Quora Question Pairing

CS 6120 Natural Language Processing
Dr. Uzair Ahmed

Dhavan Kumarpal Sanghvi
Foram Shah
Nidhi Bodar

August 14, 2022

Table Of Contents

Abstract

With our project we want to explore the domain of natural language processing by identifying duplicate questions. The application is essential as it reduces the effort of writers in terms of answering multiple versions of the same question and saves time for seekers.

Introduction

The issue of duplicate questions stems from the enormous number of visitors on the Quora website, making it hard to avoid having similar worded questions from different users. To tackle this problem we apply simple feature engineering as well as neural-network based models.

The task of identifying duplicated questions can be viewed as an instance of the paraphrase identification problem, which is a well-studied NLP task that uses natural language sentence matching (NLSM) to determine whether two sentences are paraphrased or not.

And so, our main goal for this project is to create a model that takes two questions at a time, compares their similarity and predicts if provided questions are equivalent or not with accuracy.

For the task of paraphrase identification, three types of different frameworks have been proposed; TF-IDF with Word level XG Boost classifier, TF-IDF with N-Gram level XG Boost classifier, Bert to predict their similarity. The above models are trained with a dataset, namely, "Quora Question Pairs" from Kaggle.

There are many steps, and sub-goals, that we need to take and achieve in order to reach this final goal, making this an interesting and challenging task. Our dataset that we are working with, described in the next section, requires significant preprocessing focused on size reduction. We also need to create, train, and evaluate multiple models (described in more detail below) in order to properly and confidently choose the best one to deliver. Finally, in order to properly document our process, we wrote this paper and recorded a presentation.

Related Work

The authors in [1] conducted extensive exploration of the dataset and used linear and tree-based models. And their conclusion was that a simple Continuous Bag of Words neural network model had the best performance, outdoing more complicated recurrent and attention based models. They conducted error analysis and found some subjectivity in the labeling of the dataset. In [2], The authors were more focused on feature extraction, they used Bag of Words including Countvectorizer, and Term Frequency-Inverse Document Frequency with unigram for XGBoost and CatBoost. Furthermore, they also experimented with WordPiece tokenizer which improves the model performance significantly and achieved up to 97 percent accuracy. The author in [3] proposed the first framework based on a "siamese" neural network and to cope with the limitations they investigated bilateral multi perspective matching model (BiMPM) which also applies the "compare-aggregate" framework. One of the best-performing attention-based models is the attention-based convolutional neural network (ABCNN), which achieves attention at multiple levels of granularity (word-level, phrase-level and sentence-level) by stacking multiple convolutional layers.

Data

Exploratory Data Analysis

We have performed multiple steps to perform feature extraction starting from simply counting and calculating percentage of similar and duplicate questions. As mentioned above, there are two columns question1 and question2. We've performed common and total word count, word share ratio which makes the use of the number of words similar between two questions.

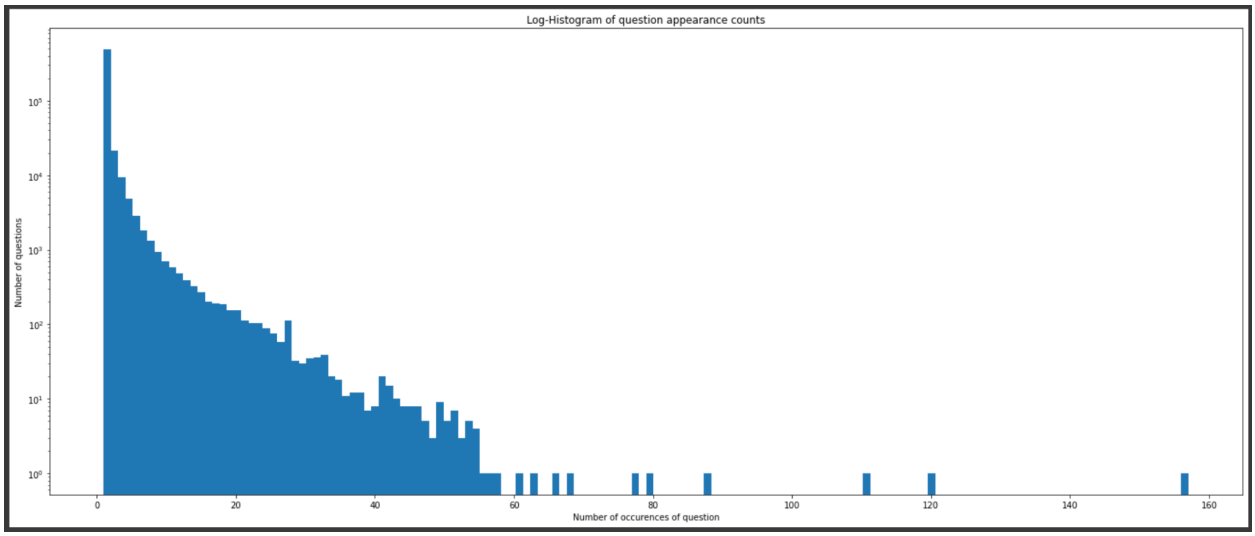

Fig 1 : Frequency of Occurrence of Question

Furthermore, We have counted how many times a particular question is appearing in the column by plotting a log histogram which calculates the occurrence of a question.

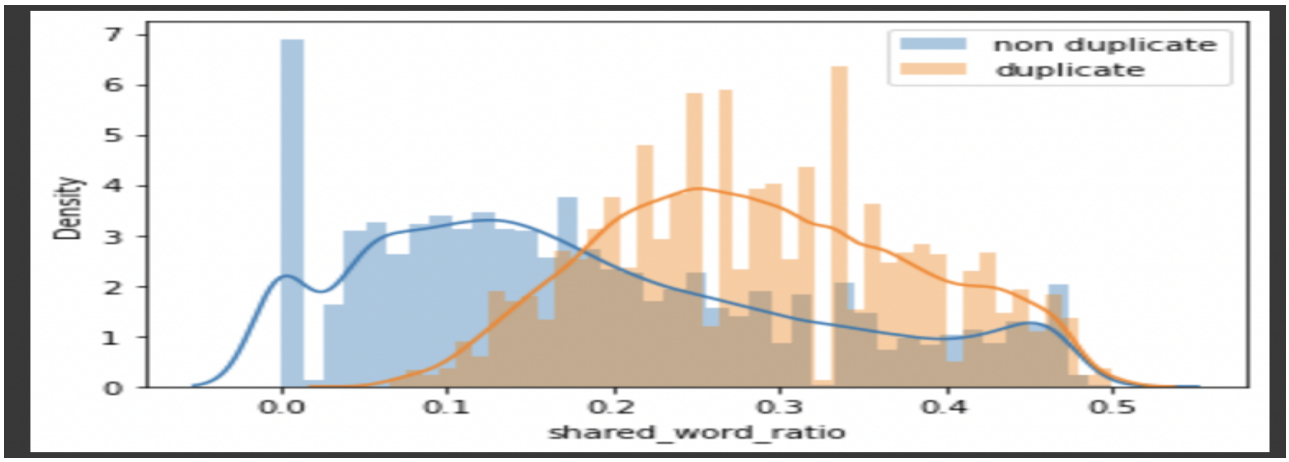

Fig 2 : Density graph of shared_word_ratio

We have also plotted the word clud graph for examining the frequency of duplicate and non-duplicate words in questions. Therefore, these are the parameters which are being considered while conducting feature extraction and data exploration.

Fig 3 : Word Cloud for duplicate question pairs

Preprocessing

All preprocessing for this project was done in Python. There were three main steps to prepare the data for model generation: (1) First challenge was to upload it to Google Drive and a Google Colab Notebook in its entirety. This was an issue because the only way to properly retain all the data was to load the .zip files to Google Drive and then unzip and read the .txt files as .csv files that are separated by either an exclamation mark, a period, or a question mark. (2) The second step was to shrink the training and testing data to a manageable size. From our readings we realized that if we used the entire training set to train our models, it could take weeks, and that would jeopardize our ability to complete the project in time. (3) The final step was to tokenize the sentences into an array of words and then lemmatize the same so that we can make sure we have each word's intended meaning and not the literal meaning in the sentence.

```
[254] df_train_0 = df_train[df_train['is_duplicate'] == 0]
      df_train_1 = df_train[df_train['is_duplicate'] == 1]

      num_of_samples = 30000

      # LIMITING DATASET DUE TO RAM ISSUES
      # if df_train_0.shape[0] < df_train_1.shape[0]:
      #   num_of_samples = df_train_0.shape[0]
      # else:
      #   num_of_samples = df_train_1.shape[0]

      equal_df = pd.concat([df_train_0.sample(n=num_of_samples, random_state=1), df_train_1.sample(n=num_of_samples, random_state=1)])
      x = equal_df.copy()
      y = x['is_duplicate']
      x = x.drop('is_duplicate', axis=1)
      x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.2, random_state = 42, stratify=y)
```

Fig 4: Reducing the size of data

Model Selections

TF-IDF + XGBoost

The first model we chose to implement a TF-IDF model with XGBoost model, normalization to a sparse matrix of occurrence counts, from word level, N-gram level and Character level. TF-IDF is the product of two statistics, term frequency and inverse document frequency, it is one of the most popular term-weighting schemes. N-Gram as “A contiguous sequence of N items from a given sample of text or speech”. We have used the range of (2,3) grams to fit the TF-IDF vector on data. XGBoost performs better than most predictive models that is why we are going to be using it to train our data. XGBoost has been used for all the above mentioned levels that can be seen from the code below.

```
tfidf_char_vect = TfidfVectorizer(analyzer='char', token_pattern=r'\w{1,}', ngram_range=(2,3), max_features=5000)
tfidf_char_vect.fit(pd.concat((x_train['clean_question1'],x_train['clean_question1'])).unique())
train_q1_trans_3 = tfidf_char_vect.transform(x_train['clean_question1'].values)
train_q2_trans_3 = tfidf_char_vect.transform(x_train['clean_question1'].values)
labels_3 = y_train['is_duplicate'].values

x_tfidf_3 = scipy.sparse.hstack((train_q1_trans_3,train_q2_trans_3))
y_tfidf_3 = labels_3

x_train_tfidf_3, x_test_tfidf_3, y_train_tfidf_3, y_test_tfidf_3 = train_test_split(x_tfidf_3, y_tfidf_3, test_size = 0.2, random

xgb_ngram_char_model = xgb.XGBClassifier(max_depth=50,
                                         n_estimators=80,
                                         learning_rate=0.01,
                                         colsample_bytree=.7,
                                         gamma=0,
                                         reg_alpha=4,
                                         objective='binary:logistic',
                                         eta=0.3,
                                         silent=1,
                                         subsample=0.8).fit(x_train_tfidf_3, y_train_tfidf_3)

xgb_ngram_char_train_prediction = xgb_ngram_char_model.predict(x_train_tfidf_3)
xgb_ngram_char_test_prediction = xgb_ngram_char_model.predict(x_test_tfidf_3)

print('==> Training score:', f1_score(y_train_tfidf_3, xgb_ngram_char_train_prediction, average='macro'))
print('==> Validation score:', f1_score(y_test_tfidf_3, xgb_ngram_char_test_prediction, average='macro'))

print(classification_report(y_test_tfidf_3, xgb_ngram_char_test_prediction))
```

Fig. 5 Character Level TF-IDF + XGBoost

BERT

Our second model we wanted to implement is BERT. It stands for Bidirectional Encoder Representations from Transformers, is based on Transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection. The Self-attention layer is applied to every layer and the result is passed through a feed-forward network and then to the next encoder. Each position outputs a vector of size 768 for a Base model which is the hidden_size.

```
input1 = Input(shape=(), dtype=tf.string)
input2 = Input(shape=(), dtype=tf.string)
input_wms = Input(shape=(1,), dtype=tf.float16)
input_wcd = Input(shape=(1,), dtype=tf.float16)

embed1 = embed(input1)
embed2 = embed(input2)

dist = Lambda(lambda x: K.sqrt(K.sum(K.square(x[0] - x[1]), axis=-1, keepdims=True)))([embed1,embed2])

concat = Concatenate(axis=1)([dist, input_wms, input_wcd])

hidden = Dense(9, activation="relu", kernel_regularizer=l2(1e-4))(concat)

out = Dense(1, activation="sigmoid", kernel_regularizer=l2(1e-4))(hidden)
model = Model(inputs=[input1, input2, input_wms, input_wcd], outputs=out)
```

Fig 6 : BERT Model Creation

```
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=5)
model_chkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min', verbose=1, save_best_only=True)

evaluate = model.fit(x=[np.array(bert_train_q1),
                        np.array(bert_train_q2),
                        np.array(bert_shared_ratio_train),
                        np.array(bert_total_count_train)],
          y=np.array(bert_y_train),
          batch_size=128,
          epochs=50,
          validation_data=([np.array(bert_test_q1),
                            np.array(bert_test_q2),
                            np.array(bert_shared_ratio_test),
                            np.array(bert_total_count_test)],
                           np.array(bert_y_test)),
          callbacks=[early_stop, model_chkpoint])
```

Fig 7 : Fitting the BERT Model on training data

Evaluation/Results

We evaluated each model on training, validation accuracy and F1 score and Recall Precision .

The training and validation accuracy of BERT kept increasing simultaneously with each epoch. We had to stop after 50 iterations as it remained constant after such iterations indicating no further improvement. Therefore, We were able to obtain 69.08% accuracy. Implementation of TF-IDF with XGBoost at word level gave an accuracy of 64.25% while at N-gram level, we obtained the lowest accuracy with the range of (2,3) N-grams. With Character level we obtained nearly 68.66% accuracy.

Overall, we can conclude that BERT has outperformed all the TF-IDF models by almost achieving a good score of nearly 70%. Therefore, the BERT model has been proven the best model for the given dataset.

Model Name	**Training Accuracy**	**Validation Accuracy**	**Precision**	**Recall**	**F1-Score**
BERT	68.18	69.08	0.67	0.69	
Word level TF-IDF + XGBoost	66.95	64.25	0.61	0.80	0.69
N-gram level TF-IDF + XGBoost	53.06	52.27	0.55	0.96	0.70
Character level + TF-IDF + XGBoost	91.54	68.66	0.67	0.72	0.69

Fig. 8: Model Result

Future Work

Due to limited computing resources , we had to limit our dataset till 60,000. Given more resources, we could experiment with the whole dataset with different parameter settings and evaluate its performance in this domain thoroughly. Additionally, we can deploy the N-gram model and conduct an experiment for reasons of lower accuracy and evaluate its performance by implementing it again on the same dataset..

Finally, we could also extend the model to run on different datasets for different platforms such as stack overflow.

Works Cited

1. Lakshay Sharma, Laura Graesser, Nikita Nangia, Utku Evci, “Natural Language Understanding with the Quora Question Pairs Dataset” , Cornell University, June 2020
2. Andreas Chandra, Ruben Stefanus, “Experiments on Paraphrase Identification Using Quora Question Pairs Dataset”, Cornell University, June 2020.
3. Z Chen, H Zhang, X Zhang, L Zhao, “Quora Question Pairs” , 2018.
4. INDIAN JOURNAL OF SCIENCE AND TECHNOLOGY, “Identifying Similar Question Pairs Using Machine Learning Techniques”, June 2021.
5. K. Abishek, Basuthkar Rajaram Hariharan & C. Valliyammai , “An Enhanced Deep Learning Model for Duplicate Question Pairs Recognition”, August 2018.