Data Mining Proposal: Quora Kaggle Competition

Luis Duque, Joa Jin, Ethan Leeman, Yingnan Liu, Sophia Zheng

March 27, 2017

1 Introduction

For our project, we propose competing in a newer Kaggle competition, “Quora Question Pairs.” Quora is a question and answer website where users can both submit and answer questions in a collaborative manner. While handling a large volume of queries, duplicate questions are asked often, making it difficult for popular, accepted answers to be found. Currently Quora uses a Random Forest model to identify identical questions, and Quora proposed a Kaggle competition to tackle this machine learning problem in natural language processing.

2 Data

Kaggle provides a training dataset and a test dataset. In the training set, there are approximately 400,000 question pairs and the target variable, whether Quora feels the questions are duplicates. Some questions appear multiple times in different pairs. For example, the question

“How can you determine the first ionization energy of lithium?” *is distinct from*
“How is the ionization energy of silicon determined?”

and that

“Will Donald Trump shut down the internet?” *is a duplicate of*
“If Donald Trump becomes president will we lose the internet?”

One issue with the data is the subjectivity of the target variable. For example, the question

“I am 24. Is it too late to get into medicine?” *is distinct from*
“Is it too late to study medicine at 23?”

but could be reasonably interpreted as duplicate or as distinct. Additionally, there is some natural human error and noise in the data. The following question pair is erroneously marked:

“What is the cultural shock?” *is distinct from*
“What is Culture Shock?”

Another issue is that the characters are in unicode, and sometimes questions with non-english words appear.

The test data is about 2,000,000 question pairs and the competitors are asked to submit a probability of being duplicate, and the submissions are scored on Logarithimic Loss.

3 Possible Approaches

Kaggle provides a discussion board, and rewards competitors for providing helpful guides and comments to other commenters. In particular, there is a "beginner's guide" by user "shubh24" [3] which provides many techniques in natural language processing, and provides a baseline for what other competitors are doing. One first idea is to take every sentence, create a set of every k strings of words, after stripping the stopwords, and compare the overlap by some metric. For $k = 1$ we would measure how distinct the words are in each question, $k = 2$ would be word-pairs, and so forth. One key issue is that a single word, even if the rest of the sentence is identical, can drastically change the question, as in the lithium vs. silicon example above. Another useful tool outlined in the above post is Wordnet. Wordnet is an English dictionary which also contains a directed graph showing relationships between words. One possible feature we could make would be to find the rare words and see how distant they are. This would try to find questions that are asked with different synonyms or similar phrases.

3.1 A Naïve approach.

All words in the English language can be classified as one of the eight different parts of speech: nouns, verbs, pronouns, adjectives, adverbs, prepositions, conjunctions and interjections. A first naïve approach would be to produce, for each question, 8 buckets of words (each containing a part of the speech) and a bucket of unidentified words. For each pair of questions we call this buckets $nouns_1, verbs_1, \dots, interjections_1, unidentified_1, nouns_2, verbs_2, \dots, interjections_2, unidentified_2$. For each pair of questions we could produce a feature by measuring how different this buckets are in some sense.

A way to produce this feature can be the following; take for instance the pair of buckets $(verbs_1, verbs_2)$. We could compare each element on $verbs_1$ with each element in $verbs_2$ (using Word2vec) and add this difference with the euclidean norm to produce $verbs_{feature}$. Similarly we could produce $nouns_{feature}, pronouns_{feature}, adjectives_{feature}, adverbs_{feature}, prepositions_{feature}$, $conjuctions_{feature}$ and $interjections_{feature}$. For the pair $(unidentified_1, unidentified_2)$ we cannot use Word2Vect, but instead we could use string comparisons (i.e. the standard strcmp function) to produce $unidentified_{feature}$.

At this point the problem of identifying if two questions has been reduced to a classification problem with 9 features and one target variable, and we could proceed to use the standard tools that we have seen on the class to perform this task (i.e. Logistic regression, naïve Bayes, support vector machines, neural networks, ...)

This approach can be modified heuristically in many ways to keep in count some grammatical considerations. For instance, the fact that the verb 'to be' appears in two questions is not a very strong evidence that both questions are the same. On the other hand, the appearance of a less common verb, say 'to mow', in both questions provides a better evidence that both questions might be the same.

A way of keeping this into consideration is by producing several buckets of verbs for each question, for instance we have 5 buckets $verbs_1^1, verbs_1^2, verbs_1^3, verbs_1^4, verbs_1^5$, where $verbs_1^1$ only contains verbs that appear very often on the training set, and $verbs_1^5$ contains the less common ones. After doing this with the other parts of the speech we will have $5 \cdot 8 = 40$ features plus a feature for the unidentified words. We would then proceed to study the problem as a classification, with the techniques mentioned before.

3.2 Neural Network approach

Among previous methods on neural network, the first step is usually transferring the words in a question into vectors using word embedding method. Word embedding is a general method used for similar algorithms that embeds words into a vector space with several hundred of dimensions. These vectors captures semantics and analogies of different words. In other words, the similarity between the words' definitions are denoted by the distance in vector the space. Currently, there are trained models including Googles WordVec, Stanford Universitys GloVe, Gensim and Deeplearning4j. Principal Component Analysis (PCA) and T-Distributed Stochastic Neighbour Embedding (t-SNE) are both used to reduce the dimensionality of word vector spaces and visualize word embeddings and clusters.

To judge whether two sentences (vector sets) are duplicates belongs to natural language sentence matching (NLSM) problem. For NLSM problem, there are two types of deep learning frameworks. One is based on the **Siamese**, the other is based on **matching-aggregation**. For Siamese framework, two sentences are encoded into two vectors individually and the matching decision is based on these two vectors. This method has smaller model and easier to train, but hasnt included the interaction between two sentences. For matching-aggregation framework, the two sentences are matched at the beginning, including word-by-word and/or phrase-by-sentence, and then the results are aggregated into vectors. This method has included the interactions between two sentences, but usually requires relatively larger model.

The first algorithm "Siamese"[2] is consist of two weight-sharing convolutional neural networks (CNNs). Each takes one input and projects it into lower-dimensions based on the CNN, where similar items are contracted and dissimilar ones are dispersed over the learned space. And there is a final layer to solve the sentence paring problem. The model is small and easy to train, and adaptive to various purposes. However, the algorithm does not incorporate interactions between sentences, and hence the matching-aggregation is developed.

For matching-aggregation framework, another possible approaching method is described in Wang's paper published recently.[1] Basically, they have introduced a bilateral multi-perspective matching (BiMPM) method with 5 levels. Word Representation is used to convert words to vectors. Context Representation Layer is used to encode contextual information. Matching Layer is used to compare the similarity of the two sentences. Here one time-step of one sentence is compared against all time-step of the other sentence. **"Aggregation Layer"** is used to convert matching vectors into fix-length vectors. **"Prediction Layer"** is used to evaluate the conditional probability.

References

[1] Z. Wang et al. Bilateral multi-perspective matching for natural language sentences.

[2] Eren Golge. Duplicate question detection with deep learning on quora dataset, 2017.

[3] Shubh24. Kaggle kernel: A beginner's guide, 2017.