Duplicate Question Detection

Darpan Dodiya dpdodiya@ncsu.edu
Mohit Vinchoo mvincho@ncsu.edu
Shantanu Sharma ssharm34@ncsu.edu
Shrijeet Joshi sjoshi22@ncsu.edu

Abstract

Every Q&A website faces the problem of duplicate questions. This is especially true when the number of questions increase. This project concentrates on identifying a pair of questions as duplicate by applying various NLP classification techniques. Multiple approaches have been explored and we have tried an iterative approach to build a better model. We start from a baseline measure of Cosine Similarity between two questions and then improve by various NLP techniques and semantic similarity using WordNet. We also experimented with other classification algorithms such as SVM (Support Vector Machine) and deep learning using Siamese Manhattan LSTM Recurrent Neural Network. Classifiers were evaluated on various metrics and F1 measure was selected to be the best measure.

1 Introduction

On every Q&A platform, multiple questions with the same intent can cause seekers to spend extra time finding the best answer to their question, and make writers feel they need to answer multiple versions of the same question. In this project we try to experiment with various techniques that help in automated tagging to duplicates with high accuracy. This project to detect duplicate questions can broadly be classified into 4 main components: data pre-processing, feature engineering, building, training and tuning models and performance evaluation.

1.1 Problem

Given a pair of questions, identify whether or not they have the same intent. Manual identification of duplicate question pairs may work for a small number of questions, however as the complexity and scale increases, this becomes practically impossible. Identifying duplicate questions will make it easier to find high quality answers to questions resulting in an improved user experience. This will enable higher user satisfaction and an improved engagement. Another motivation is that this will reduce the role of admins and moderators on the Q&A platform.

This project will use a data-set provided by online Q&A platform Quora to build a classification model to identify duplicate questions. We evaluate different techniques to identify duplication in questions at scale.

1.2 Literature Survey

Since the focus of the project was to target online forums, detecting Semantically Equivalent Questions in Online User Forums (2) gave a good head start. Post pre-processing, for representation of the questions GloVe: Global Vectors for Word Representation (3) served as a good reference point for representation of words in a sentences as a vector. The ideas proposed in From Word Embeddings To Document Distances (4) helped on leveraging distance measures to generate a classification models. (5) Gave a Siamese adaptation of Long Short-Term Memory(LSTM) to assess the semantic similarity between sentences.

2 Method

The data-set of question-pairs is given as tuples of questions and labels. The value of label is 1 for similar questions and 0 otherwise. In order to feed data to the classifier we needed to represent the data into format that the classifier can understand. We have tried an iterative approach to improve accuracy while adding features and semantic understanding. Details of each method are given below starting from baseline cosine similarity

2.1 Baseline : Cosine Similarity

The first, naive approach towards identifying question pairs is to split each sentence into words and then find cosine distance between the pair. This was done to explore if cosine similarity is a good measure of duplicity.The cosine similarity can be calculated by:

$$cos(\vec{x}, \vec{y}) = \frac{\vec{x} \cdot \vec{y}}{\parallel \vec{x} \parallel \parallel \vec{y} \parallel} \tag{1}$$

where x and y are vectors of words of the question pair. This is a naive approach without any pre-processing or cleaning and is used as a baseline.

2.2 KNN

As an extension of the baseline, we compute cosine distance between the two questions after data has undergone NLP cleaning tasks and run KNN. In this experiment we used CountVectorizer and ngrams (9) of the questions. Ngrams are useful as they help us understand positioning of words for example two words "not" and "good" when considered together give an entire new meaning. An important thing to note is that the classifier takes a tuple of (cosine similarity, label) to train. i.e. we converted our questions data into a single feature. Experiments with values of k showed that we get best results with the hyper-parameter k=3.

2.3 Feature Engineering

Feature engineering is the process of using domain knowledge of the data to create features that make machine learning algorithms work. The purpose of a feature, other than being an attribute, would be much easier to understand in the context of a problem. Our original dataset had only two features, text description of Question1 and text description of Question 2. We tried to create some features from the original dataset. One of them was to count number of common nouns and number of common verbs between two questions. This simple feature resulted in over 4% increase in accuracy.

2.4 Semantic Similarity WordNet

WordNet is a huge library of synsets for almost all words in the English dictionary. The synsets for each word describe its meaning, part of speeches, and synonyms/antonyms. The synonyms help in identifying the semantic meaning of the sentence, when all words are taken together.

In our earlier approach of finding cosine distance between questions, there was lack of semantic understanding – There might be two questions with a high percentage of common words, but different meanings. Approach described in paper (6) eliminates this by considering semantic meaning of words. This approach operates by representing each sentence as a binary vector (with elements equal to 1 if a word is present and 0 otherwise), a and b. The similarity between these sentences can be computed using the following formula:

$$sim(\vec{a}, \vec{b}) = \frac{\vec{a}\mathbf{W}\vec{b^T}}{|\vec{a}||\vec{b}|} \tag{2}$$

where W is a semantic similarity matrix containing information about the similarity of word pairs. This approach resulted in an improvement over the naive cosine similarity and 3-NN.

2.5 WMD distance with Word2Vec

WMD is a method that allows us to assess the "distance" between two sentences in a meaningful way, even when they have no words in common. The WMD distance is defined as the minimum amount of distance that the embedded words of one document need to "travel" to reach the embedded words of another document.

2.6 Siamese Manhattan LSTM

Siamese networks are a type of network that have two or more identical sub-networks in them. They have good performance when we need to check the similarity of 2 input items. This model converts the given input sentences to its vector representation. This is used in the final hidden layer. We use the similarity between these representations as a predictor to determine if the input sentences are similar [5]. Here we have used Manhattan distance to check the similarity of the sentences (MaLSTM).
We have used Google's Word2Vec Embedding to get the words for the deep learning network. In the embedding layer, the words from the above vectors are selected and looked up into the corresponding embedding. This is the main layout of this model.

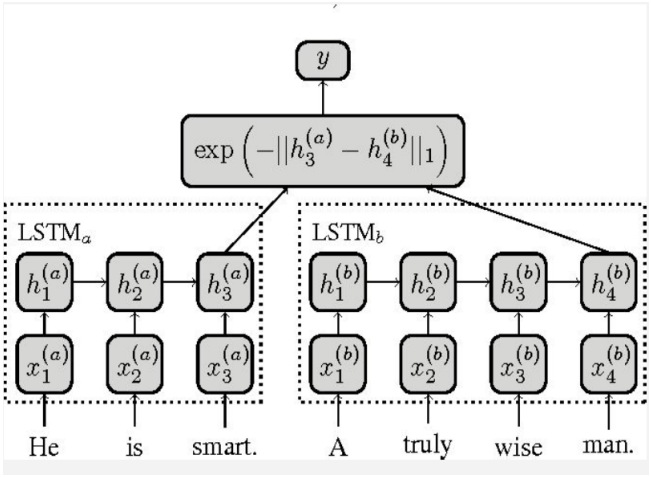

Figure 1: Siamese LSTM Architecture

3 Experiment

3.1 Hypothesis

Hypothesis 1: Duplicate questions have more cosine similarity than non-duplicate questions: The median of cosine similarity scores of non-duplicate questions is .48 where as the median duplicate questions is .69. This shows that higher the cosine similarity is present in questions that are similar.

Hypothesis 2: Removal of stop words with negative connotation: Removal of all stop words also removed words such has "not" "no" etc. these words make a lot of difference in our problem since "this is spring" "this is not spring" will be same vectors if we blindly remove all stop words including "not" and hence labeled incorrectly.

Hypothesis 3: Usage of N-grams leads to better results: Simple vectorization does not take into account the order of words but ordering of word plays an important role in understanding the question, hence using Ngrams should improve the accuracy our predictions.If we use observations of hypothesis 2 and add 2Grams while vectorizing the questions we found that test accuracy increases from 61% to 64%

Hypothesis 4: Semantic meaning gives a better prediction than only syntactic comparison: Cosine similarity only considers the word vectors between two questions and computes a simple cosine distance between the 2 vectors it does not do any semantic analysis. If the algorithm tries to understand

the underlying semantic structure of the question we should be able to get better results. We achieved this by using WordNet and later by using Google word embedding while using LSTM. The results were better, we were able to increase test accuracy to 73% with WordNet and 82% with MaLSTM network which uses Google word embedding and the Adadelta gradient descent optimizer.

3.2 Python environment and packages

We have used Python and other supporting packages for our analysis. Notable components of the environment have been described below:

Table 1: Environment details

Package/Software	Version
Python	v3.7.2
Jupyter Notebook	v5.7.6
NLTK	v3.4
Numpy	v1.15.1
Pandas	v0.23.4
Scikit-learn	v0.19.2
matplotlib	v2.2.3
TensorFlow	v1.5

3.3 Dataset

The data of duplicate questions pairs has been provided by Quora on Kaggle platform. (1) The ground truth of the dataset has been supplied by human experts on Quora. Since Human labeling is not accurate, the labels are not believed to be 100% accurate. The data is not symmetric, it is skewed towards non-duplicate questions as it takes 63.08% of the class labels. There are very few pairs with no common words.

Table 2: Data fields

Attribute	Description	Example
id	the id of a training set question pair	1
qid1, qid2	unique ids of each question	1, 2
question1, question2	the full text of each question	"How are you?", "Are you good?"
is_duplicate	the target variable, 1 if duplicate, 0 otherwise	0

Table 3: Data summary

Type	Record Count
Total question pairs	404290
Duplicate question pairs	149263 (36.91%)
Non duplicate question pairs	255027 (63.08%)
Missing values	3
Duplicate rows	0
Mean question length	59
Median question length	51

3.4 Experiment design

We have implemented a pipeline of operations in order to perform experiments for a classifier. The pipeline was more or less same but few subtle changes were done in order to improve the accuracy in each experiment. The general flow and steps are outlined in the Figure 1.

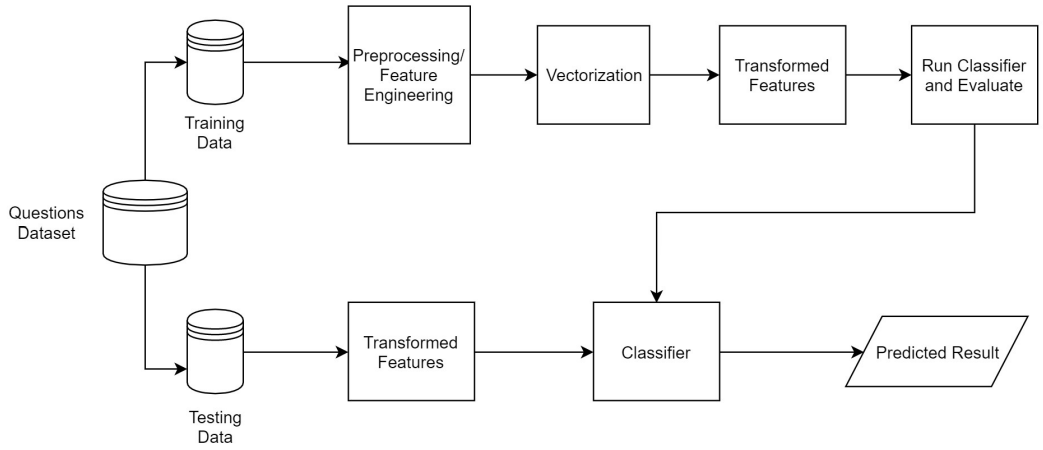

Figure 2: Flow of execution

3.4.1 Pre-processing

Every experiment that we have performed involved pre-processing of data. This step is applicable across all the experiments. The data pre-processing step started with eliminating records with duplicates and missing value. Next we eliminated non-alphabetical characters and then tokenized the questions. Our analysis showed that stop words in the questions did not add value therefore we also eliminated stop words.
After this we performed the NLP tasks of stemming and lemmetization. Once we were done with this we converted the questions to vectors using the Count Vectorizer. While performing KNN we used n-grams to perform count vectorization.

3.4.2 Baseline

As described in *Hypothesis 1*, we believed that a measure as simple as cosine similarity between two questions would serve as good indicator of baseline performance of the models. Using CountVectorizer of scikit-learn, we vectorized the sentences and calculated similarity.

3.4.3 Improving baseline

As we proved that a simple measure as cosine similarity could work well, we tried to improve baseline. The experiments included removing stop words, removing non-alpha numeric characters, removal of stop words with negative connotation. (*Hypothesis 2*). We also tokenized questions. NLTK package was extensively used to perform these experiments. This experiment improved baseline by 4-5%.

3.4.4 Semantic meanings, N-grams, KNN, Feature engineering

Once we have baseline ready and also pre-processed data, we experimented with different approaches such as KNN, WordNet (*Hypothesis 4*), N-grams (*Hypothesis 3*) to see which could give best results. We also tried to add give additional features to the classifiers. One of the features was to add the count of common verbs and common nouns between two questions.

3.4.5 Long Short-Term Memory Neural Network

In addition, artificial recurrent neural networks were explored in further experiments, such as Siamese Manhattan LSTM Recurrent Neural Network.

3.4.5.1 Siamese LSTM Architecture

We have chosen this model as it has been proved to perform exceptionally well in similarity tasks, to compare semantic meaning and intent between text (5). Both sub networks take a question as an input. These questions are converted to their vector format and inputted further to an embedding layer. Here, word embedding is created for each word in the question and such a matrix is further inputted to the two sub networks. These matrices are then propagated through the hidden units in the LSTM layer and a vector of outputs in generated on either side. We further calculate the Manhattan distance between these two vectors and take the exponent of its negative value to get a nice value between 0 and 1. This is the correlation/similarity between the two questions and this value is further used in the SGD MSE loss function to improve accuracy over multiple epochs.

Below are the best hyper-parameters chosen by us to train our model.

Table 4: Hyperparameters

Hyperparameter	**Value**
Number of Hidden Units	50
Number of epochs	10
Optimizer	Adadelta
Loss	mean squared error
Gradient clipping norm	1.25

3.4.5.2 Model:

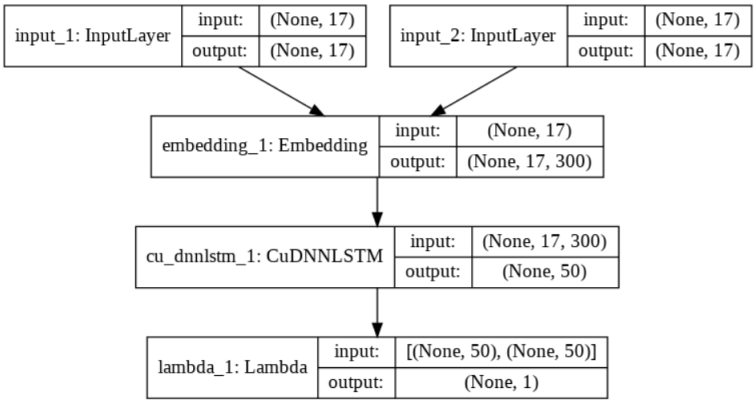

Figure 3: LSTM Model

As you can see there are 2 input layers. 17 is the number of words in the longest question in the available data set. For questions with words less than 17 we have added a padding sequence of 0s at the beginning. Next is the Embedding layer. Here we have used the Google's trained Word2Vec Model. The vector length is 300 features (7). This is the reason for the embedding layer to have a dimension of 17 x 300. This new embedding layer is then feed-ed to the LSTM Model. Our model had 50 hidden units and hence the output dimension is 50.In the last Lambda layer, the Manhattan distance is calculated and it gives a 0-1 output which determines if the questions are similar or not.

3.4.5.3 Results for Manhattan LSTM:

We ran our code for 25 epochs and we saw that the validation accuracy did not increase after the 9th epoch. Hence these are the graphs generated for 10 epochs.

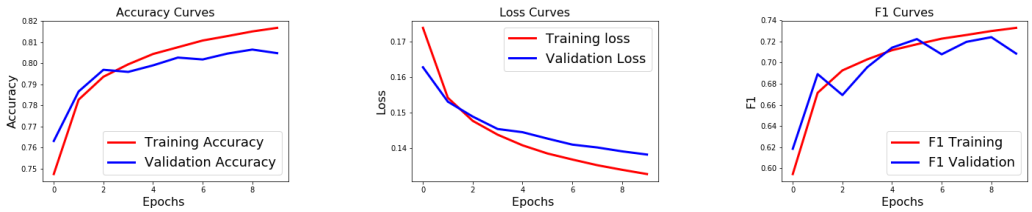

Figure 4: Accuracy, Loss and F1 scores

3.4.5.4 Future Scope for Manhattan LSTM

To further improve performance of this model, we'd like to use our own domain specific embeddings to improve the accuracy. We also want to try a few more Gradient Descent Optimizers to see if we get better results. Lastly, we'd like to use more data for training.

4 Results

4.1 Performance evaluation

The data set under consideration has binary class labels, with the records belonging to either positive or negative class. The instances of questions having duplicates are assigned positive class while others are assigned negative class implying they are nonduplicates.

There are various metrics that can be derived from the confusion matrix obtained as a result of testing the classifier on each of the projects. Different measures of our model evaluation are summarized below:

- Accuracy: Defined as the ratio of correctly predicted observation to the total observations. Higher accuracy doesn't mean that the model is the best. It is most useful when the dataset is symmetric.
- Precision: Defined as the ratio of correctly predicted positive observations to the total predicted positive observations. High precision is preferred.
- Recall: Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. High recall is preferred.
- F1 Score: F1 Score is the weighted average of Precision and Recall.

Since F1 Score takes both false positives and false negatives into account and hence we selected it as evaluation metric.

4.2 Experiment results

Below are the results from our experiments. The models were evaluated based on the evaluation criteria set in previous section. Precision and recall results have been omitted as F1 score capture both of the measures. Please refer to Table 5.

Table 5: Experiment results

Experiment	**Accuracy**	**F1 Score**
Cosine Similarity	63%	0.59
3NN	63%	0.46
3NN + Feature Engineering	67%	0.55
SVM	65%	0.50
Semantic Similarity WordNet	72%	0.51
Word2Vec + WMD	67%	0.61
MaLSTM Neural Network	82%	0.72

We started with the baseline classifier model of Cosine Similarity. It was naive implementation without any pre-processing or data cleaning involved. To our surprise, it reached accuracy of 63%. These results aligned with our *Hypothesis 1*. While the number 63% seem relatively high, we soon realized that accuracy wasn't a good metric for our case as the data was asymmetric. We chose F1 score as our target metric.

Next set of results were of KNN and SVM classifiers. 3NN classifier had the best F1 score and SVM reached score of 0.50. These results weren't as expected. We assumed that since we had cleaned and pre-processed the data, we should get improved results. It wasn't the case and the results have actually regressed proving our *Hypothesis 2* false. We do believe that removing stop words and cleaning data overall will have positive impact on the results, but in our specific cases it didn't work out.

Feature engineering did work out well. Simple 3NN had F1 score of 0.46 whereas 3NN + Feature Engineering lead it to 0.55. Alongside classical text based classifiers, we also tried out several approaches that from relevant prior work. One of them was finding Semantic Similarity WordNet (2), which resulted in good accuracy at 72% but with bad F1 score. Further, we experimented with Word Mover's Distance (WMD) approach to find measurement between two documents. (4). It didn't give good results in our use case as two questions acted as two documents resulting in sparse training points.

At the end, we had decent results with Manhattan LSTM. After 9 epochs, it reached accuracy of 82% with F1 score of 0.72, the highest of all experiments. As per our referenced work (5), we expect it to work little better than our current result. We have outlined several approaches we'd like to try in section 3.4.5.5 to improve our current result.

In essence, even the simplest Cosine Similarity classifier worked surprisingly well among all text based classifiers. MaLSTM based deep neural network model was the best overall.

5 Conclusion

Accuracy can be misleading

At times it may be desirable to select a model with a lower accuracy because better predictive power on the problem. For example, in a problem where there is a class imbalance, a model can just predict the value of the majority class for all predictions and achieve a high classification accuracy. In our dataset if a model simply predicts all pairs to be nonduplicates it will still achieve around 60% accuracy, which is misleading.

Even simple feature engineering can impact the results

We derived a simple feature of counting common nouns and common verbs between two questions. Feeding this new feature to classifiers improved accuracy by over 4%. In contrast, we tried many different ways to pre-process and clean the data - none of which had any substantial impact.

Deep Neural Networks perform better than text based classifiers

We experimented with many classical text based classifiers, but none of them performed well even after many rounds of fine tuning. Since the LSTM works with pre-trained embeddings on a very large corpus, training the model with such network results in deriving contextual relations between questions resulting in better classification.

Platforms like Google Colab can expedite experiment duration

During initial stages of this project, we often hit with resource limitation of available hardware. Long processing times made the project seem tiresome. Then we moved to Google Colab environment to run our experiments and it dramatically reduced experiment duration. Epoch that took 4 hours to run on local machine was completing in 20 secs on Google Colab.

References

[1] https://www.kaggle.com/c/quora-question-pairs/data

[2] Bogdanova, Dasha et al. "Detecting Semantically Equivalent Questions in Online User Forums." CoNLL (2015).
https://aclweb.org/anthology/K15-1013

[3] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.
https://nlp.stanford.edu/pubs/glove.pdf

[4] Matt J. Kusner , Yu Sun , Nicholas I. Kolkin , Kilian Q. Weinberger, From word embeddings to document distances, Proceedings of the 32nd International Conference on International Conference on Machine Learning, July 06-11, 2015, Lille, France
http://proceedings.mlr.press/v37/kusnerb15.pdf

[5] Jonas Mueller. Aditya Thyagarajan, From Siamese Recurrent Architectures for Learning Sentence Similarity, Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16).
http://www.mit.edu/~jonasm/info/MuellerThyagarajan_AAAI16.pdf

[6] Fernando, Samuel Stevenson, Mark. (2009). A Semantic Similarity Approach to Paraphrase Detection. Proceedings of the 11th Annual Research Colloquium of the UK Special Interest Group for Computational Linguistics.

[7] http://mccormickml.com/2016/04/12/googles-pretrained-word2vec-model-in-python/

[8] https://blog.exsilio.com/all/accuracy-precision-recall-f1\
-score-interpretation-of-performance-measures/

[9] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

GitHub Link: https://github.ncsu.edu/dpdodiya/p21_csc522