NLP - Quora challenge report

Artur Xarles Esparraguera
Enric Azuara Olivera

Master in Fundamental Principles of Data Science
Universitat de Barcelona
2022

Contents

1 Purpose of the Task

Quora is a platform to share and gain knowledge about almost anything. You can ask questions and connect with people who contribute answering with unique answers. Multiple questions with the same purpose are asked all the time, which can cause some drawbacks when trying to find the best answer or the exact question you are seeking for.

On this Task, we are using a dataset which contains pairs of questions and a label that indicates if they are duplicate or not. On this report we will show all the experiments done in order to predict when a pair of questions are asking the same or not.

2 Description of the experiments

At first, we had a simple solution implementation which was using countvectorizer and logistic model. We will explore a bit this solution and will play a bit with it to see how it affects the solution.

After that, we will generate other features and will explore other techniques trying to improve the initial result that we get from the method initially given.

2.1 Simple solution implementation

We will answer 3 questions on this regard.

1. **What problems/limitations do you think the model has?**

   First we want to state that the model performs pretty good given the simplicity of it. The result is decent given that we only applied countvectorizer and a logistic model without testing any parameters or adding other features. So the principal limitation is its simplicity, which we could view it as an advantage from another perspective.

2. **What type of errors do you get ?**

   We mainly get errors when we have cases where a pair of questions are pretty similar and it just changes few words that contains key information on the question itself. For instance, it is not the same asking "How did you get that money" or "When did you get that money". This pair of questions, in terms of similarity, we could consider them quite equal, but the key word "How" and "When" dictates the type of question we are doing. This are the main type of errors of false positives.

   The other type of errors we get are the ones that we predict are not equal but in reality they are (False negatives). For instance, one this pair of questions "What is the best civil engineering company for a job?" and "What are the best companies for a civil engineer?" we predict that they are not equal. They key point here is that there are few exact similar words, so it is hard for the model to understand that we are asking the same here.

3. **What type of features can you build to improve the basic naive solution?**

   To improve the basic naive solution we can implement is TF-IDF, which is another way to capture the times a word appears into a question, but in this case we are giving more weight to the words that appear fewer times. Another solution we can implement is lemmatization and stemming, in order to help our model to identify similar words which are not written in the same way.

2.2 Experiments and work implementation to improve the results

After trying the first implementation, we defined a set of experiments to try to improve the results. First of all we used different techniques to extract feature vectores from each of the questions. These feature vectors are the following ones:

- **Count Vectorizer (CV)**: The feature vector corresponds to an array where each element of the array represents a word. The value of each element is the number of times that the word appears in the question.
- **TF-IDF**: The feature vector corresponds to an array where each element of the array represents a word. The value of each element is the product between the number of times that the word appears in the question and a weight for each of the words. This weight can be computed as $log(\frac{|X|}{1+|X_w|})$, where $|X|$ is the total number of documents (in this case questions) and $|X_w|$ the number of documents where the word appear. In this way, we will give more weight to words that appear only in a few documents (questions).
- **Extra features**: We added some extra features in order to provide more information to the model about the similarity or not of the questions. Those features are:
  - Jaccard simmilarity: $1 - \frac{q1 \cap q2}{q1 \cup q2}$. A value between 0 and 1 where the lower it is, the more similar are question1 and question2.
  - Sorensen distance: $1 - \frac{2(q1 \cap q2)}{q1+q2}$. A value between 0 and 1 where the lower it is, the more similar are question1 and question2.
  - Feature that indicates the amount of words that are present in both questions
  - Feature that indicates the ratio of words that are present in both questions respect question 1.
  - Features that indicate the number of words of the questions and the count of unique words.
  - Features that indicate the number of digits of the questions.
  - Feature that detects if a keyword is repeated among the pair of questions. Keywords are "where", "when", "why", among others.

Through these methods we get a feature vector that represent each of the questions. All the methods are implemented from scratch and are available in the `utils.py` file. Moreover, Count Vectorizer and TF-IDF incorporate the option of applying lemmatization and stemming, techniques that will be used in some of the following experiments.

Once we have a feature vector for each question, in each of the observations where we will compare two questions we will have two feature vectors. We applied different strategies to combine them before using the features for a classifier. These strategies are the following ones, where X_1 and X_2 represent the feature vectors for question 1 and question 2 respectively:

- **Horizontal stack**: Stack both features vectors in one horizontally.
- **Absolute difference**: element-wise absolute difference between both vectors ($|X_1 - X_2|$). We get a vector with the same dimensionality as a feature vector for one question.
- **Cosine similarity**: cosine similarity between the two feature vectors ($cos(X_1, X_2)$).
- **Product between different values**: apply $-1 \times (X_1 != X_2) + X_1 \odot X_2$ to the feature vectors. It returns a 0 for elements of the vector that represent words that do not appear in any of the questions, a -1 if a word only appears in one question and a value bigger than 0 if the word appear in both questions.

Once the different feature vectors and merging strategies have been proposed, we will analyze the performance of different combinations using a simple logistic regression model trained in a part of the dataset and evaluated in another one.

At first, using the count vectorizer as feature vectors for the questions, we can see in Figure 1 the performance on terms of accuracy and logloss for the different merging strategies proposed.

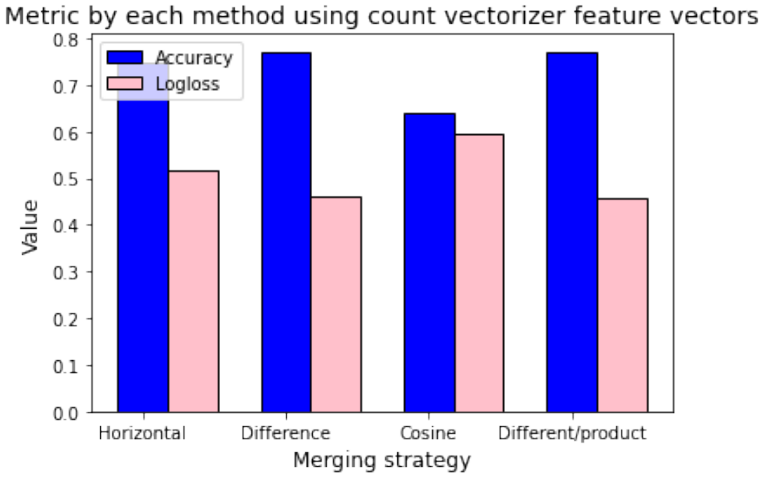

Figure 1: Performance of the model using Count vectorizer feature vectors and the different merging strategies.

It can be seen that the strategies that give better results are the difference of feature vectors and the product between different values, which give similar results. If we perform the same analysis with the TF-IDF feature vectors the results are the ones shown in Figure 2.

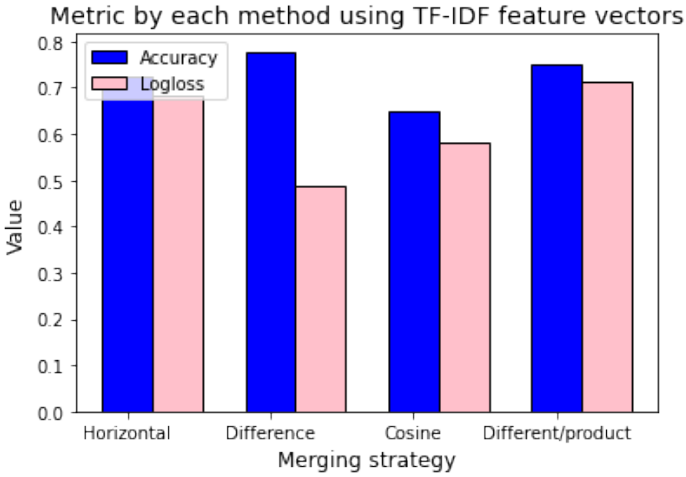

Figure 2: Performance of the model using TF-IDF feature vectors and the different merging strategies.

In this case the best performance is using the absolute difference as merging strategy which achieves an accuracy of 0.78 and a logloss of 0.49, which is a bit better in terms of accuracy that the results with Count Vectorizer features.

As we have seen, both feature vectors, Count Vectorizer and TF-IDF, give similar results and the best strategy to merge the features of both questions is the absolute difference between them. Therefore, in following experiments we used both feature vectors (CV and TF-IDF) and

implemented the absolute difference merging strategy to train the classifier. The cosine similarity, as it only contains one additional value, has also been added to the final vector. In table 1 we can see the results of this experiment. Moreover, there are also the results of applying the same method but adding lemmatization and stemming when computing the feature vectors.

Method	Accuracy	Logloss
CV + TF-IDF	0.7825	0.4782
CV + TF-IDF + lemmatization	0.7740	0.4765
CV + TF-IDF + stemming	0.7616	0.5303

Table 1: Performance using CV and TF-IDF and trying lemmatization and stemming

Regarding the results in 1, lemmatization and stemming do not help in predicting if both questions are or are not the same, as it gives worse results than the original method in terms of accuracy. Therefore, we will no use them in following experiments.

Finally we will add the previously mentioned extra features to the best model obtained until now. Therefore, following experiment will have as input the absolute difference and the cosine similarity between CV and TF-IDF feature vectors, and the set of extra features. This experiment get an accuracy of 0.7912 and a logloss of 0.4567 on the validation dataset which is a bit better than previous model. Therefore, the extra features added give to the model a bit of relevant information to find the differences between the two questions.

Once the best input for the model has been established, comparing different feature vectors and merging strategies through logistic regression, we will try to modify a bit the logistic model to check if we can acquire better results. We are going to explore as well other models.

2.3 Finding the best model

After finding the best combination of features, we explored other models as XGBoost and also changing some parameters on the logistic one.

We have done a gridsearch on the xgboost method in order to find the best combination of parameters given our computational constraints. For the logistic model, we changed to penalty loss to l1 to check if we get a better result. The experiments are:

- XGBoost: We constructed a grid of the following parameters to determine the best possible model:
  - Max_depth: It determines the maximum depth of the tree. We fixed this value into 10.
  - n_estimators: Number of trees. We tried with 1200, 1400 and 1600 trees. The best result was obtained with 1400 trees.
  - learning_rate: How fast the algorithm learns. The best learning rate on the grid search was 0.1.

  After fitting all the models and finding the best combination which was mentioned before, we obtained a way better result in terms of accuracy and log loss. In fact, we got on the validation test an accuracy of 0.809 and a logloss of 0.3869, which is a substantial improvement of what we had with the logistic model.
- Logistic: We experimented changing the loss between the l1 and the l2. After fitting the model with both losses, we determine that l2 penalty gives us a better result.

So, after the previously mentioned experiments, we determined that xgboost was a better method than the logistic one for classifying this task.

2.4 AUC-ROC curve

After finding the best model, we evaluated it with the AUC-ROC curve. The area under a receiver operating characteristic (ROC) curve is a single scalar value that measures the overall performance of a binary classifier with different thresholds. We can observe it at Figure 3.

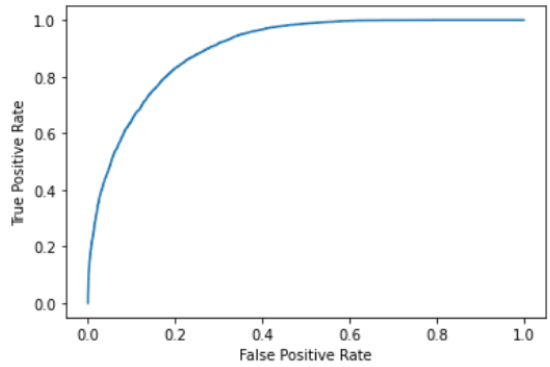

Figure 3: AUC-ROC curve for our final model.

The area under the curve is 0.899, which is a really good score. This means that our model has a good discrimination ability when determining if a pair of questions is equal or not.

3 Final model

After all the experiments, we determine that the best model is gradient boosting, using the absolute difference and the cosine similarity between CV and TF-IDF feature vectors and the list of extra features named previously at section 2.2. The score on the test set is 0.8136, which is a good result that improves the initial proposal given by a good amount (from 0.74 accuracy and 0.52 logloss to 0.813 and 0.38 respectively).