Universitat de Barcelona - MSc Fundamental Principles of Data Science

Natural Language Processing

Delivery 1: Quora Challenge – 28/04/24

Madison Chester – Dafni Tziakouri

Introduction:

For Delivery 1 of the Natural Language Processing course, our task was to develop a foundational model to tackle the Quora challenge: identifying duplicate questions using NLP techniques. Quora, established in 2009, serves as a platform where users freely pose and respond to inquiries. With over 100 million monthly visitors, the platform sees a considerable volume of repeated queries. This redundancy can lead to confusion and inefficiency for users seeking information.

Our initial objective is to construct a basic model addressing the core issues. Subsequently, we will analyze the model's limitations, identifying areas where enhancements are necessary. By augmenting the model with advanced features, our aim is to enhance accuracy metrics while minimizing error rates.

We have at our disposal two datasets: one for training and one for testing. Both datasets share identical features, which include:

- *Id:* The unique identifier for each question pair in the training set.
- *qid1, qid2:* Unique identifiers assigned to each question, facilitating pairwise comparison.
- *question1, question2:* The full text of each respective question.
- *is_duplicate:* A binary variable indicating whether a question pair is deemed duplicate (1) or not (0).

In the testing set, the is_duplicate variable is naturally excluded since it serves as our target variable. It's noteworthy that we partitioned the data into three distinct sets:

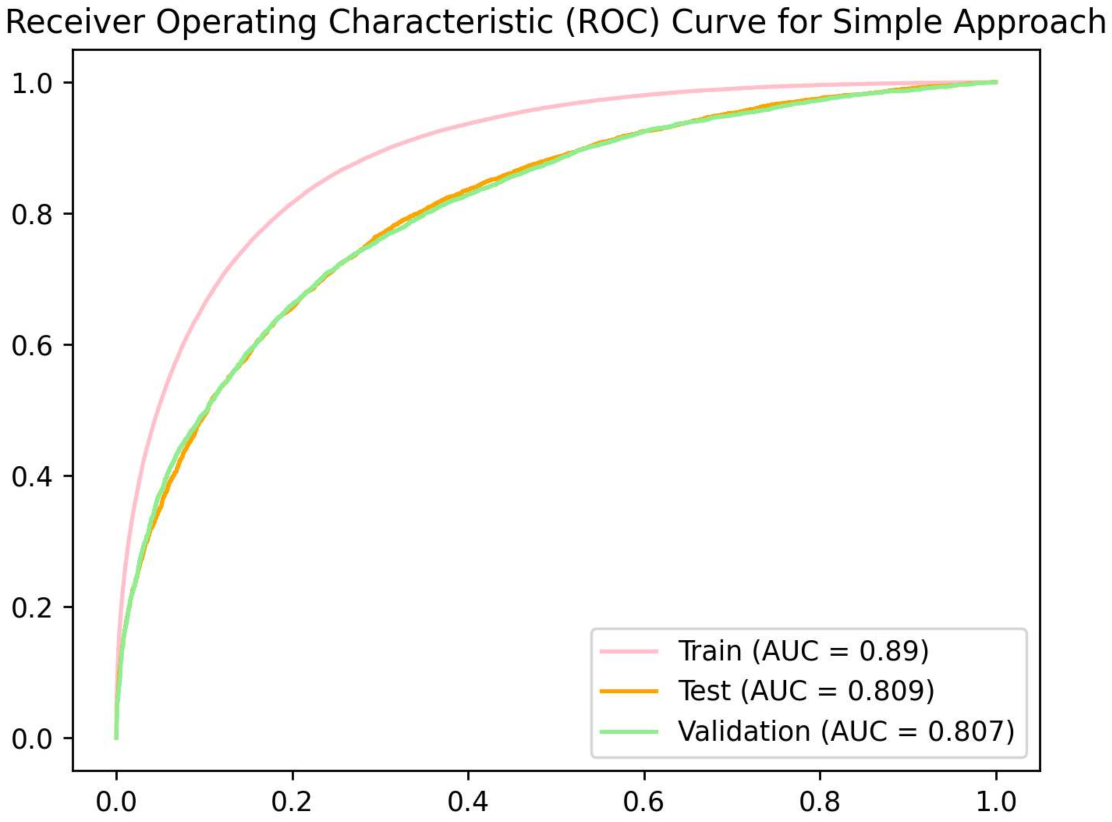
1. *Testing set:* Comprising 5% of the initial dataset, this set is reserved for evaluating model performance.
2. *Training set:* Encompassing 95% of the remaining dataset after deducting the testing set, this set serves as the primary data for model training.
3. *Validation set:* Accounting for the remaining 5% of the dataset after subtracting the testing set, this set aids in fine-tuning model parameters and assessing generalization performance.

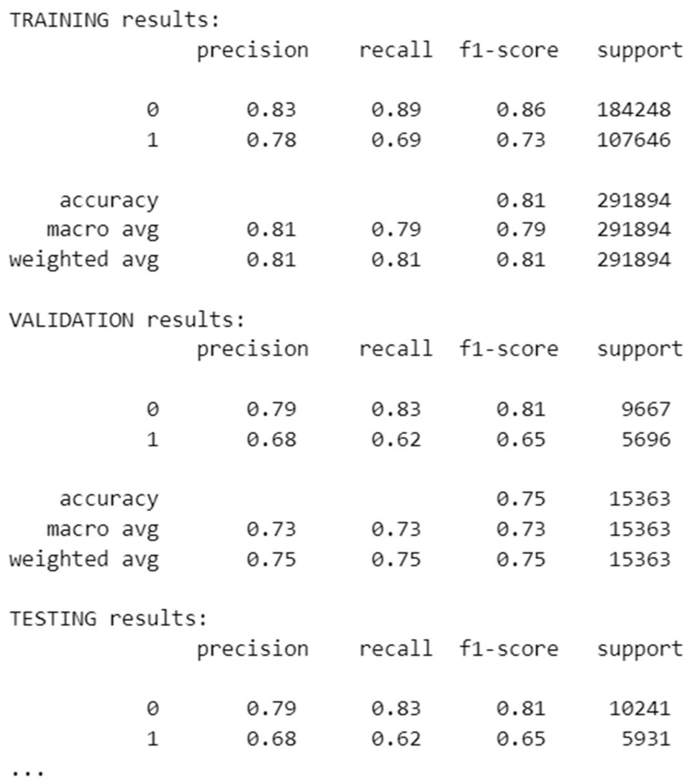
Simple Approach:

We'll begin by preprocessing the questions, ensuring uniformity by converting all words to lowercase. Following this, we'll employ a vectorizer, a fundamental yet effective technique in text processing. The vectorizer functions by transforming text documents into a matrix of token counts. In this matrix, each row corresponds to a document, while each column represents a unique token from the vocabulary. These vectors are then horizontally stacked.

For classification purposes, we'll utilize a basic logistic regression model. Our initial implementation yields promising results: a training AUC of 0.890, validation AUC of 0.807, and testing AUC of 0.809. While these metrics are satisfactory for a basic model, they remain subject to improvement, as we'll explore later on.

Included below is the ROC plot depicting our model's performance, alongside the obtained metrics:





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

           0       0.83      0.89      0.86    184248
           1       0.78      0.69      0.73    107646

    accuracy                           0.81    291894
   macro avg       0.81      0.79      0.79    291894
weighted avg       0.81      0.81      0.81    291894

VALIDATION results:
              precision    recall  f1-score   support

           0       0.79      0.83      0.81      9667
           1       0.68      0.62      0.65      5696

    accuracy                           0.75     15363
   macro avg       0.73      0.73      0.73     15363
weighted avg       0.75      0.75      0.75     15363

TESTING results:
              precision    recall  f1-score   support

           0       0.79      0.83      0.81     10241
           1       0.68      0.62      0.65      5931
...
```

These results may prompt many questions, including:

- What problems/limitations does the model has?
- What type of errors do we get?
- What type of features can we build to improve the basic naive solution?

Problems/limitations of the model: It's worth noting that our model's accuracy and AUC results are somewhat limited. This limitation could stem from the fact that our feature set is constrained, overlooking additional information beyond basic text preprocessing and vectorization.

To address this, we could explore more sophisticated approaches. For instance, we might experiment with multiple text vectorizers and combine them to capture a more comprehensive representation of the questions. Additionally, incorporating various similarity functions that compute distances in distinct ways could enhance our model's performance. We could also consider relevant features, such as whether questions begin with the same word or contain similar phrases, to enrich our feature set.

Furthermore, our basic pipeline currently employs only one classification model. To improve model performance, we could explore a range of classification algorithms. This could be achieved by employing a grid search technique to systematically evaluate and contrast the performance of different models.

Errors: While the metrics achieved are deemed acceptable for a basic model, it's important to acknowledge room for improvement, as will be demonstrated in subsequent sections. However, during our analysis, we observed certain errors attributable to insufficient information utilization from the dataset.

For instance, we noticed discrepancies where questions classified as different actually possess identical target values. This discrepancy suggests the potential value of introducing supplementary features to capture such nuances. These additional features could encompass various intuitive aspects derived from the data, aiding in refining our model's performance.

Additional features: As previously mentioned, enhancing our basic solution involves incorporating more complex features. Here are some ideas we've considered:

- *Similarity functions:* We can explore various distance metrics such as cosine similarity, Jaccard distance, Levenshtein distance, Mover Word's distance, as well as some more intricate distances that consider relationships between words.

- *Unique features:* Introducing features like whether questions start with the same word or contain similar phrases can provide additional insights from the dataset.

- *Deeper studies:* Conducting analyses on the questions to compute similarity can offer deeper understanding and potentially improve model performance.

In the following sections, we'll delve into the features we've implemented and their impact. While these metrics are satisfactory for a basic model, there's room for improvement, as we'll demonstrate in the following sections.

Improved Approach:

To enhance the preceding model, we've pursued various strategies. Streamlining our efforts, we've leveraged the sklearn pipeline, a robust tool for automating machine learning workflows. This pipeline enables us to seamlessly connect multiple stages in our machine learning process—ranging from data pre-processing and feature engineering to model training—into a unified object. This object can then be fitted to the data and utilized for making predictions.

For preprocessing, we experimented with the following techniques:

1. Lower Case: We converted all words to lowercase to ensure consistency and minimize discrepancies.
2. *Remove Stop Words:* We filtered out highly common words, customizing the list of stop words to suit our needs.
3. *Remove Punctuations:* All punctuation marks were eliminated from the text..
4. *Stemming:* Employing this technique, we reduced words to their base or root form, enhancing text analysis by capturing essential meanings. This involved removing suffixes and prefixes to create a standardized representation.
5. *British English Conversion:* Addressing variations in English spelling, such as differences between British English and other variants like American English. For instance, words like "favorite" and "favourite" were standardized.

Additionally, we removed all NaN (not a number) questions from the dataset. These preprocessing steps were crucial for refining the data and preparing it for further analysis.

We also experimented with several weighting methods/techniques:

1. *Count Vectorizer:* This method converts text documents into a matrix of token counts, with each row representing a document and each column representing 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 approach captures more contextual information and enhances text analysis.
3. *TF-IDF (Term Frequency-Inverse Document Frequency*): This technique assigns weights to tokens based on their frequency within a document and across all documents in a corpus, helping to emphasize important terms while downplaying common ones.
4. *TF-IDF-2W:* Similar to TF-IDF, but modified to consider pairs of consecutive words as tokens, enabling a deeper understanding of contextual relationships.
5. *Spacy small:* Utilizing a small pre-trained language model from the Spacy library, capable of tasks such as tokenization, part-of-speech tagging, and named entity recognition.

6. *Spacy medium:* Employing a medium-sized pre-trained language model from the Spacy library, which extends functionality to more advanced NLP tasks like dependency parsing, text classification, and text similarity analysis.
7. *Coincident Ratio:* We developed a feature calculating the ratio of overlapping words between pairs of questions. This ratio considers the total number of words in both questions being compared, providing insight into the similarity of question pairs.

We've enhanced the generation of our training set by incorporating additional features to improve results, leveraging various insights from the original dataset. Here are the key enhancements:

- *Coincident Keywords:* This feature identifies shared keywords, particularly question-initiating words like "What," "Where," or "Which," within the compared questions. Its purpose is to detect similarities in questions that likely share a common starting point. A value of 1 is returned if the questions start or contain the same word (primarily at the beginning), and 0 otherwise.
- *Jaccard Distance:* This feature calculates the Jaccard distance between two strings, measuring the ratio of the intersection set of words in each question to the union set of words between both questions.
- *Levenshtein Distance:* This feature computes the Levenshtein distance between the two questions. Essentially, it quantifies the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one question into the other. Two variations are considered:
    1. By Words: Adapting the Levenshtein algorithm to determine the number of word-level changes necessary to achieve similarity between the input questions.
    2. Regular Levenshtein Distance: Considering the entire question and analyzing each character letter by letter. However, this approach consumes significant memory resources and strains our computational capacity. Thus, the more efficient application of Levenshtein distance is the customized version focusing on word comparison.

To evaluate the distances or similarities between questions, we employed the following metrics:

1. *Cosine Similarity:* This metric measures the similarity between two non-zero vectors in an inner product space by calculating the cosine of the angle between them. It ranges from -1 (completely dissimilar) to 1 (completely similar) and is commonly used in text analysis to compare the similarity of documents based on their vector representations.
2. *Absolute Difference:* This mathematical operation computes the difference between two values without considering their sign, resulting in the absolute value of the subtraction of one value from another.
3. *Horizontal Stacking:* This method involves combining two or more arrays or matrices horizontally, placing them side-by-side. We utilized NumPy's hstack function to achieve this, 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 proves useful for merging feature matrices or label vectors in machine learning applications.

We explored a variety of models to determine the most effective one for our task. Here's a summary of the models we experimented with:

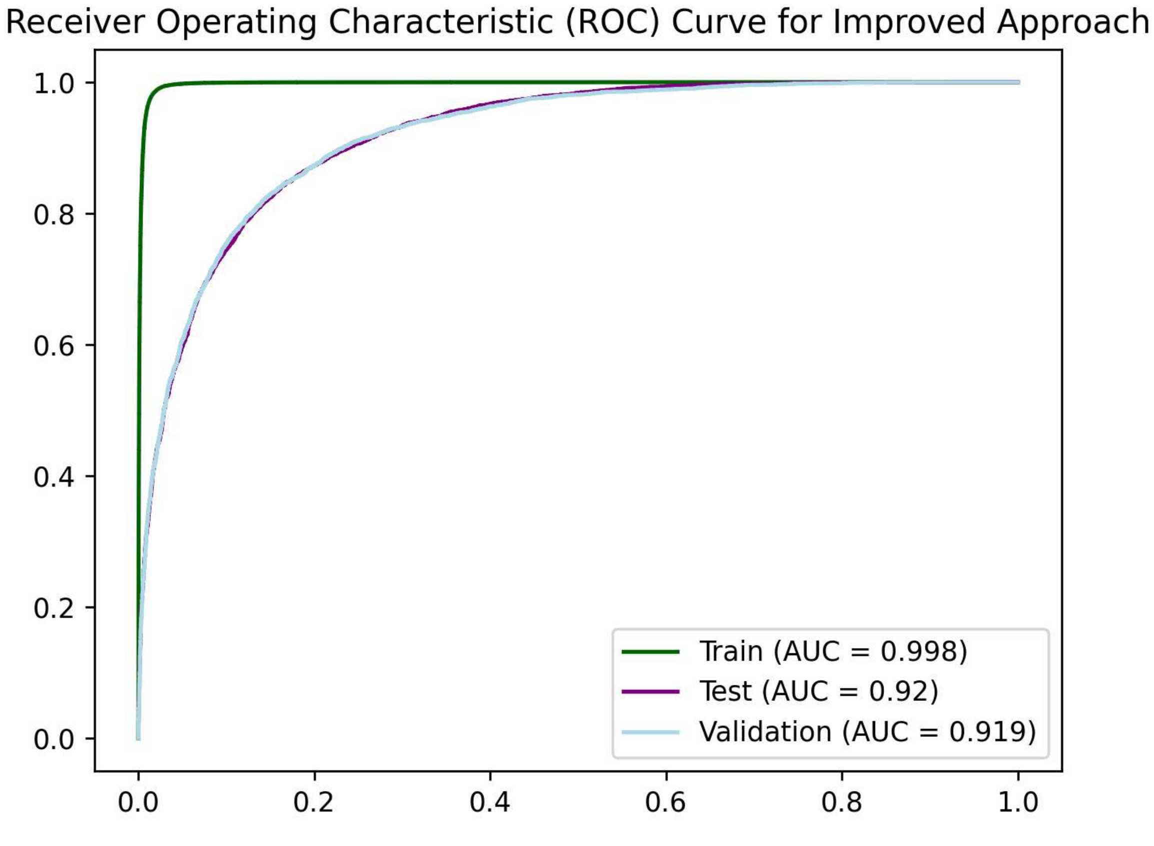
1. *AdaBoostClassifier:* An ensemble learning method that combines multiple "weak" classifiers into a single "strong" classifier by adjusting the weights of misclassified samples iteratively.
2. *RandomForestClassifier:* Another ensemble learning method that aggregates predictions from multiple decision trees trained on random subsets of the data.
3. *LogisticRegression:* Models the probability of each class given observed features using a logistic function of a linear combination of features.
4. *BernoulliNB:* A classification method assuming binary features (0 or 1) and calculates the probability of each class given the observed feature values.
5. *GaussianNB:* Assumes features follow a Gaussian distribution and calculates class probabilities based on observed feature values.
6. *KNeighborsClassifier*: Assigns a label to a data point based on the labels of its k nearest neighbors in the feature space.
7. *LinearDiscriminantAnalysis (LDA):* A classification method that finds a linear combination of features to best separate different classes, assuming a normal distribution and equal covariance matrix for all classes.
8. *QuadraticDiscriminantAnalysis (QDA):* Similar to LDA, but allows for different covariance matrices for each class.
9. *SVC (Support Vector Classifier):* Finds a hyperplane in the feature space to maximize class separation, capable of handling nonlinear boundaries using kernel functions.
10. *GradientBoostingClassifier:* Another ensemble learning method that combines multiple weak classifiers by fitting new classifiers to the residual errors of the previous ones.

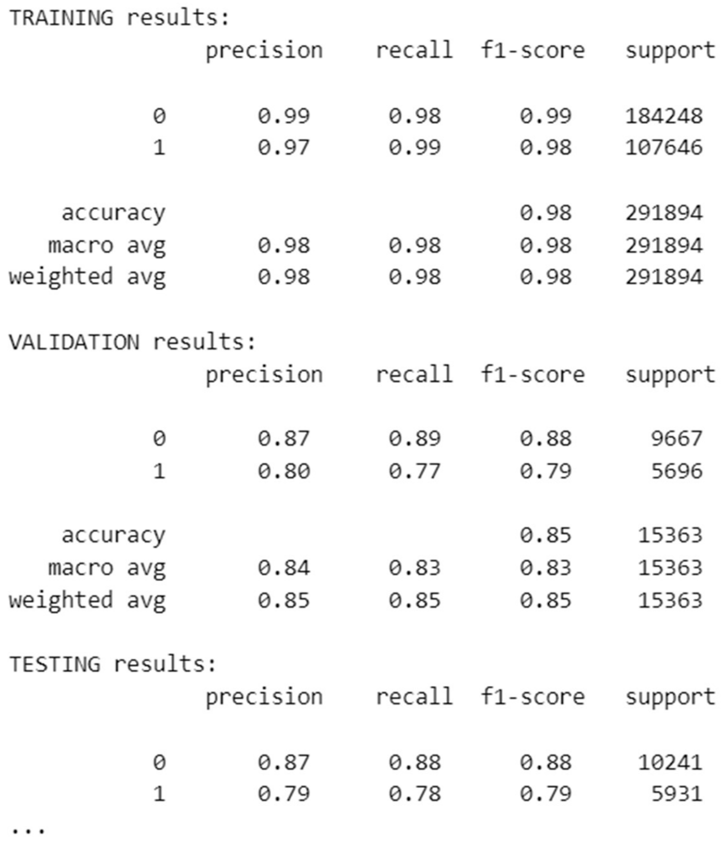
With a multitude of models at our disposal, manually experimenting with various pre-processing tools and feature generators becomes impractical. To streamline this process, we devised a comprehensive grid, mapping out all models, pre-processing techniques, and feature generators. This grid systematically explores all possible combinations, ultimately identifying the optimal configuration. However, like many endeavors, it comes with its own set of advantages and disadvantages. While it efficiently uncovers the best-performing combination, its exhaustive nature translates to extend processing times. To mitigate this challenge, we supplemented automated exploration with manual experimentation.

Below, we present one of our most promising approaches, demonstrating significant improvements. The refined model we've developed exhibits good performance metrics: a training AUC of 0.998, validation AUC of 0.919, and testing AUC of 0.919. This model embodies a comprehensive preprocessing pipeline applied to the input dataset. We employ both Count Vectorizer and TF-IDF techniques, utilizing a bin size of 2 words each to compute text representations.

Furthermore, feature generation is enriched through aggregation functions. We horizontally stack both representations to generate two sparse matrices, amplifying the richness of our feature space. Additionally, we compute the absolute distance, which further enhances the feature set, resulting in another sparse matrix.

Incorporating these enhanced features into our model yields compelling results. The figure below illustrates the corresponding performance metrics:





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

           0       0.99      0.98      0.99    184248
           1       0.97      0.99      0.98    107646

    accuracy                           0.98    291894
   macro avg       0.98      0.98      0.98    291894
weighted avg       0.98      0.98      0.98    291894

VALIDATION results:
              precision    recall  f1-score   support

           0       0.87      0.89      0.88      9667
           1       0.80      0.77      0.79      5696

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

TESTING results:
              precision    recall  f1-score   support

           0       0.87      0.88      0.88     10241
           1       0.79      0.78      0.79      5931
...
```

Conclusion:

In summary, our objective was to establish a robust model as a foundation for addressing Natural Language Processing (NLP) challenges. To achieve this, we employed a methodology featuring custom classes and a structured pipeline to manage our various models effectively. Once we had established a solid framework, we ventured into experimentation, incorporating additional features such as varied text vectorization methods and aggregation functions, alongside diverse classifiers.

Remarkably, our results surpassed expectations, with test set accuracy reaching approximately 0.81 and an AUC (Area Under Curve) of 0.92. Recognizing the room for further refinement and extensive exploration in this field, we have refined a foundational structure applicable to broad NLP problems. This framework presents a promising avenue for future enhancements and applications in both research and practical classification scenarios.