UNIVERSITAT DE BARCELONA

MsC Fundamental Principles of Data Science
Natural Language Processing
Delivery 1: Quora Challange

Universitat de Barcelona

Flàvia Ferrús,
Gerard Castro,
Clàudia Herron,
Pol Riba

April 26, 2023

Abstract

This project was based on generating a baseline solution and an improved solution for the Quora Challenge in order to apply the knowledge acquired throughout the Natural Language Processing course of the MsC in Data Science, UB. The structure of this project was based on constructing text preprocessing functions, generating features from the question pairs, and developing an optimal classification model that minimises the prediction error.

Contents

1 Introduction

Quora is a platform created in 2009, where users ask and answer answers to other users for free. A lot of the questions asked by users in Quora are repeated, as more than 100 million people visit Quora every month. Having different structured questions aiming for the same response can be misleading and time wasting for users. The objective of this project is to create an algorithm which identifies duplicate questions by using Natural Language Processing tools. This will make users' Quora experience much more valuable and will improve the quality and usability of the platform. To do so, we will first implement a basic model that solves the fundamental issues. Then we will explain what are the missing features and soft spots that this basic model has.
Once we have a fundamental solution, we will try to improve it by adding more advanced features which can help us to improve the accuracy metrics while minimising the error rate.

2 Data Available

We have two datasets available, a training and a testing dataset.
Both datasets have the same features, which are the following:

- Id: The id of a training set question pair.
- qid1, qid2: Unique ids of each question (questions are compared by pairs).
- question1, question2: Full text of each question.
- `is_duplicate`: A binary variable which indicates if a question pair is duplicate (1) or not (0).

For the test set, we will obviously delete the `is_duplicate` variable as it is our target variable. It is also important to comment that we split the data in 3 data sets:

1. 5 % from initial dataset: Testing set
2. 95 % from the remaining dataset when subtracting the Test set:: Training set.
3. 5 % from the remaining dataset when subtracting the Test set: Validation set

3 First Approach

3.1 Question Preprocessing

We first pre-process the questions by applying the following steps to the questions:

1. Lower Case: We lowercase all the words to avoid differences among exact words.

3.2 Feature Generator

We will use the count vectorizer for this first attempt. It's quite a basic approach but works well most of the times:

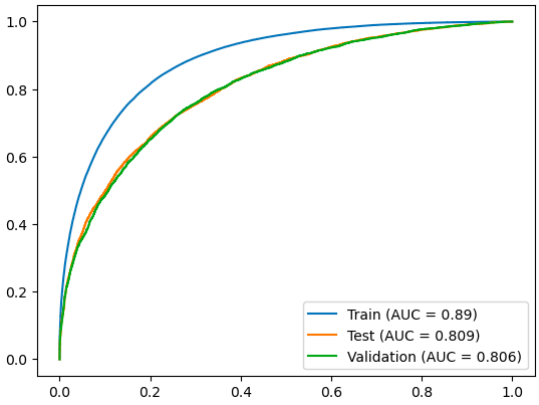
1. Count Vectorizer: A method for converting text documents into a matrix of token counts. Each row represents a document and each column represents a token in the vocabulary. We horizontally stack both vectors.

3.3 Classifier: Logistic Regression

We now implement a basic logistic regression .
The simple model we have implemented produces a training AUC of 0.890, validation 0.806 and testing 0.809. Note that the metrics obtained can be improved, as we may show on the subsequent sections, but are quite acceptable for a basic model. The obtained ROC plot is represented on Figure 1, as well as the obtained metrics.

```
VALIDATION results:
              precision    recall  f1-score   support

           0       0.78      0.83      0.81      9728
           1       0.67      0.61      0.64      5635

    accuracy                           0.75     15363
   macro avg       0.73      0.72      0.72     15363
weighted avg       0.74      0.75      0.75     15363

TESTING results:
              precision    recall  f1-score   support

           0       0.78      0.84      0.81     10178
           1       0.69      0.61      0.65      5994

    accuracy                           0.75     16172
   macro avg       0.74      0.72      0.73     16172
weighted avg       0.75      0.75      0.75     16172

Training AUC: 0.8899632185735915
Valitaion AUC: 0.8063220758125905
Testing AUC: 0.8089206813415891
```

(a) Validation and testing results



(b) ROC curve

Figure 1: Results from Simple solution.

3.4 Questions

3.4.1 Problems/limitations on the model

Node that this model has a limited accuracy and AUC results. This fact could be doe to the fact that we have a limited set of features to train the model. Therefore, there is some information we are not taking into account, other than the text vectorizer after a basic pre-processing. Therefore, some more complex aspects we can derive from the data, such as considering multiple text vectorizers and stacking them to get a broader representation of the question itself, builiding similarity functions using different ways of computing distances, or some relevant features determining whether the question starts with the same word or either contain similar worlds.

Moreover, on this basic pipeline, no different classification models are used, and this could be solved by using a grid in order to contrast different models

3.4.2 Errors

Note that the metrics obtained can be improved, as we may show on the subsequent sections, but are quite acceptable for a basic model. However, we saw some errors regarding the lack of information, gathered from the dataset. For example, we can see that there are some questions classified as different when their actual target value is equal. Therefore, introducing some extra features taking into account these and other multiple intuitive features.

3.4.3 Extra features

As aforementioned, a way of improving the basic solution may be considering some complex extra features. As commented before, some ideas that may be interesting to consider woul

be:

1. Building similarity functions using different ways of computing distances. Among different distances we have thought of implementing the cosine similarity, jaccard distance, levenshein distance, Mover Word's distance, or even some more complex distances taking into account inside distances between words.

2. Developing special features, that could bring extra information from the dataset, such as whether the question starts with the same word or either contain similar worlds.

3. Further studies, including syntactical studies on the questions in order to compute similarity

The features we actually implemented are commented and specified on subsequent sections. The simple model we have implemented produces a training AUC of 0.919, and testing 0.859. Note that the metrics obtained can be improved, as we may show on the subsequent sections, but are quite acceptable for a basic model.

4 Improved approach

To improve the previous model, we have tried different things. To do this in a more efficient way, we have used the **sklearn pipeline**, which is a powerful tool for automating machine learning workflows. It allows us to chain together multiple steps in a machine learning workflow, such as data pre-processing, feature engineering, and model training, into a single object that can be fit and used to make predictions.

4.1 Question pre-processing

We tried the following preprocessing tools:

1. Remove Stop Words: remove very common words. We customize the stop words we want to use.

2. Remove Punctuations: We remove all the punctuation signs.

3. Lower Case: We lowercase all the words to avoid differences among exact words.

4. Stemming: The process of reducing words to their base or root form, in order to capture their essential meaning and improve text analysis. This is done by removing suffixes and prefixes from words, to create a shorter, more standardized representation.

5. British: If there are words that change from British English to American or other English variants. An example would be "color" and "colour".

We also removed all the NaN questions, because it makes no sense to compare a question with an empty one

4.2 Feature Generator

We tried the following methods/techniques to weight the importance of terms:

1. Count Vectorizer: A method for converting text documents into a matrix of token counts. Each row represents a document and each column represents a token in the vocabulary.

2. Count Vectorizer 2w: A modified version of the Count Vectorizer that considers pairs of consecutive words as tokens. This can capture more context and improve text analysis.

3. TF-IDF: Term Frequency-Inverse Document Frequency is a technique used to weigh the importance of each token in a document based on how frequently it appears in that document and across all documents in a corpus.

4. TF-IDF-2W: A modified version of TF-IDF that considers pairs of consecutive words as tokens. This can capture more context and improve text analysis.

5. Spacy small: A small pre-trained language model from the Spacy library that can perform various NLP tasks such as tokenization, POS tagging, and named entity recognition.

6. Spacy medium: A medium-sized pre-trained language model from the Spacy library that can perform more advanced NLP tasks such as dependency parsing, text classification, and text similarity analysis.

7. Coincident Ratio: We also created a feature that returns the ratio of words that coincide between the pair of questions. This feature has been computed taking into account the total number of words from both questions we are comparing.

Some **extra features** we have also considered on the generation of the training set in order to get better results and so we take into account different information derived from the original data set. The most relevant extra features:

- Get coincident keywords: feature that compares the intersection of the set of key words, this is, key question words, such as What, Where, Which... with each question among the two strings compared. This feature was designed taking into account that similar questions would start with the same question keyword. It returns 1 when they start or contain (they would mostly contain it at the beginning as intended) with the same word, and 0 if not.

- Jaccard distance: feature that computes the Jaccard distance between two strings. It is based on computing the ratio between the length of the intersection set of words from each question string over the length of the union set of words between both questions.

- Levenshtein distance: feature that computes the Levenshtein distance between the two given questions. 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. In this case, we may consider the implementation of two variations

    1. By words: we adapt the Levenshtein algorithm to considering the number of words we would need to change in order to get the exact similar sentence in both input questions.
    2. Regular Levenshtein distance but considering the whole question, and studying letter by letter each character. This feature has problems of memory capacity and it tends to outload our computers. Therefore, the more optimal application of the Levenshtein distance is the custom distance computed for considering the word comparison.

For the **metrics** to compare the distances or similarity between questions we used the following:

- Cosine similarity: A measure of similarity between two non-zero vectors of an inner product space. It measures the cosine of the angle between the two vectors and ranges from -1 (completely dissimilar) to 1 (completely similar). It is often used in text analysis to compare the similarity of documents based on their vector representations.
- Absolute difference: A mathematical operation that calculates the difference between two values without regard to their sign. It is the absolute value of the subtraction of one value from another.
- Horizontal stacking: A method of combining two or more arrays or matrices horizontally, such that they are placed side-by-side. This is achieved using NumPy's hstack function, which takes a tuple of arrays or matrices and returns a new array or matrix with the same number of rows and the sum of the columns of the input arrays. Horizontal stacking can be useful for combining feature matrices or label vectors in machine learning applications.

4.3 Models

We tried the following models:

1. AdaBoostClassifier: AdaBoost (short for Adaptive Boosting) is an ensemble learning method that combines multiple "weak" classifiers into a single "strong" classifier. It does this by iteratively adjusting the weights of misclassified samples and re-training the classifier on the updated weights.
2. RandomForestClassifier: Random Forest is also an ensemble learning method that combines multiple decision trees into a single model. Each tree is trained on a random subset of the data, and the final prediction is made by aggregating the predictions of all the individual trees.
3. LinearDiscriminantAnalysis: Linear Discriminant Analysis is a classification method that finds a linear combination of features that best separates the different classes in the data. It assumes that the data is normally distributed and that the covariance matrix is equal for all classes.
4. QuadraticDiscriminantAnalysis: Quadratic Discriminant Analysis is similar to Linear Discriminant Analysis, but it allows for different covariance matrices for each class.
5. BernoulliNB: Bernoulli Naive Bayes is a classification method that assumes that each feature is a binary variable (i.e., takes on values of 0 or 1). It calculates the probability of each class given the observed values of the features.
6. GaussianNB: Gaussian Naive Bayes is a classification method that assumes that each feature follows a Gaussian (normal) distribution. It calculates the probability of each class given the observed values of the features.
7. KNeighborsClassifier: K-Nearest Neighbors is a non-parametric classification method that assigns a label to a data point based on the labels of its k nearest neighbors in the feature space.

8. SVC: Support Vector Machines is a classification method that finds a hyperplane in the feature space that maximally separates the different classes. It can also handle nonlinear boundaries by using a kernel function to map the data into a higher-dimensional feature space.

9. LogisticRegression: Logistic Regression is a classification method that models the probability of each class given the observed values of the features as a logistic function of a linear combination of the features.

10. GradientBoostingClassifier: Gradient Boosting is another ensemble learning method that combines multiple "weak" classifiers into a single "strong" classifier. It does this by iteratively fitting a new classifier to the residual errors of the previous classifier.

11. XGBClassifier: XGBoost is an optimized implementation of Gradient Boosting that uses a number of advanced techniques to improve performance.

12. CatBoostClassifier: CatBoost is another optimized implementation of Gradient Boosting that is particularly well-suited to working with categorical data. It uses several novel techniques to handle high-cardinality categorical variables and reduce overfitting.

As we have a lot of models, it is hard to try different pre-processing tools and feature generators by hand. So, to do it in a better way, we created a grid where we put all the models, pre-processing tools and feature generators. This grid tries all the possibilities and gives us the best combination. As everything in life, the grid has some pros and cons: It's an easy way to get the best result out of all the combinations but it takes very long to run. To solve this problem we decided to try some of the combinations by hand. On the subsequent section we show some results obtained by training different models.

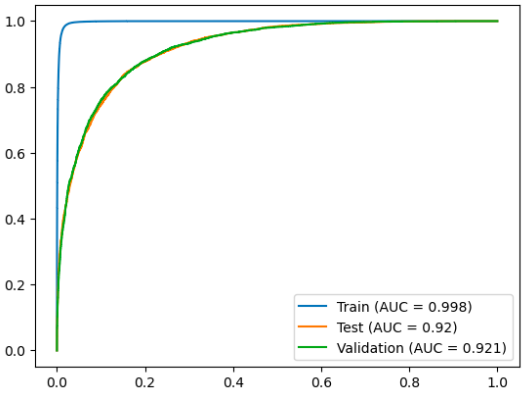
5 Results obtained

Some more complex models are shown with details on the evluation metrics in the following sections. However, multiple simpler models, without pre-processing, or with just one text vectorizer or just one aggregator in order to generate features have also been computed. We attach a folder with the pipelines trained and developed using our project.

The first experiment regarding the improved model we have implemented produces a training AUC of 0.998, validation 0.921 and testing 0.920. This model corresponds to implementing all the pre-processing over the input dataset, computing the text vectorizer using a Count Vectorizer and a TF-IDF with bins of 2 words each. Then the feature generation comes from the aggregation functions, which we apply both the horizontal stacking of both representations in order to generate two sparse matrices as features, and an absolute distance computation, also generating a sparse matrix. Finally, we include all the extra features to aforementioned to the model. The corresponding results are shown on Figure 2.

```
VALIDATION results:
              precision    recall  f1-score   support

           0       0.87      0.89      0.88      9728
           1       0.80      0.78      0.79      5635

    accuracy                           0.85     15363
   macro avg       0.84      0.83      0.84     15363
weighted avg       0.85      0.85      0.85     15363

TESTING results:
              precision    recall  f1-score   support

           0       0.87      0.88      0.88     10178
           1       0.80      0.78      0.79      5994

    accuracy                           0.84     16172
   macro avg       0.83      0.83      0.83     16172
weighted avg       0.84      0.84      0.84     16172

Training AUC: 0.9977428614760944
Valitaion AUC: 0.9210666782445243
Testing AUC: 0.9203519937701506
```

(a) Training and testing results



(b) ROC curve

Figure 2: Results from Improved solution.

6 Conclusions

We aimed to build a solid model in order to create a baseline for solving Natural Language Processing (NLP) problems. In order to do so, we adopted an approach with custom classes and the definition of a pipeline in order to store and organise our different models. Once this basic skeleton was robust and well-structured, we started innovating and trying some new extra features, combining different text vectorizers along with different aggregation function, as well as implementing diverse classifiers. The results did beat our expectations, getting accuracy values around 0.84, and AUC (Area Under Curve) of 0.92, for the test set. Knowing that there is a lot to improve and so many further studies we could conduct, we have developed and improved a basic structure for general problems involving NLP that we could for sure improve and find useful in future studies or classifying problems.

7 Appendix

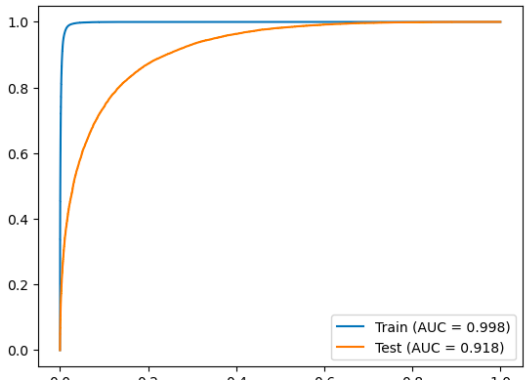
In this sections some other models that we have trained during the computation and implementation of this project are presented. Note that due to the high complexity of this models we did not have time to execute and compile all the presented models to include them in the `model_artifacts` folder along with the improved and simple solution attached on the project. However, note that the obtained results are quite satisfactory when considering the accuracy and the AUC metrics.

7.1 TF-IDF and Count Vectorizer 2 words / Horizontal stacking and Absolute difference/ Logistic Regression / with ALL the preprocessing approaches.

Results are observed on Figures 3.

```
TRAINING results:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99    163274
           1       0.97      0.99      0.98     95469

    accuracy                           0.98    258743
   macro avg       0.98      0.98      0.98    258743
weighted avg       0.98      0.98      0.98    258743

TESTING results:
              precision    recall  f1-score   support

           0       0.87      0.89      0.88     40882
           1       0.80      0.77      0.78     23804

    accuracy                           0.84     64686
   macro avg       0.83      0.83      0.83     64686
weighted avg       0.84      0.84      0.84     64686

Training AUC: 0.9979188715683424
Testing AUC: 0.9180955063517888
```

(a) Training and testing results



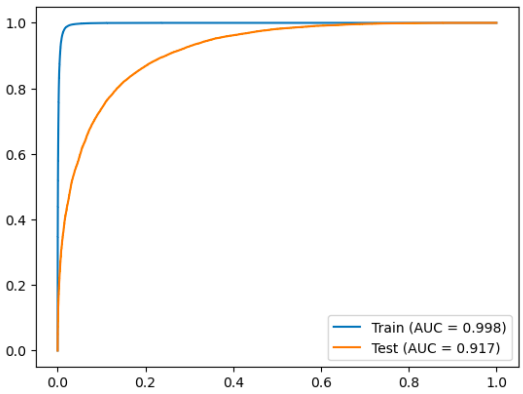
(b) ROC curve

Figure 3: Results from Experiment 1.

Same baseline model is trained, this time including some extra features, such as: `'coincident_ratio'` , `'coincident_keyword', 'jaccard'`
Results are presented on Figures 5.

```
TRAINING results:
              precision    recall  f1-score   support

           0       0.99      0.98      0.99    163274
           1       0.97      0.99      0.98     95469

    accuracy                           0.98    258743
   macro avg       0.98      0.98      0.98    258743
weighted avg       0.98      0.98      0.98    258743

TESTING results:
              precision    recall  f1-score   support

           0       0.87      0.88      0.88     40882
           1       0.80      0.77      0.78     23804

    accuracy                           0.84     64686
   macro avg       0.83      0.83      0.83     64686
weighted avg       0.84      0.84      0.84     64686

Training AUC: 0.9977573310739394
Testing AUC: 0.9168302702506028
```

(a) Training and testing results



(b) ROC curve

Figure 4: Results from Experiment 1 with extra features.

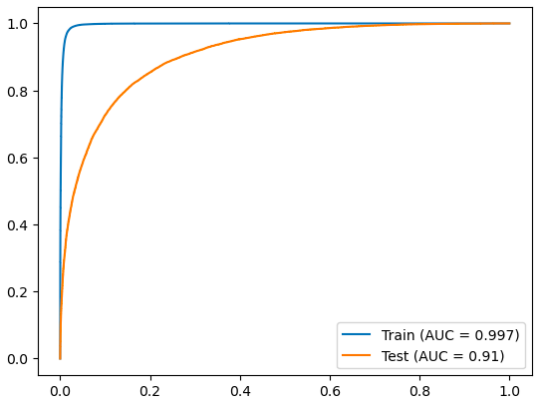
Note that the results are not significantly different from these two models when introducing the extra features into the model. however, note that the results itself, without including the extra features were already satisfactory.

7.2 TF-IDF with 1 word, CV with 2 words / Horizontal stacking and Absolute difference/ Logistic Regression / with ALL the preprocessing approaches.

Now we take an alternative approach, and use the TF-IDF with one single word in order to define the dictionary, and CV with combinations of two words, so we take into account simultaneously the information given by one or two consecutive words. In this model, the extra features included are the `'coincident_keyword', 'levenshtein_w'`.

```
TRAINING results:
              precision    recall  f1-score   support

           0       0.99      0.98      0.98    163274
           1       0.97      0.98      0.97     95469

    accuracy                           0.98    258743
   macro avg       0.98      0.98      0.98    258743
weighted avg       0.98      0.98      0.98    258743

TESTING results:
              precision    recall  f1-score   support

           0       0.86      0.89      0.87     40882
           1       0.80      0.75      0.77     23804

    accuracy                           0.84     64686
   macro avg       0.83      0.82      0.82     64686
weighted avg       0.84      0.84      0.84     64686

Training AUC: 0.9971178890829185
Testing AUC: 0.9096535167207177
```

(a) Training and testing results



(b) ROC curve

Figure 5: Results from Experiment 3.