Data Science Challenge - X.ai

Divyansh Agarwal

November 30, 2015

1 Problem Statement

The problem was to provide a binary classification algorithm classifying the time entities in email responses as Positive Time and Negative Time Entities. The dataset consisted of about 75K positive time instances and 5K negative time instances. I was able to achieve a cross-validation accuracy of 0.9650 by using a word2vec representation of words in the sentence and applying a Random Forest classifier.

The dataset was challenging because:

1. A single email response could have both positive and negative time entities associated with them. So, a global approach based on a BoW model, which considers the sentence as a collection of individual words, would have lead to an ambiguous representation.

2. The number of words per entry were small due to the nature of the dataset (email-responses). The number of references to TIME, LOCATION, PERSON, etc. also did not add much discriminative value. These all factors lead to a poor representation using a BoW model.

2 Method

2.1 Preprocessing

For each sentence where the time entity appears, following transformations were applied:

1. Punctuations were removed.

2. Stop-words (from NLTK library) were removed.

3. Number's were removed.

4. The tokens belonging to the temporal expression being considered were removed along with the other smeared information represented by PERSON, LOCATION, etc

5. In some cases, above pre-processing led to removing all the words in a sentence. For eg., 'Tuesday, 9pm' with time-entity token 'Tuesday, 9pm'

would give an empty string after pre-processing. I found that such instances always corresponded to a Positive-time entity. Thus, in such instances, I replaced the empty string by 'works' which conveyed a Positive-time entity sense.

2.2 Feature Engineering using word2vec

2.2.1 What is word2vec?

Word2vec, published by Google in 2013, is a neural network implementation that learns distributed representations for words. It does not need labels in order to create meaningful representations. This is useful, since most data in the real world is unlabeled. If the network is given enough training data (tens of billions of words), it produces word vectors with intriguing characteristics. Words with similar meanings appear in clusters, and clusters are spaced such that some word relationships, such as analogies, can be reproduced using vector math. The famous example is that, with highly trained word vectors, "king - man + woman = queen."
This representation makes for an interesting application in time-entity classification problem because words like "can, sure, works, etc" would have similar vector representation and so would words like "cannot, don't, won't.

2.2.2 Approach

I get an average word2vec representation of the modified sentence using the word2vec representation of each word in the modified sentence. These features are obtained using the **Word2Vec** deep learning model implemented in the Gensim Python package. I obtained the pre-trained model of word2vec at:
https://code.google.com/p/word2vec/. The model was learnt on part of Google News dataset (about 100 billion words). It contains 300-dimensional vector representation for 3 million words.

2.3 Classifier

2.3.1 Adaptive Boosting

The core principle of AdaBoost is to fit a sequence of weak learners on repeatedly modified versions of the data. The predictions from all of them are then combined through a weighted majority vote (or sum) to produce the final prediction. The weights are individually modified for each of these weak classifiers. At a given step, those training examples that were incorrectly predicted by the boosted model induced at the previous step have their weights increased, whereas the weights are decreased for those that were predicted correctly. As iterations proceed, examples that are difficult to predict receive ever-increasing influence. Each subsequent weak learner is thereby forced to concentrate on the examples that are missed by the previous ones in the sequence. The critical parameter is the number of features. The larger the better, but also the longer it will take to compute. Results stop getting significantly better beyond a critical number. Experiments are done by varying the number of features. Figure 1 shows the increase in score with number of features. Best cross-validation accuracy of 0.936 was obtained using 160 features.

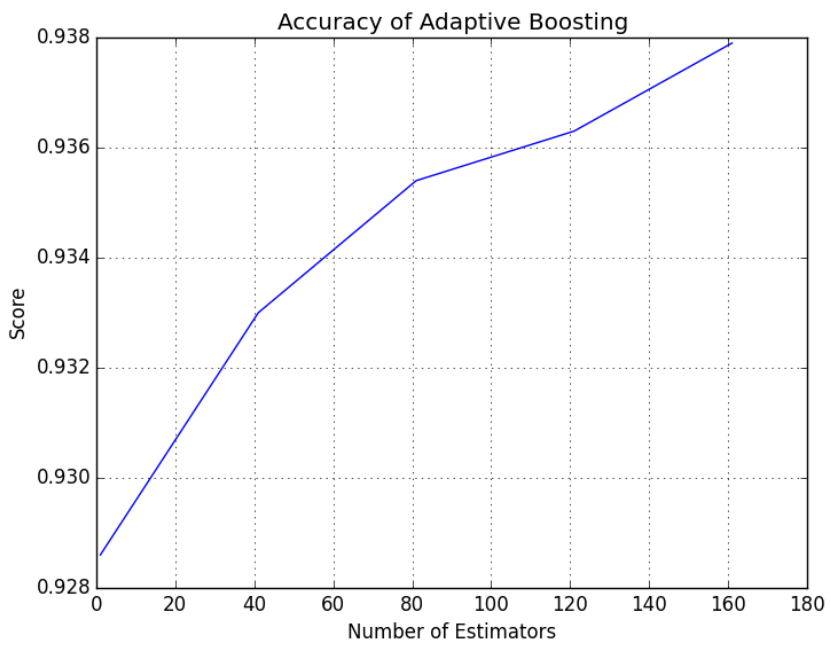

Figure 1: Accuracy of Adaptive Boosting.

2.3.2 Random Forest

Random forests is a notion of the general technique of random decision forests[1][2] that are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set. The critical parameter is number of estimators which indicates the number of decision tree in the forest.

Figure 2 shows the increase in score with number of estimators. Best cross-validation accuracy of 0.965 was obtained using 15/20 estimators.

2.4 Testing

I utilize a cross-validation strategy using a 5-fold cross-validation method. By default, scikit-learn uses a stratified sampling approach which ensures same percentage of positive and negative samples in each of the different samplings of the dataset. The training set is split into 5 smaller sets. The following procedure is followed for each of the 5 "folds":

- model is trained using 4 of the folds as training data;
- the resulting model is validated on the remaining part of the data

The approach is better than the random splitting of training set into train-set and validation-set as it ensures a better utilization of all the training samples.

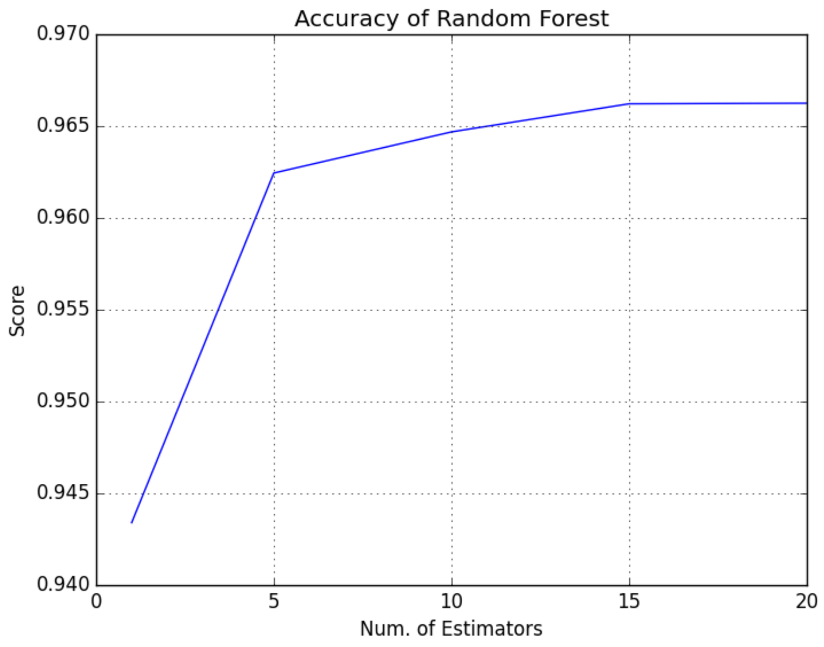

Figure 2: Accuracy of Random Forest.

2.5 Conclusion and Improvements

- Conclusion: I was able to achieve a cross-validation accuracy of 0.965 by using a word2vec representation of words in the sentence and applying a Random Forest classifier with 15 estimators.

- Improvements: System can be improved further by doing a weighted averaging of the word2vec representation of sentence rather than a simple averaging. This weighting would be done using the location of the detected time-entity token. Words farther from this token would get decreasing weights. This idea is based on the fact, that words in close vicinity convey relevant information about the token. For instance consider '**Friday** is difficult but **Saturday** works', in this case, when considering **Friday** which is a negative time token, less weight would be given to 'works' and similarly for 'Saturday' which is a positive time token less weight would be given to 'difficult'.

- Code Structure and Execution instructions are provided in the README.

- Python is used with generous help from libraries like gensim for word2vector feature representation, scikit-learn for classifier, numpy and nltk