Question De-duplication

Chalumuri, Vivek - vivekc@kth.se
Run Yan Tan - rytan@kth.se

May 2019

Abstract

This project is aimed at exploring different ways of text representation which would in turn be used to solve the Question De-duplication problem posed on Kaggle [1] by Quora [2]. We contribute by exploring Natural Language feature engineering and we demonstrate that well-engineered Natural Language features can improve model performance. While the base model is a Random Forest model, we extend our work to more advanced methods such as using word embeddings from GloVe and sentence embeddings from InferSent, and also the LSTM network.

1 Background and Introduction

Quora [2] is a popular website used by millions of people to ask questions and get answers from all around the world. With about 10 million daily user visits on this platform, users often ask questions without exhaustively reviewing the existing answers. This results in a massive volume of duplicate questions. Multiple questions with the same intent can cause users to spend more time finding a good answer to their question, and stress writers in answering multiple versions of the same question.

To tackle this problem, Quora [2] published a challenge called *Quora Question Pairs* on Kaggle [1] in 2017. In this project we target to explore different approaches to solve this problem.

2 Related Work

Since this challenge was publish in 2017, it has become one of the most explored datasets for studying semantic similarity in text. There are numerous papers, blogs and git repositories exploring this problem. The methods range from hand-made features involving simple string differences to vectors generated from transformer networks and Bi-directional LSTMs.

Given the extent of pre-existing research and available knowledge, we performed a literature survey on the most common methods and on some published git repositories and found no evidence of extensive use of Natural Language features like Parts of Speech Tags and Named Entities for feature engineering. We hypothesized that these features could supplement features and methods, therefore we proceeded to research on such features 4.2 and engineer them in our project.

3 Data

The dataset consists of 400,000 pairs of questions along with the label stating if the question pair is a duplicate or not. A sample of the dataset is shown below:

Question-1	Question-2	Is_Duplicate
What Game of Thrones villain would be the most likely to give you mercy?	What Game of Thrones villain would you most like to be at the	1
How do guys feel after rejecting a girl?	Why do girls want to be friends with the guy they reject?	0

Figure 1: Sample Data

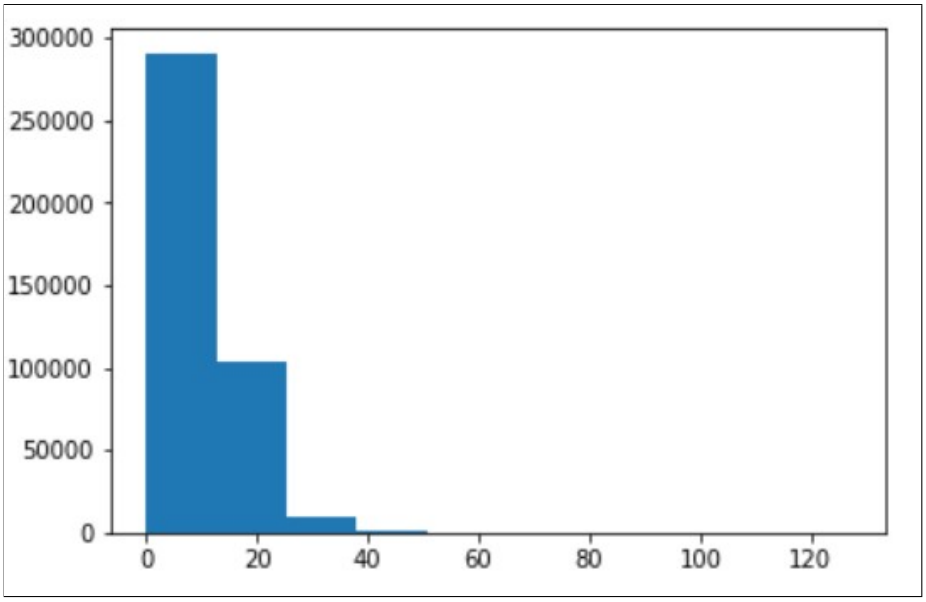

Figure 2: Question Length Distribution

4 Methods

We made use of the Scikit-learn package to extract word features from the dataset. The word vectorizer is used to tokenize the pair of questions into words. There are four main characteristics of the word vectorization:

- Removal of stop words
- Ignoring of punctuations
- Lowercasing of words
- Reducing diacritics to their base letters

Additionally, the word vectors could be represented in the 3 following ways:

- Boolean
- Bag of Words
- TF-IDF

In our experiments, we made use of the Random Forests model and explored Bag of Words and TF-IDFs features as these numerical quantities would provide more information for the model to learn more and make better classifications compared to boolean features.

Our incremental approach to the use of Natural Language features progresses in the following manner:

1. Model A1: Levenshtein Distance and String Jaccard Coefficients
2. Model A2: A1 + Part of Speech and Named Entities Features
3. Model A3: Bag of Words Features
4. Model A4: TF-IDF Features
5. Model A5: A2 + TF-IDF Features

Next, we explored word and sentence embeddings with neural networks, i.e. our model B and model C. This fulfils our objective of exploring advanced methods of text representation and textual feature extraction. At the same time, we are aware that the neural networks we used are not complex, so the results will not be as superior as deep networks.

1. Model B: GloVe Word Embeddings + LSTM
2. Model C: Sentence Embeddings + 3-layer Neural Network

4.1 Levenshtein String Distance and String Jaccard Coefficients

We started with features that can be obtained by string level comparisons between question pairs. They are:

- Levenshtein String Distance: Edit distance between the two questions
- Word Set Ratio: Ratio of unique words between the two questions
- Word sort ratio: Ratio that considers identical words in sorted order between the two questions
- WRatio : Ratio of words between the two questions after lower-casing and removing all attached punctuations

We used the library *fuzzywuzzy* [3] and *python-Levenshtein* [4] for the above features extraction.

4.2 Part of Speech and Named Entities Features

We hypothesized that Natural Language features like Parts of Speech tags and Named Entities could give us a lot of information about the semantic meaning of the question. Therefore we generated 4 such features from the question pairs. The procedures are as follows:

- Extract separately the set of Verbs, Adjectives, Nouns and Named Entities for each question.
- Apply the Jaccard Coefficient for n-grams to our case of words to derive a measure of difference in the contents of each set.
- Therefore, we would generate 4 such features, one each for Verbs, Adjectives, Nouns and Named Entities.

The metric used for generating the difference feature given sets A and B is:

$$\frac{(A \cup B) - (A \cap B)}{max(1,\ A \cup B)} \tag{1}$$

This metric ranges from 0 to 1; 0 corresponds to no difference and 1 corresponds to complete (maximum) difference.

4.3 Bag-of-Words & TF-IDF Features

Both of these methods use word frequencies to create vectors to represent documents (questions in our case), driven by the motivation that similar documents should have a similar distribution of words. Bag-of-Words representation is the count of each word in the document.

TF-IDF vectors enhance Bag-of-Words by taking into account a factor (IDF) that depends on the rarity of the word. In this sense, it gives more importance (more weight) to rare words and less importance (less weight) to common words. We reasoned that this would give our Random Forests model more information to classify the question pairs as opposed to Bag-of-Words.

$$tf\_idf = tf \times idf \tag{2}$$

$$tf(w) = \frac{No.\ of\ times\ word\ w\ appears\ in\ a\ question}{Total\ no.\ of\ words\ in\ the\ question} \tag{3}$$

$$idf(w) = log(1 + Total\ no.\ of\ questions\ /\ No.\ of\ questions\ with\ word\ w) \tag{4}$$

In our incremental model approach (A3 and A4), we initially trained our Random Forests model using Bag-of-Words and also using TF-IDFs, separately.

In our final A5 model, we combined features from TF-IDFs, Part of Speech and Named Entities Features, Levenshtein Distance and String Jaccard Coefficients. The TF-IDF vectors from the pair of questions are concatenated such that the final dimensionality which we passed into our Random Forests model is composed of 2 x $\|Vocabulary\|$ + the other Natural Language features described above.

In the construction of our Random Forests, the 2 main parameters we explored are:

- Number of trees: 5, 10* and 25
- Maximum number of sampled features per split: 20, 50* and 80
- Maximum depth: 3, 8, None*

We are also aware of the huge computations required for our large vocabulary features. The final dimensionality of the input features for model A5 is about 74,000. We achieve efficiency by using a sparse matrix implementation (Scipy sparse framework) to hold the feature data.

Eventually, we selected the parameters with * as they produced Random Forests models that can be trained within a reasonable amount of time (within 20 minutes), produced accuracies (79%-81%) very close to Random Forests models with larger parameters (implying a saturation), and produced accuracies similar to a few public git materials on this problem.

4.4 Word Embeddings

Word embeddings are the most popular and widely used representation for text analysis. Unlike any of the previous features, word embeddings offer us an insight into the context along with semantic and syntactic similarity. In order to

explore word embeddings, we used one of the largest pre-trained embeddings from Glove *glove.840B.300d.txt*[5], where each word is represented as a 300 dimensional vector. The next challenge for us is to represent a sentence using its constituent 300-d word vectors and then to feed it into an appropriate learning algorithm.

There are several approaches like averaging the word vectors or concatenating them to represent a sentence. While averaging can cause the loss of information, concatenating would multiply the dimensionality and possibly lead to performance issues. Our approach was to search for an algorithm which supports multi-dimensional input.

We used a LSTM with Glove embeddings (Model B). We padded our sentences to a fixed length of 25, which when passed through the embedding layer resulted in a matrix of size [25 x 300]. We then pass the questions separately through the LSTM cells and then concatenate their output to produce a [1 x 600] vector representing a complete pair of questions. Finally, we run the vector through 4 feed-forward neural network layers for classification.

4.5 Sentence Embeddings

While exploring more ways to represent textual questions, we came across an interesting open-source project from Facebook Research called InferSent [6]. InferSent is a method to encode the semantic representation of a whole sentence into a single vector. Therefore, we used its publicly available pre-trained model to directly encode our textual questions to obtain a single vector representing each question.

Having a single vector to hold all the semantic and contextual information of a question is an attractive idea because it allows us to subtract the two vectors from the two questions and then run several classification models on the difference vector. We applied the models Random Forests, XGBoost and a 3-layer Neural Network (Model C) on the difference vectors obtained from InferSent.

5 Results and Evaluation

Here we present the results of all our above approaches A1-A5, B and C.

Models A1-A5 use Random Forests with random stratified splits on 70% train data and 30% test data. For our deep learning models B and C, we use 60% train data, 10% validation data and 30% test data. For each model, we repeated the experiments over 15-30 runs to ensure that the results for each model are consistent with similar mean accuracies, mean precisions and mean recalls; and also with acceptable variances of these measures.

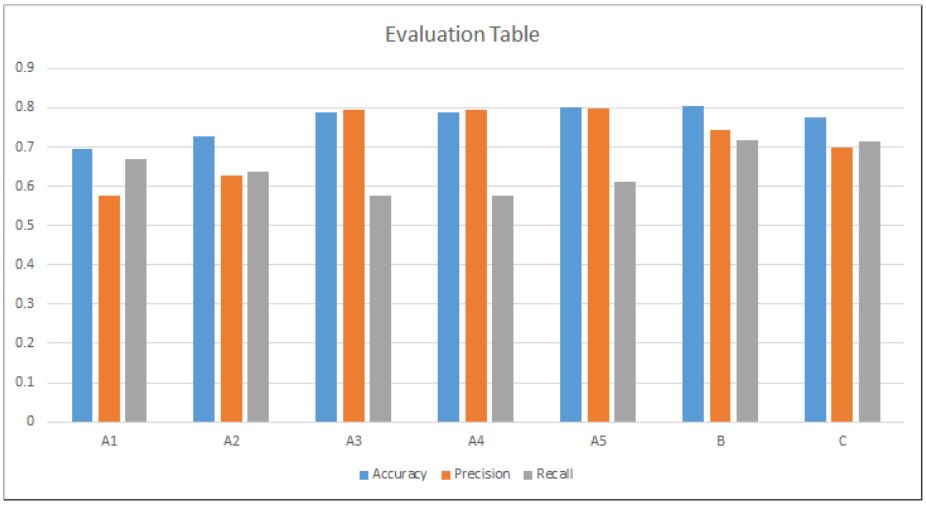

Figure 3: Results (Refer to section 4 for the models A1-A5, B and C)

Our results have supported our case for well-engineered features. Model A5, a Random Forests model built on all the natural language features and tf-idf vectors, produced an accuracy of (80.6%) that is similar to LSTM (80.8%) and better than 3-layer neural network with sentence embeddings (77.5%). Furthermore, its precision (80.7%) is higher than LSTM (74.3%) and the 3-layer neural network (69.7%). However, its recall is worse than the other two models; this can be seen as a tradeoff between precision and recall. Given the sequential enncoding and passing of information, LSTMs are regarded as one of the best methods to handle textual data - even our simple LSTM implementation produced a better recall than our Random Forests.

Furthermore, we demonstrated that proper feature engineering procedures such as normalization of our natural language features (dividing by maximum, or shifting by mean and dividing by standard deviation) improved the accuracies, precisions and recalls by up to 1.2%. We reasoned that normalization ensures all features contribute relatively evenly to our Random Forests model, and it is up to the model to learn from these similarly scaled features on how to split them based on the labels (with the objective of maximizing reduction of entropy).

6 Conclusion

Being a Kaggle problem, methods have been published over the past two years to solve this problem. Even though many deep and complex architectures for this problem are available online for reference, we wanted to explore methods that are not published, especially in the area of language feature engineering.

Using Natural Langauge features like Part-of-Speech tags and Named Entities have resulted in good incremental results (about 3%-4% gain in recall) even with the traditional Random Forests model. Furthermore, in our experimentation, concatenating tf-idf vectors of the two questions produced a much better result than averaging or subtracting them (with about 7%-9% gain in accuracy). This is strong evidence to the potential loss of information (and dimensionality) from the averaging and subtraction of vectors. Our intuitive idea of concatentating the tf-idf vectors from both questions and encouraging the Random Forests to find its best entropy-reducing decision paths has worked well.

Given that tree models like Random Forests are more tractable and explainable than Neural Networks and can result in a good performance with lesser training time and computational resources compared to Neural Networks, we believe the former are worth investigating with an added effort into feature engineering.

InferSent[6] is an insightful takeaway from our exploration in this project. As a pre-trained Bi-LSTM tool to encode semantic representation of whole sentences, it could speed up many language engineering and model building tasks. In particular, we achieved the same recall using these embeddings with a simple 3-layer neural network in comparison to the LSTM model. Interestingly, this simple neural network was much faster than LSTM model in terms of training time. Therefore, we are very receptive to the idea of sentence embedding with semantic context as this is a big step forwards from word embeddings.

References

[1] Kaggle Problem Statement. https://www.kaggle.com/c/quora-question-pairs. Accessed: 2019-05-19.

[2] Quora Website. https://www.quora.com. Accessed: 2019-05-19.

[3] FuzzyWuzzy python library. https://pypi.org/project/fuzzywuzzy/0.3.0/. Accessed: 2019-05-19.

[4] python-Levenshtein library. https://pypi.org/project/python-Levenshtein/. Accessed: 2019-05-19.

[5] Glove Embeddings library. https://nlp.stanford.edu/projects/glove/. Accessed: 2019-05-19.

[6] InferSent sentence embeddings. https://github.com/facebookresearch/InferSent. Accessed: 2019-05-19.