Quora Question Pairs

BHAVIKA TEKWANI, George Mason University

Quora is a quasi-forum designed to crowdsource knowledge and opinions on a wide range of topics. In the year 2016, there there were 29,000 questions[1] asked about the US Presidential Election alone. Other popular topics included outer space, artificial intelligence, Brexit and the Rio Olympics. Needless to say, the amount of content on Quora is extremely vast. Often, users tend to ask a question that has already been asked and answered before. On Quora, this means users spend more time waiting for answers that must have already been answered but are difficult to discover through search functions. Quora has enabled several users to become influencers in certain topics based on how many questions they answer in their area of expertise. For these users, identifying identical questions may save them time in answering and improve the quality of answers. The problem we try to solve in the Quora Question Pairs Kaggle competition is that of duplicate question identification. We are interested in identifying questions with the same intent.

CCS Concepts: • **Natural Language Processing** → Semantic Analysis; **Data Mining**

KEYWORDS

text classification, data mining

1 INTRODUCTION

Quora Question Pairs is a Kaggle competition where we are provided with pairs of questions asked by users on Quora. We are tasked with predicting the probability that any two questions have the same intent and are duplicates. Often, two questions may sound similar in terms of the general topic they belong to, however, the answers may be vastly different. Take for example, the questions “Are dogs allergic to chocolate?” and “Why are dogs allergic to chocolate?”. For humans, it is easy to identify that one of these can be answered with a yes or no and the other requires an elaborate response. The problem lies in understanding natural language rather than merely processing it.

We will explain some strategies for predicting the extent to which two questions may be similar in the following sections.

First, let us take a look at the data. We are given two datasets - a train and test set. The train dataset contains the following fields - id, qid1, qid2, question1, question2 and is_duplicate. qid1 and qid2 represent the question ID on Quora whereas id is merely an identifier for a pair. is_duplicate contains a value that may be 0 (not duplicates) or 1 (duplicate).

The test set is similar, but it contains only test_id, question1 and question2. We have to predict the value of is_duplicate for each pair of questions in the test set. This value must be in between 0 and 1.

Table 1 shows a snapshot of the dataset.

Train	Test
id	test_id
qid1	question1 [text]
qid2	question2 [text]
question1 [text]	
question2 [text]	
is_duplicate [0, 1]	

Table 1. Train and test data snapshot

The train dataset has 404289 cases and the test set has 2345795 cases. Thus, the test set is approximately 5 times larger than the train set.

We observe the distribution of duplicate question pairs in the train dataset. Only 37% of the train examples are marked as duplicates. Thus, we know that we have an imbalanced data problem.

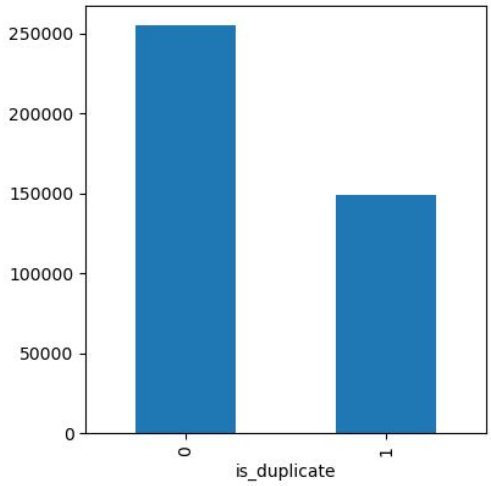

Fig. 1 The distribution of is_duplicate labels (1 indicates duplicate) in the train set

The Kaggle leaderboard evaluates submissions only on 35% of the dataset. The metric used in this competition is log loss. It is given by the following equation 1. The goal is to minimize the log loss as we make predictions for is_duplicate.

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{i,j}\log(p_{i,j})$$

Equation 1. Log loss metric

In order to avoid hand-labeling of the dataset and gaming of the leaderboard, Quora has introduced some computer generated question pairs in the test dataset. We now look at ideas for feature generation and modeling.

The text in both train and test sets has been preprocessed before feature extraction.

The steps involved in preprocessing are:

1. Removing inconsistencies in data: We replace instances of “9 11” with “911”, “dms” with “direct messages”, “USA” with “America”. This is mostly done to remove common mistakes that users make either in grammar or in spelling. The list of corrections has been adopted from user currie32 on Kaggle. I’ve added some common spelling errors to this list and processed the text accordingly.
2. Stopword removal: We use NLTK’s stopword list and add some stopwords of our own. We filter out any occurrence of these words from the question pairs.
3. Remove punctuation.
4. Stemming: We use the Porter Stemmer in NLTK to replace each word with its root word. This reduces the size of our corpus and makes the process of identifying questions that mean the same thing but vary in tense easier.

When missing values were found in the question pairs, we replaced them with empty strings but they are not excluded from analysis (i.e., they are still in the training set).

2 EXPERIMENTAL AND COMPUTATIONAL DETAILS

2.1 Feature Engineering

Based on the nature of the text data we have, which is short questions and not long paragraphs on inter-related documents, we find that some very simple summary statistics and metrics may be helpful as features. We calculate some features that will be used in the models and some which while I created for analysis or use in deep neural networks, didn’t quite fit well into the structure required by

these neural networks. Nevertheless, we will discuss all types of features built for analysis.

Let us label each featureset combining related concepts as FS-1, FS-2, FS-3, etc. Broadly, we have the following types of features - lengths of questions, word counts and character counts; fuzzy text features; wordshare index; word mover's distance and normalized word mover's distance; Sent2Vec features and a set of distance metrics, skew and kurtosis between the Sent2Vec representation of question pairs. We summarize the features in Table 2.

Featureset	Features
FS-1	Length of question 1 & 2 - len_q1, len_q2
	Length of question 1 & 2 after cleaning - len_q1c, len_q2c
	No. of words in question 1 & 2 after cleaning - words_q1c, words_q2c
	No. of characters in question 1 & 2 after cleaning - chars_q1c, chars_q2c
	Wordshare index (no of common words between question 1 & 2, normalized)
FS-2	Fuzzy features - QRatio, WRatio, Partial Ratio, Partial Tokenset Ratio, Tokenset Ratio, Partial Tokensort Ratio
FS-3	Word Mover's Distance & Normalized Word Mover's Distance
FS-4	Distance metrics - Cosine, Cityblock, Canberra, Euclidean, Minkowski, Braycurtis, Jaccard, skew of question 1 & 2 vectors.

Table 2. Featuresets created for analysis

Sent2Vec: In a Word2Vec model, each word is converted to a single dimensional vector with 300 elements. We use a similar logic and extend it to sentences as a whole. In our dataset, we consider each question in a question pair as a sentence and generate a vector representation (embedding) for it. The algorithm is as follows and was first introduced by Pagliardini et al.[3] as an unsupervised model.

While we write our own implementation of Sent2Vec, we are using Google's pretrained Word2Vec model to extract word embeddings.

Pseudocode for Sent2Vec

```
function sent2vec:
      words = word_tokenize(s)
      words = remove_stopwords(words)

      M = [ ]
      for each word in words:
            word_vec = get_wordvector(word)
            M.append(word_vec)

      for each word_vec in M:
            // Add up 300 elements for each word by index
            // Add element of word 1 to element 1 of word 2 and so on...
            v += word_vec

            // Calculate the dot product of vector 'v' with itself
            sum2 = (v * v)

            // Add up all elements of sum2 and take the square root
            sum_root = sqrt(sum2)

            return (v/sum_root)
```

The value returned from the Sent2Vec function is the sentence embedding - a 1-dimensional vector with 300 elements. We use this sentence embedding for distance calculations in FS-4. We must note that any other model like GloVe can also be used similarly for extracting word embeddings. I have not compared the performance of distance metrics when obtained via GloVe and Word2Vec, but this can be a trivial addition to the analysis too.

Word Mover's Distance (WMD): A distance function that evolved from the concept of Earth Mover's Distance. It was introduced by Kusner et al[2]. The Word Mover's Distance calculates the distance between two documents from their word embeddings. It does this by calculating the minimum distance embedded words from one document would have to travel to reach the embedded words in another document. We use the raw WMD and a normalized WMD as features for our analysis. We are using the WMD function in the *gensim* module for Python.

2.2 Feature Selection

3 RESULTS AND DISCUSSION

3.3

4 CONCLUSIONS

ACKNOWLEDGMENTS

REFERENCES

[1] A Year in Questions - Quora blog, https://qph.ec.quoracdn.net/main-qimg-5799e66af99cd2e0d6f22f3875cc92e4.webp Last accessed: 05/08/2017

[2] Matt Kusner et al., “From Word Embeddings to Document Distances,” in *International Conference on Machine Learning*, 2015, 957–966, http://www.jmlr.org/proceedings/papers/v37/kusnerb15.pdf.

[3] Matteo Pagliardini, Prakhar Gupta, and Martin Jaggi, “Unsupervised Learning of Sentence Embeddings Using Compositional N-Gram Features,” *arXiv:1703.02507 [Cs]*, March 7, 2017, http://arxiv.org/abs/1703.02507.

[4]

[5]

[6]

[7]

[8]