Quora Question Pairs

Luodan Liu

August 31, 2023

Abstract

This research utilizes the LightGBM model to classify text data, addressing the challenge of text matching to identify duplicate questions on Quora. After data preprocessing, a series of features were extracted, such as text length, word match ratio, and TF-IDF weights. The study also evaluates the strengths and weaknesses of the latest NLP model, BERT, in this context. The goal is to enhance the user experience on the platform, ensuring that users can efficiently locate high-quality answers. The accuracy rate of the test dataset reached 70%. Sourced from the renowned data science competition portal, Kaggle.

1 Introduction

In this era of information overload, platforms like Quora play an essential role in knowledge sharing. However, many questions on these platforms are repetitive or highly similar, leading to information redundancy and inefficient retrieval.To address this issue, a system capable of accurately identifying and matching similar questions is required. This would enhance the efficiency of information access, ensuring users can quickly locate high-quality answers.

This research aims to explore the use of advanced machine learning models, specifically the LightGBM model, to deal with the text matching challenge on Quora. We use python and libraries such as lightgbm and sklearn to extract a plethora of features from question texts, which will then be used to train the model.

At the time this issue was raised, the BERT model had not yet emerged. Introduced by Google in 2018, BERT can be applied to various NLP tasks. Compared to traditional machine learning algorithms and other models, BERT offers superior efficiency and accuracy. However, due to its vast number of parameters, training requires substantial computational resources. The performance of BERT in this context will also be assessed.

Ultimately, predictions will be made on the test data, and results will be derived based on the obtained loss values.And the code needed for the article is on my github page: https://github.com/LDANY/kaggle-Quora_Question_Pairs

2 Dataset

2.1 Introduction to data

We can download two files from kaggle: train.csv and test.csv. The training data consists of 404290 observations and 6 features.

- **id** - the id of a training set question pair
- **qid1, qid2** - unique ids of each question (only available in train.csv)
- **question1, question2** - the full text of each question
- **is_duplicate** - the target variable, set to 1 if question1 and question2 have essentially the same meaning, and 0 otherwise.

And the test data consists of 2345796 observations and 3 features: **test_id,question1,question2**.

	id	qid1	qid2	question1	question2	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	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	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
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

Figure 1: Visualisation of data

2.2 Data Preprocessing

For pre-processing, we combine question 1 and question 2 into a list and convert all the text within them into string format, storing it in **"train_qs"**. To facilitate word frequency counting or other operations, sentences are converted to lowercase, and each word is separated. They are stored in the order of appearance in **"words"** (words are not necessarily unique).

```
['what', 'is', 'the', 'step', 'by', 'step', 'guide',
'to', 'invest', 'in', 'share', 'market', 'in',
'india?', 'what', 'is', 'the', 'story', 'of',
'kohinoor', '(koh-i-noor)', 'diamond?', 'how', 'can',
'i', 'increase', 'the', 'speed', 'of', 'my',
'internet', 'connection', 'while', 'using', 'a',
'vpn?', 'why', 'am', 'i', 'mentally', 'very',
'lonely?', 'how', 'can', 'i', 'solve', 'it?', 'which',
'one', 'dissolve', 'in', 'water', 'quikly', 'sugar,',
'salt,', 'methane', 'and', 'carbon', 'di', 'oxide?',
'astrology:', 'i', 'am', 'a', 'capricorn', 'sun',
'cap', 'moon', 'and', 'cap', 'rising...what', 'does',
'that', 'say', 'about', 'me?', 'should', 'i', 'buy',
'tiago?', 'how', 'can', 'i', 'be', 'a', 'good',
'geologist?', 'when', 'do', 'you', 'use', 'シ',
'instead', 'of', 'しろ?', 'motorola', '(company):',
'can', 'i', 'hack'...
```

Figure 2: Visualisation of words

3 Feature Extraction

In many NLP tasks, it's often necessary to extract features from textual questions. In this Quora dataset, I extracted numerous features to help determine whether two questions are similar.

3.1 features related to word sharing

Initially, I plan to compute features related to word sharing and will use the **"word-share"** function for this purpose. I want to exclude questions composed entirely of stop words (these words generally carry no real meaning, such as "and", "the", etc.). If the question consists completely of stop words, the function will simply return a string of all zeros.

```
'0:0:0:0:0:0:0:0'
```

1. word_match

It calculates the tfidf (Term Frequency-Inverse Document Frequency) weights of the same words (shared words) in two sentences. It means the ratio of the tfidf of the same word to the tfidf of all words in the two sentences. When this value is close to 1, it means that the proportion of shared words in the two sentences is very high, on the contrary, when this value is close to 0, it signifies that there are almost no shared important words in the two sentences.

TF-IDF is a weighting method commonly used in NLP to measure the importance of words in a document. Here is the formula:

$$TFIDF(t, d) = TF(t, d) \times IDF(t)$$

Where:
- $TF(t, d)$ represents the term frequency of term t in document d

$$TF(t, d) = \frac{Number\ of\ times\ term\ t\ appears\ in\ document\ d}{Total\ number\ of\ terms\ in\ document\ d} \tag{1}$$

- $IDF(t)$ denotes the inverse document frequency of term t across the entire corpus.

$$IDF(t, D) = \log \frac{Total\ number\ of\ documents\ in\ set\ D}{Number\ of\ documents\ with\ term\ t\ in\ set\ D} \tag{2}$$

In these formulas: - t represents a term. - d represents a document. - D represents the entire collection of documents.

The higher the value of TF-IDF, the greater the importance of the term within the document. TF-IDF aims to reflect the significance of a term to a particular document relative to a set of documents or corpus.

2. tfidf_word_match

This is the ratio of the number of shared words in the two problems to the number of all words in the two problems (minus the number of shared words, since shared words are counted twice). It calculates the degree of sharing between the two problems in terms of the number of words.

3. shared_count

The number of words shared between two sentences.

4. stops1_ratio

Proportion of deactivated words to all words in questions 1.

5. stops2_ratio

Proportion of deactivated words to all words in questions 2.

6. cosine

This is the cosine similarity of the IDF weights of the shared words in the two problems. It represents the degree of similarity between the two problems in terms of IDF weight vectors.

7. shared_2gram

The degree to 2-gram sharing in both questions. Bigrams is a concept in NLP, 2gram refers to two consecutive words or tokens. in text analysis, the expressed meanings between a certain word and phrase may be completely different, or even the opposite, bigrams is used to find a specific combination of relationships between words.

8. words_hamming

The proportion of words in the same position for two questions.

Ultimately, this function returns a string containing all the feature values calculated above.

To store these feature values, I created a new dataframe named x and stored them in x after operating the word-share function on each row of the df (training set + test set).

3.2 features related to sentence length and character

- **diff_len**: the difference between the lengths of two sentences.

	word_match	word_match_2root	tfidf_word_match	shared_count	stops1_ratio	stops2_ratio	shared_2gram	cosine	words_hamming
0	0.38608	0.62136	0.57143	4.00000	1.00000	1.20000	0.41667	0.79519	0.78571
1	0.18088	0.42530	0.18182	2.00000	1.00000	0.33333	0.05263	0.41093	0.07692
2	0.17760	0.42142	0.22222	2.00000	1.33333	1.00000	0.04545	0.34088	0.14286
3	0.00000	0.00000	0.00000	0.00000	1.50000	0.80000	0.00000	0.00000	0.00000
4	0.00000	0.00000	0.00000	0.00000	0.30000	0.40000	0.00000	0.00000	0.07692
5	0.25539	0.50536	0.30769	4.00000	0.87500	0.77778	0.10000	0.48869	0.25000
6	0.00000	0.00000	0.00000	0.00000	1.00000	0.42857	0.00000	0.00000	0.00000
7	0.32292	0.56826	0.33333	1.00000	2.50000	3.50000	0.07143	0.74870	0.11111
8	0.19838	0.44540	0.33333	2.00000	1.00000	1.00000	0.28571	0.31878	0.75000
9	0.25160	0.50160	0.22222	2.00000	0.50000	0.80000	0.06250	0.50634	0.00000
10	0.00000	0.00000	0.00000	0.00000	0.28571	1.28571	0.00000	0.00000	0.00000
11	0.32704	0.57187	0.40000	2.00000	1.25000	1.66667	0.13333	0.72620	0.22222
12	0.50000	0.70711	1.00000	4.00000	0.75000	1.00000	0.30769	1.00000	0.12500
13	0.26823	0.51791	0.40000	2.00000	0.75000	1.00000	0.36364	0.52109	0.71429
14	0.40231	0.63428	0.69231	9.00000	1.00000	1.00000	0.44643	0.79243	0.93103
15	0.15369	0.39204	0.18750	3.00000	0.50000	0.77778	0.06667	0.29778	0.17647
16	0.26836	0.51803	0.33333	1.00000	1.00000	1.00000	0.33333	0.57233	0.75000

Figure 3: features related to word sharing

- **diff_caps**: the difference between the number of uppercase letters in two sentences.
- **diff_len_char**: the difference between the number of characters (excluding spaces) in two sentences.
- **diff_len_word**: the difference between the number of words in two sentences.
- **diff_avg_word**: the difference between the average length of the words in the two sentences.

3.3 Other features

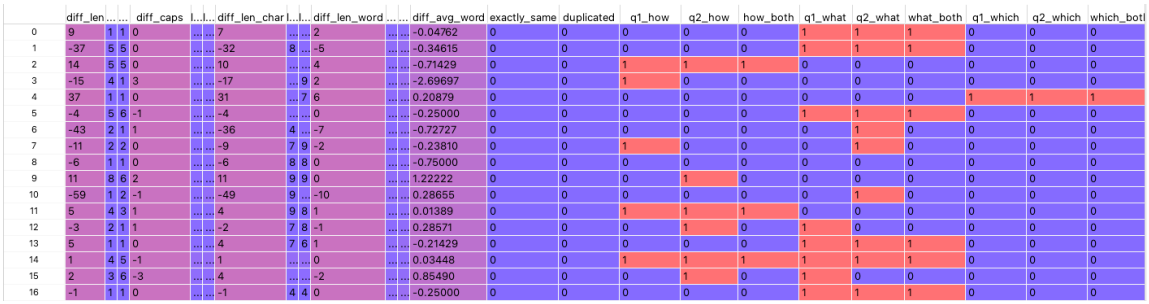
- **exactly_same**: 1 if the two sentences are identical, 0 otherwise.
- **duplicated**: 1 if the two sentence combinations are repetitive, 0 otherwise.
- **add_word_count**: the frequency of occurrence of some common interrogative words (e.g. when, how, etc.) in question 1, question 2, and both, respectively

These features are refined step by step during the problem solving process, after adding more features, the model will be more accurate and get lower loss.

4 LightGBM model training

4.1 Oversampling

In the training dataset, the category imbalance problem arises when one category has far more samples than the others. In this case, the model is likely to be biased



	diff_len			diff_caps			diff_len_char			diff_len_word			diff_avg_word	exactly_same	duplicated	q1_how	q2_how	how_both	q1_what	q2_what	what_both	q1_which	q2_which	which_both
0	9	1	1	0			7			2			-0.04762	0	0	0	0	0	1	1	1	0	0	0
1	-37	5	5	0			-32	8		-5			-0.34615	0	0	0	0	0	1	1	1	0	0	0
2	14	5	5	0			10			4			-0.71429	0	0	1	1	1	0	0	0	0	0	0
3	-15	4	1	3			-17		9	2			-2.69697	0	0	1	0	0	0	0	0	0	0	0
4	37	1	1	0			31		7	6			0.20879	0	0	0	0	0	0	0	0	1	1	1
5	-4	5	6	-1			-4			0			-0.25000	0	0	0	0	0	1	1	1	0	0	0
6	-43	2	1	1			-36	4		-7			-0.72727	0	0	0	0	0	0	1	0	0	0	0
7	-11	2	2	0			-9	7	9	-2			-0.23810	0	0	1	0	0	0	1	0	0	0	0
8	-6	1	1	0			-6	8	8	0			-0.75000	0	0	0	0	0	0	0	0	0	0	0
9	11	8	6	2			11	9	9	0			1.22222	0	0	0	1	0	0	0	0	0	0	0
10	-59	1	2	-1			-49	9		-10			0.28655	0	0	0	0	0	0	1	0	0	0	0
11	5	4	3	1			4	9	8	1			0.01389	0	0	1	1	1	0	0	0	0	0	0
12	-3	2	1	1			-2	7	8	-1			0.28571	0	0	0	1	0	1	0	0	0	0	0
13	5	1	1	0			4	7	6	1			-0.21429	0	0	0	0	0	1	1	1	0	0	0
14	1	4	5	-1			1			0			0.03448	0	0	1	1	1	1	1	1	0	0	0
15	2	3	6	-3			4			-2			0.85490	0	0	0	1	0	1	0	0	0	0	0
16	-1	1	1	0			-1	4	4	0			-0.25000	0	0	0	0	0	1	1	1	0	0	0

Figure 4: features related to sentence length and character and other features

towards predicting the dominant category, so we need to perform oversampling to balance the ratio of positive and negative samples so that the ratio of positive and negative samples in the training set is close to that in the test set (p=0.165). The ratio of positive and negative samples before we start sampling reaches 0.3692. We increase the number of negative category samples by repeatedly adding them until their number reaches the target ratio. This is actually a manual method of achieving oversampling.

```
Oversampling started for proportion: 0.369197853026293
Oversampling done, new proportion: 0.19124366100096607
```

Figure 5: Proportion of positive and negative samples before and after oversampling

4.2 Training

We use LightGBM model for training and prediction. Compared to XGBoost model, LightGBM has faster training efficiency, and lower memory usage, as well as higher accuracy, but it is not very suitable for small datasets, and is very easy to overfitting. We split the original data into a new training set and a validation set, the validation set serves to perform early stopping to avoid overfitting.

Among the parameters we set, we chose the traditional "gbdt" algorithm type, i.e. gradient boosting decision tree. For performance evaluation, we chose the log-loss metric for binary classification problems.

```
[LightGBM] [Info] Number of positive: 119337, number of negative:
   505051
```

```
[LightGBM] [Debug] Dataset::GetMultiBinFromSparseFeatures: sparse
   rate 0.888967
[LightGBM] [Debug] Dataset::GetMultiBinFromAllFeatures: sparse rate
   0.469806
[LightGBM] [Debug] init for col-wise cost 0.061845 seconds, init for
    row-wise cost 0.219248 seconds
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the
   overhead of testing was 0.073861 seconds.
You can set ‘force_row_wise=true‘ to remove the overhead.
And if memory is not enough, you can set ‘force_col_wise=true‘.
[LightGBM] [Debug] Using Sparse Multi-Val Bin
[LightGBM] [Info] Total Bins 4390
[LightGBM] [Info] Number of data points in the train set: 624388,
   number of used features: 45
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.191126 ->
   initscore=-1.442708
[LightGBM] [Info] Start training from score -1.442708
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 22
Loss after iteration 0: 0.457775
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 23
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 22
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 23
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 25
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 24
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 23
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 30
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 23
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 25
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 25
Loss after iteration 10: 0.346674
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 23
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 25

...
Loss after iteration 90: 0.293535
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 30
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 27
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 30
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 26
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 32
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 24
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 29
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 27
[LightGBM] [Debug] Trained a tree with leaves = 512 and depth = 40
```

Based on our outputs, we get the training process and some internal debugging in-

formation of the LightGBM model. Meanwhile, every 10 rounds, we get the training loss value. Finally, we get the loss value **0.293535**.

4.3 Prediction

Eventually, we output the target file:

test_id	is_duplicate
0	0.0027196
1	0.2424595
2	0.0991873
3	1.30E-05
4	0.1421143
5	0.0079764
6	0.3119061
7	0.7123736
8	0.4857922

Figure 6: submission.csv

5 BERT model

The main method I used this time was a traditional machine learning approach that calculates the correlation between texts based on many different feature values. At the time this question was asked in quora, there was no BERT model available. BERT model was proposed in 2018. It was first pre-trained on a large number of texts, and it is very powerful in the field of text matching, text recognition, etc., but at the same time, it has some requirements on the configuration of the device. In BERT model. py file, I train the text with fast pre-processing based on BERT's transformers library. But limited by the equipment, only the first 5000 data are selected for the experiment. I spent a long time on each epoch during the training process, which is extremely inefficient. The first time I got a loss 0.5787, this value will get lower and lower with more epochs. But due to the device, I can't get the final training and validation based on the bert model.

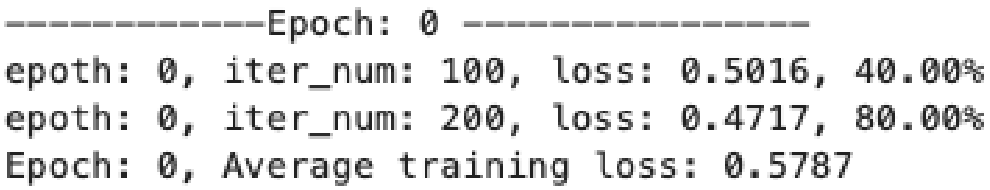


Figure 7: BERT model

6 Conclusion

With this study, we delve into the performance of LightGBM model on text matching task. By extracting features such as words, sentences etc. we get more information from it. Compared to traditional machine learning models, and methods such as Word2Vec, which is not used in this paper, the BERT model is a more accurate and convenient tool, but requires higher device performance.

Overall, this study provides a reliable and efficient tool for predicting similarity between texts. Further optimisation of feature engineering, introduction of more models and further training of the BERT model can be considered to achieve higher prediction accuracy and get higher efficiency when the equipment allows.