CAPSTONE PROJECT

MACHINE LEARNING ENGINEER NANODEGREE

SAMUEL BAAFI BOAKYE
SAMBETHSLIM@GMAIL.COM

JULY 2, 2018

PROJECT OVERVIEW

Most of the data we deal with from day to day is in the text form. It has become very imperative to find ways to extract information from language or text. Natural language processing is the field that is trying to apply machine learning and text feature extraction algorithms to be able to create translations from one language to another, sentiment analysis, speech recognition, question answering and many other applications yet to be discovered.

This project seeks to find similarities in questions from the Quora website. I was able to apply machine learning algorithm that could confidently determine a duplicate question from a non-duplicate question. An xgboost classifier algorithm was used and an Long short-term memory neural network was applied to see how it performs.

The dataset was obtained from the kaggle[1] with human generated labels. Although these labels are not 100% accurate as will be discussed later, it was assumed to be okay enough to create a model out of it.

As an avid user of the quora website and someone who has always been interested in working with text data I find it very unfortunate to find so many duplicated questions on the quora platform, which makes finding the right information very difficult. So I believe it's very important to find a way to solve this problem.

PROBLEM STATEMENT

Having duplicated questions can only make it difficult for users to find the answers they need. Some questions may have longer sentences, others may be shorter but could mean the same thing. Clearly to manually root out duplicated questions will be difficult. What Quora does to solve this problem is to use an ensemble machine learning algorithm called Random Forest which basically combines a number of Decision Trees

to build a stronger model. Questions are sentences and sentences are made up of words. Now how can you tell if question A is the same as question B? You may look at the length of the questions but some questions may be dissimilar in length but similar in meaning.

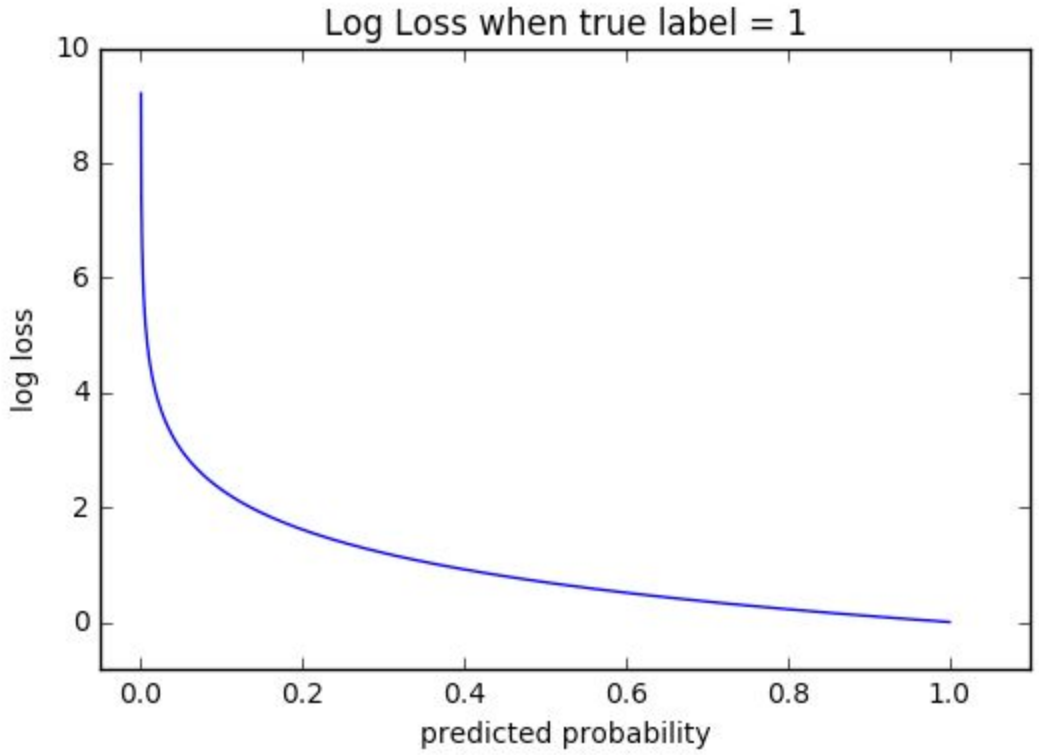
The plan of attack was as follows;

1. Get the right dataset from kaggle with real duplicated and unduplicated questions from Quora.
2. Preprocess the data by removing punctuations, white spaces, tokenization, normalization and removing stop words.
3. The next was to perform feature extraction by extracting some text metadata like question lengths and difference in lengths and also with the help of the fuzzuwuzzy[2] library extraction some features like partial ratio, token set ratio and others.
4. Gain some insight into the data with new features to see how wrong or right the human generated labels are.
5. Gain another feature from the data using word2vec with the help of the Global Word Vectors Representation file[3] (glove.6b.300d.txt) and run its through some classifier algorithms and see how it performs.
6. Add more features using the Term Frequency - Inverse Document Frequency (TF_IDF) and run it through the same classifier algorithms and see how that performs too.
7. Lastly, the dataset with be run through a recurrent neural network, specifically an LSTM to see if it performs better there.

METRICS

Logarithmic loss, commonly known as log loss (similar to cross-entropy used in the neural network) measures performance of a classification model where the input predictions is a probability between 0 and 1. The aim of using a log loss metric is to have it reduced. Hence, a really good model has a log loss of 0. Compared to accuracy which takes the count of right predictions where the predictions equal the actual values, log loss takes into account the uncertainty of the prediction. For instance, accuracy has a yes and no nature so it predicts with certainty but log loss has doubts which makes it a

much better metric.



As can be seen above, when the model makes a prediction, if the prediction is wrong it is punished so as the predicted probability decreases, the log loss increases.

In binary classification (M=2), the formula equals:

$$-(y \log(p) + (1-y) \log(1-p))$$

For example, given a class label of 1 and a predicted probability of .25, using the formula above we can calculate the log loss:

$$-(1 \log(.25) + (1-1) \log(1-.25))$$

$$-(\log(.25) + 0 \log(.75))$$

$$-\log(.25)$$

Similarly given a class label of 0 and a predicted probability of .25, we can calculate log loss as:

$$-(0 \log(.25) + (1-0) \log(0-.25))$$

$$-(1 \log(-.25))$$

$$-\log(-.25)$$

ANALYSIS

DATA EXPLORATION

The dataset used has about 400000 rows with 6 columns. The 3 important columns in the dataset are question1, question2 and is_duplicate. question1 and question2 columns contain real questions extracted from the quora website. These questions are paired so the first row of questions in question1 and question2 are duplicates according to the is_duplicate label. The questions in the dataset contain some noise like unnecessary whitespaces. Since the is_duplicate column was human generated it's also likely to contain some noise but this project was run with the assumption that the human labelling of the dataset is 100% accurate. Some people who worked on this dataset took into consideration the columns quid1 and quid2 to run their model but I find that to be a little overreaching because I believe duplicates questions should be based of the content and meaning in the questions and not their ids.

The following are the field of the dataset:

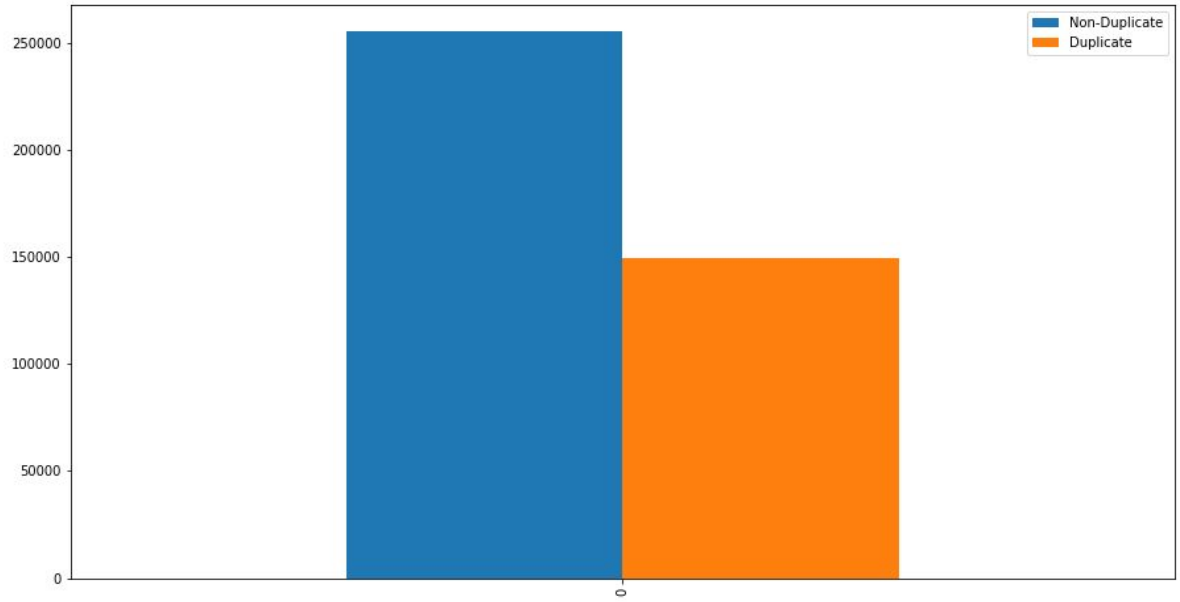
- id : the row ids
- quid1 : corresponding to the question id in the question1 column (integer)
- quid2 : corresponding to the question id in the question2 column (integer)
- question1 : real questions from quora (string)
- question2 : real questions from quora (string)
- Is_duplicate : binary values of 0s and 1s with 1 being that the question pairs are duplicates. (binary integers)

	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

Sample view of the quora_question_pairs dataset

EXPLORATORY VISUALIZATION

From the table below it can be seen that lots for the questions are non-duplicates (255027) and few are duplicates (149263). The percentage of duplicated question pairs is about 36.9%.



Non-duplicates in blue and Duplicates in orange.

TEXT PREPROCESSING

Text preprocessing techniques were applied to the questions to prepare them for feature extraction. Five text preprocessing techniques were created and applied.

1. Noise was removed. In relation to the dataset, unnecessary whitespaces where considered as noise so they were removed.
2. Contractions where also replaced with their full words. For instance, when the function finds a word like ‘didn't’, it will transformed to ‘did not’.
3. Afterwards the questions were tokenized which mean each question was separated into smaller components and transformed into a list of words.
4. At normalization stage, a number techniques were created and applied;
    a. Words were lemmatized. Lemmatization is to convert words to their base dictionary form. For instance the word ‘better’ becomes ‘good’.
    b. Non-ascii characters were removed
    c. Numbers in their symbolic forms where replaced with text.
    d. All words were also converted to lowercase and punctuations removed.

5. The last process was to remove stop words from tokenized text.

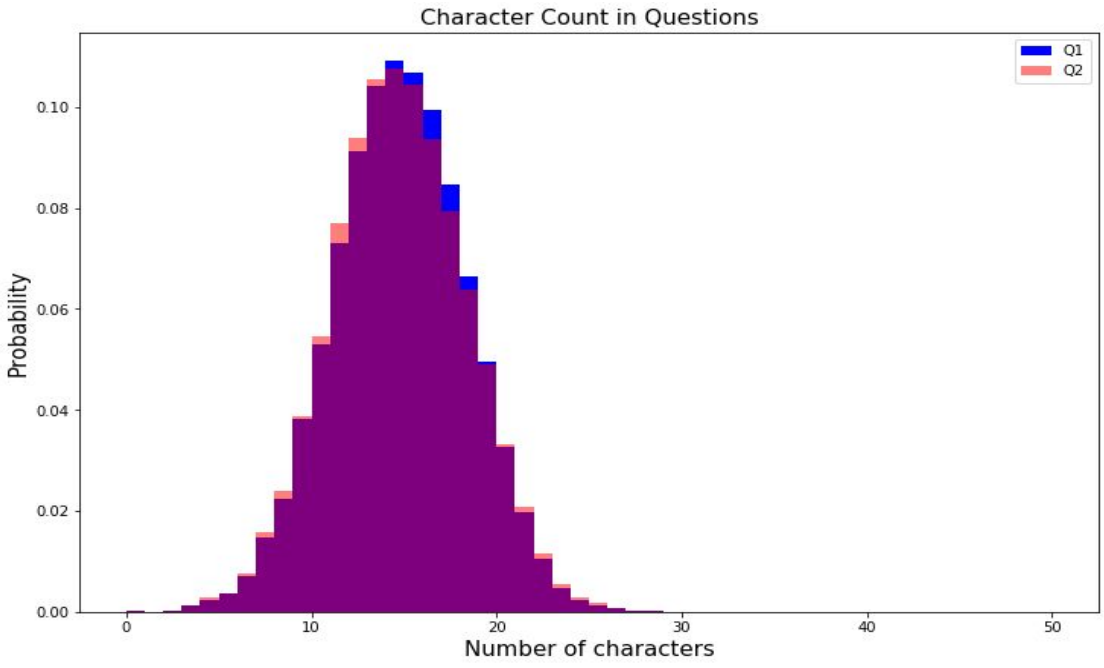
Before the feature extraction stage the unwanted columns where removed from the dataset. The id, quid1 and quid2 columns where removed from the dataset as they were seen to be less significant.

FEATURE EXTRACTION/ENGINEERING

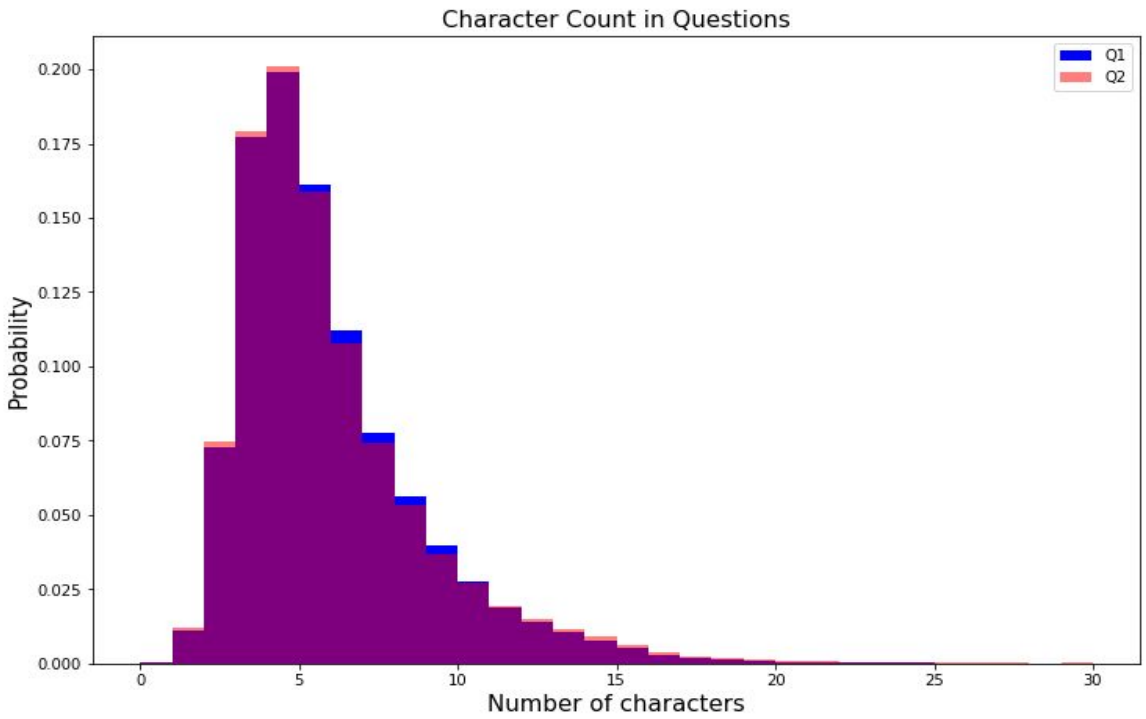
14 features were extracted from the preprocessed dataset. These are the features;

- Length of questions, to find the number of words in each question
- Difference in questions length, when question1 is subtracted from question2
- Common words, to find the intersection between questions with similar words
- Length of the characters in each question
- Fuzzywuzzy[2] features which checks string similarity in variours ways
    - Ratio
    - Partial ratio
    - Token sort ratio
    - Token set ratio
    - Qratio
    - Wratio
    - Partial token sort ratio
    - Partial token set ratio

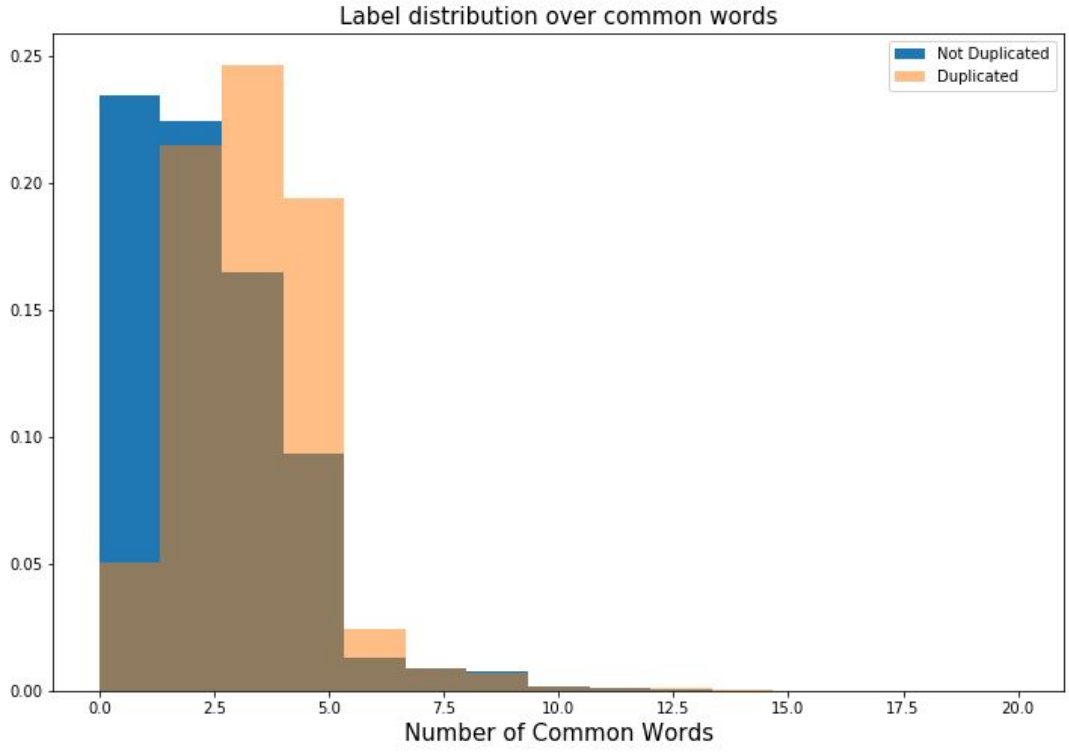
VISUALIZING EXTRACTED FEATURES



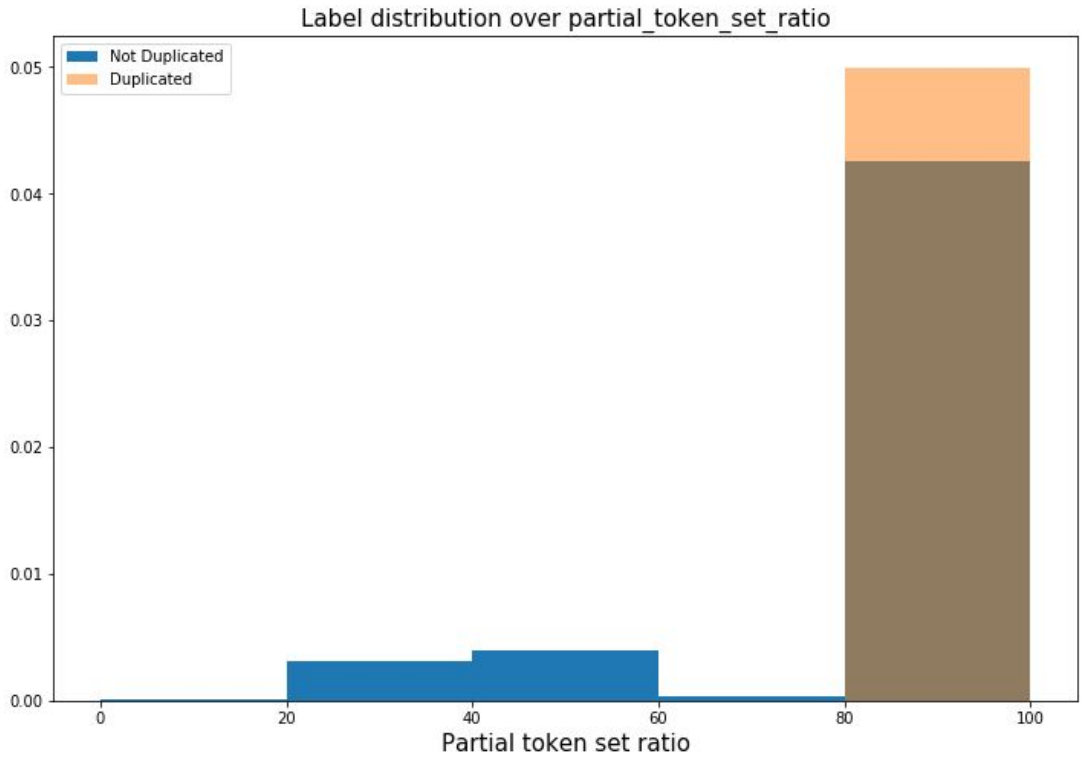
Most questions have characters between 10 and 20, maybe about 15 characters.



Above, most questions have 5 words, that's if stopwords are removed.



As expected duplicated question have a higher number of common words.

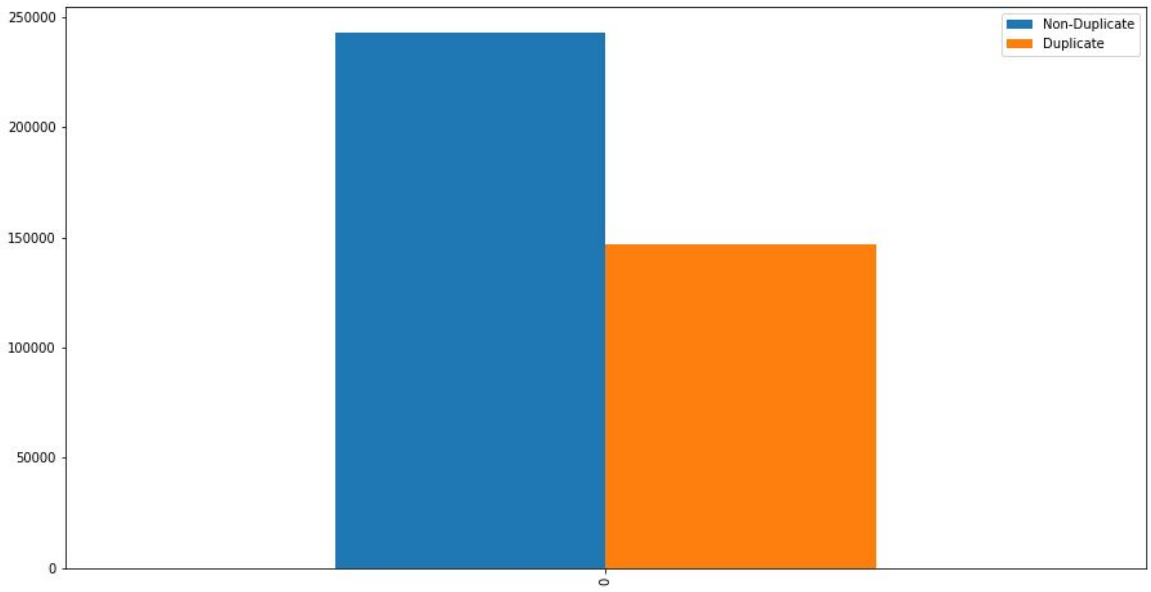


As expected a higher number of duplicated question have a high partial_token_set ratio.

Next, I wanted to check the accuracy of human labels against the accuracy of extracted features to see what they think of a duplicated question.

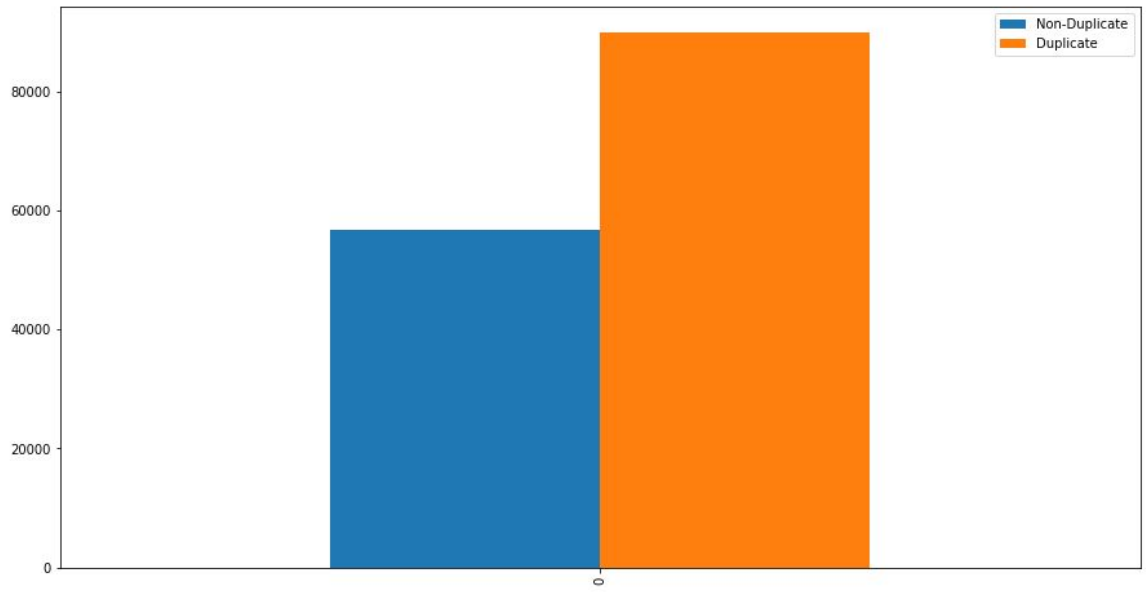
To begin, some assumptions were made.

- Let's say similar question pairs have a difference in length below or equal to 5. So assuming the assumption is correct, we should have lot of question pairs with difference in length below or equal to 5 labelled as duplicates.



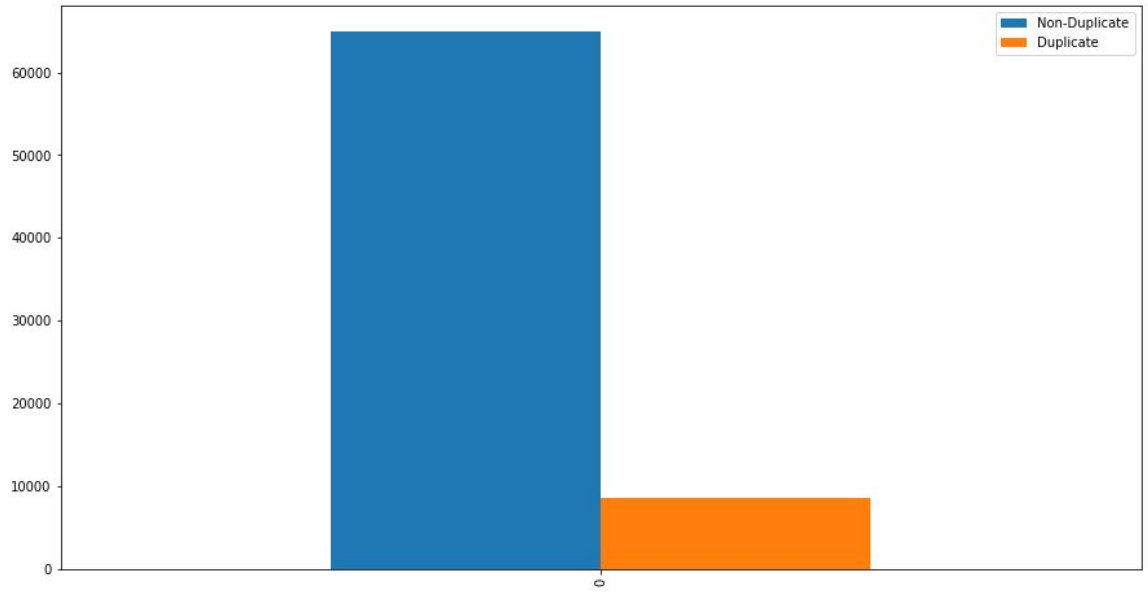
Over 20000 question pairs labelled wrongly according to the assumption

- Let's assume similar question pairs have a partial ratio above the 75th percentile.



Over 80000 question pairs labelled correctly according to the assumption.

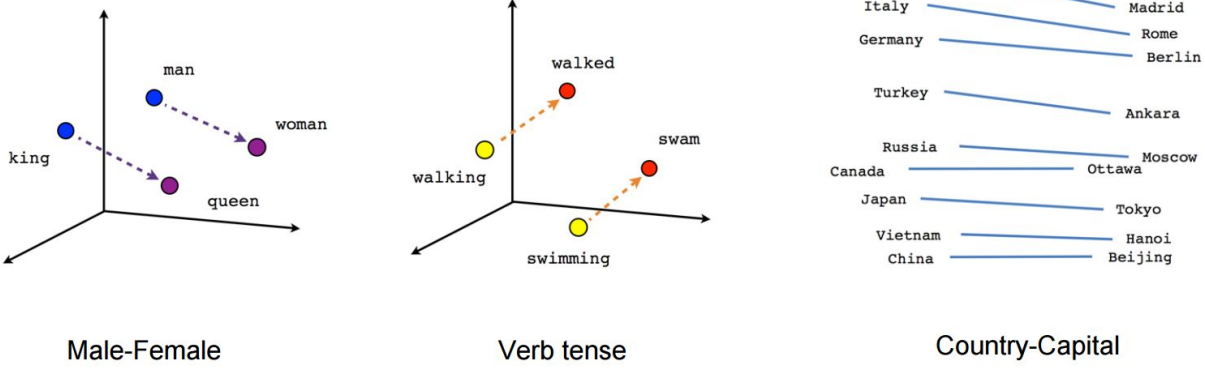
- Lets also assume similar question pairs have a token set ratio above the 75th percentile.



Over 60000 question pairs labelled wrongly according to the assumption.

WORD2VEC FEATURE EXTRACTION

There is a lot ambiguity in human language so in order for the computer to be able to understand words, the context and meaning of the words needs to be looked at. Words often have synonyms, multiple meanings and antonyms and can have different meaning with respect to the context in which they appear. Word to vectors technique comes into play as we need to represent the words in a way that makes sense to the computer. Here, words are placed in a plane of vectors. Words that normally appear in the same context appear close to each other on the vector plane.



Male-Female Verb tense Country-Capital

An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec (Analytics Vidhya)

GloVe[3] which stands for Global Vectors for Word Representation is an unsupervised learning algorithm created by some Stanford researchers, has files with vector representations for words that was used to generate a word2vec feature. The file with 6B token, 400000 vocabularies and 300d vectors was used in generating the next feature which is the word similarity feature. This feature was obtained by checking each word in a question against each word in the paired question with similar question pairs having a score closer to one and vice versa.

TF-IDF FEATURE EXTRACTION

This feature was added later to improve accuracy of the model. The tf-idf is a numerical statistic that shows how important a word is to a text in a corpus. It is very effective to retrieve important information in a text. The tf-idf value increases proportionally to the number of times a word appears in the document and is offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general. This means it is not affected by stop words in the text.

$$w_{i,j} = tf_{i,j} \times \log\left(\frac{N}{df_i}\right)$$

$tf_{i,j}$ = number of occurrences of i in j
df_i = number of documents containing i
N = total number of documents

Formulae to calculate the tf-idf of a text in a corpus

20 features were generated with the tf-idf vectorizer which was dimensionally reduced with the truncated singular value decomposition so the output can be used in the model.

BENCHMARK

Quora uses the random forest ensemble method to solve their classification problem so that was used as my benchmark model in conjunction with others. Logistic Regression, random forest and xgboost classifiers were run on the preprocessed data to get a benchmark score with the aim of improving upon it with additional features. These were the benchmark scores for the three models with the log loss scoring function.

```
Logistic Regression: -0.539474
Random Forest: -0.983306
XGBoost: -0.500558
```

METHODOLOGY & RESULTS - TRADITIONAL MODELS

LOGISTIC REGRESSION

Logistic regression, named after the function at its core called the logistic function. This results in an S-shaped or sigmoid curve that maps input values between 0 and 1.

$$f(t) = \frac{1}{1+e^{-t}}$$

Where t represents input value and e represents the exponential function.

Logistic regression uses an equation that combines input values in conjunction with weights or coefficients to output a prediction y. The output from the logistic regression function are binary values, either 0 or 1.

$$y = e^{(b0 + b1 * x)} / (1 + e^{(b0 + b1 * x)})$$

From the equation above y is the predicted output, b0 is the bias or intercept term and

b1 is the coefficient for the single input value (x). Each column in your input data has an associated b coefficient (a constant real value) that must be learned from your training data. With reference to the project's dataset, logistic regression seeks to model the probability of questions being duplicates given some pair of question features or characteristics. So logistic regression makes predictions and transforms it into a 0 or 1.

ENSEMBLE METHOD(RANDOM FOREST)

Random forest is an ensemble of decision trees (weak learners) to create a better model with the goal of reducing overfitting. Random forests are trained via bagging method. Bagging is randomly selecting a sample of the training dataset, fitting it into a model and outputting the predictions. This is sampling with replacement but with random forest, it is tree bagging. Hence, random features from the dataset are selected, fitted into a decision tree and outputting the result. The prediction with the most votes wins.

XGBOOST

Extreme Gradient Boosting (xgboost) is an implementation of gradient boosting machines. This library offer a number of useful features and in terms of computational speed and performance, it's very efficient. Boosting is an ensemble method that combines weak learners or classifiers to create a strong one. A learner is considered weak or strong with regards to how well it predicts the actual target variable. Errors are reduced during training by stacking learners on top of each other repeatedly correcting the errors of the previous learner until an accurate model is produced.

Gradient boosting uses a similar technique described above however, instead of assigning different weights to the classifiers after every iteration, this method fits the new model to new residuals of the previous prediction and then minimizes the loss when adding the latest prediction. So, in the end, you are updating your model using gradient descent and hence the name, gradient boosting[6].

MODEL CREATION WITH METADATA + WORD2VEC FEATURES

The dataset was split into a train set and a validation set of 70% and 30% respectively. Log loss was used as the scorer function with a lesser value being better. Three algorithms were chosen to find the best one that works well with the processed data. Logistic regression algorithm, the random forest ensemble method and the xgboost algorithm which is known to perform very well in Kaggle datasets.

A pipeline was created that runs each model with scaled values of the train datasets with the corresponding labels. The models were trained in 10 folds and their mean score

computed as their log loss values. Random forest performed poorly unsurprisingly because of the fact that it does not work well sparse datasets. Logistic regression algorithm did better but xgboost classifier came out on top with a log loss close to the benchmark model.

LR - Logistic Regression	RF - Random Forest	XG - XGBClassifier
ScaledLR: 0.532948		
ScaledRF: 0.978139		
ScaledXG: 0.491490		

The better model was selected and further tuned to gain a better log loss value. The xgboost classifier was tuned with the max_depth hyperparameter in a range from 3 to 9 and min_child_weight hyperparameter in a range from 1 to 5. Using the same scoring and 10 folds of the dataset these were the results:

Best: 0.467980 using {'max_depth': 8, 'min_child_weight': 1}

MODEL CREATION WITH METADATA + WORD2VEC + TF-IDF FEATURES

What if the model accuracy could be improved by adding more features. Tf-idf vectors with 20 features were added to the metadata and word2vec features and run through a pipeline with the same scoring and same number of folds. The three models where run and these were the results:

LR - Logistic Regression	RF - Random Forest	XG - XGBClassifier
ScaledLR: 0.520244 (0.002672)		
ScaledRF: 0.782523 (0.017344)		
ScaledXG: 0.480573 (0.002681)		

Clearly it can be seen there has been some improvement in all the models. Random forest still not performing well but has improved by much. Logistic Regression did not improve by much. Here again, the xgboost classifier has performs better than the rest. It has also improved a little. The best model was further tuned to gain better accuracy with the same configuration as talked about above. Here are the results;

Best: 0.447165 using {'max_depth': 8, 'min_child_weight': 1}

There was a lot of improvement, clearly reinforcing the importance of hyperparameter tuning. A lot can be done to improve the model. Maybe increasing the max_depth could give better results or adding more features to the processed data.

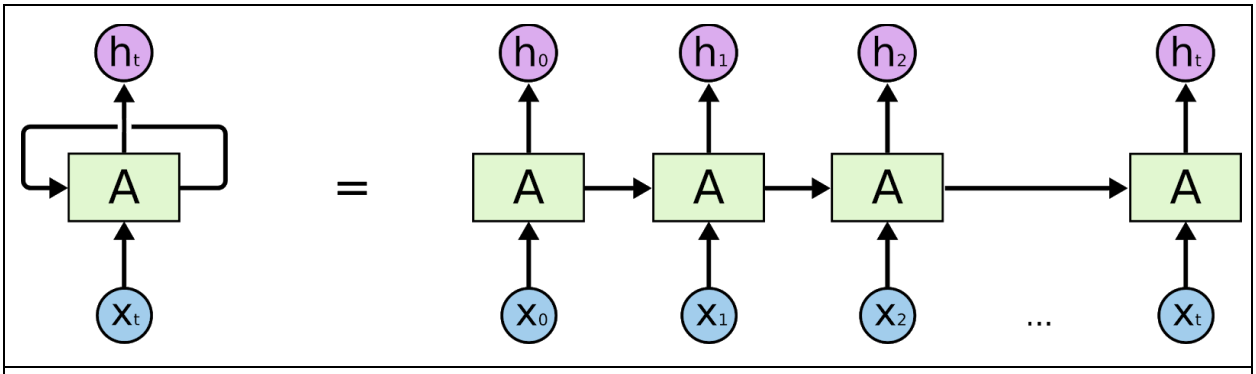
COMPLICATIONS WITH TRADITIONAL MODELS

The model found it difficult to work with values that reached infinity. I tried to add another word2vec feature called word mover's distance. So word mover's distance basically gives the distance between two documents even when both documents have no common words. The documents have no words in common, but by matching the relevant words, word mover's distance is able to accurately measure the (dis)similarity between the two sentences. The method also uses the bag-of-words representation of the documents. It was okay to convert NaN or infinity values to zero but I felt that would affect the model so I decided not to add the word mover's distance feature.

RECURRENT NEURAL NETWORK (LONG SHORT-TERM MEMORY (LSTM))

Some benefits of using a recurrent neural network is that it's very good for predicting sequential data. The objective of this study is to solve a problem which involves sequential data, which is a good model for the problem. There are a number of recurrent neural networks but the one used in this project is the LSTM which solves the recurrent neural network problem of the vanishing gradient.

A recurrent neural network takes in input and runs a loop by passing the current loops information to the next loop. Recurrent neural networks can learn to use past information that's why it fits so well for sequential data. So assuming you have a sentence and want to predict the next word, due to the way it has been structured, its able to connect previous information to the next task. But RNNs find it difficult to learn the connections between information with a wide gap. For instance, to predict the last word in a long sentence without knowing the immediate word before the last word will be difficult for the RNN.



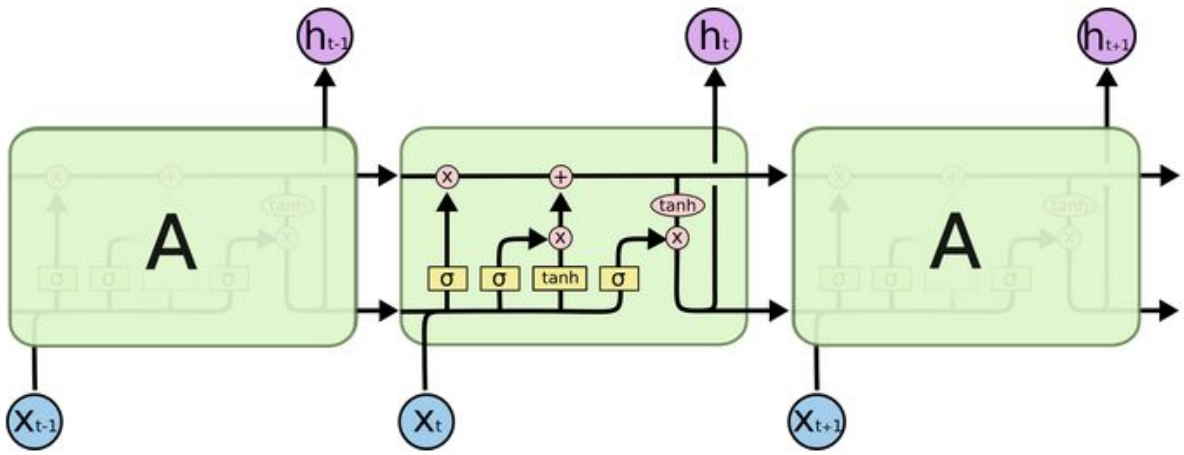
An unrolled recurrent neural network [5]

This is where LSTMs become very useful and more advanced from of RNNs. Vanilla RNNs are very capable of learning short-term dependencies but LSTMs are capable of learning long-term dependencies very well, hence why its called long short-term

memory. Instead of one neural network layer, the LSTM uses four neural network layers in different ways. An LSTM layer has a cell state and hidden state. The cell state which is a key part of an LSTM is regulated by structures called gates. These gates act as doors to either let information through or not. Using the sigmoid activation function these gates are able to make decisions for values between 0 and 1 where 0 values are not allowed through the gate.

An LSTM layer is made up of three gates. The first is the forget gate layer that looks at the previous hidden state and current input and decides how much information should be allowed through to the previous cell state. The next gate called the input gate layer decides which values to be updated to the previous cell state in conjunction with a tanh layer. Note, that the forget gate and input gate only make decisions but not actually executing these decisions. So the next stage is to execute these decisions made by the gates to update the previous cell state to a new cell state. So to create the new cell state, the values allowed through the forget gate will be multiplied to the previous cell state and the new update values from the input gate in conjunction with the tanh layer will update the previous cell state and this create a new cell state.

Next is to output a new hidden state. The new cell state is put through a tanh which creates values between -1 and 1. This is multiplied to the output of the sigmoid gate based the decision made. This create the new hidden state. So at the end of every loop, a new cell state and hidden state is created which are used in the next loop.



The repeating module in an LSTM contains four interacting layers.

DATA PREPARATION FOR LSTM

Text data from the dataset were tokenized with the text tokenization utility class from keras which vectorized the text corpus by giving each word in the corpus a unique index or integer. Next, each question in a column was converted into a sequence with a common padding of 40. Padding is done so each input to be fed into the neural network has equal length. It is assumed 40 is a good size and none of the tokenized questions has a length more than 40 and because the text corpus has different lengths, padding becomes very useful here.

Since, an embedding layer will be used in the neural network, an extract of the word embeddings from the GloVe text data was used. 400000 word vectors were extracted and used to create an embedding matrix of shape (x, y), where x is the total number of unique words tokenized from the text corpus and y is 300.

LSTM MODEL

Input tensors with shape of (40,) because of the padding length of the text data was fitted into an embedding layer with an input dimension of length of word_index (i.e. the total number of unique words tokenized from the text corpus) + 1 including values from 0 with an output dimension of 300 because the GloVe text data has a dimension of 300, using weights from the embedding matrix created. The embedding layer has a parameter called, trainable which was set to false because I did not want to update the weights on the learned words. The embedding layer was fitted into a time distributed dense layer with units of 300 and a relu activation function. The time distributed dense layer was seeded into a lambda layer to return the sum of the inputs horizontally across columns.

This structure was used for both tokenized text corpus from from the dataset of columns question1 and question2.

Another structure was created with a similar input and embedding layer but this time the embedding layer was not modeled with the embedding matrix. A dropout layer was added to the input to drop out 30% of the input to prevent overfitting during training. This was seeded into an LSTM layer with 300 units, a dropout of 30%, a recurrent dropout of 30% and the return_sequence parameter set to true so the full sequence length will be returned. Lastly, the LSTM layer was passed through a lambda layer similar as the one above.

This structure was used for both tokenized text corpus from from the dataset of columns question1 and question2. So in total, 2 pairs of models were created. These 4 models were concatenated to return a single tensor from the concatenation of all inputs. The concatenation output were normalized with a BatchNormalization layer. The model is then inputted into a dense layer of 300 units, a parametric rectified linear unit (PReLU) layer which is an advanced activation layer to help improve the accuracy of the model.

This is further run through a dropout layer of 30% and a batchnormalization layer applied again. This same structure goes on for two more times and seeded into an output layer.

The output layer is a dense layer of 1 unit and run through an a sigmoid activation layer which squash the model predictions to either 0 or 1. This was the model and implementation of an LSTM recurrent neural network.

LSTM MODEL RESULTS

A binary crossentropy loss similar to the log loss, the adam optimizer and an accuracy metrics was used to compile the model. The input data and target data were fit with a batch size of 384 with 15 epochs with a validation sample of 1%.

These were the results;

```
363648/363861 [============================>.] - ETA: 0s - loss: 0.1688 - acc: 0.9289
Epoch 00014: val_acc improved from 0.81889 to 0.82050, saving model to weight.h5
363861/363861 [==============================] - 334s 919us/step - loss: 0.1688 - acc: 0.9289 - val_loss: 0.5807 - val_acc: 0.8205
```

Here a loss of 0.1688 and a validation loss of 0.5807. There may be some overfitting here. This model can be improved by adding a convolution layer and adjusting the dropout rates.

MODEL VALIDATION AND EVALUATION

XGBoost won the race. This project used a combination of three traditional machine learning models and a recurrent neural network to solve the problem of identifying duplicate questions. Among all the techniques applied, the xgboost classifier had better results. You would expect the LSTM to be victorious but due to limitation in processing speed available, I implemented a very simple one which I'm sure with better tuning will outperform the xgboost classifier. The XGBoost classifier showed no signs of overfitting as the model was validated using different KFolds from 5 - 20%. KFolds CV score of 10 was better than the other folds. The model is very robust because of its use of boosting trees which applies scores in the leaves of its nodes.

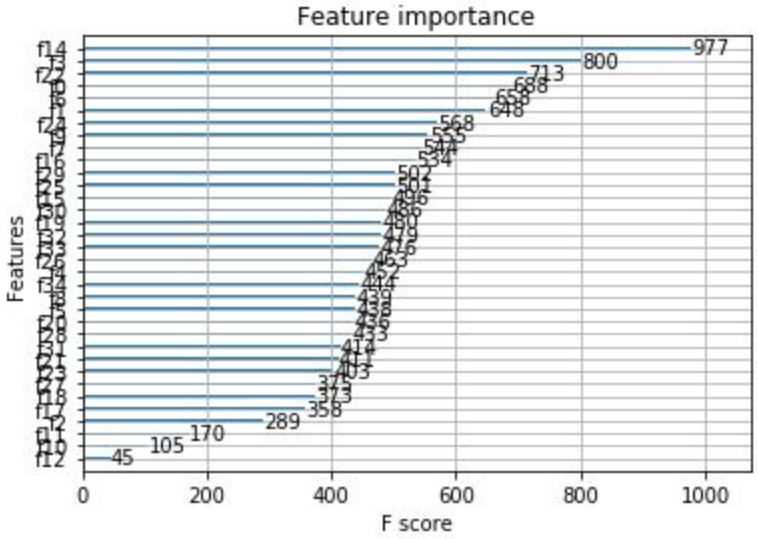
JUSTIFICATION

The final results are very significant and better than the benchmark model with regards to the traditional machine learning models. Logistic regression was second best among the three model which showed some potential to be better. Random forest was the worst but performed better in the benchmark model without the addition of word2vec

feature and tf-idf features. Xgboost was the best and compared to the benchmark model it had a log loss of 0.45 while the benchmark had 0.5. This definitely good enough to solve the problem at hand. Of course, no model is perfect but I'm confident it can accurately predict 75% of duplicate questions. The LSTM did not do so well compared to the xgboost classifier, scoring a validation loss of 0.5.

CONCLUSION

FREE FORM VISUALIZATION



Using the feature importance method of the xgboost classifier, it can be seen, the features that affected the accuracy of the model during the training and validation. It was good to observe word2vec feature named word_similarity being the most important feature, followed by common_words and tf-idf features. Collectively these features are very important for the classification problem.

REFLECTION

Machine learning models are very good with numerical data but it's not all data that appear to be in numerical form. Seeing this problem and the dataset, it was quite a challenge as to how to get a machine learning model to understand this dataset. The interesting part was finding the right features to extract from the text data. It was easy to formulate the text preprocessing plan as it was obvious what two questions will have in common if they happen to be duplicates. Two questions will need to have the same meaning, context, and probably similar words to be duplicates. Finding these characteristics from the text data was the fun part. 14 features were extracted and the fuzzywuzzy library proved to be very useful as it was able to extract intrinsic meaning or information out of the text data and interpret them into numerical values.

In this project, I have shown how to work on the quora duplicate questions problem with two approaches. One approach was to use machine learning algorithms like logistic regression, an ensemble method and xgboost. To be able able to use these algorithms, a preprocessed dataset was used and the xgboost classifier came out as the better model. The improvement with this approach is either to increase the max_depth parameter for the xgboost to get a better accuracy or to extract more important features from the dataset. The second approach was to use a recurrent neural network, an LSTM to be specific to solve the problem. The text data had to be tokenized and padded to be of same length. This was then run into the LSTM architecture and the results were not as good as the first approach. To improve the LSTM model, more layers will have to be added, maybe a convolutional model could be added and tweaking the dropout rates could also improve the model and prevent overfitting.

REFERENCES

[1] https://www.kaggle.com/c/quora-question-pairs

[2] http://chairnerd.seatgeek.com/fuzzywuzzy-fuzzy-string-matching-in-python/

[3] https://nlp.stanford.edu/projects/glove/

[4] https://github.com/bradleypallen/keras-quora-question-pairs/blob/master/

[5] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[6] https://www.kdnuggets.com/2017/10/xgboost-top-machine-learning-method-kaggle-explained.html

http://www.erogol.com/duplicate-question-detection-deep-learning/

https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/

https://github.com/abhishekkrthakur/is_that_a_duplicate_quora_question