Quora Question Pairs

Nikhilesh Puvvada-IMT2013031
Karthik Revanuru-IMT2013033
Surya Teja-IMT2013059

May 18, 2017

Contents

1 Abstract

This report details our attempt to apply Statistical Machine Learning and Deep Learning approaches to the problem of semantic matching on questions asked in Quora. Four models were used to predict if two sentences share the same intent. These four methods were detailed in the upcoming sections. An accuracy of 85 percent was achieved.

2 Introduction

Thousands of new questions are posted in Quora everyday. In order to build a high-quality knowledge base, it's important to ensure each unique question exists on Quora only once. Writers shouldn't have to write the same answer to multiple versions of the same question, and readers should be able to find a single canonical page with the question they're looking for.

For example, we'd consider questions like "What are the best ways to lose weight?", "How can a person reduce weight?", and "What are effective weight loss plans?" to be duplicate questions because they all have the same intent.

So, the goal of the problem is to detect duplicate questions. And it's very difficult to do this task manually since there are millions of questions. So we need effective approaches which does this job with as little human effort as possible.

Formally the duplicate detection problem can be defined as: Given a pair of questions q1 and q2, build a model that learns the function f(q1,q2)$\rightarrow$1/0, where 1 indicates that q1 and q2 have the same intent and 0 otherwise.

3 Data Set

[1] We are using the data set published by Quora on Kaggle. There are 149306 positive (duplicate) and 255045 negative (non-duplicate) instances. There is a class imbalance. Considering the nature of the problem it's reasonable to keep the same data bias with the ML model since negative instances are more expect-able in a real-life scenario. A small snapshot of the data set is shown below. We partitioned this data set into 90/10 train/test split.

	id	qid1	qid2	question1	question2	is_duplicate
0	0	1	2	What is the step by step guide to invest in sh...	What is the step by step guide to invest in sh...	0
1	1	3	4	What is the story of Kohinoor (Koh-i-Noor) Dia...	What would happen if the Indian government sto...	0
2	2	5	6	How can I increase the speed of my internet co...	How can Internet speed be increased by hacking...	0
3	3	7	8	Why am I mentally very lonely? How can I solve...	Find the remainder when [math]23^{24}[/math] i...	0
4	4	9	10	Which one dissolve in water quikly sugar, salt...	Which fish would survive in salt water?	0

id is a unique identification number for a question pair and qid1,qid2 are unique id's for question1,question2 of a particular question pair.

4 Data Pre-Processing

When we analyze the data, the shortest question is 1 character long (which is stupid and useless for the task) and the longest question is 1169 character (which is a long, complicated love affair question). We see that if any of the pairs is shorter than 10 characters, they do not make sense, So we remove such pairs. The average length is 59 and standard deviation is 32.
A small experiment is done to understand the statistics of the given dataset and the results are show below:

- Average number characters in question1: 59.57
- Minimum number of characters in question1: 1
- Maximum number of characters in question1: 623
- Average number characters in question2: 60.14
- Minimum number of characters in question2: 1
- Maximum number of characters in question2: 1169

Some labels are not true, especially for the duplicate ones. But we decided to continue with this because pruning is hard and requires manual effort.

5 Converting Word to Vectors

Encoding the meaning of a word into a vector is standard NLP process where these vectors can be used for any kind of training process on a corpus. The most important requirement while encoding the information of word into a vector is to preserve the meaning of a word.

An example of representation of a word considering a corpus which contains only 10 words as it's vocabulary can be [0 0 0 0 0 0 120 0 0 0]. Where the position with a non zero number is the word's position and number is the number of times this word occurs in the document.

Suppose you want find the relationship between the words "hate" and "despise". There will be no encoded relationship between the vectors of these words according to this representation.

Consider distributional similarity, which says two words are similar if they are used in similar circumstances. We need a vector representations which encodes this similarity measure.

Consider the following representations of words: 1. Skip gram model 2. Glove model

Consider the representation where Each word is a vector with a probability distribution over the vocabulary in corpus with the probabilities with which the words can occur in it's context. Context is defines as a fixed length of sentence before and after the word under consideration.

Say the corpus contains words I, love, hate, her, him. Then representation of the word is "I" can look like [0, 0.1, 0.3, 0.2, 0.4]. This means that the probability of the word "hate" being in the context(say defined with 2 words before and after the word) of "I" is 0.3.

5.1 Skip gram model

Task: predicting **softmax** probability of word a being in the context of word b

$$p(a/b) = \frac{\exp(u_a^T v_b)}{\sum_{w=1}^{V} \exp(u_w^T v_b)} \tag{1}$$

Where $p(a/b)$ *= softmax probability of word a being in the context of word b*
u_a *= word vector with a as context word*
v_b *= word vector with b as centre word*

Objective function is to minimize

$$J(\theta) = \frac{-1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j}/w_t) \tag{2}$$

Where $\theta =$ *parameters of model i.e word vectors*
T *= total number of words in the vocabulary*
m *= context parameter*
J *= loss function*

5.2 Glove Model

Let X *= the matrix of word-word co-occurrence counts*
X_{ij} *= number of times word j occurs in the context of word i*
$X_i = \sum_k X_{ik}$ *number of times a word appears in context of i*
$P_{ij} = P(j/i) = \frac{X_{ij}}{X_i}$ *probability that word j would appear in context of i*

Glove model is a weighted least squares regression model, where we have to minimize the following loss function

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \bar{w}_j + b_i + \bar{b}_j - \log X_{ij})^2 \tag{3}$$

Where V *= Size of the vocabulary*
f *= Weight function*
w_i, w_j *= context word and centre word*
$b_i, \bar{b}_j$ *= biases*
J *= function that needs to beminimized*

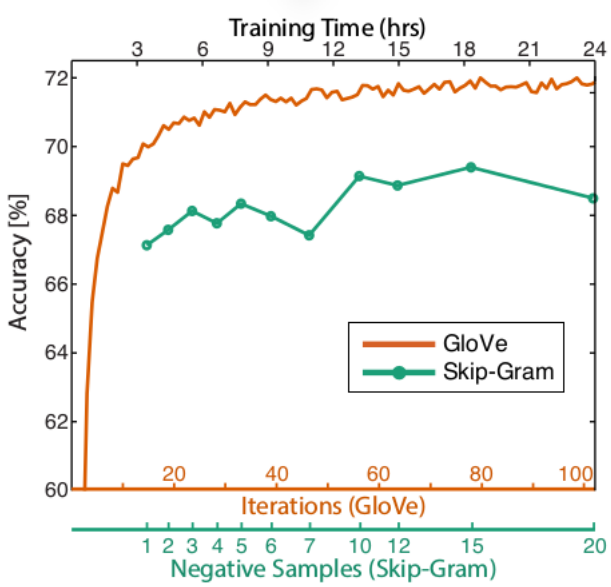
Properties of weight function f(x):
1. $limit_{x->0} \log^2 x is\ finite$
2. It should be non decreasing so that **rare co-occurences** are not overweighted
3. It should be relatively small for large values of x so that **frequent co-occurences** are not overweighted

The model which generates the vectors we are using uses the the following weight function:

$$f(x) = \begin{cases} (x/x_{max})^{\alpha} & x \leq x_{max} \\ 1 & otherwise \end{cases} \tag{4}$$

$$where : \alpha = 0.75$$
$$x_{max} = 100$$

Following is an experiment in the paper[4]



Here a training has been done for word analogy task. Glove has shown greater accuracy. Here the word vectors are 300 dimensional and corpus consisted of 400000 word vocabulary and the window size was taken to be 10.Therefore we are using glove model to convert the words in our corpus into vectors.

6 Experiments

We experimented with a variety of simple and deep learning to approach the problem. For the simple features approach, we used a mix of intuitive features like no-of-common words, length of the questions, difference in length and also some not so intuitive features like Partial ratio and token set ratio.

For deep learning, we used the concept of word2vec explained in the previous section to get vector representations from words and on top of that we have applied a simple Artificial Neural Network,LSTM and siamese architecture.

6.1 Approach1

In this approach, we extracted 8 basic features, 4 fuzzy features and passed these 12 features to the logistic regression model.

The basic features extracted are: Length of question1(character length including spaces), Length of question2(character length including spaces), Difference in two lengths, Character length of question1 without spaces, Character length of question2 without spaces, Number of words in question1, Number of words in question2, Number of common words in question1 and question2. These features can be extracted easily pandas' apply and lambda functions.

The four fuzzy features extracted are: Simple Ratio, Partial Ratio, Token sort ratio and Token set ratio.

Fuzzy String Matching, also called Approximate String Matching, is the process of finding strings that approximately match a given pattern. The closeness of a match is often measured in terms of edit distance(a.k.a Levenshtein distance), which is the number of primitive operations necessary to convert the string into an exact match. Primitive operations are usually: insertion (to insert a new character at a given position), deletion (to delete a particular character) and substitution (to replace a character with a new one). Four basic functions in fuzzy string matching are:

- Simple Ratio: It measures how exactly the same are two strings. Two examples are shown below:
  - a = "fuzzy wuzzy"
  - b = "fuzzy wuzzy!"
  - simple ratio of a,b is 96
  - a = "apple"
  - b = "fizi"
  - simple ratio of a,b is 0
- Partial Ratio: It checks how much of the shorter string is contained in the longer string
  - a = "Virginia"
  - b = "University Of Virgina"
  - Partial Ratio of a,b is 100
  - a = "fuzzy"
  - b = "wuzzy"
  - Partial Ratio of a,b is 80
- Token Sort Ratio: The token sort approach involves tokenizing the string in question, sorting the tokens alphabetically, and then joining them back into a string. For example: "new york mets vs atlanta braves" →"atlanta braves mets new vs york". We then compare the transformed strings with a simple ratio(). An example is shown below:
  - a = "Harry Potter"

- b = "Pottter, H."
- Token set ratio of a,b is 50

- Token Set Ratio: The token set approach is similar, but a little bit more flexible. Here, we tokenize both strings, but instead of immediately sorting and comparing, we split the tokens into two groups: intersection and remainder. We use those sets to build up a comparison string. Say t1,t2 are the two strings. Then they are first re-arranged as follows:
  t1 = [SORTED_INTERSECTION] + [SORTED_REST_OF_STRING1]
  t2 = [SORTED_INTERSECTION] + [SORTED_REST_OF_STRING2]
  and then compare each pair.

  The intuition here is that because the SORTED_INTERSECTION component is always exactly the same, the scores increase when (a) that makes up a larger percentage of the full string, and (b) the string remainders are more similar.

  An example is shown below:

  - a = "Harry Potter"
  - b = "Pottter, H."
  - Token set ratio of a,b is 86

fuzzywuzzy module in python is used produce the results for these four features.(for each pair of questions in the data)

These 12 features calculated for each data point (or) pair of questions are used to build a logistic regression model. This model built is used to classify a new test point as duplicate/non-duplicate pair.

With this approach we have got an accuracy of 66%

6.2 Approach 2

6.2.1 Question Embeddings

We have used Word2Vec to convert each question into a semantic vector then we have stacked a Siamese network[3] to detect if the pair is duplicate.

We have vector representations for each word in the question, but we still need to figure a way out to represent entire question into vector. One simple method is to take mean of all the word vectors in each question. Though this method is simple it works well in document classification so we have decided to use it for this problem too.

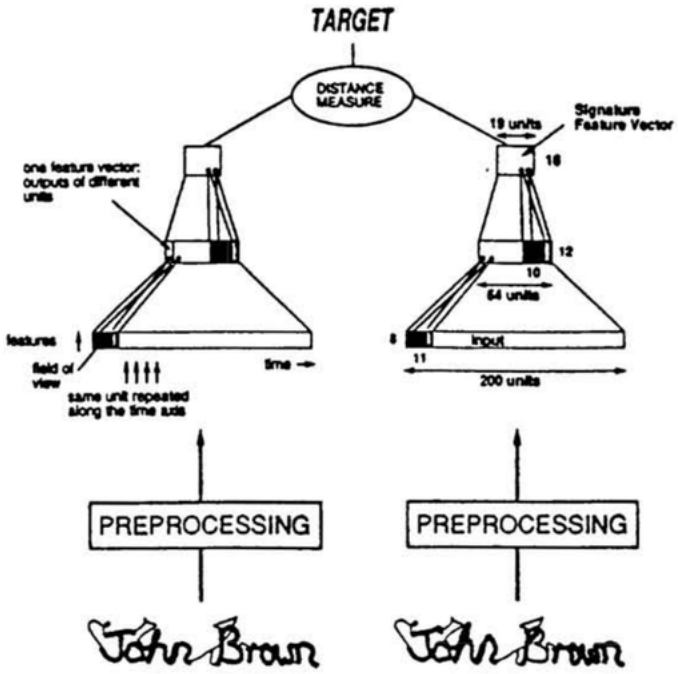
We can improve this method by taking weighted mean using Term Frequency–Inverse Document Frequency(TF-IDF) .We apply weighted average of word vectors by using these scores. It emphasizes importance of discriminating words and avoid useless, frequent words which are shared by many questions.

TF-IDF is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. It is often used as a weighting factor in information retrieval, text mining, and user modeling. The tf-idf value increases proportionally to the number of times a word appears in the document, but is often offset by the frequency of the word in the corpus, which helps to adjust for the fact that some words appear more frequently in general. TF-IDF is one of the most popular term-weighting schemes.

6.2.2 Siamese Network

Siamese neural network is a class of neural network architectures that contain two or more identical subnetworks. Identical here means they have the same configuration with the same parameters and weights. Parameter updating is mirrored across both subnetworks.

Siamese neural networks are popular among tasks that involve finding similarity or a relationship between two comparable things. One example is signature verification, where we figure out whether two signatures are from the same person. Generally, in such tasks, two identical subnetworks are used to process the two inputs, and another module will take their outputs and produce the final output. The picture below is from Bromley et al (1993)[1]. They proposed a Siamese architecture for the signature verification task.



Each subnetwork essentially produces a representation of its input in this case "Signature Feature Vector". If your inputs are of the same kind, like matching two signatures or matching two pictures, it makes sense to use similar model to process similar inputs. Siamese layer tries to push the network to learn features common for the same classes and differentiating for different classes. This way you have representation vectors with the same semantics, making them easier to compare.

We decided to use siamese network since checking if you signatures are similar is similar to finding if two sentences are similar. Given two sentences we feed them into network and compute corresponding feature vectors. The final layer calculates pair-wise distance between computed features and final loss layer considers whether these two images are from the same class or not.

6.3 Approach 3

The approach we used is the following:

- Use GloVe word embeddings(explained in next section) to create word embeddings (or) vector representation for each word in both questions.
- Form vector representations for each question in the pair from vector representations obtained for each word
- Pass the concatenated vector representation(of both the questions) to a neural network model(the model architecture used will be explained in below sections) which is trained to distinguish duplicates from non-duplicate pairs.

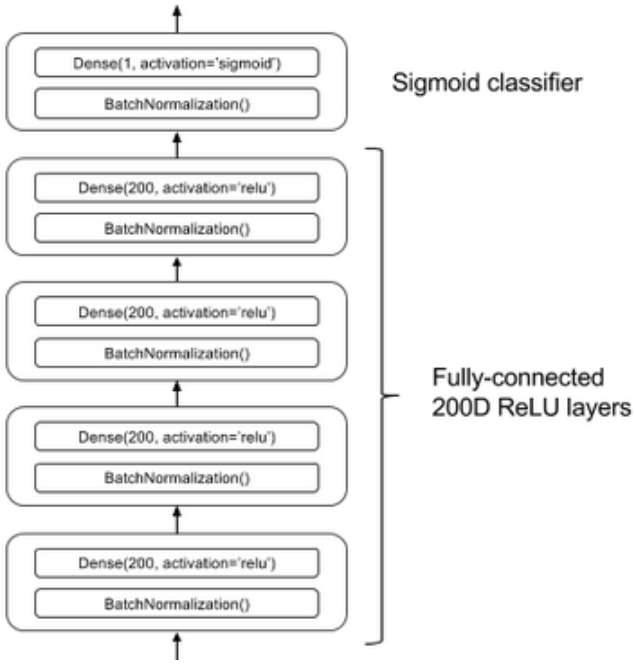
6.3.1 Question Embeddings

Once we get vector representations for each word, the next task is to get vector representations for each question in the pair. This is done by applying a max operator i.e, if the vectors for the n words in question1 are $v_1, v_2, \ldots, v_n \in R^d$(in our case d=300), then you compute $max(v_1, \ldots, v_n)$. Here we're taking the coordinate-wise maximum, i.e., the maximum is a vector u such that $u_i = min(v_i^1, \ldots, v_i^n)$. Similar is done for other question in the pair.

We have tried even mean and sum operators, but we have observed that max gives better accuracy than the others.

6.3.2 Neural Network Model

We have used a model architecture similar to Stanford Natural Language Inference(SNLI). It consists of a series of fully connected 200D layers followed by a final Sigmoid classification layer. The model architecture is shown below:



The vector representations of both the questions are merged and passed as inputs to this network and the model is trained to output a 1/0 depending on whether the concatenated vector representation corresponds to duplicate question pair/Non duplicate question pair.

This model achieves an accuracy of 82.9% when tested against 40,435 samples which is far better compared to using a Logistic regression model(in previous section) which gives an accuracy of 66%.

6.4 Approach 4

We have replaced the dense layers in the previous approach with LSTM the model is trained to output a 1/0 depending on whether the concatenated vector representation corresponds to duplicate question pair/Non duplicate question pair.

This model achieves an accuracy of 85% when tested against 40,435 samples which is better compared to previous three approaches.

7 Results

These are the best results obtained with varying models.

Approach	Accuracy
Feature Engineering + Logistic Regression	66
TD-IDF + Siamese	79
SNLI	82.9
LSTM	85

Feature engineering with logistic regression gives less accuracy compared to neural network models and this implies that features calculated are not good enough that can be used to build a classification model where as a neural network model has implicit nature of calculating the features which are good for classification.

LSTM gives a better accuracy compared to other two neural network models. This is because a Recurrent Neural Network model captures dependencies unlike the normal Neural Network models.

8 Conclusion and Future work

Here we tried to present a solution to this unique problem by composing different aspects of deep learning. Results are not perfect and akin to different optimizations. However, it is just a small try to see the power of deep learning in this domain.

Going further we can experiment with BRNN's because this network maintains two hidden layers, one for the left-to-right propagation and another for the right-to-left propagation.

References

[1] Quora question-pairs data. `https://www.kaggle.com/c/quora-question-pairs/data`. Accessed: 2017-04-04.

[2] Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

[3] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744, 1994.

[4] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.