Deducing Similarity from Quora Question Pairs

ECE 6254 Statistical Machine Learning

Term Project Report (Spring 2017)

Esther LING lingesther@gatech.edu

LIU Conghai lconghai3@gatech.edu

Melvin MATHEW melvin.julian.mathew@gatech.edu

Summary

Quora's question pairs dataset challenge essentially is a problem of quantifying semantic similarity between two short texts. We obtained sentence embeddings by summing pre-trained word vectors (word2vec and GloVe) of each word in the question, scaled by its term-frequency inverse-document frequency (tf-idf) weights. Two approaches were then considered: combining the sentence vectors with handcrafted features in a random forest model; and feeding the vectors into a neural network. For the first approach, significant effort was placed into 1) enhancing the effectiveness of the sentence embedding; such as augmenting parts-of-speech (POS) and named-entity-recognition (NER) tags to the vectors; and 2) evaluating the features using Recursive Feature Elimination (RFE) and a Random Forest model, before feeding into the Random Forest classifier. In the neural network approach, effort went to architecture customization. A Glove/word2vec+LSTM model architecture was used, with a total training time of 1 hour per each time (5 minutes per epoch, on NVIDIA GTX980M). The log-loss of each approach was computed using 20% of the initial 400,000 size dataset. Then, each result was tested using Kaggle's test 2 million unlabelled dataset. From the competition score, the random forest model yielded a log-loss of 0.43518, with rank 955/1588 teams, while the neural network gave our best score of log-loss 0.29832, ranking 185/1588 teams[1].

1 Introduction

Classifying if two questions are redundant plays an important role for websites that offer a question-answer service, such as Quora, Stack Exchange, Piazza, and general forums. Under this backdrop, this project works on Quora's dataset question-pair dataset [1]. Figure 1 contains a preliminary visualization of the training dataset. A brief overview of the original dataset is provided below:

- Approximately 400,000 question pairs
- Labels: ○ 0: Not duplicate ○ 1: Is duplicate ○ Contains some noise
- Imbalanced: negative samples (63%), positive samples (37%)

After starting the project, Kaggle released the Quora Question Pair competition, with an added test dataset of 2 million pairs, mostly containing computer generated questions.

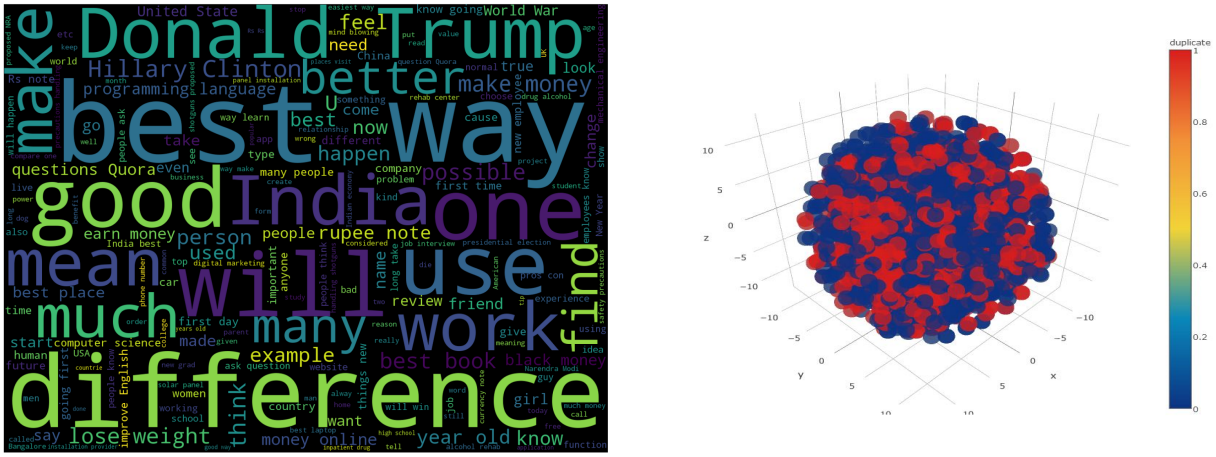

Figure 1: (left) Most Common Words via WordCloud library; (right) TSNE of Two Classes via sklearn library

[1] Both ranking as of 17-04-2017 1:07 AM GMT +8

2 Background

Question similarity classification is a problem of distinguishing semantic similarity between two short texts. Semantic similarity has traditionally been studied using a lexical database like Wordnet, but recently, directly representing words in a vector space (word embeddings) have gained interest with Mikolov et al. development of word2vec [2]. In a well trained word2vec model, related words are close to each other in the vector space. For example, the vector for the words "king" and "queen", both having a sense of royalty, would be close.

However, forming a sentence vector from word vectors that faithfully maintains the semantic meaning of the text poses the main challenge. One could average all the word vectors in a sentence, however, this loses word order [3] and places equal emphasis on all words. Removing stop-words may not be effective as well, given that the text is short and context may be lost. Boom et. al [4] reported that scaling the word vectors with tf-idf [2] weights, which assigns each word relative importance, yielded promising results for short text.

In addition to word embeddings, evaluating similarity could be done using handcrafted features. A naive method would be related to the structure of the sentence, such as the number of words, characters and common words. One could also use parts-of-speech (POS) tags and named-entity recognition (NER) tags to label the type of question. Still another could be counting the edit distance to convert the first sentence into the other, also known as Levenshtein's Distance.

For evaluating features, several techniques exist, such as filter methods, wrapper methods and embedded methods. Filter methods creates a ranking of features, but fails to capture interactions between features. On the other hand, wrapper methods tests performance of subsets of features, however, takes a longer time. This suggests a combined effort: first rank the features to remove redundant ones, and then test subsets of features.

Given the best *d* features, a recommended approach in selecting a model involves testing several kinds of models in order of complexity [5]. Thus, we start with a linear model (logistic regression) as the baseline, and then move to nonlinear models (support vector machines, random forest model, neural networks).

For deep learning method, we used long short-term memory (LSTM) as our classifier. LSTM is a recurrent neural network (RNN) architecture. Like most RNNs, an LSTM network is well-suited to learn from experience to classify, process and predict time series when there are time lags of unknown size and bound between important events. Relative insensitivity to gap length gives an advantage to LSTM over alternative RNNs and hidden-markov models and other sequence learning methods in numerous applications. Among other successes, LSTM achieved the best known results in natural language text compression. Thus, we use LSTM in our deep layers.

3 Feature Engineering (Team Lead: Esther)

Generating Sentence Embeddings

We used a pre-trained model of word2vec (Google-News-Vectors corpus of 3 billion words, with 300 features). Then, we experimented with different sentence embeddings:

[2] Term frequency - inverse document frequency

- Apply tf-idf weights to each word vector, then sum and normalize (Figure 2)
- Create new vectors from parts-of-speech (POS) and Named-Entity-Recognizer (NER) tags based on the count (POSNER vectors)
- Augment POSNER vectors to tf-idf weighted word vectors
- Augment basic features to POSNER tf-idf weighted word vectors

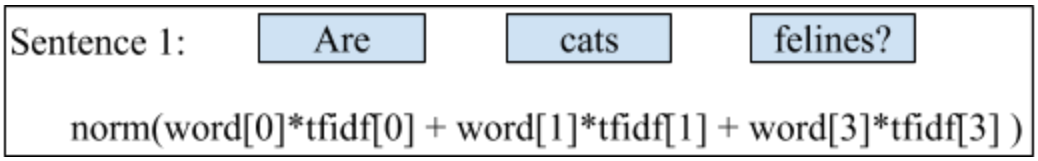

Figure 2: tf-idf weights to each word vector

Method for generating the POSNER vectors [3] (12 dimensional, normalized):

- POS: Take the count of verbs, adjectives, cardinal number, foreign word, adverb and "what, how, why" words. The tags were referenced from the Penn Treebank Project [6]
- NER: Take the count of Location, Person, Organization, Money
- Have an auxiliary column for taking the count if the word does not fall under the chosen POS or NER categories. This indirectly encodes word length in the POSNER vector.

By having tagged words with the POSNER vectors, an example like: "Did he chip into the discussion?" and "Were the chips in your favour?", the word "chip" is differentiated as a verb and a noun, and can possibly improve the performance of word2vec.

After generating the sentence embeddings, we computed various distances between the vectors (Cosine, Euclidean, City Block, Jaccard, Minkowski and Bray Curtis).

Applying WordNet for Synonyms

We also tested WordNet path similarity (based on hypernym/hyponym taxonomy distance). This was based on a limitation of word2vec. For example, "Germany" and "France", both having a notion of "country" would be close in the vector space. Thus, the sentences "Are cars from Germany?" and "Are cars from France?" would be classified as the same.

We broke up each question into three sub-sentences, consisting of nouns, verbs and adjectives (Figure 3). Then, each sub-sentence was compared with the corresponding noun/verb/adjective sub-sentence from the other question. Each word was assigned the maximum score it achieved, and then the total score for all words in the sub-sentence was used as a feature.

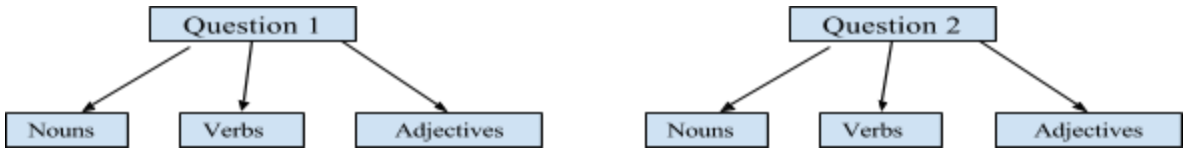

Figure 3: Creating Noun, Verb and Adjective Sub-sentences

Basic Features

These were features related to the structure of the sentence: Number of words, characters, common words, and Levenshtein distance between sentences.

[3] We first used the Stanford NLP library [7], but these took a long time to process. We then used the taggers available in the NLTK library.

4 Feature Evaluation (Team Lead: Melvin)

In total, we obtained 65 features. From the scikit-learn library, we made use of Recursive Feature Elimination (RFE), which is a wrapper method that assesses each feature's impact on the decision boundary for the classifiers explored. This is done by repeatedly reconstructing the classifier, each time removing a subset of the features used, until eventually every feature is exhausted. Weights are assigned to each feature in this process, and those with the lowest weights are pruned out of the feature space at each iteration.

5 Classification (Team Lead: Melvin)

The main challenges in classification include, training time and handling the imbalanced dataset. Due to training time constraints, we chose to evaluate two non-neural network classifiers, namely, a Logistic Regression (LR) model as the baseline model, and a Random Forest (RF) model as our final classifier. We had initially tried a Support Vector Machine (SVM) but found that it took over 12 hours on average to train per model. The RF model is inherently capable of handling the imbalanced dataset, as it averages several decision trees that are constructed using only $\sqrt{d}$ random features out of all possible d features. For the LR model, we tackled the imbalance by subsampling the dataset. The dataset is skewed in favor of more negative than positive samples, so we took the same number of negative samples as there were available positive samples.

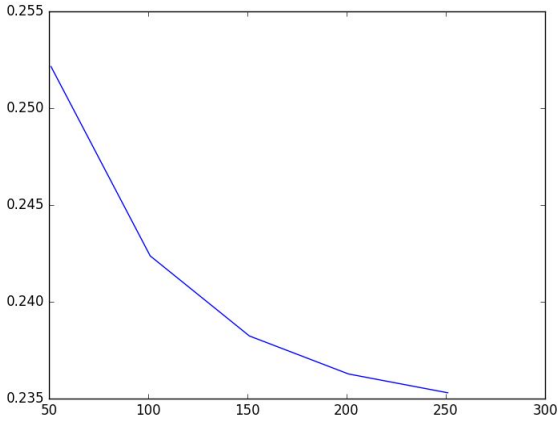

Figure 4: RF Model; tuning number of trees based on out-of-bag error

The training procedure is as follows:

- Split the features of the dataset into two main portions; 80% for training, 20% for testing
- From the 80% for training, we balanced the dataset by subsampling as described above
- Normalize the balanced training set and normalize the testing set using the mean and standard deviation of the training set[4]
- From the balanced dataset, we carried out cross-validation over a range of the parameters; LR's cost C parameter; RF's number of estimators/trees parameter
- The actual model is then trained on the whole 80% of the training data
- Finally, the classifier is tested against the normalized 20% portion of the dataset

[4] Balancing and normalization only required for the LR model

6 Extensions: Deep Learning (Team Lead: Conghai)

We implemented a Glove/Word2Vec+LSTM model on Keras. We tried both Stanford Natural Language Inference benchmark and Google News Word2vec tool, using the Glove word embedding to represent each question in the pair. Here we obtained a constructed vocabulary vector from training text like Google News or Stanford Common Crawl corpus. We used those vector files as features in our model. Up to now, we obtained about 86% accuracy with deep neural network which comprised of two translation layers, one for each question input, initialized by Google News word2vec embedding, two bidirectional LSTMs also with this embedding. Then it was followed by series of dense layers with dropout and batch normalization (we use 4 here). The network architecture is shown below in Figure 5.

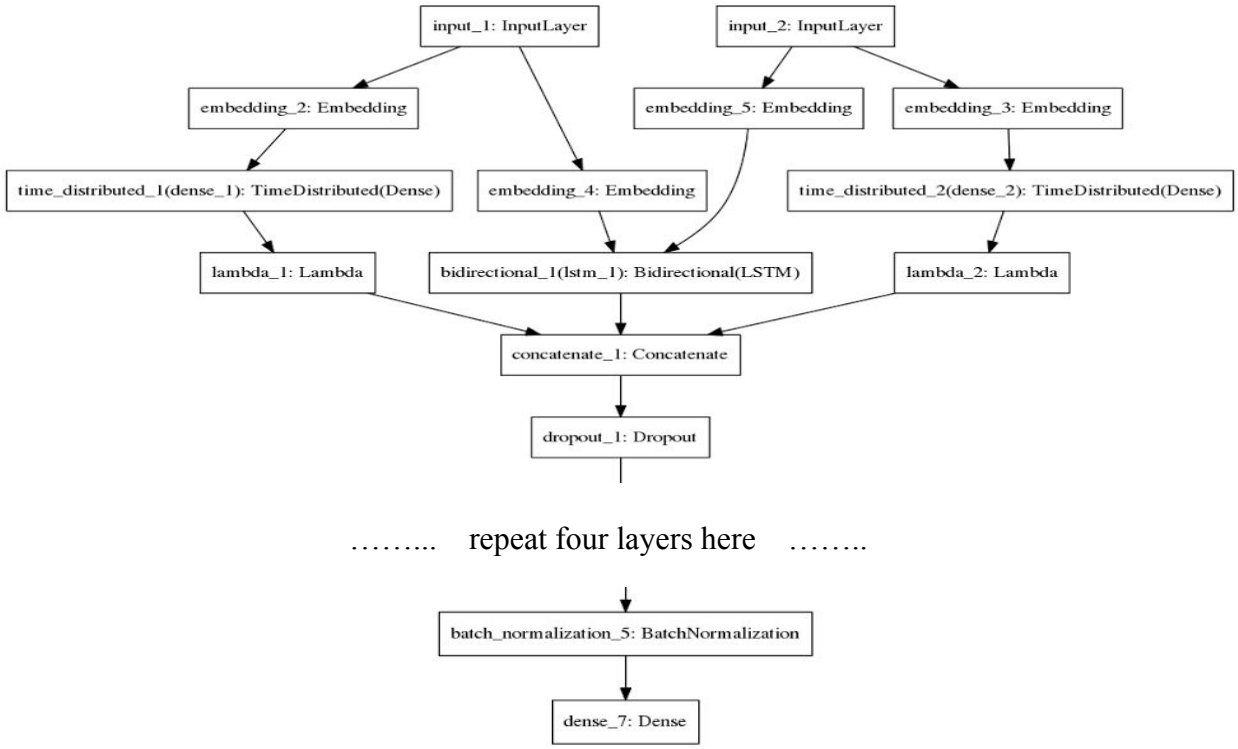

Figure 5: Neural Network Architecture

7 Results

Table 1: Comparison of classifiers

Classifier	Training Time (approx)	Original Dataset [Test] Log-Loss	Kaggle Test Set Log-Loss
Logistic Regression [imbalanced]	1 minute	0.27252 (70.22% accuracy)	N/A
Logistic Regression [balanced]	30 seconds	0.1897 (69.49% accuracy)	N/A
Random Forest Model [251 trees]	5-15 minutes	0.22352 (76.72% accuracy)	0.43518 (Rank 955/1588)
Neural Network	1-2 hours	0.2561 (86% accuracy)	0.29832 (Rank 185/1588)

The best subset of features (of size 5) after running feature evaluation on all 65 features:

- Euclidean Distance (word2vec)
- Canberra Distance (word2vec)
- Cosine Distance (tf-idf + POSNER + basic features word2vec)
- Cosine Distance (tfidf word2vec)
- Minkowski Distance (word2vec)

8 Discussion and Conclusion

Our feature types consisted of word embeddings, WordNet path similarity between word types and sentence structure. Based on RFE, the word embeddings performed the best. Moreover, the original word2vec embedding performed the best, followed by tf-idf+POSNER+basic features word2vec embedding and then tfidf word2vec embedding. One reason for this may be that the trained word2vec model already has the most useful features in some way that a simple summed sentence embedding can capture. When assessing the classifier results, the neural network was better in approximating the classifier using the basic word2vec sentence embedding, compared to hand tuning the random forest with different subsets of features. Also, the handcrafted features together with the LR and RF model took more human effort hours, compared to training the neural network.

Conclusions:

- Major difficulty is in finding good embeddings for short text fragments
- Still much room for research to develop better sentence embeddings
- Given the same embedding, the neural network was able to approximate the better model
- Suggests that the better approach is to develop sentence embedding and feed it directly to neural network

References

[1] “First Quora Dataset Release: Question Pairs - Data @ Quora - Quora.” [Online]. Available: https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs. [Accessed: 22-Mar-2017]
[2] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013 [Online]. Available: http://arxiv.org/abs/1301.3781
[3] Le, Q. and Mikolov, T., Distributed representations of sentences and documents. In Proceedings of the 31st International Conference on Machine Learning (ICML-14) (pp. 1188-1196), 2014. Online]. Available: https://arxiv.org/pdf/1405.4053.pdf
[4] De Boom, C., Van Canneyt, S., Bohez, S., Demeester, T. and Dhoedt, B., Learning semantic similarity for very short texts. In Data Mining Workshop (ICDMW), 2015 IEEE International Conference on (pp. 1229-1234), 2015 [Online]. Available: https://arxiv.org/pdf/1512.00765.pdf
[5] M.A. Davenport, ECE-6254, Gatech, Model Selection, slide 25/25, “When to use deep learning?” [Online]. Available: http://mdav.ece.gatech.edu/ece-6254-spring2017/notes/25-deep-learning.pdf
[6] “Alphabetical list of part-of-speech tags used in the Penn Treebank Project”, [Online]. Available: https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
[7] "Stanford CoreNLP – a suite of core NLP tools | Stanford CoreNLP", [Online]. Available: https://stanfordnlp.github.io/CoreNLP/