Deep Learning and Determining Duplicate Questions

Jacob L. Fenger
Oregon State University
fengerj@oregonstate.edu

Spike Madden
Oregon State University
maddens@oregonstate.edu

Chongxien Chen
Oregon State University
chencho@oregonstate.edu

Abstract

This document contains the approach Team Jacob utilized to train a classifier for detecting whether questions are duplicates. We utilized Word2Vec embeddings for vectorizing the questions as well as Keras with a Tensorflow backend for training.

1 Introduction

Quora Question Pairs is a Kaggle competition requiring challengers to use advanced techniques to determine if pairs of questions are duplicates. You can find the challenge at:

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

Challengers will need to utilize natural language processing as well as deep learning in order to create a good classification. Our team had no previous experience with natural language processing and during our implementation of the challenge, we learned a lot. There are many aspects to consider when converting sentences to number vectors. This process is otherwise known as vectorization.

Our first approach got a score of while our second approach got a score of. Explanations of these approaches are outlined in the following sections.

2 Approach #1

The first approach was influenced by a user named Lystdo who posted a kernel for the challenge. Our approach for the challenge was heavily influenced by his mainly because we had no experience with the natural language processing side of things.

2.1 Text Pre-Processing

To ensure accurate results, we pre-processed the text. Initially, this just involved removing all punctuation as well as making everything lowercase. This is necessary because many questions, especially those in the test set, containing words with weird capitalisation or spelling. Our knowledge regarding natural language processing is very limited, but we had confidence that punctuation in the questions provide no additional meaning.

Later on in the implementation of this challenge, we realized that some questions involved contractions, slang terms, or spelling mistakes. We tried our best to handle those by doing replacements with commonly known contractions/slang terms, but some still may have gotten through due to the large data set. The testing set was generated by Quora and many of the questions had spelling mis-

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

takes. A user called 'Currie32' had an interesting discussion post regarding the cleaning of text for the challenge.

2.2 Feature Extraction

After reading all of the data and pre-processing the text, a necessary step is to analyze the text and generate numeric feature vectors because the majority of machine learning algorithms require them. This process is otherwise known as vectorizaton.

We first tokenized the questions and then padded the sequences so they were all the same size. Using a pre-trained word2vec model is very helpful in the vectorization of things. There are several pre-trained models in existence. Word2Vec by Google and GloVe by Stanford are just two examples. We decided to use Word2Vec for our implementation. There is a neat package that helps with the interface to Word2Vec called Genism which is what we used as well.

2.3 Training

We utilized Keras and Tensorflow to build a long short term model (LSTM) for the classification. This consisted of an embedding layer involving an embedding matrix generated from the Word2Vec vocab. Other layers included dense and dropout layers with batch normalization in-between. Batch normalization helps with the normalization of data between layers (In short).

2.4 Results

The score we got with this approach was about After updating the text pre-processing with contraction expansion, we got reduced the log-loss by several points.

3 Approach #2

The second approach was influenced by a user named 'anokas' who posted an approach (Kernel) for the challenge. We decided to base our approach off of his and to see if we could get a higher accuracy by making necessary changes.

3.1 Analyse data

It is important to understand how our data looks like before we can train a good model. We first check how many samples there are in the train and test. Then a couple other things that can help us understanding the data. Are there many typos in the question? How many words are usually in a question? What are the words that appear the most often? How about the punctuations in the question?

3.2 Feature Extraction

TF-IDF approach

Using wordcloud, we find that many words that appear often doesn't have significant impacts on whether these questions are duplicate or not. For example, from the picture we see that "good" appears extremely often in the questions. But we can't really determine how similar two questions are based on the word "good". It could even be misleading if there is a "not" precedent to "good". It also makes sense to weigh "Bruce Lee" more than "the".

A more valuable feature to look at will be TF-IDF (term-frequency-inverse-document-frequency). If a word that doesn't appear very often in the database appears in the two questions we are comparing, then it is reasonable that we weigh this word more in the prediction of whether these two questions are duplicate.

3.3 Training

We split our train.csv into training data and validation data. Using XGBoost, we generate a prediction csv for questions in test.csv. We get a score of 0.35372 on Kaggle.

4 Simple Approaches to Combining Models

Since we take different approaches within our team and generate two csv files, we start to thinking how can we take advantage of two good models and get a better submission.

But the challenge is how do we combine two confidence and generate a new reasonable confidence? What equation will produce a better result? I start with using average mean, then I also tried geometric mean.

4.1 Average Mean of Confidence

We take the average mean of two confidence and generate our new submission.

4.2 Geometric Mean of Confidence

We take the geometirc mean of two confidence and generate our new submission.

5 Comparison of Results

6 First Place Solution

A member of the DL guys, who placed 1st in the competition, provided a summary of their approach to the Quora Question Pairs. They had a log loss score of 0.11580.

6.1 Features

Embedding features include word embeddings, sentence embeddings and encoded question pairs. As discussed in a previous approach, Word2Vec was used for the word embeddings. Doc2Vec and Sent2Vec were used for sentence embeddings but were found to not be as informative as the word embeddings from Word2Vec. Classical text mining methods were used by tokenizing the strings and looking for similarity measures on bag of character n-grams, question length, capital letters and punctuation, and certain keywords like are, can and how. Structural features were extracted from graphs built from the edges between pairs of questions from the training and testing data sets.

6.2 Models

There were two architectures for the neural networks that were used: a siamese LSTM network with GloVe embeddings and a decomposable attention network with FastText embedding. The best single models were found to be these neural networks trained on both text sequences and the text mining and structural features.

6.3 Ensemble

They used stacking with 4 layers to combine their models. The first layer, consisting of about 300 models, includes various neutral networks, algorithms like XGB and LGBM as well as Scikit classification algorithms. The second layer consists of 150 models and includes the input features and hidden layers of the best L1 pure text EISM - Evolutionary Support Vector Machine Inference Model. The third layer consists of 2 linear models and the fourth is for blending.

7 Conclusion

Individual Contributions