COMS 4721 Spring 2016: Kaggle Competition

Swapnil Khedekar, Craig Liebman, Felix Matathias

May 5, 2016

Introduction

In this brief report we describe the methodology we followed for the in-class Predictive Modeling Competition for COMS 4721, Spring 2016. We developed a binary classifier with an error rate of approximately 5% which performed well in the competition. We chose to completely ignore the domain semantics of the data and our feature selection was solely based on the characteristics of the input data, including their variance and correlation. In addition to Random Forests, we also tried Logistic Regression, Support Vector Machines, and AdaBoost with Decision Tree as the weak classifier. All classifiers performed relatively well but Random Forest was chosen for its stability and speed. This report is divided in four parts. First we describe the feature selection procedure, then we continue with the comparison of various classifiers, we proceed with the tuning of the winning classifier, and finally we report the results we achieved.

Feature Selection

The feature set is a collection of 52 numerical and categorical features. We did not conduct any research on the meaning of the feature data but rather conducted a domain agnostic feature selection. In the following we will refer to the columns by their numbers as described in the competition site. We first excluded columns which are linear combinations of other columns. We discovered that columns 18, 23, 25, 26, and, 58 were linear combinations of other columns in the feature set and thus we did not include them in the training process. We then removed columns with no variance, 29, 31, 32, and, 35. Continuing the analysis, columns 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, and, 51, had values in the set $\{1, 2, 3\}$ and since we did not use any domain knowledge we treated these variables as categorical. The only two real numerical columns, 59, and 60 were centered and normalized but further analysis proved that normalization did not have any significant influence in the classification error and thus were not normalized in the final submission in order to produce a simpler model. The pure categorical columns 0, 5, 7, 9, 8, 14, 16, 17, 20, 56, 57, along with the ones we treated as categorical, were converted using a one-hot representation to a final set of 545 feature columns. We further attempted to drop additional categorical columns based on the standard deviation of the data. Even after dropping 350 features we achieved an accuracy

within 2% of our accuracy before dropping. We did not have time to further investigate the data and decided to keep these features.

The reduction of the initial number of columns had significant impact in the analysis and running time of the algorithms we tested. Without dropping any of the input data, the one-hot representation yielded 5594 columns. After feature selection, the number of columns dropped by an order of magnitude to 545.

Algorithm Selection

We considered three major classes of models in our research: linear, non-linear and ensemble. We started with Logistic Regression for a few reasons. The algorithm is fast, e.g. compared to SVM which has cubic running time with respect to the size of the training set. At the onset we wanted to establish a baseline classifier and logistic regression is simple in that the model assumes a single linear decision boundary.

Our approach was to randomly partition the data into training and holdout sets and compare select algorithms with varying parameter settings against each other. This is presented in Figure 1 with the different classifiers and parameter settings on the horizontal axis and classifier accuracy on the vertical axis. Each point on the plot represents the average of runs. For Logistic Regression we plot C ($1/\lambda$, where λ is the regularization term) on the horizontal axis and hold out accuracy on the vertical axis. It is clear from this plot that for a sufficiently large value of C, meaning the model is not too influenced by regularization, the classifier plateaus with respect to accuracy. For a baseline this was not too bad but the results pushed us to consider non-linear models.

We also experimented with SVM, using both linear and Gaussian (RBF) kernels. We focused our effort on the RBF kernel because our available training set had more than 100,000 observations and we only had around 500 features after one-hot encoding and the results were satisfactory with an error rate slightly below our final classifier. SVM, especially with RBF kernel, had a runtime on the order multiple hours to train and test an individual model. This long iteration cycle made experimenting with different feature sets, feature transformations and different parameters nearly debilitating. Some actions we could have taken to shrink the iteration cycle are focus on linear kernel and drastically shrink the training set.

For SVM Figure 1 contains a plot of C (error penalty) on the horizontal axis and hold out accuracy on the vertical axis. A small value of C tends toward simple decision boundaries while a high value of C (depending on the data) tends toward complicated decision boundaries. This occurs because C imposes a penalty on misclassifying training samples. The figure shows that a higher C (at least until C=1000) and more complicated decision boundary actually reflect the overall data and not just the idiosynchracies of the training samples.

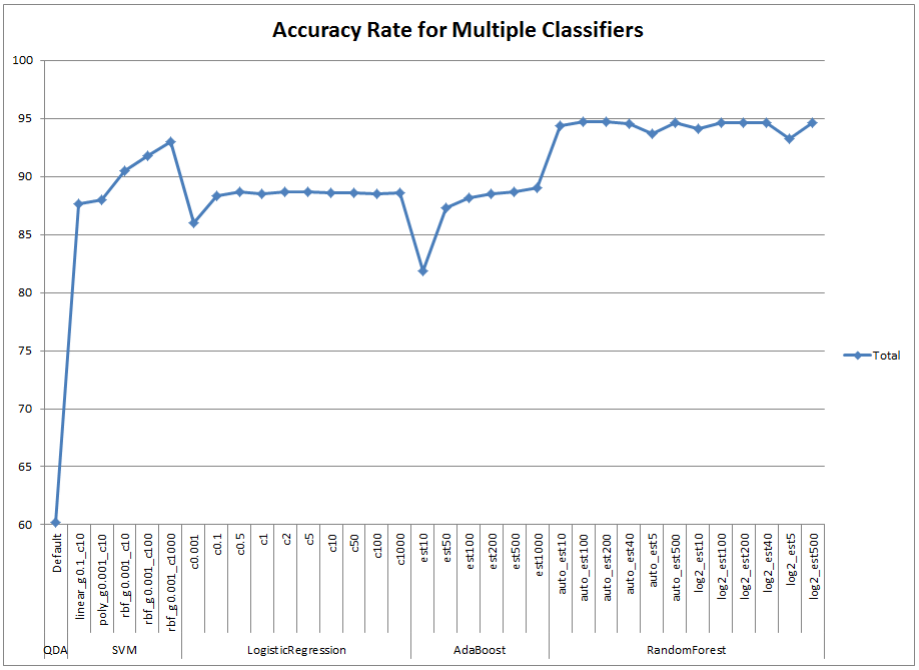

Figure 1. Accuracy of Classifiers vs Tuning Parameters

We then turned to ensemble methods. We quickly tried AdaBoost with Decision Trees of depth 1 as the weak learner but the results were only slightly better than Logistic Regression and definitely worse than SVM. We then turned to Random Forests which are ensembles of Decision Trees. What drew us to Random Forests is how it uses randomness. Unlike Decision Trees which use all features to make a split decision, Random Forests only consider a random subset of features at each split decision up to a constant and identical number for each tree. This gave us the power to test different subsets of features, albeit implicitly. From the start Random Forest performed on par with SVM, but unlike SVM the runtime is fast and provided us with short iteration cycles to tune a few parameters.

In Figure 1 we can see that Random Forests perform better and more consistently than the other algorithms we considered. Random Forests offer a number of parameters with which to tune the model. We will examine some of those in more detail. For the above reasons we proceeded with Random Forest as our final classifier.

Model Tuning and Evaluation

After the selection of Random Forest as the winning algorithm we proceeded to fine tune the model in order to achieve maximum accuracy for our holdout set. Figure 2 shows the results of changing the minimum number of samples in a leaf node in any Decision Tree learned by the algorithm to accuracy, precision, and recall on a holdout set. Intuitively, the lower this number the more details the algorithm can learn from the training data. This is similar to increasing the C parameter to SVM to learn a more complicated decision boundary. This experiment shows that having a small minimum number of samples in a leaf actually provides an improved accuracy, precision and recall. Figure 3 shows the results of changing the the minimum number of samples required in an internal node to split the samples further of

any Decision Tree in the Random Forest on accuracy, precision and recall on a holdout set. Again, we found maintaining the minimum number of samples required to split an internal node at relatively low value produced the best results across the board for these three metrics.

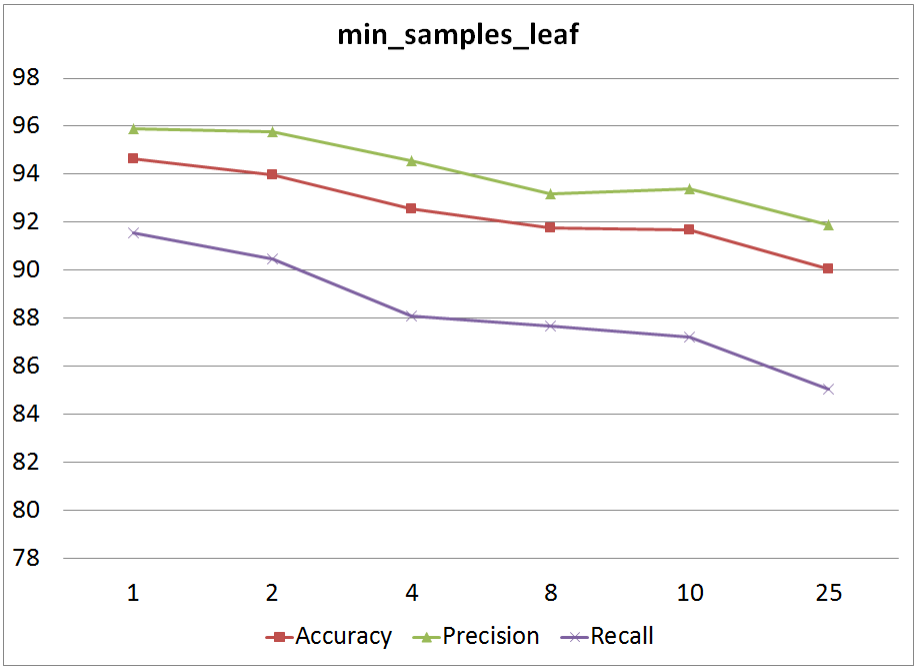

Figure 2. Accuracy of Random Forest vs Minimum Samples Per Leaf

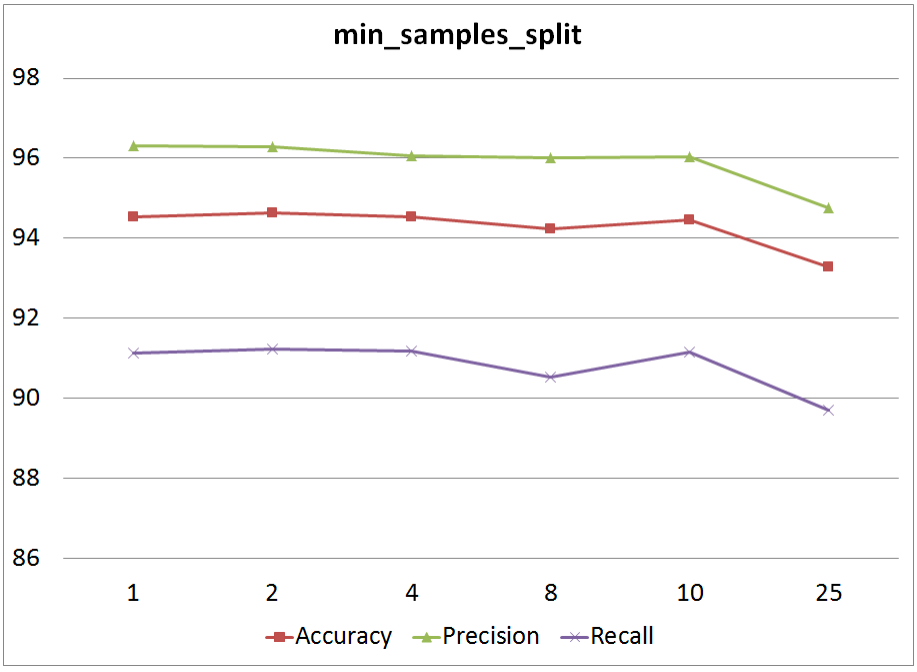

Figure 3. Accuracy of Random Forest vs Minimum Samples To Split

Finally, Figure 4 shows what happens to accuracy, precision and recall on a holdout set as we increased the number of individual Decision Trees used in the Random Forest

(n_estimators). Since each Decision Tree trains over a random set of features, increasing the number of underlying classifiers and hence the total number of features used in the Random Forest allows the algorithm to capture more variance. The decision of how many features to include in the Random Forest is controlled by the max_features parameter. We tried setting this parameter from the default of the square root of the total number of features (23 in our case) to the $\log_2$ of the total number of features (9 in our case) but the metrics we observed did not vary by much (see Figure 1).

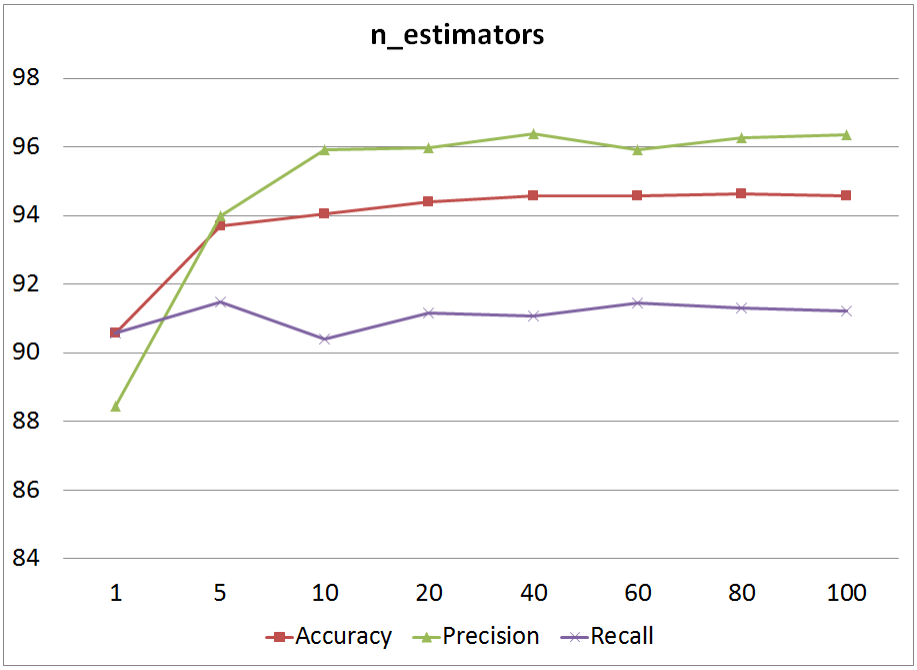

Figure 4. Accuracy of Random Forest vs Number of Estimators

Final Classifier

In our final classifier we set the minimum number of samples per leaf to one, the minimum number of samples to split an internal node to two, the number of features considered for a node split to the square root of the total number of features, and the number of Decision Trees in the forest to forty. In the Kaggle competition this classifier achieved an accuracy of 95.15% on the public quiz data and an accuracy of 94.89% on the private quiz data.

Contributions

We worked collaboratively in every aspect of the project including data exploration, coding and writeup.