DUPLICATE QUESTION PAIRS IDENTIFICATION

YUAN Yan Zhe
20728555
yyuanar@connect.ust.hk

YU Jia Hui
20718562
jyubn@connect.ust.hk

WEN Ze
20711552
zwenaf@connect.ust.hk

ABSTRACT

In the domain of Natural Language Processing (NLP), text similarity is a hot spot. It is particularly important to measure the similarity between sentences or phrases in some NLP subareas such as dialog system and information retrieval. Quora Question Pairs is a Kaggle competition, which challenges participants to tackle the problem of identifying duplicate questions. This paper works on the sentence similarity problem and deals with the task of duplicate question pairs identification.

In all, this paper proposes three models to tackle the duplicate question pairs identification problem on the Quora Question Pairs dataset. Model 1 uses traditional machine learning classifiers like SGD Classifier, Random Forest Classifier, and XGBoost Classifier with statistical, NLP and vector features extracted from question text for classification. Model 2 uses a Siamese BiLSTM based neural network structure combined with NLP features for classification. Model 3 fine-tunes BERT pre-trained model and uses a simple BiLSTM based neural network structure after BERT to adapt features from BERT for classification. All models achive a relatively good results.

1 PROBLEM DESCRIPTION

1.1 PROBLEM INTRODUCTION

Quora is a platform to ask questions and empowers people to learn from each other and to better understand the world. Over 100 million people visit Quora every month, so it's no surprise that many people ask similar questions. Multiple questions with the same intent can cause seekers to spend more time finding the best answer to their question, and make writers feel they need to answer multiple versions of the same question.

In this competition, Kagglers are challenged to tackle this Natural Language Processing problem by applying advanced techniques to classify whether question pairs are duplicates or not. (Chen et al., 2018)

1.2 PROBLEM TARGET

The target of this problem is to predict the probability of the semantic similatity of two questions.

The percentage of similarity is denfined as ”is_duplicate” which ranges from 0 to 1. The evaluation criteria is the logarithmic loss of the prediction and the groud truth.

1.3 QUALITATIVE AND QUANTITATIVE ANALYSIS

1.3.1 QUALITATIVE ASPECT

Classification Task

Classification is the process of predicting the class of given data. Classes are sometimes called as targets/ labels or categories. Classification predictive modeling is the task of approximating a mapping function (f) from input variables (X) to discrete output variables (y).

Classification task belongs to the category of supervised learning where the targets are also provided with the input data. There are many applications in classification in domains such as in credit approval, medical diagnosis, target marketing and etc.

Text Similarity Task

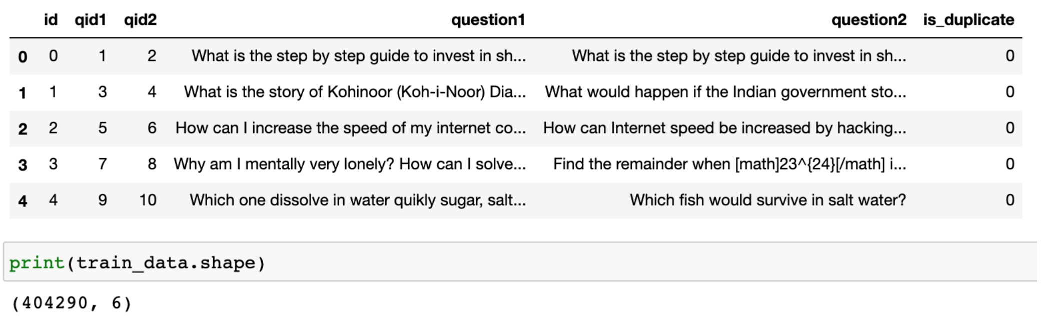
Text Similarity is one of the essential techniques of Natural Language Processing (NLP) which is being used to find the closeness between two chunks of text by it's meaning.

Two texts are classified to be similar if they infer the same meaning and have similar words or surface closeness. Semantic similarity measures the degree of semantic equivalence between two linguistic items, concepts, sentences, or documents.

1.3.2 Quantitative Aspect

About the Dataset

The training set consists of 400,000 question pairs.



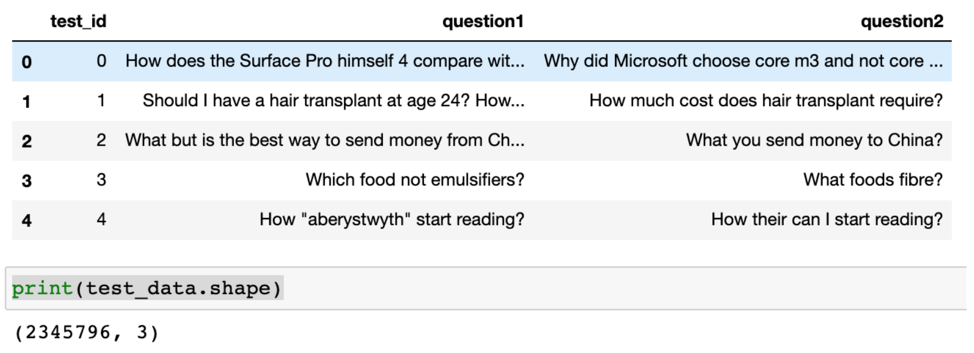
	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed 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

```
print(train_data.shape)
```

```
(404290, 6)
```

Figure 1.

The testing set consists of 2,345,796 question pairs.



	test_id	question1	question2
0	0	How does the Surface Pro himself 4 compare wit...	Why did Microsoft choose core m3 and not core ...
1	1	Should I have a hair transplant at age 24? How...	How much cost does hair transplant require?
2	2	What but is the best way to send money from Ch...	What you send money to China?
3	3	Which food not emulsifiers?	What foods fibre?
4	4	How "aberystwyth" start reading?	How their can I start reading?

```
print(test_data.shape)
```

```
(2345796, 3)
```

Figure 2.

Data fields:

- **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.

1.4 The Methodology

There are two directions to solve the problem of text similarity:

1.4.1 Method 1: Traditional Machine Learning-based

Based on traditional machine learning method, features are first extracted and then put into traditional machine learning models. In this problem, features in question pairs can be extracted by feature engineering, and then some classifier based on traditional machine learning can be trained, and finally the classification task will be realized.

1.4.2 Method 2: Deep Learning-based

The essence of deep learning method is to get the higher-dimensional representation of the data through deep learning models, and the whole process of feature extraction and classification is performed end-to-end.

1.4.3 Three Models the Paper Proposed

- **Model 1:** Based on the traditional machine learning methodology, Model 1 extracts multiple features from the text of quesiton pairs, then feeds features into traditional machine learning classifiers such as SGD Classifier, Random Forest Classifier and XGBoost Classifier.
- **Model 2:** Based on the deep learning methodology, Model 2 makes some modifications and optimizations on neural network structure, additional features and hyperparametes based no a Siamese LSTM with pretrained GloVe baseline model. As a result, Model 2 achieves a better performance than the baseline.
- **Model 3:** Based on the deep learning methodology, a fine-tuned the BERT pretrained model on qqp task followed by a Siamese BiLSTM based neural network structure for classification is implemented as the third model.

2 Models

2.1 Model 1

Based on the methodology of traditional machine learning introduced previously, this paper implements the first model (Model 1). The process of traditional machine learning model on classification tasks can be divided into two parts: input the training data or features of data into classifiers, and then train the classifiers to make decisions as accurate as possible. It is obvious that the more comprehensive the features is input into classifiers, the better result the classifiers will achieve theoretically. Therefore, feature extraction (or feature engineering) based on raw data is very important.

In Model 1, 41 kinds of statistical and NLP features combined with 192 dimensional features embedded from a pre-trained word-to-vector model are input (233 dimensions) into 3 classical machine learning classifiers for training. Fine tuning on hyperparameters is also done for each classifiers. Finally the 3 trained classifiers constitue Model 1.

In the following part, model 1 will be introduced in detail from four parts: Feature Engineering, Classifier 1, Classifier 2, and Classifier 3.

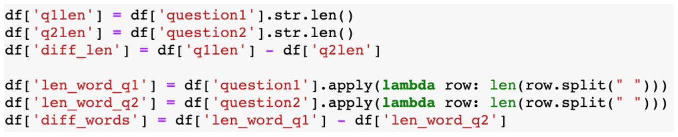
2.1.1 Feature Engineering

Feature engineering is to extract useful information from the data and process the data with professional skills, so that machine learning algorithm can behave better in datasets. In order to improve the classification effects, further feature extruction from the sentence text is needed to better represent the raw data input into classifiers. (Arora et al., 2016)

After preprocessing sentences with technologies like python NLTK package to remove stopwpords and python re package to replace common abbreviations, two kinds of features are extracted by Model 1. (web, d)

Statistical & NLP Features:
Since the data is text, this paper extracts NLP features to represent sentence. Through literature review, network searching and experience summerization, a total of 41 features are extracted. All features are shown in the Table 1.

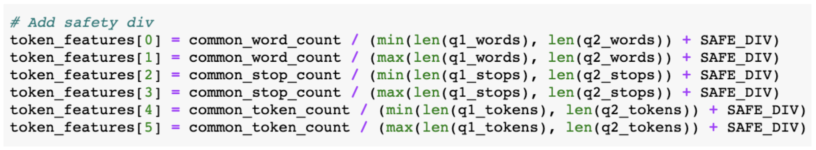
Among them, there are some statistical characteristics of sentences, such as the length of sentence 1 and sentence 2, the difference between their length, the common words between sentence 1 and sentence 2 and so on. Some codes in this part are shown in Figure 3.



```
df['q1len'] = df['question1'].str.len()
df['q2len'] = df['question2'].str.len()
df['diff_len'] = df['q1len'] - df['q2len']

df['len_word_q1'] = df['question1'].apply(lambda row: len(row.split(" ")))
df['len_word_q2'] = df['question2'].apply(lambda row: len(row.split(" ")))
df['diff_words'] = df['len_word_q1'] - df['len_word_q2']
```

Figure 3.

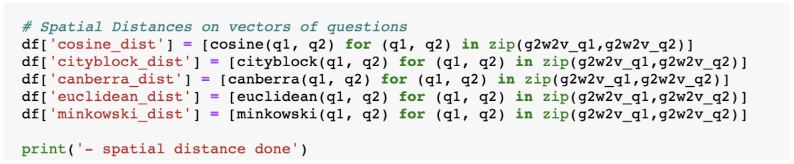
There are mathematical features based on statistical features. For example, cwc_min means the number of shared words in sentence 1 and sentence2 divided by the minimum number of words in sentences 1 and 2 (plus a safe_div to avoid 0 in the denominator). Some codes in this part are shown in Figure 4.



```
# Add safety div
token_features[0] = common_word_count / (min(len(q1_words), len(q2_words)) + SAFE_DIV)
token_features[1] = common_word_count / (max(len(q1_words), len(q2_words)) + SAFE_DIV)
token_features[2] = common_stop_count / (min(len(q1_stops), len(q2_stops)) + SAFE_DIV)
token_features[3] = common_stop_count / (max(len(q1_stops), len(q2_stops)) + SAFE_DIV)
token_features[4] = common_token_count / (min(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)
token_features[5] = common_token_count / (max(len(q1_tokens), len(q2_tokens)) + SAFE_DIV)
```

Figure 4.

There are various distances measurements between sentence vectors, such as Cosine distance, Canberra distance, etc. realized through functions in Scipy.Spatial.Distance. The sentence vectors are obtained by word-to-vector embedding using the glove2word2vec function in Gensim package and averaging each word vector in the sentence.Some codes in this part are shown in Figure 5. (Kusner et al., 2015) (web, e)



```
# Spatial Distances on vectors of questions
df['cosine_dist'] = [cosine(q1, q2) for (q1, q2) in zip(g2w2v_q1,g2w2v_q2)]
df['cityblock_dist'] = [cityblock(q1, q2) for (q1, q2) in zip(g2w2v_q1,g2w2v_q2)]
df['canberra_dist'] = [canberra(q1, q2) for (q1, q2) in zip(g2w2v_q1,g2w2v_q2)]
df['euclidean_dist'] = [euclidean(q1, q2) for (q1, q2) in zip(g2w2v_q1,g2w2v_q2)]
df['minkowski_dist'] = [minkowski(q1, q2) for (q1, q2) in zip(g2w2v_q1,g2w2v_q2)]

print('- spatial distance done')
```

Figure 5.

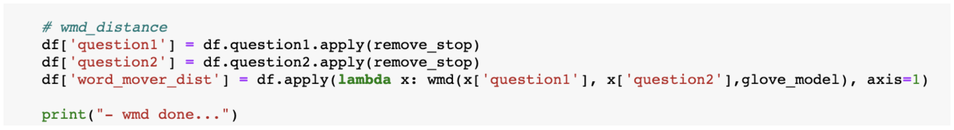
In addition, there are some other NLP features obtained from papers or network resources. For example, word moving distance (WMD) is used. WMD is a combination of semantic distance of words in two documents. Euclidean distance is calculated from the word vectors of any two words in two question sentences, and then weighted sum is used as sentence distance. (Kusner et al., 2015) What's more, Fuzzywuzzy feature is also used. (web, f) It is a simple and easy-to-use fuzzy string matching toolkit which uses Levenshtein Distance to calculate word distances. In Model 1, features like fuzzy_ratio (simple match), token_sort_Ratio (order ignored matching) and other characteristics are used. In all, these magic features have been proved to be good features in semantic similarity measurement tasks. Some codes in this part are shown in Figure 6 and Figure 7:

Statistics and NLP features mine text features and express sentences at a higher level. However, it is not enough for this task since they lack semantic features. Therefore, vector features representing question sentences semantically is needed so as to construct the final question pair feature matrix.

Vector Features:
The idea of extracting vector features mainly comes from the deep learning methodology of seman-

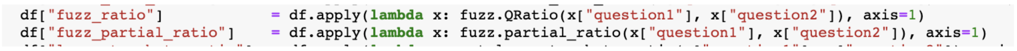
Statistical & NLP Features	Description
q1len	Length of q1
q2len	Length of q2
diff_len	len(q1)-len(q2)
q1_n_words	Number of words in q1
q2_n_words	Number of words in q2
diff_n_words	The difference
caps_count_q1	Number of capital words of q1
caps_count_q2	Number of capital words of q2
diff_caps	The difference
len_char_q1	Number of characters of q1
len_char_q2	Number of characters of q2
diff_len_char	The difference
avg_word_len1	len(char)/len(word) of q1
avg_word_len2	len(char)/len(word) of q2
diff_avg_word	The difference
word_Common	Number of common unique words in q1 and q2
word_Total	Total num of words in Question 1 + Total num of words in q2
word_share	(word_common)/(word_Total)
2gram_share_	word share on 2 gram
exactly_same	exactly the same
last_word_eq	Check if Last word of both questions is equal or not last_word_eq = int(q1_tokens[-1] == q2_tokens[-1])
first_word_eq	Check if First word of both questions is equal or not first_word_eq = int(q1_tokens[0] == q2_tokens[0])
abs_len_diff	Abs. length difference abs_len_diff = abs(len(q1_tokens) - len(q2_tokens)
mean_len	Average Token Length of both Questions mean_len = (len(q1_tokens) + len(q2_tokens))/2
cwc_min	Ratio of common_word_count to min lenghth of word count of Q1 and Q2 cwc_min = common_word_count / (min(len(q1_words), len(q2_words))
cwc_max	Ratio of common_word_count to max lenghth of word count of Q1 and Q2 cwc_max = common_word_count / (max(len(q1_words),len(q2_words))
csc_min	Ratio of common_stop_count to min lenghth of stop count of Q1 and Q2 csc_min = common_stop_count / (min(len(q1_stops), len(q2_stops))
csc_max	Ratio of common_stop_count to max lenghth of stop count of Q1 and Q2 csc_max = common_stop_count / (max(len(q1_stops), len(q2_stops))
ctc_min	Ratio of common_token_count to min lenghth of token count of Q1 and Q2 ctc_min = common_token_count / (min(len(q1_tokens), len(q2_tokens))
ctc_max	Ratio of common_token_count to max lenghth of token count of Q1 and Q2 ctc_max = common_token_count / (max(len(q1_tokens), len(q2_tokens))
wmd_dist	Thesis reference:`http://proceedings.mlr.press/v37/kusnerb15.pdf`
cosine_dist	Cosine distance here is the cosine distance between two glove based vectors.
cityblock_dist	just follow the official defination
canberra_dist	just follow the official defination
euclidean_dist	just follow the official defination
minkowski_dist	just follow the official defination
fuzz_ratio	`https://github.com/seatgeek/fuzzywuzzy#usage`
fuzz_partial_ratio	`https://github.com/seatgeek/fuzzywuzzy#usage`
token_sort_ratio	`https://github.com/seatgeek/fuzzywuzzy#usage`
token_set_ratio	`https://github.com/seatgeek/fuzzywuzzy#usage`
longest_substr_ratio	longest_substr_ratio=len(longest common substring) / (min(len(q1_tokens), len(q2_tokens))

Table 1: Statistical & NLP features.



```
# wmd_distance
df['question1'] = df.question1.apply(remove_stop)
df['question2'] = df.question2.apply(remove_stop)
df['word_mover_dist'] = df.apply(lambda x: wmd(x['question1'], x['question2'],glove_model), axis=1)

print("- wmd done...")
```

Figure 6.



```
df["fuzz_ratio"]            = df.apply(lambda x: fuzz.QRatio(x["question1"], x["question2"]), axis=1)
df["fuzz_partial_ratio"]    = df.apply(lambda x: fuzz.partial_ratio(x["question1"], x["question2"]), axis=1)
```

Figure 7.

tic similarity. The essence of deep learning method is to use deep neural network model to learn the semantic representations of sentences. This paper holds that if the n-dimensional vector representation of a sentence can be obtained, then the flattened n-dimensional vector can also be used as n features of the sentence.

There are two ways to learn the representations of sentence vectors: One is to design the deep neural network structure, the other is to use pre-trained models. In order to simplify the model, a pre-trained model is used in Model 1 for vector embedding.

The pre-trained model is a model that has been trained in advance on datasets. It is an application of transfer learning. The pre-training process usually consists of two steps: The first step is to obtain a task independent pre-trained model from large-scale data through self supervised learning. The model learns semantic representations of a word in a specific context, which is independent of specific tasks. The second step is to fine tune the network for downstream tasks. The advantage of the pre-trained model is that the training cost is low, the convergence speed can be faster with downstream tasks, and the performance of the model can be effectively improved. In other words, the pre-trained method can be considered to learn based from a better initial status, so as to achieve better performance.

There are two strategies in the usage of pre-trained models. The first is to directly embed words to vector representations, and the second is to get the word-to-vector embedding after fine-tuning the pre-trained model for the training set of the downstream task. In general, the latter often performs better in the testing set because it focuses on features of the training set in the downstream task. In Model 1, this paper uses the first strategy directly. On the other hand, this paper uses the latter method in both Model 2 and Model 3. (Sun et al., 2019)

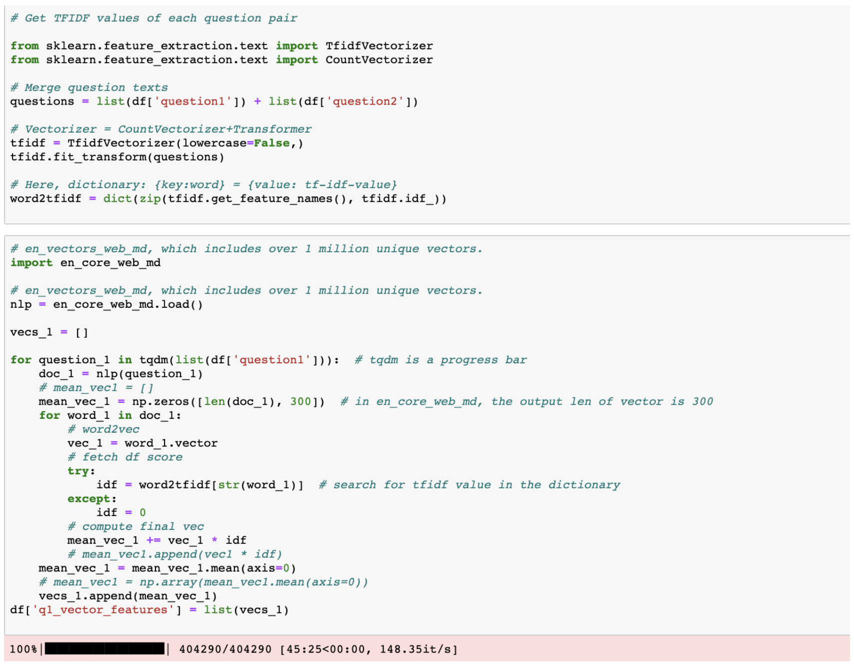
There are many kinds of pre-trained models for NLP tasks. In Model 1, widely-used word-to-vector models are considered for embedding. However, the data are sentence pairs, so how to express sentences through word-to-vector embedding becomes a difficult problem. Model 1 uses the idea of average by averaging each word vector in a sentence as the representation of the sentence vector. What's more, in order to distinguish every word in the sentence, Model 1 also uses the idea of smooth inverse frequency, and calculates the TF-IDF weighted average of word vectors in a sentence. Finally Model 1 flattens the sentence vector to n features as the vector feature for sentences.

There are many kinds of word-to-vector pre-trained models, such as GloVe, Word2Vec, etc. Model 1 uses en_core_web_sm in Spacy package. Spacy is an industrial natural language processing tool, which supports a variety of basic functions of natural language processing. (web, c) For English word embedding, Spacy provides a variety of English pre-trained models. From the consideration of simplicity, Model 1 chooses en_core_web_sm, an English multi task CNN trained on OntoNotes as the pre-trained model. (Srinivasa-Desikan, 2018)

To sum up, Model 1 adopts en_core_web_sm pre-trained model for word embedding and uses TFIDF weighted average to represent sentence as the vector features. Some codes in this part are shown in Figure 8.

Of course, there are other embedding pre-trained models like Sentence2Vec and Doc2Vec. They are considered as future work in this project.

A screenshot of the final features of training data after feature engineering is shown in Figure 9: After extracting features from the training and testing data, training data with features are divided

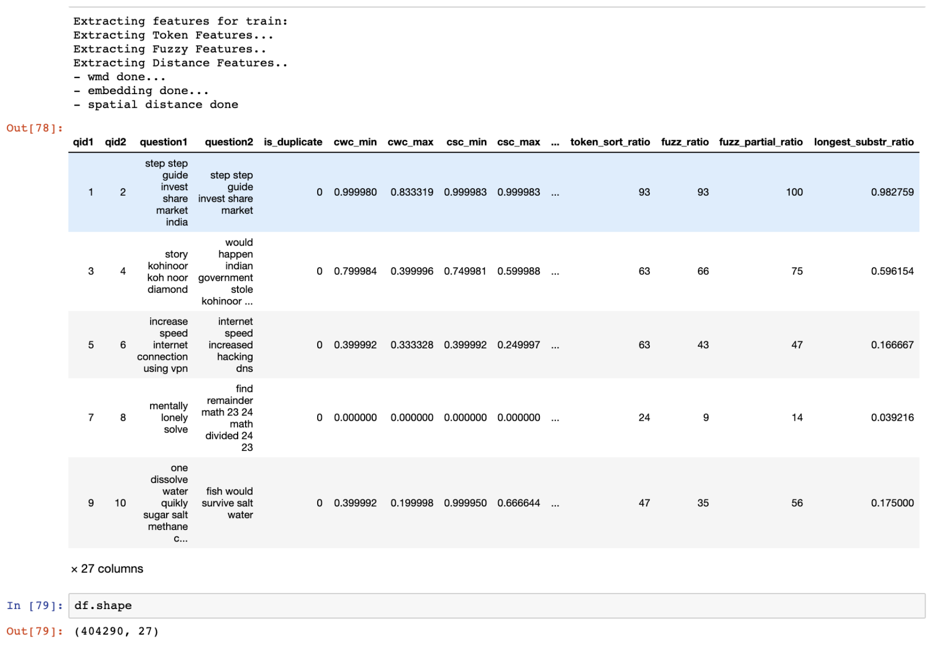


```
# Get TFIDF values of each question pair

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

# Merge question texts
questions = list(df['question1']) + list(df['question2'])

# Vectorizer = CountVectorizer+Transformer
tfidf = TfidfVectorizer(lowercase=False,)
tfidf.fit_transform(questions)

# Here, dictionary: {key:word} = {value: tf-idf-value}
word2tfidf = dict(zip(tfidf.get_feature_names(), tfidf.idf_))
```

```
# en_vectors_web_md, which includes over 1 million unique vectors.
import en_core_web_md

# en_vectors_web_md, which includes over 1 million unique vectors.
nlp = en_core_web_md.load()

vecs_1 = []

for question_1 in tqdm(list(df['question1'])):  # tqdm is a progress bar
    doc_1 = nlp(question_1)
    # mean_vec1 = []
    mean_vec_1 = np.zeros([len(doc_1), 300])  # in en_core_web_md, the output len of vector is 300
    for word_1 in doc_1:
        # word2vec
        vec_1 = word_1.vector
        # fetch df score
        try:
            idf = word2tfidf[str(word_1)]  # search for tfidf value in the dictionary
        except:
            idf = 0
        # compute final vec
        mean_vec_1 += vec_1 * idf
        # mean_vec1.append(vec1 * idf)
    mean_vec_1 = mean_vec_1.mean(axis=0)
    # mean_vec1 = np.array(mean_vec1.mean(axis=0))
    vecs_1.append(mean_vec_1)
df['q1_vector_features'] = list(vecs_1)
```

```
100%|██████████| 404290/404290 [45:25<00:00, 148.35it/s]
```

Figure 8.



```
Extracting features for train:
Extracting Token Features...
Extracting Fuzzy Features..
Extracting Distance Features..
- wmd done...
- embedding done...
- spatial distance done
```

Out[78]:

qid1	qid2	question1	question2	is_duplicate	cwc_min	cwc_max	csc_min	csc_max	...	token_sort_ratio	fuzz_ratio	fuzz_partial_ratio	longest_substr_ratio
1	2	step step guide invest share market india	step step guide invest share market	0	0.999980	0.833319	0.999983	0.999983	...	93	93	100	0.982759
3	4	story kohinoor koh noor diamond	would happen indian government stole kohinoor ...	0	0.799984	0.399996	0.749981	0.599988	...	63	66	75	0.596154
5	6	increase speed internet connection using vpn	internet speed increased hacking dns	0	0.399992	0.333328	0.399992	0.249997	...	63	43	47	0.166667
7	8	mentally lonely solve	find remainder math 23 24 math divided 24 23	0	0.000000	0.000000	0.000000	0.000000	...	24	9	14	0.039216
9	10	one dissolve water quikly sugar salt methane c...	fish would survive salt water	0	0.399992	0.199998	0.999950	0.666644	...	47	35	56	0.175000

× 27 columns

```
In [79]: df.shape

Out[79]: (404290, 27)
```

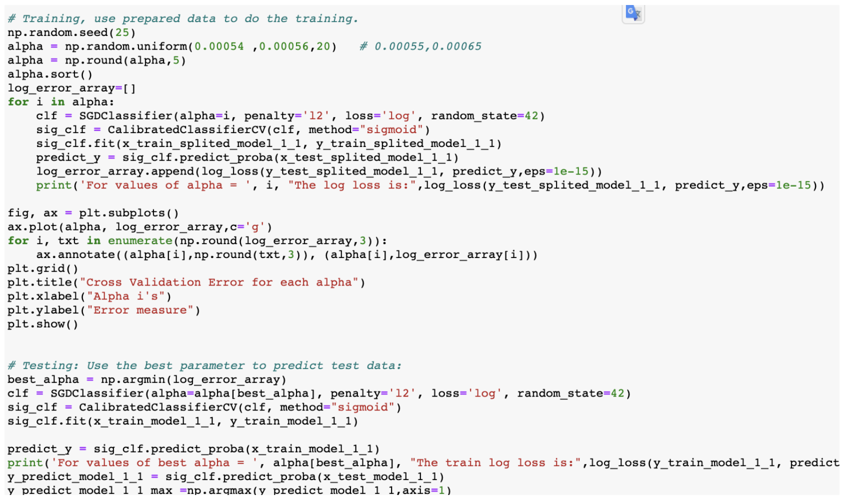
Figure 9.

into training and validating data by 7:3. Then data is preprocessed (mainly preprocessed by filling null values with 0) and then feed into the following three classifiers:

2.1.2 Classifier 1: SGD Classifier

Sklearn's SGD classifier trains a linear classifier (like logistic regression, SVM, etc.) by using the method of random gradient descent. In other words, one epoch has only one process of iteration and updation, so the training process converges faster when dealing with big data. However, the disadvantage is that a single sample may be noisy, which means not every iteration moves towards the optimal direction overally.

Model 1 uses SGD classifier as the first classifier. Since SGD is sensitive to feature scaling, there is a scaling preprocessing of the input data. What's more, learning rate, penalty type and loss function type are tuned through the whole training process. The score of the model with the best combination of parameters is 0.42400 on the test set. The core codes are in Figure 10:



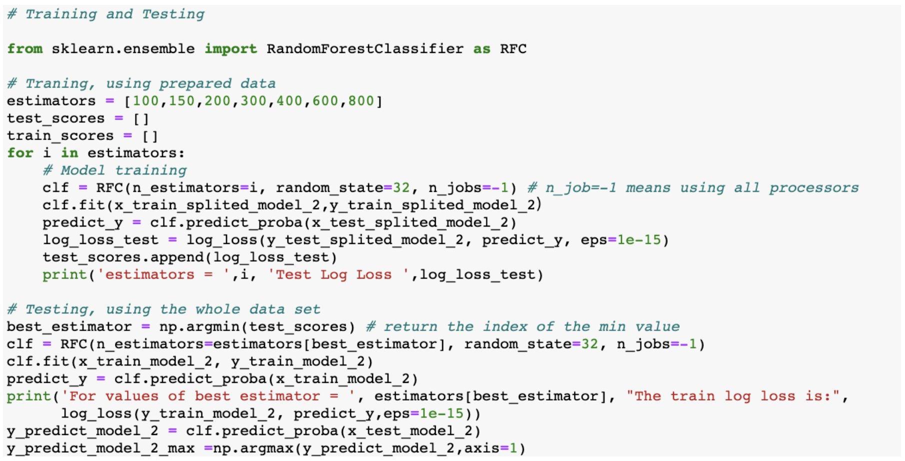
```
# Training, use prepared data to do the training.
np.random.seed(25)
alpha = np.random.uniform(0.00054 ,0.00056,20)   # 0.00055,0.00065
alpha = np.round(alpha,5)
alpha.sort()
log_error_array=[]
for i in alpha:
    clf = SGDClassifier(alpha=i, penalty='l2', loss='log', random_state=42)
    sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
    sig_clf.fit(x_train_splited_model_1_1, y_train_splited_model_1_1)
    predict_y = sig_clf.predict_proba(x_test_splited_model_1_1)
    log_error_array.append(log_loss(y_test_splited_model_1_1, predict_y,eps=1e-15))
    print('For values of alpha = ', i, "The log loss is:",log_loss(y_test_splited_model_1_1, predict_y,eps=1e-15))

fig, ax = plt.subplots()
ax.plot(alpha, log_error_array,c='g')
for i, txt in enumerate(np.round(log_error_array,3)):
    ax.annotate((alpha[i],np.round(txt,3)), (alpha[i],log_error_array[i]))
plt.grid()
plt.title("Cross Validation Error for each alpha")
plt.xlabel("Alpha i's")
plt.ylabel("Error measure")
plt.show()


# Testing: Use the best parameter to predict test data:
best_alpha = np.argmin(log_error_array)
clf = SGDClassifier(alpha=alpha[best_alpha], penalty='l2', loss='log', random_state=42)
sig_clf = CalibratedClassifierCV(clf, method="sigmoid")
sig_clf.fit(x_train_model_1_1, y_train_model_1_1)

predict_y = sig_clf.predict_proba(x_train_model_1_1)
print('For values of best alpha = ', alpha[best_alpha], "The train log loss is:",log_loss(y_train_model_1_1, predict
y_predict_model_1_1 = sig_clf.predict_proba(x_test_model_1_1)
y_predict_model_1_1_max =np.argmax(y_predict_model_1_1,axis=1)
```

Figure 10.

2.1.3 Classifier 2: Random Forest Classifier

Random forest classifier is a method of ensemble learning. It is a classifier with multiple decision trees. On the basis of bagging, it performs random attribute selection during training. And the output category is determined by the majority of output classes from all trees.

In Model 1, random forest is used as the second classifier, and the n_estimators and max_ depth are tuned. The score of the model with the best combination of parameters is 0.38000. The core codes are shown in Figure 11:



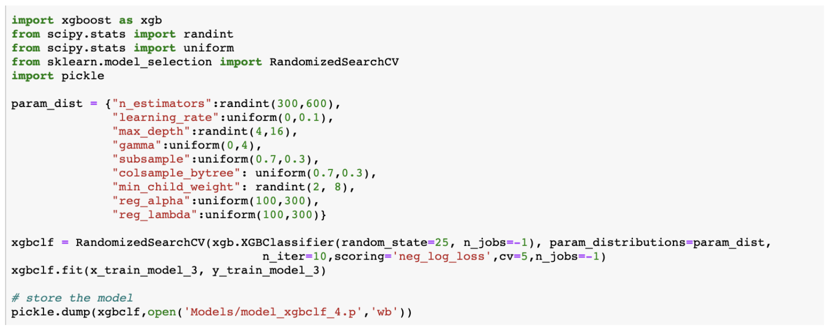
```
# Training and Testing

from sklearn.ensemble import RandomForestClassifier as RFC

# Traning, using prepared data
estimators = [100,150,200,300,400,600,800]
test_scores = []
train_scores = []
for i in estimators:
    # Model training
    clf = RFC(n_estimators=i, random_state=32, n_jobs=-1) # n_job=-1 means using all processors
    clf.fit(x_train_splited_model_2,y_train_splited_model_2)
    predict_y = clf.predict_proba(x_test_splited_model_2)
    log_loss_test = log_loss(y_test_splited_model_2, predict_y, eps=1e-15)
    test_scores.append(log_loss_test)
    print('estimators = ',i, 'Test Log Loss ',log_loss_test)

# Testing, using the whole data set
best_estimator = np.argmin(test_scores) # return the index of the min value
clf = RFC(n_estimators=estimators[best_estimator], random_state=32, n_jobs=-1)
clf.fit(x_train_model_2, y_train_model_2)
predict_y = clf.predict_proba(x_train_model_2)
print('For values of best estimator = ', estimators[best_estimator], "The train log loss is:",
      log_loss(y_train_model_2, predict_y,eps=1e-15))
y_predict_model_2 = clf.predict_proba(x_test_model_2)
y_predict_model_2_max =np.argmax(y_predict_model_2,axis=1)
```

Figure 11.

2.1.4 Classifier 3: XGBoost Classifier

XGBoost, namely extreme gradient boosting, is an optimization of GBDT (Gradient Boosting Decision Trees). GBDT trains a tree with training data and the corresponding true values (i.e. the standard answer) in the first iteration, and then uses this tree to predict the training set. From the deviation between the predicted values and the true values, the 'residuals' can be obtained by subtracting the two values. Next, the second tree is trained. At this time, the true values are no longer used, the residuals are used instead as the true values in the second round training. After two trees are trained, the residual of each training sample can be obtained again to the third iteration and so on so forth. XGBoost improves and optimizes the loss function, regularization term and column sampling of GBDT. It is famous for better learning effects and faster speed. (Chen & Guestrin, 2016)

Model 1 uses XGBoost as the third classifier. In order to find the optimal parameters, Sklearn's RandomizedSearchCV function is used to fine-tune the parameters like n_ estimators, learning_ rate, max_depth and so on. Parameters are adjusted randomly in a certain range, and the best set of parameters is selected as the final parameter set after many experiments. (web, a) As a result, the best result of the model on the test set is 0.34376. Some codes are in Figure 12:



```python
import xgboost as xgb
from scipy.stats import randint
from scipy.stats import uniform
from sklearn.model_selection import RandomizedSearchCV
import pickle

param_dist = {"n_estimators":randint(300,600),
              "learning_rate":uniform(0,0.1),
              "max_depth":randint(4,16),
              "gamma":uniform(0,4),
              "subsample":uniform(0.7,0.3),
              "colsample_bytree": uniform(0.7,0.3),
              "min_child_weight": randint(2, 8),
              "reg_alpha":uniform(100,300),
              "reg_lambda":uniform(100,300)}

xgbclf = RandomizedSearchCV(xgb.XGBClassifier(random_state=25, n_jobs=-1), param_distributions=param_dist,
                                   n_iter=10,scoring='neg_log_loss',cv=5,n_jobs=-1)
xgbclf.fit(x_train_model_3, y_train_model_3)

# store the model
pickle.dump(xgbclf,open('Models/model_xgbclf_4.p','wb'))
```

Figure 12.

2.2 Model 2 Siamese BiLSTM with additional features

A Siamese neural network, which is also called twin neural network, is an artificial neural network that uses the same weights to deal with two different input vectors while computing comparable output vectors. Often one of the output vectors is computed first, then forming a baseline against which the other output vector is compared. (Mueller & Thyagarajan, 2016)In the paper of Siamese Recurrent Architectures for Learning Sentence Similarity, a siamese Long Short-Term Memory(LSTM) model for Quora question pairs problem is defined with the name of Manhattan LSTM Model, which is shown in Figure 13. (Zhang et al., 2020)

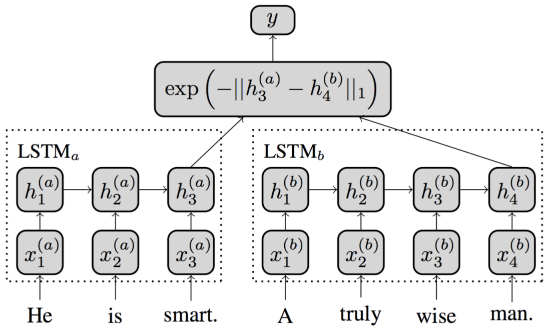
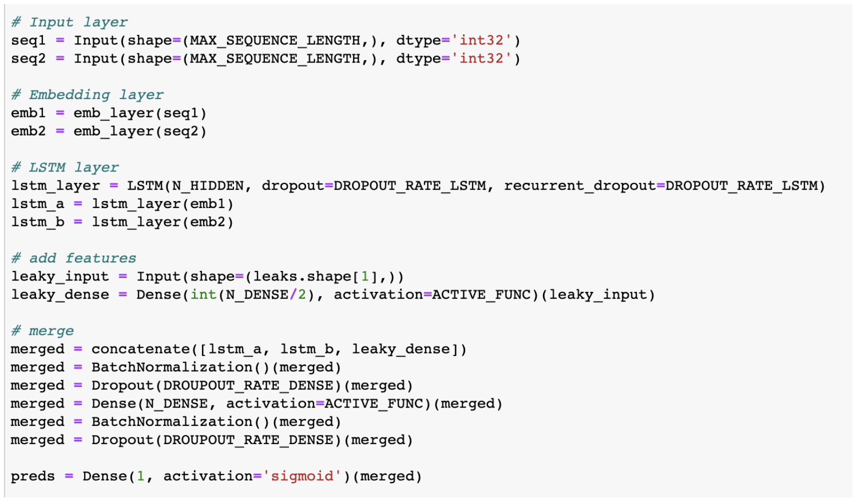

y
$\exp\left(-\|h_3^{(a)} - h_4^{(b)}\|_1\right)$
LSTM$_a$
LSTM$_b$
$h_1^{(a)}$ $h_2^{(a)}$ $h_3^{(a)}$
$h_1^{(b)}$ $h_2^{(b)}$ $h_3^{(b)}$ $h_4^{(b)}$
$x_1^{(a)}$ $x_2^{(a)}$ $x_3^{(a)}$
$x_1^{(b)}$ $x_2^{(b)}$ $x_3^{(b)}$ $x_4^{(b)}$
He is smart.
A truly wise man.

Figure 13.

Elior Cohen has done some experiments on MaLSTM based on the same dataset and his result is around 80% for validation accuracy, which is not bad. The motivation for trying LSTM network is from his experiments. So his model is regarded as a baseline for neural network solution in this project. Some new analysis is done on the baseline model. Based on these experiments, a new model Model 2 is implemented, which is called 'Siamese BiLSTM with additional features'. (Imtiaz et al., 2020)

2.2.1 Baseline Process

A notebook from Kaggle is used as reference to baseline model. Main process is introduced as below, the main codes are shown in Figure 14. (web, g):



```
# Input layer
seq1 = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')
seq2 = Input(shape=(MAX_SEQUENCE_LENGTH,), dtype='int32')

# Embedding layer
emb1 = emb_layer(seq1)
emb2 = emb_layer(seq2)

# LSTM layer
lstm_layer = LSTM(N_HIDDEN, dropout=DROPOUT_RATE_LSTM, recurrent_dropout=DROPOUT_RATE_LSTM)
lstm_a = lstm_layer(emb1)
lstm_b = lstm_layer(emb2)

# add features
leaky_input = Input(shape=(leaks.shape[1],))
leaky_dense = Dense(int(N_DENSE/2), activation=ACTIVE_FUNC)(leaky_input)

# merge
merged = concatenate([lstm_a, lstm_b, leaky_dense])
merged = BatchNormalization()(merged)
merged = Dropout(DROUPOUT_RATE_DENSE)(merged)
merged = Dense(N_DENSE, activation=ACTIVE_FUNC)(merged)
merged = BatchNormalization()(merged)
merged = Dropout(DROUPOUT_RATE_DENSE)(merged)

preds = Dense(1, activation='sigmoid')(merged)
```

Figure 14.

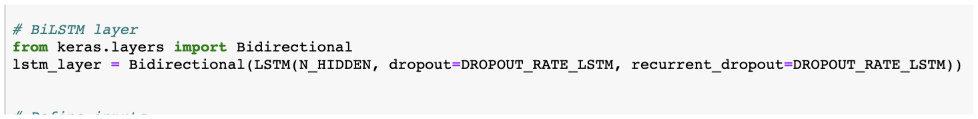
- Definate hyperparameters. Many hyperparameters need to be set before all experiments, such as the number of output neuron in LSTM layers and dense layers and dropout rates for two kinds of neural network layers. Random numbers in certain intervals are used in original notebook. However, in order to reproduce the result of model and make comparison with other improvement, fixed hyperparameters should be used in this project. (250, 120, 0.33, 0.33) is chosen as the fixed hyperparameter set for best performance in all experiment results, each standing for output neuron number in LSTM layers and dense layers and dropout rates for LSTM layers and dense layers.
- Preprocess data. Convert all words to lower case and remove unwanted characters. Fill null data with "empty".
- Convert word to number according to word index. This process is mainly done by Tokenizer. After tokenizing words in all sentences, pad all data with max sequence length.
- Get leaky features. In the original notebook, single question frequency and question pair frequency are used as leaky features. It is intuitively reasonable, so this paper does the same thing.
- Initialize word vectors for embedding layer. The baseline model uses results from pre-trained GloVe model as the inital word-embedding matrix. GloVe is a well-known model for obtaining vector representations for words. It uses an unsupervised learning algorithm to train word-word co-occurrence statistics from a large corpus and is wildly used in NLP problems. (Pennington et al., 2014)
- Design neural network model structure. This is the most important step in whole process. The skeleton for this model is constructed by two LSTM layers (each processing one question in a given question pair), one dense layer for leaky features plus two dense layers for combined features. Sigmoid function is used as the activation function. Those two LSTM layers in the Figure 14 are exactly where "Siamese" lies. Batch normalization is done to avoid gradient vanishing and exploding gradient problems.
- Add class weights. Re-weight classes in order to fit the share in test set. This can help with the improvement of final accuracy. Later, a experiment without re-weighting classes will be presented to show the improvement.

- Train the model. Binary crossentropy is used as loss and nAdam is used as optimizer.

2.2.2 Improvement from Baseline

Improvement 1: Siamese Bidirectional LSTM Structure

Undirectional LSTM can only receive input message from the past while bidirectional LSTM can get information from past and future simultaneously. This implies later can understand semantic relationship better because of the increasing of the amount of input information available to the network. So bidirectional structure is tried as Figure 15 shows. (web, i)



```
# BiLSTM layer
from keras.layers import Bidirectional
lstm_layer = Bidirectional(LSTM(N_HIDDEN, dropout=DROPOUT_RATE_LSTM, recurrent_dropout=DROPOUT_RATE_LSTM))
```

Figure 15.

After training for 4 epochs, siamese bidirectional LSTM structure scores 0.18550, which is a big improvement from baseline score 0.19300.

Improvement 2: Siamese-LSTM with More Features

Now that frequency can be used as leaky features, how about features obtained in model 1? Then we try to add more features to the original model and double the number of neurons in dense layer which is constructed for leaky features. After training for 4 iterations, score 0.16515 is obtained. This is the best score in the whole project.

Improvement 3: Fine-tuning the Network Parameters and Structure

(1), (2), (3) change model hyperparameters and are based on siamese LSTM model (original model), while (4), (5), (6) change the neural network structure and are based on siamese LSTM model with 40 more features. Former baseline scores 0.19300 and latter baseline scores 0.16515.

(1) Change model hyperparameters

First, preprocessing raw data by removing stopwords and doing stemming is tired. After this step, it is easier for model to process input sentences. However, final score increases a lot, which means it is a bad way to preprocess data. Maybe the reason behind it is sentences without stopwords and being stemming are not ”real sentences” any more, making it more complicated for model to detect their semantic relationships.

(2) Do not use leaky features

Try how model performs without any leaky features. Again, result is worse. A conclusion can be made here, that is leaky features really help a lot.

(3) Do not re-weight classes

Re-weight classes in training dataset to fit class share in testing dataset. Because data distribution is the same in different datasets, so final result may be greater. This point is verified by the score of the model trained with dataset not being re-weighted.

(4) Add substract and multiply on LSTM features

Do some experiments on feature varieties such as adding substract and multiply results of LSTM featues. All of them score a little higher, which means adding this feature may help gain test scores.

(5) Change the number of output neurons in a dense layer

Since lots of features are added to the model, the model may tend to pay more attention to leaky features than LSTM features. Maybe changing the number of output neurons in the dense layer designed for leaky features may deal with this problem. However, different conclusion is got from the result. The model with less number of output neurons scores 0.16681, a little bit higher than the baseline 0.16515.

(6) Add a dense layer after merging all features

Just have a try. However, the result is not satisfying, scoring 0.18400, which means just one Dense layer is enough for this task.

2.2.3 Conclusions on Accuracy

Models	Training Set Accuracy
Siamese LSTM Baseline	0.8559
Siamese Bidirectional LSTM	0.8555
Siamese BiLSTM with NLP features	0.8723

Table 2: Training Set Accuracy.

Figures in the table above implies Siamese BiLSTM model with NLP features fits the training set best. Performances of Siamese LSTM model and Siamese Bidirectional LSTM model are almost the same but the former is a bit better. So we collect all our experient results, and implement Model 2, the network structure diagram is shown in Figure 16.

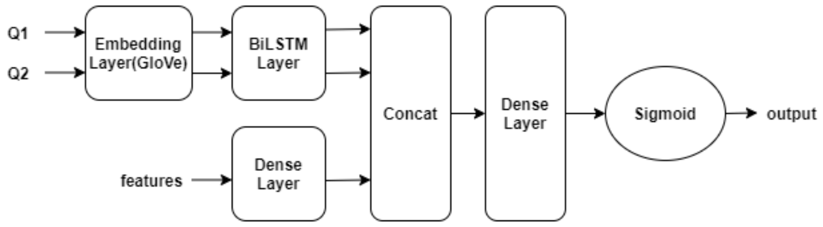


Figure 16.

2.3 Model 3

2.3.1 BERT based methodology

BERT (Bidirectional Encoder Representation from Transformers) is an state-of-art language representation model that maps a token sequence into a sequence of vectors. BERT is trained using two unsupervised tasks: Masked Language Model (MLM) and Next Sentence Prediction (NSP). BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. The pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications. (Devlin et al., 2018) (Vaswani et al., 2017). The BERT sketch map is shown in Figure 17.

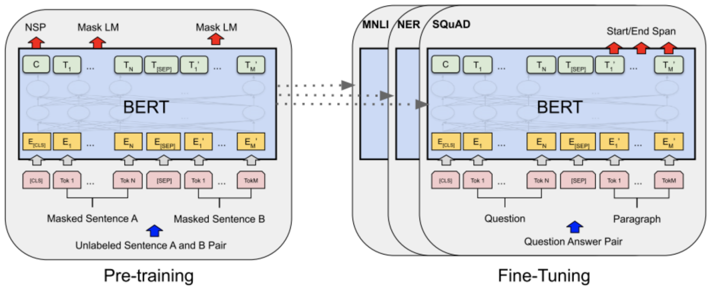


Figure 17.

There are two ways to use BERT in qqp dataset. One way is to use the pre-trained BERT to embed sentences without fine-tuning, then just feed the embedded features into neural network sturctures and output the classification on whether the question pair is duplicate or not. Actually there are some implementations on that such as ’SBERT’ (Sentence BERT, or sentence-transformations package), there are also packages like ’bert as a service’ for users to apply the pre-trained model easily. Another way is to do fine-tuning with BERT model on downstream tasks (qqp dataset), which may achieve a better performance since the pre-trained model will be trained on the downstream training dataset. This paper implemented the second idea. (Reimers & Gurevych, 2019) (web, l) (web, b) (web, h) (web, j) (web, k)

2.3.2 MODEL DESCRIPTIONS

In model 3, both fine-tuning on BERT and a Siamese BiLSTM based neural network structure are implemented, so it can be devided into two parts. The main diagram of Model 3 is shown in Figure 18.

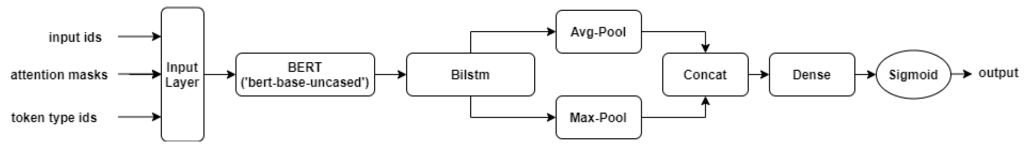


Figure 18

- The first part is BERT pre-trained model (using ’bert-base-uncased’ version) . In this part, if trainable is set to Flase, BERT will not be fine tuned, otherwise BERT is fine-tuned to embed question pairs into vectors.
- The second part is a Siamese BiLSTM based neural network sturcture consists of trainable layers. It is designed to adapt the pre-trained features from BERT, and perform classification with embedded vectors from fined tuned BERT.

We also use the following strategies in model training and testing. In all, these things are done step by step:

- Load and preprocess the raw data.
- create a data generator and generate training and testing data for the model
- set bert_model.trainable=False, train the model (train the layers in the neural network structure)
- set bert_model.trainable=Ture, train the model again (fine-tuning BERT)
- make predictions on the testing set
- get submissions on qqp problem

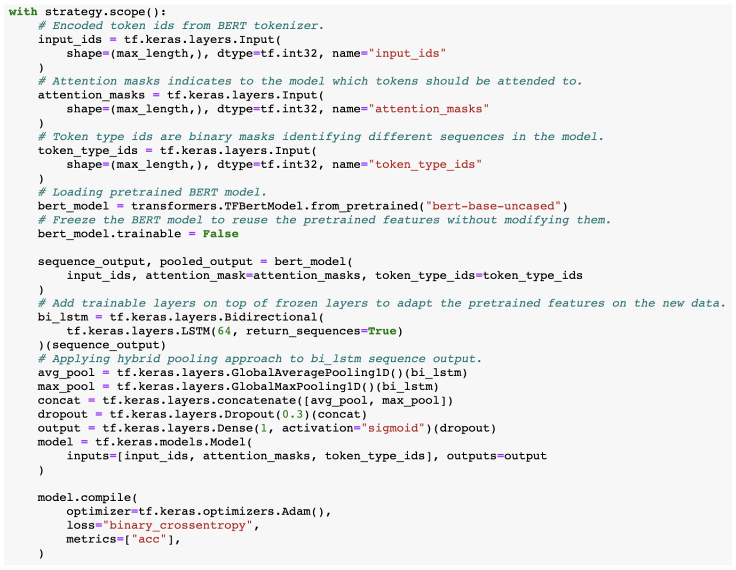
As a result, the model scores 0.27835 on test dataset. Core codes are shown as Figure 19:

3 RESULTS

Models	**Score**
SGDclassifier	0.42400
Random Forest	0.38000
XGBoost	0.34376
Siamese BiLSTM with NLP features	**0.16515**
Fine-Tune BERT	0.27835

Table 3: Models Score.

- The result submissions are evaluated on the log loss between the predicted values and the ground truth.
- The best score this paper got is 0.16515



```python
with strategy.scope():
    # Encoded token ids from BERT tokenizer.
    input_ids = tf.keras.layers.Input(
        shape=(max_length,), dtype=tf.int32, name="input_ids"
    )
    # Attention masks indicates to the model which tokens should be attended to.
    attention_masks = tf.keras.layers.Input(
        shape=(max_length,), dtype=tf.int32, name="attention_masks"
    )
    # Token type ids are binary masks identifying different sequences in the model.
    token_type_ids = tf.keras.layers.Input(
        shape=(max_length,), dtype=tf.int32, name="token_type_ids"
    )
    # Loading pretrained BERT model.
    bert_model = transformers.TFBertModel.from_pretrained("bert-base-uncased")
    # Freeze the BERT model to reuse the pretrained features without modifying them.
    bert_model.trainable = False

    sequence_output, pooled_output = bert_model(
        input_ids, attention_mask=attention_masks, token_type_ids=token_type_ids
    )
    # Add trainable layers on top of frozen layers to adapt the pretrained features on the new data.
    bi_lstm = tf.keras.layers.Bidirectional(
        tf.keras.layers.LSTM(64, return_sequences=True)
    )(sequence_output)
    # Applying hybrid pooling approach to bi_lstm sequence output.
    avg_pool = tf.keras.layers.GlobalAveragePooling1D()(bi_lstm)
    max_pool = tf.keras.layers.GlobalMaxPooling1D()(bi_lstm)
    concat = tf.keras.layers.concatenate([avg_pool, max_pool])
    dropout = tf.keras.layers.Dropout(0.3)(concat)
    output = tf.keras.layers.Dense(1, activation="sigmoid")(dropout)
    model = tf.keras.models.Model(
        inputs=[input_ids, attention_masks, token_type_ids], outputs=output
    )

    model.compile(
        optimizer=tf.keras.optimizers.Adam(),
        loss="binary_crossentropy",
        metrics=["acc"],
    )
```

Figure 19.

4 Conclusions and Future Work

4.1 Conclusions

By dealing with the task of duplicate question pair identification, this paper proposes 3 model from traditional machine learning aspect to deep learning aspect. As a result, Model 2 performs relatively best score among all models.

Effectively detecting duplicate questions not only saves time for seekers to find the best answer to their questions, but also reduces the effort of writers in terms of answering multiple versions of the same question.

4.1.1 Summary

- For the classification problem of sentence similarity on the qqp dataset, we proposed 3 models.
- Model 1 is a traditional machine learning model based on the feature engineering.
- Model 2 is deep learning based, we optimized a Siamese BiLSTM with NLP features. As a result, our new model achieved a better performance than the baseline.
- Model 3 is also deep learning based, we fine-tune BERT on the qqp task(downstream task), and implemented a Siamese BiLSTM based neural network structure for classification as our third model.
- All these models obtained a pretty good test score. Among these models, Model 2 achieves the most prominent effect.
- In the future, we will focus on optimizations on Model 3 and model stacking technologies to optimize our MODEL and achieve a even better result.

4.2 Future Work

4.2.1 Do more about Model 3:

The model scores **0.27835**. Although not better than our featured Siamese-BiLSTM model (because this model is not mature), there are some improvements will be done in the future:

- Add more features statistical & NLP features from as a supplement.
- Adjust the neural network structure after BERT

- Tuning on hyperparameters, adding training epoches (The result of Model 3 is obtained based on just 1 epoch training due to time and resources limits).

4.2.2 Do more about model stacking

Ensemble learning improves the results of machine learning by combining multiple models, which allows better prediction performance than a single model. Stacking is a method of ensemble learning, which can improve the accuracy of prediction. Stacking usually considers heterogeneous weak learners, learns them in parallel, and trains a 'meta model' to combine them, and outputs a final prediction result according to the prediction results of different weak models.

So in our project, the most clear next step would be to ensemble the best of each models to achieve a even better performance.

4.3 Contributions

Group Members	**Contributions**
YUAN Yan Zhe	Main coding on Model 1, Model 2 and Model 3. Video&Report.
WEN Ze	Participate in Feature Engineering on Model 1 Implement Improvement 3 on Model 2. PPT&Video&Report.
YU Jia Hui	Implement Siamese LSTM baseline Implement Improvement 1&3 on Model 2 PPT&Video&Report.

Table 4: Contributions.

REFERENCES

https://blog.csdn.net/u010657489/article/details/51952785, a.

https://github.com/hanxiao/bert-as-service#building-a-qa-semantic-search-engine-in-3
b.

https://spacy.io/models/en, c.

http://www.nltk.org/5, d.

https://radimrehurek.com/gensim/, e.

https://github.com/seatgeek/fuzzywuzzy#usage, f.

https://www.kaggle.com/amoyyean/lstm-with-glove, g.

https://huggingface.co/transformers/model_doc/bert.html, h.

https://keras.io/api/layers/recurrent_layers/lstm/, i.

https://keras.io/examples/nlp/semantic_similarity_with_bert/, j.

https://keras.io/examples/nlp/masked_language_modeling/, k.

https://www.sbert.net/docs/training/overview.html?highlight=get_word_embedding_dimension, l.

Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. 2016.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Zihan Chen, Hongbo Zhang, Xiaoji Zhang, and Leqi Zhao. Quora question pairs, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

Zainab Imtiaz, Muhammad Umer, Muhammad Ahmad, Saleem Ullah, Gyu Sang Choi, and Arif Mehmood. Duplicate questions pair detection using siamese malstm. *IEEE Access*, 8:21932–21942, 2020.

Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International conference on machine learning*, pp. 957–966, 2015.

Jonas Mueller and Aditya Thyagarajan. Siamese recurrent architectures for learning sentence similarity. In *thirtieth AAAI conference on artificial intelligence*, 2016.

Jeffrey Pennington, R. Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

Bhargav Srinivasa-Desikan. *Natural Language Processing and Computational Linguistics: A practical guide to text analysis with Python, Gensim, spaCy, and Keras*. Packt Publishing Ltd, 2018.

Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pp. 194–206. Springer, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Lingxi Zhang, Zhiyong Feng, Wei Ren, and Hong Luo. Siamese-based bilstm network for scratch source code similarity measuring. In *2020 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 1800–1805. IEEE, 2020.