EE 5239 – Introduction to Nonlinear Optimization

Final Project Report

Duplicate Classification on Quora Question Pairs

Zitao Yang	yang6887@umn.edu
Kai Ye	ye000064@umn.edu

Dec 18th, 2019

Abstract
The project aims to classify whether the question pairs on Quora are duplicate or not. First, the data is preprocessed to clean anomalies and make it easier to be analyzed by NLP. Features are extracted in four types from the lemmatized question sets. Each type of features is analyzed and respectively and then merged to generate classification models using different algorithms like SVM and LR. The final prediction performance of the model is evaluated with log-loss error and F1-score metric. The overall features provide the best prediction result with log-loss=0.17035 on the test set and F_1-score=0.9144 on cross-validation, which indicates a convincing scheme to solve similar near-duplicate detection problems on other online forums.

I. Problem Introduction

The Internet has been one of the most transformative and fast-growing technologies in the past decades. With the popularity of smart devices and convenient access to the Internet, online forums have become an important source of information with topical discussions. Since we have technical and daily problems all the time, it's easy to find some related topics and questions on these forums and then find useful answers from posted discussions. However, while online discussion sites become increasingly popular, lots of similar topics and questions raised by different people come up on the forums and it is difficult to locate the best answer we want. As a result, classifying duplicate topics and question pairs and merging them will make it easier for the community to locate the most suitable questions and answers. Inspired from a problem on Kaggle[1], the project is meant to develop an effective scheme to distinguish pairs of questions with the same meaning on Quora. And the method can also be adapted to solve similar issues on other online discussion forums like Reddit and GitHub.

II. Literature Review

To classify duplicate questions, the first step is to obtain the semantics meaning of questions based on natural language processing (NLP). V. Gupta introduced several classical text mining techniques in his survey on text mining applications [1]. Multiple text mining frameworks can be established for different purposes like keyword extraction, summarization, and categorization. The primary objective of text mining is to identify distinct facts and relations in text. In the question answering (QA) system dealing with finding the best solution to a given problem, n-gram extraction is used to refine the keywords and text crawlers are implemented to select the most suitable candidate answer. Pre-trained word embedding tools such as GloVe [2] and Google's Word2vec are also widely used in extract semantics features. The text is tokenized and lemmatized to its initial form. The tokens from the text are integrated into a bag of words using dictionary functions. Each word has already been mapped as a vector of real numbers. The sum of the embedding vectors of the bag can be used to compare the similarities between questions.

Structure-based features are introduced by K. Muthmann et al. in their work on near-duplicate detection for online forums [3]. Since the posting and threads in a forum are usually interconnected. Hyper-graph is built to deal with thread grouping and internal referral and answering postings. The high correlation between near-duplicate candidates is reasonable while the prediction performance is moderate due to insufficient matching strategies and the unbalance between duplicate and non-duplicate cases.

As for research concerning question-answering (QA) forums, Y. Wu et al. proposed an algorithm to automatically detect the near-duplicate threads based on similarity [4]. A dataset including 3 million questions from Baidu Zhidao, a Chinese QA forum, is used in the experiment.

A detailed description of the given question, the best available answer, and the question itself are combined and treated as a thread. Jaccard distance and distributed index are used to measure the pairwise similarities among the forum threads. A simple linear regression model using pairwise similarities from three parts of question threads is implemented and the best F1-score is 66.22% with precision=94.98% and recall=50.82%.

T. Addair tested a series of deep learning methods including the conventional neural network (CNN), long short-term memory network (LSTM), and a hybrid model of these two on another dataset from Quora [5]. The Siamese neural network architecture is adopted along with GloVe for word embedding. Shingling algorithm is implemented to compare the similarity between questions using their text only. Each question is tokenized and represented as a bag of four "shingles". The Jaccard similarity between two bags of each question pair is calculated. And among the models he built, the LSTM model gives the best prediction results with F1-score = 0.757.

Based on the same public dataset from Quora, C. Saedi et al. implement three kinds of models on a balanced subset with 300k question pairs: Jaccard similarity indexing, Support vector machine (SVM) based on n-grams (n=4) embedding, and deep convolutional neural networks [6]. The models are tested on subsets with increasing size. The accuracy is measured in the F1-score metric and the best result is 77.64% using DCNN. The average performance of SVM is around 68%.

III. Proposed Approach

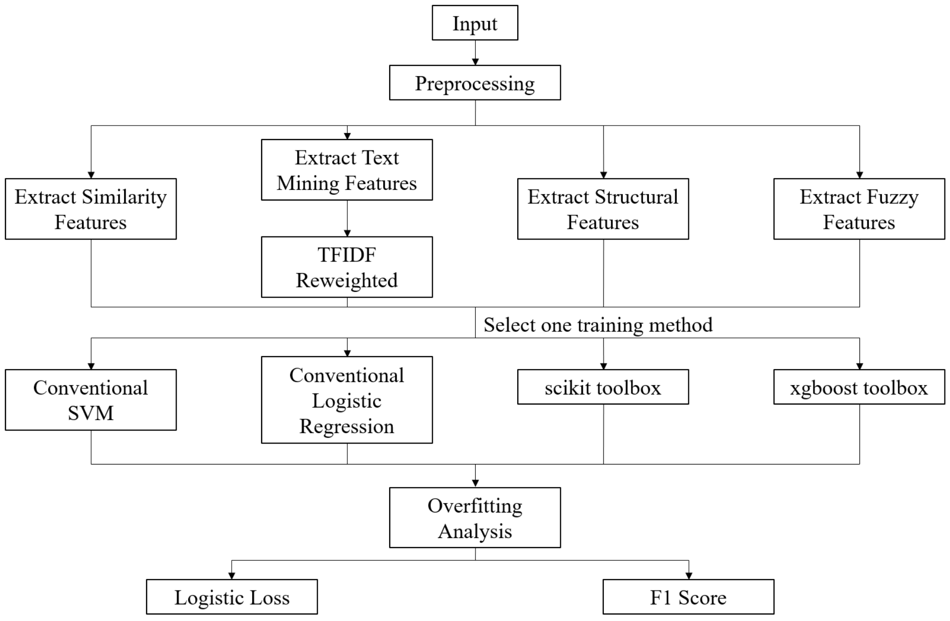

Figure 1. Flow Chart of the Entire Process

Similar to conventional machine learning problems based on NLP, this project proposes to solve the problem as the flow chart shown in Figure 1.

A. Preprocessing of data

Before all the processing steps, we first take a brief overview of the given data sets. There are 404290 pairs of questions in the train set and 2345796 pairs in the test set. All the questions in the train set are genuinely from Quora while the questions in the test set were supplemented with computer-generated question pairs by Kaggle for anti-cheating reasons. Around 63% of questions in the training set are non-duplicate and the others are duplicates. Thus the prediction performance of our models on the training set should be higher than 63%, otherwise, we can achieve it by setting all labels as non-duplicate.

The number of non-stop words in the questions ranges from 0 to 248, and the length of 95% of the questions is fewer than 50 words. The main part of the question length distribution is plotted in Figure 2.

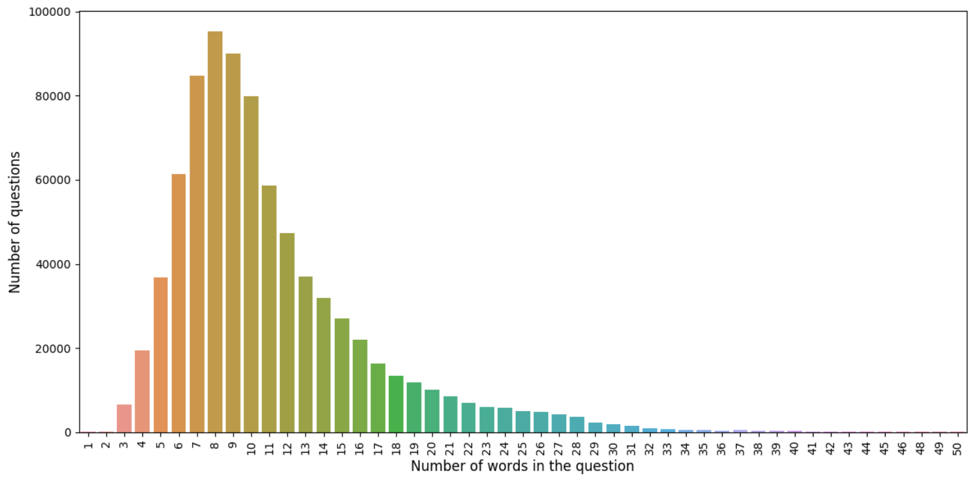

Figure 2. The distribution of questions with word-length of 0 to 50

The questions are preprocessed first to replace all the abbreviations for better semantics analysis. A spelling checker is implemented based on the word rank in Google's word2vec to detect possible mistakes. And linguistic preprocess is conducted before using word embedding.

B. Feature extraction

a. Classical Text Mining

For each question pair, some shallow attributes are checked first, such as if the questions start the same, or ends the same, the absolute difference of lengths, and the average length of questions. To measure the similarity, the number of shared words is counted and normalized over the total number of unique words in each question. To get some intuitionistic insights into the pairwise similarity, the ratio of common words calculated by the number of shared words normalized on all the unique words is plotted in Figure 3 with a 95% confidence interval.

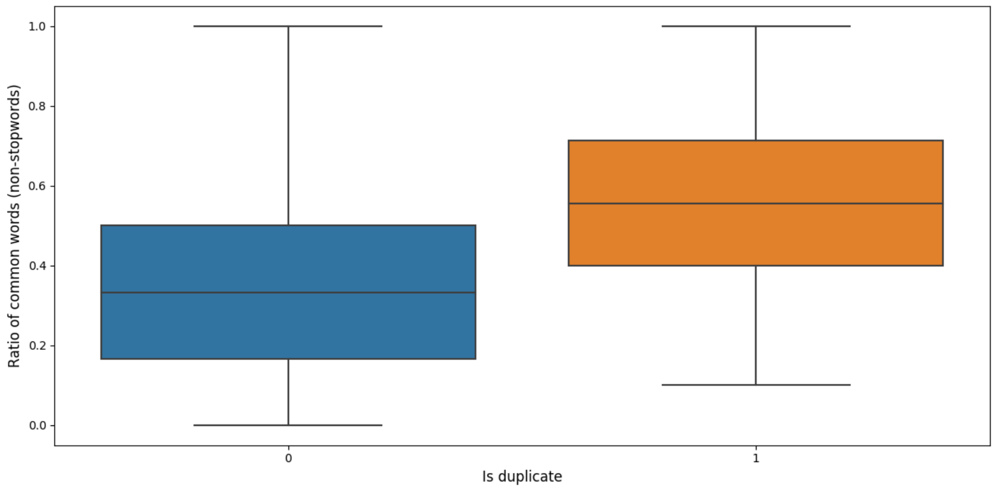

Figure 3. The ratio of common words for duplicate and non-duplicate question pairs

As shown in the figure, nearly all of the question pairs have common words and the duplicate questions have a higher expected ratio of common words. Thus the number of common words is useful and might not be very effective. Furthermore, words can be divided into stop-words and non-stop words for better semantics analysis. For all the features listed above, since there is no order within each question pair, the number of common words is normalized over the unique words in each question and resorted for better generalization.

And term frequency-inverse document frequency (TF-IDF) is adopted to reweigh the word share [7]. The TF-IDF value is positively proportional to the frequency a word appears in the document while it offsets the number of documents containing the word. In other words, we would care more about uncommon shared words rather than words like "the", "is" and so on. This is implemented by assigning weights negatively proportional to the frequency of words in all questions, and the reweighted common word share becomes more convincing. A sufficient-large smoothing coefficient is added to prevent overfitting effects of extremely rare words.

Another method to measure the similarity besides counting common words is the Jaccard coefficient mentioned in the last part. Given question A&B in the form of sets of unique words, the Jaccard similarity coefficient (JSC) can be calculated as $JSC(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A|+|B|-|A \cap B|}$, where $|\cdot|$ means the number of elements in the set.
In other words, it's defined as the number of elements that are in either set but not shared by both, divided by the total number of unique elements. The Dice similarity coefficient (DSC), which is defined as $DSC(A,B) = \frac{2|A \cap B|}{|A|+|B|}$, is also used to measure the similarity between tokenized questions.

While JSC and DSC deal with similarity in the word level, the Levenshtein distance introduces a new method to compare similarity in a character scale. The Levenshtein distance counts the number of character-level substitutions to transform a given string to the other one. Character-level substitutions consist of adding, deleting and substituting characters with another one. Given question 1&2 with length a and b respectively, it can be obtained by recursively executing the following formula until the goal is achieved:

$$lev_{a,b}(i,j) \begin{cases} \max(i,j) & if \min(i,j) = 0, \\ min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1_{(a_i \neq b_j)} \end{cases} & otherwise \end{cases}$$

Besides the similarity metrics mentioned above, the "*Fuzzywuzzy*" package [8] implemented based on python's *difflib* offers a series of fuzzy similarity measurement metrics as listed below:
Q-ratio = $\frac{length\ of\ \ longest\ substring}{length\ of\ the\ longer\ string}$, all filtered with ASCII characters only.
Partial ratio = $\frac{length\ of\ \ shorter\ string}{length\ of\ the\ maximum\ substring}$,
Token sort ratio = Q-ratio after all the words are tokenized and sorted
Token set ratio = max($\frac{|A \cap B|}{|A|}$, $\frac{|A \cap B|}{|B|}$, $\frac{|A|}{|B|}$), assuming that $|A| < |B|$.

These functions are all based on the Levenshtein distance at the character level. And these methods gives detailed but fuzzy features about the pairwise similarity matching, which is a reasonable trial while the effects are not guaranteed.

b. Structural features

With the help of the pre-labeling of online forums, structural features are useful when we are dealing with questions on a certain topic. Graphs can be built based on the inner correlations between questions in the same class. For this problem, one question is compared to multiple questions, in other words, it may exist in several different question pairs. Based on this property, we established a graph from the edges between pairs of questions. In order to get more comprehensive included in the graph structure, we concatenated the training set and the test set thus edges from all question pairs are covered in the graph. And the graph is implemented using the "*networkx*" [9] package in python.

Since graph is a very informative data structure, different kinds of features can be extracted from the graph and its subgraphs such as number of neighbors, the neighbors of question neighbors, and so on. In this project, we built the graph in an undirected form and only first order neighbors are considered since counting higher order neighbors may generate other complicated issues like loops in the graphs.

By counting the number of first-order neighbors, we can get the frequency feature that shows how many times is this question compared with other questions. The distribution of the number of first-order neighbors is plotted in Figure 4 along with the average value of the duplicate indicator. The plot implies that a question that is compared often in question pairs is more likely to be duplicate. Due to the anomaly at the left end, it would be better to set a upper limit to avoid overfitting.

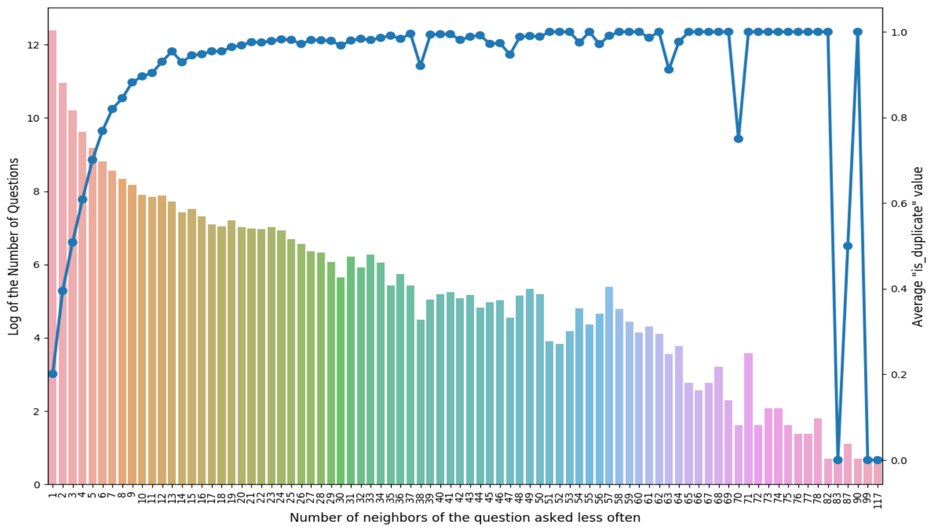

Figure 4. The distribution of the number of neighbors and the mean duplicate-indicator value

Also, the intersection of the neighbors of two questions is also a useful indicator since questions that're compared with more common questions have more chance to be duplicate. The intersection features are normalized over number of neighbors of each problem and sorted along with the frequency features. Similar distribution plot is shown below for the intersection feature. The predictability of this feature is effective and convincing. However, a upper limit on the number of common neighbors would also be helpful to avoid the overfitting effects from the long tail shown in the figure.

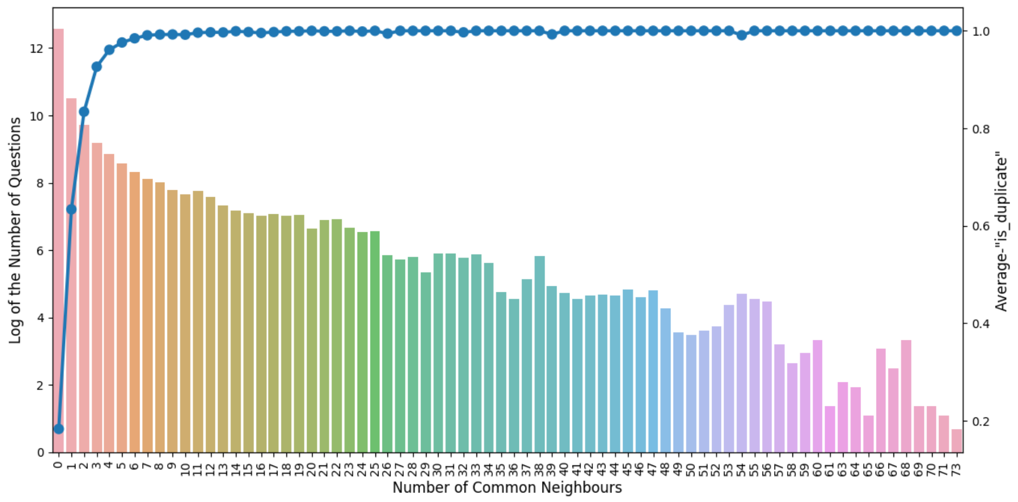

Figure 5. . The distribution of the number of shared neighbors and the mean duplicate-indicator value

The k-core degeneracy of the graph is also considered. The k-core of a graph is a maximal subgraph that contains nodes of degree k or more, in other words, all the vertices in the k-core subgraph are connected to at least k vertices. By analyzing the maximum k-degeneracy of nodes, it's useful to represent the effects of clustering in the graph built from questions. The maximum k-degeneracy is calculated with an upper limit of ten and resorted in each question pair for normalization.

c. Word Embedding

Word embedding tools like Word2vec and GloVe are widely utilized in related projects using deep learning. Each word in the pretrained embedding dictionary can be represented as a vector. Each question, which is a bag of words, can be treated as the sum of the word vectors. Then the similarity metrics used before can be applied on each vector pair instead of text. Since deep learning frameworks are not implemented in this project, the efficiency of word embedding is quite moderate and will not be discussed in this report.

C. Classification

a. Support Vector Machine Method

To apply Support Vector Machine to this problem. First set the initial value of λ. Then we can maximize the Lagrange Multiplier function

$$L(\lambda) = \sum_i \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i\lambda_j\, b_i b_j {a_i}^T a_j$$

$$s.t.\, 0 \le \lambda_i \le c\ \forall i$$

The problem can be also regarded as

$$\min \frac{1}{2}\sum_{i,j} \lambda_i\lambda_j\, b_i b_j {a_i}^T a_j - \sum_i \lambda_i$$

Then apply the gradient descent method to find the optimized λ.

$$\nabla L(\lambda) = 1 - \sum_i \lambda_i\, b_i a_i$$

Then we can find the optimized x

$$x = \sum_i \lambda_i \, b_i a_i$$

b. Logistic Regression Method

The loss function is

$$L(x) = \frac{1}{M}\sum_{i=1}^{M} \ln\left(1 + e^{-b_i x^T a_i}\right)$$

Then we can find the gradient as

$$\nabla L(x) = -\frac{1}{M}\sum_{i=1}^{M} \frac{b_i a_i}{1 + e^{b_i a_i{}^T x}}$$

If we use the constant step size method, we can use the Lipschitz Condition to find a proper step size.

$$\|\nabla^2 L(x)\| = \frac{1}{M}\sum_{i=1}^{M} \frac{b_i{}^2 a_i a_i{}^T e^{b_i a_i{}^T x}}{\left(1 + e^{b_i a_i{}^T x}\right)^2}$$

$$\|\nabla^2 L(x)\| \le \frac{1}{M}\sum_{i=1}^{M} \|a_i\|^2$$

c. Long Short-Term Memory Networks (LSTM)

Recurrent neural networks are known for their ability to generate text. This can allow the output of the neural network to be free-form text. LSTM is a special RNN algorithm that enables the neural networks have memory and decide what to memorize and what need to forget. The inner structure consists of input gate, forget gate and output gate. An LSTM can be trained on a textual document, and output new text that appears to be of the same form as the training data set. Typically, LSTM will learn human grammar from the source document, which can help us get the semantic idea of each question on Quora.

IV. Results and discussion

A. Training Results

Conventional SVM Algorithm, referring to *SVM.py*

x = [9.7753e-05, 6.22648e-05, -3.17637e-22, 0, 6.51698e-05, 2.13364e-05, 7.7728e-05, 6.2424e-05, 1.05879e-22, -7.41154e-22, 7.71341e-05, 5.22952e-05, 0, 3.48621e-05, 0.00120796, 0.00119053, 0.00689912, 0.00620799, 0.00580941, 0.00790402, 5.19612e-05, 0, 0.000586548, 0.000907855, 0.000145954, 0.00181571, 0.00234619]

Applying Ax to obtain the predicted value, most of the values are greater than 1, we will discuss the result in the discussion section.

Conventional Logistic Regression Algorithm, referring to *LogRegression.py*

x = [0.000109117, 8.57657e-05, 0.000105294, 8.06794e-05, 0.00010346, 8.16338e-05, 8.98024e-05, 7.22269e-05, 3.19839e-05, 2.66639e-05, 8.93962e-05, 7.13238e-05, 0.000101085, 7.89831e-05, 0.000459552, 0.0017264, 0.013646, 0.0118172, 0.0115041, 0.01208, 7.30727e-05, 2.03169e-05, 0.00022123, 0.000386326, 7.70101e-05, 0.00123265, 0.00184249]

Applying Ax to obtain the predicted value, most of the values are greater than 1, we will discuss the result in the discussion section.

Python Built-in scikit function, referring to *scikit_LR.py*

The *sklearn.svm.libsvm.predict_proba()* function can predict probability that the label is 0 or 1, it is C language based SVM model.
Logistic Loss value generated by Kaggle submission
Classical Feature Result: 0.46133
Similarity Feature Result: 0.48978
Structural Feature Result: 0.28619
TFIDF Feature Result: 0.47624
Overall Feature Result: 0.22698

Python Built-in xgboost function, referring to *XGB.py*
Logistic Loss value generated by Kaggle submission
Classical Feature Result: 0.34391
Similarity Feature Result: 0.39484
Structural Feature Result: 0.22615
TFIDF Feature Result: 0.44972
Overall Feature Result: 0.17035
F1 Value generated by *sklearn_metric*
Classical Feature Result: 0.8003628489794872
Similarity Feature Result: 0.8106702199900063
Structural Feature Result: 0.8993388768594088
TFIDF Feature Result: 0.8086138195236325
Overall Feature Result: 0.9143871157862369
The prediction performance is calculated in logistic-loss metric as required by Kaggle.

B. Evaluation

a. Logistic Loss Function

As mentioned at the logistic regression algorithm, the logistic loss formula is $\frac{1}{M}\sum_{i=1}^{M} \ln(1 + e^{-b_i x^T a_i})$. Since there is no truth data of the testing csv file, we need to upload our predicted result to Kaggle and get the value.

b. F1 Score

For comparison with previous related works, the results are also calculated in F1-score metric given by [Accuracy Rate Function].
The classifier is trained using 60% training data and the performance of the prediction on the rest 40% data is calculated using the F$_1$-score metric, which is given by $F_1 = \left(\frac{2}{recall^{-1}+precision^{-1}}\right) = 2 \cdot \frac{precision \cdot recall}{precision+recall}$. In this equation, the precision P is the number of correctly classified samples divided by the number of all classified samples returned by the classifier. The recall R is the number of correctly classifier samples divided by the number of all samples that should be in this class. F$_1$-score ranges from 0 in the worst case to 1 when both precision and recall are perfect.

c. Overfitting Analysis

Applying logistic regression in R, observe the result to check the credibility of the model.
Result:
```
Coefficients:
                        Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)          -2.570e-01  4.494e-03 -57.198  < 2e-16 ***
cwc_min               2.762e-01  8.734e-03  31.621  < 2e-16 ***
cwc_max               3.192e-02  1.127e-02   2.832  0.00463 **
csc_min               1.341e-02  5.260e-03   2.549  0.01079 *
csc_max              -6.083e-02  7.711e-03  -7.889 3.06e-15 ***
ctc_min               1.896e-01  1.238e-02  15.319  < 2e-16 ***
ctc_max              -4.263e-01  1.547e-02 -27.559  < 2e-16 ***
cww_min              -2.117e-01  2.250e-02  -9.405  < 2e-16 ***
cww_max              -4.545e-01  2.620e-02 -17.347  < 2e-16 ***
csw_min               4.754e-01  1.434e-02  33.149  < 2e-16 ***
csw_max              -5.683e-01  1.647e-02 -34.496  < 2e-16 ***
ctw_min               2.659e-01  2.308e-02  11.523  < 2e-16 ***
ctw_max               6.421e-01  2.616e-02  24.549  < 2e-16 ***
first_eq              2.283e-02  1.479e-03  15.441  < 2e-16 ***
lastt_eq              2.462e-02  1.616e-03  15.237  < 2e-16 ***
diff_len              1.758e-03  1.984e-04   8.862  < 2e-16 ***
mean_len             -8.069e-03  1.749e-04 -46.129  < 2e-16 ***
token_set_ratio       2.132e-04  9.899e-05   2.153  0.03130 *
token_sort_ratio      6.865e-03  1.112e-04  61.713  < 2e-16 ***
fuzz_ratio           -1.673e-03  1.279e-04 -13.084  < 2e-16 ***
fuzz_partial_ratio    3.166e-03  1.099e-04  28.817  < 2e-16 ***
longest_substr_ratio -3.827e-01  5.208e-03 -73.473  < 2e-16 ***
min_kcore             2.805e-03  1.170e-03   2.396  0.01656 *
max_kcore            -4.339e-05  2.792e-04  -0.155  0.87652
common_neighbor_count 4.268e-02  1.024e-03  41.675  < 2e-16 ***
common_neighbor_ratio 6.838e-01  4.889e-03 139.850  < 2e-16 ***
min_freq              2.424e-03  1.469e-04  16.493  < 2e-16 ***
max_freq             -3.789e-03  6.770e-05 -55.971  < 2e-16 ***
```

We can find that the feature of max_kcore has a large probability value, so we are supposed to remove this feature. And 0.00001 gain is achieved on F1-score by deleting it.

C. Discussion

a. Main Challenge

For the conventional SVM and Logistic Regression function, the running time is $O(n \times iteration\ number^2)$, in this case, n = 404290, which is pretty large, due to the time consuming factor, the efficiency of these 2 algorithms are pretty low, the maximum iteration cannot be set too high, the test result will not be accurate. As a result, the computed coefficient cannot predict the duplicate question pairs.

In addition, for a large number of features, there will be some overfitting issues. Deleting the useless features are quite necessary to make the training process better.

b. Challenge Solution

i. Use a higher performance server or machine, apply more time to run the algorithm.
ii. Using the Python built-in functions, such as *scikit* or *xgboost*, which contain functions of logistic regression and SVM, to fasten the running time.
iii. Applying regression to the feature data in R, finding the p-value for each feature, to find out the optimized features.

V. Conclusion

This project shows a comprehensive scheme to solve duplicate classification on Quora data set. Different types of features are extracted and tested respectively. The overall features along with the *xgboost* model achieved the best prediction results with log-loss-score=0.17035 on the test set and F1-score=0.9144 on cross-validation. The result is moderate compared to the highest score on Kaggle, which has a log-loss of 0.13. However, compared to the results introduced in related work, our model has made great improvement by adding more efficient features based on NLP.

In terms of the possible extensions to this project, several coefficients can be fine-tuned in the feature extraction models, such the smoothing coefficient in TF-IDF and the upper limit in the structure-based features. It’s interesting to find structure-based feature has the most effective predictability and further work on analyzing the question graph might be a good trial. Word embedding can be further analyzed along with deep learning tools. LSTM and CNN methods can be implemented to improve the efficiency of the problem. Also, for the overfitting problem, we can conduct more analysis on R, to find the best features that can help to predict the duplicate question pairs.

VI. Teamwork

Kai Ye: The literature review on related topics; the implementation and research of different feature extraction methodologies.

Zitao Yang: Implement the classification and result evaluation part.

- Write the SVM and Logistic Regression algorithms.
- Use the *scikit* and *xgboost* toolbox to help the classification process.
- Compute the logistic loss and F1 value of the result from different methods.
- Evaluate the model credibility.
- Discover more advanced algorithms.

References

[1] V. Gupta, G. S. Lehal and others, "A survey of text mining techniques and applications," *Journal of emerging technologies in web intelligence,* vol. 1, no. 1, pp. 60-76, 2009.

[2] J. Pennington, R. Socher and C. Manning, "Glove: Global vectors for word representation," *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP),* pp. 1532-1543, 2014.

[3] K. Muthmann, W. M. Barczynski, F. Brauer and A. Loser, "Near-duplicate detection for web-forums," *Proceedings of the 2009 International Database Engineering & Applications Symposium,* pp. 142-151, 2009.

[4] Y. Wu, Q. Zhang and X. Huang, "Efficient near-duplicate detection for q&a forum," *Proceedings of 5th International Joint Conference on Natural Language Processing,* pp. 1001-1009, 2011.

[5] T. Addair, "Duplicate question pair detection with deep learning," *Stanf. Univ. J,* 2017.

[6] C. Saedi, J. Rodrigues, J. Silva, V. Maraev and others, "Learning profiles in duplicate question detection," *2017 IEEE International Conference on Information Reuse and Integration (IRI),* pp. 544-550, 2017.

[7] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation,* vol. 28, no. 1, pp. 11-21, 1972.

[8] J. Gonzalez, P. Rodrigues and A. Cohen, "Fuzzywuzzy: Fuzzy string matching in python," 2017.

[9] A. Hagberg, D. Schult, P. Swart, D. Conway, L. Seguin-Charbonneau, C. Ellison, B. Edwards and J. Torrents, "Networkx. High productivity software for complex networks," https://networkx.lanl.gov/wiki, 2013.