

Machine Learning Nano Degree Final Project

Addressing Quora Duplicated Questions using NLP

Dengpan Yuan

01/07/2019

# Index

1.	Definiation	
1.1.	Background . . . . .	3
1.2.	Question Analysis . . . . .	3 - 4
1.3.	Measurement . . . . .	4 - 5
2.	Analysis	
2.1.	Behavior Analysis . . . . .	5 - 6
2.2.	Frequency Analysis . . . . .	6 - 7
2.3.	Length, Quantity Distribution . . . . .	7 - 8
	Word Analysis. . . . .	9 - 10
2.4.	Algorithm and Techniques	
2.4.1.	TF-IDF Algorithm . . . . .	10
2.4.2.	TF-IDF Analysis . . . . .	10 - 11
2.4.3.	XGBoost . . . . .	12 - 13
2.5.	Technical Support . . . . .	14
3.	Procedures	
3.1.	Data Cleaning . . . . .	14 - 15
3.2.	word_sharing的easy_submission . . . . .	15
3.3.	TF-IDF Implementation . . . . .	15 - 16
3.4.	Training TF-IDF Data	
3.4.1.	Data Preparation. . . . .	17
3.4.2.	XGBoost Training . . . . .	18 - 19
4.	Result	
4.1.	Kaggle Competition. . . . .	19
4.2.	Comparison of Others. . . . .	20
5.	Conclusion	
5.1.	Thoughts and Inspiration . . . . .	21
5.2.	Further Improvement . . . . .	21
6.	Reference . . . . .	22

# 1 Definition

## 1.1 Background

This project will train a text recognizer on Kaggle through NLP to train Quora 400mb with a total of 404290 data on the problem group, and will measure the final results of participation in the competition with log-loss.

There are hundreds of millions of users on Quora, so these users will inevitably ask repeated questions, so there may be many questions on Quora. Quora is based on the random forest algorithm [1], based on the decision tree algorithm to determine Whether two questions are duplicates, so our goal is to calculate whether every two questions are duplicates or not. Doing so will make Quora's questions more streamlined and more impressive, and the community will have higher quality questions and answers.

The data set used in the project is provided by Quora, where the data is manually labeled, so there may be errors [2]. In addition to errors, Kaggle's anti-cheating mechanism, in order to avoid the accidental nature of the contestants, kaggle added some data to the test.csv file, which also increased the difficulty of the competition, and the accuracy of our recognition will decline.

## 1.2 Question Analysis

The data of the project comes entirely from the actual problem. Due to the repetitive nature of the problem, it is a difficult problem to judge, which belongs to the np problem. The following example is from kaggle and can be used to illustrate the uniqueness of this project:

Q1: What is the **biggest** natural number?

Q2: What is the **smallest** natural number?

It is easy to see that this is two different. If you simply quote the similarity algorithms common in NLP, such as Euclidean distance algorithm, Pearson correlation coefficient, or Jeckard coefficient calculation, it is easy to get the opposite result: because all the words in them are the same except the most critical word, biggest. And smallest. These two words are very important, so we can't just compare whether there are more similar words in the two questions,

and decide whether there is a higher probability. So we need to measure the importance of words, not the number of words in common.

## 1.3 Measurement

Because I use the kaggle platform, this platform uses the log-loss function [3] to measure the accuracy of the model. This function will compare our final result set with the results of manual labeling, and finally get a A number from 0 to 1. The smaller the number, the more accurate the model. The log-loss mathematical consensus is as follows:

$$-\frac{1}{N} \sum_{i=1}^N [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)]$$

Where N represents the size of the data set,  $p_i$  represents the prediction result of the model at label i, and  $y_i$  represents the prediction result of the final official model of Kaggle. We can observe that in order to make our log-loss value smaller, as a contestant, we can reduce the value of  $p_i$  as much as possible.

## 2 Analysis

### 2.1 Behavior Analysis

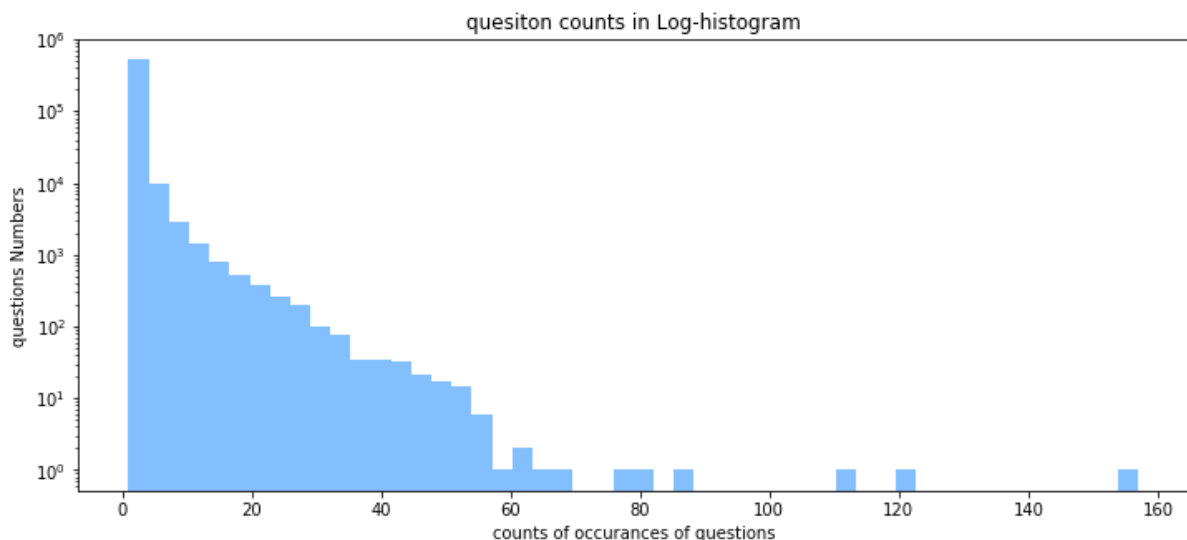
Among them, Quora provides a total of 2 data sets train.csv, test.csv and. The main thing we use to build the model is train.csv, so first observe the characteristics of train.csv, as shown in the figure:

	id	qid1	qid2	question1	question1	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0

1	1	2	4	What is the step by step guide to invest in sh...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0

As can be seen from the figure, there are 6 columns, and the id represents the pairing id of the problem pair. qid1, qid2 represent the ids of the pairing questions, question1, question2 are the detailed descriptions of the questions, id\_duplicate represents whether the two questions are duplicated, 0 is no, 1 is yes. In addition, you can see that the problem with qid of 2 in the first and second lines appears twice, that is, the same problem can occur multiple times, which results in the occurrence of certain words more often. The frequency is not uniform, and the trained model may tend to words that appear more frequently. So in the first step we need to make statistics on the frequency of problems.

## 2.2 Frequency Analysis



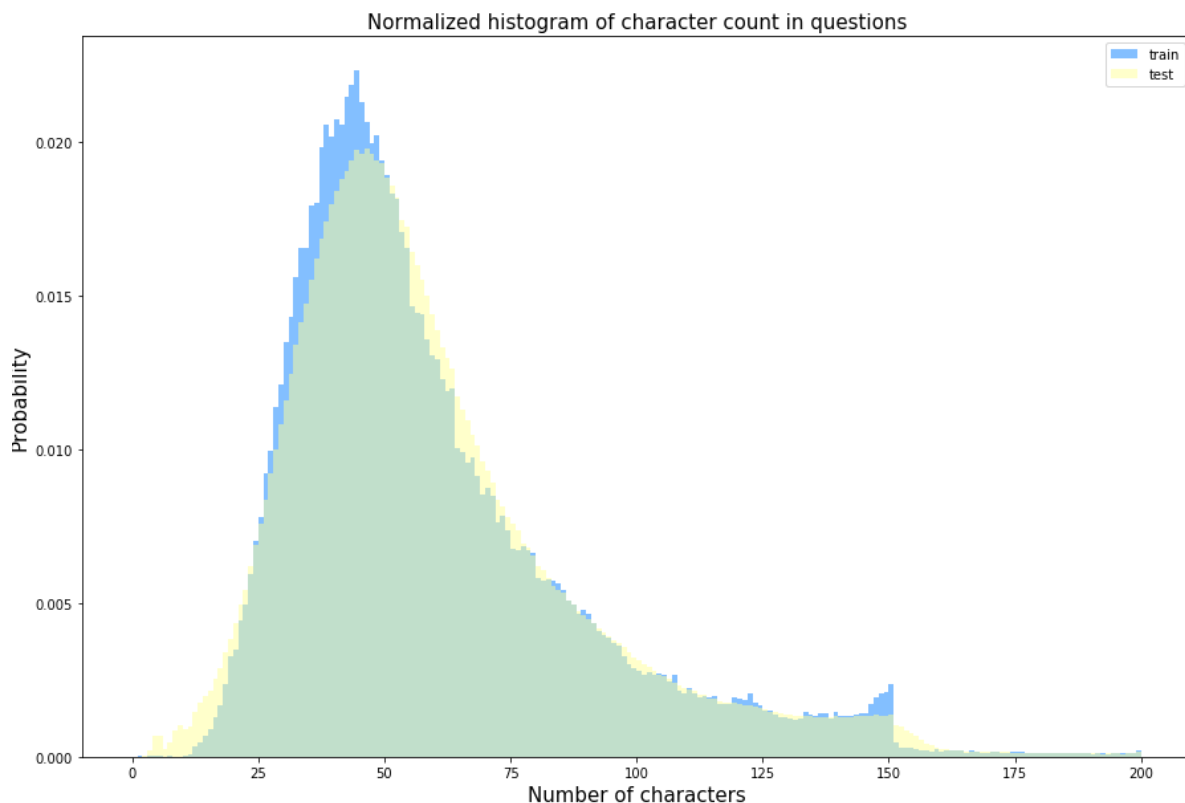
This is a histogram of the distribution of the number of occurrences of the problem that I drew using matplotlib. From the above figure, we can see that most problems only occur once, of

course, there is also one problem that appears nearly 60 times, so the matching degree of the problem is It is uniform and does not have much impact on the results of our training. Such a data set is also a relatively standard data set. In addition, I also calculated the ratio of positive data. The calculation method is as follows:

$$TP = \sum_{i=0}^N \frac{Ti}{N}$$

Where N represents the total data set of train.csv, and T is the column of is\_duplicate. Through this formula (that is, calculating the average value), the final ratio of our positive data. The final result is 37%. Positive data is very helpful for my results, which will be discussed in the next chapter.

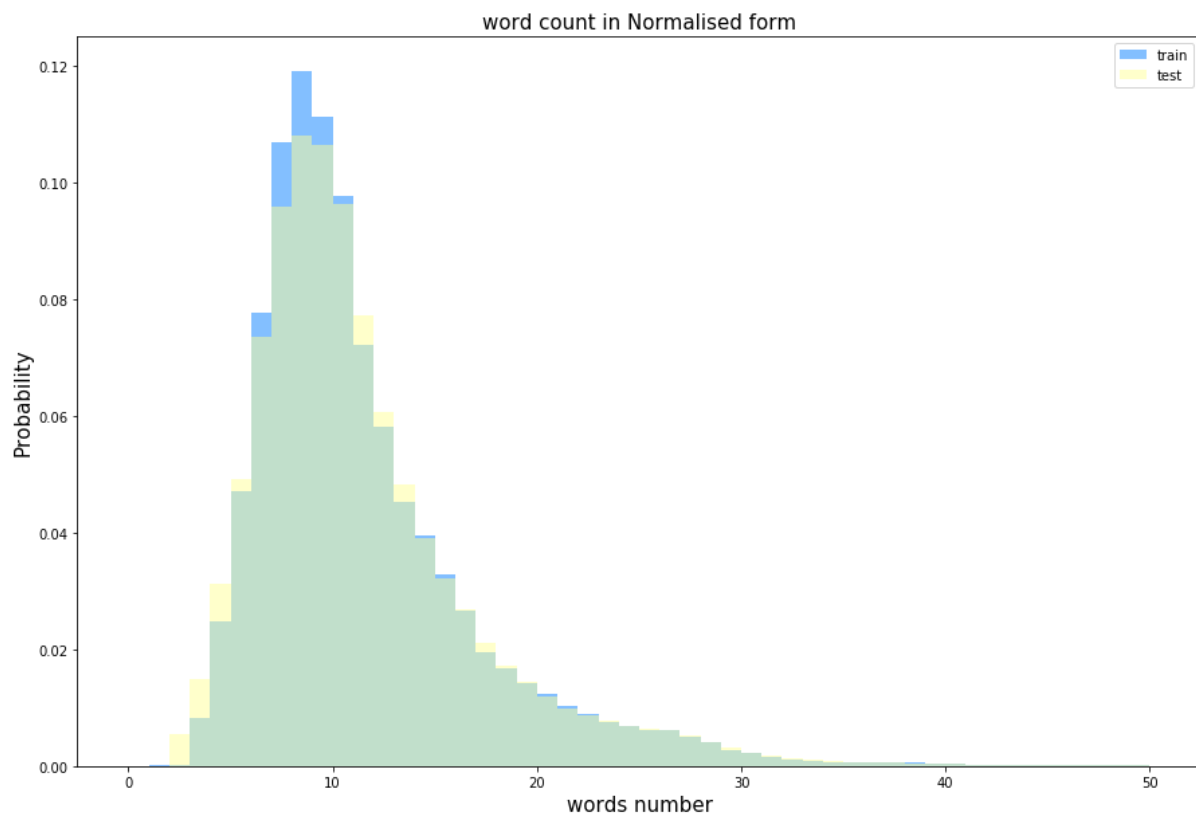
## 2.3 Length, Quantity Distribution



The problem length measurement is also very helpful for our model construction. Imagine this question: What is the relationship between the problem length and the problem similarity? The answer is of course that most of the similar questions have the same length. With the help of

matplotlib, I plotted the length distribution histograms of train.csv and test.csv. By analyzing the length, we can see from the above problem length distribution histogram that most problems are between 25-75 characters, So for both training and test sets, these problem pairs have about the same length.

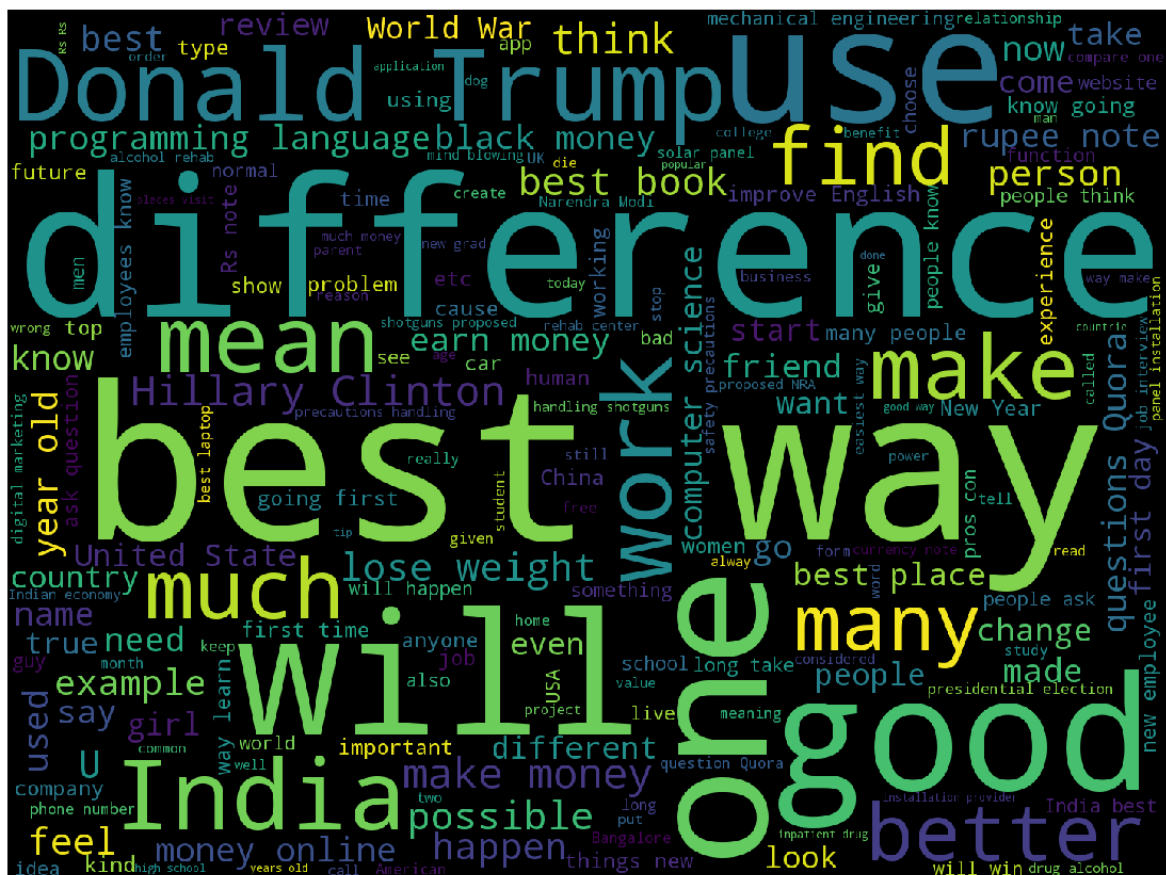
By analyzing the length of the problem, it is helpful for us to study the similarity of the problem. Next, we need to analyze the words in more detail, specifically the number of words in the problem. Although most problems have the same length from the above picture, this does not mean that most of these problems have similar word lengths. Some problems may have 17-character words such as biotransformation, but they are paired. Questions appear with 5 cat words totaling 15 characters, so their question lengths will be the same. Therefore, we predict that if the word distribution map is roughly similar to the above figure, it means that most problems have the feature of similar words.



The above is a histogram of the word distribution in each question. The number of words in most questions is less than 20, and you can see that this is roughly the same as the first picture, so we have come to the conclusion that most questions have similar words. This also laid the foundation for making bold assumptions in our subsequent research.

## 2.4 Word Cloud Analysis

Before I study further, I need to understand that the problem mentioned at the beginning may not necessarily have the same length (even if there are the same number of words), and there is a very high word coincidence rate, so I conclude that they are duplicate problem. So in order to solve this problem, I made a word cloud to observe the frequency of words.



In the picture above, the number of words appears. It can be seen that the word that appears the most is best, then the way, will ... These words appear the most. According to the logic of our question, these words are some dummy words. Such a question: what is the best way to...? The most important thing is the second half of the question. These words don't seem to be keywords, just some words. More rigorous research will be mentioned later.



## 2.5 Algorithm and Techniques

### 2.5.1 TF-IDF Algorithm

Here I use the TF-IDF (term frequency-inverse document frequency) algorithm. The biggest benefit of this algorithm is to measure the importance of the word. It should be noted that the importance of the word is not linearly related to the occurrence frequency of the person. Understand that TF-IDF works like this: importance increases in proportion to the number of times a word appears in a document, but is offset by the frequency of words in the corpus. So this is why search engines often use variants of the TF-IDF weighting scheme as the central tool for scoring and ranking the relevance of documents given a user query. Applying the TF-IDF algorithm in our actual project can not be more suitable, because most of these words are hypothetical words observed from the word cloud above. Using TF-IDF can eliminate this logical necessity, thus Make our model successfully identify repetitive problems based on the importance of words.

### 2.5.2 TF-IDF Analysis

TF-IDF is divided into two parts. The first is the TF part, which is the word frequency. It refers to the probability of a given word appearing in the file. TF looks the same as our ordinary problem of measuring NPL similarity. In fact, it is to normalize the number of words and prevent the problem of longer characters. The specific expression formula is as follows:

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{i,j,k}}$$

It should be noted that the  $n_{i,j}$ ,  $n_{i,j,k}$  represent the number of occurrences of all words in the entire csv file, so  $TF_{i,j}$  represents the word frequency of the words  $n_{i,j}$ . Therefore, we can see that if a word appears more often, its TF will increase; if it appears less, its TF will decrease. The second part is IDF, the inverse document frequency. This is a measure of the universal importance of a word. The specific formula is as follows:

$$IDF_i = \lg\left(\frac{|D|}{|\{j: t_i \in d_j\}| + 1}\right)$$

Where  $|D|$  is the total number of words,  $|\{j: t_i \in d_j\}|$  represents the number of times the word  $j$  appears, and 1 is to prevent the denominator from being 0. So we can see that  $IDF_i$  is different from ID, because the more times a word appears, the larger the denominator of  $IDF_i$ , the

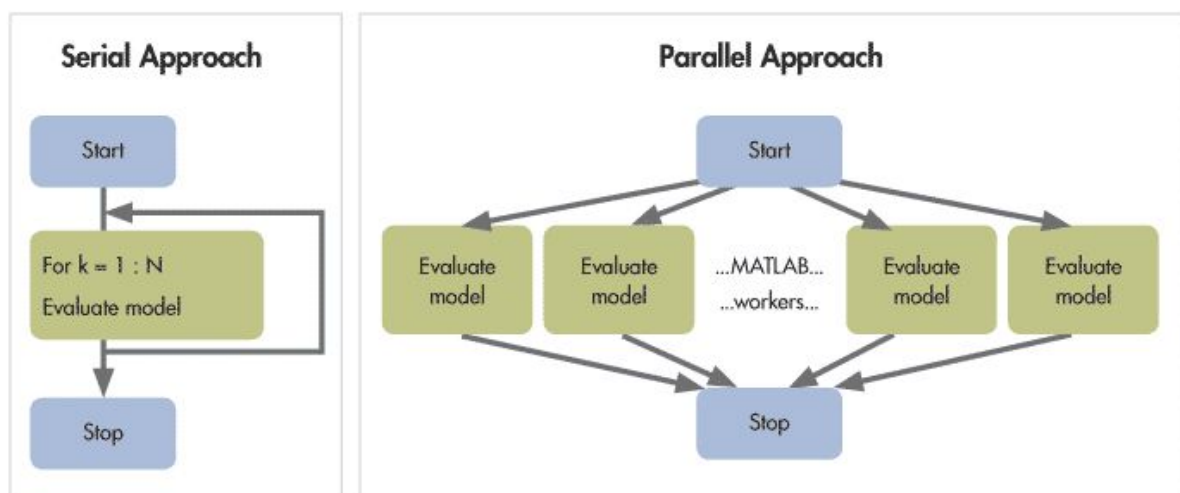
smaller the overall value of  $IDF_i$ ; and vice versa. The final step is to merge our TF-IDF, the two of them jointly determine the importance of a word.

$$TF-IDF_{i,j} = TF_{i,j} \times IDF_i$$

The final word importance is jointly determined by TF-IDF. On the one hand, TF guarantees that many words appear high in importance, on the other hand, IDF offsets this bias, thereby retaining the importance of rare words. In general, the high-frequency words in a particular file and the low-frequency words in the entire word database set, using TF-IDF can produce higher weight. Therefore, TF-IDF tends to filter out common words and retain important words [4].

### 2.5.3 XGBoost

In this project, I used the XGBoost training model. Next, I will introduce why I chose XGBoost instead of other models. XGBoost has the following characteristics: The first is to speed up our training, because the parallel calculation is adopted and written in C++, so the calculation speed is faster than the traditional model in sklearn [5], the following is Zhang Tu is the reason why parallel computing is faster than ordinary computing. The CPU processes more tasks per unit time, so it is more efficient.



Secondly, the parameter setting of XGBoost is very convenient. For example, our `evaluateatn_metricx` is simply set to the `log_loss` evaluation matrix by params `['eval_metric'] = 'logloss'`, which facilitates subsequent submission verification.

The third point is that our model sets logistic regression. The reason for adopting this model is that our word importance does not show a linear relationship with the frequency of word frequency, and so logistic regression is more in line with the context. Here I use the binary logistic regression, the formula is as follows:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_{\theta}(x^{(i)}), y^{(i)})$$

The Cost function is as follows. The Cost function exists in all classification methods, such as linear function, logistic regression, Decision tree ... But the cost function of each model is different. Here is the Cost function of logistic regression:

$$\begin{aligned} \text{Cost}(h_{\theta}, y) &= -\log(h_{\theta}(x)) && \text{if } y == 1 \\ &= -\log(1 - h_{\theta}(x)) && \text{if } y == 0 \end{aligned}$$

What needs to be explained is that  $h(x)$  is our logistic function. In order to specifically explain the principle, its formula is as follows:

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

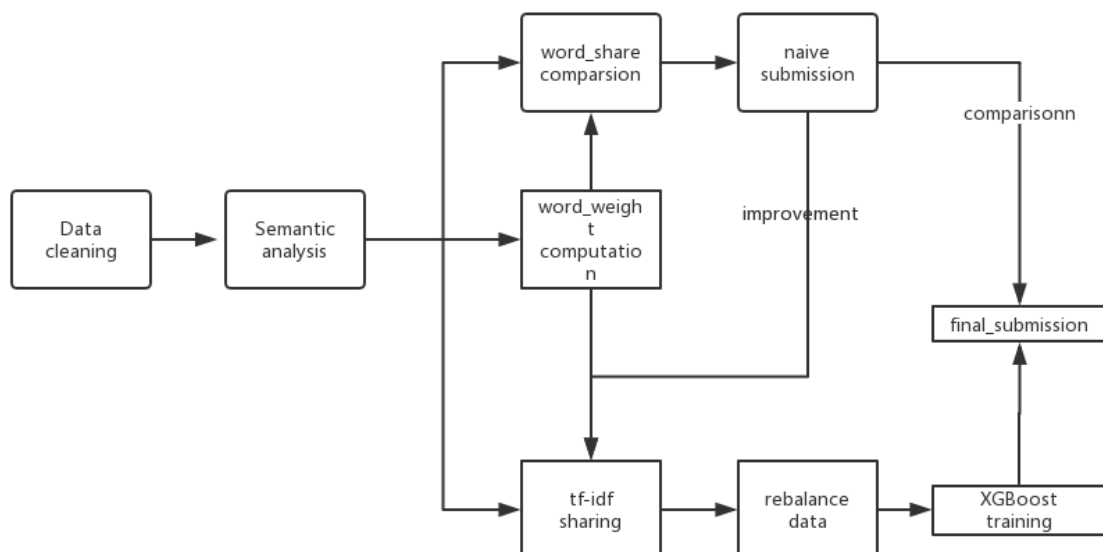
Pushing down from the bottom is a Dataframe composed of the weights after our final rebalancing. The Cost function calculates the deviation of the model so that it will continue to shrink during regression and finally get a satisfactory model. The final  $J()$  is our true regression function, where  $m$  is the total number of rows in the Dataframe,  $y$  is the `is_duplicated` column,  $y == 1$  means it is a duplicate problem, and  $y == 0$  means it is not a duplicate column. In XGBoost, I ended up setting the loop to repeat 400 times. Repeating the loop many times will cause overfitting, and too few will cause underfitting. The initial 400 is based on the output of `get_booster()`. `Best_iteration` function of XGBoost. Finally, this section explains why binary classification is used: because our final result is only two, repeated or non-repeated.

## 2.6 Technical Support

Because this project is a natural language processing problem, Python's NLTK package is used. The NLTK package is an easy-to-use interface that uses more than 50 corpora and vocabulary resources (such as WordNet), and a set for classification and tokenization. Text processing library for stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-grade NLP libraries, and active discussion forums. Therefore, based on a mature platform and a wide range of users, I chose NLTK to use it in the project, mainly using the stopwords function of NLTP, which can help me clean the data and convert all questions into words, which facilitates subsequent word frequency statistics TF-IDF calculation. In addition, sklearn is still used. The most important thing is to calculate the accuracy of the model and verify the `log_loss` and `roc_auc_score` of the model.

## 3 Procedures

Here is flow chart of the procedures :



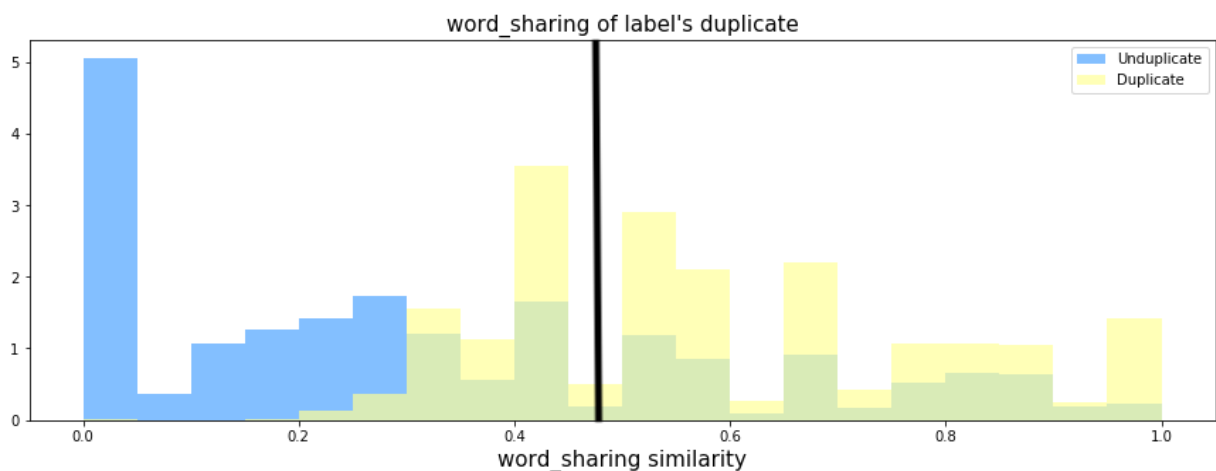
### 3.1 Data Cleaning

The first step is data cleaning, which includes the most important tool is `stop_words` in `nlk` and the `Collection` package that comes with Python, so the basic process is as follows:



## 3.2 word\_sharing of easy\_submisstion

The navie\_submission in the second part is to explore a simple word frequency statistics method, and then submit the final text. In order to facilitate later comparison, the words with the same word\_sharing are visualized.



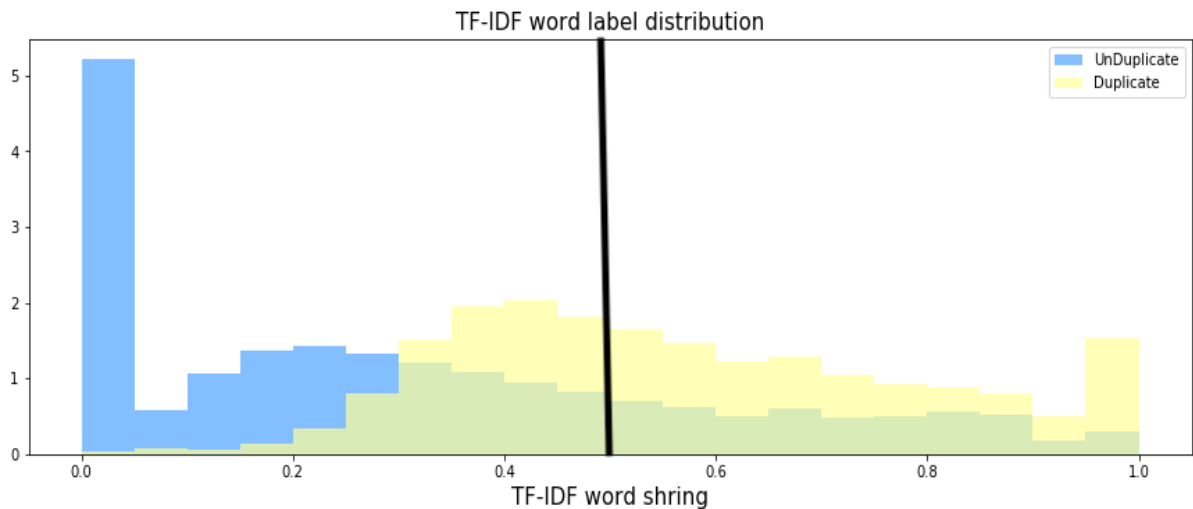
It can be seen that most of the duplicate data is less than 0.5, and most of the duplicate data is greater than 0.5. For more detailed conclusions, I calculated the proportion of data greater than 0.5 in the duplicate and duplicate data, respectively:

unduplicaet : 0.2765432

duplicate: 0.5269956

This means that the vast majority of problem pairs in unduplicate data have fewer identical words, and the vast majority of words in duplicate words have more identical words. This conclusion is more in line with our common sense. But it cannot be ignored that 27.6% of the data in the unduplicate problem have more than 50% similarity. So we need to do the TF-IDF algorithm for these problems and re-measure its weight. But before that, let's do a simple easy\_submission to observe the score obtained from the data without TF-IDF processing. The result is 0.5432, which is not very high. This simple\_submission needs to reduce log-loss to improve the accuracy of the model degree.

### 3.3 TF-IDF Implementation



Comparing this picture with the above picture, we find that the distribution in this picture is more uniform in word\_share, rather than scattered and scattered in word\_sharing. For the convenience of observation, I also calculated the proportion of data greater than 0.5 in this TF-IDF graph:

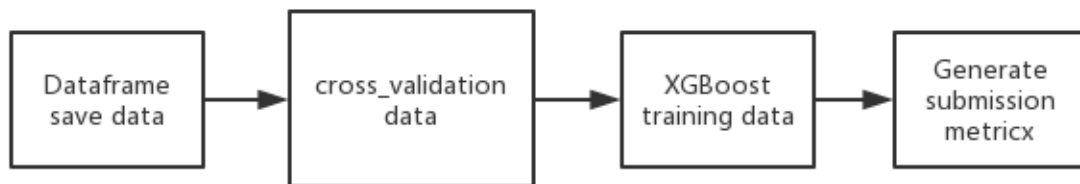
Unduplicate: 0.2484834

Duplicate: 0.5633981

Both of these data are higher than the TF-IDF data that was simply sought before, so the conclusion is that most of the problems in Unduplicate are less "important" (because TF-IDF calculates words Of importance, but in my previous function, the sum of the weights of the words is calculated, so it represents the importance of the problem; in Duplicate's data, most problems are more "important"; and TF-IDF's Values higher than word\_sharing represent the importance of the problem over the similarity between words.

## 3.4 Training TF-IDF Data

The next step is the training data. The following is the process of training data:



### 3.4.1 Data Preparation

After completing the construction of the TF-IDF function, my work is to apply the questions in the test.csv file to apply. (TF-IDF), then you can get the TF-IDF values of all the question pairs; here we need to pay attention to Be sure to first divide the ross\_validation step, divide the dataframe into x\_train, x\_valid, y\_train, y\_valid, and then sample train.csv and val, so as to facilitate subsequent inspection.

## 3.4.2 XGBoost Training

The parameter that XGBoost needs to set is the learning rate, the data needs to be stored in the form of DMatrix, and the training algorithm is binary logistic regression, so after setting these parameters, you can train the data. The following is a partial screenshot of 400 trainings, which can be maintained until the final train\_logloss remains the same:

```
[370] train-logloss:0.161581 valid-logloss:0.163908
[16:21:28] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:29] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra nodes, 0 pruned nodes, max_depth=4
[16:21:29] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:29] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:30] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra nodes, 0 pruned nodes, max_depth=4
[16:21:30] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:30] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:31] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra nodes, 0 pruned nodes, max_depth=4
[16:21:31] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra nodes, 0 pruned nodes, max_depth=4
[16:21:31] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[380] train-logloss:0.161581 valid-logloss:0.163908
[16:21:32] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:32] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra nodes, 0 pruned nodes, max_depth=4
[16:21:33] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:33] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:33] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:34] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra nodes, 0 pruned nodes, max_depth=4
[16:21:34] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:35] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:35] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra nodes, 0 pruned nodes, max_depth=4
[16:21:35] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra nodes, 0 pruned nodes, max_depth=4
[390] train-logloss:0.161581 valid-logloss:0.163908
[16:21:36] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:36] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:36] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra nodes, 0 pruned nodes, max_depth=4
[16:21:37] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra nodes, 0 pruned nodes, max_depth=4
[16:21:37] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra nodes, 0 pruned nodes, max_depth=4
[16:21:38] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:38] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 26 extra nodes, 0 pruned nodes, max_depth=4
[16:21:38] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 30 extra nodes, 0 pruned nodes, max_depth=4
[16:21:39] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 28 extra nodes, 0 pruned nodes, max_depth=4
[399] train-logloss:0.161581 valid-logloss:0.163908
```














It can be seen that the value of log-loss is continuously falling, which is in line with our expectations. The final iteration of 400 times reached a satisfactory result of 0.161581.

## 4 Result

### 4.1 Kaggle Competition

Because kaggle is a competition-type platform, you can submit data to compete with other players, and the final results will be displayed on the leaderboard. Therefore, as the most contestant, I also submitted the final final\_submission.csv and other players. Here are the rankings on the leaderboard:

429	▲ 3	SauravKaushik8		0.16151	3	2y
430	▼ 12	liubieqian		0.16154	18	2y
431	▼ 5	DL		0.16155	21	2y
432	▲ 82	Nil Stark		0.16158	42	2d
433	▼ 4	Andreisun		0.16159	107	2y
434	▲ 76	Harw		0.16161	28	2y
435	▲ 3	Winston		0.16162	14	2y
436	▲ 10	Milk&Cereal		0.16166	22	2y
437	—	Bill S		0.16168	33	2y
438	▼ 2	cheng 2		0.16168	14	2y
439	▼ 12	AshutoshNirala		0.16169	36	2y

It can be seen that the final ranking is 432, which is the top 14%, so our model is also relatively successful.

## 4.2 Comparison of Others

At the end of the competition, I also tried to observe the kernel of other players, and compared with them, and found the following deficiencies:

- The feature found is not enough. In the sharing of player Jared [6], there is a feature that can reduce the log-loss value of his kernel by 0-0.1. This feature is mainly based on the frequency of problems and exists in the question. He stored this feature through hashtable and then performed count\_value, so the final score can be greatly improved.

- The second point is the correctness of the data itself. The competitor CPMP [7] mentioned in his sharing how he corrected 25% of the incorrect information in the data generated by the computer. The method is Peter Vorvig's spell checker [8]. Simply put, there are many word vectors, and then the correct words are obtained by comparison. Therefore, if the word in the question is correct, it will be very helpful for subsequent training, especially in my TF-IDF. For example, a wrong word apple will have a large weight. This word will be very important and the error will be very high.

- The last point is the rationality of the algorithm. TF-IDF algorithm is a very mainstream algorithm. There is no doubt about it. Player Philipp [9] added t-SNE charts to TF-IDF. This can find more features, such as the normalized\_share\_count and subsampled\_data he found. . These are useful for subsequent analysis.

## 5 Conclusion

### 5.1 Thoughts and Inspiration

This project has brought me a lot of inspiration and thinking. Generally speaking, big problems need to be reduced to small problems, but still requires rigorous processes:

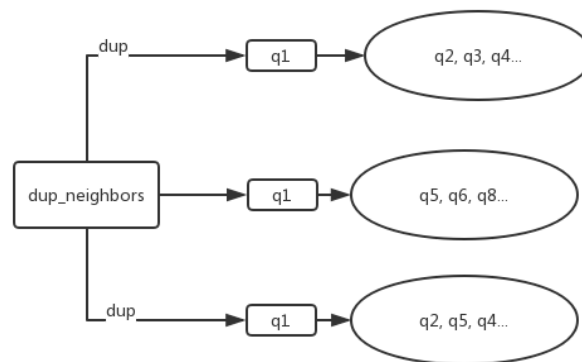
1. What model to choose is PCA, bag of words, LDA, or TF-IDF. Of course, these must be selected according to feature engineering. This has led to my feature engineering exploration. Which feature should be selected, and which feature is useful. ?
2. Similarity calculation methods, including cosine similarity algorithm, or edit distance. Of course, I chose the simplest word share method, because the amount of calculation is smaller, so I need to find a balance between calculation performance and algorithm complexity.

## 5.2 Further Improvement

The results above have stated better practices of other players, so I will do more work in feature engineering and find more useful features to enhance the accuracy of the model. And a correction will be made to the data itself, so that the automatically generated data interference is reduced, and the results are more accurate.



## 5.3 Conjecture Model Optimization

According to udacity's follow-up review suggestions, I have added new structural features. The main idea is to synthesize a sentence pair itself into a picture, which is explained as follows:



Our original sampling method returned an imbalanced dataset with real examples of more duplicate pairs than non-repeated data. Therefore, we supplemented the dataset with negative data. One source of negative data is paired "relevant questions" that, although related to similar topics, are not truly semantically equivalent. So by composing the problem into a graph, we can rediscover new features and improve accuracy.

In addition, I also used Peter Vorvig's spell checker, and the final log-loss value dropped a little to 0.16138:

421	▲ 12	seanappler		0.16124	16	2y
422	▼ 2	anttip		0.16128	63	2y
423	▲ 11	Nil Stark		0.16138	87	2d
424	▲ 6	Sergey Zhitansky		0.16140	20	2y
425	▲ 9	DataDugong		0.16143	8	2y
426	▲ 9	Fred Navruzov		0.16148	21	2y
427	▼ 3	lakomka33		0.16148	41	2y
428	▲ 11	newebug		0.16149	48	2y

Because the previous navie\_submission failed to board the leader\_board because of time limitation, it is a bit regrettable:

[simple\\_xgb.csv](#)

11 days ago by [Nil Stark](#)

[add submission details](#)

0.35544

0.35372



## 6 Reference

- [1]. <https://colab.research.google.com/github/jakevdp/PythonDataScienceHandbook/blob/master/notebooks/05.08-Random-Forests.ipynb#scrollTo=mPaThD8ix1vi>
- [2]. <https://www.kaggle.com/c/quora-question-pairs/data>
- [3]. <https://www.kaggle.com/dansbecker/what-is-log-loss>
- [4]. <https://zh.wikipedia.org/wiki/Tf-idf>
- [5]. <https://www.quora.com/What-makes-xgboost-run-much-faster-than-many-other-implementations-of-gradient-boosting>
- [6]. <https://www.kaggle.com/jturkewitz/magic-features-0-03-gain>
- [7]. <https://www.kaggle.com/cmpmml/spell-checker-using-word2vec>
- [8]. <https://github.com/mmihaltz/word2vec-GoogleNews-vectors>
- [9]. <https://www.kaggle.com/philschmidt/quora-eda-model-selection-roc-pr-plots>