Quora Question Pairs Classification

Junhao Lin, Xianrui She

I. Abstract

In this project, we explore the power of NLP models to detect the difference between two question-a-question pairs. The main objective is to classify whether two questions are considered the same regarding content and intent. Hence, we select two supervised machine learning models, logistic regression and recurrent neural network(RNN), to achieve this goal. After running both models, we obtain an accuracy of 0.69 for logistic regression and 0.72 for the RNN on the test set. While we think the results are decent, they are primarily limited by hardware and can be much improved with more tuning and better machines.

II. Introduction

So far, we have been dealing with NLP problems with single text inputs for quite much. Hence, it immediately becomes interesting when encountering the problem with question pairs. Unlike a single input, which can be easily tokenized and transformed into a bag of words, a pair input would require some extra thinking. Two ways of dealing with this are to mesh the pair into a single input or to treat them like two different problems and combine them somehow. Both methods are illustrated in the models we use.

For the first approach, we stack two input bag-of-words matrices horizontally to form a new input matrix with the same number of rows but double the number of columns. In other words, each token in the corpus corresponds to two columns, one for the count in the first question and the other for the second question. In theory, combining the inputs needs to make more sense since two independent questions do not form a meaningful singular question. Still, in practice, it works well with bag-of-words for logistic regression, as is demonstrated later.

As for the second approach, we fit both questions to two identical models and then combine the results with building a new model resembling the ensemble idea. We apply this to the RNN model.

III. Dataset

Our data is provided by a modeling challenge from Kaggle. There are 400K records with three relevant features: the first question, the second question, and the binary label indicating if the questions are considered the “same.” While the definition of “same” is not clearly defined, it is embedded in the data labels and can be easily understood given the inputs. Here are two example pairs representing the two classes:

How do I read and find my YouTube comments?	Duplicate
How can I see all my Youtube comments?	
What is web application?	Not duplicate
What is the web application framework?	

It is worth noting that the pairs considered “not duplicate” can be hard to detect even by human eyes, for they can differ by only a tiny detail. In addition to the content of the questions, there is also an imbalance in class distribution. The number of non-duplicate pairs(250K) outweighs the duplicate ones(150K), so we sample non-duplicate pairs for a better class balance.

IV. Modeling

A. Logistic Regression

The first model we use is logistic regression. We use the SGDClassfier with log loss from the scikit-learn library for implementation. Using the stacking method introduced above, we can combine two questions into a single bag-of-words input matrix. With it, we build our baseline model with stochastic gradient descent. We can only use 20% of the original data to keep the pre-processing and training within a reasonable time. Then, we experiment with several techniques to improve the baseline model.

The first attempt is to use grid-search on a set of parameters. The parameters we choose to experiment with are the penalty term, learning rate, and tolerance. These parameters affect the training the most, so we give each parameter three values to choose from.

The second attempt is to try different determination thresholds. Since the logistic regression outputs probabilities for each class, we can use different thresholds to determine above what value should be considered positive predictions and vice versa.

After the above experiments, we developed an improved model version and used it as the final result for logistic regression.

B. RNN with GLoVe embedding and LSTM

The second model we use is LSTM RNN with GLoVe embedding. Using GLoVe, we can obtain vector representations for our dataset by mapping the words into meaningful space where the distance between the words is related to semantic similarity (Jeffrey Pennington, 2014).

After the embedding, we utilize the LSTM RNN model and train it with our dataset. The LSTM networks, or Long Short Term Memory networks, a special kind of RNN, are instrumental in NLP learning compared to other RNNs. We choose this particular RNN

because it helps with the Gradient Vanishing Problem. Vanilla RNNS are incapable of handling long-range dependencies. Long-range dependencies mean that if we are trying to generate a word based on previous inputs, then if the gap between the word to be generated and the previous input considered is small, RNNs will be able to predict the next word perfectly, but if the gap is large, RNNs fail to perform.

Unlike writing tokenization and normalization methods for our logistic regression model, we use the Keras tokenization library as a contrast to see if this can increase model performance. After tokenizing by the Keras library, we extract columns “question_1” and “question_2” and transform them into sequences, i.e., in the form of numbers since the machine can only process numbers, not texts. We define the maximum length of each question, and the questions which contain less than the required length are padded with zeros to make the length of the sentence equal to the mentioned length. For example:

maxlen = 5

sent1 = ['I', 'love','apples']

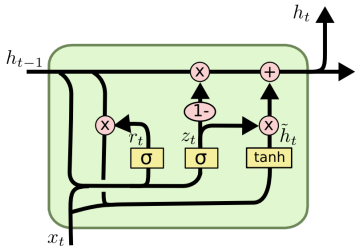
Post-sequencing: sent1 = [1,2,3]

Post-padding: sent1 = [1,2,3,0,0]

Next, we train our LSTM model with our dataset. The LSTM model operates using three gates: Input gate, Forget gate, and Output gate. The first step in our LSTM model is to decide what information we will throw away from the cell state. This decision is made by a sigmoid layer called the forget gate layer. It looks at ht-1 and xt and outputs a number between 0 and 1 for each number in the cell state Ct−1. One represents "completely keep this," while 0 means "completely get rid of this." The next step is to decide what new information we will store in the cell state. Here's our method:

1. First, a sigmoid layer, the input gate layer, decides which values we'll update.
2. Next, a tanh layer creates a vector of new candidate values, C~t, that could be added to the state.
3. Then, We'll combine these two to update the state.
4. Finally, we need to decide what we're going to output.

First, we run a sigmoid layer which decides what parts of the cell state we output. Then, we put the cell state through tanh (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate so that we only output the parts we decided to. (Christopher, 2015)



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

How LSTM RNN works (Christopher, 2015)

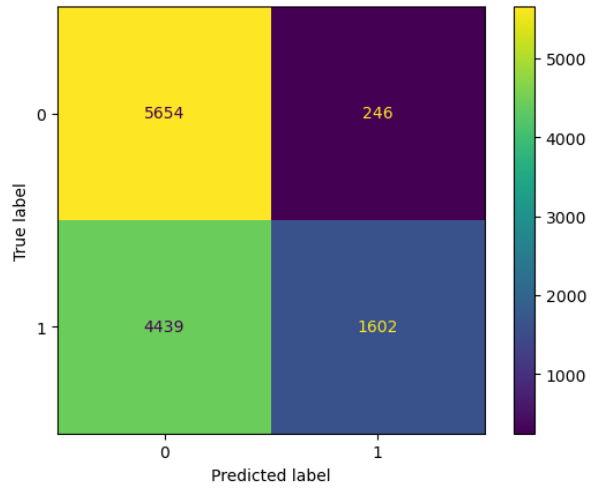
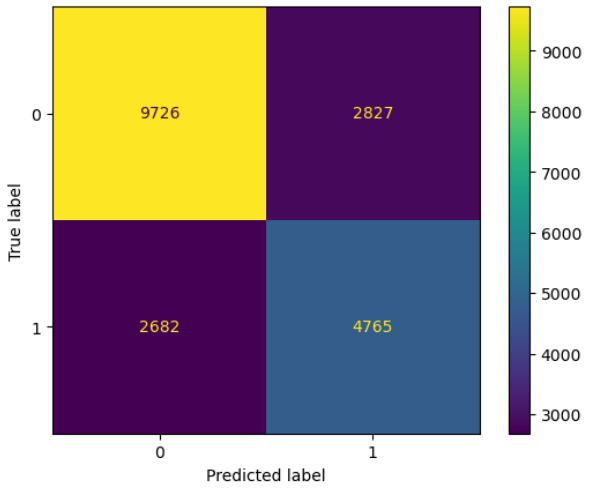
Eventually, we evaluated our model against data we randomly sampled from the training set (we did not use the entire training data as it is too large and time-consuming). The performance of LSTM RNN model is greater than our previous Logistic Regression model, and provides us with greater insight into RNN application on NLP problems.

V. Results & Conclusions

The accuracy we gain from both models are as follows:

Baseline Logistic Regression	66.99%
Gridsearch Optimized Logistic Regression	69.36%
LSMT RNN with GloVe embedding	72.45%

Confusion matrices for both model:

Logistic Regression	LSTM RNN with GloVe embedding
	

Training process for LSTM model:

Epoch	Loss	Accuracy
1	0.6532	0.6288
2	0.6182	0.6314
3	0.5825	0.6886
4	0.5548	0.7153

5	0.5218	0.7412
6	0.4945	0.7612
7	0.4619	0.7828
8	0.4409	0.7949
9	0.4107	0.8116
10	0.3849	0.8264

Our experiments with different models prove that the difference in performance is based on model choice, hyper-parameter tuning, and even hardware performances.

Surprisingly, stacking two questions as a single input works well with logistic regression. The model can achieve nearly 0.7 accuracies. With a powerful-enough machine that can handle more data, we expect the performance to be even higher.

On the other hand, the LSTM model works much better regarding overall accuracy and the false negative/positive rates. Compared to the logistic regression, the LSTM model reaches 0.82 accuracies during training, only to fall to 0.72 in the test set. This unexpected drop from training to test is likely due to the train-test split that slices a particular type of question into the same collection. The logistic regression also predicts more false negatives than false positives; the LSTM model makes it more balanced. It is likely due to the preservation of word order hence grammars in the questions, proving that keeping long-term relationships helps with the performance.

VI. Future plan

This is not our ideal model experiment because we notice another RNN variation, the Siamese Network, a specialized network that compares two inputs. The professor recommended an article to us, a tutorial for image comparison implemented using Siamese Network. However, due to system environment conflict and lack of knowledge, we failed to adapt a Siamese Network model for the NLP problem. If feasible, we will try to investigate and implement a functional Siamese Network model for this problem in the future.

Work Cited

GloVe: Global Vectors for Word Representation, Jeffrey Pennington and Richard Socher and Christopher D. Manning, 2014. Url: http://www.aclweb.org/anthology/D14-1162

Understanding the LSTM Networks, Christopher Olah, 2015. Url: Understanding LSTM Networks -- colah's blog