EECS 349 - Project Report

Quora Question Pairs Similarity

Omkar Satpute and Aditya Kumar

1 Task

Our task is to predict the similarity between questions posted on Quora to find duplicate questions. The similarity is found for pairs of questions based on the features extracted from the text of the questions such as number of common words, ratio of common words and so on.

We feel this task is important because lumping duplicate questions together can save users a lot of time and even assist them in finding answers they might not have gotten to in case they didnt find and go through all the duplicate questions. We feel this might really benefit Piazza as well, since duplicate questions are posted there often.

2 Data

The data set which we have utilized is provided by Quora.There are 404301 pairs of questions in the data set. That means there are essentially 404301 examples. The attributes provided are:

- id: Example row id
- qid1: The unique ID of question1 in the pair
- qid2: The unique ID of question2 in the pair
- question1: Text of question 1 in the pair.
- question2: Text of question 2 in the pair.
- is_duplicate: Denotes whether the questions are duplicate. (Trying to predict this)

2.1 Splitting the data

Initially we divided the data 90:10, keeping the 10 percent for testing at the end. This was working when we had only 7 features, however, once we added tf-idf weighted word vectors, the number of features came to a total of 775. This increased the size and complexity of our data too much and our machines were not able to handle this amount of data. So we used a subset of 200,000 examples at the end, splitting it into a 70:30 split for training and testing, using part of the training set for validation as well.

2.2 Preprocessing of data

We have performed the following pre-processing tasks on this data set:

- Checking of null values in any of the attribute and replacing it with an empty string.
- Performing a check if there are any rows with the same pair of questions.
- Removing Punctuations.
- Removing Stopwords.
- Expanding contractions.
- Stemming

2.3 Features extracted

Initially we started off by extracting the following features that were intuitive to us:

- word_Common: Number of common unique words in Question 1 and Question 2.
- word_share: word_Common / word_Total

We felt that only two features might not be enough to build a good classifier, so we used a python library called FuzzyWuzzy for extracting the following features:

- fuzz_ratio : Simple measure calculated using edit distance
- fuzz_partial_ratio: Considers only shorter string for matching to get around inconsistent sub strings.
- token_sort_ratio : Sorts tokens alphabetically and then compares the strings.
- token_set_ratio : Tokens are split into two groups- intersection and remainder and these are used to build up a comparison string.
- longest_substr_ratio : Ratio of length longest common sub string to min length of token count of Q1 and Q2

We ran classifiers using these features and got mediocre results, so looked into word vectors for getting more features. We used tf-idf weighted word vectors to extract an additional 384 features per question, resulting in 768 new features.

3 Methods, Results and Analysis

3.1 Methods

We ran five classifiers on our data. One of them being a Random Model which assigns class labels randomly to examples. We used this model as the baseline, that is, the actual classifiers should perform at least better than this. The other classifiers are Logistic Regression, Support Vector Machine, Random Forest and XGBoost.

Also, to do a comparative study to see how much of a difference word vectors make, we ran these classifiers on data with only the initial 7 features and then again after adding the tf-idf weighted word vectors.

3.2 Hyperparameter Tuning

We tried changing parameters while running the classifiers. For Logistic regression and SVM, we changed the alpha parameter which is a constant that multiplies the regularization term. We tried 10^{-5}, 10^{-4}...10^{2} as values for alpha and used the results of the best value of alpha. We found that the results vary a lot depending on alpha, following are the graphs showing the changes in log-loss on the final data:

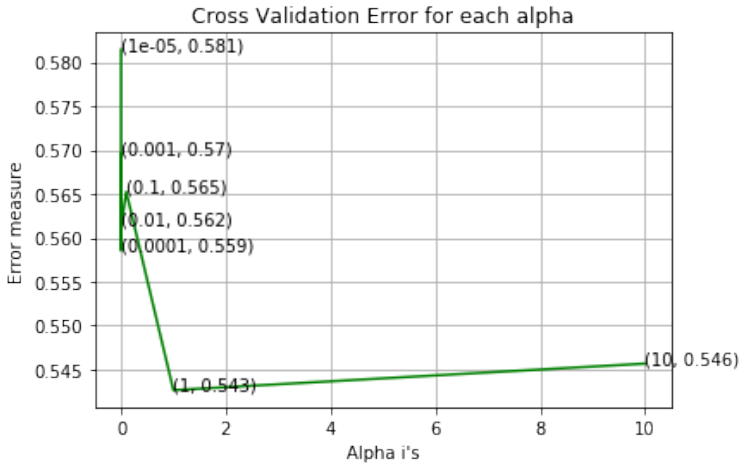

Figure 1: Alpha tuning for Logistic Regression

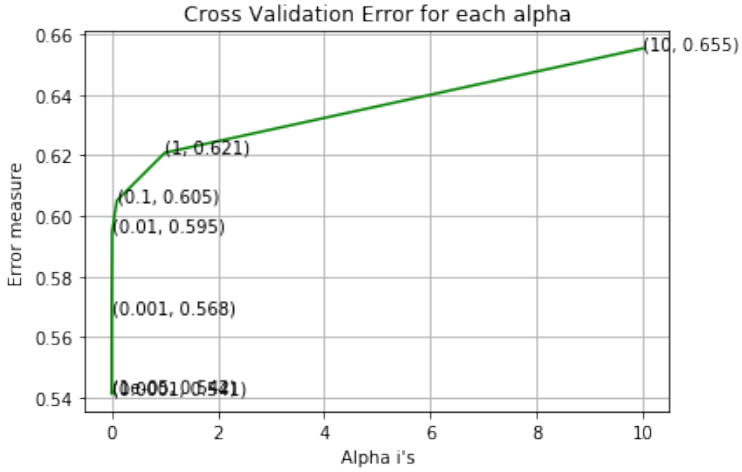

Figure 2: Alpha tuning for SVM

XGBoost and Random Forest were taking hours to run on our machines, so weren't able to tune parameters in these classifiers. Although, we did try changing the maximum depth for Random Forest and settled on 10 as the maximum depth because at that value Random Forest was beating XGBoost on the data with 7 features before adding word vectors.

Also, we realize that probably using a polynomial kernel with SVM might give better results, so that is in the list of things we want to try next.

3.3 Results

We have two sets of results because we ran the classifiers on sets of data, one with 7 features extracted using FuzzyWuzzy and string matching and the other set with tf-idf weighted word vectors added, just to see how much of a difference word vectors make.

Classifiers	Accuracy	Log Loss	F1 Score
Random Model	49.95%	0.880	0.491
Logistic Regression	65.90%	0.564	0.625
Support Vector Machine	65.70%	0.567	0.619
Random Forest	72.25%	0.490	0.714
XGBoost	71.87%	0.495	0.708

Figure 3: Results with 7 features before adding tf-idf word vectors

Classifiers	Accuracy	Log Loss	F1 Score
Random Model	50.19%	0.885	0.494
Logistic Regression	71.13%	0.542	0.672
Support Vector Machine	70.29%	0.541	0.660
Random Forest	73.53%	0.502	0.713
XGBoost	75.85%	0.464	0.747

Figure 4: Results after adding tf-idf word vectors

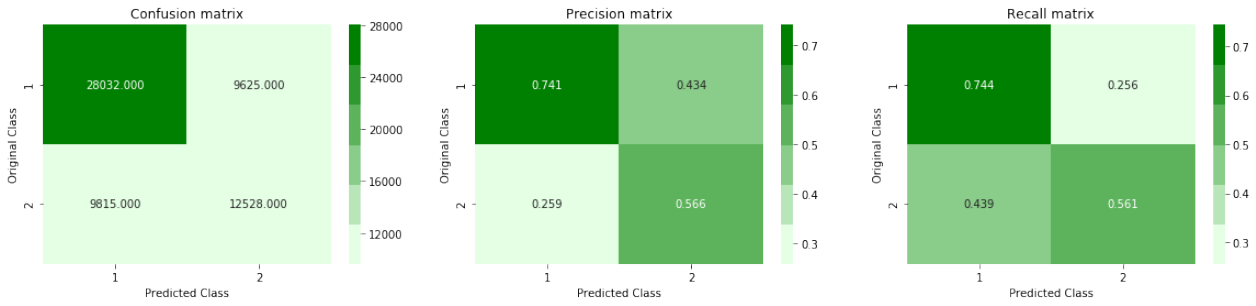

Figure 5: Confusion, Precision and Recall Matrices for XGBoost (best classifier) on the data with tf-idf weighted word vectors added.

3.4 Analysis

The results clearly show that adding tf-idf weighted word vectors was the right approach because performance of all the classifiers has improved and XGBoost is the best classifier among the five we used.

Logistic Regression and Support Vector Machine (SVM) both performed worse than XGBoost and Random Forest in both cases. We think this might be beacuase SVM and Logistic Regression are linear classifiers and our data is not linearly separable.SVM with the kernel trick using a polynomial kernel might perform better, so that's something we want to try in the future.

An interesting finding was that Random Forest performed better than XGBoost on the data with only 7 features. We were not expecting this, but we theorize that it is so probably because the number of features is less and the max depth of Random Forest is set to 10 while max depth of XGBoost is set to 4. We tried Random Forest with max depth set at a lesser value and it failed to outperform XGBoost. So, the less number of features and the increased depth of Random Forest might be the reason it is able to outperform XGBoost. As we can see in the results for the data with tf-idf weighted word vectors added, since the number of features was over 700, Random Forest could not perform better than XGBoost because it probably had a hard time classifying data with 775 features with max depth of 10.

3.5 Challenges

The major challenge we faced was handling a large data set. Initially our data set was not too large, however, adding tf-idf weighted word vectors lead to addition of 768 features making our data set huge. Our machines were not able to handle this much data and we were either getting memory errors or our machines were crashing.

We talked about this to Professor Downey and he suggested using a subset of the data instead of using the entire data. So, we ended up using 200,000 examples of the total 404,301 examples to run the classifiers. We tried a smaller subset but the classifiers were performing worse with lesser data. This is why the reported accuracies in our status report were higher, because at the time we had only 7 features so we were able to run the classifiers on the entire data. So, we believe with powerful machines, if we are able to use the entire data, the classifiers will definitely perform better,

4 Future

We used spaCy models in our project which are trained on the language used on Wikipedia, which we feel does not cover the colloquial language that is used on sites like Quora, so in order to deal with this, we theorize that a character-level model might work better and thus should be our next step. We also think looking into better data pre-processing techniques like lumping synonyms together should help us get better results. Finally, the lack of powerful hardware kept us from using the entire set of data which hindered our results. So, trying to use the entire data set on more powerful machines will be at the top of our priority list.

5 Contribution

- Omkar Satpute
  - Added newer features, ran classifiers on both sets of data.
  - Partially set up the website ,wrote later half of the report.
- Aditya Kumar
  - Performed pre-processing of data, added initial features.
  - Partially set up the website, wrote first half of the report.

Although the tasks were divided as above, most of the work was done in team meetings so both contributed to all the tasks mentioned above.