A Project Report

On

## **EFFICIENT DUPLICATE QUESTION DETECTION IN QUORA**

BY

**Hemanth Kakkireni (SE21UARI049)**

**Sai Shreekar Reddi Maligireddi (SE21UARI080)**

**Shashank Mutyala (SE21UARI087)**

**Srinivas Nalla (SE21UARI089)**

**Sai Varun Padmanabham (SE21UARI130)**

**SUBMITTED IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS OF**

**COURSE PROJECT IN AI 3106: Foundations of NLP**



**ECOLE CENTRALE SCHOOL OF ENGINEERING**

**HYDERABAD**

**(December 2023)**

# PROBLEM STATEMENT

### 1. Problem Definition

Quora is a platform for Question and Answer. One of the many problems that Quora face is the duplication of questions. Duplication of question ruins the experience for both the questioner and the answerer. Since the questioner is asking a duplicate question, we can just show him/her the answers to the previous question. And the answerer doesn’t have to repeat his/her answer for essentially the same questions.

For example, we have a question like “Who is the Prime Minister Of India?” and there are some answers to that question. Later someone else asks another question like “Who is the Current Prime Minister Of India?”. We can see that both the questions are asking the same thing. Even though the word "Current “for the question are different, the intention of both questions is the same. So the answers will be the same for both questions. That means we can just show the answers to the first question. That way the person who is asking the question will get the answers immediately and people who have answered already the first question don’t have to repeat themselves.

### 2. Existing Work

* **Duplicate Detection Algorithms**: Utilize NLP techniques, including tokenization and semantic similarity measures, to identify duplicate questions.
* **Content-Based Similarity Matching**: Compare question content, keywords, and context to determine similarity and detect duplicates.
* **Machine Learning Models**: Train models on labeled datasets, incorporating features like textual information and syntactic structures to predict duplicate questions.

### 3. Our Contribution

**Data Collection and Preprocessing of dataset :** Sai Varun Padmanabham

**Exploratory Data Analysis (EDA) :** Sai Shreekar Reddi Maligireddi

**Algorithm and Model Research :** Hemanth Kakkireni

**Algorithm Development and Coding :** Shashank Mutyala

**Integration and Testing :** Srinivas Nalla

**Documentation and Report :** Entire Team

## CONTENTS

Title page………………………………………………………….........……. 1

[Problem Definition 2](#_Toc65373)

[1. Implementation 4](#_Toc65374)

2. Results and Discussion ………………………………………………....... 12

References……………………………………………………………..........13

Appendix A…………………………………………………………….........13

Appendix B…………………………………………………………….........13

# 

# IMPLEMENTATION

**Data sets used:**

We have collected the dataset from both Google and Kaggle, merging them for a comprehensive dataset.

During preprocessing, you randomly sampled some data and applied the following transformations:

1. Replaced certain special characters with their string equivalents. For example, '%' was replaced with 'percent.'
2. Substituted numbers with their string equivalents.
3. Applied decontracting by replacing contracted words with their equivalent expanded forms. For instance, 'I've' was expanded to 'I have,' and 'wasn't' to 'was not.'

These preprocessing steps aim to enhance the dataset's consistency and readability for better analysis and model training.

**Environment:**

**Code Execution Platform:**

* Utilized **Google Colab, Pycharm** as the primary platform for running our code.

**Python Frameworks and Libraries:**

* Employed Python for scripting and integration.
* Utilized **Pandas** and **NumPy** for data manipulation and numerical operations.
* Leveraged **Matplotlib** and **Seaborn** for data visualization during exploratory analysis.
* Implemented machine learning models using **Scikit-learn**.
* Applied **NLTK** for natural language processing (NLP) tasks.
* Utilized **stopwords** for text processing.
* Employed **Fuzzywuzzy** for string similarity computations.
* Incorporated **Pickle** for model serialization.
* **Streamlit** was used for building an interactive user interface.

This comprehensive stack ensured an effective and collaborative approach to address the Quora duplicate question problem.

**Proposed Technique / Methodology:**

(describe your method to solve the problem – imagine describing your code to someone, include algorithm showcasing steps you took to solve the problem)

To know the duplicate questions we decided to take some features for identifying the repeated question they are:

### 1. Length of Each question in question column (i.e. q1 & q2)



This code creates two new columns, ***'q1\_len' and 'q2\_len'***, in the DataFrame ***'new\_df'***, which contain the character lengths of the respective strings in the 'question1' and 'question2' columns.

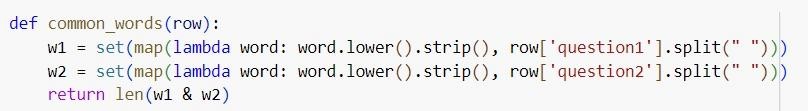
### 2. Number of words in each question



This code adds two new columns, ***'q1\_num\_words'*** and ***'q2\_num\_words'***, to the

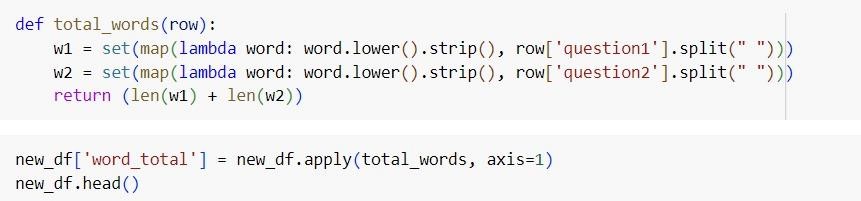
DataFrame 'new\_df', calculating the word count for each entry in the 'question1' and 'question2' columns by splitting the strings based on spaces and counting the resulting elements.

### 3. Identify if there are some common words



This code defines a ***function common\_words(row)*** that takes a row from the DataFrame ***new\_df***, splits the strings in the 'question1' and 'question2' columns into sets of lowercase words, strips whitespace, and calculates the count of common words between these sets. Subsequently, the function is applied to each row of the DataFrame using apply() along axis=1 to create a new column ***'word\_common'*** containing the count of common words between ***'question1' and 'question2'.***

### 4. Total words in q1&q2



The provided code defines a function ***total\_words***(row) that calculates the total count of distinct words in both ***'question1' and 'question2'*** columns within each row of the DataFrame new\_df. It creates a new column ***'word\_total'*** by summing up the counts of distinct words from both columns for every row using apply() along ***axis=1***. This new column stores the combined count of unique words from ***'question1***' and ***'question2'*** in each row of the DataFrame.

### 5. Word share = Common words/Total Words



This code snippet creates a new column named ***'word\_share'*** in the DataFrame ***'new\_df'***. It calculates the word share by dividing the count of common words (***'word\_common'***) by the total count of distinct words (***'word\_total'***) in each row.

The resulting values are rounded to two decimal places using the ***round() function***.

***Advanced Features :***

**cwc\_min**: This is the ratio of the number of common words to the length of the smaller question

**cwc\_max**: This is the ratio of the number of common words to the length of the larger question

**csc\_min**: This is the ratio of the number of common stop words to the smaller stop word count among the two questions

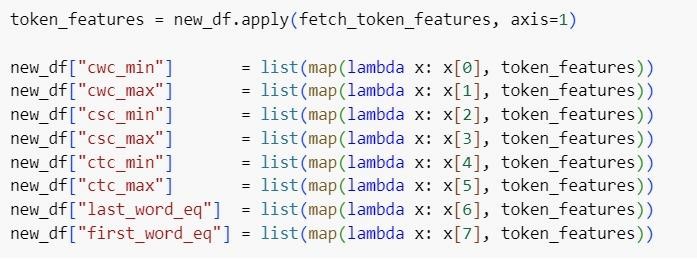
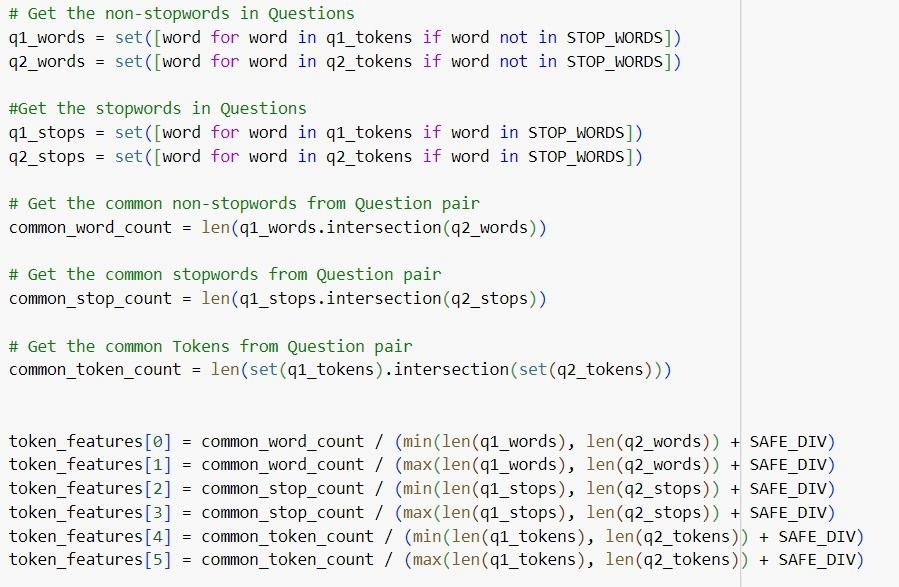
**csc\_max**: This is the ratio of the number of common stop words to the larger stop word count among the two questions

**ctc\_min**: This is the ratio of the number of common tokens to the smaller token count among the two questions

**ctc\_max**: This is the ratio of the number of common tokens to the larger token count among the two questions

**last\_word\_eq**: 1 if the last word in the two questions is same, 0 otherwise

**first\_word\_eq**: 1 if the first word in the two questions is same, 0 otherwise

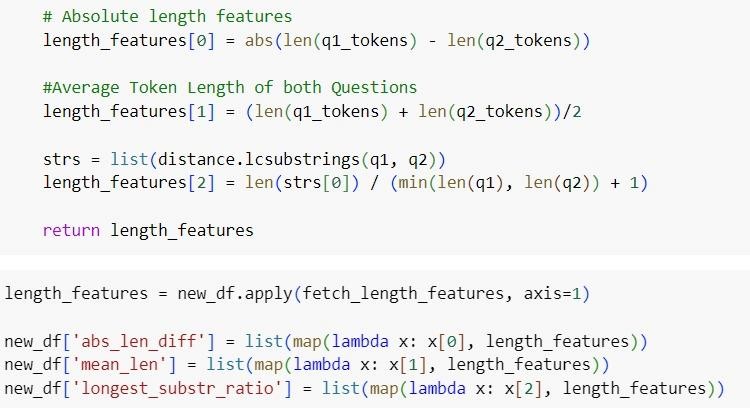


This code defines a function, ***fetch\_token\_features***(row), that computes various token-level features for comparing two text strings, 'question1' and 'question2', in a DataFrame ***'new\_df'***. These features include measures such as common word count, common stop word count, common token count, among others. These features are calculated for each row and stored in separate columns: ***'cwc\_min'***, ***'cwc\_max'***, ***'csc\_min', 'csc\_max', 'ctc\_min', 'ctc\_max', 'last\_word\_eq'***, and ***'first\_word\_eq'*** in the DataFrame ***'new\_df'***. These columns capture different aspects of similarity between the two questions based on their tokens and word properties. ***Length Based Features :***

**mean\_len**: Mean of the length of the two questions (number of words)

**abs\_len\_diff**: Absolute difference between the length of the two questions (number of words)

**longest\_substr\_ratio**: Ratio of the length of the longest substring among the two questions to the length of the smaller question



This code defines a function, ***fetch\_length\_features***, to compute length-related features such as absolute length difference, average token length, and the ratio of the length of the longest common substring to the minimum length of two strings. These features are calculated for each row of ***'question1' and 'question2'*** in the DataFrame ***'new\_df'*** and stored in respective columns ***'abs\_len\_diff', 'mean\_len'***, and ***'longest\_substr\_ratio'.***

***Fuzzy Features:***

**fuzz\_ratio**: fuzz\_ratio score from fuzzywuzzy

**fuzz\_partial\_ratio**: fuzz\_partial\_ratio from fuzzywuzzy

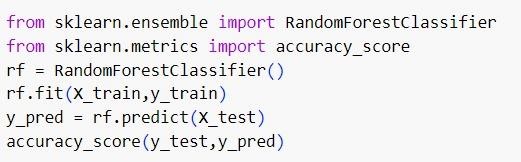
**token\_sort\_ratio**: token\_sort\_ratio from fuzzywuzzy

**token\_set\_ratio**: token\_set\_ratio from fuzzywuzzy

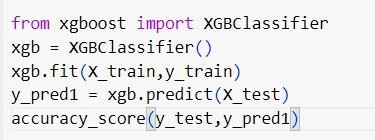




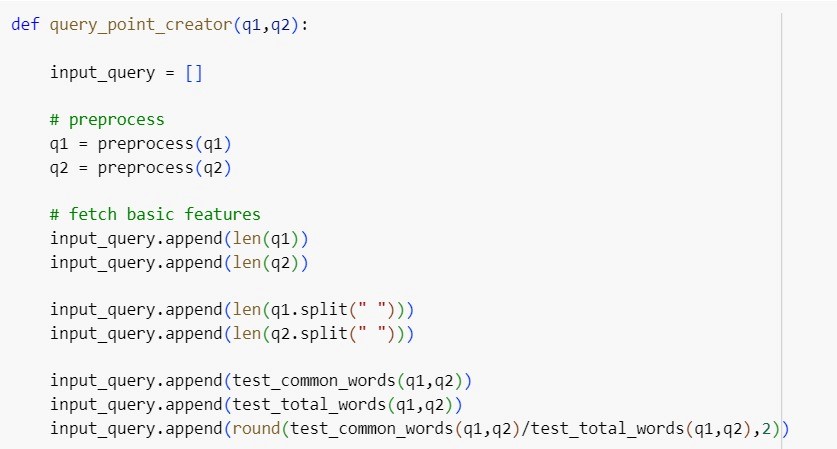
This code uses the ***fuzzywuzzy*** library to compute fuzzy matching features between ***'question1' and 'question2'*** in the DataFrame ***'new\_df'***. It calculates various similarity ratios such as ***'fuzz\_ratio', 'fuzz\_partial\_ratio', 'token\_sort\_ratio'***, and ***'token\_set\_ratio'*** based on different fuzzy matching algorithms. These features are computed for each row and stored in respective columns in the DataFrame ***'new\_df'***. They provide measures of similarity between the two questions based on their string matching characteristics.



The code uses ***scikit-learn's*** ***Random Forest Classifier*** to train a model ***(rf)*** with ***X\_train and y\_train data,*** make predictions on ***X\_test***, and then calculates the accuracy of these predictions using ***accuracy\_score(y\_test, y\_pred).***



The code uses ***scikit-learn's XGBoost Classifier*** to train a model **(xgb)** with ***X\_train*** *and* ***y\_train*** data, make predictions on ***X\_test***, and then calculates the accuracy of these predictions using ***accuracy\_score(y\_test, y\_pred).***



The function query\_point\_creator takes in *two input strings*, ***q1*** and ***q2***, preprocesses them, and generates a ***feature vector*** to represent these queries for a machine learning model.

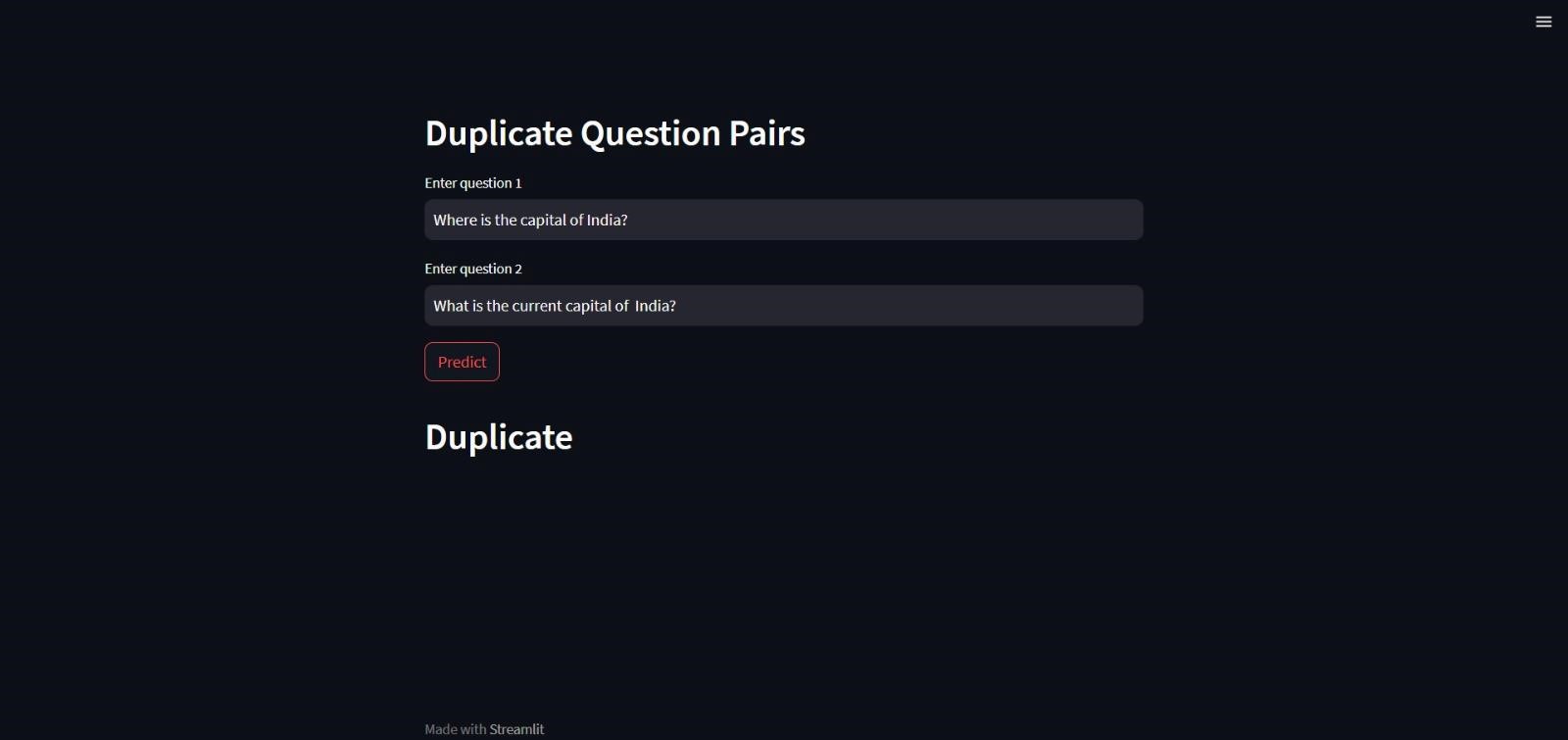
***Explanation:***

* The function preprocesses the ***input strings q1 and q2.***
* Basic features such as lengths of q1 and q2, word counts, and ratios of common to total words are calculated and appended to ***input\_query***.
* ***Token-based features, length-based features***, and **fuzzy matching features** are computed and added to input\_query.
* ***Bag-of-Words (BoW)*** representations for ***q1*** and ***q2*** are generated using a ***CountVectorizer (cv),*** and these ***BoW*** representations are concatenated with the feature vector.
* The function returns a combined feature vector that includes ***basic, tokenbased, length-based, fuzzy matching features***, and ***BoW*** representations for ***both input queries q1 and q2.***

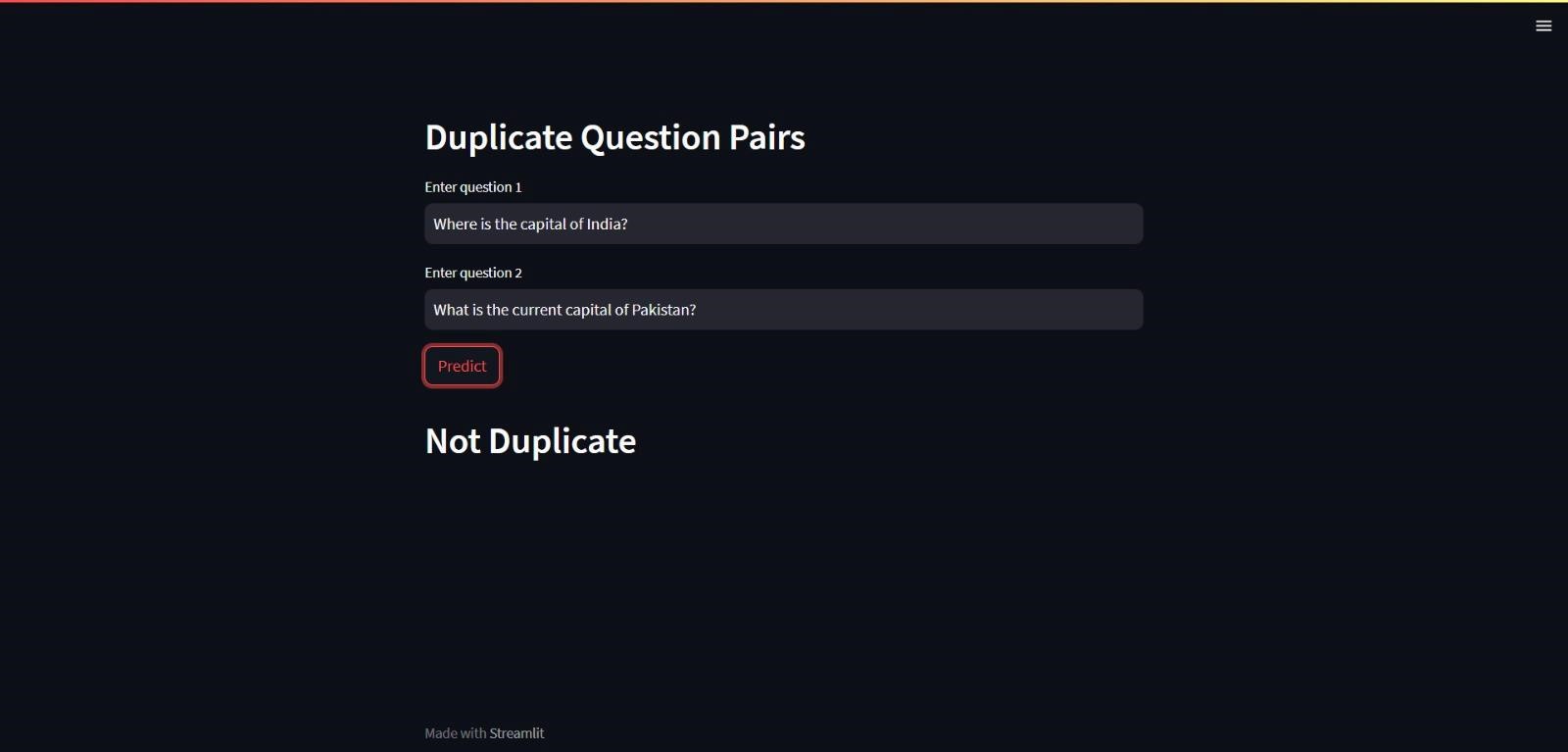
**RESULTS AND DISCUSSION**

We utilized a function called 'pickle' to save two files named ***'model.pkl'*** and ***'cv.pkl'***. The ***'model.pkl'*** file contains a trained machine learning model, and ***'cv.pkl'*** stores information related to the text processing technique used.

We integrated these files into a ***Streamlit application*** in ***PyCharm***, to create a website. This website utilizes the trained model to predict and provide insights or evaluations based on user-provided text inputs. Users can interact with this website to input text queries, and the underlying machine learning model processes these queries to generate results or predictions.



This above question was ***Duplicate***



This above question was ***Not Duplicate***

**REFERENCES**

## **APPENDIX A**

**URL TO COLAB PYTHON SCRIPT:**

<https://colab.research.google.com/drive/1aJgIpuhcc9JHNnNQ7jOhlFplIw0Tqp64?usp=sharing>

**LINK TO GITHUB REPO:**

<https://github.com/shashankmutyala/Duplicate_Quora_Questions_Dectection.git>

## **RESOURCE:**

***DATASET:***

<https://mahindraecolecentrale-my.sharepoint.com/:x:/g/personal/se21uari087_mahindrauniversity_edu_in/Ee9qDvzewBROrCo39WTn64wB90Bk1gNWHhZc6CZpfJ4E0g?e=8vP9R2>

## **APPENDIX B:**

***SEE BELOW:***

|  |  |
| --- | --- |
| import numpy as np import pandas as pd import matplotlib.pyplot as plt import seaborn as sns  import re  from bs4 import BeautifulSoup  import warnings warnings.filterwarnings('ignore') |  |

df = pd.read\_csv

(

'train.csv'

)

new\_df = df.sample

(

30000

,

random\_state=

2

)

**id**

**qid1**

**qid2**

**question1**

**question2**

**is\_duplicate**

new\_df.head

()

**398782** 398782 496695 532029 What is the best marketing automation tool for... What is the best marketing automation tool for... 1

**115086** 115086 187729 187730 I am poor but I want to invest. What should I do? I am quite poor and I want to be very rich. Wh... 0

**327711** 327711 454161 454162 I am from India and live abroad. I met a guy f... T.I.E.T to Thapar University to Thapar Univers... 0

**367788** 367788 498109 491396 Why do so many people in the U.S. hate the sou... My boyfriend doesnt feel guilty when he hurts ... 0

**151235** 151235 237843 50930 Consequences of Bhopal gas tragedy? What was the reason behind the Bhopal gas trag... 0

|  |
| --- |
| def preprocess(q):  q = str(q).lower().strip()  # Replace certain special characters with their string equivalents q = q.replace('%', ' percent') q = q.replace('$', ' dollar ') q = q.replace('₹', ' rupee ') q = q.replace('€', ' euro ') q = q.replace('@', ' at ') q = q.replace('[math]', '')  # Replacing some numbers with string equivalents (not perfect, can be done better to account for more cases) q = q.replace(',000,000,000 ', 'b ') q = q.replace(',000,000 ', '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)  # Decontracting words  # https://en.wikipedia.org/wiki/Wikipedia%3aList\_of\_English\_contractions  # https://stackoverflow.com/a/19794953 contractions = { "ain't": "am not",  "aren't": "are not",  "can't": "can not",  "can't've": "can not have",  "'cause": "because",  "could've": "could have",  "couldn't": "could not",  "couldn't've": "could not have",  "didn't": "did not",  "doesn't": "does not",  "don't": "do not",  "hadn't": "had not",  "hadn't've": "had not have",  "hasn't": "has not",  "haven't": "have not",  "he'd": "he would",  "he'd've": "he would have", "he'll": "he will",  "he'll've": "he will have",  "he's": "he is",  "how'd": "how did",  "how'd'y": "how do you",  "how'll": "how will",  "how's": "how is",  "i'd": "i would",  "i'd've": "i would have", "i'll": "i will",  "i'll've": "i will have",  "i'm": "i am",  "i've": "i have",  "isn't": "is not",  "it'd": "it would",  "it'd've": "it would have", "it'll": "it will",  "it'll've": "it will have",  "it's": "it is",  "let's": "let us",  "ma'am": "madam",  "mayn't": "may not",  "might've": "might have",  "mightn't": "might not",  "mightn't've": "might not have",  "must've": "must have",  "mustn't": "must not",  "mustn't've": "must not have", "needn't": "need not",  "needn't've": "need not have",  "o'clock": "of the clock",  "oughtn't": "ought not",  "oughtn't've": "ought not have",  "shan't": "shall not",  "sha'n't": "shall not",  "shan't've": "shall not have",  "she'd": "she would",  "she'd've": "she would have", "she'll": "she will",  "she'll've": "she will have",  "she's": "she is",  "should've": "should have",  "shouldn't": "should not",  "shouldn't've": "should not have",  "so've": "so have",  "so's": "so as",  "that'd": "that would",  "that'd've": "that would have",  "that's": "that is",  "there'd": "there would",  "there'd've": "there would have",  "there's": "there is",  "they'd": "they would",  "they'd've": "they would have", "they'll": "they will",  "they'll've": "they will have",  "they're": "they are",  "they've": "they have",  "to've": "to have",  "wasn't": "was not",  "we'd": "we would",  "we'd've": "we would have", "we'll": "we will",  "we'll've": "we will have",  "we're": "we are",  "we've": "we have",  "weren't": "were not",  "what'll": "what will",  "what'll've": "what will have",  "what're": "what are",  "what's": "what is",  "what've": "what have", "when's": "when is",  "when've": "when have",  "where'd": "where did",  "where's": "where is",  "where've": "where have",  "who'll": "who will",  "who'll've": "who will have",  "who's": "who is",  "who've": "who have", "why's": "why is",  "why've": "why have",  "will've": "will have",  "won't": "will not",  "won't've": "will not have",  "would've": "would have",  "wouldn't": "would not",  "wouldn't've": "would not have",  "y'all": "you all",  "y'all'd": "you all would",  "y'all'd've": "you all would have",  "y'all're": "you all are",  "y'all've": "you all have",  "you'd": "you would",  "you'd've": "you would have", "you'll": "you will",  "you'll've": "you will have",  "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")  # Removing HTML tags q = BeautifulSoup(q) q = q.get\_text()  # Remove punctuations pattern = re.compile('\W') q = re.sub(pattern, ' ', q).strip()  return q |

preprocess

(

"I've already! wasn't <b>done</b>?"

)

'i have already was not done'

new\_df['question1'] = new\_df['question1'].apply(preprocess)

new\_df

[

'question2'

]

= new\_df

[

'question2'

]

.apply

(

preprocess

)

**id**

**qid1**

**qid2**

**question1**

**question2**

**is\_duplicate**

new\_df.head

()

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **398782** | 398782 | 496695 | 532029 what is the best marketing automation tool for... | what is the best marketing automation tool for... | 1 |
| **115086** | 115086 | 187729 | 187730 i am poor but i want to invest what should i do | i am quite poor and i want to be very rich wh... | 0 |
| **327711** | 327711 | 454161 | 454162 i am from india and live abroad i met a guy f... | t i e t to thapar university to thapar univers... | 0 |
| **367788** | 367788 | 498109 | 491396 why do so many people in the u s hate the sou... | my boyfriend doesnt feel guilty when he hurts ... | 0 |
| **151235** | 151235 | 237843 | 50930 consequences of bhopal gas tragedy | what was the reason behind the bhopal gas tragedy | 0 |
| new\_df['q1\_len'] = new\_df['question1'].str.len() new\_df['q2\_len'] = new\_df['question2'].str.len() | | | |  |  |

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(" "))) new\_df.head()

**id qid1 qid2 question1 question2 is\_duplicate q1\_len q2\_len q1\_num\_words q2\_num\_words**

**398782** 398782 496695 532029 what is the best marketing automation tool for... what is the best marketing automation tool for... 1 75 76 13 13

**115086** 115086 187729 187730 i am poor but i want to invest what should i do i am quite poor and i want to be very rich wh... 0 48 56 13 16

**327711** 327711 454161 454162 i am from india and live abroad i met a guy f... t i e t to thapar university to thapar univers... 0 104 119 28 21

**367788** 367788 498109 491396 why do so many people in the u s hate the sou... my boyfriend doesnt feel guilty when he hurts ... 0 58 145 14 32

**151235** 151235 237843 50930 consequences of bhopal gas tragedy what was the reason behind the bhopal gas tragedy 0 34 49 5 9

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)

new\_df['word\_common'] = new\_df.apply(common\_words, axis=1) new\_df.head()

**id qid1 qid2 question1 question2 is\_duplicate q1\_len q2\_len q1\_num\_words q2\_num\_words word\_common**

**398782** 398782 496695 532029 what is the best marketing automation tool for... what is the best marketing automation tool for... 1 75 76 13 13 12

**115086** 115086 187729 187730 i am poor but i want to invest what should i do i am quite poor and i want to be very rich wh... 0 48 56 13 16 8

**327711** 327711 454161 454162 i am from india and live abroad i met a guy f... t i e t to thapar university to thapar univers... 0 104 119 28 21 4

**367788** 367788 498109 491396 why do so many people in the u s hate the sou... my boyfriend doesnt feel guilty when he hurts ... 0 58 145 14 32 1

**151235** 151235 237843 50930 consequences of bhopal gas tragedy what was the reason behind the bhopal gas tragedy 0 34 49 5 9 3

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))

new\_df['word\_total'] = new\_df.apply(total\_words, axis=1) new\_df.head()

**id qid1 qid2 question1 question2 is\_duplicate q1\_len q2\_len q1\_num\_words q2\_num\_words word\_common word\_total**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **398782** | 398782 | 496695 | 532029 what is the best marketing automation tool for... | what is the best marketing automation tool for... | 1 | 75 | 76 | 13 | 13 | 12 | 26 |
| **115086** | 115086 | 187729 | 187730 i am poor but i want to invest what should i do | i am quite poor and i want to be very rich wh... | 0 | 48 | 56 | 13 | 16 | 8 | 24 |
| **327711** | 327711 | 454161 | 454162 i am from india and live abroad i met a guy f... | t i e t to thapar university to thapar univers... | 0 | 104 | 119 | 28 | 21 | 4 | 38 |
| **367788** | 367788 | 498109 | 491396 why do so many people in the u s hate the sou... | my boyfriend doesnt feel guilty when he hurts ... | 0 | 58 | 145 | 14 | 32 | 1 | 34 |
| **151235** | 151235 | 237843 | 50930 consequences of bhopal gas tragedy | what was the reason behind the bhopal gas tragedy | 0 | 34 | 49 | 5 | 9 | 3 | 13 |
| new\_df['word\_share'] = round(new\_df['word\_common']/new\_df['word\_total'],2) new\_df.head() | | | |  |  |  |  |  |  |  |  |

**id qid1 qid2 question1 question2 is\_duplicate q1\_len q2\_len q1\_num\_words q2\_num\_words word\_common word\_total word\_share**

**398782** 398782 496695 532029 what is the best marketing automation tool for... what is the best marketing automation tool for... 1 75 76 13 13 12 26 0.46

**115086** 115086 187729 187730 i am poor but i want to invest what should i do i am quite poor and i want to be very rich wh... 0 48 56 13 16 8 24 0.33 **327711** 327711 454161 454162 i am from india and live abroad i met a guy f... t i e t to thapar university to thapar univers... 0 104 119 28 21 4 38 0.11

**367788** 367788 498109 491396 why do so many people in the u s hate the sou... my boyfriend doesnt feel guilty when he hurts ... 0 58 145 14 32 1 34 0.03

**151235** 151235 237843 50930 consequences of bhopal gas tragedy what was the reason behind the bhopal gas tragedy 0 34 49 5 9 3 13 0.23

All Advanced Features are focused on tokens, non-stopwords & stopwords

Advanced Features cwc\_min: This is the ratio of the number of common words to the length of the smaller question cwc\_max: This is the ratio of the number of common words to the length of the larger question csc\_min: This is the ratio of the number of common stop words to the smaller stop word count among the two questions csc\_max: This is the ratio of the number of common stop words to the larger stop word count among the two questions ctc\_min: This is the ratio of the number of common tokens to the smaller token count among the two questions ctc\_max: This is the ratio of the number of common tokens to the larger token count among the two questions last\_word\_eq: 1 if the last word in the two questions is same, 0 otherwise rst\_word\_eq: 1 if the rst word in the two questions is same, 0 otherwise

import

nltk

|  |
| --- |
| from nltk.corpus import stopwords def fetch\_token\_features(row):  q1 = row['question1'] q2 = row['question2']  SAFE\_DIV = 0.0001 # Use for denominator not to become zero  STOP\_WORDS = stopwords.words("english") # token\_features initially given by 8 'zeros' since 8 features token\_features = [0.0]\*8  # Converting the Sentence into Tokens:  q1\_tokens = q1.split() q2\_tokens = q2.split()  if len(q1\_tokens) == 0 or len(q2\_tokens) == 0: return token\_features  # Get the non-stopwords in Questions  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])  #Get the stopwords in Questions 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])  # Get the common non-stopwords from Question pair common\_word\_count = len(q1\_words.intersection(q2\_words))  # Get the common stopwords from Question pair common\_stop\_count = len(q1\_stops.intersection(q2\_stops))  # Get the common Tokens from Question pair  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)  # Last word of both question is same or not  token\_features[6] = int(q1\_tokens[-1] == q2\_tokens[-1])  # First word of both question is same or not token\_features[7] = int(q1\_tokens[0] == q2\_tokens[0]) return token\_features |

nltk.download('stopwords')

[nltk\_data] Downloading package stopwords to /root/nltk\_data...

[nltk\_data] Unzipping corpora/stopwords.zip.

True

token\_features = new\_df.apply(fetch\_token\_features, axis=1)

new\_df["cwc\_min"] = list(map(lambda x: x[0], token\_features)) new\_df["cwc\_max"] = list(map(lambda x: x[1], token\_features)) new\_df["csc\_min"] = list(map(lambda x: x[2], token\_features)) new\_df["csc\_max"] = list(map(lambda x: x[3], token\_features)) new\_df["ctc\_min"] = list(map(lambda x: x[4], token\_features)) new\_df["ctc\_max"] = list(map(lambda x: x[5], token\_features)) new\_df["last\_word\_eq"] = list(map(lambda x: x[6], token\_features))

new\_df

[

"first\_word\_eq"

]

=

list

(

map

(

lambda

x

:

x

[

7

,

]

token\_features

))

**id**

**qid1**

**qid2**

**question1**

**question2**

**is\_duplicate**

**q1\_len**

**q2\_len**

**q1\_num\_words**

**q2\_num\_words**

**...**

**word\_total**

**word\_share**

**cwc\_min**

**cwc\_max**

**csc\_min**

**csc\_max**

**ctc\_min**

**ctc\_max**

**last\_word\_eq**

**first\_word\_eq**

new\_df.head

()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **398782** | 398782 | 496695 | 532029 | what is the best marketing automation tool for... | what is the best marketing automation tool for... | 1 | 75 | 76 | 13 | 13 | ... | 26 | 0.46 | 0.874989 | 0.874989 | 0.999980 | 0.999980 | 0.923070 | 0.923070 | 1.0 | 1.0 |
| **115086** | 115086 | 187729 | 187730 | i am poor but i want to invest what should i do | i am quite poor and i want to be very rich wh... | 0 | 48 | 56 | 13 | 16 | ... | 24 | 0.33 | 0.666644 | 0.499988 | 0.714276 | 0.624992 | 0.583328 | 0.466664 | 1.0 | 1.0 |
| **327711** | 327711 | 454161 | 454162 | i am from india and live abroad i met a guy f... | t i e t to thapar university to thapar univers... | 0 | 104 | 119 | 28 | 21 | ... | 38 | 0.11 | 0.000000 | 0.000000 | 0.428565 | 0.272725 | 0.149999 | 0.115384 | 0.0 | 0.0 |
| **367788** | 367788 | 498109 | 491396 | why do so many people in the u s hate the sou... | my boyfriend doesnt feel guilty when he hurts ... | 0 | 58 | 145 | 14 | 32 | ... | 34 | 0.03 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.0 | 0.0 |
| **151235** | 151235 | 237843 | 50930 | consequences of bhopal gas tragedy | what was the reason behind the bhopal gas tragedy | 0 | 34 | 49 | 5 | 9 | ... | 13 | 0.23 | 0.749981 | 0.599988 | 0.000000 | 0.000000 | 0.599988 | 0.333330 | 1.0 | 0.0 |
| 5 rows × 21 columns  Length Based Features | | | mean\_len: Mean of the length of the two questions (number of words) | abs\_len\_diff: Absolute difference between the length of the two questions (number of words) longest\_substr\_ratio: Ratio of the length of the longest substring among the two questions to the length of the smaller question |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| pip install Distance | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Collecting Distance  Downloading Distance-0.1.3.tar.gz (180 kB)  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 180.3/180.3 kB 3.3 MB/s eta 0:00:00  Preparing metadata (setup.py) ... done  Building wheels for collected packages: Distance  Building wheel for Distance (setup.py) ... done  Created wheel for Distance: filename=Distance-0.1.3-py3-none-any.whl size=16258 sha256=7f3e6a219b2554e28518f10f038f91cc42522cae6cf9c3424f4dcec0da7ee75d  Stored in directory: /root/.cache/pip/wheels/e8/bb/de/f71bf63559ea9a921059a5405806f7ff6ed612a9231c4a9309  Successfully built Distance  Installing collected packages: Distance  Successfully installed Distance-0.1.3 | | | | | | | | |  |  |  |  |  |  |  |  |  |  |  |  |  |
| import distance def fetch\_length\_features(row):  q1 = row['question1'] q2 = row['question2'] length\_features = [0.0]\*3  # Converting the Sentence into Tokens:  q1\_tokens = q1.split() q2\_tokens = q2.split()  if len(q1\_tokens) == 0 or len(q2\_tokens) == 0: return length\_features  # Absolute length features  length\_features[0] = abs(len(q1\_tokens) - len(q2\_tokens))  #Average Token Length of both Questions  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 | | | | | | | | | | | | | | | | | | | | | |

length\_features = new\_df.apply(fetch\_length\_features, axis=1)

new\_df['abs\_len\_diff'] = list(map(lambda x: x[0], length\_features)) new\_df['mean\_len'] = list(map(lambda x: x[1], length\_features))

new\_df

[

'longest\_substr\_ratio'

]

=

list

(

map

(

lambda

x

:

x

[

2

,

]

length\_features

))

**id**

**qid1**

**qid2**

**question1**

**question2**

**is\_duplicate**

**q1\_len**

**q2\_len**

**q1\_num\_words**

**q2\_num\_words**

**...**

**cwc\_max**

**csc\_min**

**csc\_max**

**ctc\_min**

**ctc\_max**

**last\_word\_eq**

**first\_word\_eq**

**abs\_len\_diff**

**mean\_len**

**longest\_substr\_ratio**

**398782**

398782

496695

532029

what is the best marketing automation tool for...

what is the best marketing automation tool for...

1

75

76

13

13

...

0.874989

0.999980

0.999980

0.923070

0.923070

1.0

1.0

0.0

13.0

0.855263

new\_df.head

()

**115086** 115086 187729 187730 i am poor but i want to invest what should i do i am quite poor and i want to be very rich wh... 0 48 56 13 16 ... 0.499988 0.714276 0.624992 0.583328 0.466664 1.0 1.0 3.0 13.5 0.224490

**327711** 327711 454161 454162 i am from india and live abroad i met a guy f... t i e t to thapar university to thapar univers... 0 104 119 28 21 ... 0.000000 0.428565 0.272725 0.149999 0.115384 0.0 0.0 6.0 23.0 0.047619

**367788** 367788 498109 491396 why do so many people in the u s hate the sou... my boyfriend doesnt feel guilty when he hurts ... 0 58 145 14 32 ... 0.000000 0.000000 0.000000 0.000000 0.000000 0.0 0.0 17.0 21.5 0.050847

**151235** 151235 237843 50930 consequences of bhopal gas tragedy what was the reason behind the bhopal gas tragedy 0 34 49 5 9 ... 0.599988 0.000000 0.000000 0.599988 0.333330 1.0 0.0 4.0 7.0 0.542857

5 rows × 24 columns

Fuzzy Features fuzz\_ratio: fuzz\_ratio score from fuzzywuzzy fuzz\_partial\_ratio: fuzz\_partial\_ratio from fuzzywuzzy token\_sort\_ratio: token\_sort\_ratio from fuzzywuzzy token\_set\_ratio: token\_set\_ratio from fuzzywuzzy

|  |  |
| --- | --- |
| pip install fuzzywuzzy |  |

Collecting fuzzywuzzy

Downloading fuzzywuzzy-0.18.0-py2.py3-none-any.whl (18 kB)

Installing collected packages: fuzzywuzzy

Successfully installed fuzzywuzzy-0.18.0

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

fuzzy\_features = new\_df.apply(fetch\_fuzzy\_features, axis=1)

# Creating new feature columns for fuzzy features

new\_df['fuzz\_ratio'] = list(map(lambda x: x[0], fuzzy\_features)) new\_df['fuzz\_partial\_ratio'] = list(map(lambda x: x[1], fuzzy\_features)) new\_df['token\_sort\_ratio'] = list(map(lambda x: x[2], fuzzy\_features))

new\_df

[

'token\_set\_ratio'

]

=

list

(

map

(

lambda

x

:

x

[

3

,

]

fuzzy\_features

))

print

(

new\_df.shape

)

(30000, 28)

**id**

**qid1**

**qid2**

**question1**

**question2**

**is\_duplicate**

**q1\_len**

**q2\_len**

**q1\_num\_words**

**q2\_num\_words**

**...**

**ctc\_max**

**last\_word\_eq**

**first\_word\_eq**

**abs\_len\_diff**

**mean\_len**

**longest\_substr\_ratio**

**fuzz\_ratio**

**fuzz\_partial\_ratio**

**token\_sort\_ratio**

**token\_set\_ratio**

**398782**

398782

496695

532029

what is the best marketing automation tool for...

what is the best marketing automation tool for...

1

75

76

13

13

...

0.923070

1.0

1.0

0.0

13.0

0.855263

99

99

99

99

new\_df.head

()

**115086** 115086 187729 187730 i am poor but i want to invest what should i do i am quite poor and i want to be very rich wh... 0 48 56 13 16 ... 0.466664 1.0 1.0 3.0 13.5 0.224490 69 67 65 74

**327711** 327711 454161 454162 i am from india and live abroad i met a guy f... t i e t to thapar university to thapar univers... 0 104 119 28 21 ... 0.115384 0.0 0.0 6.0 23.0 0.047619 26 29 34 43

**367788** 367788 498109 491396 why do so many people in the u s hate the sou... my boyfriend doesnt feel guilty when he hurts ... 0 58 145 14 32 ... 0.000000 0.0 0.0 17.0 21.5 0.050847 29 41 23 30

**151235** 151235 237843 50930 consequences of bhopal gas tragedy what was the reason behind the bhopal gas tragedy 0 34 49 5 9 ... 0.333330 1.0 0.0 4.0 7.0 0.542857 55 70 48 69

5 rows × 28 columns

ques\_df = new\_df[['question1','question2']] ques\_df.head()

|  |  |
| --- | --- |
| **question1** | **question2** |

|  |  |  |
| --- | --- | --- |
| **398782** | what is the best marketing automation tool for... | what is the best marketing automation tool for... |
| **115086** | i am poor but i want to invest what should i do | i am quite poor and i want to be very rich wh... |
| **327711** | i am from india and live abroad i met a guy f... | t i e t to thapar university to thapar univers... |
| **367788** | why do so many people in the u s hate the sou... | my boyfriend doesnt feel guilty when he hurts ... |
| **151235** | consequences of bhopal gas tragedy | what was the reason behind the bhopal gas tragedy |

final\_df = new\_df.drop(columns=['id','qid1','qid2','question1','question2']) print(final\_df.shape) final\_df.head()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (30000, 23) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **is\_duplicate** | **q1\_len** | **q2\_len** | **q1\_num\_words** | **q2\_num\_words** | **word\_common** | **word\_total** | **word\_share** | **cwc\_min** | **cwc\_max** | **...** | **ctc\_max** | **last\_word\_eq** | **first\_word\_eq** | **abs\_len\_diff** | **mean\_len** | **longest\_substr\_ratio** | **fuzz\_ratio** | **fuzz\_partial\_ratio** | **token\_sort\_ratio** | **token\_set\_ratio** |

**398782** 1 75 76 13 13 12 26 0.46 0.874989 0.874989 ... 0.923070 1.0 1.0 0.0 13.0 0.855263 99 99 99 99

**115086** 0 48 56 13 16 8 24 0.33 0.666644 0.499988 ... 0.466664 1.0 1.0 3.0 13.5 0.224490 69 67 65 74

**327711** 0 104 119 28 21 4 38 0.11 0.000000 0.000000 ... 0.115384 0.0 0.0 6.0 23.0 0.047619 26 29 34 43

**367788** 0 58 145 14 32 1 34 0.03 0.000000 0.000000 ... 0.000000 0.0 0.0 17.0 21.5 0.050847 29 41 23 30

**151235** 0 34 49 5 9 3 13 0.23 0.749981 0.599988 ... 0.333330 1.0 0.0 4.0 7.0 0.542857 55 70 48 69

5 rows × 23 columns

from sklearn.feature\_extraction.text import CountVectorizer

# merge texts questions = list(ques\_df['question1']) + list(ques\_df['question2'])

cv = CountVectorizer(max\_features=3000)

q1\_arr, q2\_arr = np.vsplit(cv.fit\_transform(questions).toarray(),2)

temp\_df1 = pd.DataFrame(q1\_arr, index= ques\_df.index) temp\_df2 = pd.DataFrame(q2\_arr, index= ques\_df.index) temp\_df = pd.concat([temp\_df1, temp\_df2], axis=1) temp\_df.shape

(30000, 6000)

final\_df = pd.concat([final\_df, temp\_df], axis=1) print(final\_df.shape) final\_df.head()

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (30000, 6023) |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| **is\_duplicate** | **q1\_len** | **q2\_len** | **q1\_num\_words** | **q2\_num\_words** | **word\_common** | **word\_total** | **word\_share** | **cwc\_min** | **cwc\_max** | **...** | **2990** | **2991** | **2992** | **2993** | **2994** | **2995** | **2996** | **2997** | **2998** | **2999** |

**398782** 1 75 76 13 13 12 26 0.46 0.874989 0.874989 ... 0 0 0 0 0 0 0 0 0 0

**115086** 0 48 56 13 16 8 24 0.33 0.666644 0.499988 ... 0 0 0 0 0 0 0 0 0 0

**327711** 0 104 119 28 21 4 38 0.11 0.000000 0.000000 ... 0 0 0 0 0 0 0 0 0 0

**367788** 0 58 145 14 32 1 34 0.03 0.000000 0.000000 ... 0 0 0 1 0 0 0 0 0 0

**151235** 0 34 49 5 9 3 13 0.23 0.749981 0.599988 ... 0 0 0 0 0 0 0 0 0 0

5 rows × 6023 columns

from sklearn.model\_selection import train\_test\_split

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)

from sklearn.ensemble import RandomForestClassifier from sklearn.metrics import accuracy\_score rf = RandomForestClassifier() rf.fit(X\_train,y\_train) y\_pred = rf.predict(X\_test) accuracy\_score(y\_test,y\_pred)

0.7905

from xgboost import XGBClassifier xgb = XGBClassifier() xgb.fit(X\_train,y\_train) y\_pred1 = xgb.predict(X\_test) accuracy\_score(y\_test,y\_pred1)

0.7946666666666666

from sklearn.metrics import confusion\_matrix # for random forest model confusion\_matrix(y\_test,y\_pred)

array([[3299, 513], [ 744, 1444]])

# for xgboost model confusion\_matrix(y\_test,y\_pred1)

array([[3237, 575],

[ 657, 1531]])

def test\_common\_words(q1,q2):

w1 = set(map(lambda word: word.lower().strip(), q1.split(" "))) w2 = set(map(lambda word: word.lower().strip(), q2.split(" "))) return len(w1 & w2)

def test\_total\_words(q1,q2):

w1 = set(map(lambda word: word.lower().strip(), q1.split(" "))) w2 = set(map(lambda word: word.lower().strip(), q2.split(" "))) return (len(w1) + len(w2))

|  |
| --- |
| def test\_fetch\_token\_features(q1,q2):  SAFE\_DIV = 0.0001  STOP\_WORDS = stopwords.words("english") token\_features = [0.0]\*8  # Converting the Sentence into Tokens:  q1\_tokens = q1.split() q2\_tokens = q2.split()  if len(q1\_tokens) == 0 or len(q2\_tokens) == 0: return token\_features  # Get the non-stopwords in Questions  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])  #Get the stopwords in Questions 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])  # Get the common non-stopwords from Question pair common\_word\_count = len(q1\_words.intersection(q2\_words))  # Get the common stopwords from Question pair  common\_stop\_count = len(q1\_stops.intersection(q2\_stops))  # Get the common Tokens from Question pair  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)  # Last word of both question is same or not token\_features[6] = int(q1\_tokens[-1] == q2\_tokens[-1])  # First word of both question is same or not token\_features[7] = int(q1\_tokens[0] == q2\_tokens[0]) return token\_features |
| def test\_fetch\_length\_features(q1,q2):  length\_features = [0.0]\*3  # Converting the Sentence into Tokens:  q1\_tokens = q1.split() q2\_tokens = q2.split()  if len(q1\_tokens) == 0 or len(q2\_tokens) == 0: return length\_features  # Absolute length features length\_features[0] = abs(len(q1\_tokens) - len(q2\_tokens))  #Average Token Length of both Questions  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 |
| def test\_fetch\_fuzzy\_features(q1,q2):  fuzzy\_features = [0.0]\*4  # fuzz\_ratio fuzzy\_features[0] = fuzz.QRatio(q1, q2)  # fuzz\_partial\_ratio fuzzy\_features[1] = fuzz.partial\_ratio(q1, q2)  # token\_sort\_ratio fuzzy\_features[2] = fuzz.token\_sort\_ratio(q1, q2)  # token\_set\_ratio  fuzzy\_features[3] = fuzz.token\_set\_ratio(q1, q2) return fuzzy\_features |
| def query\_point\_creator(q1,q2):  input\_query = []  # preprocess q1 = preprocess(q1) q2 = preprocess(q2)  # fetch basic features input\_query.append(len(q1)) input\_query.append(len(q2))  input\_query.append(len(q1.split(" "))) input\_query.append(len(q2.split(" ")))  input\_query.append(test\_common\_words(q1,q2)) input\_query.append(test\_total\_words(q1,q2)) input\_query.append(round(test\_common\_words(q1,q2)/test\_total\_words(q1,q2),2))  # fetch token features  token\_features = test\_fetch\_token\_features(q1,q2) input\_query.extend(token\_features)  # fetch length based features length\_features = test\_fetch\_length\_features(q1,q2) input\_query.extend(length\_features)  # fetch fuzzy features  fuzzy\_features = test\_fetch\_fuzzy\_features(q1,q2) input\_query.extend(fuzzy\_features)  # bow feature for q1 q1\_bow = cv.transform([q1]).toarray()  # bow feature for q2  q2\_bow = cv.transform([q2]).toarray()  return np.hstack((np.array(input\_query).reshape(1,22),q1\_bow,q2\_bow)) |

q1 = 'Where is the capital of India?' q2 = 'What is the current capital of Pakistan?' q3 = 'Which city serves as the capital of India?'



import pickle

pickle.dump(rf,open('model.pkl','wb')) pickle.dump(cv,open('cv.pkl','wb'))