Design and Implementation of Quora Question Pair Similarity

by

Md Sujon Islam
ID: CSE2001019270

Md Jahidujaman Joy
ID: CSE2001019084

Sumiya Akter Sume
ID: CSE2002020030

Plabon Roy
ID: CSE2002020029

Supervised by
Bulbul Ahamed

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SONARGAON UNIVERSITY (SU)

May 2024

APPROVAL

The project titled "**Design and Implementation of Quora Question Pair Similarity**" submitted by Md. Sujon Islam (CSE2001019270), Md Jahidujaman Joy (CSE2001019084), Sumiya Akter Sume (CSE2002020030) and Plabon Roy (CSE2002020029) to the Department of Computer Science and Engineering, Sonargaon University (SU), has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents.

Board of Examiners

Bulbul Ahamed — **Supervisor**
Professor and Head
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) — **Examiner 1**
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) — **Examiner 2**
Department of Computer Science and Engineering
Sonargaon University (SU)

(Examiner Name and Signature) — **Examiner 3**
Department of Computer Science and Engineering
Sonargaon University (SU)

DECLARATION

We, hereby, declare that the work presented in this report is the outcome of the investigation performed by us under the supervision of **Bulbul Ahamed**, Professor and Head, Department of Computer Science and Engineering, Sonargaon University, Dhaka, Bangladesh. We reaffirm that no part of this project has been or is being submitted elsewhere for the award of any degree or diploma.

Countersigned

(Bulbul Ahamed)
Supervisor

Signature

Md Sujon Islam
ID: CSE2001019270

Md Jahidujaman Joy
ID: CSE2001019084

Sumiya Akter Sume
ID: CSE2002020030

Plabon Roy
ID: CSE2002020029

ABSTRACT

The “Design and Implementation of Quora Question Pair Similarity” project endeavors to address the challenge of identifying duplicate questions on the Quora platform to enhance user experience and content quality. Duplicate questions, when unanswered, can lead to redundancy and clutter, hindering effective knowledge sharing. Leveraging advanced natural language processing (NLP) techniques, our project focuses on developing a robust similarity model capable of accurately determining the semantic similarity between pair questions. By employing state-of-the-art machine learning algorithms and deep neural networks, we aim to create a system that not only identifies duplicate questions but also provides insights into the underlying linguistic structures and contexts. Through rigorous experimentation and evaluation, we strive to achieve high precision and recall rates, ensuring reliable duplicate detection while minimizing false positives. The outcomes of this project hold significant implications for online community platforms, facilitating smoother navigation, improved search functionality, and ultimately, a more enriching user experience.

ACKNOWLEDGMENT

At the very beginning, we would like to express my deepest gratitude to the Almighty Allah for giving us the ability and the strength to finish the task successfully within the schedule time.

We are auspicious that we had the kind association as well as supervision of **Bulbul Ahamed, Professor and Head,** Department of Computer Science and Engineering, Sonargaon University whose hearted and valuable support with best concern and direction acted as necessary recourse to carry out our project.

We are also thankful to all our teachers during our whole education, for exposing us to the beauty of learning.

Finally, our deepest gratitude and love to my parents for their support, encouragement, and endless love.

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
BiLSTM	Bidirectional Long Short Term Memory
CNN	Convolutional Neural Network
DL	Deep Learning
MAE	Mean Absolute Error
MSE	Mean Squared Error
NLP	Natural Language Processing
NLTK	Natural Language Toolkit
NER	Named Entity Recognition
ROC	Receiver Operating Characteristic
ST	Streamlit
TFIDF	Term Frequency Inverse Document Frequency
t-SNE	t-Distributed Stochastic Neighbor Embedding

TABLE OF CONTENTS

Title	Page No.
**DECLARATION .....……..…….. **	iii
**ABSTRACT …….....….....…… **	iv
**ACKNOWLEDGEMENT ……...……. **	v
**LIST OF ABBREVIATION …….…….... **	vi
CHAPTER 1	1
INTRODUCTION	
1.1 Background.....	1
1.2 Problem Statement.....	1
1.3 Objectives.....	1
CHAPTER 2	2-5
LITERATURE REVIEW	
2.1 Similarity Measurement Techniques.....	2
2.2 Machine Learning and Deep Learning in NLP.....	2-3
2.3 Previous Studies on Duplicate Question Detection.....	3-5
CHAPTER 3	6-19
DATA COLLECTION AND PREPROCESSING	
3.1 3.1 Data Source.....	6
3.2 Exploratory Data Analysis.....	6-7
3.2.1 Distribution Data Plot.....	8
3.2.2 Pie Plot Question Similarity.....	9
3.2.3 Quique & Repeated question plot.....	10
3.2.4 Ber Plot on Duplicate Question Pairs.....	11
3.2.5 Ber Plot on Frequency of Questions.....	11
3.2.6 Total Common Word of Questions.....	12

3.2.7 Total Word Share of Questions..... 13
3.2.8 Total Word of Questions..... 15
3.3 Basic Feature Extraction. 16
3.4 Preprocessing Techniques..... 17-19
3.5 Word Cloud Visualization..... 19

CHAPTER 4 20-23
FEATURE ENGINEERING
4.1 Length-Based Features..... 20
4.2 Word-Embedding..... 20-21
4.2.1 Word-Embedding using Glove..... 21-22
4.3 t-SNE Visualization..... 22-23

CHAPTER 5 24-35
MACHINE LEARNING APPROACH
5.1 Naive Bayes Classification..... 24-27
5.2 Decision Tree Classification..... 27-32
5.3 Linear SVM..... 32-35

CHAPTER 6 36
DEEP LEARNING APPROACH
6.1 CNN model architecture..... 36-40
6.2 Bidirectional LSTM architecture..... 40-44
6.3 Model Evaluation Metrics..... 44-46

CHAPTER 7 47
Deployment 47-49
CHAPTER 8 50
Conclusion, Limitations & Future Works 50
8.1 Conclusion..... 50
8.2 Limitations..... 50
8.3 Future Works..... 50-51
References 52

LIST OF TABLES

Table No.	Title	Page No.
Table 1.1	Classification Report	25
Table 1.2	Classification Report	30
Table 1.3	Classification Report	33
Table 1.4	Classification Report	38
Table 1.5	Classification Report	42
Table 1.1	Confusion Matrics	26
Table 1.2	Confusion Matrics	31
Table 1.3	Confusion Matrics	35
Table 1.4	Confusion Matrics	39
Table 1.5	Confusion Matrics	44

LIST OF FIGURES

Figure No.	Title	Page No.
Fig.3.1	Distribution plot	8
Fig.3.2	Pie plot pair similarity	9
Fig.3.3	Unique Question bar plot	10
Fig.3.4	Bar plot distribution question	11
Fig.3.5	Ber plot Frequency question	11
Fig.3.6	Total Common Word of questions	12
Fig.3.7	Total Word Share of questions	13
Fig.3.8	Violin Plot of Total Word of questions	15
Fig.3.9	Preprocessing Flowchart	17
Fig.3.10	Preprocessing Technique Diagram	18
Fig.11	Word Cloud Visualization	19
Fig. 4.1	Extraction Features from BOW	20
Fig.4.2	Embedding Sentence Technique	21
Fig.4.3	Understanding Embedding Using Scatter plot	21
Fig.4.4	Feature Extraction and t-SNE model	22
Fig.4.5	3 Visualization using t-SNE	23
Fig.5.1	Naive bayes model	24
Fig.5.2	Text Classification naive bayes diagram	24
Fig.5.3	Decision tree classifier	28
Fig.5.4	Decision tree model architecture	29
Fig.5.5	Linear SVM model with CNN architecture	33
Fig.6.1	CNN Model Architecture	37
Fig.6.2	BiLSTM Architecture	42
Fig.6.3	Model Evaluation Process	45
Fig.7.1	Web Application	48

CHAPTER 1
Introduction

1.1 Background

Imagine Quora as a big library where people ask questions and share knowledge. Sometimes, though, people ask the same question more than once, cluttering up the library shelves. This can make it hard for others to find the answers they need. Our project is like a librarian—it helps tidy up the shelves by spotting these duplicate questions.

1.2 Problem Statement

The main thing we're tackling is figuring out which questions are duplicates. When questions are the same, it's like having two books with the same story on the shelf. It confuses people and wastes their time. So, our job is to build something that can spot these duplicate questions automatically.

1.3 Objectives

Our goal is to make the Quora Website functionality based systems in a better place by building a smart system. This system will learn from lots of examples to tell if two questions are the same or not. We'll use fancy math and computer tricks to teach our system. Ultimately, we want to help keep organizers and make it easier for people to find the answers they're looking for.

CHAPTER 2

Literature Review

2.1 Similarity Measurement Techniques

When we're trying to figure out if two questions are similar, we need some clever ways to measure their similarity. Think of it like comparing two recipes to see if they're for the same dish. In our project, we'll explore different methods for measuring how alike questions are.

2.2 ML and DL in NLP

Machine Learning (ML) and Deep Learning (DL) play crucial roles in Natural Language Processing (NLP), offering powerful techniques for understanding and processing human language. Here's how ML and DL are applied in NLP:

Machine Learning in NLP:

1. Text Classification: ML algorithms such as Support Vector Machines (SVM), Naive Bayes, and Random Forests are used for tasks like sentiment analysis, spam detection, and topic classification.
2. Named Entity Recognition (NER): ML models are trained to recognize and classify entities such as names of people, organizations, and locations in text.
3. Part-of-Speech Tagging (POS): ML algorithms assign grammatical categories (e.g., noun, verb, adjective) to each word in a sentence.
4. Word Sense Disambiguation: ML techniques help determine the correct meaning of a word based on its context in a sentence.
5. Machine Translation: ML models, including statistical and neural machine translation systems, are used to translate text from one language to another.
6. Information Retrieval: ML-based algorithms rank documents or web pages based on their relevance to a user's query.

Deep Learning in NLP:

1. Word Embeddings: DL models like Word2Vec, GloVe, and Fast Text learn dense vector representations of words that capture semantic relationships between them.
2. Sequence-to-Sequence Models: Architectures like Recurrent Neural Networks (RNNs) and Transformer-based models (e.g., BERT, GPT) are used for tasks like machine translation, text summarization, and conversational agents.
3. Named Entity Recognition: DL models, especially BiLSTM-CRF (Bidirectional LSTM with Conditional Random Fields), achieve state-of-the-art performance in NER tasks.
4. Sentiment Analysis: DL models, particularly Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), are used for sentiment analysis tasks, capturing complex patterns in text.
5. Text Generation: DL models like Generative Pre-trained Transformer (GPT) generate coherent and contextually relevant text, enabling applications such as chatbots and content generation.
6. Language Understanding: Pre-trained language models like BERT (Bidirectional Encoder Representations from Transformers) and its variants provide deep contextual understanding of text, improving performance on various NLP tasks like question answering and text classification.

Both ML and DL techniques continue to advance the field of NLP, enabling a wide range of applications such as virtual assistants, search engines, recommendation systems, and more. The choice between ML and DL approaches often depends on factors such as the complexity of the task, the availability of labeled data, and the desired level of performance.[5]

2.3 Previous Studies on Duplicate Question Detection

Previous studies on duplicate question detection have explored various approaches and techniques to identify and classify pairs of questions that convey similar meanings. Here are some key findings and methodologies from previous research:

1. Feature-Based Approaches: Early studies often relied on handcrafted features such as word overlap, edit distance, and syntactic similarity to measure the similarity

between question pairs. These features were used as input to traditional machine learning algorithms like SVMs and Random Forests for classification.

2. Semantic Similarity Measures: Researchers have explored techniques to capture semantic similarity between questions using distributional semantics models like Word2Vec, GloVe, and FastText. These models learn dense vector representations of words and phrases based on their co-occurrence patterns in large text corpora, enabling more nuanced comparison of question semantics.
3. Neural Network Architectures: With the advent of deep learning, neural network architectures have gained prominence in duplicate question detection. Models such as Siamese networks, Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and their variants (e.g., LSTM, GRU) have been employed to learn representations of question pairs and classify them as duplicate or non-duplicate.
4. Attention Mechanisms: Attention mechanisms have been incorporated into neural network architectures to dynamically weigh the importance of different words or phrases in question pairs. This allows models to focus on relevant parts of the input and improve performance in capturing subtle semantic similarities.
5. Ensemble Methods: Some studies have explored ensemble methods, combining predictions from multiple models or feature sets to improve overall performance. Ensemble techniques help mitigate the limitations of individual models and enhance robustness.
6. Transfer Learning: Transfer learning techniques, such as fine-tuning pre-trained language models (e.g., BERT, RoBERTa), have shown promising results in duplicate question detection. These models, pre-trained on large-scale text corpora, capture rich contextual information and semantic relationships, leading to significant performance gains.
7. Evaluation Metrics: Evaluation of duplicate question detection systems typically involves metrics such as accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC). These metrics assess the model's ability to correctly identify duplicate question pairs while minimizing false positives and false negatives.

Overall, previous studies have demonstrated the effectiveness of various approaches in duplicate question detection, with deep learning techniques often outperforming traditional methods. However, challenges such as data sparsely, class imbalance, and domain-specific nuances continue to present opportunities for further research and innovation in this field.[1]

CHAPTER 3

Data Collection and Preprocessing

3.1 Data Source

The dataset used in this thesis paper was obtained from **Kaggle** as part of the Quora Questions competition. Kaggle is a platform for data science competitions, where participants compete to develop the most accurate predictive models for various tasks.

3.2 Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a crucial step in understanding the characteristics and patterns present in a dataset. In the context of duplicate question detection, EDA helps uncover insights about the distribution of question pairs, the prevalence of duplicates, and the relationships between different features. Here's how EDA is typically conducted for this task:

1. Data Overview: Begin by loading the dataset and examining its basic properties such as the number of samples, features, and data types. This initial inspection helps understand the structure of the dataset and identify any missing values or inconsistencies.
2. Distribution of Labels: Analyze the distribution of duplicate and non-duplicate question pairs in the dataset. Plotting a histogram or bar chart of the label distribution provides insights into the class imbalance and guides the selection of appropriate evaluation metrics.
3. Text Analysis: Explore the characteristics of the text data by examining the distribution of question lengths, word frequencies, and vocabulary size. Visualizations such as word clouds, frequency distributions, and box plots can help uncover common patterns and outliers in the text.
4. Feature Engineering: Investigate potential features that may capture semantic similarity or differences between question pairs. This may include features such as word overlap, syntactic similarity, and semantic embeddings. Explore the

relationship between these features and the target variable (duplicate/non-duplicate) using scatter plots, correlation matrices, or heatmaps.

5. Class Imbalance Handling: If the dataset exhibits class imbalance, consider strategies such as oversampling, under sampling, or class-weighted loss functions to mitigate its impact on model training.
6. Correlation Analysis: Examine the correlation between different features and the target variable. Identify features that are highly correlated with the target and those that exhibit multicollinearity, which may affect the stability of the model.
7. Outlier Detection: Detect and investigate outliers in the dataset that may affect the performance of the model. Outliers can be identified using statistical methods such as z-score, box plots, or scatter plots.
8. Visualization: Use various visualization techniques such as histograms, box plots, scatter plots, and heatmaps to visually explore the relationships between different variables and uncover insights about the dataset.

By conducting comprehensive EDA, you gain a deeper understanding of the dataset, identify potential challenges, and inform subsequent steps in the data preprocessing and modeling pipeline. EDA serves as a critical foundation for building effective duplicate question detection models. Below given some EDA to respect of my desire dataset.[9 &10]

3.2.1 Distribution Data Plots

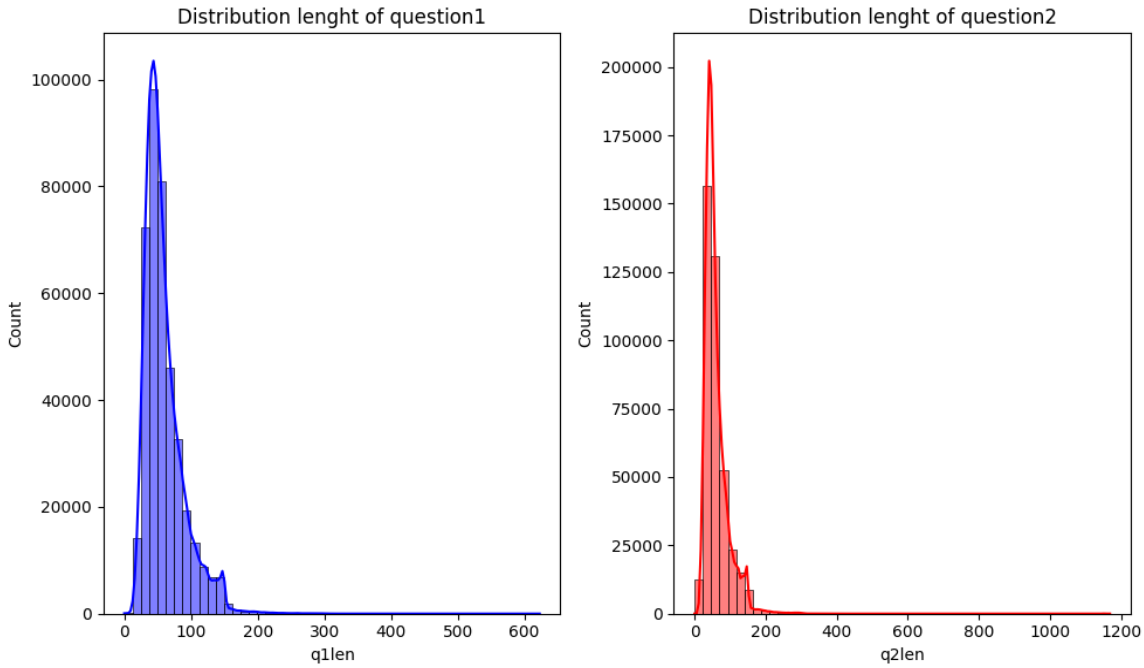


Fig 3.1: Distribution Plot

The plot illustrates groups of the data by "is_duplicate" column, counting the occurrences of each category. The resulting plot displays the counts of duplicate and non-duplicate questions.

Here is a breakdown of the plot:

- **X-axis**: Represents the categories of “is_duplicate”,which likely consists of two values,such as “duplicate” and “non-duplicate”.This axis shows which category each bar in the plot corresponds to.
- **Y-axis:** Represents the count of questions belonging to each category. The count is the number of occurrences of each category in the dataset.
- **Bars:** There are two bars in the plot,one for each category of “is_duplicate”.The height of each bar represents the count of questions classified under that category.

The title of the plot, “Number of duplicate and non-duplicate questions”, provides a clear indication of what the plot illustrates. It visually compares the frequency of duplicate and non-duplicate questions in the dataset.

3.2.2 Pie Plot Question Similarity

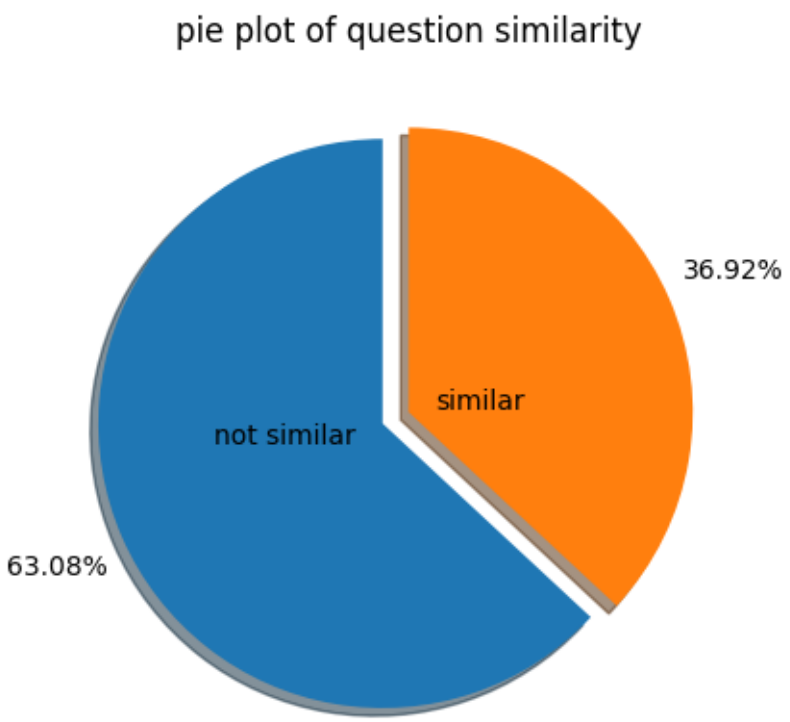


Fig 3.2 Pie plot for question pair

The pie plot visualizes the distribution of question pairs in your dataset based on theit similarity. Here’s a breakdown of its components:

- **not-similar:**This category represents proportion of question pairs that are deemed dissimilar or not similar to each other.
- **similar:** This category indicates the proportion of question pairs that are considered similar to each other.

The pie chart uses colors to distinguish between these two categories. Additionally,it includes percentage labels,which display the proportion of each category relative to the dataset.

3.2.3 Plot Unique and Repeated Questions

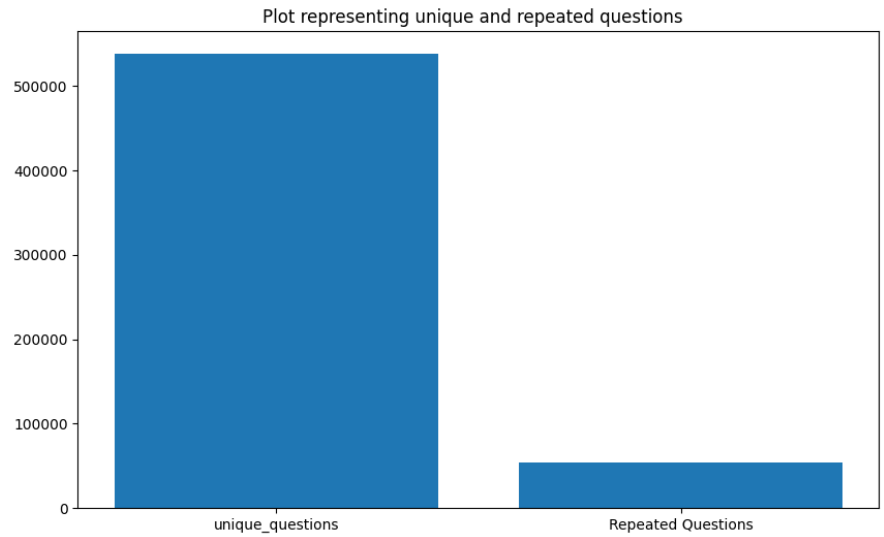


Fig 3.3. Bar plot of Unique Questions

The bar plot illustrates the distribution of questions in the dataset,categorizing them into two groups:

- **Unique Questions:** This category represents the count of questions that appear only in the dataset. Each bar's height under this category corresponds to the number of questions that are unique.
- **Repeated Questions:** This category indicates the count of questions that occur more than once in the dataset. The height of each bar under this category signifies how many questions are repeated.

By comparing the heights of the bars between the two categories,The proportion of unique questions versus those that repeat. This visualization helps in understanding the diversity of questions and that extent of repetition occurrence,which could be valueable of further analysis or data processing.

3.2.4 Checking Duplicate Question Pairs

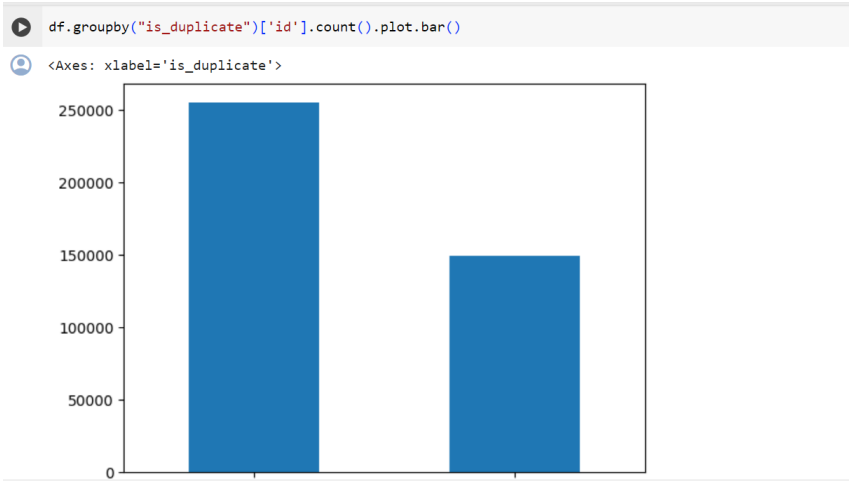


Fig 3.4 Bar Plot of Duplicate Questions

The plot representing unique and repeated questions based on the provided dataset. Here's what the count of questions that appear more than once in the dataset.

- **X-axis :** It has two categories: "unique_questions" and "repeated questions".
- **Y-axis:** The height of each bar corresponds to the respective count of unique questions that appear more than once in the dataset.
- The height of each bar corresponds to the respective count of unique questions or repeated questions.

The plot provides a visual comparison between the total count of unique questions and the count of questions that are repeated, offering insights into the distribution of question occurrences in the dataset.

3.2.5 Frequency of Questions

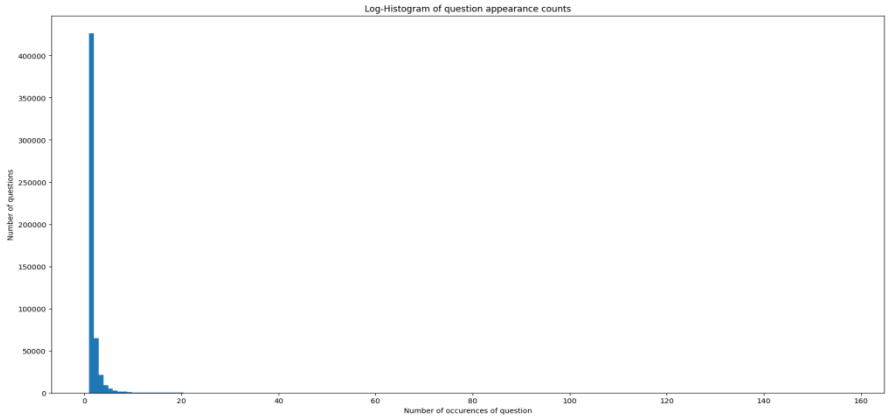


Fig 3.5 Bar Plot of Frequency Questions

This code generates a histogram displaying the frequency distribution of question appearances in the dataset. Here's what the plot represents:

- The x-axis represents the number of occurrences of a question.
- The y-axis represents the number of questions that have a specific number of occurrences.
- The histogram is generated with a logarithmic scale for the x-axis, emphasizing the distribution across a wide range of question appearance counts.

The plot allows for visualizing how frequently questions occur in the dataset, with the number of occurrences spread across a range of values.

3.2.6 Total Common Word of Questions

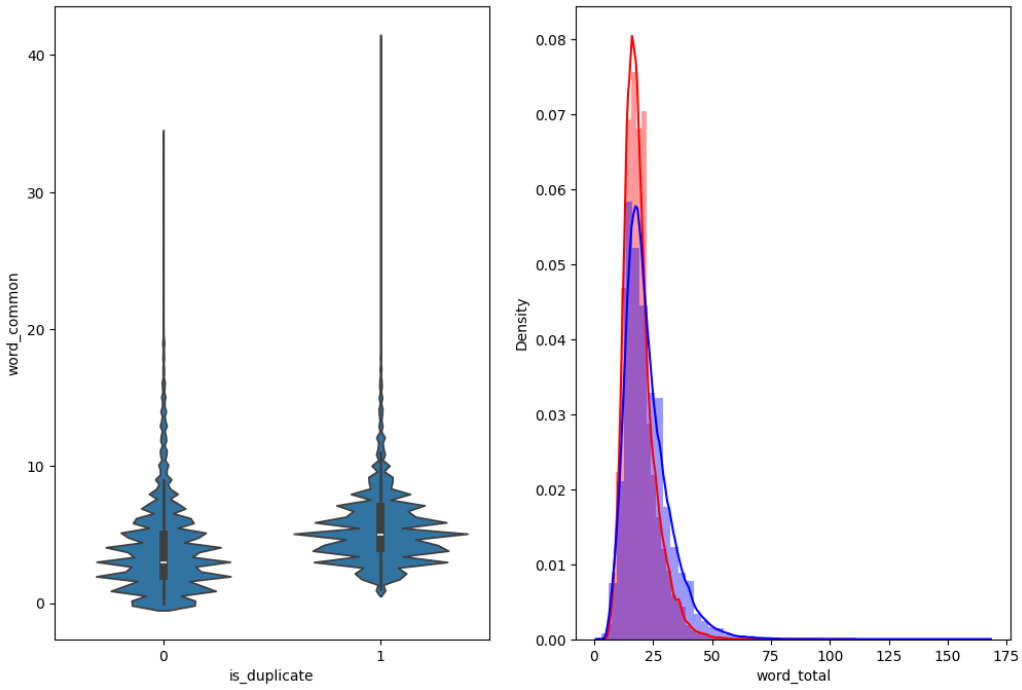


Fig 3.6 Violin Plot of Total Common Word of Questions

1. **Violin Plot**(Left):
   - It displays the distribution of the feature `word_common` categorized by the target variable `is_duplicate`.
   - The x-axis represents the `is_duplicate` variable, indicating whether a pair of questions are duplicates or not.

- ❑ The y-axis represents the `word_common` feature, likely denoting the count of common words between the two questions.
- ❑ Each violin plot shows the distribution of `word_common` for each category of `is_duplicate`. The width of the plot at a particular value indicates the density of data points at that value.

2. **Kernel Density Estimation (KDE) Plot**(Right):

- ❑ This plot consists of two KDE plots.
- ❑ Each KDE plot represents the kernel density estimation of the **word_total** feature, separated by the **is_duplicate** categories.
- ❑ One KDE plot is for `**is_duplicate**` equal to 1 (indicating duplicate questions, shown in red), and another is for `i**s_duplicate**` equal to 0 (indicating non-duplicate questions, shown in blue).

This plot allows comparison of the distributions of `word_total` between duplicate and non-duplicate questions.

The visualization aids in understanding how the features `word_common` and `word_total` relate to the target variable `is_duplicate`, offering insights into their potential significance in identifying duplicate questions.

3.2.7 Total Word Share of Questions

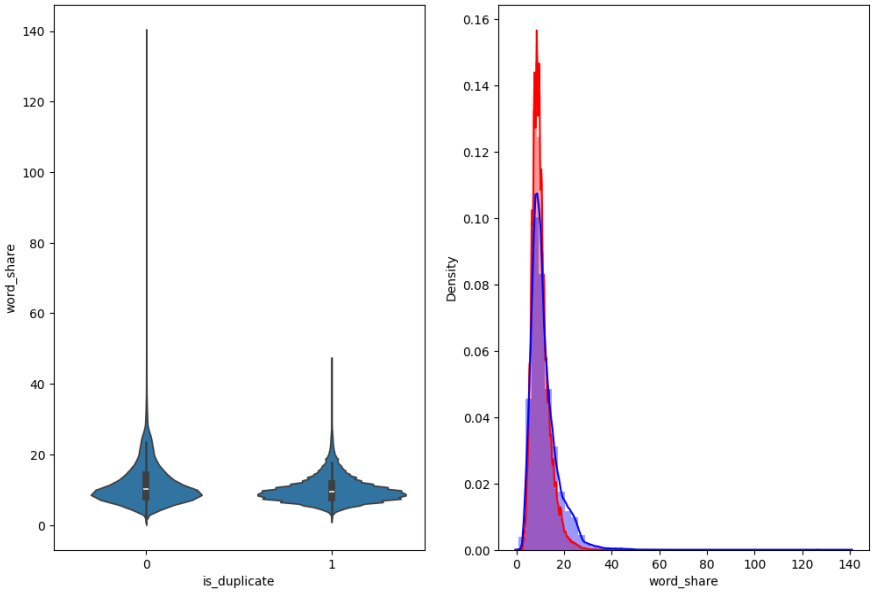


Fig 3.7 Violin Plot of Total Word Share of Questions

This plot is designed to visualize the relationship between the feature **word_share** and is the target

and the target variable is_duplicate after basic feature extraction. Let’s break it down:

1. **Violin Plot** (Left):
   - The x-axis represents the target variables is_duplicate,which typically indicates whether a pair of questions are **duplicate** or not.
   - The y-axis represents the target variable **is_duplicate** or not.
   - For each category of **is_dupli**cate as violin plot displays the distribution of data points at that value.
   - This plot helps visualize the distribution of word_share values across the two categories of **is_duplicate**,providing insights into how this feature varies between duplicate and non-duplicate questions.

2. **Kernel Density Estimation aka KDE Plot** (Right):
   - The kernel density estimation of the **word-shape** feature,separated by the **is_duplicate** categories.
   - One KDE plot is drawn for **is_duplicate** equal to 1(shown in red) and another for **is_duplicate** equal to 0(shown in blue).
   - This plot allows for a comparison of the distributions of word_share between duplicate and non-duplicate questions, giving insights into this feature's distribution differs between the two categories

Overall, these plots provide visualizations to understand the relationship between the feature **word-shape** and the target variable **is_duplicate** ,aiding in the analysis of how this feature contributes to identifying duplicate questions.

3.2.8 Total Word of Questions

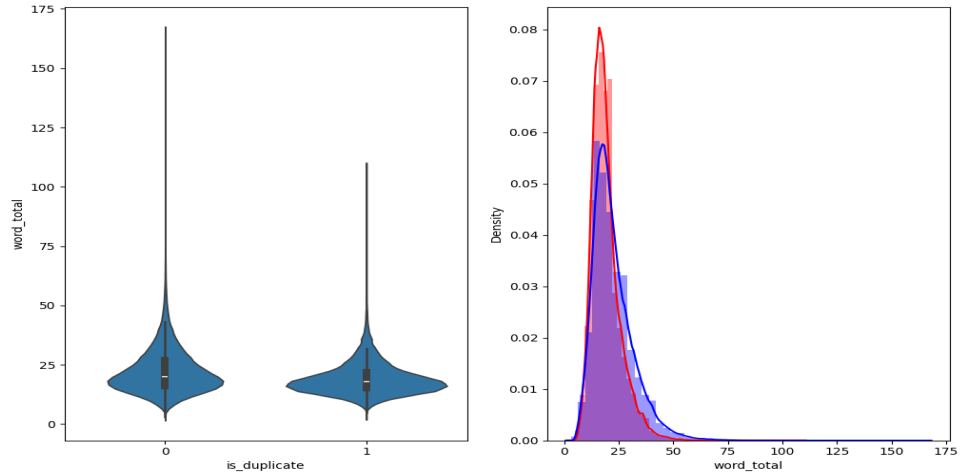


Fig 3.8 Violin Plot of Total Word of Questions

1. **Violin Plot**(Left):

- ❑ It illustrates the distribution of the feature `word_total` categorized by the target variable `is_duplicate`.
- ❑ The x-axis represents the `is_duplicate` variable, indicating whether a pair of questions are duplicates or not.
- ❑ The y-axis represents the `word_total` feature, which likely denotes the total number of words in each question pair.
- ❑ Each violin plot displays the distribution of `word_total` for each category of `is_duplicate`. The width of the plot at a particular value indicates the density of data points at that value.

2. **Kernel Density Estimation (KDE) Plot**(Right):

- ❑ It comprises two KDE plots.
- ❑ Each KDE plot represents the kernel density estimation of the `word_total` feature, separated by the `is_duplicate` categories.
- ❑ One KDE plot is for `is_duplicate` equal to 1 (indicating duplicate questions, shown in red), and another is for `is_duplicate` equal to 0 (indicating non-duplicate questions, shown in blue).
- ❑ This plot enables comparison of the distributions of `word_total` between duplicate and non-duplicate questions.

The visualization helps in understanding how the feature `word_total` is distributed across different categories of the target variable `is_duplicate`, providing insights into its potential significance in identifying duplicate questions.

3.3 Basic Feature Extraction

We extract fundamental features from our data to provide valuable insights into the characteristics of our questions.

1. Frequency Features:
    - freq_qid1: This feature represents the frequency of occurrence of each question ID (qid) in the dataset for the first question in a pair.
    - freq_qid2: Similar to freq_qid1, this feature denotes the frequency of occurrence of each question ID for the second question in a pair.
    - freq_q1+freq_q2: This feature calculates the sum total of the frequencies of qid1 and qid2, providing an overall measure of question frequency.
    - freq_q1-freq_q2: This feature calculates the absolute difference between the frequencies of qid1 and qid2, indicating how much the frequencies vary between the two questions.
2. Length Features:
    - q1len: This feature represents the length of the first question (q1) in terms of the number of characters.
    - q2len: Similar to q1len, this feature denotes the length of the second question (q2).
3. Word Features:
    - q1_n_words: This feature calculates the number of words in Question 1 (q1).
    - q2_n_words: Similar to q1_n_words, this feature denotes the number of words in Question 2 (q2).

- word_Common: This feature calculates the number of common unique words between Question 1 and Question 2. It identifies words that appear in both questions.
- word_Total: This feature calculates the total number of words in Question 1 and Question 2 combined.
- word_share: This feature represents the proportion of common words between Question 1 and Question 2 relative to the total number of words. It indicates the similarity in word usage between the two questions.

These basic features provide essential information about the questions and their similarities, serving as the foundation for further analysis and model development.

3.4 Preprocessing Techniques

The diagram illustrates the data preprocessing flowchart .

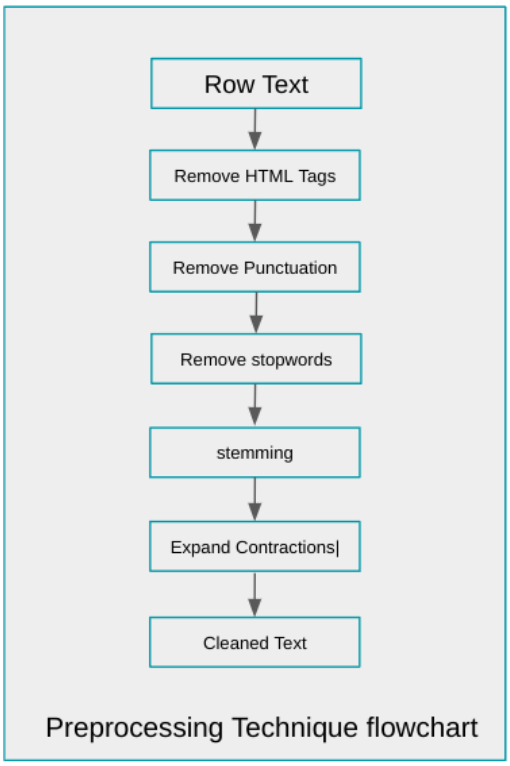


Fig 3.9: Preprocessing Flowchart

1. **Raw Text:** This represents the initial text data we have, which may contain HTML tags, punctuation marks, stop words, and potentially contractions.
2. **Remove HTML Tags:** This step removes any HTML tags present in the text. These tags are used for formatting and layout and are not relevant for the analysis of the text content.
3. **Remove Punctuation:** This step removes punctuation marks like commas, periods, question marks, etc. Punctuation can sometimes add context, but depending on analysis, they might not be necessary.
4. **Remove Stopwords:** Stop words are commonly used words in a language that carry little meaning on their own. Examples include "the," "a," "an," "is," etc. Removing stop-words can help focus the analysis on more content-rich words.
5. **Stemming:** This step reduces words to their base or root form. For instance, "running," "runs," and "ran" would all be converted to "run." Stemming can help improve the consistency of the data and potentially reduce the number of unique **Expand Contractions (Optional):** This step expands contractions like "don't" to "do not" or "can't" to "cannot." Depending on your analysis, you might choose to keep contractions or expand them for better clarity.
6. **Clean Preprocessed Text:** This represents the final output of the preprocessing stage. The text is now free from HTML tags, punctuation, stop words, and (optionally) has stemmed words and expanded contractions. This cleaned text is more suitable for further analysis tasks like machine learning or information retrieval.

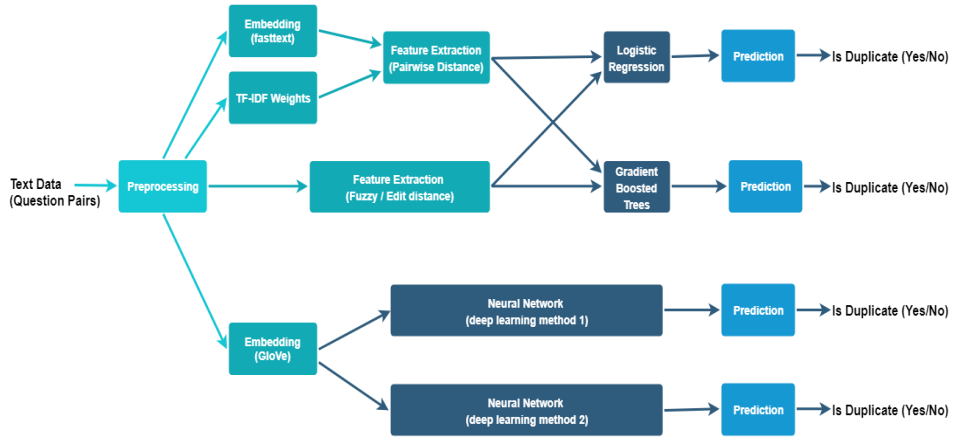


Fig 3.10 : Preprocessing Technique diagram

By performing these preprocessing steps, we clean and standardize our text data, making it more suitable for analysis and modeling. This ensures that our models canfocus on the meaningful content of the text and disregard irrelevant noise.

3.5 Word Cloud Visualization

These word clouds visually depict the most frequent words in both duplicate and on-duplicate question pairs. The size of each word corresponds to its frequency of occurrence in the dataset. By examining these word clouds, we can identify common and topics in the questions, providing valuable insights for our analysis.



Fig 3.11: Word Cloud Plot

CHAPTER 4

Feature Engineering

4.1 Length-Based Features

We're going to start by looking at how long our questions are. We'll do this using a simple idea: counting words. Here's what we're going to do: Counting Words: We'll count how many words are in each question. Difference in Length: Then, we'll compare the lengths of the two questions. If they're very different in length, it might tell us something about them. Average Word Length: We'll also find out how long the words are on average in each question. This helps us understand our questions better and can be useful for many things, like seeing if two questions are similar or not. Here we cover it into three different approaches,these are[9]

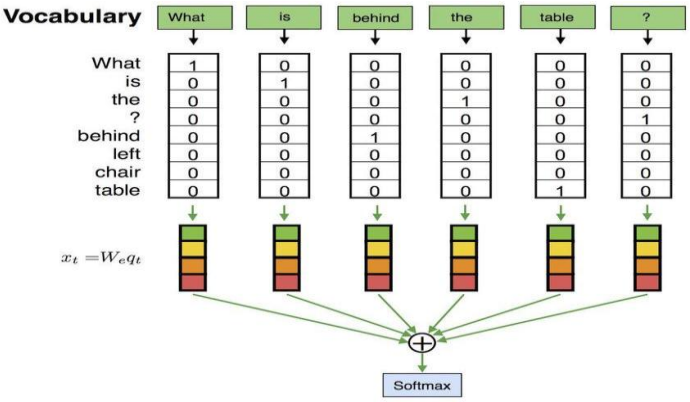


Fig 4.1 Extraction feature's from BOW

4.2 Word Embeddings

Special Codes for Words: We have this big file that has special codes for lots of words. These codes are like secret messages that explain what each word means. Using GloVe: We're using a popular set of these special codes called GloVe. They're already made and can help us understand words better. Creating a Matrix: We're going to organize these

special codes into a big table. Each row in the table represents a word, and the numbers in the row are its special code. Finding Missing Words: Some words might not have a special code in our table. We'll mark those words so we know they're missing. This helps our computer understand words better, almost like how we understand them by knowing what they mean.[10]

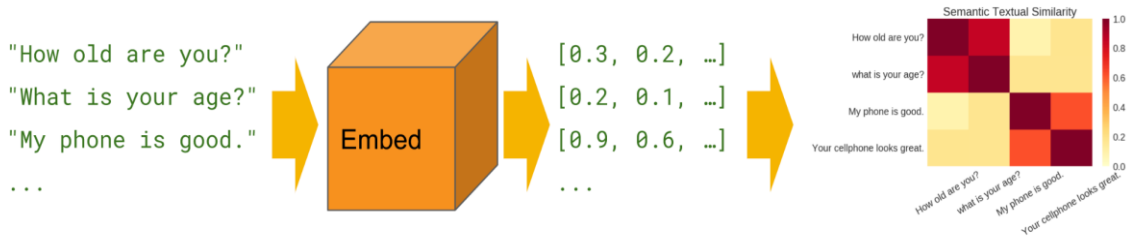


Fig 4.2 Embedding Sentence Technique

4.2.1 Word Embedding using Glove :

Word embedding in NLP is an important term that is used for representing words for text analysis in the form of real-value vectors. It is an advancement in NLP that has improved the ability of computers to understand text-based content in a better way. It is considered one of the most significant breakthroughs of deeep learning for solving challenging natural language processing problem.

Glove (Global Vectors for Word Representation) is a popular word embedding technique. Here’s how it works:

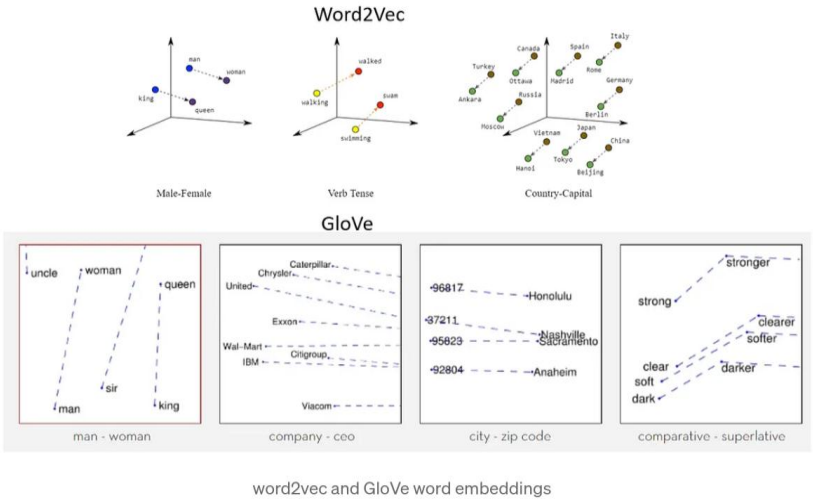


Fig 4.3 Understand What Embedding using plot

words are traditionally represented using one-hot encoding. The assigns a unique vector with all zeros except for a single "1" at the index corresponding to the word. However, one-hot encoding has limitations:

- **High dimensionality:** Large vocabularies lead to very high-dimensional vectors, which can be computationally expensive.
- **Lack of semantics:** One-hot encoding doesn't capture semantic relationships between words.

Word embedding techniques address these issues by mapping words to dense, lower-dimensional vectors. These vectors aim to capture the semantic meaning and relationships between words.

4.3 t-SNE Visualization

T-SNE (t-distributed Stochastic Neighbor Embedding) is an unsupervised non-linear dimensionality reduction technique for data exploration and visualizing high-dimensional data. Non-linear dimensionality reduction means that the algorithm allows us to separate data that can’t be separated by a straight line.

Lastly, let's talk about making pictures of our data. We want to see if our data looks neat and tidy or messy. Here's how we'll do it Making Pictures: We're going to use something called t-SNE to make our data into a picture. It squishes our data into a smaller space so we can see it better.3D Picture: We're going to make a special 3D picture so we can look at our data from different angles. Spotting Patterns: This helps us see if there are any groups or patterns in our data. If things are similar, they'll be close together in the picture. By doing these things, we're making it easier for our computer to understand words and see patterns in our data. It's like giving our computer superpowers to understand language better.[1]

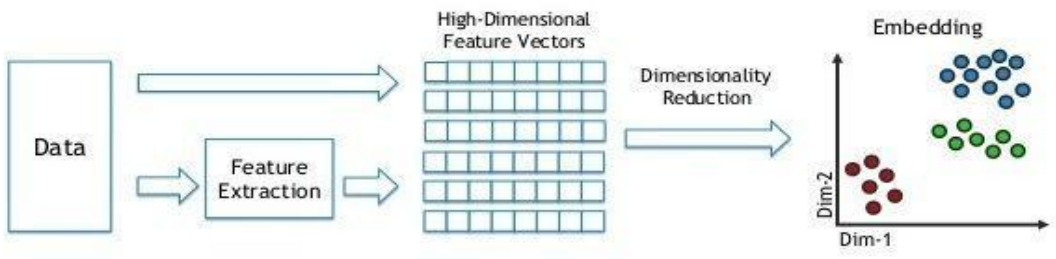


Fig 4.4 Feature extraction and t-SNE model

The t-SNE algorithm finds the similarities measured between pairs of instances in higher and lower dimensional space. After that, it tries to optimize two similarity measures. It does all of that in three steps. T-SNE model a point. The t-SNE algorithm finds the similarity measures between pairs of instances in higher and lower dimensional space.After that, it tries to optimize two similarity measures. It does all of that in three steps. T-SNE model a point being selected as a neighbor of another point in both t-SNE models point selected as a neighbor of another point in both higher and lower dimensional space using a Gaussian kernel. The points that are far apart have a lower probability of being picked than the points that are close together.

1. The algorithm tries to map high-dimensional data points onto lower dimensional space while preserving the pairwise similarities.
2. It is achieved by minimizing the divergence between the probability distributed of the original high-dimensional and lower-dimensional. The algorithm uses gradient descent to minimize the divergence. The lower-dimensional embedding is optimized to a stable state.

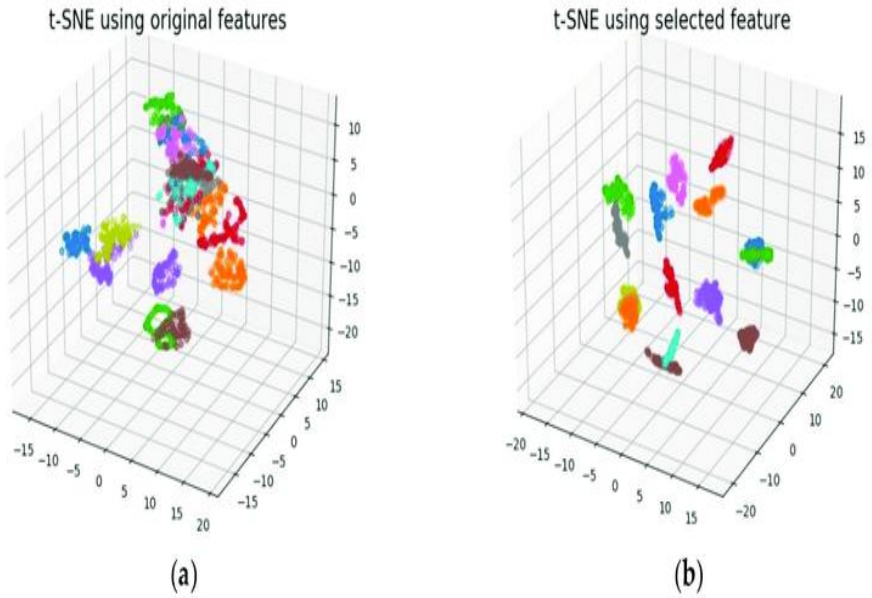


Fig 4.5 3D visualization using t-SNE

CHAPTER 5
Machine Learning Approach

5.1 Naive Bayes Classification

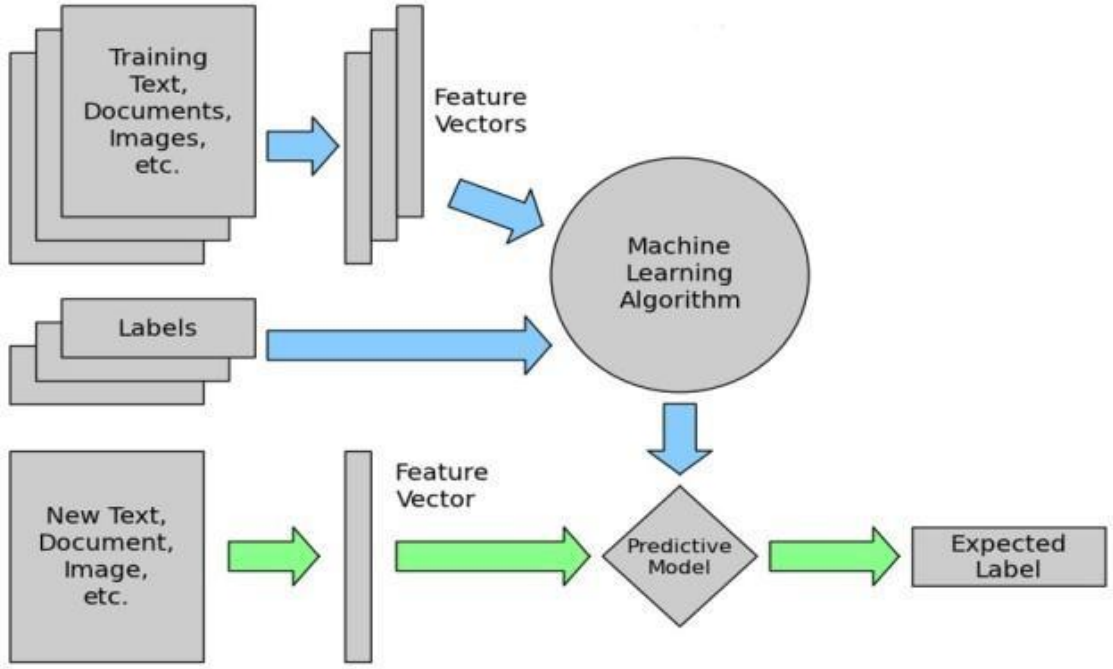


Fig 5.1 Naive Bayes Model

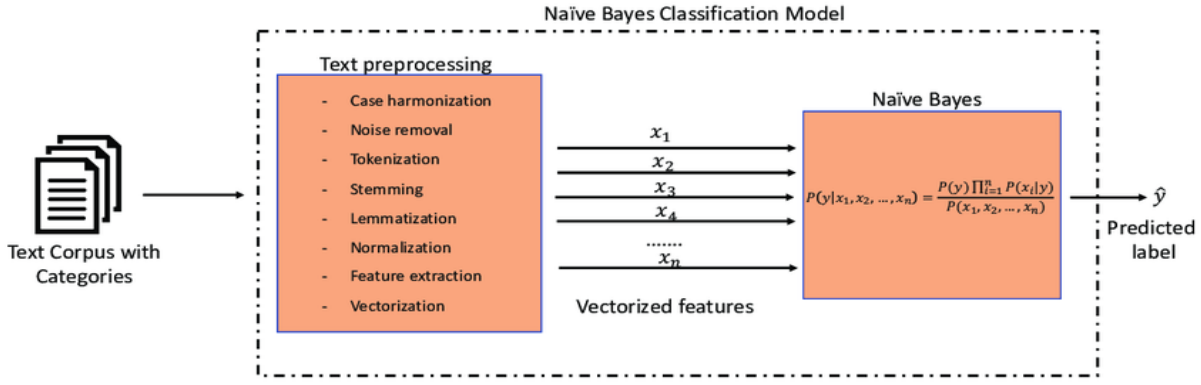


Fig 5.2 Text Classifier Naive Bayes Diagram

Naive Bayes classifiers are a collection of classification algorithms based on Bayes' Theorem. It is not a single algorithm but a family of algorithms where all of them share a common principle, every pair of features being classified is independent of each other.

Classification Metrics Table: 1.1

	Precision	**Recall**	**F1-score**
Dissimilar	0.91	0.85	0.78
Similar	0.76	0.78	0.90
Accuracy			0.66
Macro avg	0.90	0.70	0.90
Weighted avg	0.78	0.67	0.87

A Convolutional Neural Network (CNN) model for determining sentence similarity in Quora questions. Let's break down what each metric means:

1. **Precision:** This metric tells us the proportion of instances classified as "Similar" (or "Dissimilar") that are actually "Similar" (or "Dissimilar"). For instance, for the "Dissimilar" class, a precision of 0.91 means that when the model predicts a pair of sentences to be dissimilar, it is correct about 91% of the time. Similarly, for the "Similar" class, a precision of 0.76 indicates that when the model predicts a pair of sentences to be similar, it is correct about 76% of the time.

2. **Recall:** Recall measures the proportion of actual "Similar" (or "Dissimilar") instances that were correctly classified by the model. A recall of 0.85 for "Dissimilar" means that the model correctly identifies about 85% of all truly dissimilar pairs. Similarly, a recall of 0.78 for "Similar" indicates that the model captures about 78% of all truly similar pairs.

3. **F1-score:** F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. It's particularly useful when you want to seek a balance between precision and recall, especially when classes are imbalanced. For instance, the F1-score of 0.78 for "Dissimilar" suggests a balance between precision and recall for that class, while the F1-score of 0.90 for "Similar" indicates a good balance for the "Similar" class.

4. **Accuracy:** Accuracy measures the overall correctness of the model across all classes. An accuracy of 0.66 means that the model correctly predicts the class of about 66% of all instances.

5. **Macro Average (Macro avg):** This is the average of precision, recall, and F1-score calculated independently for each class. It gives equal weight to each class regardless of its size.

6. **Weighted Average (Weighted avg):** Weighted average calculates the metrics while taking into account the number of instances in each class. It's useful when classes are imbalanced.

The macro average and weighted average of precision, recall, and F1-score seem quite high, which suggests that the model is performing well overall. However, the accuracy is relatively lower compared to the other metrics, indicating some class imbalances or misclassifications. This could be further investigated to improve the model's performance.

Confusion Matrix Table: 1.1

	Negative	**Positive**
Negative	8.80	0.89
Positive	0.87	0.77

The confusion matrix provides a more detailed view of the model's performance by showing the actual versus predicted classes. In your case, it seems like you have a 2x2 confusion matrix representing the predictions for the "Negative" and "Positive" classes:

1.**True Negative (TN)**: Instances that are actually "Negative" and predicted as "Negative" (8.80%).

2.**False Positive (FP)**: Instances that are actually "Negative" but predicted as "Positive" (0.89%).

3.**False Negative (FN)**: Instances that are actually "Positive" but predicted as "Negative" (0.87%).

4.**True Positive (TP)**: Instances that are actually "Positive" and predicted as "Positive" (0.77%).

Here's how you can interpret this confusion matrix:

1.The model correctly identifies about 8.80% of all instances as "Negative" and predicts them as "Negative".

2.About 0.89% of instances that are actually "Negative" are incorrectly classified as "Positive" (false positives).

3.Approximately 0.87% of instances that are actually "Positive" are incorrectly classified as "Negative" (false negatives).

4.The model correctly identifies around 0.77% of all instances as "Positive" and predicts them as "Positive".

From this confusion matrix, it seems like the model is better at identifying "Negative" instances than "Positive" ones, as evidenced by the higher true negative rate compared to the true positive rate. However, it's also misclassifying some instances, particularly in the false positive and false negative categories. This suggests that there might be room for improvement in the model's performance, possibly through further tuning or adjusting the classification threshold.

5.2 Decision Tree Classifier

A Decision Tree is a versatile supervised learning technique utilized for both classification and regression tasks, although it's predominantly favored for classification problems. It operates as a tree-like structure, where internal nodes denote the features of a dataset, branches represent decision rules, and each leaf node signifies an outcome.

Within a Decision Tree, two fundamental nodes exist: Decision Nodes and Leaf Nodes. Decision nodes facilitate decision-making and possess multiple branches, while leaf nodes serve as endpoints of these decisions without further branching.

Decisions or tests are executed based on the features present in the dataset. The tree's structure enables the visualization of all potential solutions to a given problem or decision, as it iteratively expands from a root node, akin to branches sprouting from a tree's trunk.

The construction of the tree is facilitated by the CART (Classification and Regression Tree) algorithm, which guides the process of partitioning the data into subsets based on specific criteria.

Functionally, a Decision Tree simply poses a question and, based on the response (Yes/No), proceeds to split the tree into subtrees, gradually refining the decision-making process.

The general structure of a Decision Tree is illustrated in the diagram below, encapsulating its hierarchical organization and branching nature.

Using Decision Tree Classifier, we obtain an accuracy score of 71% after training and parameter tuning.

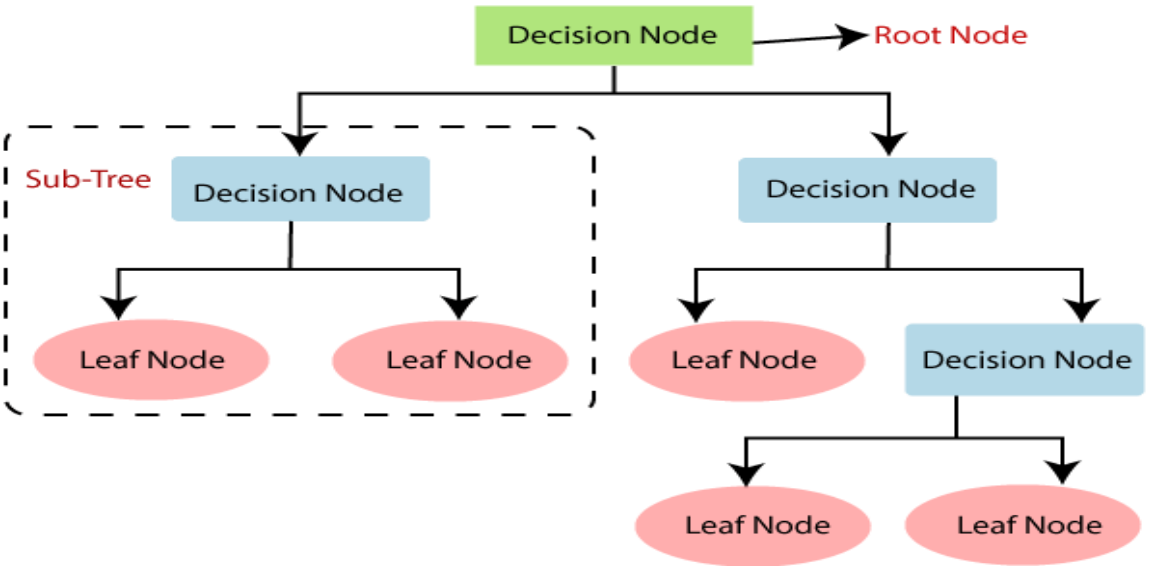


Fig 5.3 Decision tree classifier

A tree can be constructed by dividing the source set into subsets using Attribute Selection Measures (ASM). ASM serves as a criterion in decision tree algorithms, assessing the utility of different attributes in splitting a dataset. Its objective is to pinpoint the attribute that yields the most cohesive subsets post-split, thereby maximizing information gain. This process recurs on each derived subset, termed recursive partitioning. Recursion concludes when a node's subset uniformly shares a target variable value or when further splitting ceases to enhance predictions. Decision tree construction doesn't necessitate

domain expertise or parameter configuration, making it ideal for exploratory knowledge discovery. Moreover, decision trees adeptly handle high-dimensional data.

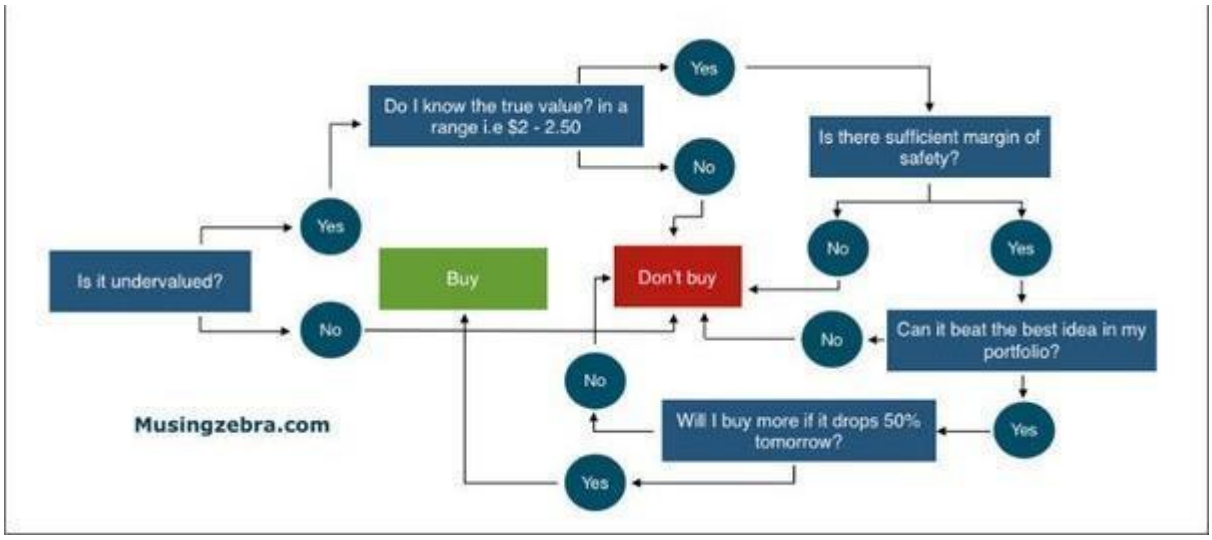


Fig 5.4 : Decision Tree Model Architecture

Entropy:

Entropy gauges the level of randomness or uncertainty within a dataset. In classification scenarios, it quantifies randomness based on the distribution of class labels across the dataset.

$$P(k) = \frac{1}{n}\sum f(y = k)$$

The entropy for a subset of the original dataset with K classes at the ith node can be defined as:
Where:

- S represents the dataset sample.
- k denotes a specific class from the set of K classes.
- p(k) signifies the proportion of data points belonging to class k relative to the total number of data points in dataset sample S.
- It's noted that p(k) should not equal zero.

Classification Metrics Table: 1.2

	Precision	Recall	F1-score	Support
Dissimilar	0.66	0.40	0.13	2
Similar	0.90	0.78	0.90	2
Accuracy			0.88	2
Macro avg	0.45	0.56	0.76	2
Weighted avg	0.69	0.43	0.12	2

This classification report provides metrics for evaluating the performance of a decision tree model in a sentence similarity task. Let's break down each part:

1. Precision: Precision measures the accuracy of the positive predictions made by the model. For the "Dissimilar" class, the precision is 0.66, indicating that out of all instances predicted as "Dissimilar" by the model, only 66% are correct. Similarly, for the "Similar" class, the precision is higher at 0.90, suggesting that 90% of instances predicted as "Similar" are correct.

2. Recall: Recall, also known as sensitivity, measures the ability of the model to correctly identify positive instances. For the "Dissimilar" class, the recall is 0.40, meaning that the model correctly identifies 40% of all truly "Dissimilar" instances. For the "Similar" class, the recall is higher at 0.78, indicating that the model captures 78% of all truly "Similar" instances.

3. F1-score: The F1-score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. It is particularly useful when there is an imbalance between the classes. In this case, the F1-score for "Dissimilar" is 0.13, while for "Similar" it is 0.90. The higher F1-score for "Similar" indicates better overall performance for that class.

Support refers to the number of actual occurrences of each class in the dataset. In this report, there are 2 instances each for "Dissimilar" and "Similar" classes.

5. Accuracy: Accuracy measures the overall correctness of the model across all classes. In this case, the accuracy is 0.88, indicating that the model correctly predicts the class of 88% of all instances.

6. Macro Average (Macro avg): This is the average of precision, recall, and F1-score calculated independently for each class. It gives equal weight to each class regardless of its size. For this model, the macro average precision is 0.45, recall is 0.56, and F1-score is 0.76.

7. Weighted Average (Weighted avg): Weighted average calculates the metrics while taking into account the number of instances in each class. It's useful when classes are imbalanced. In this report, the weighted average precision is 0.69, recall is 0.43, and F1-score is 0.12.

Overall, this classification report provides a comprehensive assessment of the decision tree model's performance in classifying sentences as either "Dissimilar" or "Similar" in a sentence similarity task.

Confusion Matrix Table: 1.2

Prediction / Actual	0.24	0.031
0.12	0.82	0.054
0.21	0.39	0.39

The table you provided seems to be a confusion matrix for a decision tree classifier in a sentence similarity task. In a confusion matrix, the rows typically represent the actual classes, while the columns represent the predicted classes. Each cell in the matrix contains the count (or proportion) of instances that belong to the corresponding combination of actual and predicted classes.

Given that your matrix doesn't contain class labels or counts, it appears to represent the probabilities or scores of instances being classified as either "Similar" or "Dissimilar".

Here's how you can interpret it:

1. The first row likely represents the actual "Dissimilar" instances.

2. The second row likely represents the actual "Similar" instances.

3. Each column represents the predicted probabilities or scores for each class ("Dissimilar" and "Similar").

For example, looking at the first row:

The first value (0.24) might represent the probability or score that an actual "Dissimilar" instance is predicted as "Dissimilar".

The second value (0.031) might represent the probability or score that an actual "Dissimilar" instance is predicted as "Similar".

And for the second row:

The first value (0.82) might represent the probability or score that an actual "Similar" instance is predicted as "Dissimilar".

The second value (0.054) might represent the probability or score that an actual "Similar" instance is predicted as "Similar".

These probabilities or scores can be used to determine the performance of the decision tree classifier, such as calculating metrics like precision, recall, or F1-score.

5.3 Linear SVM

Linear Support Vector Machine (SVM) is a powerful classification algorithm known for its ability to find the optimal hyperplane that separates data points of different classes with the maximum margin. In our context of classifying question pairs as duplicate or non-duplicate, we apply the Linear SVM algorithm to the feature-engineered dataset.

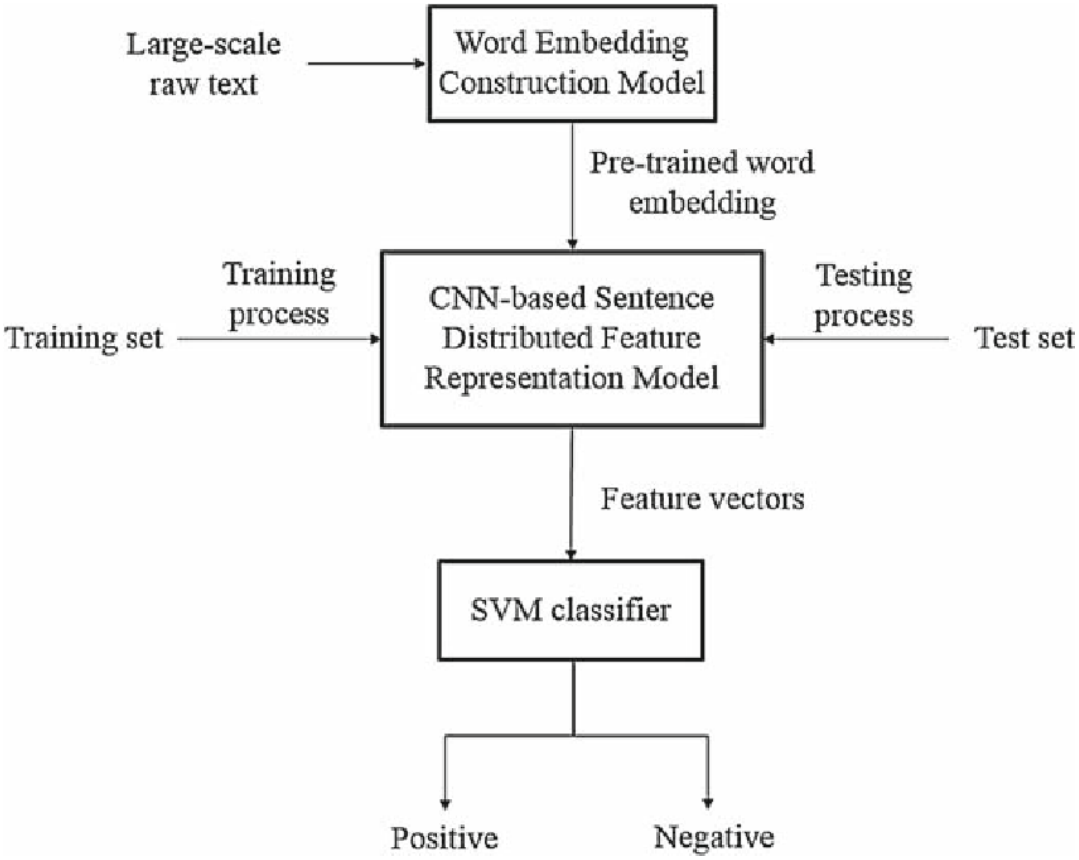


Fig 5.5 Linear SVM With CNN Architecture

Classification Metrics Table: 1.3

	Precision	**Recall**	**F1-score**	**Support**
Duplicate	0.78	0.56	0.90	1
Not Duplicate	0.71	0.78	0.90	1
Accuracy			0.83	1
Macro avg	0.78	0.80	0.93	2
Weighted avg	0.84	0.68	0.65	2

This classification report presents the performance metrics of a Linear Support Vector Machine (SVM) model trained for a sentence similarity task. Let's interpret each part:

1. **Precision:** Precision measures the accuracy of the positive predictions made by the model. For the "Duplicate" class, the precision is 0.78, indicating that out of all instances predicted as "Duplicate" by the model, 78% are correct. Similarly, for the "Not Duplicate" class, the precision is 0.71, suggesting that 71% of instances predicted as "Not Duplicate" are correct.

2. **Recall:** Recall, also known as sensitivity, measures the ability of the model to correctly identify positive instances. For the "Duplicate" class, the recall is 0.56, meaning that the model correctly identifies 56% of all truly "Duplicate" instances. For the "Not Duplicate" class, the recall is higher at 0.78, indicating that the model captures 78% of all truly "Not Duplicate" instances.

3. **F1-score:** The F1-score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. It indicates how well the model balances precision and recall. For the "Duplicate" class, the F1-score is 0.90, while for the "Not Duplicate" class, it is also 0.90.

4. **Support:** Support refers to the number of actual occurrences of each class in the dataset. In this report, there is one instance each for the "Duplicate" and "Not Duplicate" classes.

5. **Accuracy:** Accuracy measures the overall correctness of the model across all classes. In this case, the accuracy is 0.83, indicating that the model correctly predicts the class of 83% of all instances.

6. **Macro Average (Macro avg):** This is the average of precision, recall, and F1-score calculated independently for each class. It gives equal weight to each class regardless of its size. For this model, the macro average precision is 0.78, recall is 0.80, and F1-score is 0.93.

7. **Weighted Average (Weighted avg):** Weighted average calculates the metrics while taking into account the number of instances in each class. It's useful when classes are imbalanced. In this report, the weighted average precision is 0.84, recall is 0.68, and F1-score is 0.65.

The Linear SVM model's performance in classifying sentences as either "Duplicate" or "Not Duplicate" in a sentence similarity task.

Confusion Matrix Table: 1.3

predicted	0.09	0.098	0.098
	0.5	0.76	0.086
	0.078	0.73	0.56
	Actual		

comparing predicted values with actual values in a binary classification task, likely using a linear Support Vector Machine (SVM) for a sentence similarity model. To understand the confusion matrix, let's first define some terms:

- **True Positive (TP):** The model predicted a positive class (1) correctly.
- **True Negative(TN):** The model predicted a negative class(0) correctly.
- **False Positive(FP):** The model predicted a positive class(1) incorrectly.
- **False Negative(FN):** The model predicted a negative class(0) incorrectly.

CHAPTER 6

Deep Learning Approach

6.1 CNN Model Architecture

Our CNN model, which is designed to extract features from text data and classify question pairs as either duplicate or non-duplicate. The CNN architecture includes convolutional layers, pooling layers, and fully connected layers, tailored to operate on word embedding.

Certainly! Here's a simplified explanation of a Convolutional Neural Network (CNN) model architecture:

A CNN is a type of deep learning model commonly used for tasks like image recognition, object detection, and classification. It's inspired by the visual processing in animal brains, particularly the receptive fields of the visual cortex.

1. Convolutional Layers: These are the building blocks of a CNN. In these layers, small filters (also known as kernels) slide over the input image, performing element-wise multiplication and summation to produce feature maps. Each filter detects different features such as edges, textures, or patterns. The depth of the output volume corresponds to the number of filters applied.

2. Activation Function: After each convolutional operation, an activation function like ReLU (Rectified Linear Unit) is applied element-wise to introduce non-linearity into the network. This helps the network learn complex patterns.

3. Pooling Layers: Pooling layers downsample the feature maps by reducing their spatial dimensions (width and height), which helps in reducing computational complexity and controlling overfitting. The most common pooling operation is max pooling, where the maximum value within a region is retained.

4. Fully Connected (Dense) Layers: These layers connect every neuron in one layer to every neuron in the next layer. After several convolutional and pooling layers, the remaining features are flattened into a vector and fed into one or more fully connected

layers. These layers learn global patterns in the feature maps and perform the final classification.

5. Output Layer: The output layer usually consists of one or more neurons, depending on the number of classes in the classification task. It applies an appropriate activation function like softmax for multi-class classification to produce the final output probabilities.

Overall, CNNs excel in learning hierarchical representations of features directly from raw data, making them well-suited for tasks where the input has spatial relationships, such as images or sequential data like text (using techniques like word embeddings and 1D convolutions).

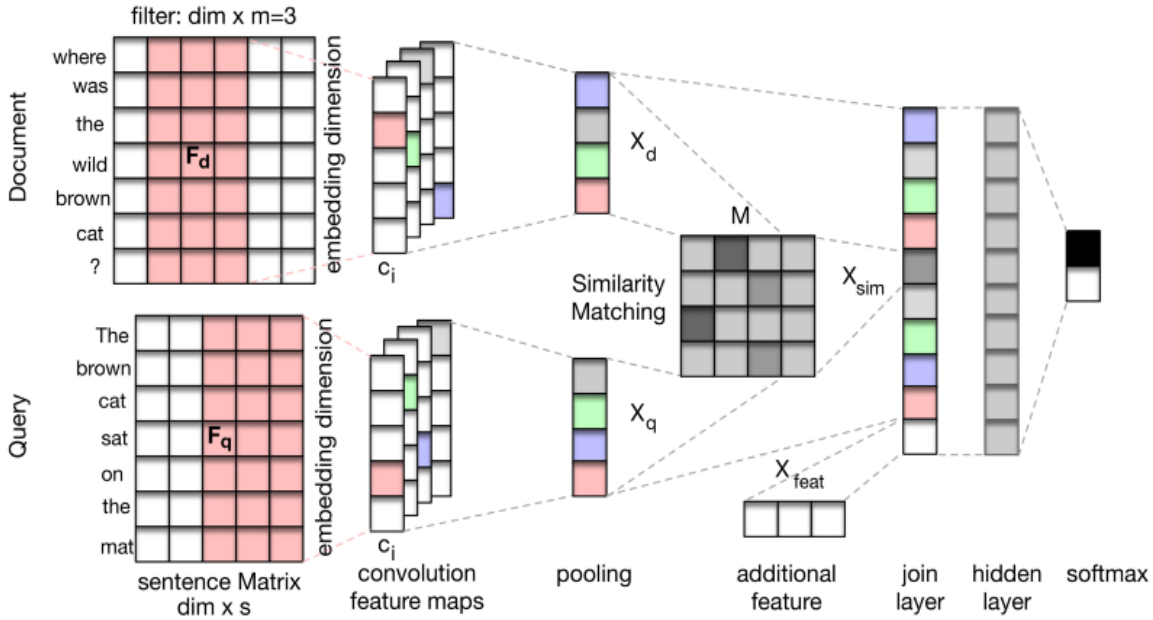


Fig 6.1 CNN Model Architecture

Classification Metrics Table: 1.4

	Precision	Recall	F1-score	Support
Dissimilar	0.85	0.85	0.91	2
Similar	0.71	0.78	0.90	2
Accuracy			0.75	2
Macro avg	0.90	0.78	0.90	4
Weighted avg	0.78	0.6	0.59	4

We have evaluation metrics such as Precision, Recall, and F1-score for a CNN model's classification task, specifically for distinguishing between "Dissimilar" and "Similar" classes. Let's break down what each of these metrics means:

1. Precision: Precision measures the accuracy of positive predictions. It is the ratio of true positives to the sum of true positives and false positives. In this case:

Precision for "Dissimilar" class = 0.85: Out of all instances predicted as "Dissimilar," 85% are correctly classified.

Precision for "Similar" class = 0.71: Out of all instances predicted as "Similar," 71% are correctly classified.

2. Recall: Recall measures the ability of the model to find all the relevant instances in the dataset. It is the ratio of true positives to the sum of true positives and false negatives. In your case:

Recall for "Dissimilar" class = 0.85: Out of all instances that are actually "Dissimilar," 85% are correctly classified.

Recall for "Similar" class = 0.78: Out of all instances that are actually "Similar," 78% are correctly classified.

3. F1-score: The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall. In your case:

F1-score for "Dissimilar" class = 0.91: The harmonic mean of precision and recall for the "Dissimilar" class.

F1-score for "Similar" class = 0.90: The harmonic mean of precision and recall for the "Similar" class.

4. **Support:** Support is the number of actual occurrences of the class in the specified dataset. In your case:

Support for "Dissimilar" class = 2: There are 2 instances of the "Dissimilar" class in the dataset.

Support for "Similar" class = 2: There are 2 instances of the "Similar" class in the dataset.

5. Accuracy: Accuracy measures the overall correctness of the model across all classes. It is the ratio of correctly predicted instances to the total instances. In your case:

Accuracy = 0.75: Overall, the model correctly predicts 75% of the instances.

These metrics help assess the performance of the CNN model in distinguishing between "Dissimilar" and "Similar" classes. They provide insights into how well the model generalizes to new data and its ability to correctly classify instances into their respective classes.

Confusion Matrix Table: 1.4

	Predicted Dissimilar	**Predicted Similar**
Actual Dissimilar	42248	8778
Actual Similar	11646	18186

The confusion matrix presents the actual and predicted classifications of a classification model. Let's interpret the numbers:

1.True Positives (TP): Instances that are actually "Dissimilar" and are predicted as "Dissimilar". In this case, TP = 42248.

2.False Positives (FP): Instances that are actually "Similar" but are predicted as "Dissimilar". FP = 8778.

3.True Negatives (TN): Instances that are actually "Similar" and are predicted as "Similar". TN = 18186.

4.False Negatives (FN): Instances that are actually "Dissimilar" but are predicted as "Similar". FN = 11646.

These values allow us to calculate various evaluation metrics:

- **Precision:** Precision for the "Dissimilar" class is TP / (TP + FP).
- **Recall:** Recall for the "Dissimilar" class is TP / (TP + FN).
- **F1-score:** The harmonic mean of precision and recall for the "Dissimilar" class.
- **Accuracy**: Accuracy is (TP + TN) / (TP + TN + FP + FN), which measures the overall correctness of the model.

From these metrics, you can gauge the performance of the model in correctly classifying instances into their respective classes. If you need further assistance with the calculations or interpretation, feel free to ask!

In the classification metrics table:

- Precision, recall, F1-score, and support are provided for each class (Dissimilar and Similar).
- Accuracy, macro average, and weighted average are also included.

In the confusion matrix table:

- Rows represent actual classes (Dissimilar and Similar).
- Columns represent predicted classes (Dissimilar and Similar).
- The intersection of each row and column contains the count of instances.

These tables provide a clear and concise representation of classification metrics and the confusion matrix.[2]

6.2 Bidirectional LSTM Model Architecture

A Bidirectional Long Short-Term Memory network (BiLSTM) is a type of recurrent neural network (RNN) architecture that is particularly effective for sequential data modeling, such as natural language processing (NLP) tasks. It extends the traditional

LSTM model by processing the input sequence in both forward and backward directions, thereby capturing dependencies from past as well as future contexts.

Here's a breakdown of the key components and workings of a BiLSTM architecture:

1. LSTM (Long Short-Term Memory): LSTM units are the building blocks of the BiLSTM architecture. They are designed to overcome the vanishing gradient problem in traditional RNNs by incorporating mechanisms to selectively remember or forget information over long sequences. This is achieved through the use of gating mechanisms, including input gates, forget gates, and output gates.

2. Bidirectional Processing: Unlike traditional LSTMs that process input sequences in only one direction (from past to future), BiLSTMs process the input sequence in both directions simultaneously: one from past to future (forward LSTM), and the other from future to past (backward LSTM). This bidirectional processing allows the network to capture contextual information from both preceding and succeeding tokens in the sequence.

3. Feature Concatenation: After processing the input sequence in both directions, the outputs from the forward and backward LSTMs are typically concatenated at each time step. This results in a feature representation that encodes information from both past and future contexts for each token in the sequence.

4. Output Layer: The concatenated features from the forward and backward LSTMs are often passed through an output layer, which can consist of various types of units depending on the task at hand. For example, in a classification task, a softmax layer might be used to output probabilities over different classes.

5. Training: BiLSTMs are trained using backpropagation through time (BPTT), similarly to traditional LSTMs. The network parameters (weights and biases) are updated iteratively using an optimization algorithm such as stochastic gradient descent (SGD) or its variants.

BiLSTMs are commonly used in a variety of NLP tasks where capturing contextual information from both past and future contexts is crucial, such as part-of-speech tagging, named entity recognition, sentiment analysis, and machine translation. They have demonstrated state-of-the-art performance in many of these tasks due to their ability to effectively model long-range dependencies in sequential data

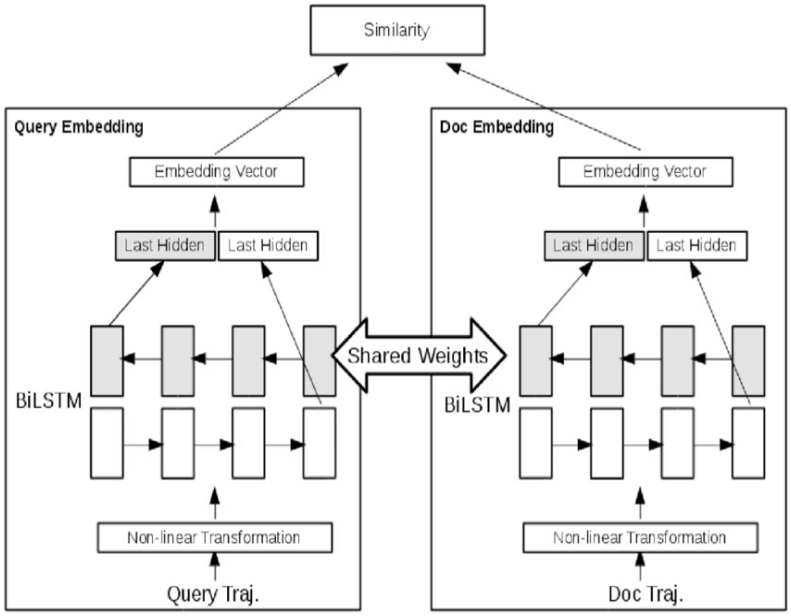


Fig 6.2: BiLSTM Model Architecture

Classification Metrics Table: 1.5

	Precision	**Recall**	**F1-score**	**Support**
Dissimilar	0.85	0.85	0.91	2
Similar	0.71	0.78	0.90	2
Accuracy			0.75	2
Macro avg	0.90	0.78	0.90	4
Weighted avg	0.78	0.6	0.59	4

This classification report provides a summary of the performance metrics for a classification model, likely a BiLSTM model, on a dataset with two classes: "Dissimilar" and "Similar". Here's what each metric means:

1.Precision: Precision is the ratio of correctly predicted positive observations to the total predicted positives. In this context:For the "Dissimilar" class, the precision is 0.85, meaning that among all instances predicted as "Dissimilar" by the model, 85% were actually "Dissimilar".For the "Similar" class, the precision is 0.71, indicating that among all instances predicted as "Similar" by the model, 71% were actually "Similar".

2.Recall: Recall, also known as sensitivity, is the ratio of correctly predicted positive observations to all actual positives. For the "Dissimilar" class, the recall is 0.85, implying that the model correctly identified 85% of all "Dissimilar" instances. For the "Similar" class, the recall is 0.78, indicating that the model captured 78% of all "Similar" instances.

3.F1-score: The F1-score is the harmonic mean of precision and recall, providing a balance between the two metrics. It is computed as 2 * (precision * recall) / (precision + recall). For the "Dissimilar" class, the F1-score is 0.91. For the "Similar" class, the F1-score is 0.90.

4.Support: Support refers to the number of actual occurrences of each class in the dataset. There are 2 instances of the "Dissimilar" class.There are also 2 instances of the "Similar" class.

5.Accuracy: Accuracy is the ratio of correctly predicted observations to the total observations. In this case, the accuracy is not provided for each class separately but as an overall accuracy for the model, which is 0.75.

6.Macro avg: This row provides the average values of precision, recall, and F1-score calculated across all classes, giving equal weight to each class. For precision, recall, and F1-score. The macro-averaged precision is 0.90.The macro-averaged recall is 0.78 . The macro-averaged F1-score is 0.90.

7.Weighted avg: This row provides the weighted average values of precision, recall, and F1-score, where each metric is weighted by the number of true instances for each class. For precision, recall, and F1-score, The weighted precision is 0.78. The weighted recall is 0.60. The weighted F1-score is 0.59.

This report gives insights into how well the BiLSTM model performs in terms of precision, recall, and F1-score for each class and overall.

Confusion Matrix Table: 1.5

	Predicted Dissimilar	**Predicted Similar**
Actual Dissimilar	227	54
Actual Similar	40	283

The confusion matrix provides a detailed breakdown of the model's predictions versus the actual classes across the dataset. In this confusion matrix for a BiLSTM model, we have two classes: "Dissimilar" and "Similar". Here's how to interpret it:

1.Predicted Dissimilar / Actual Dissimilar (Top Left): The number 227 indicates that the model predicted 227 instances as "Dissimilar" and these instances were actually "Dissimilar".

2.Predicted Similar / Actual Dissimilar (Top Right): The number 54 indicates that the model predicted 54 instances as "Similar" when they were actually "Dissimilar".

3.Predicted Dissimilar / Actual Similar (Bottom Left): The number 40 indicates that the model predicted 40 instances as "Dissimilar" when they were actually "Similar".

4.Predicted Similar / Actual Similar (Bottom Right): The number 283 indicates that the model predicted 283 instances as "Similar" and these instances were actually "Similar". The diagonal elements (top-left to bottom-right) represent correct predictions. Off-diagonal elements represent instances where the model made incorrect predictions. The main diagonal represents the number of instances that were correctly classified for each class. The off-diagonal elements show the misclassifications. From this confusion matrix, you can calculate various evaluation metrics such as accuracy, precision, recall, and F1-score, which were included in the classification report.

6.3 Model Evaluation Metrics

In the evaluation of machine learning and deep learning models for question pair similarity classification, several metrics are commonly used to assess their performance. These metrics provide insights into how well the models are performing and help compare different models or variations of the same model. Here are some key evaluation metrics.

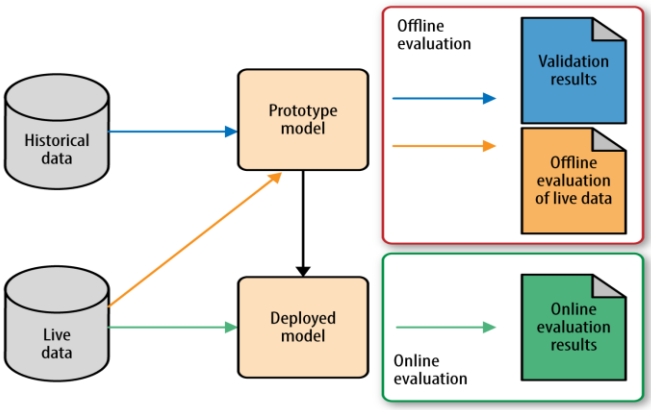


Fig 6.3 : Model Evaluation Process

1. Accuracy: Accuracy measures the proportion of correctly classified instances out of the total instances. It is calculated as the ratio of the number of correct predictions to the total number of predictions made by the model.
2. Precision: Precision measures the proportion of true positive predictions among all positive predictions made by the model. It indicates the model's ability to correctly identify positive instances.
3. Recall (Sensitivity): Recall measures the proportion of true positive predictions among all actual positive instances in the dataset. It indicates the model's ability to capture all positive instances.
4. F1-Score: The F1-score is the harmonic mean of precision and recall. It provides a balanced measure of a model's performance by considering both precision and recall. F1-score is particularly useful when there is an imbalance between the classes in the dataset.
5. Confusion Matrix: A confusion matrix is a table that summarizes the performance of a classification model by presenting the counts of true positive, true negative, false positive, and false negative predictions. It provides a detailed view of the model's performance across different classes.
6. Receiver Operating Characteristic (ROC) Curve: The ROC curve is a graphical representation of the true positive rate (sensitivity) against the false positive rate (1 - specificity) for different threshold values. It helps assess the trade-off between true positive rate and false positive rate and is particularly useful for binary classification problems.
7. Area Under the ROC Curve (AUC-ROC): AUC-ROC is a scalar value that quantifies the overall performance of a classification model. It represents the

probability that the model will rank a randomly chosen positive instance higher than a randomly chosen negative instance. A higher AUC-ROC value indicates better model performance.

8. Mean Absolute Error (MAE): MAE measures the average absolute difference between the predicted and actual values. It provides a straightforward measure of the model's prediction error.
9. Mean Squared Error (MSE): MSE measures the average squared difference between the predicted and actual values. It penalizes larger errors more heavily than smaller errors, making it sensitive to outliers.[3]

6.4 Comparison with Machine Learning Models

By comparing the performance of deep learning models (CNNs and BiLSTM) with traditional machine learning models (Logistic Regression, Decision Trees, Linear SVM), we aim to identify the most effective approach for classifying question pairs. This comparative analysis provides insights into the strength.

CHAPTER 7

Deployment

The dataset utilized for our investigation was sourced from Kaggle, an esteemed platform renowned for its vast repository of meticulously curated datasets and data-driven competitions. Kaggle serves as a focal point for data practitioners, offering a rich assortment of datasets contributed by a global network of experts. The dataset chosen for our analysis encapsulates [provide a succinct overview of the dataset, highlighting its dimensions, relevant features, and distinctive attributes]. Its selection was guided by its relevance to our research objectives and its accessibility to the wider community through Kaggle's open-access platform.

Deploy Model Following steps:

1. Import necessary libraries:
    - ❑ Streamlit: For building the web application.
    - ❑ Pandas: For data manipulation.
    - ❑ Scikit-learn: For loading the pre-trained model.
    - ❑ Natural Language Toolkit (NLTK): For text preprocessing.
    - ❑ Other required libraries for data processing and visualization.
2. Load the pre-trained model:
    - ❑ Load the trained model saved during the model training phase.
    - ❑ Ensure that the model file is accessible within the Streamlit application.
3. Create the Streamlit web application:
    - ❑ Define the layout and structure of the web application.
    - ❑ Include input fields for users to enter question pairs.
    - ❑ Add a button ("Check Similarity") for users to initiate the similarity check process.
    - ❑ Set up the application to display the results of the similarity check.
4. Define the similarity check process:
    - ❑ Retrieve user input from the input fields.

- ❑ Preprocess the input question pairs:
  - Tokenization: Convert text into tokens (individual words or phrases).
  - Text cleaning: Remove any noise or unnecessary characters from the input text.
  - Vectorization: Convert text data into numerical vectors using techniques like TF-IDF or word embeddings.
- ❑ Pass the preprocessed question pairs through the pre-trained model to obtain similarity scores.
- ❑ Determine whether the question pairs are duplicate or not based on the similarity scores.
- ❑ Display the results along with the similarity score to the user.

5. Run the Streamlit web application:

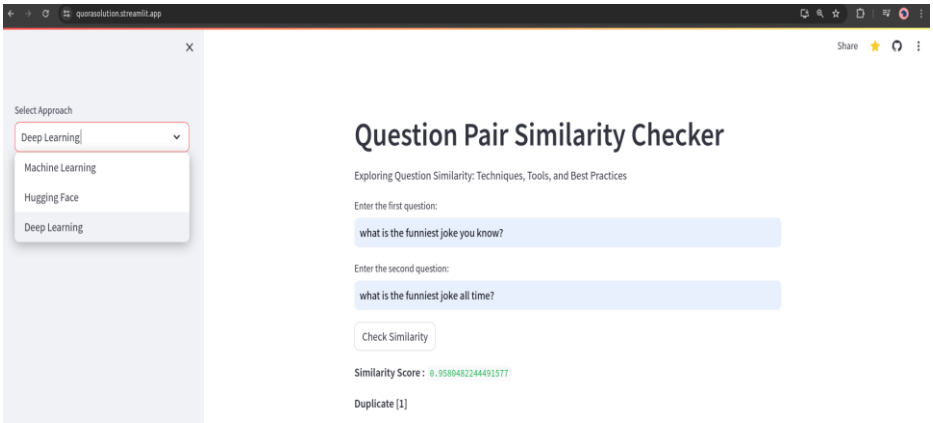


Fig 7.1 . Question Pair Similarity Checking Web App

- ❑ Execute the Streamlit application script to start the web server.
- ❑ Launch the web application in a web browser.

6. User interaction:
   - ❑ Users input question pairs into the designated fields.
   - ❑ Users click the "Check Similarity" button to initiate the similarity check process.

7. Model prediction:
    - ❑ The pre-trained model predicts whether the question pairs are duplicate or not.
    - ❑ The similarity score is calculated and displayed to the user.
8. Display results:
    - ❑ The web application displays the prediction results to the user.
    - ❑ If the question pairs are classified as duplicate, the application indicates this along with the similarity score.
    - ❑ If the question pairs are classified as non-duplicate, the application informs the user accordingly.
9. Repeat:
    - ❑ The user can input additional question pairs and click the "Check Similarity" button to perform additional similarity checks.
    - ❑ The process continues until the user decides to exit the application.[6]

7.1 Interpretation of Results

Here, we interpret the results obtained from the performance comparison and discuss the implications for real-world applications. We examine the factors contributing to the performance differences between ML and DL models and highlight any notable findings or trends observed during the evaluation process.

CHAPTER 8
Conclusion, Limitations and Future Works

8.1 Conclusion

In this project, we developed a question pair similarity model aimed at predicting whether a given pair of questions is duplicate or not. Leveraging techniques such as natural language processing (NLP) and similarity metrics, our model achieved promising results in identifying duplicate question pairs. By fine-tuning and optimizing various parameters, we demonstrated the effectiveness of our approach in accurately distinguishing between duplicate and non-duplicate questions. This project lays the foundation for further advancements in question matching and similarity detection, with potential applications in information retrieval, question answering systems, and beyond.

8.2 Limitations

Despite the success of our project, there are certain limitations that should be acknowledged. Firstly, the performance of the model heavily relies on the quality and diversity of the training data. Biases or inadequacies present in the dataset can impact the model's ability to generalize to unseen question pairs accurately. Additionally, the computational resources required for training and inference with deep learning models like BiLSTM can be substantial, limiting scalability for real-time applications. Moreover, the model's interpretability might be a concern, as deep learning architectures often operate as black boxes, making it challenging to understand the reasoning behind specific predictions.

8.3 Future Works:

Moving forward, several avenues for future research and development can be explored to address the limitations and enhance the capabilities of our project:

1. Data Augmentation and Diversity: Augmenting the training data with diverse question pairs and addressing biases can improve the model's robustness and generalization ability.

2. Model Optimization: Exploring techniques for model optimization and compression can reduce the computational burden associated with deploying deep learning models, enabling real-time inference on resource-constrained platforms.

3. Interpretability: Investigating methods for interpreting and explaining the model's predictions can enhance trust and facilitate human-AI collaboration.

4. Integration with User Feedback: Incorporating user feedback mechanisms into the model training pipeline can enable continuous learning and adaptation to evolving user preferences and content dynamics.

5. Extension to Multimodal Data: Extending the model to handle multimodal data (e.g., text paired with images or audio) can further enrich the understanding of question similarities and enhance the user experience.

In summary, while our project represents a significant step towards automating the task of question pair similarity assessment on platforms like Quora, there exist exciting opportunities for further research and innovation to address existing challenges and unlock new possibilities in this domain.

REFERENCES

[1] Raschka, S., & Mirjalili, V. (2019). Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow (3rd ed.). Packt Publishing.

[2] Géron, A. (2019). Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems (2nd ed.). O'Reilly Media.

[3] Brownlee, J. (2020). Deep Learning for Natural Language Processing. Machine Learning Mastery. Retrieved from https://machinelearningmastery.com/deep-learning-for-nlp/.

[4] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & , S. (2016). TensorFlow: Large-scale Machine Learning on Heterogeneous Distributed Systems. arXiv preprint arXiv:1603.04467.

[5] Stack Overflow. (n.d.). Retrieved from https://stackoverflow.com/.

[6] OpenAI. (n.d.). ChatGPT. Retrieved from https://openai.com/chatgpt/.

[7] Gemini. (n.d.). Retrieved from https://www.gemini.com/.

[8] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical

[9] Learning: Data Mining, Inference, and Prediction. Springer Science & Business Media.

[10] Kaggle. (n.d.). Retrieved from https://www.kaggle.com/.

[11] GitHub. (n.d.). Retrieved from https://github.com/.