Quora: Is that a duplicate question?

Usman Nisar
Department of Computer Science
University of Georgia
Athens, GA 30602
`nisar@cs.uga.edu`

1 Introduction

Quora is a question-and-answer site where questions are asked, answered, edited and organized by its community of users. With a user base of well over 100 million people, it's no surprise that many people ask similarly worded questions. Multiple questions with the same intent can cause seekers to spend more time finding the best answer to their question, and make writers feel they need to answer multiple versions of the same question. Quora values canonical questions because they provide a better experience to active seekers and writers, and offer more value to both of these groups in the long term. However, the redundancy results in a poor experience for the end users.

Below, I go through my approach as I go step-by-step through this problem. Firstly, I will go through the dataset and try to dump some simple statistics followed by different approaches to solve the duplicate question.

2 Dataset and data cleaning

The dataset is given as two csv files:

- train.csv
- test.csv

The dataset has the following columns:

- id: The id of a training set question pair
- qid1, qid2: Unique ids of each question. Only in training csv.
- question1, question2: The full text of each question
- is_duplicate: The target variable, 1 if question1 and question2 have essentially the same meaning, 0 otherwise. Obviously, only in training file.

Training Statistics	
Size of csv file	64 MB
Total number of question pairs	404290
Total number of questions	537933
Questions appearing multiple times	111780
Positive (Duplicate) pairs	36.92%

Total number of question pairs for testing are 2345796. The training dataset has a total of 255045 non-duplicate and 149306 duplicate pairs. This shows a noticeable degree of imbalance between the positive and negative instances. However, I did not work along the directions of oversampling to have a balance between positive and negative instances because this bias is reflective of the reality as well, i.e., there are fewer duplicate questions as well. The average length of the question is 59 characters with a standard deviation of 32.

I removed all the questions that had fewer than 8 characters, as a vast majority of them did not make any meaningful sense. Moreover, I also stripped the sentences of stop words using the nltk [1] stopwords corpus.

3 Approaches

3.1 String Matching

I started off with the simplest approach of pure string matching, i.e., take a pair of questions, apply a distance measure like the ones we studied in class and based on a certain threshold, determine the value for target variable. Since I used the string matching techniques as a preliminary approach, I just tested it on the training dataset and just calculated the accuracy on it. I used the following 3 distance measures:

Cosine Similarity

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space that measures the cosine of the angle between them [2]. Cosine similarity is a very popular method for measuring similarity for text. One of the reasons for the popularity is that it is very efficient to evaluate, especially for sparse vectors, as only the non-zero dimensions need to be considered.

$$cos(\theta) = \frac{A \cdot B}{||A|| * ||B||} \tag{1}$$

Cosine similarity gave an accuracy of 66% on the training dataset.

Jaccard Index

The Jaccard index is also known as intersection over union [3]. It is a very popular statistic used for comparing the similarity and diversity of sample sets. dg

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{2}$$

The Jaccard distance also returned a very comparable similarity to the Cosine method at 65%.

Sørensen–Dice coefficient

The Sørensen–Dice index is another measure comparing the similarity, given as:

$$QS = \frac{2|X \cap Y|}{|X| + |Y|} \tag{3}$$

The accuracy obtained by Sørensen–Dice coefficient was 62%.

Effect of TF-IDF

Term Frequency-Inverse Document Frequency (or TFIDF for short) is a very popular technique for text analysis. The basic idea of TFIDF is to identify words from a given corpus that are more helpful in discriminating between strings. It is a product of two parts: Term frequency and Inverse Document Frequency. In the term frequency, the idea is to count the number of words and use this count as

[1]http://www.nltk.org/
[2]https://en.wikipedia.org/wiki/Cosine_similarity
[3]https://en.wikipedia.org/wiki/Jaccard_index

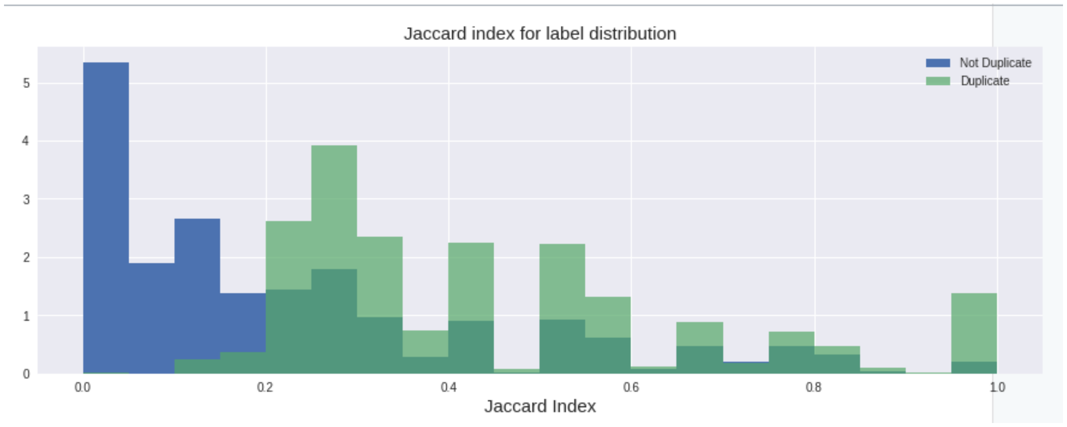

Figure 1: Jaccard Index

additional weight for the word. However, the issue with this approach is some words (like stop words, etc.) would have the highest count but they are not as useful since they are a part of almost every sentence. To circumvent that problem, we also get the document frequency (in whole corpus) of those words and divide the term frequency by it. That way, *only* those words get the highest weightage that are repeated but only in some sentences and not as frequent as stop words.

I tried this approach to see how much this can effect the general distribution of target variable. The results are interesting as can be seen in the graph below. As compared to Figure 1, the results are much better with TFIDF as shown in Figure 2 below.

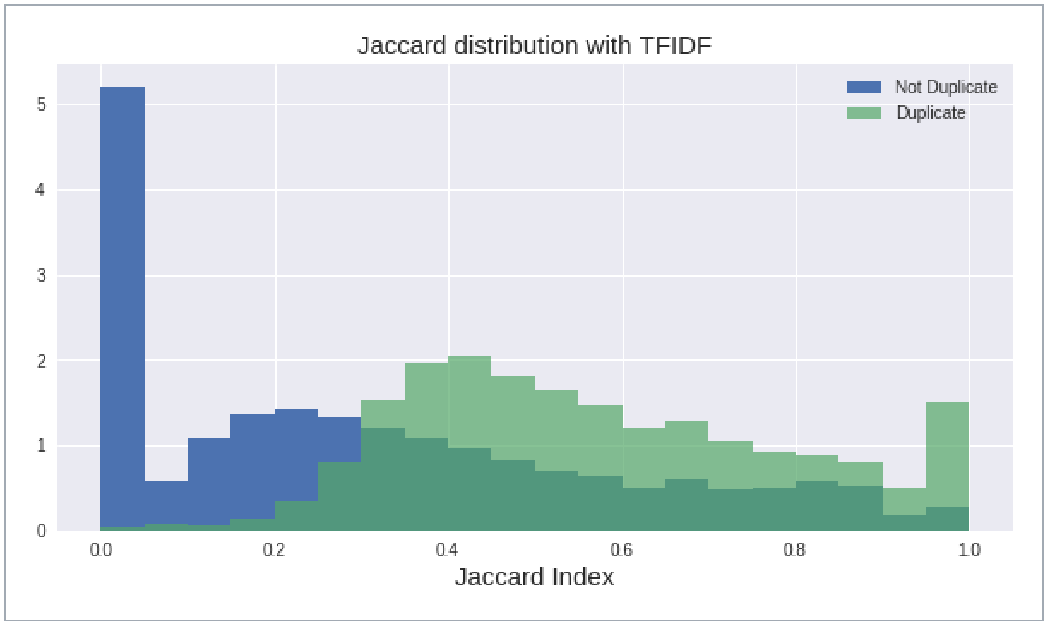

Figure 2: Jaccard Index with TFIDF

3.2 Ensemble Methods

Ensemble methods use multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. An ensemble is itself a supervised learning algorithm, because it can be trained and then used to make predictions, representing a single hypothesis. This hypothesis, however, is not necessarily contained within the hypothesis space of the models from which it is built. Thus, ensembles can be shown to have more flexibility in the functions they can represent. This flexibility can, in theory, enable them to over-fit the training data more than a single model would, but in practice, they have proven to be very useful for machine learning tasks. Though, there are different types of ensemble methods, I primarily focused on two most common methods - namely Random Forest and Gradient Boosting which fall under the domain of bagging and boosting respectively.

Both of these models rely heavily on the use of Decision Trees and in majority of the cases outperforms classic decision trees in precision as well as accuracy.

3.2.1 Random Forest

Random Forests operate by constructing a multitude of decision trees at training time and outputting the class by weighing the contributions of the individual trees in some way. Random decision forests correct for decision trees' habit of overfitting to their training set. This stochasticity is one of the reasons why it is not necessary to be able to get the same results after retraining the data. All the subtrees are independent and are created on a subset of training data (number of data points as well as number of attributes).

I implemented Random Forest on the Quora model and got an accuracy score of 75% which is substantially better than the simple string matching techniques mentioned above. The parameters of the tree depth were kept consistent to the default values.

3.2.2 Gradient Boosting

Gradient boosting is another ensemble method that unlike bagging, works in a stage-wise fashion where early learners operate on simple models of the data and then the hard-example (examples that don't fit well with simple models) are learned by later learners that tend to be more complex. Finally, this ensemble of weak learners is joined together by some weighting technique to give a final classification or regression output.

Gradient Boosting with XGBoost

XGBoost is short for "Extreme Gradient Boosting" [4]. Over the recent years, it has quickly become one of the most efficient and accurate methods for supervised learning problems. It has successfully been used for different problems, such as regression, classification, ordering, etc. Gradient boosting works on the idea of coming up with a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

XGBoost employs tree ensembles as its underlying model. Tree ensemble is a set of classification and regression trees (CART). As a combination of both, they give us a more powerful model as compared to simple decision trees. See the example below in Figure 3 [5].

Here, we try to classify the members of a family into different leaves, and assign them the score on the corresponding leaf. In CART, a real score is associated with each of the leaves. Moreover, instead of using a single tree, a *combination* of trees is used and hence it is called the ensemble model. Random forest is another famous type of ensemble technique, however in a lot of recent competitions, XGBoost seems to be better at supervised learning problems.

XGBoost has a number of parameters like learning rate, maximum depth, maximum leaf nodes, etc. Following are some of the results of varying the maximum depth while keeping the learning rate constant at 0.2. I have also embedded the results that I got with the raw data (i.e., without preprocessing). Though very small, but clearly we can see a little difference.

[4]http://xgboost.readthedocs.io/en/latest/model.html
[5]http://xgboost.readthedocs.io/en/latest/model.html

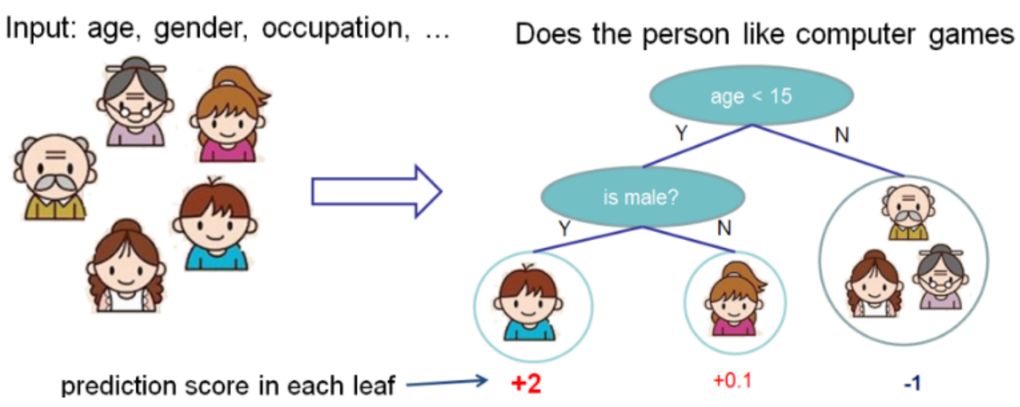
Input: age, gender, occupation, ...
Does the person like computer games
age < 15
Y
N
is male?
Y
N
prediction score in each leaf
+2
+0.1
-1

Figure 3: Example of tree ensemble from XGBoost documentation

Depth	Accuracy Score (TFIDF)	Accuracy Score (Raw)
4	81.6	81.1
5	82.1	81.0
6	82.5	82.2
7	83.2	82.5

3.3 Neural Networks - Siamese Networks

Neural networks are perhaps on the most talked about techniques that have come to light recently. Though not that new, new faster machines in the form of GPUs, availability of significant amounts of data and outpouring of research in the area, the technique has just recently begun to show its promise. They have been successfully used to do all types of machine learning tasks. Often criticized for their *lack* of process in building where most of the times, the user is just playing with different architectures and tuning the hyperparameters, it is essentially a black box technique that can deliver excellent results but without an interpretation of the results. For example, in the case of decision trees, one can follow different paths and easily transform them into rules whereas that is very difficult with neural networks. Despite all the criticism, they are the driving force between the significant leaps in the areas of image classification, segmentation, autonomous driving, speech recognition, etc. Based on the way how they are structured, they come in different variants like Convolutional Neural Networks and use varying depth, referred to as deep learning.

Siamese networks are a class of neural network architectures that are used for finding similarities (or a relationship) between two different entities. They contain two or more identical subnetworks. Identical here means they have the same configuration with the same parameters and weights. Parameter updating during backpropagation is mirrored across both subnetworks.

They are popular among tasks that involve finding similarity or finding relationship between two comparable things. They were proposed initially for signature verification where the task was to determine whether two signatures are from the same person. Since then, they have evolved into other application areas as well. Nowadays, paraphrase matching is one of their major application areas which is the main goal behind this problem as well.

The idea behind Siamese Networks is that two identifical subnetworks are used to process the two inputs (two questions in our case) and another module will take their outputs and produce the final output. This other module will be responsible to calculate the distance between the two sentences and based on that distance, figure out if the two are same or not. Following is an image in Figure 4 from the original Siamese Networks paper [6].

Some recent work have sued Siamese networks to score relevance between a question and an answer candidate. So one input is a question sentence, the other input is an answer, and the output is how relevant is the answer to the question. Questions and answers may not look exactly the same, but if

[6] Signature Verification using a "Siamese" Time Delay Neural Network

the goal is to extract semantic similarity between them, a Siamese network has been shown to work too.

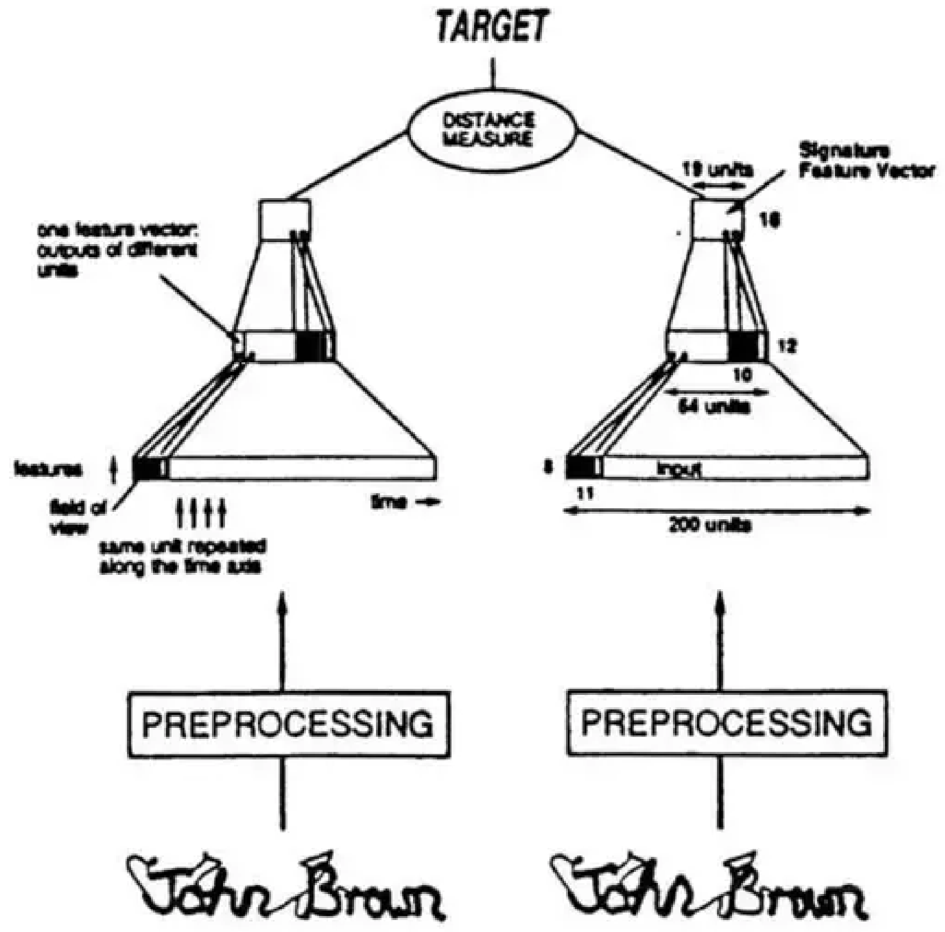

Figure 4: Siamese Network for signature verification

My approach to implement Siamese networks consisted of two main steps:

- Transform the input sentences into vector forms
- Use the pairs of transformed questions (i.e., vectors) to train the network

Word Embeddings

One approach to transform the string input into a vector for the neural network is to use a Word2Vec conversion. There are numerous algorithms to produce such word embeddings. Word2Vec takes as its input a large corpus of text and produces a vector space, typically of several hundred dimensions, with each unique word in the corpus being assigned a corresponding vector in the space. Word vectors are positioned in the space such that words that share common contexts/meanings in the corpus are located in closely to one another [7].

Word2Vec can utilize either of the following two model architectures to produce a representation:

[7]https://en.wikipedia.org/wiki/Word2vec

- Continuous Bag-of-Words (CBOW): In CBOW, model predicts the current word from a window of surrounding context words.
- Skip-Gram: In Skip-Gram, distance is also factored in and the nearby words are weighed more heavily than the distant context words, making them slower but gives better results in real life.

There are two famous algorithms for Word2Vec, namely GLOVE and Google's network architecture. For this project, I relied on GLOVE due to its widespread adoption and plenty of tutorials online. After acquiring these word vectors for each of the questions, I took a mean of all word vectors of both the questions to represent them as a single entity.

Network Configuration

I was only able to run very small neural networks because of the hardware constraints and consequently, didn't get very good results with the accuracy maxing out at 65%. The pipeline was very simple with 3 dense layers interspersed with a dropout of 0.5 and relu activation. By adding more layers, the number of parameters were just too much for my machine to handle so I could not explore it any further.

Acknowledgments

Different resources from books, online were consulted to put together the code and report. Majority have been referenced.

4 Conclusion

Finding relationships between two entities is a very common problem and has far reaching implications across a broad range of subjects. In this small study, I tried to address the specific problem of finding duplicity between a pair of questions. Quora dataset was used that has been distributed via Kaggle. I tried three major techniques consisting of string matching, gradient boosting and neural networks. Results with string matching were not as great but still better than random (i.e., 50% since its a 2 class problem). With all the different distance measures, I consistently got in between 62-66%. Gradient boosting got me a much better accuracy of 82%. I also tried numerous variations of small Siamese networks but unfortunately ran into the expected issue of significant hardware requirements and consequently, was unable to get any meaningful results out of it. Concluding, based on my experiments, XGBoost seems to be the clear winner in terms of result accuracy as well as efficiency to give excellent results. Some other things that I may have tried were more preprocessing of the data before feeding it to gradient boosting technique.