Djona Fegnem

Capstone Project

Machine Learning Engineer Nanodegree

November 27, 2017

Project: Identify Pairs of Sentences with the Same Intent in Online User Forums.

Definition

Project Overview

Two questions with different vocabulary and syntactic structure asked on an online forum can have the same answer. Detecting semantic similarity between them can be a challenging task for traditional machine learning algorithms, which rely on a subject-mater expert to carefully extract useful features from data prior training. The advance of neural network[1] technology has provided methods to automatically uncover words meaning by mapping them into low-dimensional vectors. This ability pairs with a recent computer vision technology called Convolutional Neural Network (CNN) has proven to be successful in detecting semantic similarity between pairs of sentences[2].

In this project, we will apply the same techniques on Quora dataset[3]. We will use as our baseline model an algorithm called eXtreme Gradient Boosting[4] (XGBoost) , trained on the same data after manually extracting useful features. Our experiment shows that the former method outperforms the latter.

[1] https://arxiv.org/abs/1301.3781

[2] https://www.aclweb.org/anthology/K15-1013

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

[4] https://xgboost.readthedocs.io/en/latest/

Problem Statement

The goal is to create a model capable of identifying pairs of questions with the same answers. The tasks involved are the following:

1. Download and preprocess the Quora dataset.
2. Train a CNN classifier with the data.

The final model can be used to improve the user experience in online forum websites by quickly providing answers to previously asked and answered questions.

Metrics

We have imbalanced class distribution in our dataset. The proportion of negative instances is about twice the proportion of positive instances. As a result, we will use the f1 score as our evaluation metric.

$$f1_{score} = 2 * \frac{precision * recall}{precision + recall}$$

$$precision = \frac{true\_positive}{true\_positive + false\_positive}$$

$$recall = \frac{true\_positve}{true\_positive + false\_negative}.$$

- true_positive: the number of similar pairs of sentences our model has successfully identified.
- false_positive: the number of similar pairs our model has identified as non-similar.
- false_negative: the number of non-similar pairs our model has identified as similar.
- The precision measures the proportion of positive instances correctly classified out of all the one returned by our model during evaluation.

- The recall or true positive rate measures the proportion of positive instances correctly classified out of all the positive instances in our dataset during our evaluation.

The f1 score is a single metric that captures both precision and recall. It gives the same weight to the two metrics.

Analysis

Data Exploration

	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

Figure 1: sample of data.

Our data has 404,290 entries and 6 columns. The columns hold the following data:

1. Id: a unique integer value that represents the id of each entry.
2. qid1: a unique integer value that represents the id of the question 1 text.
3. qid2: a unique integer value that represents the id of the question 2 text.
4. question1: a text that represents the first question.
5. question2: a text that represents the second question.
6. is_duplicate: a value of 1 or 0 that indicates either similar or non-similar.

We have done the followings during the pre-processing step:

1. Checked and removed missing entries[5].
2. Checked for duplicate questions and duplicate question pairs[6].
3. Removed all non-alphanumeric characters in sentences[7].
4. Removed the unnecessary columns: id, qid1, qid2.

At the end of our data pre-processing step, our final dataset has 404,281 entries and 3 columns, namely question1, question2, and is_duplicate.

Exploratory Visualization

Figure 2 and Figure 3 show some useful visualization. *Figure 3* shows that we have more non-duplicate entries than we have duplicates. The non-duplicates account for roughly 63%, and the duplicates account for roughly 37% of our entire data. Furthermore, *Figure 2* suggests that sentences length – number of token in each sentence - are right skewed with values centered between 0 and 30 for question 1 and between 0 and 50 for question 2.

[5] section 'check and remove missing entries' in data_preprocessing.ipynb.

[6] section 'Check for duplicates' in data_preprocessing.ipynb

[7] section 'Text Analysis' in data_preprocessing.ipynb.

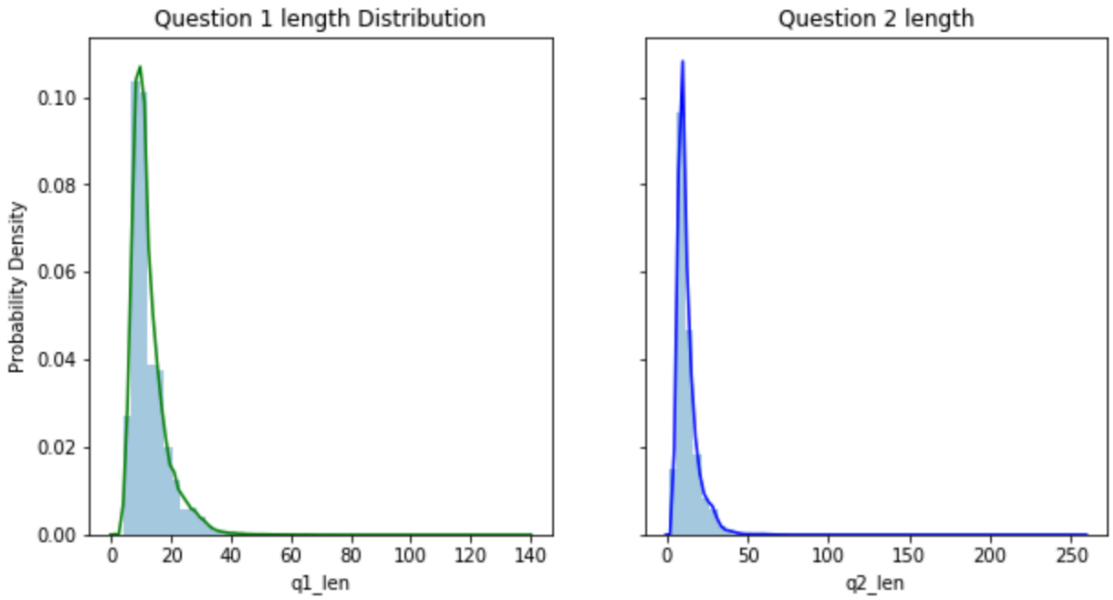

Figure 2 Density plot of sentences length

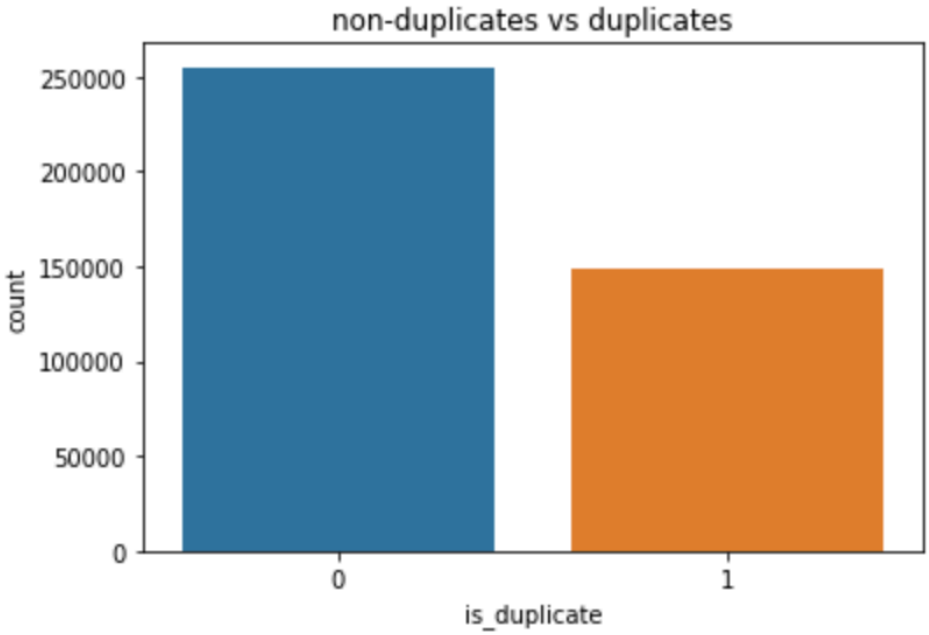

Figure 3 Number of duplicate pairs (1) and non-duplicate pairs (0)

Algorithms and Techniques

Our classifier uses two principal algorithms: skip-gram neural network, and convolutional neural network.

- **The skip-gram model**

  The skip-gram neural network is a prediction based embedding. It is a shallow neural network with three layers: an input layer, a hidden layer, and an output layer. It aims to predict the probability distribution between the center words and context words in terms of words vector. Its objective function is to maximize the probability of any context word given the current center word. For each estimation step, the algorithms take one word (one-hot encoded representation) as the center word and predict its context words up to some window size. It defines the probability distribution of a word appearing in the given center word context and chooses the vector representation that maximizes that probability distribution. The advantage of this method is that it captures words semantic. The drawback is that the quality of the word representation is dependent on the corpus size.

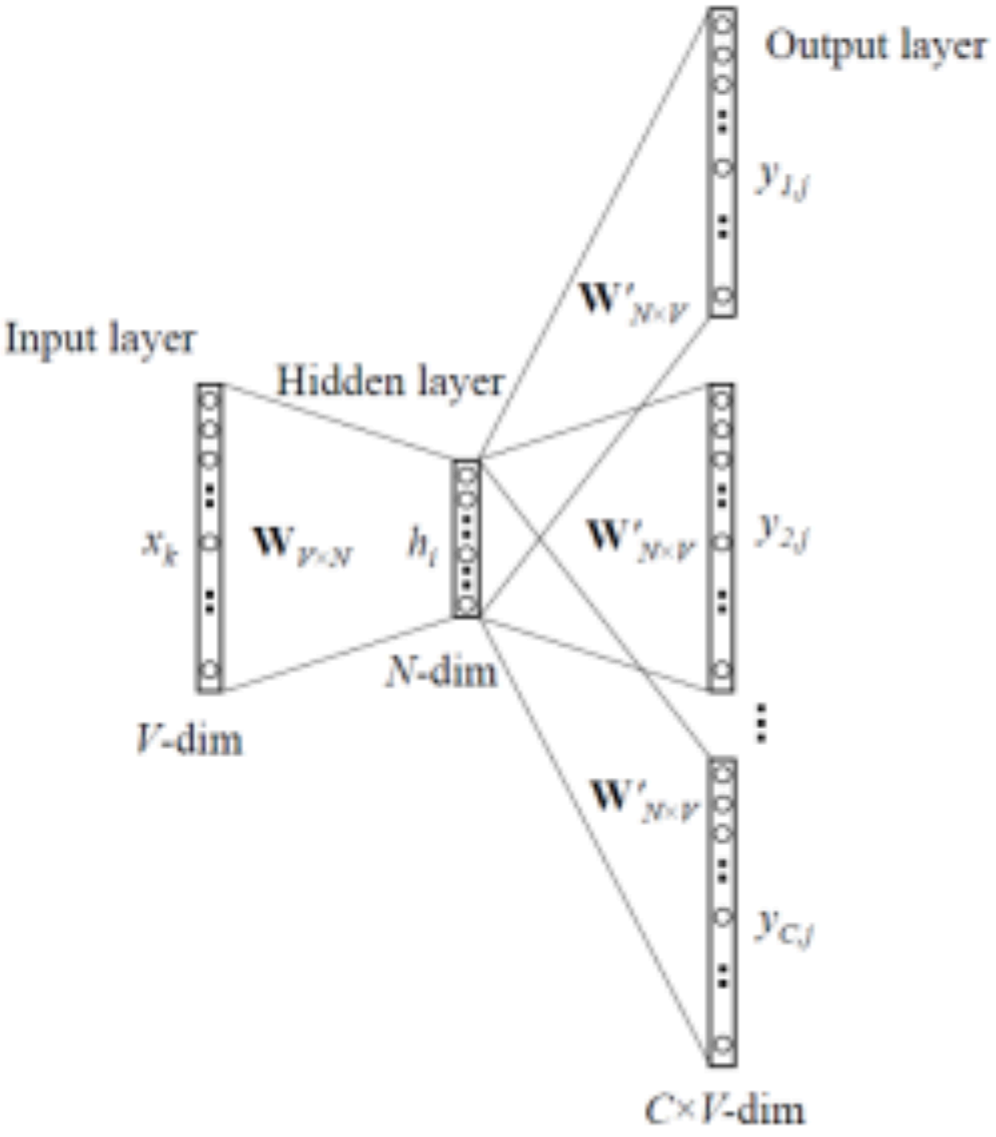

Figure 4 Skip-gram model architecture

- **Convolutional Neural Network (CNN)**

  Originally invented for computer vision, CNN models have subsequently been shown to be effective for natural language processing task[8]. It utilizes

[8] Convolutional Neural Networks for Sentence Classification, Yoon Kim, New York University

filters to extract local and spatial features from the input data through the convolution operation.

The training involves the following steps:

1. A tokenized pair of text is fed into an in-domain skip-gram neural network model. The model transforms each word into a low-dimensional vector representation.
2. The convolutional layer is used to construct two vector representations of each sentence.
3. The CNN computes the cosine similarity score between the two vectors. The cosine similarity score is a real value between 0 and 1. Scores close to 0 indicate that the pair is non-similar and scores close to 1 indicates that the pair is similar. We will use a threshold value during testing to discriminate between the two classes.

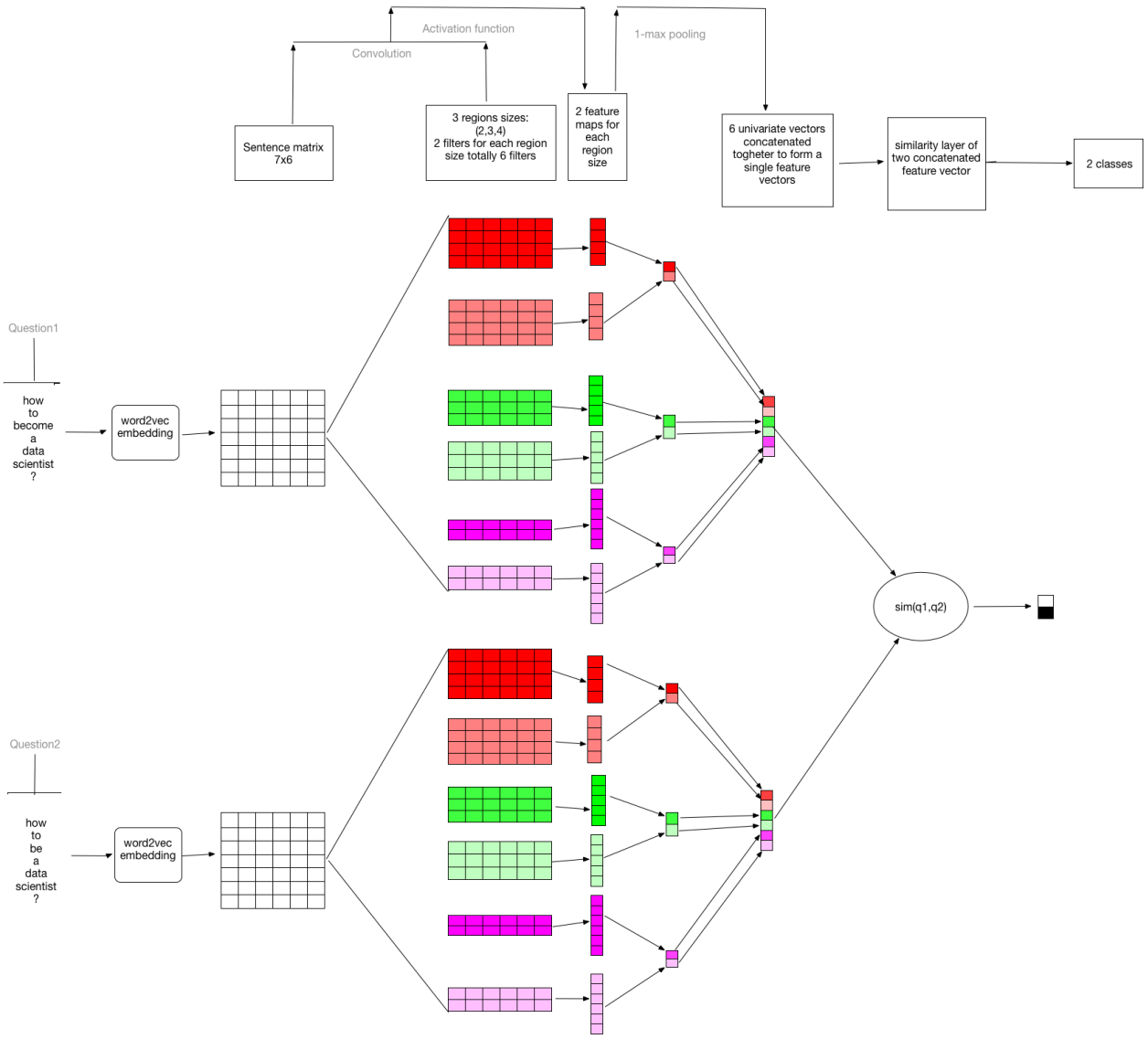

Figure 5 Network architttecture

Benchmark

As a baseline model, we used the XGBoost algorithm. We started by performing some features engineering on the dataset. The goal of the features engineering step is to extract useful features from our text data.

1. **XGBoost**

   eXtreme Gradient Boosting or XGBoost is an implementation of gradient boosting decision tree algorithm with the aim to improve performance and push the limit of computations resources for boosted tree algorithms[9]. Gradient boosting is an approach where new models that predict the residuals or errors of prior models are created and then added together to make the final prediction. It is called gradient boosting because it uses a gradient descent algorithm to minimize the loss when adding new models. XGBoost is extensively used by machine learning practitioners to create state of art data science solutions[10].

2. **Term-document matrix with TF-IDF**

   The term-document matrix represents words or terms based on documents they contain. Values in the matrix can be either the word count, the term frequency inverse document frequency(TF-IDF) or any other frequency

[9] https://www.quora.com/What-is-the-difference-between-the-R-gbm-gradient-boosting-machine-and-xgboost-extreme-gradient-boosting

[10] https://github.com/dmlc/xgboost/tree/master/demo#machine-learning-challenge-winning-solutions

based value. The TF-IDF captures both the frequency of terms within the document and its rarity in the corpus.

$$w_{t,d} = (1 + \log_{10} tf_{t,d}) * \log_{10} \frac{N}{df_t}$$

Where:

$w_{t,d} =$ the tf − idf weight of the term t in the document d.

$tf_{t,d} =$ the frequency of $the\ term\ t\ in$ the document d.

$df_t = the\ document\ frequency\ of\ the\ term\ t.$

$N = the\ total\ number\ of\ documents\ in\ the\ corpus$

We will use term-document matrix with TF-IDF weights during the modeling of our benchmark. Each document or question will consist of a TF-IDF vector in R^|V|, with |V| being our corpus' vocabulary size.
R means our vector contains real value.
TF stands for term frequency, it is the number of time a particular term appears in a document.
IDF stands for inverse document frequency, it is the inverse of the number of documents in the corpus that contains a particular term.

3. **Data preprocessing**[11]

   We have preprocessed the data by removing stop-words and punctuations.

4. **Feature engineering**[12]

   The following features have been extracted:

   - The cosine similarity score: this similarity score is calculated by using a vector representation of each text. This representation is calculated

[11] Section 'Remove punctuations and stopwords' in data_preprocessing.ipynb

[12] Section 'Feature engineering' in data_preprocessing.ipynb

using a latent semantic structure model[13] computed by using singular vector decomposition applied on the term-document matrix with term-frequency inverse document frequency.

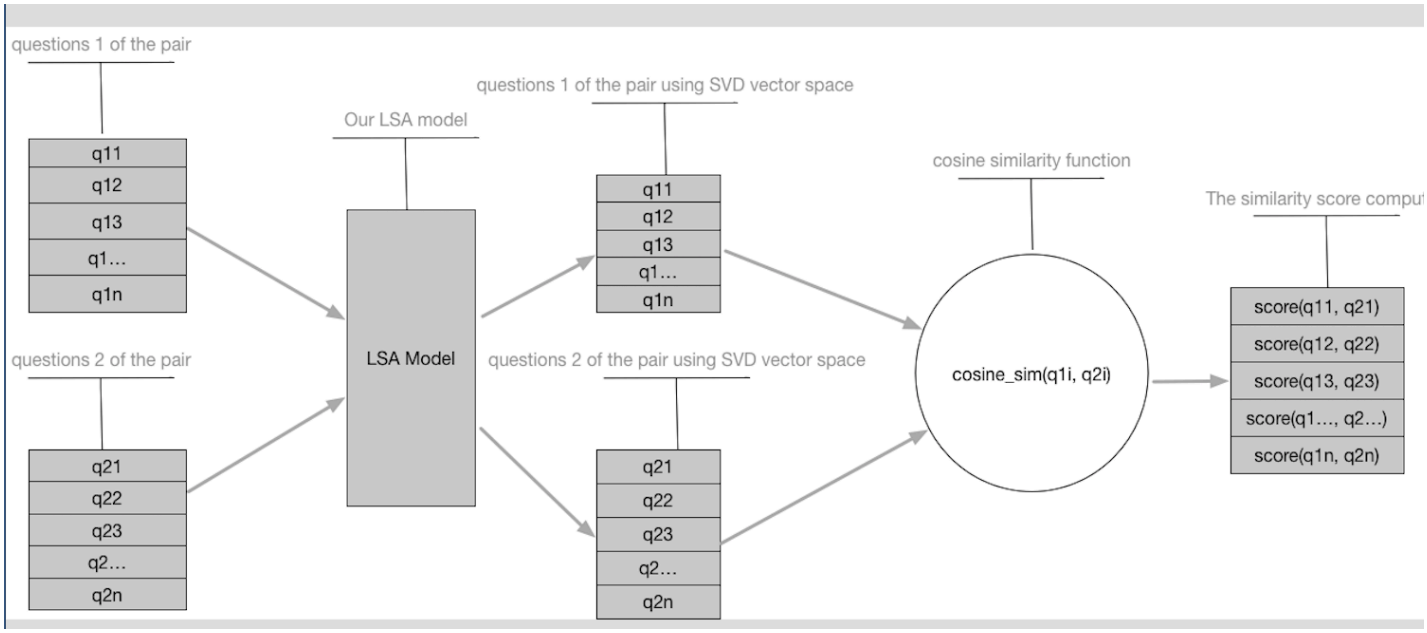

Figure 6 cosine similarity computation process

- The number of tokens in each sentence.
- The number of unique tokens in each sentence.
- The number of shared tokens between the pair.

5. Visualizations

Figure 8 shows the distribution of our features. The questions length and questions vocabulary length are all right skewed. The 'share_voc_len' distribution is almost normal, and the similarity score distribution is binomial. *Figure 9* shows a strong positive relationship between the similarity score and the shared vocabulary length. *Figure 10* shows that the similarity score and the shared vocabulary length are the most important features in our dataset.

[13] http://lsa.colorado.edu/papers/JASIS.lsi.90.pdf

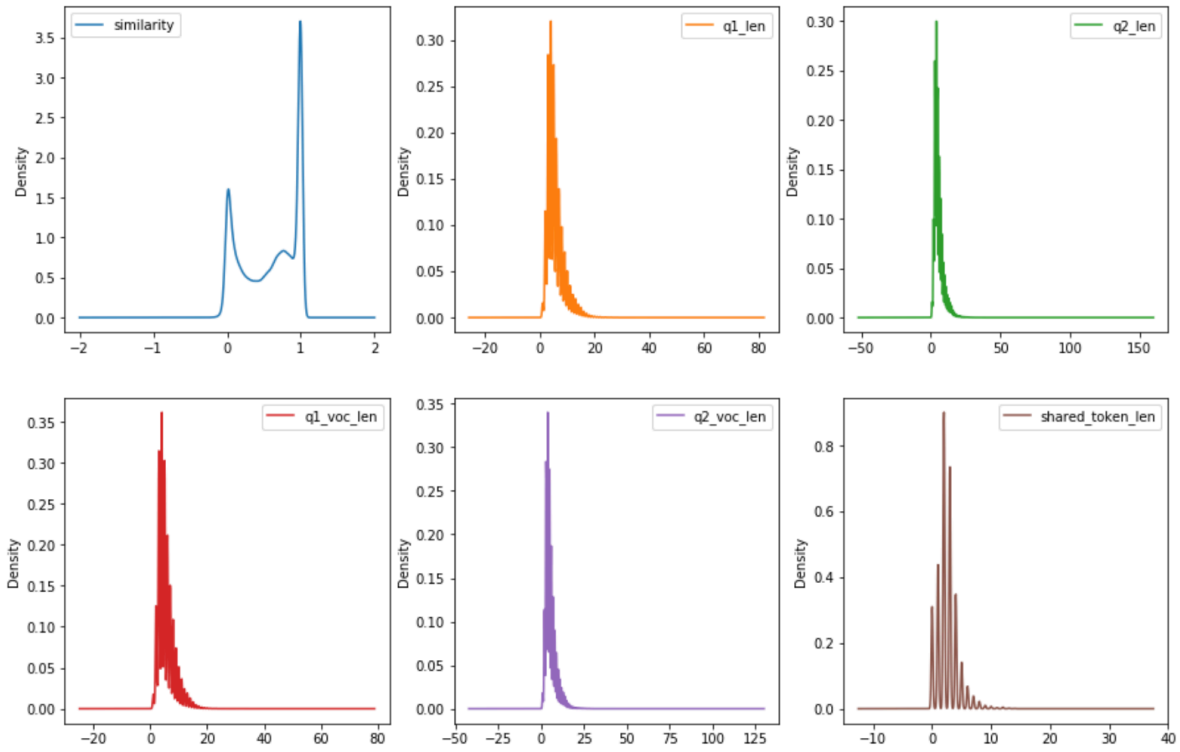

Figure 7 Features distribution

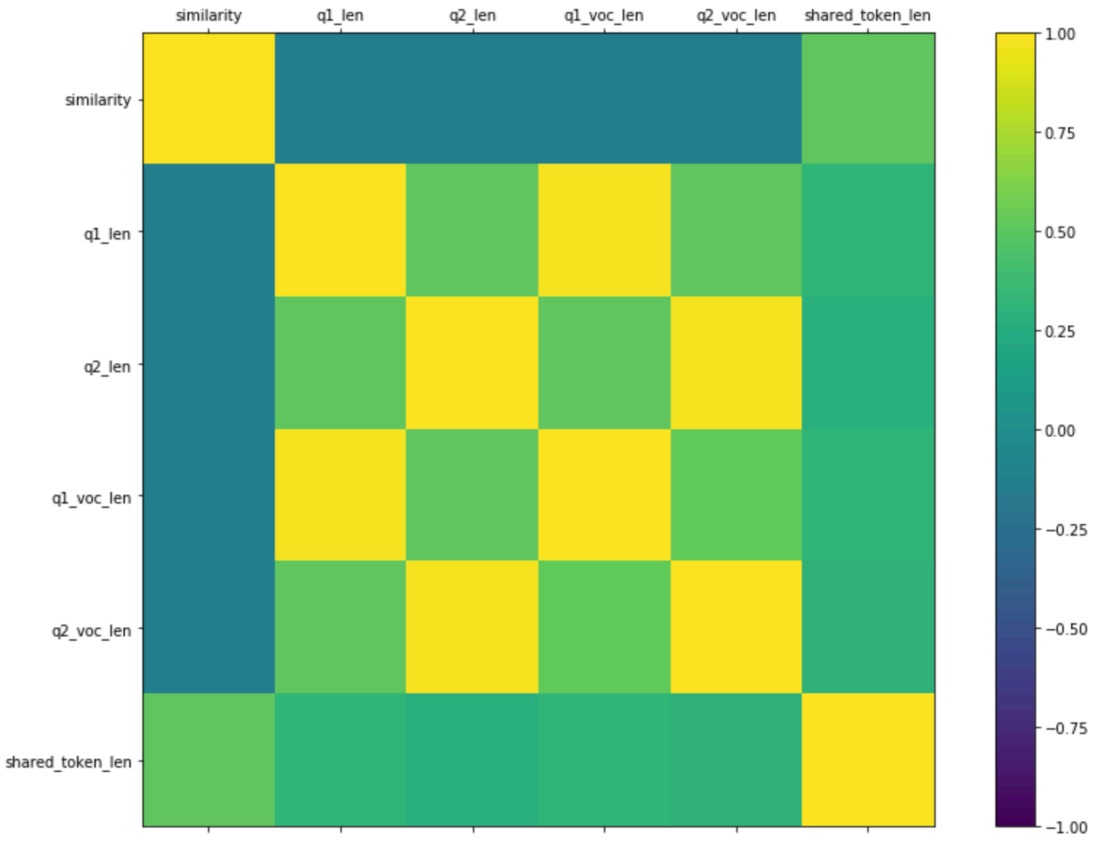

Figure 8 Correlation matrix

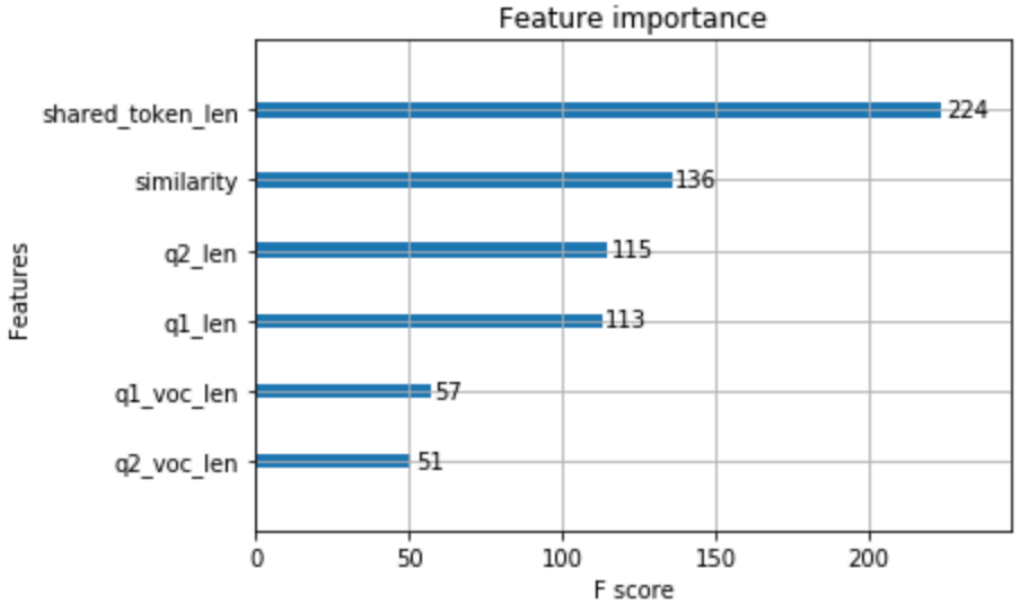

Figure 9 Feature importance

6. **Split data**[14]

   After the feature engineering step, the dataset has been split into training[15] (90%, 353,625 instances) and test[16] sets roughly (10%, 40403 instances).

7. **Training and Cross-validation**

   The benchmark model has been trained with the preprocessed dataset and the following parameters has been tuned:

   - n_estimators: number of boosted trees to fit.
   - learning_rate: learning rate.
   - max_depth: maximum tree depth for base learners.

   We used as objective function the logistic loss. We used the best hyper-parameters to fit our final model.

[14] Section 'Split data for xgboost' in data_preprocessing.ipynb

[15] quora_xgb_train.pickle in data folder of the project

[16] quora_xgb_test.pickle in data folder of the project

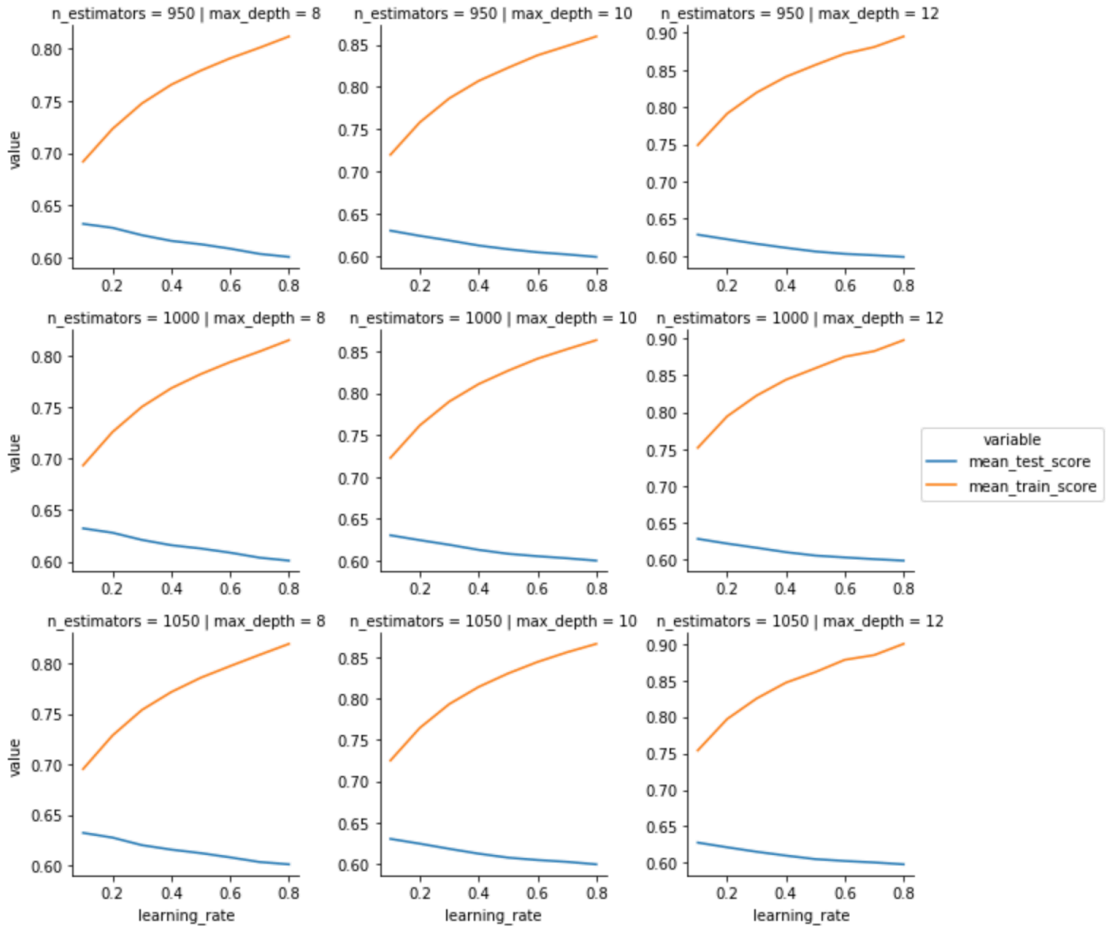

Figure 10 Hyperparameter cross-validation

8. Evaluation[17]

We have evaluated our final model on the test data. The final fbeta score we have obtained was 0.645.

[17] section 'final xgboost model' in data_preprocessing.ipynb

Methodology

Data preprocessing

To make our sentences the same length, we have padded[18] sentences less than 40 tokens with empty string, and we have only taken the first 40 tokens for sentences greater than 40 tokens. Furthermore, to make our data suitable for CNN, we have created the vector representation of each word in sentences. More specifically, we have performed the followings on our data[19]:

1. Built an in-domain skip-gram neural network model[20] using our entire corpus text. We used Gensim[21] - a natural language processing tool- to build our skip-gram neural network model.
2. Used the model built earlier to get the words vector representation.

At the end of this preprocessing step, each sentence is replaced by a Numpy ndarray of real number with a shape equal of (sentence length, embedding length). In other for each class in our data to have the same number of instances[22], we have taken all positive instances, and we have randomly selected the same number of instances within the negative class. The final dataset have 298,526 instances and 3 columns. To evaluate our model on unseen data, we have further divided our

[18] Section 'Pad sentences' in data_preprocessing.ipynb

[19] Section 'word representations' in data_preprocessing.ipynb

[20] The code is in w2vec_model.py in the models folder.

[21] https://radimrehurek.com/gensim/

[22] Section 'split CNN data' in data_preprocessing.ipynb

dataset into a training set[23] (90%, 268,673 instances) and a test set[24] (10%, 29,853 instances).

Convolutional Neural Network Definition

Once we have computed the vector representations of each word, our data is ready for CNN. The model architecture implementation[25] is described in as follow:

1. Input Layer: this layer has the pre-processed data.
2. 2D Convolution Layer: the goal of this layer is to capture spatial relationship among word using filters. We have applied different filters size. We have applied filter of size [2, 3, 4, 5, 6, 7, 8, 9, 10] to capture spatial relationship among - 2, 3, 4, 5, 6, 7, 8, 9, 10 - words that follow each other.
3. 2D max pooling layer: this layer reduces the size of the previous output by performing the max pooling 2D operation on each previous output filter. The max pooling filter has size (sentence_len – filter_size + 1, 1).
4. Merge Layer – Concat operation: this layer concatenate all the output filters from the previous layer for each sentence.
5. Flatten Layer: the output of the previous layer is then further flattened into a big vector representing the vector representation of each sentence.
6. Merge Layer – Cosine similarity operation: the vector of each sentence is then combined in this layer using a normalized dot product operation. The normalized dot product measures the proximity of two vectors. The output of this layer is a real number between 0 and 1.

[23] quora_cnn_train.pickle in data folder of the project

[24] quora_cnn_test.pickle in data folder of the project

[25] cnn_model.py

Training Procedure

Our network is trained by minimizing the mean squared error over the training set. Given a question pair *(q1, q2)*, the network with parameter set *W* computes a similarity score *s(q1, q2)*. Let *y(q1, q2)* be the correct label of the pair, where its possible values are 1(equivalent questions) or 0 (not equivalent questions). We used Adam[26] optimization algorithm with mini-batch to minimize the mean squared error with respect to *W* over the mini-batch. We used a batch size of 256 and a number of epochs of 100. The epoch is the number of time all the training set will be used to update the weights. At each epoch, our entire dataset is divided into mini-batches of equal size (256). At each pass, one mini-batch is randomly selected. The forward pass is computed using the mean squared error function loss. Then the backward pass computes the gradient using the back propagation algorithm. Finally, the Adam optimization algorithm is used to update the weights. The network has been trained with Keras[27] and Tensorflow[28] as the backend.

Implementation

The implementation involves the following:

1. Define the network parameter: filter size, stride, epochs, batch size, etc.
2. Define the metrics: fbeta.
3. Load the preprocessed data.
4. Split it into training and validation: we used 90/10.
5. Reshape the data such that it is suitable for CNN.
6. Define the model using 'CnnModel' class

[26] http://cs231n.github.io/neural-networks-3/

[27] https://keras.io/

[28] https://www.tensorflow.org/

7. Compile the model with loss (mean squared error), optimization algorithm (adam), and metric (fbeta).
8. Train the model.
9. Once the model is trained, load it and use different metrics using 'make_fbeta' function to select the best threshold.

The most challenging part of the implementation was how to implement the 'CnnModel' class, and the 'make_fbeta' function in a modular and reusable manner.

Refinement

During training, we tried multiple filters size. We started with 32 filters of heights [2, 3, 4, 5]. Our evaluation score was around 0.7. By increasing the number of filters (512) and the number of filters height [2, 3, 4, 5, 6, 7, 8, 9, 10] we were able to improve our metric to 0.81. Furthermore, we have also applied the following: regularization technique[29]:

- Dropout regularization rate, with value of 0.3.
- Max-norm with value of 4.

[29] http://jmlr.org/papers/v15/srivastava14a.html

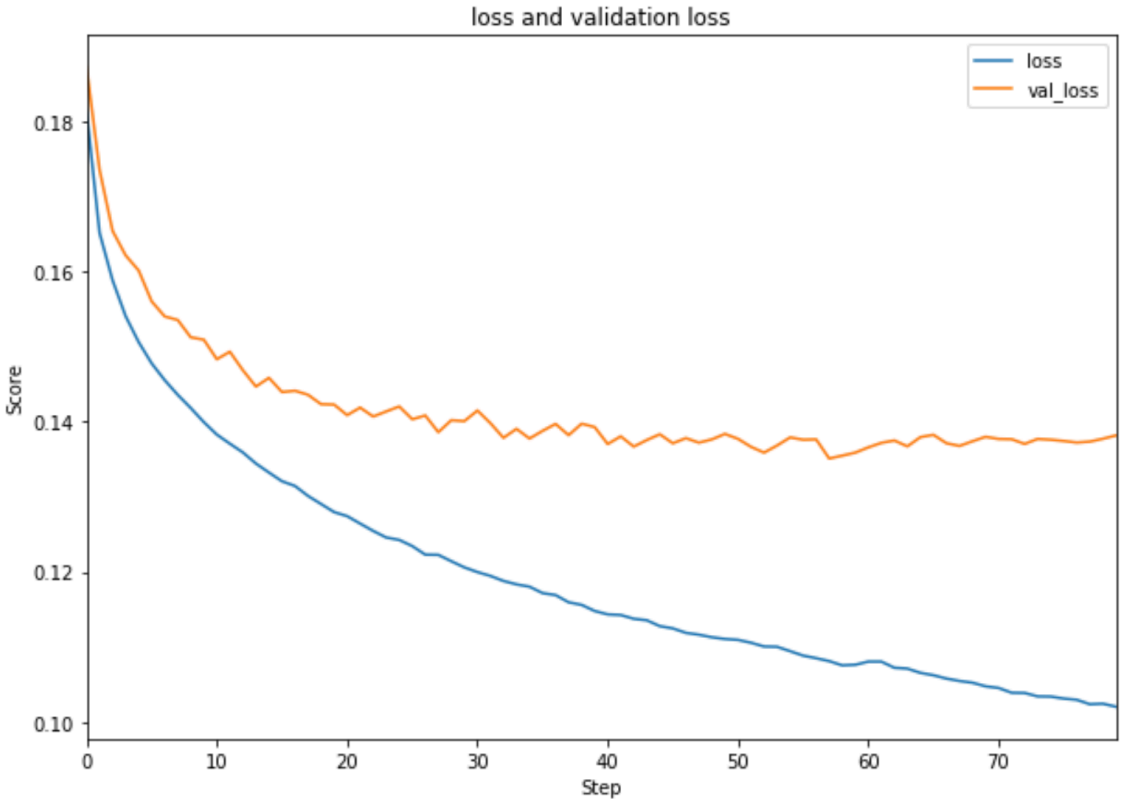

Figure 11 validation loss vs training loss

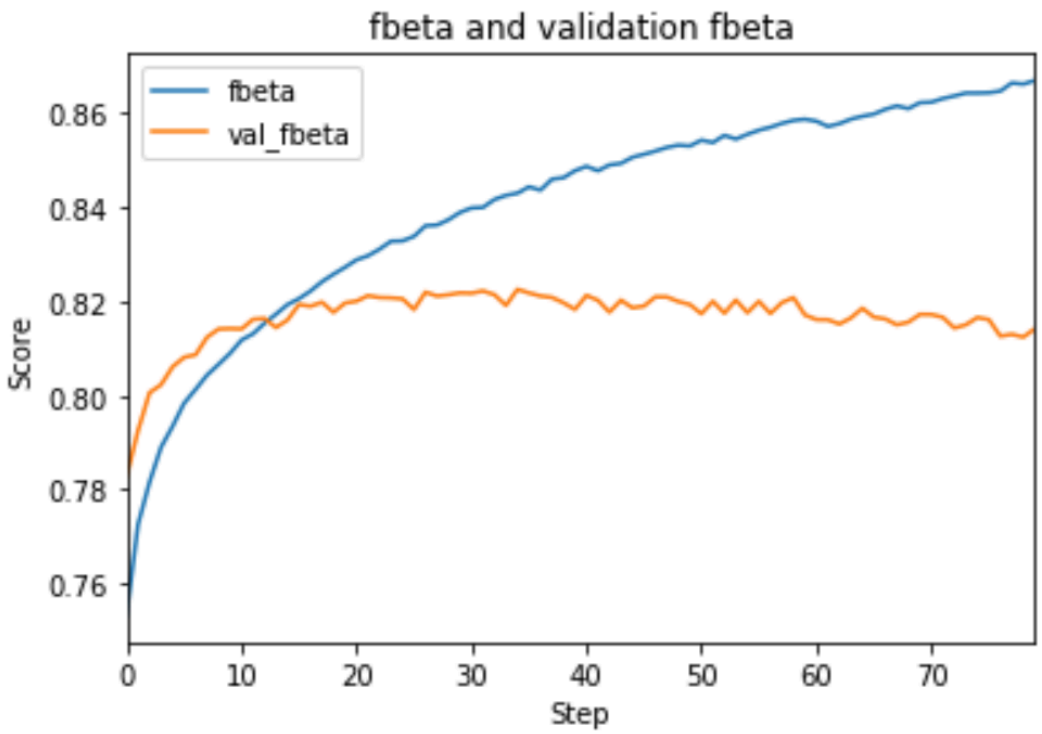

Figure 12 fbeta score

Results

Model Evaluation

Our model has been trained with a fixed threshold of k= 0.5 (threshold_shift = 0). A threshold of 0.5 means that predictions less than the threshold belong to the negative class, and predictions greater or equal than the threshold belong to the positive class. To choose the best threshold, we have evaluated our model on test

data with different threshold value. The final threshold 0.7 (threshold_shift= -0.2) is the one that maximize the overall fbeta score of 0.836.

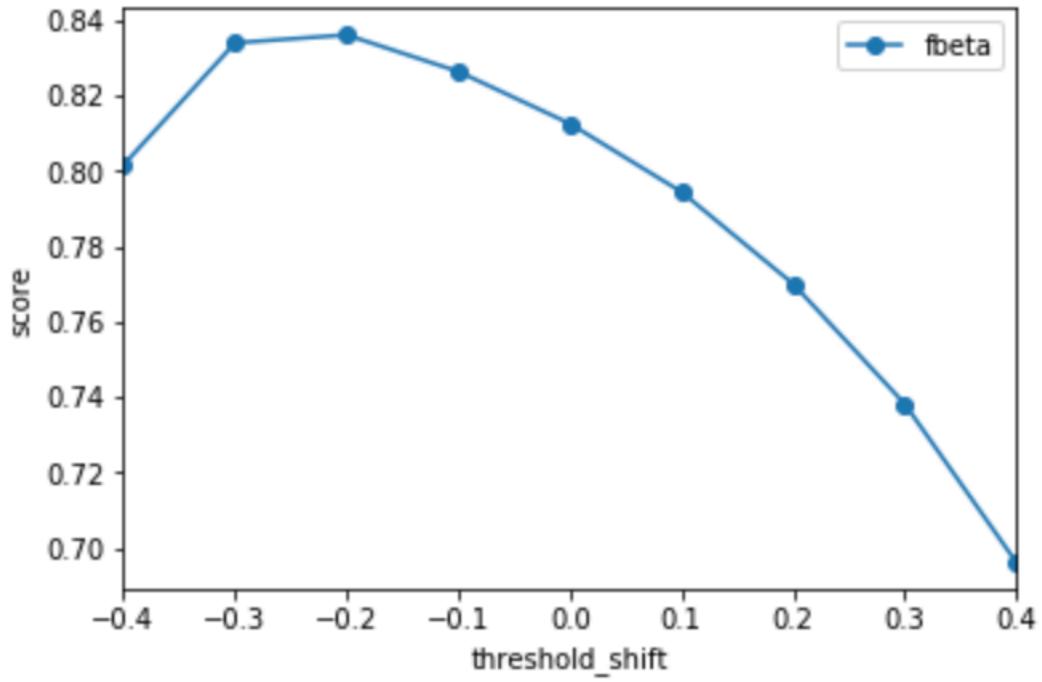

Figure 13 different threshold_shift

Justification

The CNN model performed better than the benchmark. This is due to the way both models are trained. The XGBoost model needs a human expert to extract useful features prior training, and the algorithm depends on his/her ability to identify useful features. The features we computed rely on the frequency of individual tokens. They do not capture the relationship among tokens in the sentence. Additionally, we have attempted to extract domain knowledge of each sentence using an LSA model and use that knowledge to compute their similarity score in

the vector space. LSA cluster documents (sentences) with the same topic close to each other in the space. It does not capture relationship among words within a sentence. Thus, this can explain the low fbeta score.

On the other hand, the CNN model extracts features automatically. It captures local and spatial relationships among words within sentences and uses that knowledge to compute their similarity. Everything is done automatically we don't need any domain expert in other to get good results.

Conclusion

Free form visualization

```
!!!!!!!---- Predicting Similarity ----!!!!!!!
Sentence 1:who is john doe ?
Sentence 2:who is the president of the united states ?
Both sentences are not similar with score: [ 0.09765264]
```

Figure 14 evaluation

In *figure 15,* our model is able to know that 'john doe' is not the president of the USA.

```
!!!!!!!---- Predicting Similarity ----!!!!!!!
Sentence 1:who is Donald J Trump ?
Sentence 2:who is the president of the united states ?
Both sentences are similar with score: [ 0.97374898]
```

Figure 15 second evaluation

In *figure 16,* our model is able to predict that Donald J Trump is the president of the USA, with a very high score near to 1.

```
!!!!!!!---- Predicting Similarity ----!!!!!!!
Sentence 1:who is Barack Obama ?
Sentence 2:who is the president of the united states ?
Both sentences are similar with score: [ 0.96072662]
```

Figure 16 third evaluation

In *figure 17*, our model is able to know that there is some similarity between Barack Obama and the president of the united states, indeed Barack Obama was our former president. This dataset was published in the early 2017, Donald J Trump was the newly elected president.

```
!!!!!!!---- Predicting Similarity ----!!!!!!!
Sentence 1:who is FRANCOIS Hollande ?
Sentence 2:who is the president of FRANCE ?
Both sentences are not similar with score: [ 0.00047577]
```

Figure 17 forth evaluation

```
!!!!!!!---- Predicting Similarity ----!!!!!!!
Sentence 1:who is EMMANUEL MACRON ?
Sentence 2:who is the president of FRANCE ?
Both sentences are not similar with score: [ 0.00053833]
```

Figure 18 fifth evaluation

Even though our model does well at predicting sentence similarity in the previous 2 examples. *Figure 17* and *Figure 18* show that the model does a bad job at predicting similarity between sentences that ask questions about French president names and their roles. This is a data problem. There wasn't enough instances in our training set to allow our model to learn about French presidents.

```
!!!!!!!---- Predicting Similarity ----!!!!!!!
Sentence 1:how to become a EU citizen ?
Sentence 2:how to become a EU citizen ?
Both sentences are similar with score: [ 0.99999994]
```

Figure 19 sixth evaluation

Futhermore when two sentences are identical the model prediction is almost 1.

Reflection

Our problem was to identify question pairs with the same intent. We used as a baseline an XGBoost model. We showed that traditional machine learning models require a domain expert to extract useful features. Our model performed better than the baseline. During the training process, I have encountered some challenges. The first one was during the data-preprocessing step. The challenge was to decide what tokens within sentences I should keep for training. Initially, I have removed English stop-words and punctuations in sentences, but by doing so my model performed poorly. As a result, I decided to keep all tokens. The model started having good results within few epochs (5). Another challenge that I faced was during hyper-parameter tuning. I tried L2 regularization, but there weren't any significant improvements. I ran it on about 10 epochs and compared the result. Using dropout and max-norm, I improved my model.

The final model can be used in a real-world application since misses won't affect negatively the user experience. The end user can help gather additional data to improve the model. This can be achieved by supplying the predicted similar question to the users when he/she asks questions. If the user agrees that the question he asked is similar to the one provided, we can provide him/her immediately with an answer.

The most interesting part of this project is that I started this project knowing nothing about natural language processing, and convolutional neural network. But by reading multiple research papers many time, I ended up having a depth knowledge in the subjects. I was able to use libraries to implement a research paper – this is the first research paper I ever implemented right by myself without any external help -. Furthermore, I was impressed how well the model were able to discriminate between syntactically different sentences such as the one with US presidents, and also somehow if it has given higher score to the actual president than the previous one. Additional, I have improved my skills in reading research papers.

Improvement

Besides collecting additional data from the end user, the model can be improved by tuning hyper-parameters as follow:

- Increase the embedding length of each word.
- Increase the number of filters
- Increase the number of epochs
- Try different values of drop-out, and max-norm
- Try different batch size

References

- Detecting Semantically Equivalent Questions in Online User Forums, Dasha Bogdanova.
- Efficient Estimation of Word Representations in Vector Space, Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, Google Inc., Mountain View, CA.
- Indexing by Latent Semantic Analysis, Scott Deerwester, sept 1990.
- http://cs231n.github.io/
- Convolutional Neural Networks for Sentence Classification, Yoon Kim, New York University