Duplicate Question Pairs

Sumbitted by:- Hardik Verma (102283044)

Namit Nayyar (102103831)

Introduction:-

In recent years, the proliferation of online platforms and communication tools has led to an unprecedented amount of textual data being generated and shared. As a result, the need for effective natural language processing (NLP) solutions has become increasingly crucial. One common challenge in this domain is identifying duplicate question pairs, where different users may pose similar or identical questions.

Problem Statement:

The problem of detecting duplicate question pairs is a fundamental issue in NLP and has significant implications for various applications, such as information retrieval, question answering systems, and community moderation. The goal of this machine learning project is to develop a robust and efficient model that can accurately identify duplicate question pairs.

Key Challenges:

Semantic Understanding: Determining duplicate questions requires not only syntactic analysis but also a deep understanding of the semantic meaning of the questions. Questions may be phrased differently but convey the same underlying intent.

Data Imbalance: The dataset may be imbalanced, with a small percentage of question pairs being duplicates. This imbalance can pose challenges in training a model that generalizes well to both duplicate and non-duplicate pairs.

Variability in Expression: Users may pose similar questions using diverse expressions, making it challenging to create a model that can generalize across a wide range of linguistic variations.

Computational Efficiency: Processing large volumes of textual data efficiently is crucial for real-world applications. The model should be scalable to handle a significant number of question pairs in real-time.

Dataset:-

Dataset Link - https://www.kaggle.com/c/quora-question-pairs

Number of columns = Number of attributes = 6

Number of rows = Number of Tuples = 404290

Name of attributes = id, qid1, qid2, question1, question2, is_duplicate

Methodology:-

(I) Bag of Words:- In natural language processing (NLP), a "bag of words" (BoW) is a simple and popular way of representing text data for analysis. It's a method where a piece of text (such as a sentence or a document) is represented as an unordered collection of words,

disregarding grammar and word order but keeping track of word frequency.

For example, consider the two sentences:

Sentence 1: "The cat sat on the mat."

Sentence 2: "The dog played in the garden."

Creating a bag of words representation might result in a vocabulary like:

{"the": 1, "cat": 2, "sat": 3, "on": 4, "mat": 5, "dog": 6, "played": 7, "in": 8, "garden": 9}

The vectorized representations of the sentences might be:

Sentence 1: [1, 1, 1, 1, 1, 0, 0, 0, 0]

Sentence 2: [1, 0, 0, 0, 0, 1, 1, 1, 1]

Each number in the vector corresponds to the frequency of the word at that position in the vocabulary within the respective sentence.

(II) Count Vectorizer:- Count Vectorizer is a popular tool in Python's scikit-learn library used to convert a collection of text documents into a matrix of token counts, essentially implementing the bag of words model. It simplifies the process of creating a bag of words representation.

(III) Random Forest Classifier:- A Random Forest Classifier is an ensemble learning method used for both classification and regression tasks. It operates by constructing a multitude of decision trees during training and outputs the mode (for classification) or mean prediction (for regression) of the individual trees. The "forest" in Random Forest comes from the idea that multiple decision trees together can provide more robust and accurate predictions than a single decision tree.

Advantages over other models:-

Random Forest classifiers offer several advantages over other machine learning models, making them a popular choice for various applications. Here are some of the key benefits of Random Forests:

1. High Accuracy:- Random Forests generally provide high accuracy in both classification and regression tasks. The ensemble nature of the model helps reduce overfitting and improves generalization to unseen data.
2. Robust to Overfitting:- By aggregating the predictions of multiple decision trees, Random Forests are less prone to overfitting compared to individual decision trees. The random selection of features for each split and the use of bootstrapped samples contribute to the model's robustness.
   Handle Both Numerical and Categorical Data:- Random Forests can handle a mix of numerical and categorical features without requiring extensive preprocessing. They are capable of dealing with a wide range of data types.
3. Feature Importance:- The model provides a measure of feature importance, indicating which features contribute most to the predictive performance. This information can be valuable for feature selection and gaining insights into the underlying patterns in the data.
4. Implicit Handling of Missing Data:- Random Forests can handle missing values in the dataset. When making a decision at a particular node, if a certain feature is missing, the model can still make a decision based on other available features.
5. Efficient Handling of Large Datasets:- Random Forests can efficiently handle large datasets with a high number of features. The parallelization of tree construction allows for faster training times compared to some other algorithms.

6. Versatility:- Random Forests can be applied to a wide range of tasks, including classification and regression. They are suitable for various types of data and are less sensitive to the choice of hyperparameters.

Accuracy in Random Forest Classifier:

```
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)
```

```
0.7635
```

(IV) How to further increase accuracy:

To further increase accuracy we will have to do some kind of Feature Generation. By increasing the number of features ,we try to increase the efficiency of our model.

The Features we added:

Q1len - Character length of Question 1

Q2len - Character length of Question 2

Q1words - Number of words in Question 1

Q2words - Number of words in Question 2

Wordscommon- Number of common unique words

Wordstotal- Sum of total number of words in both questions

Wordshare - ratio of words common to words total

Accuracy after adding these features:

After adding these features with earlier BOW, we get an accuracy of 77.2%

```
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)
```

✓ 4m 55.9s

```
0.7726
```

How To Improve Accuracy further:

On further research on the contest, we found some of the features that will help us to increase the accuracy further. These features are divided into 3 categories:-

1.Token Features

Tokens- All words in the sentence.

Stopwords- All the words that are used in sentence forming.

Word- All tokens except stopwords are simply called stop words.

CWC-min: Ratio of number of common words to length of smaller question.

CWC-max: Ratio of number of common words to length of larger question.

CSC-min: Ratio of number of common stopwords to min number of stop words in pair.

CSC-max: Ratio of number of common stopwords to max number of stop words in pair.

CTC-min: Ratio of number of common tokens to larger token count between pair.

CTC-max: Ratio of number of common tokens to larger token count between pair.

Lastword-eq: 1 if last word of both question is same, 0 otherwise.

Firstword-eq: 1 if last word of both question is same, 0 otherwise.

2.Length based Features

Mean_len: Mean of the length of the two questions

Abs_len_diff: Absolute difference between the length of the two questions

Longest_substr_ratio: Ratio of length of the longest substring among the two questions to the length of the smaller question.

3.Fuzzy Features

Fuzzy Features

Fuzz_ratio

Fuzz_partial_ratio

Token_sort_ratio

Token_set_ratio

Accuracy of the model with all features.

After adding all the features, we now have a total of 22 features.

With these features we got an accuracy of 79.44%.

```
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)
```

✓ 1m 43.8s

```
0.7944
```

Other Model options

One of other famous model we can use is XGB classifier.

With XGB classifier, we actually get a higher accuracy of 79.6%

```
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(X_train,y_train)
y_pred1 = xgb.predict(X_test)
accuracy_score(y_test,y_pred1)
```

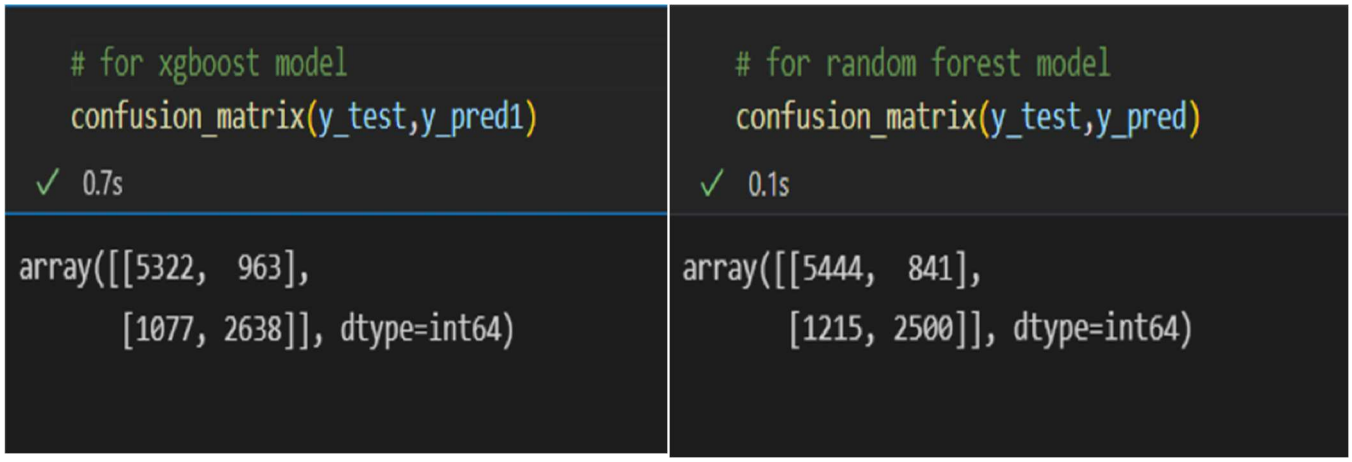
✓ 4m 51.9s

```
0.796
```

Q. Which model should be adopted?

To answer that question, we have to keep in mind what we actually want from the model. We need to know if the questions are similar or not, i.e., if we need to provide the same answer or different. So, we need to observe how the misclassification will affect us. Let's plot the confusion matrix.

For our particular problem, we will need the model such that it has less False Positive rate, because if the model predicts these questions as duplicated and they are actually non-duplicated, so the answer given by algorithm will not be correct. As the False Positive (FP) rate of random forest classifier(841) is less than that of Xgboost(963), we will adopt random forest classifier for our model.



Individual Roles:-

Hardik Verma:- Data Preprocessing

Namit Nayyar:- Feature Extraction