THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

UML – 501
Machine Learning

Question Pair Similarity Inference

Instructor : Dr. Parteek Bhatia
Submitted by : Abhishek Prajapat

Table of Contents

INTRODUCTION

Quora is a place to gain and share knowledge—about anything. It’s a platform to ask questions and connect with people who contribute unique insights and quality answers. This 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 similarly worded 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. Quora values canonical questions because they provide a better experience to active seekers and writers, and offer more value to both of these groups in the long term.

Due to this it becomes very important to correctly identify which questions are similar and hence process that question only once.

PROBLEM STATEMENT

- Identify which questions asked on Quora are duplicates of questions that have already been asked.

- This could be useful to instantly provide answers to questions that have already been answered.

- We are tasked with predicting whether a pair of questions are duplicates or not.

- Data Samples :-

  - Question1 = "How can I be a good geologist?"
    Question2 = "What should I do to be a great geologist?"
    Is_duplicate = "1"

  - Question1 = "What is the step by step guide to invest in the share market in india?"
    Question2 = "What is the step by step guide to invest in the share market?"
    Is_duplicate = "0"

Here “1” means that the questions are duplicate and “0” means they are not duplicates.

PROBLEM CONSTRAINTS

- The cost of misclassification can be very high.
- We want a probability of a pair of questions to be duplicates so that we can choose any threshold.
- No strict latency concerns which inference.
- Interpretability is partially important.

METHODOLOGY

1. Data Collection:

Source: Kaggle.

Data Overview:-

- Data is stored in a file “train.csv”

- Contains Five Columns - “qid1”, “qid2”, “question1”, “question2”, “is_duplicate”

  - “qid1” - id for question1
  - “qid1” - id for question2
  - “question1” - raw text of question1
  - “question2” - raw text of question2
  - “is_duplicate” - label (ground truth)

- Size of “train.csv” - 60MB

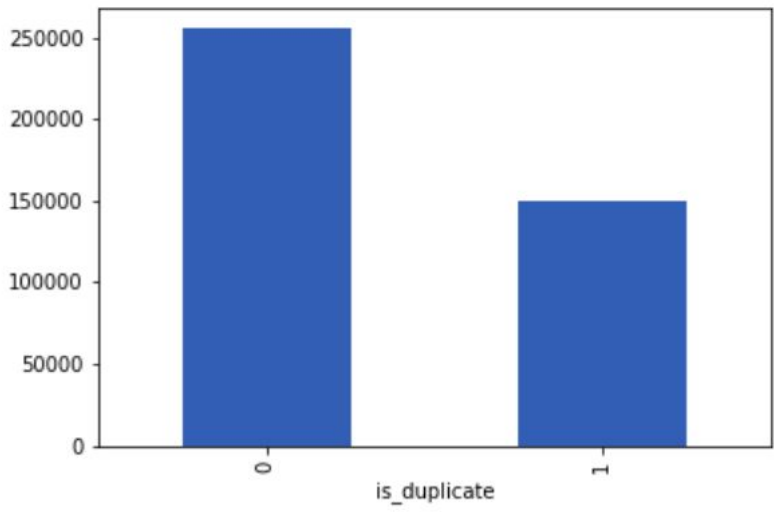
- Number of rows in “train.csv” - 404,290

2. DATA ANALYSIS and PROBLEM MAPPING:

- Checked for duplicate rows
- Removed rows containing null values
- The analysis of label distribution came out to be following (63.08% are not duplicates and 36.92% are duplicates)

```
df.groupby("is_duplicate")['id'].count().plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f194261d650>
```



- After these initial insights the problem was mapped to be a binary classification problem.
- Metrics used: log-loss and Binary Confusion Matrix.

3. DATA PRE-PROCESSING:

Before any sort of data cleaning some numeric features were extracted from the raw text of questions which are was following:-

- **freq_qid1** = Frequency of qid1's
- **freq_qid2** = Frequency of qid2's

- **q1len** = Length of q1
- **q2len** = Length of q2

- **q1_n_words** = Number of words in Question 1
- **q2_n_words** = Number of words in Question 2

- **word_Common** = (Number of common unique words in Question 1 and Question 2)
- **word_Total** =(Total num of words in Question 1 + Total num of words in Question 2)

- **word_share** = (word_common)/(word_Total)

- **freq_q1+freq_q2** = sum total of frequency of qid1 and qid2

- **freq_q1-freq_q2** = absolute difference of frequency of qid1 and qid2

After the following cleaning steps on raw text:-

- HTML removal
- Punctuations removal
- Stemming
- Stopwords removal
- Expanding concatenations

Some more features were extracted using “fuzzy wuzzy” [2] :-

- **cwc_min** : Ratio of common_word_count to min length 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 length 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 length 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 length 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 length 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 length of token count of Q1 and Q2
  ctc_max = common_token_count / (max(len(q1_tokens), len(q2_tokens))

- **last_word_eq** : Check if First word of both questions is equal or not
  last_word_eq = int(q1_tokens[-1] == q2_tokens[-1])

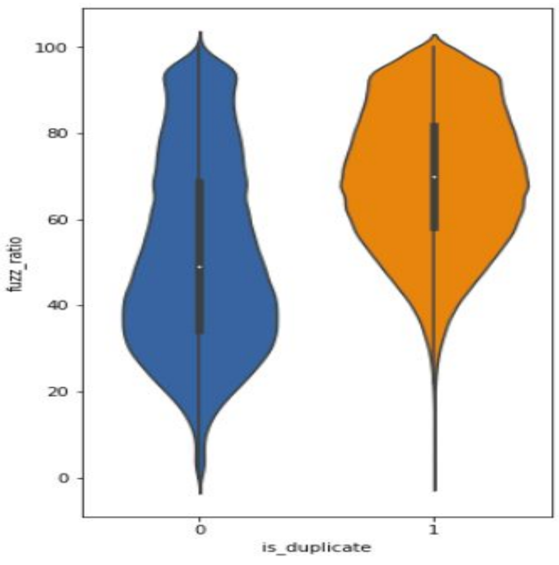
- **first_word_eq** : Check if First word of both questions is equal or not
  first_word_eq = int(q1_tokens[0] == q2_tokens[0])

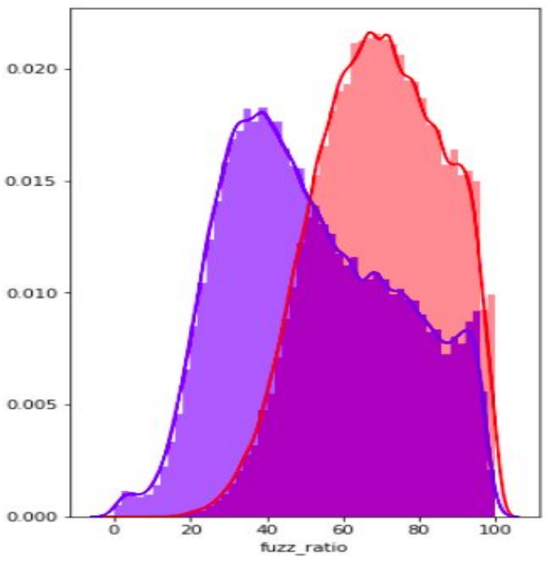
- **abs_len_diff** : Abs. length difference
  abs_len_diff = abs(len(q1_tokens) – len(q2_tokens))

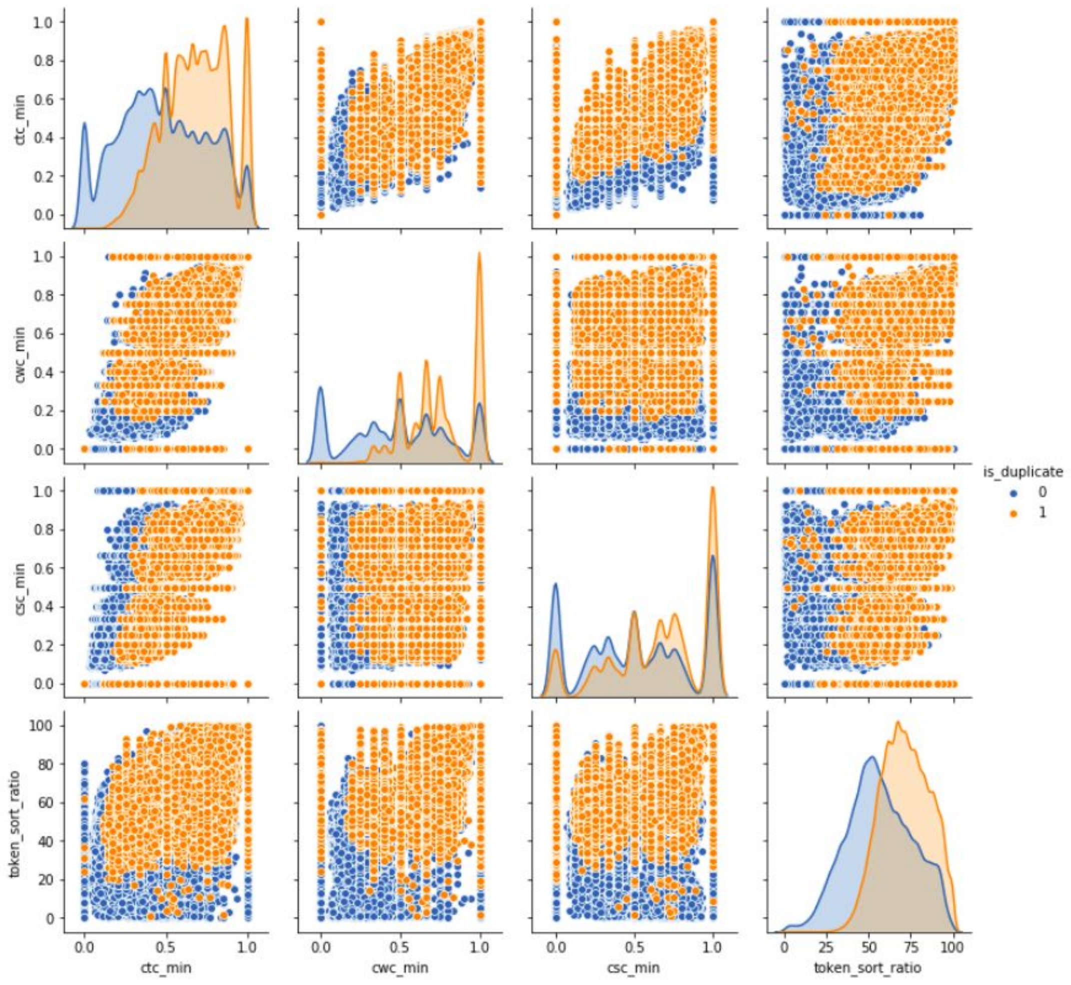
- **mean_len** : Average Token Length of both Questions
  mean_len = (len(q1_tokens) + len(q2_tokens))/2

- **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

- Following are the visual results:-







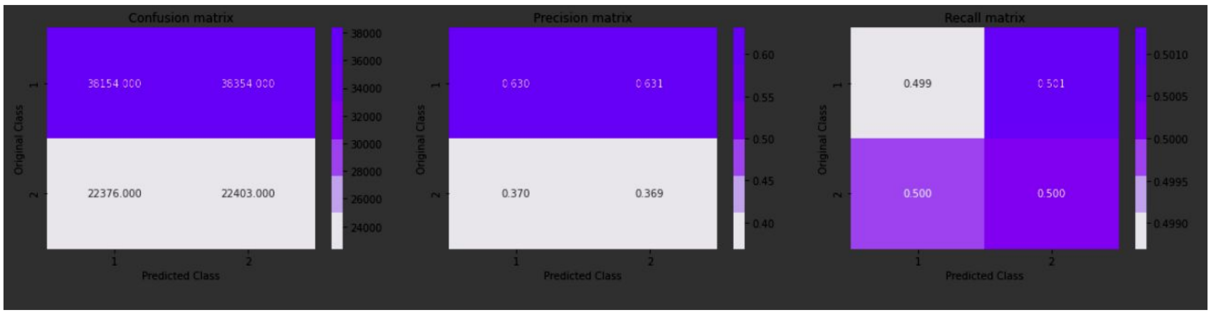
These features provide significant information for use while identifying duplicates.

- After this we analyse correlation of extracted features and select 10 features based on Recursive Feature Elimination with Logistic-Regression as our base model.

- With this we proceed for making a baseline model.

4. MODELING:

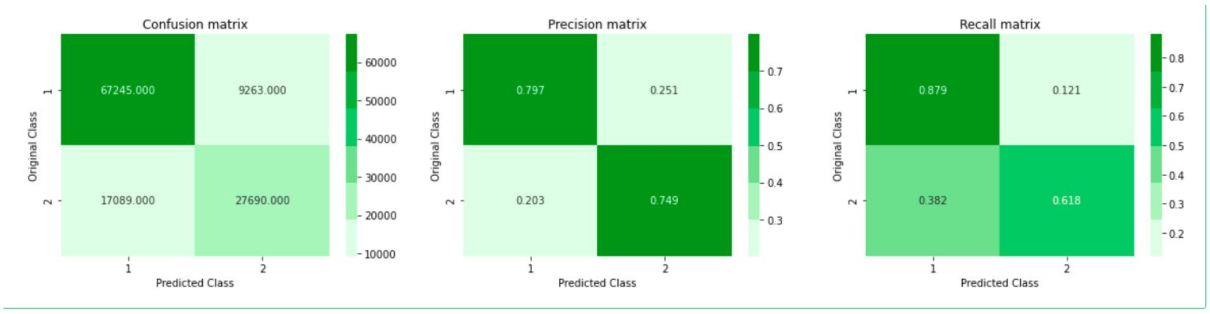
- First we set a Random model as our baseline model and get a log-loss of 0.8871 and with the following confusion matrix.



- We then use "Glove-wiki-gigaword-100" for converting word to vectors. It converts a word available in the file to a 100 dimensional vector.

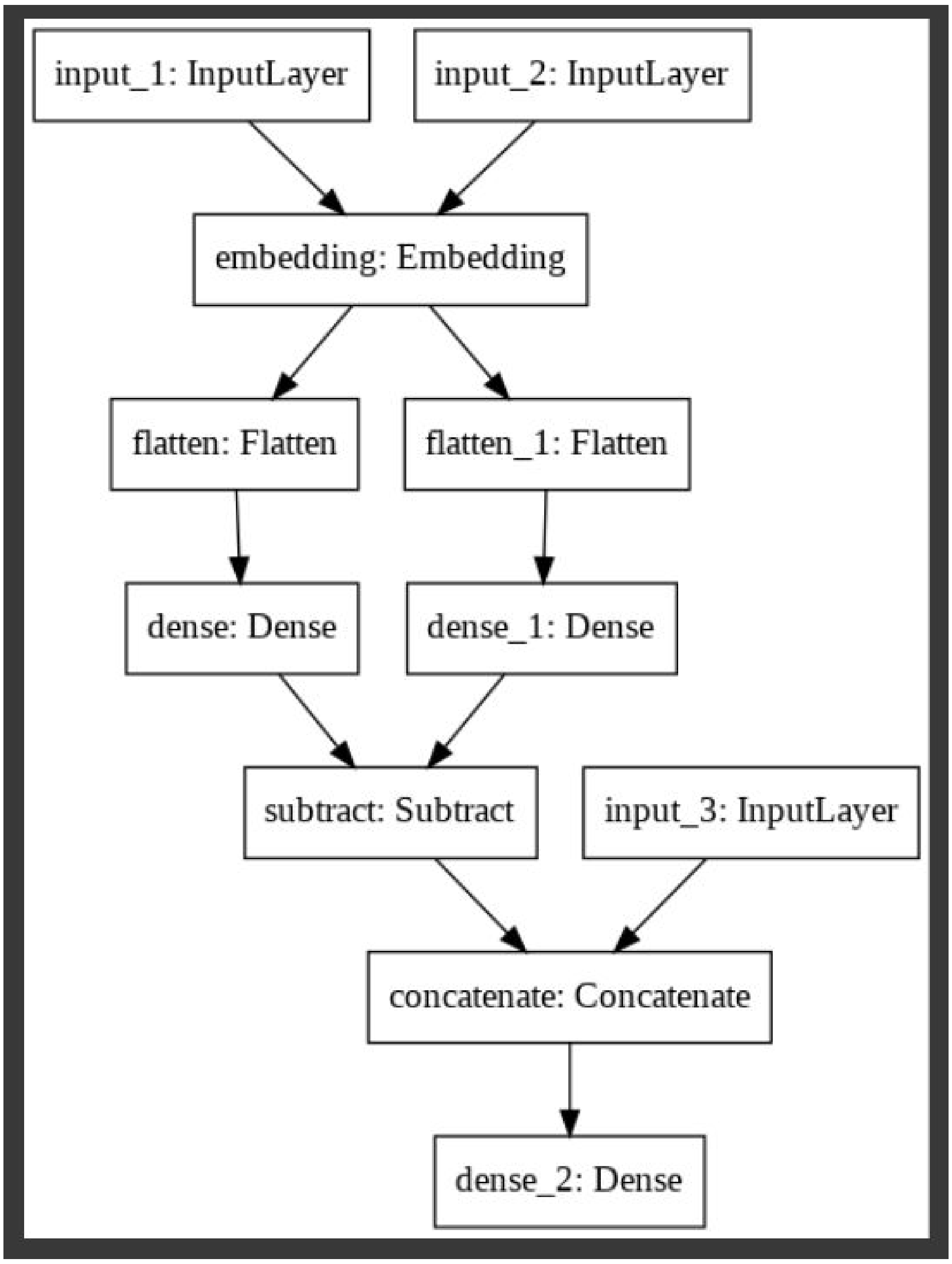
- We then make sentence vectors for all question adding individual word vectors.

- After this we used Logistic Regression with alpha=1 and get a log-loss of 0.4327 on training data and 0.5200 on test data and the following confusion matrix.

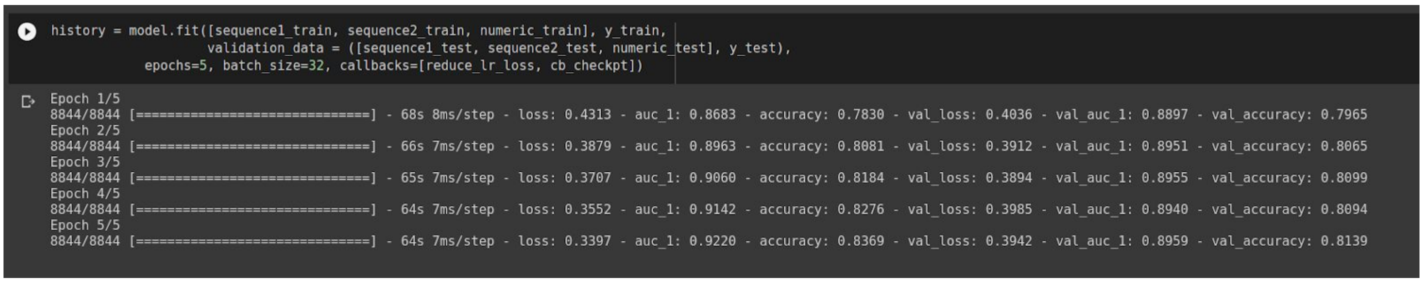


- As expected results are better than baseline model but are not enough and as there is no overfitting this indicates we need more complex models.

- After the following results we proceed for a neural network based architecture. For computational reasons we choose a relatively shallow neural network.



Following are the observed results.



```
history = model.fit([sequence1_train, sequence2_train, numeric_train], y_train,
                    validation_data = ([sequence1_test, sequence2_test, numeric_test], y_test),
               epochs=5, batch_size=32, callbacks=[reduce_lr_loss, cb_checkpt])
```

```
Epoch 1/5
8844/8844 [==============================] - 68s 8ms/step - loss: 0.4313 - auc_1: 0.8683 - accuracy: 0.7830 - val_loss: 0.4036 - val_auc_1: 0.8897 - val_accuracy: 0.7965
Epoch 2/5
8844/8844 [==============================] - 66s 7ms/step - loss: 0.3879 - auc_1: 0.8963 - accuracy: 0.8081 - val_loss: 0.3912 - val_auc_1: 0.8951 - val_accuracy: 0.8065
Epoch 3/5
8844/8844 [==============================] - 65s 7ms/step - loss: 0.3707 - auc_1: 0.9060 - accuracy: 0.8184 - val_loss: 0.3894 - val_auc_1: 0.8955 - val_accuracy: 0.8099
Epoch 4/5
8844/8844 [==============================] - 64s 7ms/step - loss: 0.3552 - auc_1: 0.9142 - accuracy: 0.8276 - val_loss: 0.3985 - val_auc_1: 0.8940 - val_accuracy: 0.8094
Epoch 5/5
8844/8844 [==============================] - 64s 7ms/step - loss: 0.3397 - auc_1: 0.9220 - accuracy: 0.8369 - val_loss: 0.3942 - val_auc_1: 0.8959 - val_accuracy: 0.8139
```

Our neural network gave us a best test-loss of 0.3894 and that is also the best in all our models but there are some points to take note of.

1. The parameters of the Embedding layer were not put as “trainable”.

2. During the padding of sequences there is loss of information due to truncating.

3. There is overfitting in the model and hence we should add regularization.

4. input_1 and input_2 and input_3 refer to “question1_sequence” , “question2_sequence” and “extracted numeric features”.

5. In creating sequences only top 20,000 words were considered and other words lead to information loss.

6. In layers “dense” and “dense_1” only 64 units were used in the model.

LIMITATIONS and OVERCOMES

- Computational cost is the biggest limitation due to which we can’t utilize large models with high-dimensional word-vectors.
- We are using pre-trained embeddings without fine-tuning.
- Spelling mistakes, collections of words (ex. Free time, new york) also pose problems.
- Adversarial examples are near impossible to correctly classify.

OVERCOMES:-

- Train on large and better systems/machines.
- Use Attention and Transformers for getting features.
- Train on Adversarial Examples.
- Use Augmentation by replacing the words with their synonyms.
- Correct spelling mistakes and other cleaning.
- Fine-tune word vectors for your problem.

FUTURE SCOPE

1. It could be used in every platform similar to quora. Example-stack overflow.

2. We could use the Fine-tuned Embedding in other problems.

3. Feature extracted methods of the following problem can be utilized over other problems.

4. The architecture of the model can be used while doing similarity analysis over other types of datasets. ex.- One shot Face-recognition.

REFERENCES

- https://www.kaggle.com/c/quora-question-pairs
- https://github.com/seatgeek/fuzzywuzzy#usage
- https://www.tensorflow.org/tutorials
- https://www.tensorflow.org/tutorials/text/word_embeddings
- https://radimrehurek.com/gensim/apiref.html