SB Capstone 3: Quora Semantic Pairing Report

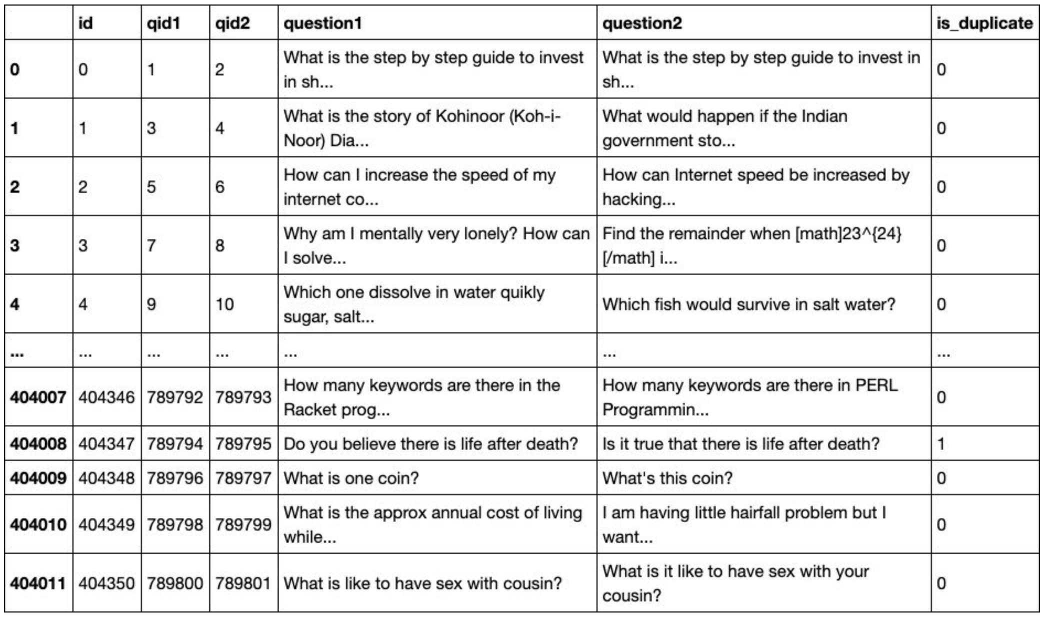
Problem Set Description

Quora is a widely used online forum where users can post and answer questions. As a large gathering place of people's queries across a range of topics, duplicate questions are unavoidable. For instance, it is likely to have the questions *"Do you believe there is life after death?"* and *"Is it true that there is life after death?"* posted on separate forums although they essentially ask the same question. It would be beneficial for us if these duplicates can be detected so that once a user inputs a question, they can be recommended to a forum with a similar question. The user can then make the decision to scroll through the recommended forum or post the original question into a new one.

Data

The dataset q_quora.csv is curated through Kaggle, a forum that generates datasets for data analytics practitioners. The data is 404012 rows and 6 columns after performing data cleaning (described in *Data Cleaning* section). The features consist of:

- *id:* unique id for a pair of questions to be compared
- *qid1:* unique id for question 1 of a pair
- *qid2:* unique id for question 2 of a pair
- *question1:* question 1 of a pair
- *question2:* question 2 of a pair
- *is_duplicate:* binary indicator of whether the pair of questions have the same semantic meaning (1 if duplicate, 0 if not)



	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0
...	...	...	...	...	...	...
404007	404346	789792	789793	How many keywords are there in the Racket prog...	How many keywords are there in PERL Programmin...	0
404008	404347	789794	789795	Do you believe there is life after death?	Is it true that there is life after death?	1
404009	404348	789796	789797	What is one coin?	What's this coin?	0
404010	404349	789798	789799	What is the approx annual cost of living while...	I am having little hairfall problem but I want...	0
404011	404350	789800	789801	What is like to have sex with cousin?	What is it like to have sex with your cousin?	0

Project Overview

We will be building three models for this project. For the tf-idf vectorizer and universal sentence encoding model, we will do the following:

1. We will use natural language processing tools to create tf-idf or word embedding of the questions, which is essentially converting the question texts into numerical vector representations.
2. After the conversion, we use metrics like cosine similarity and Levenshtein distance to assign a similarity score between a pair of numerical question vectors.
3. We split the dataset into training and test sets.
4. On the training data, we set a range of threshold values between 0 and 1. For each threshold, any similarity score above the threshold will be assigned 1 (duplicate) and 0 if equal to or less than the threshold.
5. With the predicted assignments (0 or 1), we plot various thresholds (on x-axis) and its accuracy (y-axis) in predicting the duplicate and non-duplicate class on the training data. We determine the *optimal threshold* value which is where the line plots for the duplicate and non-duplicate accuracies intersect.
6. We will use this optimal threshold value to predict the class on the test set and output the confusion matrix which will show precision, recall and f-1 score for the duplicate and non-duplicates.

Model 3, which will be the FuzzyWuzzy model will be the simplest case where it uses the Levenshtein distance (LD) metric between two sentences to compute a score. Once we get the similarity scores with the LD , steps 3 through 6 are the same.

Data Cleaning

The data cleaning portion consisted of the following steps:

1. *Missing values:* After filtering the rows with the missing values, each row was missing a question for a pair. For instance, the pair id 105796 had *"How can I develop android app?"* for question 1 and *Nan* for question 2. Since you can't compare a string with a missing value, we omitted these rows from our dataframe as there were only three rows with missing values.
2. *Unique values for predictor feature:* The predictor feature *"is_duplicate"* should only have the values 0 and 1 representing non-duplicate and duplicate pairs respectively. A look at the unique values showed there were instances of text in this column. Since this doesn't make sense, we omitted these rows as well as it consisted only 1% of the rows of this dataset.
3. *General data exploration:* We performed basic analysis such as unique value counts for each feature.

Preprocessing

Since the preprocessing for each model is distinct, we will go over each separately.

Model 1 Preprocessing: TF-IDF Vectorizer

Prior to using the TF-IDF vectorizer, we must preprocess the question text into corpuses or bag of words. The function *pre_process* will perform the following:

1. Convert all words of question to lower case.
2. Create word tokenizations for each question.
3. Remove all stopwords unless the only remaining words are stop words.
4. Optimize corpus by using unidecode.

Next, we convert the corpus of our question pairs into tf-idf vectors representing the term frequency per sentence and inverse data frequency of a word across both sentences. Essentially, the tf-idf vectorizer will take the word frequency within each sentence and also calculate the inverse of the occurrences of the word across both sentences and create numerical vectors representing this term frequencies and inverse data frequencies. Similar sentences will have tf-idf vectors that are almost all 0. For calculation examples, I will provide a link at the end of the report that shows how the tf-idf vectorizer works for a pair of sentences. The tf-idf pairs are the numerical vectors that will be used in the model.

Model 2 Preprocessing: Universal Sentence Encoder

In the TF-IDF vector model, we had to determine the preprocessing steps to creating a corpus of words, such as converting questions to lower cases and removing stop words. In the universal sentence encoder model, there is a built in embedding tool that will turn the question texts into the word embeddings as described in the *Model Considerations* section. The Universal Sentence Encoder converts the sentence into a 512 dimensional vector. In this vector space, words that are closer in the vector space means they are similar in meaning. We will use the two 512 dimensional vectors in the modeling section.

Model 3 Preprocessing: FuzzyWuzzy

The FuzzyWuzzy metric is a method for comparing string similarities. It uses the Levenshtein Distance metric to calculate the differences between two strings, or our question pairs. Here is an informal definition from Wikipedia:

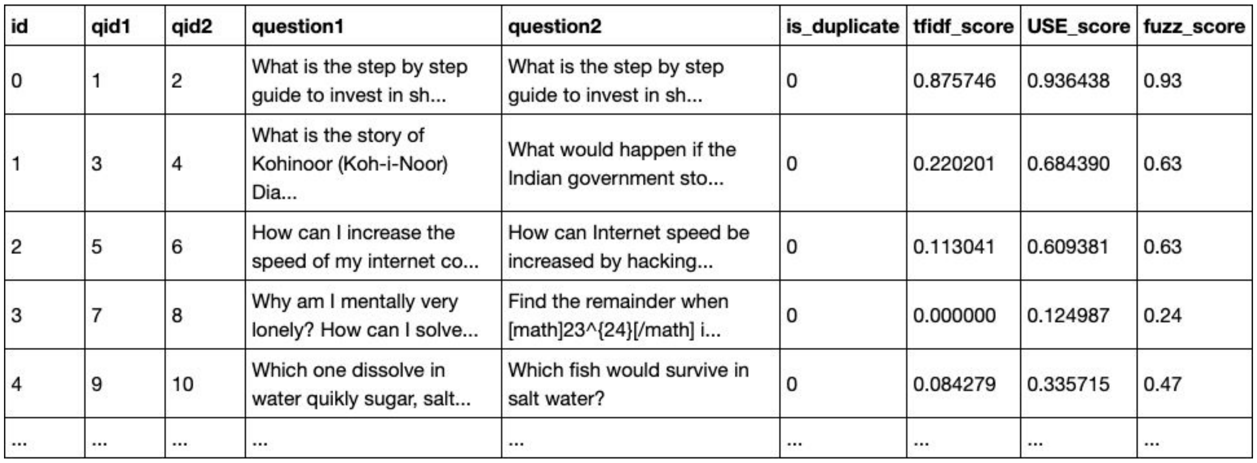
"*Informally, the Levenshtein distance between two words is the minimum number of single-character edits (insertions, deletions or substitutions) required to change one word into the other.*"

We will use the built-in *token_sort_ratio* method to compute the Levenshtein distance score between pairs of questions. The output score will be between 0 and 100. We will divide the outputs by 100 to give a value between 0 and 1, which is the range for the cosine similarity.

Modeling

Cosine similarity & Levenshtein distance

For the tf-idf vector and USE word embedding representations for model 1 and 2 respectively, we will compute the cosine similarity between each pair. The cosine similarity score for model 1 will be defined as *"tf-idf score"* and the score for model 2 will be defined *"USE_score"*. We already have the Levenshtein distance score for model 3 (FuzzyWuzzy) which we will define as *"fuzz_score"*. The following table shows the results:



id	qid1	qid2	question1	question2	is_duplicate	tfidf_score	USE_score	fuzz_score
0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0	0.875746	0.936438	0.93
1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0	0.220201	0.684390	0.63
2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0	0.113041	0.609381	0.63
3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0	0.000000	0.124987	0.24
4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0	0.084279	0.335715	0.47
...	...	...	...	...	...	...	...	...

Determining Optimal Similarity Score Thresholds from Training Data

We divide our data into training and test sets. Next, we must determine the optimal similarity score thresholds from our training data that will provide the best accuracy in predicting the class a pair of questions belongs to (duplicate or non-duplicate). For each model, we set the following similarity threshold scores in a list: [0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]. To determine the accuracy for each class, we divide the training data into duplicates and non-duplicates by using the *"is_duplicate"* values. Then, for a respective model and a threshold value:

1. Assign 1 to the question pair if the similarity score (cosine similarity or Levenshtein) is above the threshold and 0 if the question is less than or equal to the threshold.
2. For both the duplicate and non-duplicate classes, take the total number of the predicted class for that threshold and divide it by the total number of actual instances of that class. This provides the accuracy measure.
3. Also take the average of the duplicate and non-duplicate accuracies.

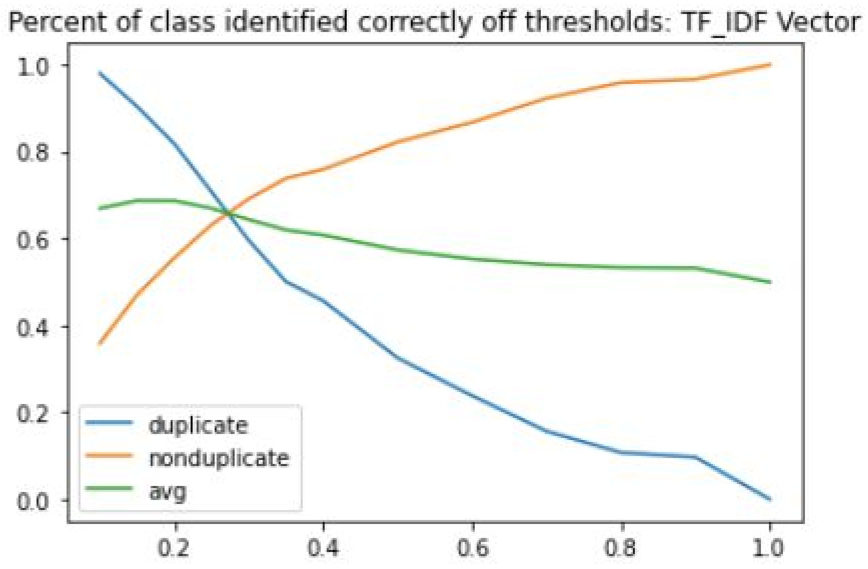
Doing this for each threshold similarity score value for a model, we can create line plots with the thresholds on the x-axis and accuracies on the y-axis. The optimal threshold will be the intersection of the duplicate and non-duplicate line plots.

Testing on the Test Data
We will use the determined threshold on our test data. On the test data, perform step 1 of the previous section where we use the threshold to assign/predict 1 or 0. Next, plot a confusion matrix that compares the predicted class values to the actual class values. The matrix will show the precision, recall and f-1 scores.

Results

Model 1 Plot and Confusion Matrix: TF-IDF Vectorizer

Here we plot the accuracy of our classes vs the cosine similarity thresholds based on the tf-idf vectorizer pairs. On this training data, the optimal point is around 0.3 cosine similarity score. We use this threshold and test its performance on the test data.

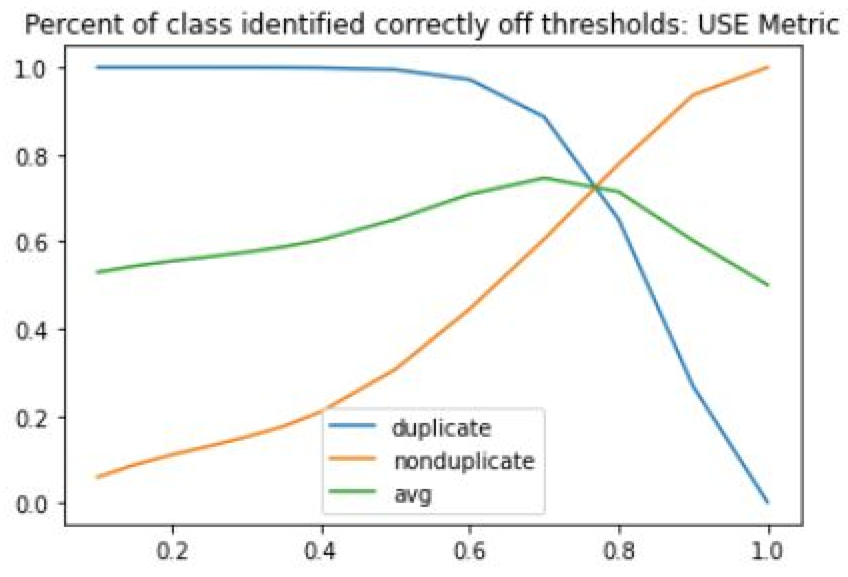


The following confusion matrix shows the results of using a cosine similarity score of 0.3 for tf-idf question pairs on the test data. We obtain a precision and recall of 0.75 and 0.69 respectively for the non-duplicate pairs and a precision and recall of 0.53 and 0.60 for the duplicate question pairs. The overall f-1 score is 0.66.

```
f1-score=0.661
              precision    recall  f1-score   support

           0       0.75      0.69      0.72     50967
           1       0.53      0.60      0.56     29836

    accuracy                           0.66     80803
   macro avg       0.64      0.64      0.64     80803
weighted avg       0.67      0.66      0.66     80803
```

Model 2 Plot and Confusion Matrix: Universal Sentence Encoder (USE)

Here we plot the accuracy of our classes vs the cosine similarity thresholds based on the USE 512 dimensional word embedding vectors. On this training data, the optimal point is around 0.78 cosine similarity score. We use this threshold and test its performance on the test data.

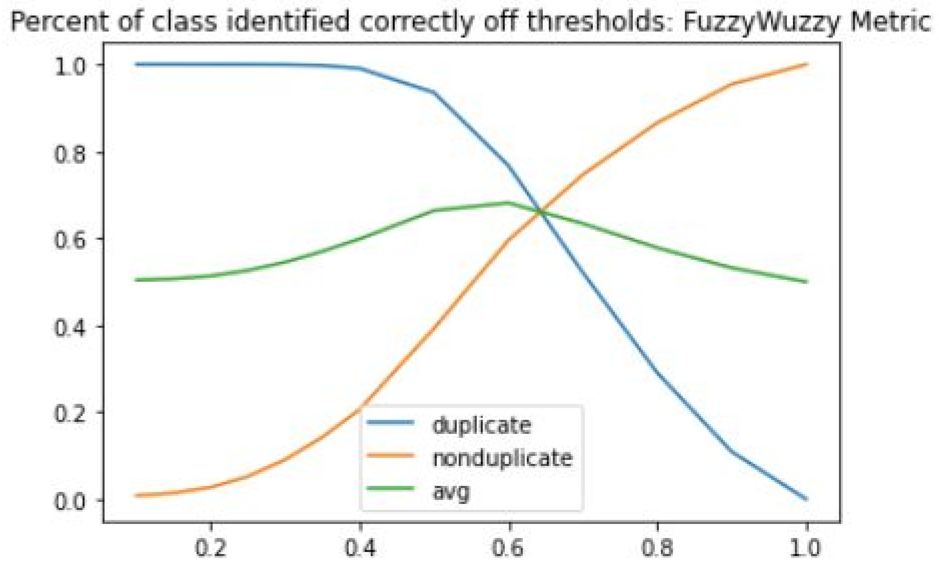


The following confusion matrix shows the results of using a cosine similarity score of 0.78 for USE word embeddings on the test data. We obtain a precision and recall of 0.82 and 0.74 respectively for the non-duplicate pairs and a precision and recall of 0.62 and 0.71 for the duplicate question pairs. The overall f-1 score is 0.737. The performance is much more robust than our tf-idf vectors. This could be due to the fact that all sentences are encoded into the same pattern of 512 dimensional vectors. These vectors can be compared in a predetermined word-vector space where words/sentences with similar semantics have similar directional vectors.

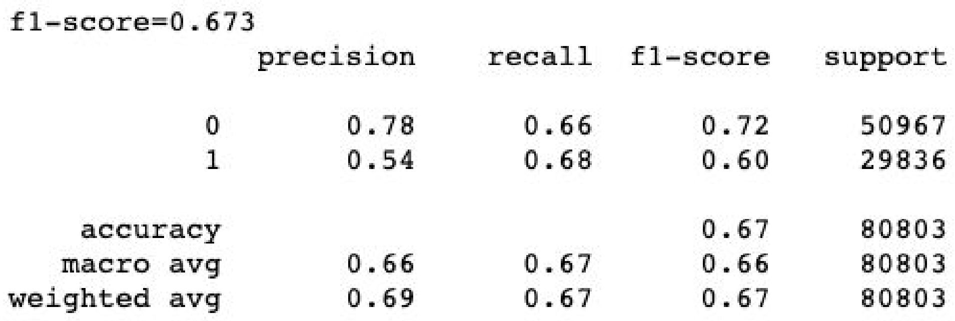
```
f1-score=0.737
              precision    recall  f1-score   support

           0       0.82      0.74      0.78     50967
           1       0.62      0.71      0.66     29836

    accuracy                           0.73     80803
   macro avg       0.72      0.73      0.72     80803
weighted avg       0.74      0.73      0.74     80803
```

Model 3 Plot and Confusion Matrix: FuzzyWuzzy

Here we plot the accuracy of our classes vs the Levenshtein Distance (LD) thresholds based on the fuzzywuzzy pairs. On this training data, the optimal point is around 0.64 LD score. We use this threshold and test its performance on the test data.



The following confusion matrix shows the results of using a Levenshtein Distance score of 0.64 for fuzzywuzzy question pairs on the test data. We obtain a precision and recall of 0.78 and 0.66 respectively for the non-duplicate pairs and a precision and recall of 0.54 and 0.68 for the duplicate question pairs. The overall f-1 score is 0.673. The performance is better than our tf-idf vectorizer. This could be due to the fact that fuzzywuzzy package has built-in tools that perform the preprocessing and calculations for us under the hood while we had to determine the pre-processing steps in the tf-idf model. Perhaps including the stop words could build a more robust model for the tf-idf. Model 3 did not perform better than the USE model.



```
f1-score=0.673
              precision    recall  f1-score   support

           0       0.78      0.66      0.72     50967
           1       0.54      0.68      0.60     29836

    accuracy                           0.67     80803
   macro avg       0.66      0.67      0.66     80803
weighted avg       0.69      0.67      0.67     80803
```

Best Model: Universal Sentence Encoder
The Universal Sentence Encoder model had the best performance. It leverages powerful tools such as the *Deep Averaging Network* to create our vectors that prove useful to sentence similarity.

Model Deployment

In terms of model deployment, the following idea is suggested. Generate two questions. Feed them into the USE model that we created. Under the hood, it will convert the questions into the 512 dimensional word embeddings and compute the cosine similarity between the vectors. If the score is above our determined threshold of 0.78, we will say the questions are duplicates. Otherwise, the questions are not duplicates.

Future Improvements

Metric Other than Vector Similarity
For model 1 and model 2, we used the cosine similarity to compute the similarity score between numerical vectors. Maybe we can use K nearest neighbor to better understand if the two questions pairs are close enough in distance to be considered similar. Going forward, I would research into more similarity score options

Data Cleaning
While building the models, I realized that some question pairs were labeled as non-duplicates when they arguably are duplicates. One instance is question id 404350. Though mildly inappropriate, the questions are, *"What is like to have sex with cousin?"* and *"What is it like to have sex with your cousin?"* If we can find instances like this and change the values, the performance of our model could improve.

References:

TF-IDF Vector Calculation Example for a pair of sentences:
https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/

Universal Sentence Encoder Vector Explanation:
https://amitness.com/2020/06/universal-sentence-encoder/

Fuzzywuzzy Explanation:
https://amitness.com/2020/06/universal-sentence-encoder/