3000  # Adjust as needed
sampled_df = new_df_train.sample(n=sample_size, random_state=52)

# Tokenize questions using RoBERTa tokenizer
question1_tokens = tokenizer(sampled_df['text_question1'].tolist(), padding=True, truncation=True, return_tensors='pt', max_length=128)
question2_tokens = tokenizer(sampled_df['text_question2'].tolist(), padding=True, truncation=True, return_tensors='pt', max_length=128)

# Encode questions using RoBERTa model with reduced batch size
batch_size = 8
question1_embeddings = []
question2_embeddings = []

with torch.no_grad():  # Disable gradient tracking for efficiency
    for i in range(0, len(sampled_df), batch_size):
        batch_question1_tokens = {k: v[i:i+batch_size] for k, v in question1_tokens.items()}
        batch_question2_tokens = {k: v[i:i+batch_size] for k, v in question2_tokens.items()}

        # Encode questions using RoBERTa model
        batch_question1_outputs = roberta_model(**batch_question1_tokens)
        batch_question2_outputs = roberta_model(**batch_question2_tokens)

        # Extract pooled output (CLS token)
        batch_question1_embeddings = batch_question1_outputs.pooler_output
        batch_question2_embeddings = batch_question2_outputs.pooler_output

        question1_embeddings.append(batch_question1_embeddings)
        question2_embeddings.append(batch_question2_embeddings)
```

Figure 4.26: Tokenize questions

4. Concatenate the questions embeddings and use XGBClassifier to calculate the accuracy & Classification Report.

```
# Concatenate question embeddings
question1_embeddings = torch.cat(question1_embeddings, dim=0)
question2_embeddings = torch.cat(question2_embeddings, dim=0)
X_combined = torch.cat([question1_embeddings, question2_embeddings], dim=1).numpy()

# Generate labels for the sampled data
labels = sampled_df['is_duplicate'].values

# Train Gradient Boosting Classifier
model_xgb = xgb.XGBClassifier()
model_xgb.fit(X_combined, labels)

# Predictions and evaluation
predictions_train = model_xgb.predict(X_combined)
print('Train Accuracy score:', accuracy_score(labels, predictions_train))
print("Classification Report:\n", classification_report(labels, predictions_train))
```

Figure 4.27: BERT XGBClassifier

5. Use LightGBMClassifier for prediction. Calculate the accuracy and classification report.

```
# Train LightGBM Classifier
model_lgb = lgb.LGBMClassifier()
model_lgb.fit(X_combined, labels)

# Predictions and evaluation
predictions_train = model_lgb.predict(X_combined)
print('Train Accuracy score:', accuracy_score(labels, predictions_train))
print("Classification Report:\n", classification_report(labels, predictions_train))
```

Figure 4.28: BERT LightGBMClassifier

Initially, sample records are tested for the Quora dataset then final results are evaluated based on the original corpora.

Chapter 5

Evaluation

5.1 Introduction

Building a machine learning model is only half the battle. Once you've trained your model, you need a way to assess its effectiveness. Here's where evaluation metrics come in. These metrics provide crucial insights into how well the model performs on unseen data. This introduction dives into four key metrics used in classification tasks: Accuracy, Precision, Recall, and F1-Score.

With Python's sklearn package, the measurements can be computed. The scores are provided by the library in two main categories: weighted-avg and macro-avg. In order to account for the magnitude of the positive and negative samples, we have considered weighted averages in our analysis. Understanding these metrics will equip us to analyze your model's strengths and weaknesses, ultimately guiding you toward building robust and reliable models.

5.2 Accuracy

5.2.1 Quora

Train and test accuracy results are specified in the 5.1 for each model using Quora corpora with XGBClassifier & LightGBMClassifier. The lowest accuracy is observed for the GloVe model i.e. 74% for train data and 70% for test data for XGBClassifier whereas it is 69% for train and test data using LightGBMClassifier. As compared to GloVe, the TF-IDF model improves the accuracy by 2% (76%) increase for train data & 6% (75%) increase for test data using XGBClassifier whereas, it rises by 5% (70%) for train data &

the test data (74%) using LightGBMClassifier.Further, the Word2Vec model performed well by hiking the accuracy upto 81% in train data and 77% in the test data using XGBClassifier and 76% in train data and 75% in the test data using LightGBMClassifier. The highest accuracy is provided by SBERT and BERT models using both classifiers with a maximum 86% accuracy with quantization and 88% accuracy without quantization. This shows that the model performance is affected by quantization as the accuracy reduces after quantization.

	Accuracy			
	XGBClassifier		LightGBMClassifier	
Model	Train	Test	Train	Test
TF-IDF	0.76	0.75	0.75	0.74
Word2Vec	0.81	0.77	0.76	0.75
SBERT	0.88	0.85	0.84	0.83
SBERT (Quantization)	0.86	0.83	0.81	0.81
GloVe	0.74	0.69	0.70	0.69
BERT	0.88	0.85	0.84	0.83
BERT(Quantization)	0.86	0.83	0.81	0.81

Figure 5.1: Quora accuracy

5.2.2 MRPC

Train and test accuracy results are specified in the 5.2 for each model using MRPC corpora with XGBClassifier & LightGBMClassifier.The accuracy results using both the classifiers are different for the test and train datasets for the TF-IDF model only and constant for the remaining models. The lowest accuracy is 75% for train and test data using LightGBMClassifier. In contrast, XGBClassifier's performance is 95% for test & train data. The highest accuracy is provided by Word2Vec, GloVe, SBERT, and BERT models using both classifiers with 100% accuracy. SBERT & BERT with Quantization also resulted in 100% accuracy.In this case, Quantization results are the same.

	Accuracy			
	XGBClassifier		LightGBMClassifier	
Model	Train	Test	Train	Test
TF-IDF	0.95	0.95	0.75	0.75
Word2Vec	1.00	1.00	1.00	1.00
SBERT	1.00	1.00	1.00	1.00
SBERT (Quantization)	1.00	1.00	1.00	1.00
GloVe	1.00	1.00	1.00	1.00
BERT	1.00	1.00	1.00	1.00
BERT(Quantization)	1.00	1.00	1.00	1.00

Figure 5.2: MRPC accuracy

5.2.3 Comparison with other paper

Catboost and XGBoost are used to find the accuracy in the paper (Chandra and Stefanus, 2020) for Count vectorizer and TF-IDF on Quora question pairs. As compared to the XGBClassifier, the accuracy percentage is less. There is a slight difference in accuracy between using LightGBMClassifier & CatBoost.This proves that XGBClassifier and LightGBMClassifier are best for determining the accuracy of Quora question pairs.

Methods	Accuracy
CV-XGBoost	68.09
CV-CatBoost	74.66
TF-IDF-XGBoost	69.14
TF-IDF CatBoost	**75.39**

Figure 5.3: Accuracy using Catboost & XGBoost

5.3 Precision, Recall, and F1-Score

5.3.1 Quora

Precision, Recall, and F1-scores are calculated for each model with MRPC test and train datasets using XGBClassifier and LightGBMClassifier 5.5. The

weighted averages of SBERT and BERT are the highest i.e. 88%. In contrast, GloVe has the lowest weighted average of 74% precision , recall, and F1-sore for train data & approx 69% precision, recall and F1-sore for test data using XGBClassifier. The result is slightly different than the LightGBMClassifier. LightGBMClassifier showed less percentage of precison, recall and F1-score ranging between 68% to 70%. The second lowest weighted average is calculated for the TF-IDF model with percentages ranging from 73% to 77%. There is an increase of performance for the Word2Vec model with 81% precision, recall, F1-score as compared to TF-IDF for train dataset using XGBClassifier.

	XGBClassifier						LightGBMClassifier					
	Precision		Recall		F1 Score		Precision		Recall		F1 Score	
Model	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
TF-IDF	0.76	0.75	0.77	0.75	0.76	0.74	0.75	0.74	0.75	0.75	0.74	0.73
Word2Vec	0.81	0.77	0.81	0.77	0.81	0.77	0.76	0.75	0.76	0.75	0.76	0.75
SBERT	0.88	0.85	0.88	0.85	0.88	0.85	0.84	0.83	0.84	0.83	0.84	0.83
SBERT (Quantization)	0.86	0.83	0.86	0.83	0.86	0.83	0.81	0.80	0.81	0.81	0.81	0.80
GloVe	0.74	0.68	0.74	0.69	0.74	0.69	0.69	0.68	0.70	0.69	0.69	0.68
BERT	0.88	0.85	0.88	0.85	0.88	0.85	0.84	0.83	0.84	0.83	0.84	0.83
BERT (Quantization)	0.86	0.83	0.86	0.83	0.86	0.83	0.81	0.80	0.81	0.81	0.81	0.80

Figure 5.4: Precision, Recall & F1-Score for Quora dataset

5.3.2 MRPC

Precision, Recall, and F1-scores are calculated for each model with MRPC test and train datasets using XGBClassifier and LightGBMClassifier 5.5. The weighted averages of Word2Vec,GloVe,SBERT and BERT are the highest (100%). Achieving perfect precision, recall, and F1 score suggests that these models have learned the underlying patterns in the data extremely well. It can make predictions with complete accuracy, with no false positives or false negatives. TF-IDF has the lowest weighted averages. It is approximately 74% using LightGBMClassifier and 95% using XGBClassifier.

	XGBClassifier						LightGBMClassifier					
	Precision		Recall		F1 Score		Precision		Recall		F1 Score	
Model	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
TF-IDF	0.95	0.95	0.95	0.95	0.95	0.95	0.75	0.74	0.75	0.75	0.74	0.73
Word2Vec	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
SBERT	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
SBERT (Quantization)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
GloVe	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
BERT	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
BERT(Quantization)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

Figure 5.5: Precision, Recall & F1-Score for MRPC dataset

Overall, it is observed that the models with best performances are BERT,

SBERT, Word2Vec and GloVe to determine the paraphrases for plagiarism detection for small corpus like MRPC. For large corpus like Quora, SBERT and BERT performed the best.Refer python notebook for MRPC [1] and Quora [2] corpora.

[1] https://github.com/banibhatia/msc_dissertation/blob/main/Embedding_dissertation_MRPC.ipynb

[2] https://github.com/banibhatia/msc_dissertation/blob/main/Embedding_dissertation_Quora_Results_Glove_SBERT.ipynb

Chapter 6

Conclusions

6.1 Main Conclusions

In this research, many statistical word embedding models, including TF-IDF, Word2Vec, SBERT, BERT, and GloVe, and corpus-based deep learning models are proposed to be compared. The two public corpora, MRPC and Quora, are used to test the models. Understanding how different word embedding algorithms perform on the aforementioned corpora and how preprocessing procedures affect assessment metrics is made easier with the aid of this research project. It is concluded that Word2Vec, BERT, SBERT, and GloVe produce the most accurate and efficient results with 100% accuracy, precision, recall, and F1-score for MRPC dataset.SBERT and BERT performed the best with maximum 88% accuracy, precision, recall, and F1-score for the Quora dataset. SBERT and BERT with Quantization in the MRPC dataset performed equally as SBERT and BERT without Quantization whereas in the Quora dataset it has underperformed. This result states that Quantization is not useful in improving the performance in a larger dataset like Quora.

Using XGBClassifier and LightGBMClassifier on each model displays the almost same results for the MRPC dataset, but there are slight differences between the readings for the Quora dataset. The most effective classification model is the XGBClassifier as an outcome due to its performance better than the LightGBMClassifier. It may also be inferred from the experimental findings that other techniques, like negative sampling, label noise reduction, dissimilarity measures, and so on, can be used to improve the performance of standalone deep learning models.

6.2 Delivery against objectives

This research is successfully carried out against its objectives. Publicly available corpora 'Quora' and 'MRPC' are used to find out the semantic similarity and best models out of five word embedding models are identified based on the evaluation metrices.

6.3 Future work

Techniques to identify different kinds of paraphrases might be suggested for further research.Further study on additional cutting-edge models, such as ELMO, USE, GPT-3, etc., can be conducted in the future. For comparing conventional models with cutting-edge versions, LSI can also be taken into account. Detecting the type of paraphrase that has been added or deleted can be a fascinating addition to the variety of procedures that has been suggested. Additionally, a system that integrates many methods for paraphrase type identification may be created to identify a wide variety of paraphrase kinds.

6.4 Personal reflection

This research project delving into paraphrase identification for plagiarism detection using various word embedding models like TF-IDF, SBERT, BERT, GloVe, and Word2Vec has been an enriching exploration of NLP's potential in academic integrity. One of the initial challenges involved understanding the strengths and weaknesses of each word embedding model. Comparing and contrasting Word2Vec, TF-IDF's statistical approach with the contextual power of GloVe, SBERT and BERT required a comprehensive analysis. This process not only deepened my knowledge of these models but also highlighted the importance of selecting the right tool for the job.

References

Altheneyan, A. and Menai, M. E. B. (2020). Evaluation of state-of-the-art paraphrase identification and its application to automatic plagiarism detection. *International Journal of Pattern Recognition and Artificial Intelligence*, 34(04):2053004.

Alvi, F., Stevenson, M., and Clough, P. (2021). Paraphrase type identification for plagiarism detection using contexts and word embeddings. *International Journal of Educational Technology in Higher Education*, 18(1):42.

Bohra, A. and Barwar, N. (2022). A deep learning approach for plagiarism detection system using bert. In *Congress on Intelligent Systems: Proceedings of CIS 2021, Volume 2*, pages 163–174. Springer.

Chandra, A. and Stefanus, R. (2020). Experiments on paraphrase identification using quora question pairs dataset. *arXiv preprint arXiv:2006.02648*.

Chawla, S., Aggarwal, P., and Kaur, R. (2022). Comparative analysis of semantic similarity word embedding techniques for paraphrase detection. In *Emerging Technologies for Computing, Communication and Smart Cities: Proceedings of ETCCS 2021*, pages 15–29. Springer.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Gangadharan, V., Gupta, D., Amritha, L., and Athira, T. (2020). Paraphrase detection using deep neural network based word embedding techniques. In *2020 4th International Conference on Trends in Electronics and Informatics (ICOEI)(48184)*, pages 517–521. IEEE.

Ilyas, M., Malik, N., Bilal, A., Razzaq, S., Maqbool, F., and Abbas, Q. (2021). Plagiarism detection using natural language processing techniques. *Technical Journal*, 26(01):90–101.

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T.-Y. (2017). Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30.

Kim, S.-W. and Gil, J.-M. (2019). Research paper classification systems based on tf-idf and lda schemes. *Human-centric Computing and Information Sciences*, 9:1–21.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Viji, D. and Revathy, S. (2022). A hybrid approach of weighted fine-tuned bert extraction with deep siamese bi–lstm model for semantic text similarity identification. *Multimedia Tools and Applications*, 81(5):6131–6157.

Vrbanec, T. and Meštrović, A. (2020). Corpus-based paraphrase detection experiments and review. *Information*, 11(5):241.

Vrbanec, T. and Meštrović, A. (2023). Comparison study of unsupervised paraphrase detection: Deep learning—the key for semantic similarity detection. *Expert systems*, 40(9):e13386.

REFERENCES