Identifying duplicate questions from Quora
Annamalai Narayanan
13 Oct 2017

Solution link: https://github.com/MLDroid/quora_duplicate_challenge
The problem:
Given more than 400,000 pairs of questions from Quora, as you know, the challenge is about predicting whether or not a given pair of questions are semantically same.

Stats:
Before starting out on the Machine Learning portion of the solution, here are some preliminary stats that I looked at:

Total # of samples: 404290
# of positive samples (semantically same questions): 255027
# of negative samples (semantically different questions): 149263
Max length of a question1 (in words): 125
Max length of question1 (in characters): 623
Min length of a question1 (in words): 1
Min length of question1 (in characters): 1
Avg length of a question1 (in words): 10.94
Avg length of question1 (in characters): 59.53
Max length of a question2 (in words): 237
Max length of question2 (in characters): 1169
Min length of a question2 (in words): 1
Min length of question2 (in characters): 1
Avg length of a question2 (in words): 11.19
Avg length of question2 (in characters): 60.11
Unigram vocab size: 85,846
Bigram vocab size: 1,293,732
Trigram vocab size: 2,825,036

The setup:
I have used a server (with Ubuntu 16, 128 GB of RAM and Tesla M40 GPU card) for this task. The entire code was done in Python. The following packages were (mainly) used:

- Pandas
- Sklearn
- Keras (TF backend)
- Xgboost
- NLTK
- Gensim

- Networkx
- Fuzzywuzzy

Train-Test splits:
Training set: 90% of the samples (incl. 33% validation set for 3 fold CV)
Test set: 10% of the samples
All the classifier models are selected based on 3 fold CV (optimized for accuracy)

Preprocessing:
I have used the following preprocessing steps in all the methods that I attempted (except for word embedding based deep learning approaches):
General preprocessing

- Incorrect spellings were corrected (using Python autocorrect package).
- All the words in the question pairs are converted into lowercase.
- English stopwords (e.g., an, the, etc.) are removed. NLTK stopwords [1] were used in this process.
- Words are stemmed (using NLTK porter stemmer [2]).

The solutions, results and discussion
The approaches that I have attempted:

Table 1: Comparison of Machine Learning models for identifying similar questions on Quora

Model	Accuracy (in %)
TF IDF (upto trigrams)	80.32
TF IDF (upto trigrams) + SVD	72.27
Doc2vec	60.91
Siamese LSTM	80.91
Siamese Deep Network (1D CNN + LSTM)	82.32
Engineered feature + xgboost	81.79
Graph of words (novel approach)	60.68

1. **TFIDF, TFIDF+SVD**: This method uses the famous baseline text categorization features, namely, term frequency-inverse document frequency from the Bag of Words (BoW) model [3]. Firstly, a vocabulary of all the ngrams (upto trigrams) were extracted each question is converted into a TFIDF vector (using sklearn's TfidfVectorizer). Singular Value Decomposition (SVD) is used to reduce the dimensionality of these vectors

and to accentuate semantics of words. The pair of questions were concatenated and fed to the classifiers like Linear SVM, xgboost and MLP classifier. The code for this feature+model is available in the files: tfidf_model.py and classify.py. The accuracy of this model is around 80%.

2. **Doc2vec** (initialized with the word embeddings): Alternative to the hand crafted features used in the ngram based BoW model mentioned above, one could use representation learning for document embedding. A state of the art method for such a model is Le and Mikolov's [4] doc2vec. This model uses a single layer feedforward neural net to learn the document embeddings. This model heavily leverages training on large volumes of data. However, our dataset of 400,000+ sentences is way too less for the model to learn meaningful representations. This reflects in the poor performance of the model. This poor performance is despite the fact that I have initialized the weight of the output layer using pretrained word embeddings. The code for this feature+model is available in the files: doc2vec_model.py and classify.py. The accuracy of this model is around 68%.

3. **Siamese LSTM**: Siamese neural network architecture proposed by Bromley et al [5] is an effective way of assessing whether or not a pair of objects are similar. Originally, this architecture was tested on image similarity assessment tasks. However, [6] extended the same for assessing similarities of sentences. I used a Siamese LSTM to find whether or not a pair of questions are semantically same. This model performs reasonably well (nearly 81% accurate). However, stacking carefully crafted layers of convolution/LSTM and RELU can help improve the performance. The code for this model is available in siamese_lstm.py.

4. **Siamese Deep Neural Net** (state of the art): As mentioned in point (3), the Siamese architecture with carefully crafted layers of convolution/LSTM, RELU and dropout was proposed by Abishek Thakur in the Kaggle challenge for this task [7]. I used this solution and found that it performs excellently. This reinforces the fact that deep feature learners (which are fine tuned for this task) could help circumvent the limitations of using hand crafted features. This model achieves more than 82% accuracy. The code for the same is available at [7].

5. **Feature engineering + xgboost**: Again, Abisek Thakur proposed [7] a simpler approach (i.e., w/o deep learners) for achieving near state-of-the-art results. This model uses features such as length of questions in terms of both words and characters, difference in their lengths, various distance measures (such as cosine, manhattan distance, levenshtein distance, word mover's distance etc.). This model uses xgboost classifier. Its results are very close to the best model i.e., Siamese deep net.

6. **Graph of words**: Finally, I did attempt a novel approach to identify the similarities between every pair of question (NOTE: in general, a value of similarity between a pair of

objects is also referred to as kernel value). To this end, I extended the work of Nikolentzos et al [8]. Initially, a question is modeled as graph (nodes in this graph are words and the linear sequence of order of words determine the edges in the graph). Then, all the shortest paths (across all the nodes) in a given pair of graphs are computed. Finally, the similarity between the given pair of graphs is computed using the modified shortest path kernel (pls refer to [] for more details). Originally, this work didn't involve POS tags. I added POS tags to the graphs. However, I observe that the POS tags do not help in boosting the accuracy of the model significantly!

Other approaches that I considered using, but failed to implement (as I ran out of time):

1. Attention based LSTM models
2. Bi LSTM models
3. Attention based BiLSTM models
4. FastText
5. Multichannel CNN (based on Yoon Kim's [] work)
6. Deep Graph Kernels + Graph of words

References:

1. http://www.nltk.org/book/ch02.html
2. http://www.nltk.org/howto/stem.html
3. Manning, D. A. C. "Introduction." *Introduction to Industrial Minerals*. Springer Netherlands, 1995. 1-16.
4. Le, Quoc, and Tomas Mikolov. "Distributed representations of sentences and documents." *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*. 2014.
5. Bromley, Jane, et al. "Signature verification using a" siamese" time delay neural network." *Advances in Neural Information Processing Systems*. 1994.
6. Mueller, Jonas, and Aditya Thyagarajan. "Siamese Recurrent Architectures for Learning Sentence Similarity." *AAAI*. 2016.
7. https://github.com/MLDroid/quora_duplicate_challenge/tree/master/baselines/abiskek_thakur
8. Nikolentzos, Giannis, et al. "Shortest-Path Graph Kernels for Document Similarity." *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. 2017.