NLP – Delivery 1

Eyuel Muse, Dario Gallego, Sergio Hernández

Quora Challenge

We are asked to consider a simple solution and an improved one in order to solve the Quora challenge: https://www.kaggle.com/c/quora-question-pairs/overview. The challenge consists on identify if a given pair of questions have the same meaning/intention. For this purpose, each sample of the dataset presents a pair of strings, containing the two questions. And also, a target feature, indicating if they have the same meaning or not.

1. Simple solution

As the simple solution, we have implemented a Count Vectorizer model, following the preparation notebook "*delivery_1_quora.ipynb*".

First of all, as we don't have validation/test data with labels, we have had to create the split asked by the teacher. In order to share the same train, validation and test splits across teams.

After that, simply giving the data to the Count Vectorizer resulted in an error. This happened because the dataset contained objects that are not strings, e.g. floats. So, it was necessary to preprocess the data casting it as strings. And then, we could fit the count vectorizer with all the questions from the training set (joining both pairs of questions) and obtain the desired vectorization for question1 and question2 fields.

These vectorizations are represented with sparse matrices, as far as we can obtain hundreds of thousands of features, that contain mostly zeros. After training a logistic model with these features, we can obtain a set of predictions, and calculate the metrics shown in Table 1.

	Roc AUC	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Train	0.787490	0.813972	0.782060	0.686707	0.731288
Validation	0.720278	0.749007	0.677167	0.610729	0.642234
Test	0.729360	0.757791	0.695546	0.618778	0.654920

Table 1: Evaluation metrics of the simple model.

We will compare these results with the ones from the improved solution but, right now, we can say that they do not seem that bad, according to the difficulty of this “question meaning” task. However, a more complex approach that just the counts of the number of words for each question, could help. So, in the improved model, we will replace the count vectorizer with a tf-idf one, in order to emphasize the most relevant words present in our training questions.

With vectorizations like these, we have a large number of features for both question1 and question2. But the machine learning model doesn’t get any direct information about which are from each question, in order to compare them. To try to solve that, we can add new features with information about a direct comparison between a question pair. For this reason, in the improved model we will calculate some distances between them, for example, the cosine distance between the tf-idf vectors. And we will use these distances also as features.

Word embeddings (and their average at sentence level) also could help, thanks to their capability of capturing semantic and syntactic relationships between words. So, they could add a plus to the word counting based approach of tf-idf. And more specific features for our question comparison task could also be considered. For example, to compare if the question word that appears in each question of the pair is the same.

2. Improved solution

We train again a logistic model, in order to compare the improved solution with the simple one. But with new different features (all concatenated) to improve the performance of our model in detecting if two questions are the same.

We consider the tf-idf vectors, the cosine distance between the tf-idf vectors, embeddings at sentence level, the cosine distance between these embeddings, the jaccard distance at sentence level and various approaches for word comparison.

2.1. Preprocessing

In the simple model, we could give to the count vectorizer the data without preprocessing it too much (because it applies its own preprocessing and tokenizes). But now, we also need to reach the token level in our own preprocessing, in order to create some of our features.

First, we cast the data as strings, to solve the problem of having another type of data, as floats, for example. And we also lower it, in order to don’t have differences between lower and upper case letters.

Next, we remove a list of 47 stop words from each question. Because these words don't carry much meaning and can make two questions more different. Finally, we obtain words by tokenizing using a regular expression.

The regular expression pattern used also considers numbers as words. Although the similarity between near numbers cannot be captured, we think that considering numbers in this "questions comparison" scenario could make sense. Because two equivalent questions should ask about the exact same number.

2.2. TF-IDF and cosine distance

For TF-IDF, we used the version provided *scikit-learn* without much customization other than the number of features. We tried different sizes, but the one that yielded the best performance was essentially not limiting the features at all.

We also calculated the cosine distance between the TF-IDF representations of questions 1 and 2.

2.3. Word embeddings and cosine distance

In order to obtain the dense word embeddings, we used the word2vec model provided by *gensim*. We calculate the embedding of each token in a sentence using 300 features and it computes the average of all the word embeddings for the current sentence.

Then we compute an additional feature by calculating the cosine distance between the embeddings of questions 1 and two.

2.4. Jaccard distance at sentence level

As a complementary feature, we consider the jaccard distance at sentence level, taking as elements of the compared sets the tokenized words. So, we can pay attention to the word intersection between each question pair.

And with this, we generate a distance feature in a more direct way, from the original tokenized questions. And not from the vectorized questions of tf-idf or from the embeddings.

2.4. Word-comparing features

For the word-comparing features, we tried two approaches. The first one compares all tokens of the shorter question against the tokens of the other question with the same index. We then divide the total number of matched tokens by the length of the longer question to avoid disregarding any unmatched tokens. This approach aims to identify questions that have similar wording, particularly in the shared length of the questions.

The second approach checks the first keyword of each question to detect commonalities in the types of questions being asked. We search for the first keyword on each question and then we check if they are the same type of keyword. We group certain keywords, such as "how," "can," and "could," and "what" and "which," as they can be interchanged without altering the meaning of the question.

While these features improved the performance of our models, there is still room for improvement by incorporating other features like lemmatization, grammatical structure, named entities, synonyms, among others. However, we must also consider the computational time required for such additional features.

3. Evaluation

The metrics used to evaluate the performance of the models are Roc AUC, Accuracy, Precision, Recall and F1-Score. In general, the higher the value of these metrics, the better the model's performance.

	Roc AUC	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Train					
Simple model	0.787490	0.813972	0.782060	0.686707	0.731288
Improved model	0.803598	0.824304	0.782531	0.724793	0.752556
Validation					
Simple model	0.720278	0.749007	0.677167	0.610729	0.642234
Improved model	0.767298	0.791838	0.738920	0.673725	0.704818
Test					
Simple model	0.729360	0.757791	0.695546	0.618778	0.654920
Improved model	0.767925	0.792233	0.743156	0.673381	0.706550

Table 2: Evaluation metrics of both models (simple and improved).

Looking at the results from Table 2, we can see that the improved model outperforms the simple model in all metrics across all sets. For example, the improved model has a higher Roc AUC of 0.767925 compared to the simple model's 0.729360 in the test set. Similarly, the improved model has a higher accuracy of 0.792233 compared to the simple model's 0.757791 in the test set.

The improved model's superior performance could be attributed to the combination of features that it uses. By including additional features beyond just the count of words, such as embedding and distance measures, the improved model is able to capture more information about the similarity between questions, resulting in better performance.

Overall, the improved model is the better model for this task, as it consistently outperforms the simple model in all metrics and sets.