Yggdrasil:
Natural speech learning with random forests

Joseph D. Romano*[1] and Alexandre Yahi†[1]

[1]Departments of Biomedical Informatics, Systems Biology, and Medicine, Columbia University

May 4, 2016

1 Introduction

Natural language processing and speech recognition comprise one of the largest applications of machine learning and data mining techniques, spanning diverse industries such as economics, healthcare, information retrieval, and personal computing<REFs>.

2 Description of provided data

The original input we were provided with consisted of 126837 data records and 52 features, 36 of which were numeric (including binary 0/1) and 16 categorical. Each categorial feature had a variable number of potential categories. It is unclear what each feature vector represents, but the labels corresponding to some of the categorical feature vectors provide clues as to their meanings. For example, the categorical feature vector labeled "26" includes terms such as `vacknoweldge_acknowledge`, `vacknowledge_explain`, and `vacknowledge_clarify`, suggesting that it encodes the type of response given by one of the two parties holding the conversation. Other features seem to encode information about prepositional phrases, object comparisons, negation, and others. Some further information about the original encoding of the data is available in <ref>.

3 Learning approach

Our optimal learning approach for the binary speech classification problem involved two main steps:

1. Encoding all categorical feature vectors as sets of binary vectors via expansion
2. Training a random forest classifier and predicting over unlabeled data

We will discuss each of these below:

*jdr2160@cumc.columbia.edu
†ay2318@cumc.columbia.edu

3.1 Data preprocessing and feature design

One noteworthy characteristic of most random forest classifiers is that they are unaffected by scaling, centering, and other monotonic transformations, since all operations on the data are simply linear partitioning (although it is possible to design non-linear decision boundaries for partitioning the data, doing so is generally unnecessary with random forests).
In order to feed categorical features into our model, we performed a one-hot encoding. This preprocessing step transform a categorical feature with n possible values into n distinct binary features where only one of them can be positive for each learning instance.
For consistency purposes, we performed the feature expansion with both training and quizz dataset to keep track of the expansion order of the features and make sure we kept track of the final feature list. We then separated the datasets back and split numerical and expanded binary features for each set.

3.2 Model description

Random forest is a meta-algorithm of the family of bagging algorithms. It is performing bootstrap sampling to train decision trees with the particularity of doing the split on only $\sqrt{d}$ of the d features. <CITE Leo Breiman paper> Although random forest can accept any type of data, should they be ordinal, categorical or numerical, the implementation of the random forest algorithm in scikit-learn required numerical input, hence our pre-processing step. We decided to use 600 estimators which offered a good compromise of computing-time and performances as the accuracy plateaued starting around 500 decision trees. The quality of the split of each tree was assessed with the Gini impurity function. The stopping criterion was that nodes would be expanded until all the leaves are pure. Each iteration of the bagging meta-algorithm involved bootstrap samples when building trees.

3.3 Model selection

For the initial exploration of the data, we set up a pipeline with 10-fold cross validation to get a sense of which learning algorithm would perform the best. We found that random forest was returning above baseline predictions just by training at numerical variables (0.893 compared to MLP 0.74), and that support vector machine (SVM) was not appropriate for this dataset in addition of taking much more time to train. [Give results about other methods we tried]

4 Results

4.1 Predictor evaluation

We trained our model on the concatenation of numerical features and categorical features expanded into binary ones for the reasons above mentioned. We randomly split the training data into a training set (33%) and a hold-out set to evaluate the performances and tune our random forest model parameters. We performed a 10-fold cross validation on the hold out set.

4.2 Results of evaluation and analysis

For the evaluation of the random forest method we picked, we had to tune the number of trees (estimators) per iteration. In order to select empirically the best number of estimators based on the labeled set available, we computed the cross-validation score, the accuracy, the average precision, the F1 score, the precision, the recall and the area under the ROC.

Comment the table.

Estimators	CV score	Accuracy	Average Precision	F1 score	Precision	Recall	AUROC
5	92.99	92.98	94.01	91.86	93.46	90.31	92.67
10	93.63	93.64	94.96	92.48	95.96	89.24	93.15
50	94.21	94.21	95.42	93.18	96.32	90.24	93.77
100	94.14	94.14	95.35	93.10	96.20	90.20	93.70
250	94.22	94.22	95.43	93.19	96.35	90.23	93.78
500	94.25	94.25	95.45	93.23	96.33	90.32	93.82
600	94.22	94.22	95.44	93.19	96.39	90.19	93.77
750	94.22	94.22	95.43	93.20	96.33	90.26	93.79

5 Discussion

This dataset highlighted several important concepts in machine learning. First, the No Free Lunch Theorem reminds us that there is no perfect learning algorithm that can perform well in any situation. This problem presents specificity that has failed highly tunable models like SVM. SVM needs fine setting of kernel and hyper-parameters and we could not find the appropriate set of parameters during the exploratory phase of the study to keep this method as a potential candidate. We tried other powerful methods such as Multi Layer Perceptron, a neural network method that did not perform well at first with a quasi-Newton method (0.74 accuracy) and plateaued at 0.93 once we selected a stochastic gradient-based optimization algorithm. Usually neural networks are methods with high variance while random forest is considered a small variance model. It did not required any centering and standardization at the difference of the other algorithms.
We can therefore hypothesize that the data that we have It is also important to note that because we were using scikit-learn, one of the few packages allowed for this challenge with Python, we were limited by the fact that all its methods only accept numerical features. This led us too perform a one-hot expansion that resulted in a dramatic increase in features

6 Conclusions

7 Author contributions

JDR and AY jointly performed model selection and designed the final model. JDR wrote, formatted, and documented the source code for Yggdrasil. JDR and AY both participated in analysis of the results and submissions to the Kaggle competition. AY prepared the manuscript.