Approach for the Quora Question Pairs Challenge

YesOfCourse

July 14, 2017

Abstract

In the Quora Question Pairs Challenge, we were asked to build a model to classify whether question pairs are duplicates or not (multiple versions of the same question). This documents describes our team's solution which can be divided into diffrent parts: Pre-processing, Feature Engineering, Modeling and Post-processing.

Personal details

Team Name: **YesOfCourse**

Members:

- **Liang Pang**
  Email: pl8787@gmail.com
- **Yixing Fan**
  Email: fanyixing111@gmail.com
- **Jianpeng Hou**
  Email: houjp1992@gmail.com
- **Xinyu Yue**
  Email: yuexinyu@software.ict.ac.cn
- **Guocheng Niu**
  Email: oxguocheng@gmail.com

Competition: Quora Question Pairs

Contents

1 Summary

Our solution consisted of four main parts: Pre-processing, Feature Engineering, Modeling and Post-processing. What's more, we developed a light weight Machine Learning framework **FeatWheel** to help us to finish ML jobs, such as feature extraction, feature merging and so on.

In pre-processing, we process the text of data with text cleaning, word stemming, removing stop words and shared words and can form different versions of original data. In feature engineering, we extracted features based on various versions of data. The features can be classified in to three categoriesStatistical Features, NLP Features and Graph Features. In modeling, we build deep models, boosting models (using XGBoost, LightGBM) and linear models (Linear Regression) and build a multi-layer stacking system to ensemble different models together. As we all know, the distribution of the training data and test data are quite different, so we made post-processing on the prediction results. We cut the data into different parts according to the clique size and rescale the results in different parts.

1.1 Flowchart

The flowchart of our method is shown in Fig. 1.

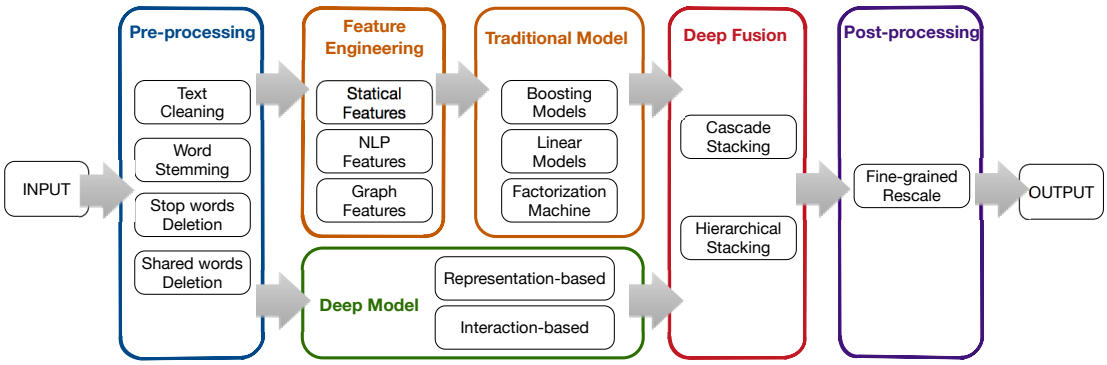

Figure 1: The flowchart of our method.

1.2 Submission

Submissions were evaluated on the log loss between the predicted values and the group truth. In specific, the best single model we have obtained during the competition was an XGBoost model with tree booster of Public LB score **0.12653** and Private LB score **0.13067** (without post-process). Our final submission was a stacking result of multiple models. This submission scored **0.11450** on Public LB and **0.11768** on Private LB (with post-process), ranking **4** out of **3396** teams.

2 Pre-processing

Pre-processing

3 Feature Engineering

Feature Engineering

4 Modeling

4.1 Data Partition

Five-fold cross-validation is performed on this data set. In detail, the original training data set is randomly divided into five parts (S1-S5) each with approximately the same size. In each fold, one part is held-out for validation, another part is held-out for testing and the learning algorithm is trained on the remaining data. The above process is iterated five times so that each part is used as the validation data and test data exactly once, where the averaged metric values out of five runs are reported for the algorithm. Table 1 illustrates how to split the training data set for cross validation.

Table 1: Data Partition for Cross Validation

Fold	Training	Validation	Test
Fold1	S1, S2, S3	S4	S5
Fold2	S2, S3, S4	S5	S1
Fold3	S3, S4, S5	S1	S2
Fold4	S4, S5, S1	S2	S3
Fold5	S5, S1, S2	S3	S4

5 Deep Text Matching Models

In this section, we introduce two categories of deep text matching models []: representation based models and interaction based models. Both of them play importance role in this competition, especially in the stage of stacking.

5.1 Representation Based Models

Representation based model is one of the siamese architecture [], which firstly proposed in face verification task. As shown in Fig. 2, given two pieces of text x and y, representation based model first encode each of them to the independent sentence representations, $R(x)$ and $R'(y)$. Then calculate the similarity score $f(x, y)$ between

these two representations, such as cosine similarity, dot product or feed them to an MLP (multi-layer perception).

$$f(x, y) = g(R(x), R'(y)). \tag{1}$$

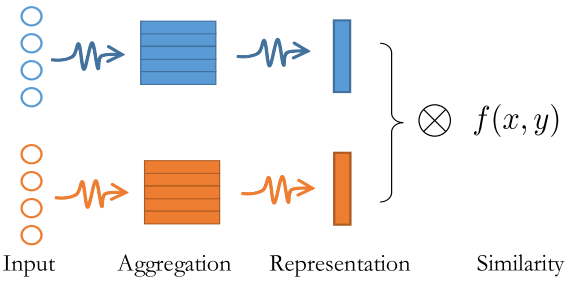

Figure 2: The Siamese Architecture.

The encode function, R and R', can be designed as the CNN (convolutional neural network) or RNN (recurrent neural network), both of them can capture the sequential information of the text.

5.2 Interaction Based Models

Interaction based model treats text matching problem in another way, while it first construct an interaction matrix based on the word representations, then the patterns are extracted base on the interaction matrix. Different from representation based model which focuses on make good sentence representations, interaction based model focuses on construct interaction representations. We can describe this process in the formula below,

$$f(x, y) = g(R^m(M(x, y))), \tag{2}$$

where M denotes the interaction matrix between x and y, R^m denotes the representation of the interaction matrix and the function g is used to generate the similarity score.

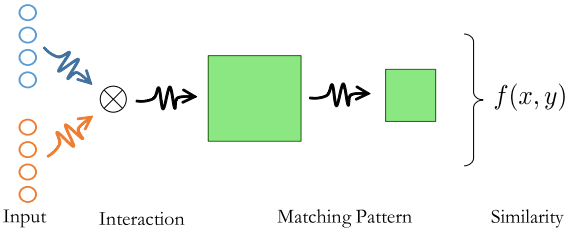

Figure 3: The Interaction Based Architecture.

The model MatchPyramid [] and Match-SRNN [] belong to this category. In this competition, we define a much complex M to construct a interaction tensor (not matrix

because it is 3-dimension) and then apply 2D-GRU, which first be introduced in Match-SRNN.

5.3 Best Single Deep Text Matching Model

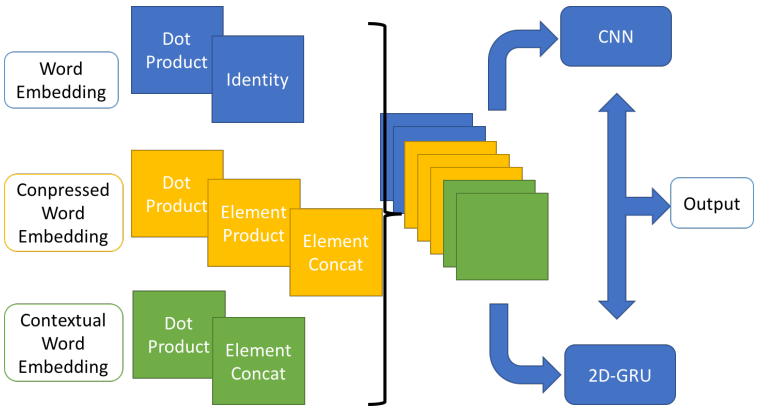

Figure 4: The architecture of the best single deep text matching model.

In order to gain a better performance, we find that a structure level combination is useful. As shown in Figure. 4, we first construct multiple channels of matching matrices; then concatenate them into a 3-dimensional tensor as the input of the following network; finally, apply the hierarchical convolutional operation and the 2-dimensional GRU operation on this matching tensor respectively.

5.3.1 Matching Tensor

We use three types of data sources, namely original word embeddings, compressed word embeddings and contextual word embeddings.

Word Embedding: The word embeddings are pre-trained using Word2Vec. We use released 300-dimensional word embeddings, such as Google-News and Glove-640B.

Compressed Word Embedding: We use a linear function to compressed original word embedding into a low dimension. For example, we apply a linear transform matrix $W(300 \times 20)$ to compress 300-dimensional embedding into 20-dimensional embedding.

Contextual Word Embedding: Considering the contextual information around current word, we construct a contextual word embedding. In order to obtain the contextual information, we follow the structure proposed in MV-LSTM []. A bi-directional LSTM is adopted to build contextual word embeddings.

Based on these three data sources, we construct multiple channels matching matrix, called matching tensor, considering different similarity functions. In this single model, we use identity function, dot product function, element product function and element concatenate function.

5.3.2 Patterns Extraction and Aggregation

After constructing matching tensor which contains fine-grained matching signals, we need to extract or aggregate matching patterns from the matching tensors. As we have mentioned, MatchPyramid and Match-SRNN provide two kinds of aspect to tackle this issue. In MatchPyramid, a hierarchical convolutional operation is applied to extract matching signals patterns; while in Match-SRNN, a spacial GRU operation is applied to find a matching alignment and aggregate the matching signals.

Then, we concatenate the outputs of CNN and 2D-GRU, and feed them into the following full connected layers, in order to obtain the final matching score (the probability of the question duplication).

6 Deep Fusion

6.1 Single Stacking

Stacking is a way of combining multiple models. The usual method for stacking is split the training set into two disjoint sets. Then train several base learners on the first part and test the base learners on the second part. Finally, using the predictions from second part as the input, and the correct responses as the output, train a higher level learner.

As you can see, the traditional stacking method use only part of the data to construct higher level learner. In order to use entire data while stacking, we proposed a novel way. The algorithm described as follows:

1. Randomly split offline data set into five subsets (5-fold CV for example).
2. Fit and make predictions based on this split with specified model 5 times: select 3 parts for training, 1 for validation and 1 for test every time.
3. Concat the prediction results on test data and use it as a new feature for offline data set.
4. Make prediction for online data 5 times with models generated in steps 2 and collect the results.
5. Random select a prediction result for every data points in online data set and form the new feature for online data set.
6. Using the new feature generated in steps 3 and 5 together with original features train a higher level learner and predict for online data set.

Alg. 6.1 described how to stack a specified model into a higher level model. The flowchart of the Algorithm is shown in Fig. 5.

6.2 Cascade Stacking

With the single-stacking algorithm proposed in Chap. 6.1, we can construct a more complex stacking structure named Cascade Stacking. The framework of the Cascade Stacking is shown in Fig. 6. The steps of the cascade stacking described as follows:

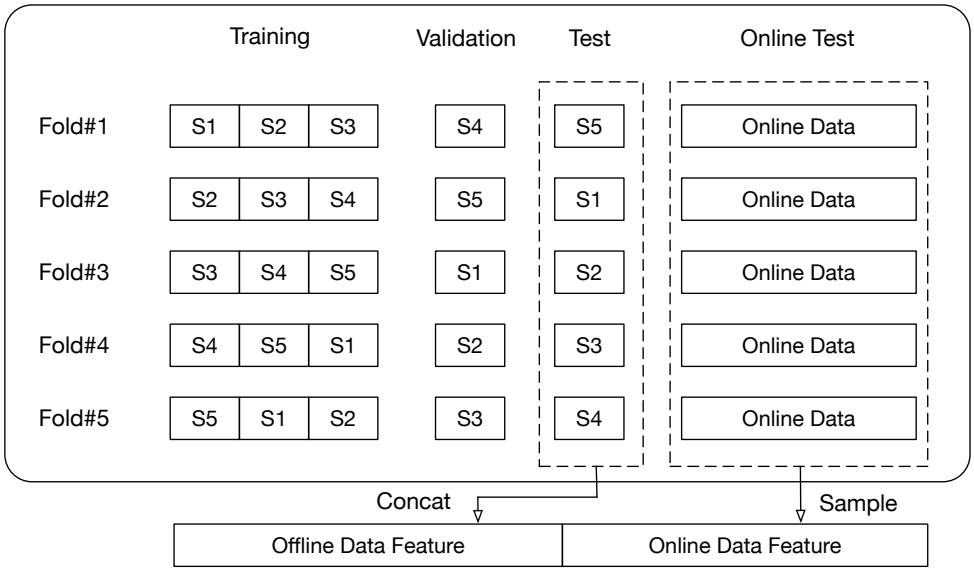

Figure 5: The flowchart of single stacking.

1. Fit and make predictions based on original features with diverse models, such as XGBoost, LightGBM and Deep Models.
2. Convert the prediction results into new features through single-stacking algorithm.
3. Fit and make predictions based on all of the transformed features and original features with diverse models, such as XGBoost, LightGBM.
4. Fit and make predictions with Linear Regression Models based on all of the transformed features and without original features.
5. Repeat steps 3 and 4 until prediction score is converged.
6. Generate the final prediction result based on all of previous transformed features and original features with XGBoost.

6.3 Hierarchical Stacking

Although Cascade Stacking method can improve the performance obviously compared to other stacking methods, it has some problems which are difficult to deal with:

1. When the original features are increased, we must rebuild all of the models generated during cascade stacking from the bottom to the top. This rebuilding process is extremely time consuming.
2. During the number of layers increased gradually, the importance of the original features is reduced. The construction of the model is mainly based on the transformed features.

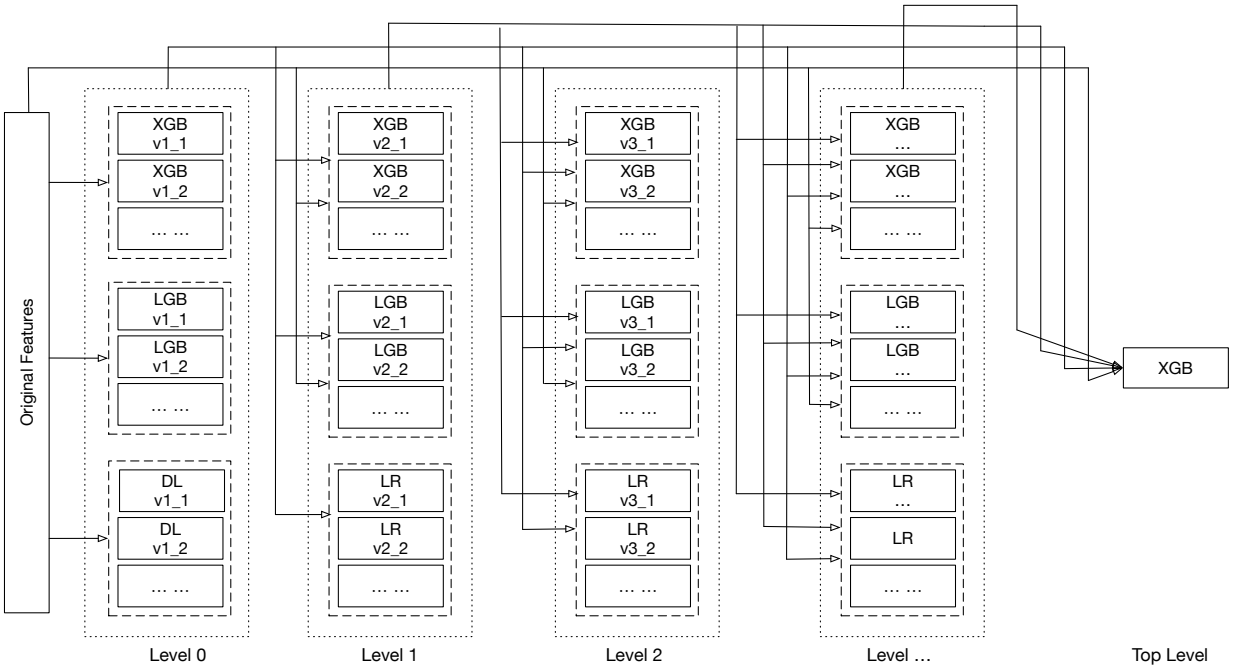

Figure 6: The framework of cascade stacking.

In order to deal with those problems, we proposed another interesting structure of stacking named **Hierarchical Stacking**. The framework of the Hierarchical Stacking is shown in Fig. 7. The steps of the hierarchical stacking described as follows:

1. Fit and make predictions based on original features with diverse models, such as XGBoost, LightGBM and Deep Models.
2. Convert the prediction results into new features through single-stacking algorithm.
3. Fit and make predictions based on last layer transformed features and new extracted features with diverse models, such as XGBoost, LightGBM.
4. Fit and make predictions with Linear Regression Models based on last layer transformed features and without any original features.
5. Repeat steps 3 and 4 until prediction score is converged.
6. Generate the final prediction result based on last layer transformed features with XGBoost.

7 Post-processing

As the distribution of the offline data set (train.csv) and online data set (test.csv) are quite different, we did a post-processing procedure for the prediction results of online

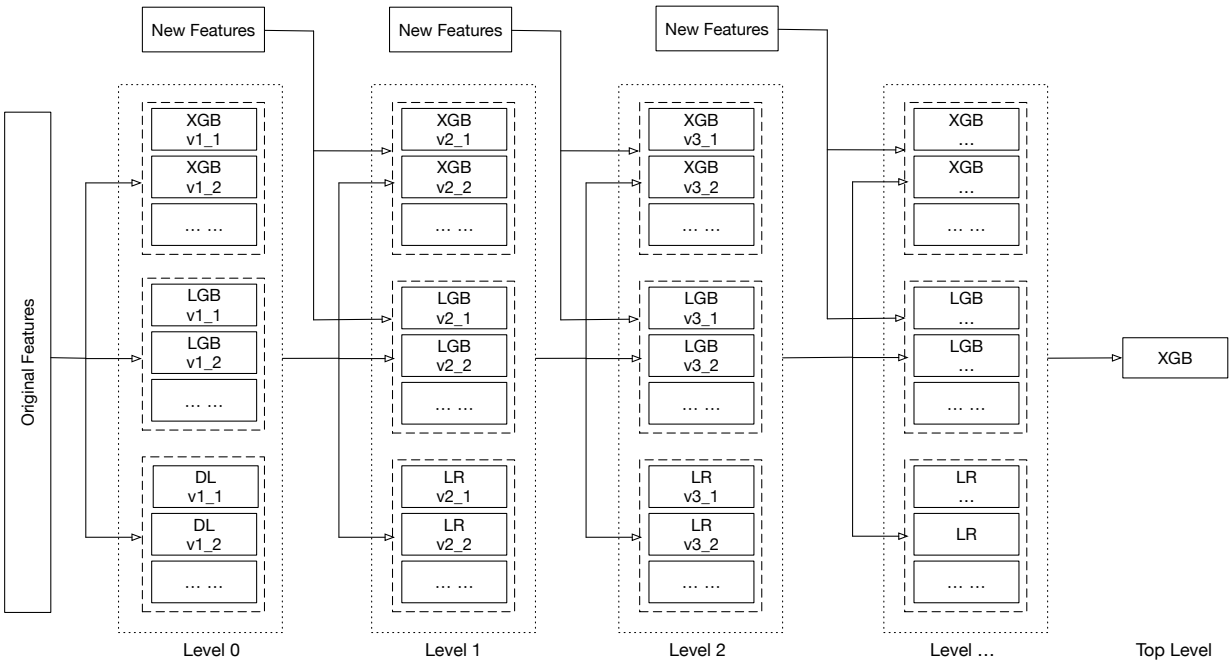

Figure 7: The framework of hierarchical stacking.

data set. We cut the data set into multiple sets according to the clique size and rescale the results in different parts separately.

7.1 Split

In order to reduce the impact of inconsistent between offline data set and online data set, we must split the data set into different parts. The standards of the division is the feature 'graph_edge_max_clique_size' and the feature 'graph_edge_cc_size'. The meaning of the features is as follows:

- **graph_edge_max_clique_size** (mc_size): Number of nodes contained in the largest clique of the edge.
- **graph_edge_cc_size** (cc_size): Number of nodes contained in the connected component of the edge

With these features, we can divide the data set into four parts and the size of each part is described in Table 2.

However, there are lots of fake data points in the online data set (test.csv). So the size of different parts for online data set is not true. We need to solve the nonlinear equations to obtain the true data ratio and positive samples ratio of different parts for online data.

Suppose the online data is cut into two parts. Let k_1, k_2 denote the data ratio of different parts for online data, and let r_1, r_2 be the positive samples ratio of different parts for online data. We can assigned the same value as prediction results for different

Table 2: Data Partition for Post-processing

	$mc\_size < 3$ $cc\_size < 3$	$mc\_size < 3$ $cc\_size \geq 3$	$mc\_size = 3$	$mc\_size > 3$
train.csv	118,065	173,164	40,482	72,579
test.csv	1,933,597	344,283	23,841	44,075

parts v_1, v_2. Then we can get the *score* back based on these values. In this way , you can get an equation as follows:

$$k_1 * (r_1 * log(v_1) + (1 - r_1) * log(1 - v_1)) + k_2 * (r_2 * log(v_2) + (1 - r_2) * log(1 - v_2)) = -score \quad (3)$$

Then we can change the value of v_1, v_2 and can get the corresponding scores. Using those values, we can construct nonlinear equations. Based on equations, we will known the value of k_1, k_2, r_1, r_2.

In this competition, we have more variables to be solved and the solution is the same.

Finally, we get the data ratio and positive samples ratio of different parts for data sets. The data ratio is shown in Table 3. The positive samples ratio is shown in Table 4.

Table 3: Data Ratio of Different Parts

	$mc\_size < 3$ $cc\_size < 3$	$mc\_size < 3$ $cc\_size \geq 3$	$mc\_size = 3$	$mc\_size > 3$
train.csv	29.20%	42.83%	10.01%	17.95%
test.csv	30.50%	52.19%	6.08%	11.24%

Table 4: Positive Samples Ratio of Different Parts

	$mc\_size < 3$ $cc\_size < 3$	$mc\_size < 3$ $cc\_size \geq 3$	$mc\_size = 3$	$mc\_size > 3$
train.csv	23.35%	14.95%	62.32%	97.26%
test.csv	5.74%	4.50%	40.88%	96.50%

7.2 Rescale

As we get the positive samples ratio of different parts, we can rescale the prediction results in different parts seperately based on those prior knowledge.

For the specified part of data set, let $r_{offline}$ denote the positive samples rate on offline data set and let r_{online} be the positive samples rate on online data set. The original values of prediction is y, then we can get the rescaled prediction values as follows:

$$y_{rescale} = \frac{\frac{t_{test}}{t_{train}} * y}{\frac{t_{test}}{t_{train}} * y + \frac{1-t_{test}}{1-t_{train}} * (1-y)} \tag{4}$$

8 ML Framework

We developed a light weight Machine Learning framework **FeatWheel** to help us finishing ML jobs, such as feature extraction, feature merging and so on. We will introduce the framework in this section.

8.1 Characteristics

- **Simple**: With FeatWheel, you can focus on extracting features. All you need to do is to generate feature files in specified format and write a config file.
- **Flexible**: You can specify the features you need and FeatWheel can help you finish the procedure of feature merging. You can also store the merging result on the disk to speed up the next loading process.
- **Efficient**: If you want to split the data set into training, validation and test data, you just need to generate index files. You don't need make any changes to feature files and data set files.
- **Reliable**: Each time of the model training and prediction will generate a separate output directory and will keep the operating environment of this running. This is very useful for the reproduction of results.

8.2 Structure

```
|- project                  # root directory of project
|    |- bin                 # executable file directory
|    |    |- feature.py
|    |    |- preprocess.py
|    |    |- model.py
|    |    |- postprocess.py
|    |- conf                # configuration file directory
|    |    |- python.conf
|    |- daa                 # data file directory
|    |    |- source         # input data source
|    |    |- feature        # feature file directory
|    |    |- index          # index file directory
|    |    |- label          # label file directory
|    |    |- out            # output directory
|    |- log                 # log file directory
```

8.3 Format

The storage format of features is a sparse matrix. Different features can be stored in different files and the format of each file is as follows:

```
row_number<space>column_number
feature_id<colon>feature_value<space>feature_id<colon>feature_value ...
```

This format of feature files is named as **SMAT**. If the feature value is empty, it can be omitted. The example is as follows:

```
2 4
0:1 2:2
1:2 3:1 4:5.2
```

9 Acknowledgement

We would like to thank the Kaggle team and Quora for organizing this competition. Thanks to every member of the team for the day and night.