|  |  |
| --- | --- |
|  |  |
| AI Project Final Report |  |
|  |  |
|  | Abdul Rafay 21-NTU-CS-1197Arooba Zaman 21-NTU-CS-1211 |
|  | Submitted to: Mam Sajida |

**Table of Contents**

[**Table of Figures** 4](#_Toc168905744)

[**List of Tables** 4](#_Toc168905745)

[**1. Introduction** 5](#_Toc168905746)

[**2. Problem Statement** 5](#_Toc168905747)

[**3. Motivation** 6](#_Toc168905748)

[**4. Objective** 6](#_Toc168905759)

[**5. Dataset Description** 7](#_Toc168905760)

[**6. Methodology** 8](#_Toc168905761)

[**7. Flowchart of the Workflow** 11](#_Toc168905762)

[**8. Pre-Processing Technique** 11](#_Toc168905763)

[**9. Feature Engineering** 15](#_Toc168905764)

[**10. EDA (Exploratory Data analysis):** 20](#_Toc168905765)

[**11. Vectorization** 27](#_Toc168905767)

[**12. Test and Train data** 28](#_Toc168905768)

[**13. Model Training and Evaluation** 29](#_Toc168905769)

[**14. Modules and Tools used** 31](#_Toc168905772)

[**15. Confusion Matrix** 31](#_Toc168905773)

[**16. Accuracy and Precision of Model** 32](#_Toc168905774)

[**17. Example of similar Applications** 38](#_Toc168905776)

[**18. Conclusion** 39](#_Toc168905777)

## **Table of Figures**

[**Figure 1 7**](#_Toc168830964)

[**Figure 2 8**](#_Toc168830965)

[**Figure 3 8**](#_Toc168830966)

[**Figure 4 9**](#_Toc168830967)

[**Figure 5 20**](#_Toc168830968)

[**Figure 6 20**](#_Toc168830969)

[**Figure 7 21**](#_Toc168830970)

[**Figure 8 21**](#_Toc168830971)

[**Figure 9 22**](#_Toc168830972)

[**Figure 10 23**](#_Toc168830973)

[**Figure 11 23**](#_Toc168830974)

[**Figure 12 25**](#_Toc168830975)

[**Figure 13 30**](#_Toc168830976)

[**Figure 14 30**](#_Toc168830977)

[**Figure 15 32**](#_Toc168830978)

[**Figure 16 33**](#_Toc168830979)

[**Figure 17 33**](#_Toc168830980)

[**Figure 18 34**](#_Toc168830981)

[**Figure 19 34**](#_Toc168830982)

[**Figure 20 35**](#_Toc168830983)

[**Figure 21 35**](#_Toc168830984)

[**Figure 22 36**](#_Toc168830985)

[**Figure 23 36**](#_Toc168830986)

## **List of Tables**

[**Table 1 6**](#_Toc168830996)

[**Table 2 6**](#_Toc168830997)

[**Table 3 31**](#_Toc168830998)

[**Table 4 31**](#_Toc168830999)

## **1. Introduction**

This project aims to tackle this challenge by harnessing the capabilities of machine learning, particularly leveraging algorithms like XGBoost and RandomForest. Additionally, by integrating the ChatGPT API, this initiative seeks to enhance the sophistication of the detection process, providing concise and informative responses to identified duplicates.

## **2. Problem Statement**

Within the intricate ecosystem of online platforms, duplicate question detection emerges as a pivotal issue, exerting profound implications on the efficacy and user satisfaction of question answering systems, forums, and online communities. The omnipresence of duplicate queries not only inundates platforms with redundant content but also undermines the navigational prowess of search functionalities. Consequently, there arises an imperative need for automated solutions equipped to swiftly and accurately discern duplicate questions.

Conventional approaches to duplicate question detection often hinge on manual scrutiny or rudimentary rule-based frameworks, both of which are laborious and fraught with inaccuracies, particularly within domains boasting extensive troves of user-generated content. Hence, there is an exigency for sophisticated machine learning methodologies adept at discerning nuanced semantic parallels between question pairs.

This endeavor seeks to confront this challenge head-on by harnessing the potency of machine learning, with a specific emphasis on harnessing algorithms like XGBoost and RandomForest. Moreover, by seamlessly integrating the ChatGPT API, this initiative aspires to augment the cognitive faculties of the detection process, furnishing concise and enlightening responses to identified duplicates.

## **3. Motivation**

## **Enhancing Question-Answering Systems**

## Improving the efficiency and accuracy of question-answering systems and online forums.

## **Streamlining Content**

## Detecting duplicate questions to streamline content and provide users with relevant answers.

## **Improving User Satisfaction**

## Enhancing user satisfaction and engagement by delivering precise and helpful responses.

## **Integration of ChatGPT API**

## Adding another layer of intelligence through the integration of the ChatGPT API for concise and informative responses.

## **MySQL Database Integration:**

## Implementing a MySQL database to store user registration data and question-answer pairs. This includes allowing users to register and login, submit questions, and fetch answers for duplicate questions stored in the database. If a question is not a duplicate, an answer is generated using the ChatGPT API and stored in the database.

## **4. Objective**

The primary objective of this project is to create a robust model for accurately identifying duplicate questions. The specific goals encompass various stages of the project, including:

* **Preprocessing Techniques:**

Implementing advanced preprocessing techniques to cleanse and standardize text data, ensuring high-quality input for model training.

* **Feature Extraction:**

Extracting relevant features from pairs of questions to effectively capture both similarities and differences between them.

* **Model Training:**

Utilizing state-of-the-art machine learning algorithms such as XGBoost and RandomForest to train the model on the extracted features.

* **Performance Evaluation:**

Evaluating the performance of the trained models using a comprehensive set of metrics, including accuracy, precision, and confusion matrix analysis.

* **Integration of ChatGPT API:**

Integrating the ChatGPT API to enhance the intelligence of the detection process, providing concise and informative responses for identified duplicate questions.

* **User Interface Development:**

Developing an intuitive and user-friendly interface using Streamlit framework, facilitating seamless interaction with the model and its functionalities.

These objectives collectively aim to deliver a sophisticated solution for duplicate question detection, leveraging advanced techniques and technologies to optimize content quality and user experience in online platforms and communities.

## **5. Dataset Description**

1. **Dataset Background and Collection:**
   * **Source:** The dataset was obtained from a competition hosted by Quora on Kaggle, focusing on identifying duplicate question pairs.
   * **Collection:** Quora actively collects data on questions and answers from users. This dataset consists of question pairs provided by Quora, along with labels indicating duplicate or non-duplicate pairs.

Table 1

|  |  |  |
| --- | --- | --- |
| Classes | Attributes | Observations |
| 2 | id, qid1, qid2, question1, question2, is\_duplicate | Over 400,000 entries |

Table 2

|  |  |
| --- | --- |
| Variables | Description |
| id | Unique identifier for each question pair |
| qid1, qid2 | Unique identifiers for the questions in each pair |
| question1, question2 | Text of the questions |
| is\_duplicate | Binary variable indicating duplicate status |

1. **Data Collection Frequency and Justification:**
   * **Frequency:** Not explicitly stated, likely collected over time.
   * **Justification:**
     + Real-world relevance and diverse data.
     + Task relevance in information retrieval and NLP.
     + Large dataset size for robust model training.
   * **Choice Justification:** Chosen for real-world applicability, availability of labeled data, and active community engagement on Kaggle.
2. **Dataset Attributes and Class Labels:**

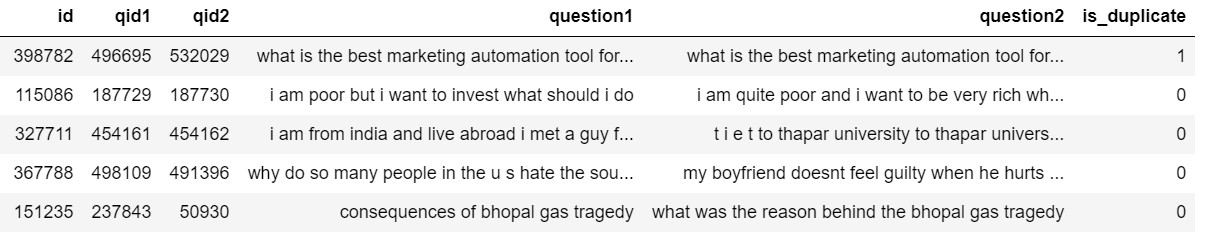


Figure 1

1. **Dataset Link:**

<https://www.kaggle.com/c/quora-question-pairs/data>

## **6. Methodology**

We have meticulously designed a methodology that amalgamates the robustness of the Random Forest algorithm, the user-friendly interface of Streamlit, and the advanced response generation capabilities of the ChatGPT API. This fusion empowers our system to effectively identify duplicate questions and furnish pertinent responses seamlessly.

**1.Random Forest and XGBoost:**

Initially, we explored the potential of both the Random Forest and XGBoost algorithms for classification tasks, particularly in discerning duplicate questions. While XGBoost offers exceptional performance in many scenarios, we found that Random Forest outperformed it in our specific context. Hence, we ultimately transitioned to utilizing Random Forest for its robustness in handling vast datasets characterized by high dimensionality and intricate relationships.

**Objective Function for XGboost:**

XGBoost aims to minimize a regularized objective function:

**L(Θ)=∑(y^i,yi)+ ∑Ω(fk)**

**Random Forest:**

* For a classification problem, the final prediction is obtained by majority voting: **y^=model{Tb(x)}b=1**
* For a regression problem, the final prediction is obtained by averaging the predictions: **y^=1/B∑Tb(x)**

**Random Forest:**

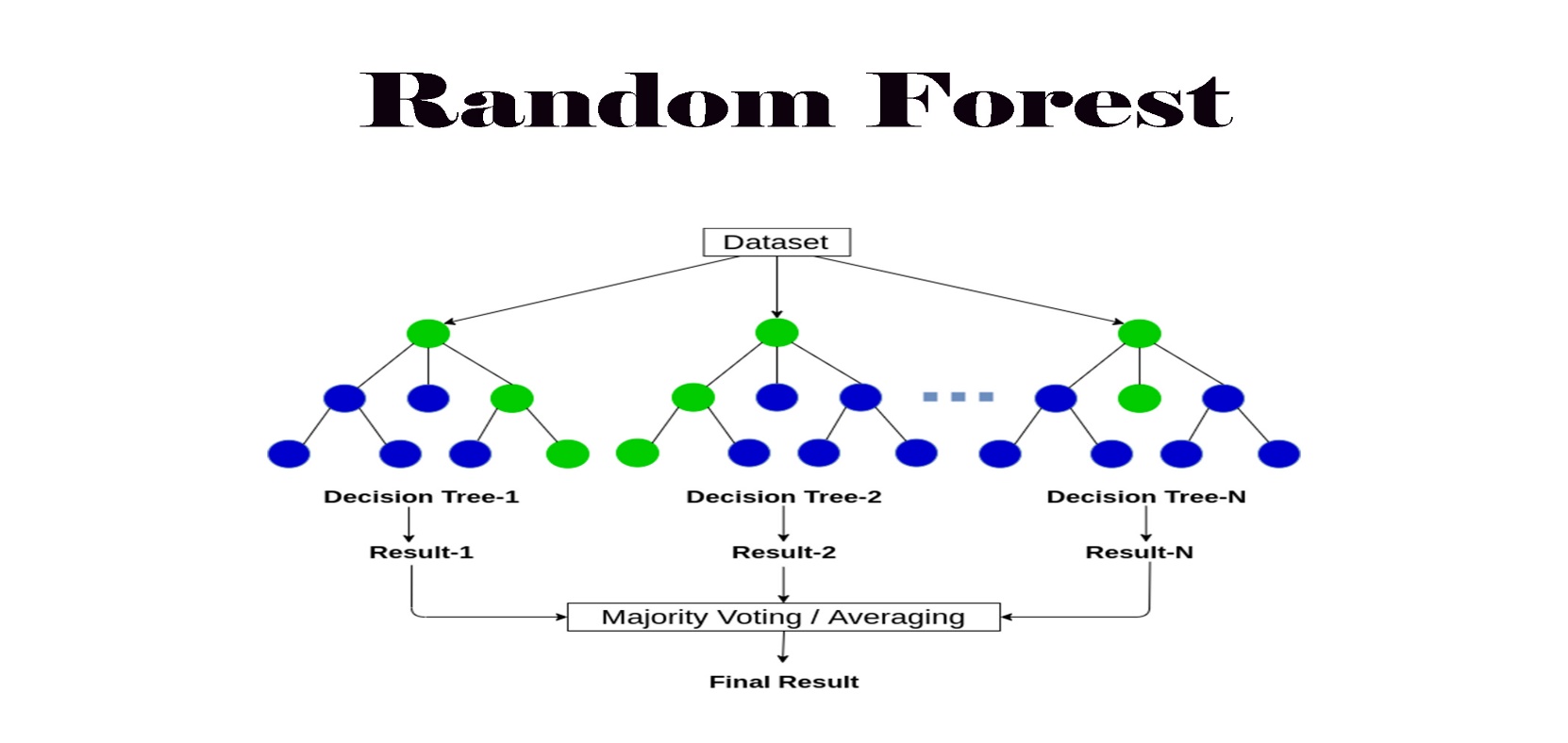


Figure 2

**XGBoost:**

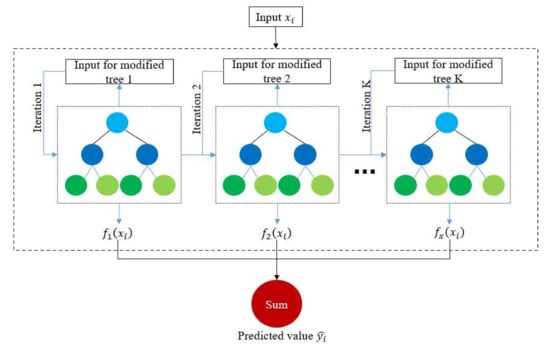


Figure 3

**2.Streamlit UI:**

To ensure a seamless user experience, we harnessed Streamlit, a renowned framework for building web applications using Python. Streamlit facilitates the creation of dynamic and interactive interfaces directly from Python scripts, obviating the need for extensive frontend development. Through our Streamlit UI, users can effortlessly input queries, visualize model predictions, and explore outcomes in real-time.

**3.ChatGPT API Integration:**

For the generation of comprehensive answers to user queries, we seamlessly integrated the ChatGPT API into our application. ChatGPT, an avant-garde conversational AI model developed by OpenAI, boasts the capability to generate responses akin to those of humans based on the input it receives. Leveraging the ChatGPT API enables our application to furnish informative and contextually relevant responses, augmenting the overall user experience.

**4.MySQL Database:**

We designed a MySQL database to store user registration data and question-answer pairs. This database is structured with a **users** table for storing user credentials and a **questions** table for storing questions and their corresponding answers. During registration, user data is securely stored in the database. Upon login, authenticated users can submit their questions. If the submitted question is a duplicate of a previously asked question in the database, the system fetches and returns the corresponding answer. If it is not a duplicate, the system generates an answer using the ChatGPT API and saves both the question and answer in the database.

**4.Workflow Overview:**

* **User Interaction**: Users engage with the Streamlit UI by inputting questions they wish to evaluate for duplicates.
* **Model Prediction**: Input questions undergo preprocessing and are fed into the Random Forest model for prediction.
* **Visualization**: Predictions are visualized and presented to users through the intuitive Streamlit UI.
* **ChatGPT Response**: In instances where users seek additional information or clarification, the ChatGPT API is invoked to generate informative responses.
* **Seamless Interaction**: Users receive model predictions alongside additional answers generated by ChatGPT, fostering a seamless and informative interaction experience.

By synergistically combining the Random Forest algorithm for question matching, Streamlit for interface development, and ChatGPT API for response generation, our methodology facilitates an efficient and user-friendly solution for identifying duplicate questions and delivering relevant responses.

## **7. Flowchart of the Workflow**

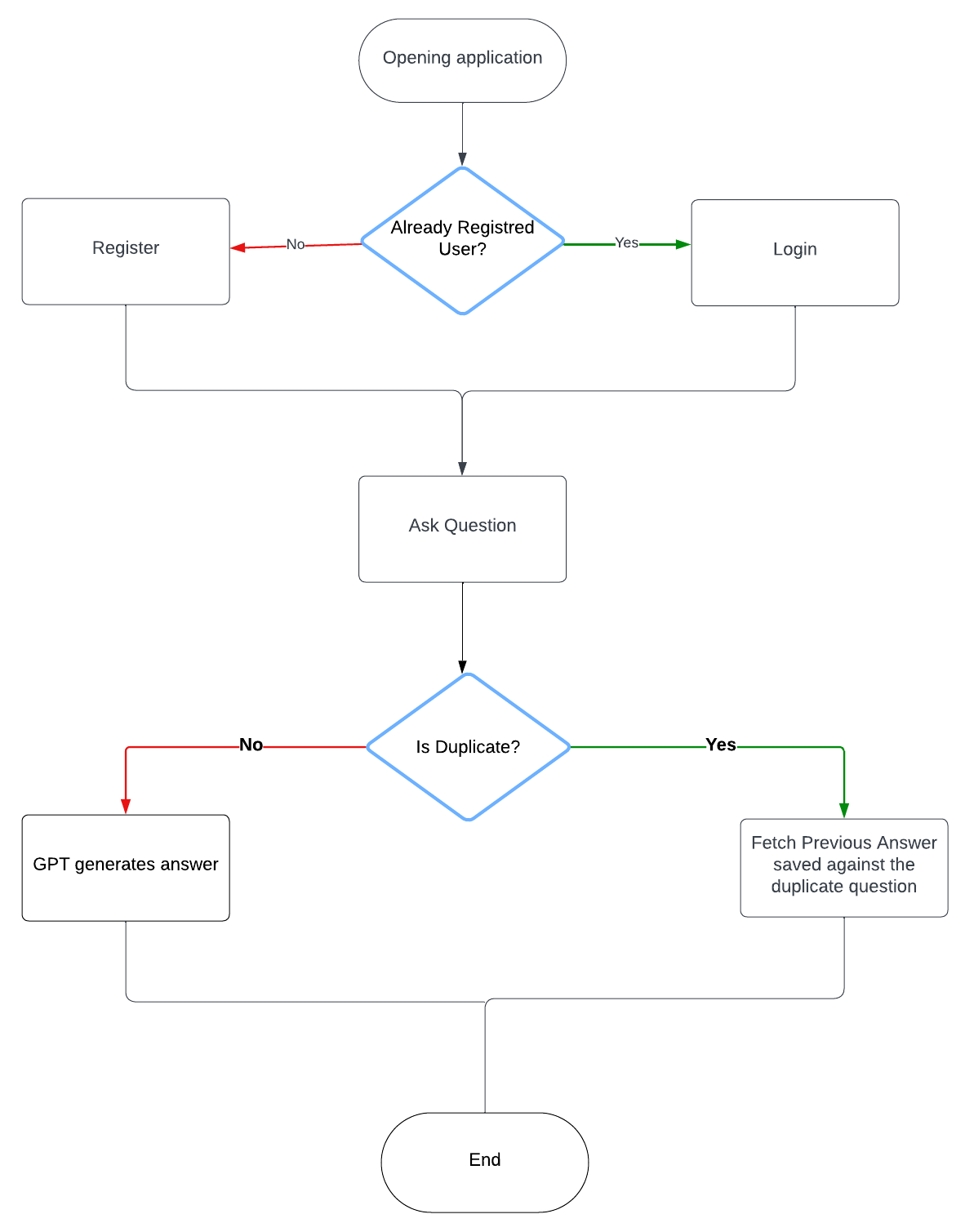


Figure 4

## **8. Pre-Processing Technique**

**Pseudocode:**

|  |
| --- |
|  |
| FUNCTION preprocess\_text(q): |
| q = to\_lowercase(q) |
| q = strip\_whitespace(q) |
| q = replace\_special\_characters(q) |
| q = remove\_unnecessary\_patterns(q) |
| q = replace\_large\_numbers(q) |
| q = decontract\_words(q) |
| q = remove\_html\_tags(q) |
| q = remove\_punctuations(q) |
| RETURN q |
|  |
| FUNCTION to\_lowercase(q): |
| RETURN q converted to lowercase |
|  |
| FUNCTION strip\_whitespace(q): |
| RETURN q with leading and trailing whitespace removed |
|  |
| FUNCTION replace\_special\_characters(q): |
| special\_characters = {'%': ' percent', '$': ' dollar', '₹': ' rupee', '€': ' euro', '@': ' at'} |
| FOR each character in q: |
| IF character is in special\_characters: |
| REPLACE character with corresponding value in special\_characters |
| RETURN q |
|  |
| FUNCTION remove\_unnecessary\_patterns(q): |
| RETURN q with '[math]' pattern removed |
|  |
| FUNCTION replace\_large\_numbers(q): |
| large\_number\_patterns = {',000,000,000': 'b', ',000,000': 'm', ',000': 'k'} |
| FOR each pattern in large\_number\_patterns: |
| REPLACE pattern in q with corresponding value in large\_number\_patterns |
| RETURN q |
|  |
| FUNCTION decontract\_words(q): |
| contractions = {'ain't': 'am not', "aren't": 'are not', ...} |
| FOR each word in q.split(): |
| IF word is in contractions: |
| REPLACE word with corresponding value in contractions |
| RETURN q |
|  |
| FUNCTION remove\_html\_tags(q): |
| RETURN textual content extracted from q using BeautifulSoup |
|  |
| FUNCTION remove\_punctuations(q): |
| punctuation\_pattern = compile\_pattern('\W') |
| q = replace\_matches(punctuation\_pattern, ' ', q) |
| RETURN q stripped of all non-word characters |
|  |
| FUNCTION feature\_engineering(new\_df): |
| new\_df['q1\_len'] = calculate\_length(new\_df['question1']) |
| new\_df['q2\_len'] = calculate\_length(new\_df['question2']) |
| new\_df['q1\_num\_words'] = calculate\_num\_words(new\_df['question1']) |
| new\_df['q2\_num\_words'] = calculate\_num\_words(new\_df['question2']) |
| new\_df['word\_common'] = calculate\_common\_words(new\_df) |
| new\_df['word\_total'] = calculate\_total\_words(new\_df) |
| new\_df['word\_share'] = calculate\_word\_share(new\_df) |
| RETURN new\_df |
|  |
| FUNCTION calculate\_length(question): |
| RETURN length of question |
|  |
| FUNCTION calculate\_num\_words(question): |
| RETURN number of words in question |
|  |
| FUNCTION calculate\_common\_words(new\_df): |
| FOR each row in new\_df: |
| Calculate common words between question1 and question2 |
| RETURN common words count |
|  |
| FUNCTION calculate\_total\_words(new\_df): |
| FOR each row in new\_df: |
| Calculate total words in both questions combined |
| RETURN total words count |
|  |
| FUNCTION calculate\_word\_share(new\_df): |
| FOR each row in new\_df: |
| Calculate word share ratio between common words and total words |
| RETURN word share ratio |

The preprocessing of text data is a crucial step in preparing the dataset for further analysis and model training. Below are the steps taken to preprocess the text data in this study:

**1. Lowercasing and Stripping:**

|  |
| --- |
| **q = str(q).lower().strip()** |

Converts the entire text to lowercase to ensure uniformity and removes any leading or trailing whitespace.

**2. Replacing Special Characters:**

|  |
| --- |
| **q = q.replace('%'' percent')**  **q = q.replace('$'' dollar ')**  **q = q.replace('₹'' rupee ')**  **q = q.replace('€'' euro ')**  **q = q.replace('@'' at ')** |

Replaces certain special characters with their corresponding textual equivalents to make the text more readable and consistent.

**3. Removing Unnecessary Patterns:**

|  |
| --- |
| **q = q.replace('[math]''')** |

The pattern [math] is removed as it does not contribute to the textual analysis and appears frequently in the dataset.

**4. Replacing Large Numbers with String Equivalents:**

|  |
| --- |
| **q = q.replace('000000000 ''b ')**  **q = q.replace('000000 ''m ')**  **q = q.replace('000 ''k ')**  **q = re.sub(r'([0-9]+)000000000'r'\1b'q)**  **q = re.sub(r'([0-9]+)000000'r'\1m'q)**  **q = re.sub(r'([0-9]+)000'r'\1k'q)** |

Converts large numerical values into their abbreviated forms (e.g., million to 'm', billion to 'b') for simplification and readability.

**5. Decontracting Words:**

|  |
| --- |
| **contractions = {** |
| **"ain't": "am not",** |
| **"aren't": "are not",** |
| **...** |
| **"you're": "you are",** |
| **"you've": "you have"** |
| **}** |
|  |
| **q\_decontracted = []** |
|  |
| **for word in q.split():** |
| **if word in contractions:** |
| **word = contractions[word]** |
| **q\_decontracted.append(word)** |
|  |
| **q = ' '.join(q\_decontracted)** |
| **q = q.replace("'ve", " have")** |
| **q = q.replace("n't", " not")** |
| **q = q.replace("'re", " are")** |
| **q = q.replace("'ll", " will")** |
|  |

Expands common contractions into their full forms (e.g., "can't" to "cannot", "you're" to "you are") to ensure consistency and clarity in the text.

**6. Removing HTML Tags:**

|  |
| --- |
| **q = BeautifulSoup(q).get\_text()** |

Utilizes BeautifulSoup to strip out any HTML tags that may be present in the text, leaving only the raw textual content.

**7. Removing Punctuations:**

|  |
| --- |
|  |
| **pattern = re.compile('\W')** |
| **q = re.sub(pattern, ' ', q).strip()** |

Removes all non-word characters (punctuation marks) to clean the text further and avoid unnecessary tokens during analysis.

## **9. Feature Engineering**

In this section, we perform feature engineering on our dataset to extract various characteristics from the question pairs. These features help in understanding the similarity between the questions and are crucial for identifying potential duplicates.

**Pseudocode :**

|  |
| --- |
| FUNCTION calculate\_basic\_features(new\_df): |
| new\_df['q1\_len'] = calculate\_question\_length(new\_df['question1']) |
| new\_df['q2\_len'] = calculate\_question\_length(new\_df['question2']) |
| new\_df['q1\_num\_words'] = calculate\_word\_count(new\_df['question1']) |
| new\_df['q2\_num\_words'] = calculate\_word\_count(new\_df['question2']) |
| new\_df['word\_common'] = calculate\_common\_words(new\_df) |
| new\_df['word\_total'] = calculate\_total\_words(new\_df) |
| new\_df['word\_share'] = calculate\_word\_share(new\_df) |
|  |
| FUNCTION calculate\_question\_length(question\_series): |
| FOR each question in question\_series: |
| Calculate the length of the question |
| RETURN length\_series |
|  |
| FUNCTION calculate\_word\_count(question\_series): |
| FOR each question in question\_series: |
| Count the number of words in the question |
| RETURN word\_count\_series |
|  |
| FUNCTION calculate\_common\_words(new\_df): |
| FOR each row in new\_df: |
| Calculate the number of common words between question1 and question2 |
| RETURN common\_words\_series |
|  |
| FUNCTION calculate\_total\_words(new\_df): |
| FOR each row in new\_df: |
| Calculate the total number of unique words in both questions combined |
| RETURN total\_words\_series |
|  |
| FUNCTION calculate\_word\_share(new\_df): |
| FOR each row in new\_df: |
| Calculate the ratio of common words to total words |
| RETURN word\_share\_series |
|  |
| FUNCTION calculate\_advanced\_features(new\_df): |
| new\_df['token\_features'] = new\_df.apply(fetch\_token\_features, axis=1) |
| new\_df['length\_features'] = new\_df.apply(fetch\_length\_features, axis=1) |
| new\_df['fuzzy\_features'] = new\_df.apply(fetch\_fuzzy\_features, axis=1) |
|  |
| FUNCTION fetch\_token\_features(row): |
| q1 = row['question1'] |
| q2 = row['question2'] |
| token\_features = [0.0] \* 8 |
| # Calculate token features |
| ... |
| RETURN token\_features |
|  |
| FUNCTION fetch\_length\_features(row): |
| q1 = row['question1'] |
| q2 = row['question2'] |
| length\_features = [0.0] \* 3 |
| # Calculate length features |
| ... |
| RETURN length\_features |
|  |
| FUNCTION fetch\_fuzzy\_features(row): |
| q1 = row['question1'] |
| q2 = row['question2'] |
| fuzzy\_features = [0.0] \* 4 |
| # Calculate fuzzy features |
| ... |
| RETURN fuzzy\_features |

**Basic Features**

**1. Length of Each Question**

We calculate the length of each question to understand the textual difference between them.

|  |
| --- |
| **# Length of each question** |
| **new\_df['q1\_len'] = new\_df['question1'].str.len()** |
| **new\_df['q2\_len'] = new\_df['question2'].str.len()** |

**Reason:** The length of the questions might indicate if one question is more detailed or verbose than the other. Significant differences in length could suggest the questions are not duplicates.

**2. Number of Words in Each Question**

We count the number of words in each question.

|  |
| --- |
| **# Number of words in each question** |
| **new\_df['q1\_num\_words'] = new\_df['question1'].apply(lambda row: len(row.split(" ")))** |
| **new\_df['q2\_num\_words'] = new\_df['question2'].apply(lambda row: len(row.split(" ")))** |

**Reason:** Similar to the length, the word count helps in assessing the complexity and verbosity of the questions. Questions with a similar number of words are more likely to be duplicates.

**3. Common Words**

We calculate the number of common words between the two questions.

|  |
| --- |
| **# Function to count common words** |
| **def common\_words(row):** |
| **w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))** |
| **w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))** |
| **return len(w1 & w2)** |
|  |
| **# Apply the function to get common words count** |
| **new\_df['word\_common'] = new\_df.apply(common\_words, axis=1)** |

**Reason:** Common words between the questions indicate overlapping content and context, which is a strong signal of potential duplication.

**3. Total Words**

We calculate the total number of unique words in both questions combined.

|  |
| --- |
|  |
| **# Function to count total words** |
| **def total\_words(row):** |
| **w1 = set(map(lambda word: word.lower().strip(), row['question1'].split(" ")))** |
| **w2 = set(map(lambda word: word.lower().strip(), row['question2'].split(" ")))** |
| **return (len(w1) + len(w2))** |
|  |
| **# Apply the function to get total words count** |
| **new\_df['word\_total'] = new\_df.apply(total\_words, axis=1)** |
|  |
| **# Calculate word share** |
| **new\_df['word\_share'] = round(new\_df['word\_common'] / new\_df['word\_total'], 2)** |

**Reason:** The total number of unique words helps in understanding the overall vocabulary used in both questions. The ratio of common words to total words (word share) gives an indication of how similar the two questions are.

**Advanced Features**

**1. Token Features**

We extract features based on token overlap, stop words, and position of tokens.

|  |
| --- |
| **from nltk.corpus import stopwords** |
|  |
| **def fetch\_token\_features(row):** |
| **q1 = row['question1']** |
| **q2 = row['question2']** |
|  |
| **SAFE\_DIV = 0.0001** |
| **STOP\_WORDS = stopwords.words("english")** |
|  |
| **token\_features = [0.0] \* 8** |
|  |
| **q1\_tokens = q1.split()** |
| **q2\_tokens = q2.split()** |
|  |
| **if len(q1\_tokens) == 0 or len(q2\_tokens) == 0:** |
| **return token\_features** |
|  |
| **q1\_words = set([word for word in q1\_tokens if word not in STOP\_WORDS])** |
| **q2\_words = set([word for word in q2\_tokens if word not in STOP\_WORDS])** |
|  |
| **q1\_stops = set([word for word in q1\_tokens if word in STOP\_WORDS])** |
| **q2\_stops = set([word for word in q2\_tokens if word in STOP\_WORDS])** |
|  |
| **common\_word\_count = len(q1\_words.intersection(q2\_words))** |
| **common\_stop\_count = len(q1\_stops.intersection(q2\_stops))** |
| **common\_token\_count = len(set(q1\_tokens).intersection(set(q2\_tokens)))** |
|  |
| **token\_features[0] = common\_word\_count / (min(len(q1\_words), len(q2\_words)) + SAFE\_DIV)** |
| **token\_features[1] = common\_word\_count / (max(len(q1\_words), len(q2\_words)) + SAFE\_DIV)** |
| **token\_features[2] = common\_stop\_count / (min(len(q1\_stops), len(q2\_stops)) + SAFE\_DIV)** |
| **token\_features[3] = common\_stop\_count / (max(len(q1\_stops), len(q2\_stops)) + SAFE\_DIV)** |
| **token\_features[4] = common\_token\_count / (min(len(q1\_tokens), len(q2\_tokens)) + SAFE\_DIV)** |
| **token\_features[5] = common\_token\_count / (max(len(q1\_tokens), len(q2\_tokens)) + SAFE\_DIV)** |
| **token\_features[6] = int(q1\_tokens[-1] == q2\_tokens[-1])** |
| **token\_features[7] = int(q1\_tokens[0] == q2\_tokens[0])** |
|  |
| **return token\_features** |

**Reason:** These features capture various aspects of token similarity, including overlap of non-stop words, stop words, and overall token overlap. They also check if the first and last words are the same, which can be indicative of similar questions.

**2. Length Features**

We measure differences in length and the longest common substring between the questions.

|  |
| --- |
| **import distance** |
|  |
| **def fetch\_length\_features(row):** |
| **q1 = row['question1']** |
| **q2 = row['question2']** |
|  |
| **length\_features = [0.0] \* 3** |
|  |
| **q1\_tokens = q1.split()** |
| **q2\_tokens = q2.split()** |
|  |
| **if len(q1\_tokens) == 0 or len(q2\_tokens) == 0:** |
| **return length\_features** |
|  |
| **length\_features[0] = abs(len(q1\_tokens) - len(q2\_tokens))** |
| **length\_features[1] = (len(q1\_tokens) + len(q2\_tokens)) / 2** |
|  |
| **strs = list(distance.lcsubstrings(q1, q2))** |
| **length\_features[2] = len(strs[0]) / (min(len(q1), len(q2)) + 1)** |
|  |
| **return length\_features** |

**3. Fuzzy Features**

We use fuzzy matching to calculate similarity scores.

|  |
| --- |
| **from fuzzywuzzy import fuzz** |
|  |
| **def fetch\_fuzzy\_features(row):** |
| **q1 = row['question1']** |
| **q2 = row['question2']** |
|  |
| **fuzzy\_features = [0.0] \* 4** |
|  |
| **fuzzy\_features[0] = fuzz.QRatio(q1, q2)** |
| **fuzzy\_features[1] = fuzz.partial\_ratio(q1, q2)** |
| **fuzzy\_features[2] = fuzz.token\_sort\_ratio(q1, q2)** |
| **fuzzy\_features[3] = fuzz.token\_set\_ratio(q1, q2)** |
|  |
| **return fuzzy\_features** |
|  |

**Reason:** Fuzzy matching provides various measures of similarity, such as token sorting and token set ratios, which are useful for capturing different types of textual similarity.

## **10. EDA (Exploratory Data analysis):**

1. **Word Cloud**

|  |
| --- |
| **from wordcloud import WordCloud** |
| **import matplotlib.pyplot as plt** |
|  |
| **# Combine all the text from the questions** |
| **all\_text = ' '.join(new\_df['question1'].astype(str) + ' ' + new\_df['question2'].astype(str))** |
|  |
| **# Generate the word cloud** |
| **wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(all\_text)** |
|  |
| **# Display the word cloud** |
| **plt.figure(figsize=(10, 5))** |
| **plt.imshow(wordcloud, interpolation='bilinear')** |
| **plt.axis('off')** |
| **plt.show()** |

## 

Figure 5

1. **Pair Plot**

The pair plot created by this code snippet is a powerful visualization tool that allows you to explore the pairwise relationships between features and their distributions, segmented by the target variable (**is\_duplicate**).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **sns.pairplot(new\_df[['ctc\_min'** | **'cwc\_min'** | **'csc\_min'** | **'is\_duplicate']]** | **hue='is\_duplicate')** |

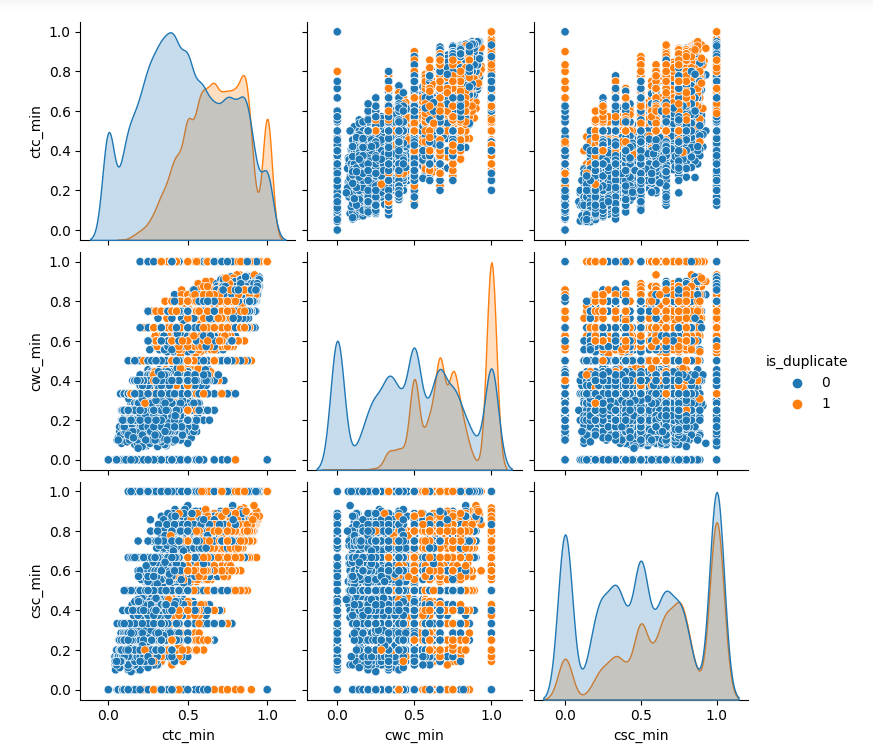


Figure 6

|  |
| --- |
| **sns.pairplot(new\_df[['ctc\_max'**  **'cwc\_max'**  **'csc\_max'**  **'is\_duplicate']]**  **hue='is\_duplicate')** |

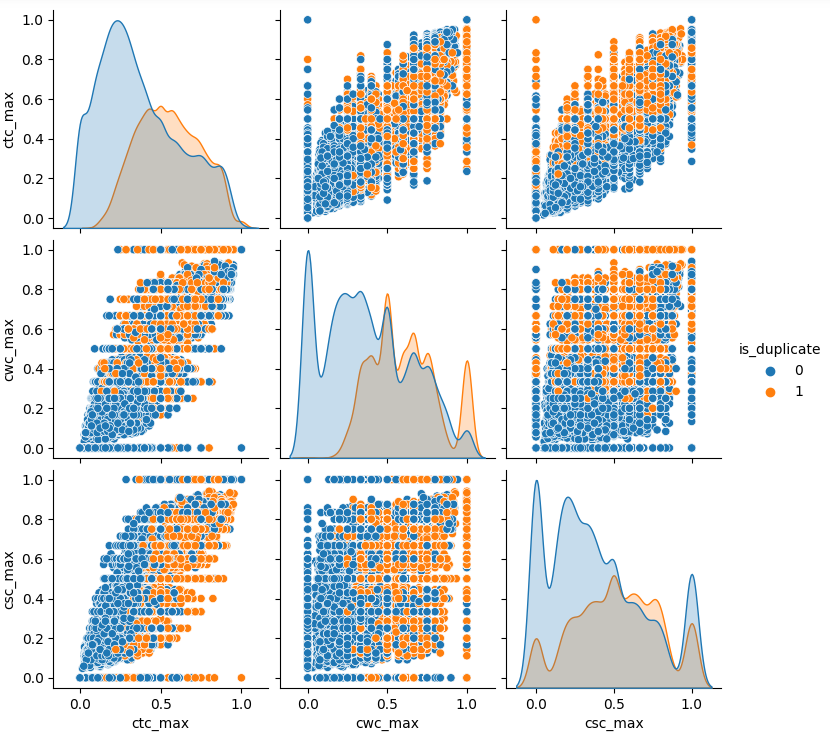


Figure 7

|  |  |  |  |
| --- | --- | --- | --- |
| **sns.pairplot(new\_df[['last\_word\_eq'** | **'first\_word\_eq'** | **'is\_duplicate']]** | **hue='is\_duplicate')** |

A graph of a number of numbers

Description automatically generated with medium confidence

Figure 8

|  |
| --- |
| **sns.pairplot(new\_df[['mean\_len'**  **'abs\_len\_diff'**  **'longest\_substr\_ratio'**  **'is\_duplicate']]**  **hue='is\_duplicate')** |

A group of graphs showing different sizes of data

Description automatically generated with medium confidence

Figure 9

|  |
| --- |
| **sns.pairplot(new\_df[['fuzz\_ratio'**  **'fuzz\_partial\_ratio'**  **'token\_sort\_ratio'**  **'token\_set\_ratio'**  **'is\_duplicate']]**  **hue='is\_duplicate')** |

A group of blue and orange dots

Description automatically generated

Figure 10

1. **2D Diagram**

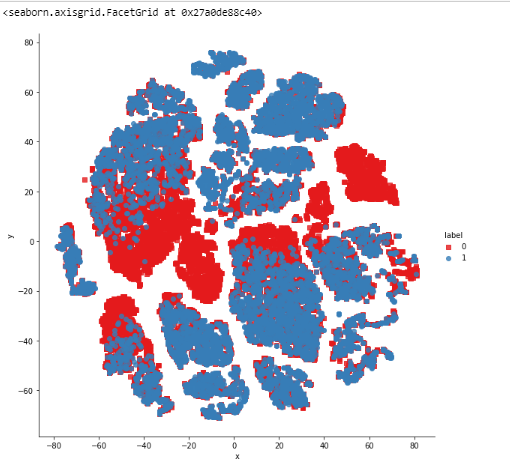
****

Figure 11

1. **3D Diagram**

|  |
| --- |
| **import plotly.graph\_objs as go** |
| **import plotly.tools as tls** |
| **import plotly.offline as py** |
| **py.init\_notebook\_mode(connected=True)** |
|  |
| **trace1 = go.Scatter3d(** |
| **x=tsne3d[:,0],** |
| **y=tsne3d[:,1],** |
| **z=tsne3d[:,2],** |
| **mode='markers',** |
| **marker=dict(** |
| **sizemode='diameter',** |
| **color = y,** |
| **colorscale = 'Portland',** |
| **colorbar = dict(title = 'duplicate'),** |
| **line=dict(color='rgb(255, 255, 255)'),** |
| **opacity=0.75** |
| **)** |
| **)** |
|  |
| **data=[trace1]** |
| **layout=dict(height=800, width=800, title='3d embedding with engineered features')** |
| **fig=dict(data=data, layout=layout)** |
| **py.iplot(fig, filename='3DBubble')** |

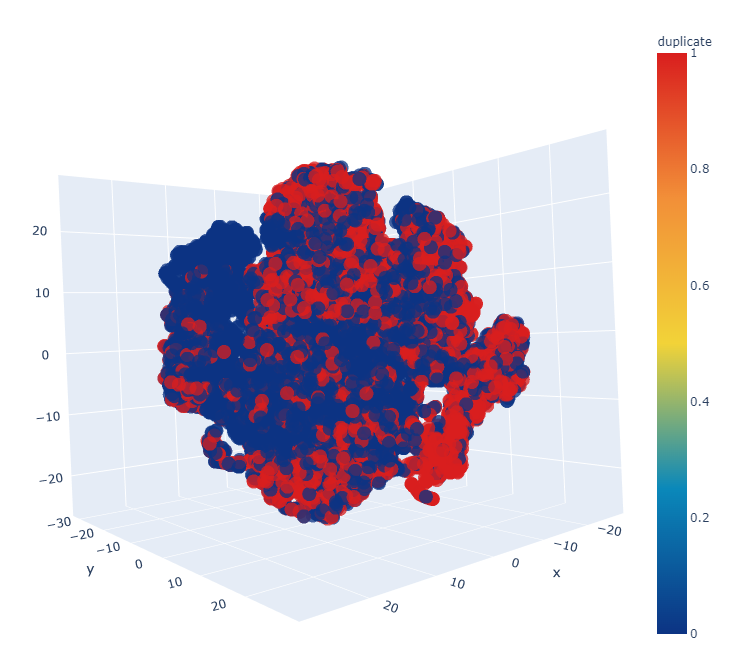


Figure 12

## **11. Vectorization**

**Pseudocode:**

|  |
| --- |
| FUNCTION vectorization(ques\_df): |
| IMPORT CountVectorizer FROM sklearn.feature\_extraction.text |
| IMPORT numpy as np |
|  |
| # Merge texts |
| questions = list(ques\_df['question1']) + list(ques\_df['question2']) |
|  |
| # Create CountVectorizer |
| cv = CountVectorizer(max\_features=3000) |
|  |
| # Transform questions |
| transformed\_questions = cv.fit\_transform(questions).toarray() |
|  |
| # Split transformed arrays |
| q1\_arr, q2\_arr = np.vsplit(transformed\_questions, 2) |
|  |
| RETURN q1\_arr, q2\_arr |
|  |
|  |

**Reason:** Bag of Words (BoW) model is a basic but powerful method for text representation. By converting text into numerical vectors, we can quantify the presence of words and compare the questions more effectively. Using a maximum of 3000 features ensures that we capture the most important words without making the feature space too large.

## **12. Test and Train data**

To evaluate the model's performance, we split the dataset into training and testing sets.

**Pseudocode:**

|  |
| --- |
| # Pseudocode for Train-Test Split |
|  |
| # Import train\_test\_split function from sklearn |
| import train\_test\_split from sklearn.model\_selection |
|  |
| # Split the dataset into features (X) and target variable (y) |
| features = final\_df.iloc[:, 1:].values |
| target = final\_df.iloc[:, 0].values |
|  |
| # Call the train\_test\_split function to split the data into training and testing sets |
| X\_train, X\_test, y\_train, y\_test = train\_test\_split(features, target, test\_size=0.2, random\_state=1) |

**Real code:**

|  |
| --- |
| **from sklearn.model\_selection import train\_test\_split** |
|  |
| **# Split the data into training and testing sets** |
| **X\_train, X\_test, y\_train, y\_test = train\_test\_split(final\_df.iloc[:, 1:].values, final\_df.iloc[:, 0].values, test\_size=0.2, random\_state=1)** |

**Reason:** Splitting the dataset into training and testing sets ensures that we can evaluate the model on data it hasn't seen during training. This helps in assessing the model's generalization ability and prevents overfitting.

## **13. Model Training and Evaluation**

After extracting features from our dataset, we proceed to train and evaluate machine learning models. We will use two different classifiers: Random Forest and XGBoost, to predict whether pairs of questions are duplicates.

### 1. Random Forest Classifier

**Pseudocode:**

|  |
| --- |
| FUNCTION train\_random\_forest\_classifier(X\_train, y\_train, X\_test, y\_test): |
| IMPORT RandomForestClassifier, accuracy\_score FROM sklearn.ensemble, sklearn.metrics |
|  |
| # Initialize the Random Forest classifier |
| rf = RandomForestClassifier() |
|  |
| # Train the model on the training data |
| rf.fit(X\_train, y\_train) |
|  |
| # Predict on the test data |
| y\_pred = rf.predict(X\_test) |
|  |
| # Calculate the accuracy of the model |
| rf\_accuracy = accuracy\_score(y\_test, y\_pred) |
|  |
| PRINT "Random Forest Accuracy:", rf\_accuracy |

We use a Random Forest classifier to train our model. Random Forest is an ensemble method that combines multiple decision trees to improve the model's accuracy and control overfitting.

**Real code:**

|  |
| --- |
| **FUNCTION train\_random\_forest\_classifier(X\_train, y\_train, X\_test, y\_test):** |
| **IMPORT RandomForestClassifier, accuracy\_score FROM sklearn.ensemble, sklearn.metrics** |
|  |
| **# Initialize the Random Forest classifier** |
| **rf = RandomForestClassifier()** |
|  |
| **# Train the model on the training data** |
| **rf.fit(X\_train, y\_train)** |
|  |
| **# Predict on the test data** |
| **y\_pred = rf.predict(X\_test)** |
|  |
| **# Calculate the accuracy of the model** |
| **rf\_accuracy = accuracy\_score(y\_test, y\_pred)** |
|  |
| **PRINT "Random Forest Accuracy:", rf\_accuracy** |

**Reason:** Random Forest is chosen for its robustness and ability to handle large datasets with higher dimensionality. It performs well in avoiding overfitting and provides good generalization on unseen data.

### 2. XGBoost Classifier

Next, we use the XGBoost classifier, a powerful gradient boosting algorithm known for its performance and efficiency in handling large datasets and complex relationships.

**Pseudocode:**

|  |
| --- |
| # Pseudocode for XGBoost Classifier Training and Evaluation |
|  |
| # Import XGBoost library |
| import xgboost |
|  |
| # Initialize XGBoost classifier |
| xgb\_classifier = xgboost.XGBClassifier() |
|  |
| # Train the classifier using the training data |
| xgb\_classifier.train(training\_data) |
|  |
| # Use the trained classifier to predict labels for the test data |
| predicted\_labels = xgb\_classifier.predict(test\_data) |
|  |
| # Calculate the accuracy of the predictions |
| accuracy = calculate\_accuracy(actual\_labels, predicted\_labels) |
|  |
| # Output the accuracy score |
| display\_accuracy(accuracy) |

**Real code:**

|  |
| --- |
| **from xgboost import XGBClassifier** |
|  |
| **# Initialize the XGBoost classifier** |
| **xgb = XGBClassifier()** |
|  |
| **# Train the model on the training data** |
| **xgb.fit(X\_train, y\_train)** |
|  |
| **# Predict on the test data** |
| **y\_pred1 = xgb.predict(X\_test)** |
|  |
| **# Calculate the accuracy of the model** |
| **xgb\_accuracy = accuracy\_score(y\_test, y\_pred1)** |
| **print(f"XGBoost Accuracy: {xgb\_accuracy}")** |

**Reason:** XGBoost is included due to its superior performance in various machine learning tasks, especially those involving classification. It optimizes the training process using advanced regularization techniques, which help in improving model accuracy and preventing overfitting.

## **14. Modules and Tools used**

**Python:**

*Description*: Python serves as the primary programming language for implementing the project.

**Libraries:**

* **Pandas, NumPy**:
  + *Description*: Pandas and NumPy are fundamental libraries for data manipulation and analysis.
* **Scikit-learn**:
  + *Description*: Scikit-learn provides a comprehensive suite of machine learning algorithms and evaluation metrics, facilitating model development and assessment.
* **XGBoost, RandomForest**:
  + *Description*: XGBoost and RandomForest are machine learning algorithms employed for model training. They offer robustness and efficiency in handling classification tasks, such as identifying duplicate questions.
* **ChatGPT API**:
  + *Description*: The ChatGPT API, developed by OpenAI, is utilized for generating single answers tailored to duplicate questions. It leverages state-of-the-art conversational AI capabilities to provide contextually relevant responses.
* **Streamlit**:
  + *Description*: Streamlit is a versatile framework utilized for building interactive web applications with Python. It facilitates the creation of dynamic user interfaces directly from Python scripts, simplifying the development process and enhancing user experience.

## **15. Confusion Matrix**

A confusion matrix is a useful tool in the AI cancer detection project for **assessing how well the generated model categorizes** duplicate questions. The confusion matrix shows how the model's predictions stack up against the actual ground truth labels.

* **For Random Forest** :

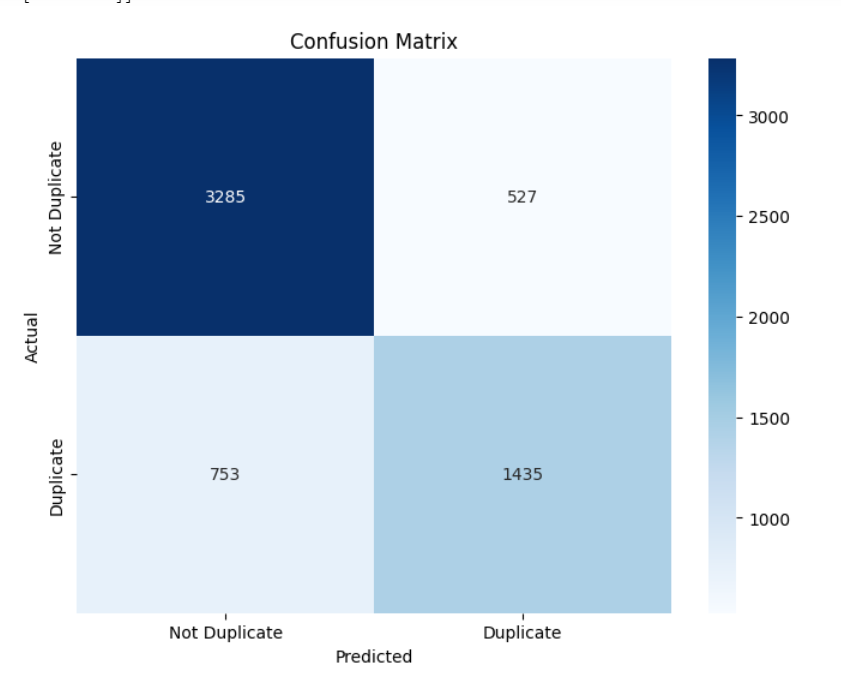


Figure 13

* **For XGBoost** :

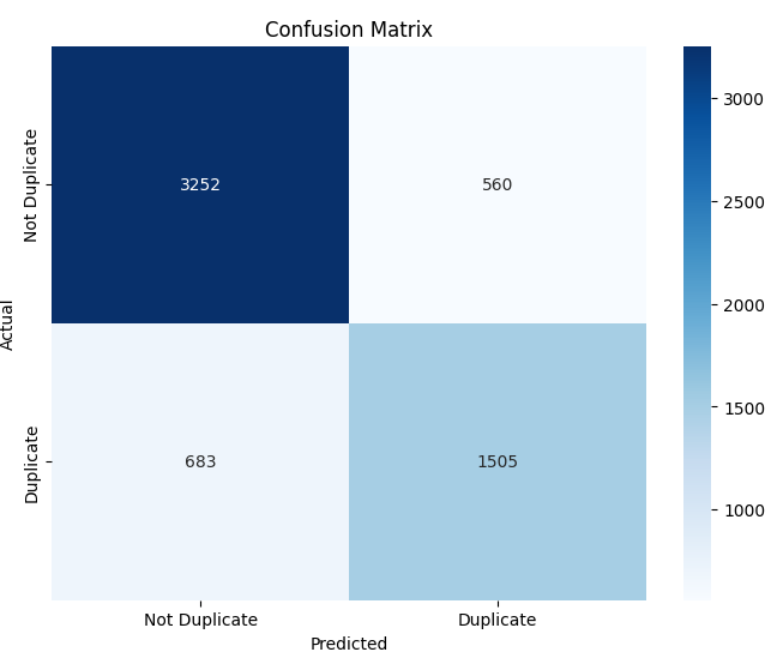


Figure 14

The confusion matrix created during the model evaluation is displayed in the accompanying image. The matrix is a square table with columns and rows, where each column represents the predicted labels, and each row represents the true labels of the samples. The **matrix's entries show the counts or percentages of samples that correspond to various combinations of the true and expected labels**.

## **16. Accuracy and Precision of Model**

**For Random Forest** :

Table 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.81 | 0.86 | 0.84 | 3812 |
| 1 | 0.733 | 0.66 | 0.69 | 2188 |
| accuracy |  |  | 0.79 | 6000 |
| macro avg | 0.766 | 0.766 | 0.766 | 6000 |
| weighted avg | 0.78 | 0.79 | 0.78 | 6000 |

**For XGBoost** :

Table 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.83 | 0.85 | 0.84 | 3812 |
| 1 | 0.73 | 0.69 | 0.71 | 2188 |
| accuracy |  |  | 0.79 | 6000 |
| macro avg | 0.78 | 0.77 | 0.77 | 6000 |
| weighted avg | 0.77 | 0.79 | 0.79 | 6000 |

#### **Accuracy:**

Accuracy is a measure of how well a model correctly predicts the overall instances or samples. It represents the ratio of the correctly predicted samples to the total number of samples in the dataset. The formula for accuracy is:

Accuracy = (Number of Correct Predictions) / (Total Number of Predictions)

Accuracy is typically expressed as a percentage, ranging from 0% to 100%. A higher accuracy indicates a better-performing model, as it correctly predicts a larger proportion of the samples.

#### **Precision:**

Precision is a measure of how well a model correctly predicts positive instances out of all the instances it predicts as positive. It represents the ratio of the correctly predicted positive samples to the total number of samples predicted as positive. The formula for precision is:

Precision = (Number of True Positives) / (Number of True Positives + Number of False Positives)

Precision is also commonly expressed as a percentage, ranging from 0% to 100%. A higher precision indicates a lower rate of false positives, meaning that the model is more selective in correctly identifying positive instances.

#### **F-Measure:**

#### The F-measure, also known as the F1 score, is a combined evaluation metric that balances both precision and recall (or sensitivity) of a model. It is particularly useful in scenarios where there is an imbalance between the classes or when both precision and recall need to be taken into account.

#### The F-measure is calculated using the harmonic mean of precision and recall. The formula for the F-measure is:

#### 𝐹−𝑚𝑒𝑎𝑠𝑢𝑟𝑒=2×(𝑃𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛×𝑅𝑒𝑐𝑎𝑙𝑙)(𝑃𝑟𝑒𝑐𝑖𝑠𝑖𝑜𝑛+𝑅𝑒𝑐𝑎𝑙𝑙)/(Precision+Recall)(Precision×Recall)​

#### ​

#### The F-measure ranges from 0 to 1, with 1 representing the best possible performance. A higher F-measure indicates a better trade-off between precision and recall.

#### **Recall:**

In the context of machine learning and evaluation metrics, recall (also known as sensitivity, hit rate, or true positive rate) is a measure of how well a model can identify positive instances from the total number of actual positive instances in a dataset. It quantifies the model's ability to correctly classify positive samples.

Recall is calculated using the following formula:

Recall = (Number of True Positives) / (Number of True Positives + Number of False Negatives)

In this formula, true positives (TP) represent the number of samples that are correctly classified as positive by the model, and false negatives (FN) represent the number of positive samples that are incorrectly classified as negative.

#### **Application 01**

#### **Inputs and Outputs:**

## 

Figure 15

#### **Application 02**

#### **Inputs and Outputs:**

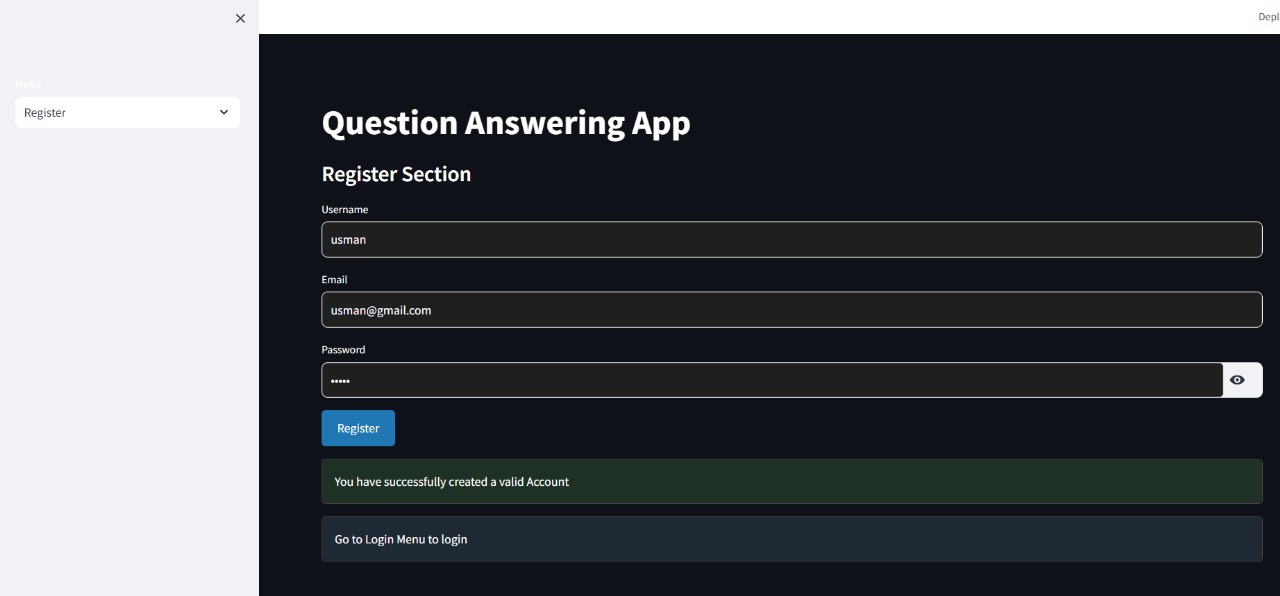


Figure 16

A screenshot of a computer

Description automatically generated

Figure 17

A screenshot of a computer

Description automatically generated

Figure 18

A screenshot of a computer

Description automatically generated

Figure 19

A screenshot of a question answer

Description automatically generated

Figure 20

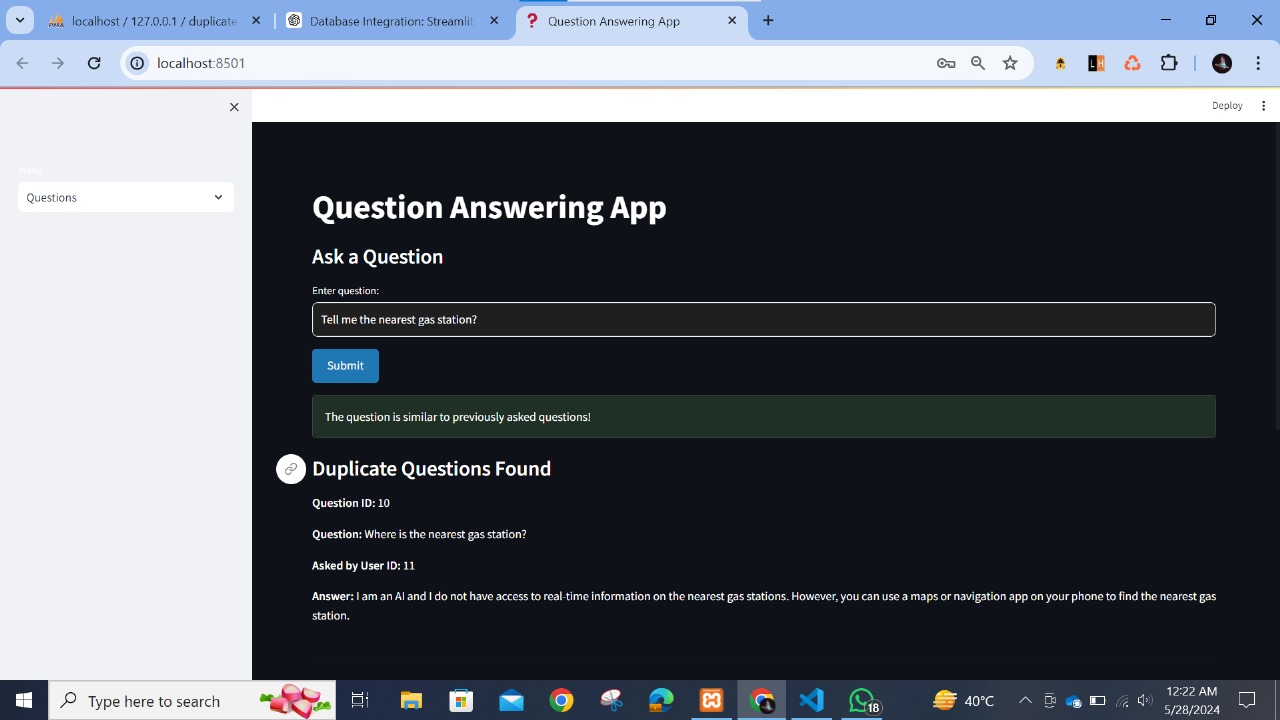


Figure 21

#### **Database View:**

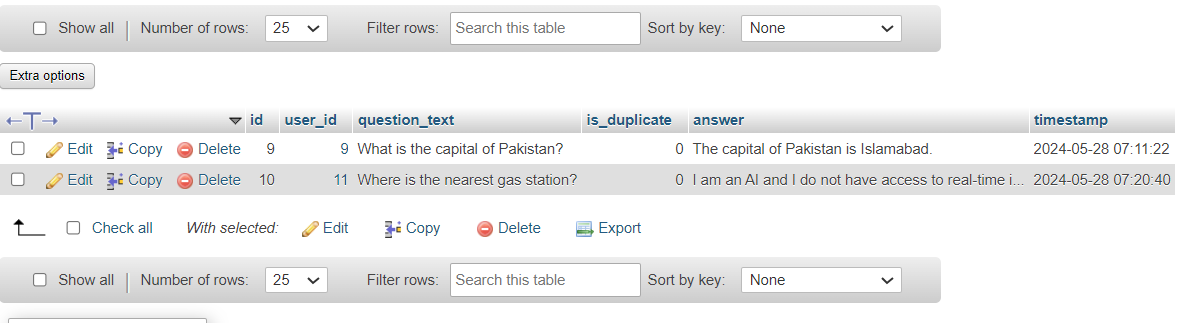


Figure 22

A screenshot of a computer

Description automatically generated

Figure 23

## **17. Example of similar Applications**

**Stack Overflow:**

* **Functionality:** Users can ask programming-related questions. If a similar question has already been asked, users are directed to the existing answer.
* **Technology:** Utilizes a large database of previously asked questions and answers, along with search and matching algorithms to detect duplicates.

**Quora:**

* **Functionality:** Users ask questions on a wide range of topics. If the question has already been asked, it is suggested to the user.
* **Technology:** Uses a combination of database searches and machine learning to identify similar questions and retrieve answers.

**Reddit:**

* **Functionality:** Users can post questions in various subreddits. Moderators or bots often link to previously asked questions if duplicates are found.
* **Technology:** Relies on community-driven moderation and search algorithms to manage duplicate content.

## **18. Conclusion**

This project demonstrates the effectiveness of machine learning algorithms in detecting duplicate questions, enhanced by advanced preprocessing techniques and integration with ChatGPT. By evaluating metrics such as accuracy, precision, and recall, we observed notable performance from both Random Forest and XGBoost models. While Random Forest exhibited slightly higher precision, XGBoost demonstrated marginally better recall. These findings underscore the importance of comprehensive metric evaluation in model assessment. Through meticulous preprocessing and integration with ChatGPT, this project showcases the potential synergy between machine learning and conversational AI, offering promising avenues for real-world application and advancement in duplicate question detection.