Machine Learning Engineer Nanodegree – Quora question pairs

Capstone Report

Rahul Choudhury
16th july, 2017

I. Definition

Project Overview

This project is taken from the Kaggle platform, and is currently available under the name Quora Question Pairs in this link: https://www.kaggle.com/c/quora-question-pairs. Quora is a platform where anyone can make a question, and other users give answers. According to Quora, they receive over 100 million visits every month, so it's very common that the question someone asks has already been answered by others. In order to make it easier for someone to find an answer, they use a Random Forest model to identify duplicate questions.

This project is about finding a model that can determine whether a pair of questions has the same meaning or not. The input data is a data set of about 400,000 pair of questions with a human provided label stating if they have the same meaning. As for testing, Kaggle provides a set of unlabeled pairs, which are then compared to their own human labeled results.

Problem Statement

The problem to be solved is to determine if two different questions asked by Quora users, are they have the same meaning? There might be lot difference in the way users ask a question, so it's quite challenging to understand the intend of the questions, and put them in the same bucket. This is more of a type classification problem, where you basically answer with yes or no.

For solving this problem, I have analyzed the basic aspects from the training set, and obtaining a benchmark predictor. Then, I have preprocessed the data and create new features which proved to be helpful. After this, I have tested some models and determine

which is the best for using in this case. I have then tuned the parameters using GridSearchCV, and applied the chosen model to the testing set. Finally, I have uploaded the results to Kaggle and compare the log-loss obtained with the one from the benchmark. This log-loss proved to be smaller, which means that the model is better than a naive predictor, and gives some extra information about the data.

Metrics

The evaluation metric that I have used for measuring the performance of the models is log los between the true and the predicted values. Log Loss quantifies the accuracy of a classifier by penalizing false classifications. Minimizing the Log Loss is basically equivalent to maximizing the accuracy of the classifier,

The mathematical formula for determining the log loss is the following:

-log P(yt|yp) = -(yt log(yp) + (1 - yt) log(1 - yp))

where y is the real value and p the predicted one.

II. Analysis

Data Exploration

In the training data set we can find the following information:

id	qid1	qid2	question1	question2	is_duplicate
			invest in share market in india?	share market?	
1	3	4	What is the story of Kohinoor (Koh-i-Noor) Diamond?	What would happen if the Indian government stole the Kohinoor (Koh-i-Noor) diamond back?	0
2	5	6	How can I increase the speed of my internet connection while using a VPN?	How can Internet speed be increased by hacking through DNS?	0
3	7	8	Why am I mentally very lonely? How can I solve it?	Find the remainder when [math]23^{24} [/math] is divided by 24,23?	0

- qid1: Unique ID of question 1
- qid2: Unique ID of question 2
- question1: Content of question 1
- question2: Content of question 2
- is_duplicate: A label stating if both questions are the same ‘1’ or not ‘0’

In the test data set we can find the following information:

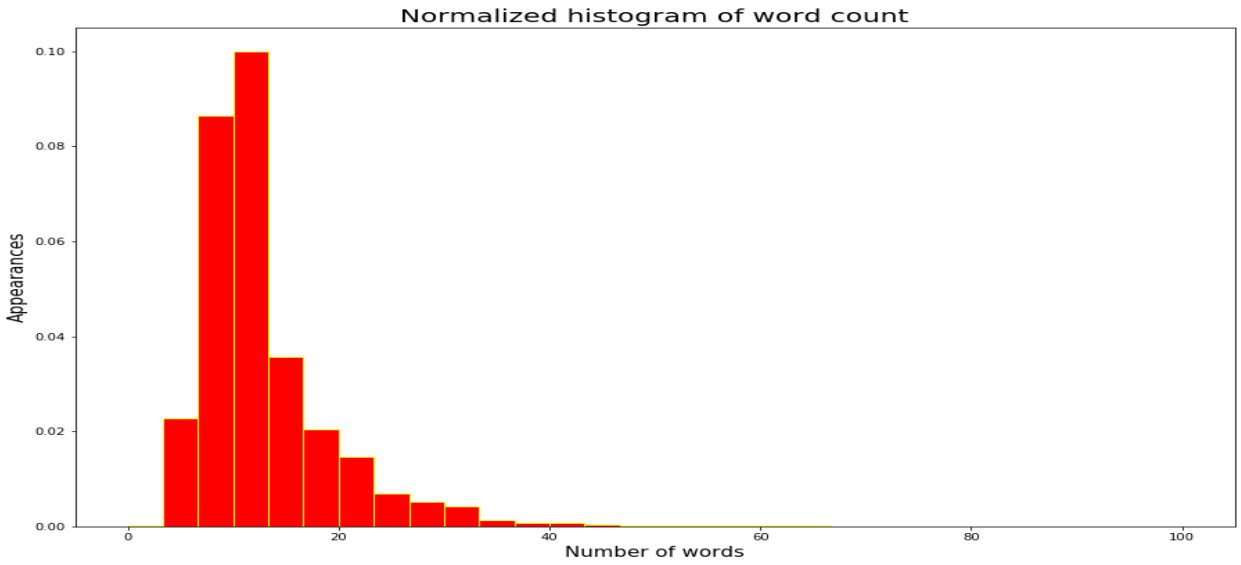
test_id	question1	question2
0	How does the Surface Pro himself 4 compare with iPad Pro?	Why did Microsoft choose core m3 and not core i3 home Surface Pro 4?
1	Should I have a hair transplant at age 24? How much would it cost?	How much cost does hair transplant require?
2	What but is the best way to send money from China to the US?	What you send money to China?
3	Which food not emulsifiers?	What foods fibre?

Input data set information:

Number of rows: 404290
Number of Duplicate pair of questions: 149263
Number of Non-duplicate pair of questions: 255027
Number of Unique questions: 537933

Exploratory Visualization

I have done overall analysis of the questions in the training set using the nltk librabry for this, and plotted a histogram to see the frequency of the lengths of the questions. As we see here, most of the questions are composed from about 8 to 12 words. There are some of them which can have even more than 40 words.



Algorithms and Techniques

For this project I have evaluate 3 algorithms: AdaBoost, Random Forest and XGBoost. The reason for this is that they are all ensemble methods, which work well for problems where the features are weak learners. In this problem all the features will be so, because there doesn't seem to be one that is highly correlated with the meaning of the question.

AdaBoost

This method uses a base algorithm (by default, Decision Trees) and applies it repeatedly, each time improving it. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. In order to do this, on every iteration it increases the weight assigned to each of the data points were it previously made a wrong classification. After n times, the final model is made from the weighted sum of all the learners. These weights are updated based on the error rate of the classifier, which is the total number of misclassifications divided by the training set size. The weight for a given classifier grows exponentially as the error approaches 0. If a classifier has a 50% accuracy it is the same as random, so the weight assigned is 0.

Random Forest

Here, what is called a *bagging* approach is used. To build a decision tree, the algorithm starts with a random subset of the training samples. The final prediction will be an average of individual estimators. The idea is that each of these estimators provides a low bias, which is desired, but high variance. Averaging them keeps the bias low while also lowering the variance.

XGBoost

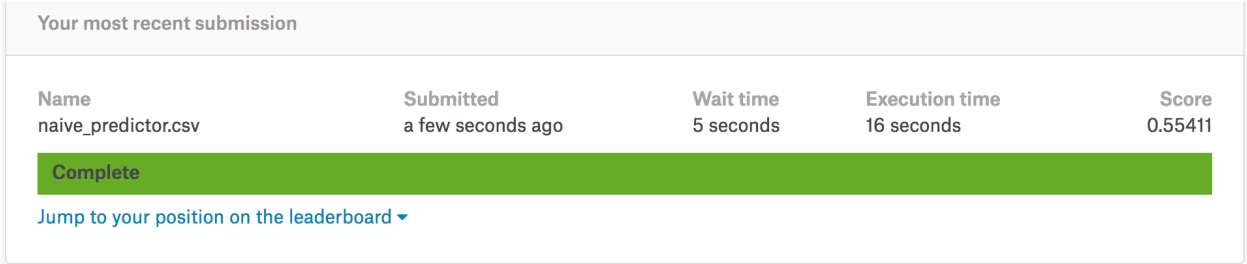
This algorithm, which name stands for *Extreme Gradient Boosting*, is a model that takes *weak* predictive models and ensembles them to create a stronger one. Any model that works better than random can be used. This is exactly the same as in AdaBoost. The difference is the way in which each algorithm updates the weight of each predictor. In XGBoost, this is done by taking the gradient of the loss function. In this way, the algorithm updates the weights taking a step in the direction that minimizes the loss function.

Benchmark: Naive predictor

The benchmark model used for comparison is the measure the proportion of pairs in the training set that refer to the same question, and assume that this is the probability of a new pair of having the same meaning. therefore, for every new pair, the result is the same. This process then have a simple output, which has 1 with some *p* probability and 0 with probability *1 - p*.

The log-loss obtained was **0.55411**. This is used as a benchmark, and any model with a lower value is considered as better.

Kaggle score:



III. Methodology

Data Preprocessing

I have executed the following steps for data cleaning and preprocessing.

- **Remove stop words:** In these steps have removed stop words these words actually does not provide any information to find the similarity between two questions, eg: you, that, yours, are, etc
- **Apply stemming**: stemming algorithm find the root of a word, like box, boxes are will be consider similar after applying the algorithm, it will became box.
- **Tokenize:** This will split the questions in multiple words.
- **Tf-Idf:** term frequency help to find the importance of each word and inverse document frequency find the relative importance of a word in document, which diminishes the weight of terms that occur very frequently in the document set and increases the weight of terms that occur rarely.
- **DiffLib:** This is a Python library that will help me finding differences between 2 strings. Specifically, I will be using the SequenceMatcher method.

By doing above mentioned steps, I will get a new set of features, which will be composed of:

- **shared_weights:** a measure of how many words both questions share, weighted by the tf-idf value of each word.
- **shared_count_scaled:** it will represent the normalized amount of words that they have in common.
- **len_dif:** the difference between the lengths of the questions.
- **z_match_ratio:** the value obtained from the SequenceMatcher method.

Implementation

After applying all the steps stated above, I got a new data set with four features (shared_weights, shared_count_scaled, len_diff and z_match_ratio).Having this new data set, I used the train_test_split method from sklearn to split it between a training and a testing set, consisting on 80% and 20%, respectively. Now I compared the 3 models chosen before, to see if any of them got a significantly smaller *log loss* than the rest. The results were the following:

- AdaBoost: 0.68
- Random Forest: 0.64
- Random Forest: 0.64

Refinement

As the differences were not too high, I tried some tuning on the parameters for each model, using GridSearchCV. The new results obtained were:

- AdaBoost: 0.54
- Random Forest: 0.50
- XGBoost: 0.49

I choose XGBoost, not only because it got the best log loss, but also because it is the fastest to train. Before tuning even more the algorithm, I made an adjustment on the testing set. Based on this article from Kaggle and it's comments, I know that the distribution of positive and negative cases in the test data is not the same as the one in the training data: https://www.kaggle.com/davidthaler/quora- question-pairs/howmany-1-s-are-in-the-public-lb/run/1013730. In the training data we have around 37% of positive cases, while the test data has around 16.5%. So for having a better score from Kaggle I need to re balance the data. With this new data, I tried some extra tuning, and got a log

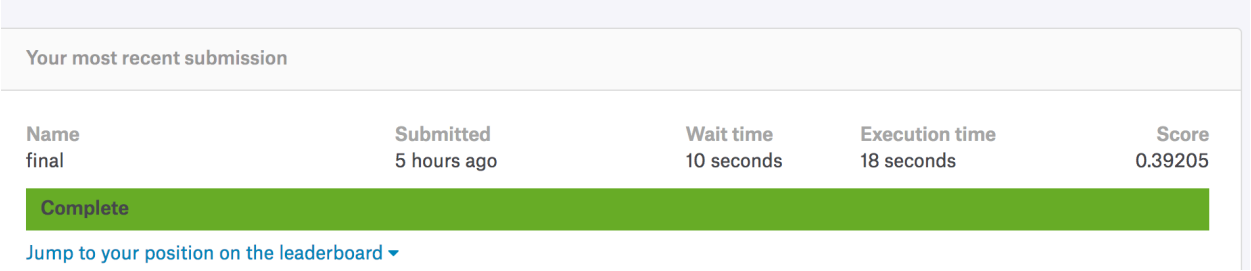
loss of 0.35. The parameters used in this final model were the following:

objective = 'binary:logistic', eval_metric = 'logloss', n_estimators = 500, reg_alpha = 0.5
learning_rate = 0.3, gamma = 0, reg_lambda = 1, min_child_weight = 1

IV. Results

Model Evaluation and Validation

Having already decided the model to use, I applied the same transformations to the provided testing data set, in order to get the new features and be able to make predictions. This provided a different input to test the model, besides the one obtained from the *train_test_split* method. This new testing set doesn't have the labels in order to test it by my own, but it is possible to do it on the *Kaggle* platform. This is good, because this data is not used during the training set, so it provides a very robust indicator. I think that my model could serve as a first approach for analyzing a problem like this one, but could not be used as a general setting, because it is not as accurate as it should be. In order to be useful for a company to make predictions, *log loss* should be much lower, so that results can be trusted more. **Justification** The final score obtained in the *Kaggle* platform is **0.39205**, which is an improvement over the benchmark of 0.55411. This value should be improved in order to make significantly good recommendations.

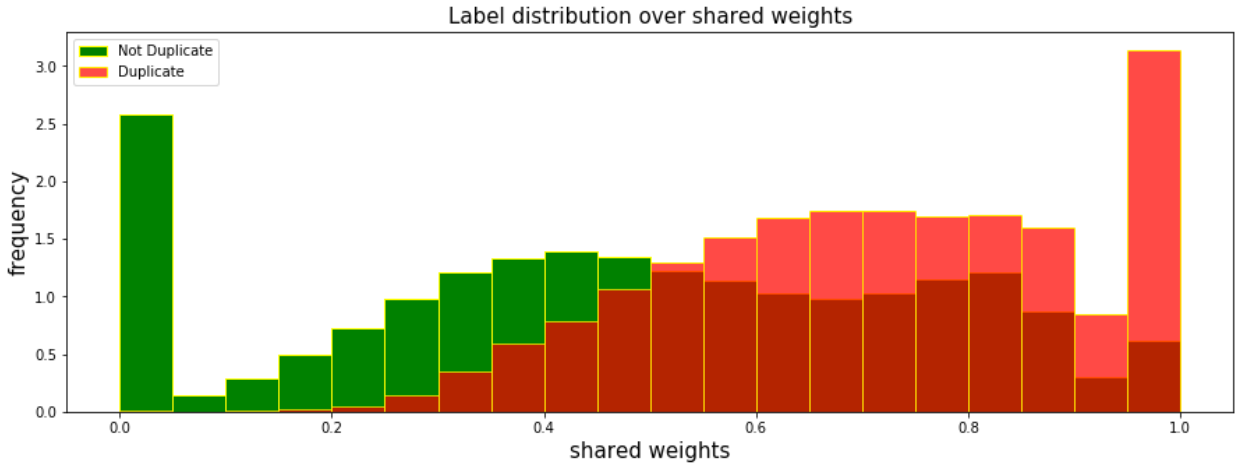


V. Conclusion

Free-Form Visualization

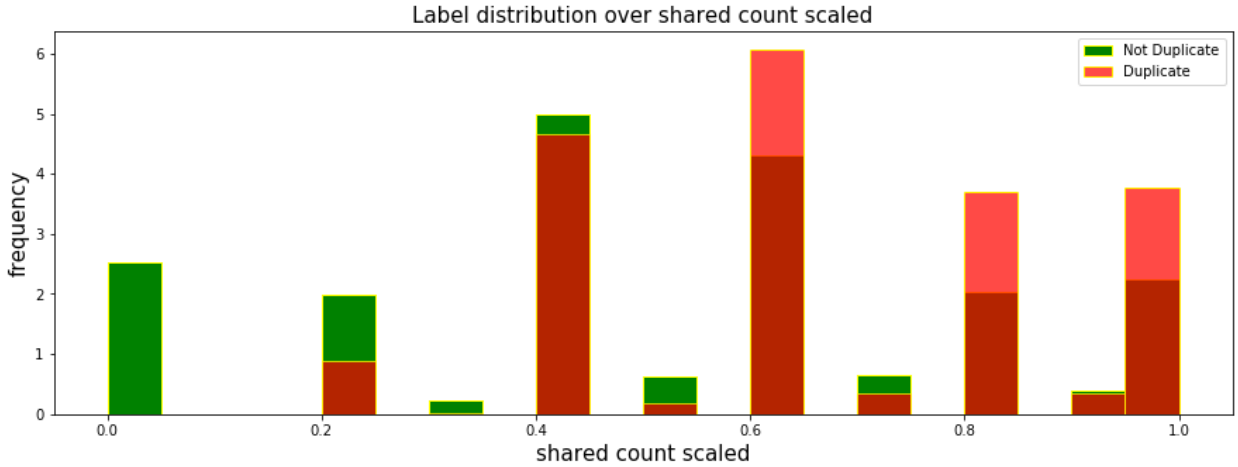
I have added graphs for each of the new features here, the graphs are showing how the features are valuable here and able to make a good distinction between two cases. I will show the distribution of each of the four new features, for both labels.

Shared weights:



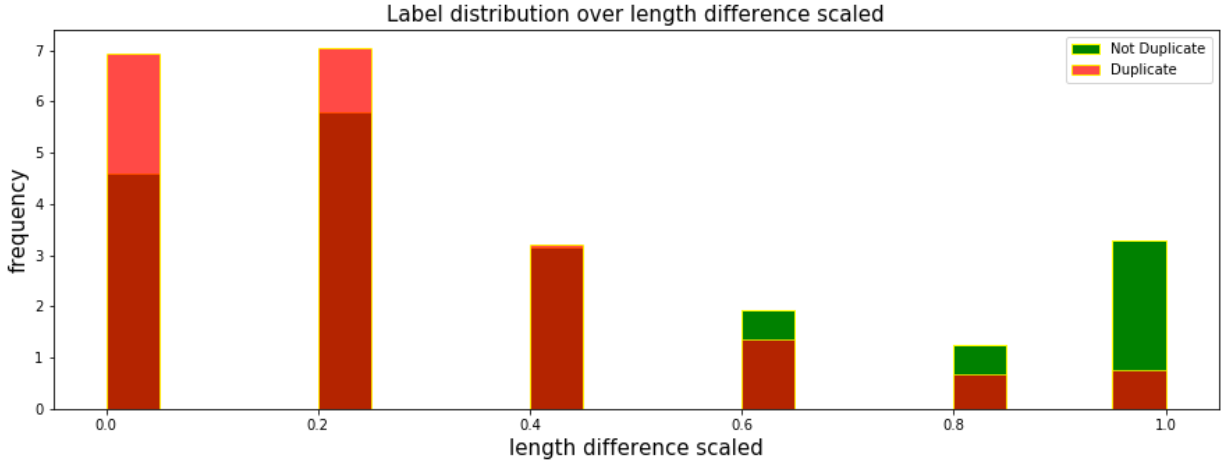
Questions with shared weight > 0.5 found to be same, as they have more words in common.

Shared count:



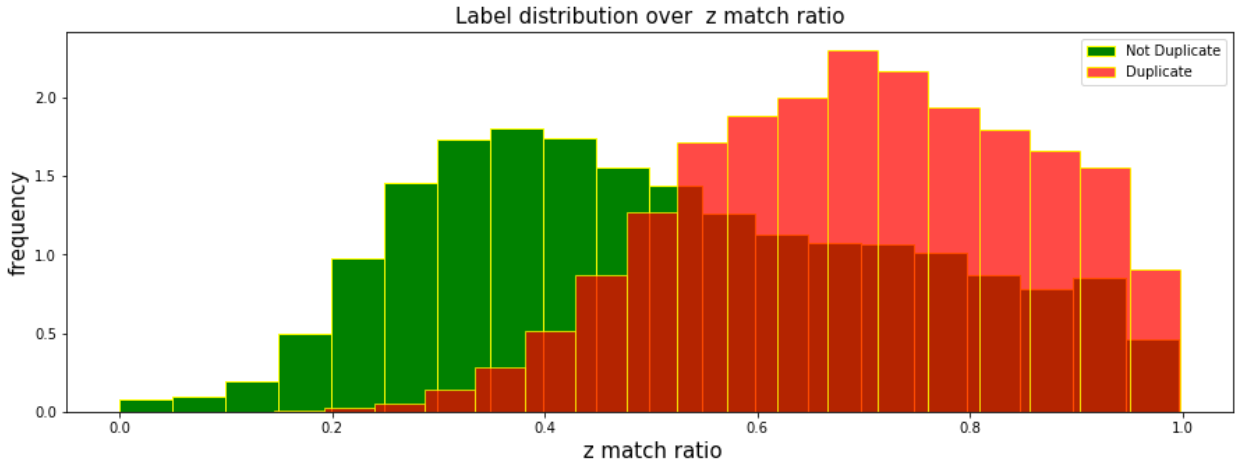
Pairs with a higher value tend to be duplicated in more cases than the ones with lower values.

Length difference:



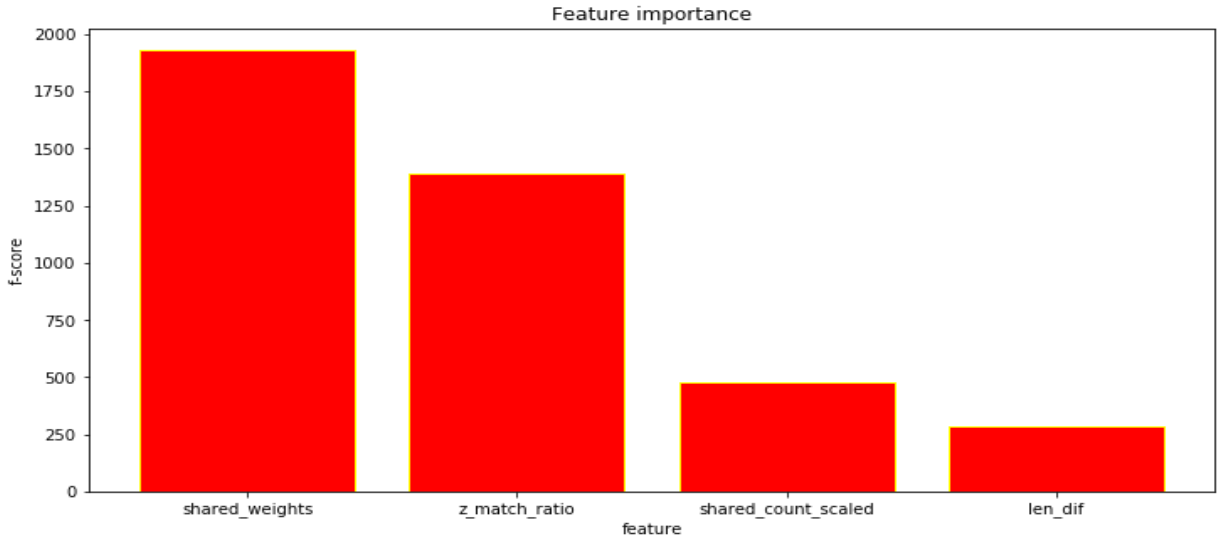
Question pair found to be duplicate more where difference between length is less.

Zmatch –ratio:



Again, pair have similar with more probability where z match ratio is > 0.5

Features important chart(f-score):



from the above figure, its clear that shared weight and z_match_ratio are more important than shared_count and len_diff.

Reflection

For this project, I started analyzing the basic aspects of the dataset, and getting a naive predictor to use as a benchmark and see if my model would represent an improvement. After this, I preprocessed the training data, and tested that each new feature showed graphically a different distribution between duplicate and non duplicate pair of questions. Having obtained this, I compared the three models which I believed could be useful for this problem. Given that performances were similar when tested with default parameters, I used GridSearchCV for tunning each of them, and comparing the new results. I chose

XGBoost for it's good performance, and it's fast training. I finally preprocessed the testing data, used my chosen model for prediction, and tested the performance within the Kaggle framework. I got a better result than the benchmark.
I found this problem to be very interesting, given it's high complexity. Natural language processing is involved, The fact that there are a lot of different ways to ask the same question, some of them very different between each other, makes this problem a very hard one for obtaining a good result. Also, it's not easy to use neural networks in these kind of problems, because each word could be seen as a feature and there are thousands of them.
I think that my model could serve as a first approach for analyzing a problem like this one.

Improvement

I think there are different ways to improved this algorithm. For example, every word should be contrasted to it's synonyms, in order to group together words that are now consider different but refer to the same. Another improvement could be giving words that start with capital letters a greater weight for TF-IDF analysis.

The problem could also be treated from a different approach, which I would have used if I knew how. As seen in this paper (https://web.stanford.edu/class/cs224n/reports/2759336.pdf), Convolutional Neural Networks can be used for sentence classification. I think that this approach can yield much better results, given the fact that deep learning catches much better aspects where humans are good at, like it does with image recognition.