PYTHON DEEP LEARNING PROJECT

Project Name: Finding Similar Question Pairs.

Submitted By,
DILEEP KUMAR DURGAM - 8,
SIREESHA PANDALA - 22

DECEMBER 2018

INTRODUCTION

Quora is a Huge manifesto Where ideas,innovations are published or shared to the world . It associates individuals that looking for information to the general population who have learning. There are a billions of individuals who are using Quora so it is common to have similar ideas and doubts,which is nothing but , Huge number of similar questions are asked. As numerous inquiries with a similar goal can make searchers invest more energy in seeking for the top solution to their requests and also make writers feel that they are answering multiple versions of the samequestions. Quora principles canonic questionings since they afford a enhance understanding to dynamic searchers and scholars, and suggests an incentive to mutual crowds in Future.

The objective of this project is to anticipate similarity between given pair of questions. The Questions in our data for is given by which contains group of features. The Questions provided by the literature professionals are not with 100% precision because genuine importance of questions cannot be predicted excatly.

Random Forest model is a classifier which the quora using now to classify identical texts. Here we have to use natural language processing which means formatting the text using nltk library followingly we have to perform innovative methods to predict similarity between the questions .

DATASET DESCRIPTION

The Features of the data are

- id : primary key of the Question pair.
- qid1, qid2 : ids of question1 and question2 (present in train info)
- question1, question2 – text included in questions
- is duplicate : variable which is a target, is 1 if both questions have maximum of same meaning, and 0 otherwise.

The train data:

	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

This is how the training data is given.

The test data looks as follows:

	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?

The test data only contains questions but not their id's as in train data, as you can see above.

A general report of the train data:

- The train data has 6 columns.
- Contains 404289 instances.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 404290 entries, 0 to 404289
Data columns (total 6 columns):
id             404290 non-null int64
qid1           404290 non-null int64
qid2           404290 non-null int64
question1      404289 non-null object
question2      404288 non-null object
is_duplicate   404290 non-null int64
dtypes: int64(4), object(2)
memory usage: 18.5+ MB
```

A general report of the train data:

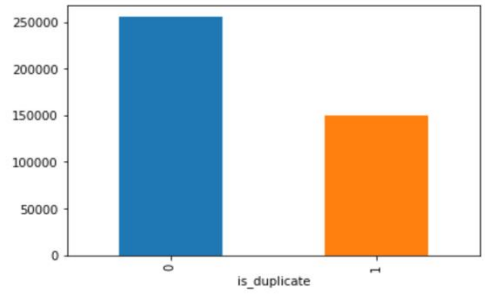
- The train data has 3 columns.
- Contains 1048572 instances.

.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 3 columns):
test_id      1048575 non-null int64
question1    1048574 non-null object
question2    1048572 non-null object
dtypes: int64(1), object(2)
memory usage: 24.0+ MB
```

- We plotted graph for the instance-"is duplicate", to know data distribution.
- The graph demonstrations that, Train data has 36.897% 1 's and 62.89% 0's.

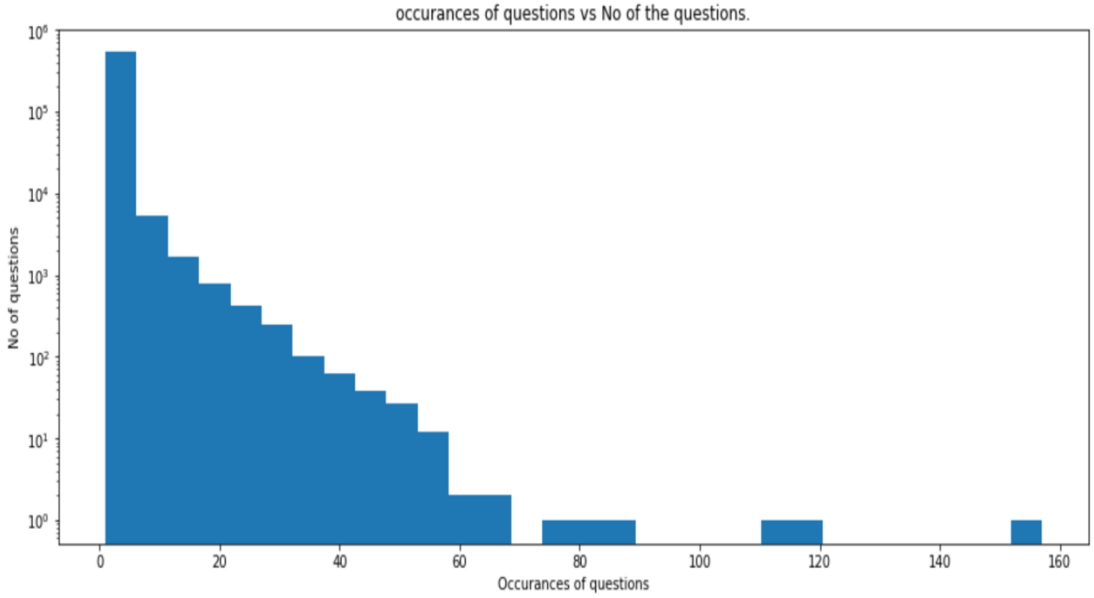
```
<matplotlib.axes._subplots.AxesSubplot at 0x1192515c0>
```



The plot shows the is_duplicate distribution in the train data.

- We plotted the number of questions vs frequency.
- Plot illuminates that there only few questions which are repeated more times, there are more questions which are repeated few times.

```
Text(0.5,1,'occurances of questions vs No of the questions.')
```



IMPLEMENTATION

Programming Languages Implemented:

- We used PYTHON (3.0) for implementing most of the tasks, as all our team mates are flexible using it and as python has all the required libraries inbuilt which are required for analysis and implementation of the task.
- We used the Anaconda Jupyter (a Python Notebook) to implement our code as it is quick at processing the data and also gives block wise implementation results.

Preliminary Implementation:

- Initially, we have performed some analysis on the data to check how the data is distributed, to help us extract useful features from the given data set. For this we used matplotlib library to draw some graphs between different features in the dataset. All this analysis is shown clearly in the python notebook attached at the end.
- From this analysis, we have observed that the about 36% of the training data are labeled 1 for is duplicate and rest are labeled 0.

```
train_duplicate_mean = train['is_duplicate'].mean()
print ("mean of train data is_duplicate column",train_duplicate_mean)
```

```
mean of train data is_duplicate column 0.369197853026293
```

- Also, there are only few questions that are being repeated more than once and most off the questions are just showing up once in the data.

```
question_id_1 = train['qid1'].tolist()
question_id_2 = train['qid2'].tolist()
question_id = pd.Series(question_id_1+question_id_2)
plt.figure(figsize=(15,6))
plt.hist(question_id.value_counts(), bins= 30)
plt.yscale('log', nonposy='clip')
plt.xlabel('Occurances of questions')
plt.ylabel('No of questions')
plt.title('occurances of questions vs No of the questions.')
```

- After getting an idea about how the data is distributed we applied Natural Language Processing techniques to extract words from the questions.
- From the extracted words, we are checking the percentage of common words between the two questions in a pair and adding this word share count to the feature set.

```
from nltk.corpus import stopwords as st
stopwords_set = set(st.words("english"))

def word_dict(sentence):
    question_words_dict = {}
    for word in sentence.lower().split():
        if word not in stopwords_set:
            question_words_dict[word] = 1
    return question_words_dict
def common_words_percentage(entry):
    question_1_words = word_dict(str(entry['question1']))
    question_2_words = word_dict(str(entry['question2']))

    if len(question_1_words) == 0 or len(question_2_words) == 0:
        return 0
    shared_in_q1 = [word for word in question_1_words.keys() if word in question_2_words]
    feature_Ratio = ( 2*len(shared_in_q1) )/(len(question_1_words)+len(question_2_words))
    return feature_Ratio
```

- From this feature, along with the is duplicate label is used as the training data for the initial implementations.
- For the classification task, we choose to use XG Boosting classifier since it is the one of best classifiers to achieve higher accuracy.
- We took log loss as the error metric, we have incorporated it as the error metric into our model.
- We split this training data accordingly to apply cross validation on the data set and pass it to the XG boosting classifier. • The classifiers trains until it attains a constant log loss on the validation data set for at least 50 rounds.

```
from sklearn.cross_validation import train_test_split

X_TrainData, X_ValidData, Y_TrainData, Y_ValidData = train_test_split(X_TrainData, Y_TrainData, test_size=0.20,
                                                                      random_state=4242)
```

- After training on this feature, we passed the test data to classify it and submitted this as an initial result to check the log loss of this model on the test data.

Extracting more Relevant Features:

- Since, here we are checking the similarity between two different sentences we tried to implement multiple natural language processing techniques like TF-IDF, jaccard similarity, cosine similarity.

Cosine Similarity:

```
from sklearn.metrics.pairwise import cosine_similarity as cs
import re, math
from collections import Counter

WORD = re.compile(r'\w+')
def _cosine_similarity(vector_1, vector_2):
     intersection = set(vector_1.keys()) & set(vector_2.keys())
     numerator = sum([vector_1[x] * vector_2[x] for x in intersection])

     sum1 = sum([vector_1[x]**2 for x in vector_1.keys()])
     sum2 = sum([vector_2[x]**2 for x in vector_2.keys()])
     denominator = math.sqrt(sum1) * math.sqrt(sum2)

     if not denominator:
        return 0.0
     else:
        return float(numerator) / denominator

def sentence_transform(sentence):
     words = WORD.findall(sentence)
     return Counter(words)

def cosine_sim(row):
    vector1 = sentence_transform(str(row['question1']))
    vector2 = sentence_transform(str(row['question2']))
    sim = _cosine_similarity(vector1,vector2)
    return sim

X_TrainData['cosine_sim'] = train.apply(cosine_sim,axis = 1,raw = True )
X_TestData['cosine_sim'] = test.apply(cosine_sim,axis = 1,raw = True )
```

Jaccard Similarity Implementation:

- we implemented Jaccard similarity using our own code(pure-code) without using any libraries.
- we have divided intersection of the common words and total words between the two questions that has to be compared to get the jaccard similarity.
- we used nltk library to tokenize the sentence

```
import nltk
def jaccard_similarity_coefficient(row):
    if (type(row['question1']) is str) and (type(row['question2']) is str):
        words_1 = row['question1'].lower().split()
        words_2 = row['question2'].lower().split()
    else:
        words_1 = nltk.word_tokenize(str(row['question1']))
        words_2 = nltk.word_tokenize(str(row['question2']))

    joint_words = set(words_1).union(set(words_2))
    intersection_words = set(words_1).intersection(set(words_2))
    return len(intersection_words)/len(joint_words)
```

TF-IDF Implementation:

- Here, we implemented TF-IDF using the sk-Learn Library.
- While assigning this library we have tried on different parameters as found the following to work better.

```
def tfidf_weights(entry):
    question_1_words = word_dict(str(entry['question1']))
    question_2_words = word_dict(str(entry['question2']))
    if len(question_1_words) == 0 or len(question_2_words) == 0:
        return 0

    common_wts_1 = [weights.get(w, 0) for w in question_1_words.keys() if w in question_2_words]
    common_wts_2 = [weights.get(w, 0) for w in question_2_words.keys() if w in question_2_words]
    common_wts = common_wts_1 + common_wts_2
    whole_wts = [weights.get(w, 0) for w in question_1_words] + [weights.get(w, 0) for w in question_2_words]

    feature_tfidf = np.sum(common_wts) / np.sum(whole_wts)
    return feature_tfidf
```

- We took the “min-df” = ’50’, which means that word which are such that inappropriate words are eliminated. Maximum number of features as ’300000’ and n gram range is in between (1,10).

```
list_of_questions = (train['question1'].str.lower().astype('U').tolist() +
                     train['question2'].str.lower().astype('U').tolist())

from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df = 50,max_features = 3000000,ngram_range = (1,10))
X = vectorizer.fit_transform(list_of_questions)
idf = vectorizer.idf_
weights = (dict(zip(vectorizer.get_feature_names(), idf)))
```

- After TF-IDF scores for the words have been applied and stored in a dictionary, we used these weights to find the common word share between two sentences.
- By performing this we were able to gain a more relevant feature rather than normal word share.
- Even after applying TF-IDF, we observed that the improvement in the log loss was nominal.

```
train = train.dropna(axis=0, how='any')
test = test.dropna(axis=0, how='any')
X_TrainData = pd.DataFrame()
X_TestData = pd.DataFrame()
X_TrainData['common_word_percent'] = train.apply(common_words_percentage, axis=1, raw=True)
X_TrainData['feature_ifidf'] = train.apply(tfidf_weights, axis = 1, raw = True)
Y_TrainData = train['is_duplicate'].values
X_TestData['common_word_percent'] = test.apply(common_words_percentage, axis = 1, raw = True)
X_TestData['feature_ifidf'] = test.apply(tfidf_weights, axis = 1, raw = True)
```

Applying XG Boosting:

- After extracting all the features, we split the data into training and validation data using train test split library available in Sk Learn. Here, we took 20% of data for validation.
- Now, for implementing XG Boosting we tried various parameters that were available for XG Boosting, we implemented XG Boosting using the following parameters.
- We gave Learning task for XG Boosting as binary: logistic this applies logistic regression for binary classification, outputs probability.
- The evaluation metric is chosen to be log loss.
- The Learning rate ('eta') for the classifier is chosen to be 0.02.
- The max depth of the tree as 4.

We tried changing the size but there was no improvement in the log loss.

- Now, by applying these attributes we trained our model on the training data by setting the early stopping rounds as 50 and number of iterations as 500.
- The log loss on the validation data after training was around 0.4432.

```
import xgboost as xgb

xg_TrainData = xgb.DMatrix(X_TrainData, label=Y_TrainData)
xg_ValidData = xgb.DMatrix(X_ValidData, label=Y_ValidData)

watchlist = [(xg_TrainData, 'train'), (xg_ValidData, 'valid')]

bst = xgb.train({'objective':'binary:logistic','eval_metric':'logloss','eta':0.02,'max_depth' :5},
                xg_TrainData, 500, watchlist, early_stopping_rounds=50, verbose_eval=10)
```

Features

- Features we are created are , TFIDF word similarity, Jaccardian distance , Common word percentage, Cosine Similarity

- Below figure has the head of final features extracted

	common_word_percent	feature_ifidf	Jacard_Distance	cosine_sim
0	0.727273	0.903160	0.769231	0.944911
1	0.307692	0.817249	0.250000	0.536875
2	0.363636	0.685127	0.200000	0.253546
3	0.000000	0.594645	0.000000	0.000000
4	0.000000	0.497811	0.111111	0.419314
5	0.470588	0.683637	0.347826	0.556415
6	0.000000	0.865550	0.000000	0.000000
7	0.500000	0.598609	0.333333	0.503953
8	0.500000	1.000000	0.600000	0.801784
9	0.363636	0.792924	0.200000	0.444444
10	0.000000	0.419755	0.041667	0.133333
11	0.571429	0.622064	0.416667	0.471405
12	1.000000	1.000000	0.666667	0.668153
13	0.571429	0.804868	0.625000	0.925820
14	0.818182	1.000000	0.833333	0.957447
15	0.315789	0.694276	0.148148	0.218218
17	0.000000	0.339680	0.052632	0.102062

Results

Log Loss:

• Our final train model have few more features than orginally got from data set we have a log loss of 0.44291 on test data.

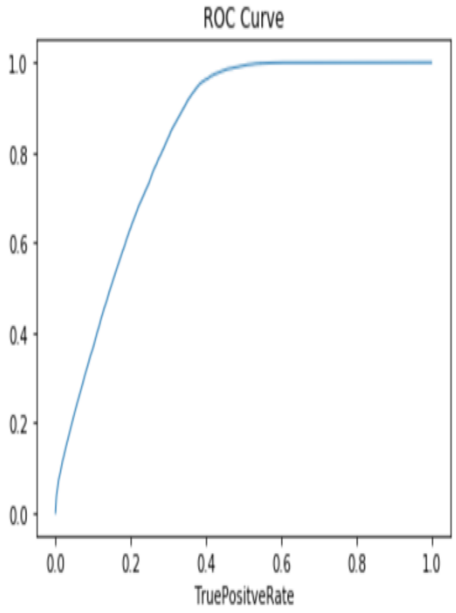
• Below you can see the training process of XGBoost classifier and the final log loss on the validation data.

```
[17:58:14] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 56 extra nodes, 0 pruned nodes, max_depth=5
[17:58:14] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 62 extra nodes, 0 pruned nodes, max_depth=5
[17:58:14] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 38 extra nodes, 0 pruned nodes, max_depth=5
[17:58:15] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 58 extra nodes, 0 pruned nodes, max_depth=5
[17:58:15] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 48 extra nodes, 0 pruned nodes, max_depth=5
[17:58:15] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 46 extra nodes, 0 pruned nodes, max_depth=5

[17:58:15] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 56 extra nodes, 0 pruned nodes, max_depth=5
[17:58:15] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 58 extra nodes, 0 pruned nodes, max_depth=5
[490]   train-logloss:0.439318  valid-logloss:0.440654
[17:58:16] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 48 extra nodes, 0 pruned nodes, max_depth=5
[17:58:16] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 62 extra nodes, 0 pruned nodes, max_depth=5
[17:58:16] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 46 extra nodes, 0 pruned nodes, max_depth=5
[17:58:16] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 58 extra nodes, 0 pruned nodes, max_depth=5
[17:58:16] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 60 extra nodes, 0 pruned nodes, max_depth=5
[17:58:17] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 60 extra nodes, 0 pruned nodes, max_depth=5
[17:58:17] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 62 extra nodes, 0 pruned nodes, max_depth=5
[17:58:17] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 62 extra nodes, 0 pruned nodes, max_depth=5
[17:58:17] src/tree/updater_prune.cc:74: tree pruning end, 1 roots, 60 extra nodes, 0 pruned nodes, max_depth=5
[499]   train-logloss:0.439143  valid-logloss:0.44053
```

ROC Curve:

- We plotted the ROC curve to know the functioning of our classifier.
- Area under Curve ROC(AUC) = 0.834

```
Area under Roc Curve 0.8341650564840387

Text(0.5,1,'ROC Curve')
```

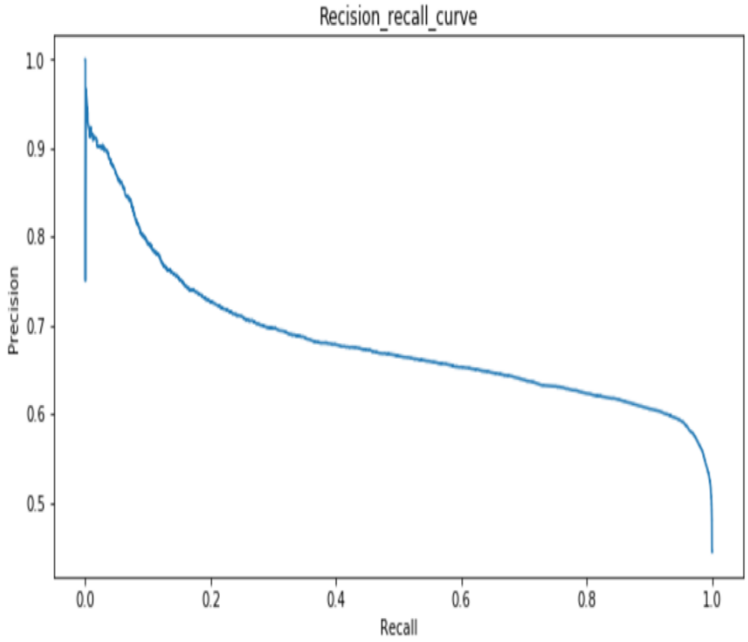


Precision Recall Curve:

• We have plotted Precision Recall curve in the graph and we got area of 0.682

• This plots Recall on X-axis and Precision on Y-axis.

Acuracy through precision_recall_curve 0.6821685636932522



Conclusion:

We gone through innumerous of procedures which were utilized to find a closeness of pair of questions, since predicting the real significance of sentennce is ceaselessly unclear, we might not know the 100% exactness of pair sentences are practically identical. We processed text and added similarities as features into new columns to get more accurate model and Training is done on XGboost classifier to predict similarity of the questions. By doing classification on test data utilizing the features we created produced actuate a test log loss of around 0.4.