Lucas Newman

February 25, 2019

**State-Farm Data Science Interview**

**Classification Exercise Report**

**Data Cleaning & Model Overview**

The goal of this exercise was to build 2 effective classifiers for predicting “y” in the given data set. Prior to beginning modeling, significant data cleaning had to be accomplished. Specifically, the data types of each variable were observed. Some columns were incorrectly object types and were converted to numeric data types. A few of the columns contained categorical data that were transformed utilizing dummy variables. Transforming the data to ensure it was the correct data type was critical before addressing missing values.

Upon inspection, it was found that every column besides the target column, “y”, contained missing values (818 missing values in the training data). To treat missing values, procedures such as: dropping rows with missing columns, heuristic mean/median imputation, predicting missing values with supervised techniques are often used—despite hindering aspects of statistical learning--due to their simplicity. For example, simply dropping columns leads to information loss, while heuristic approaches add variance and bias, and predicting missing values leads to proxy-like substitutes. Thus, an alternative approach was implemented. The python library “fancyimpute” was employed due to its variety of more sophisticate imputation algorithms. Although, often considered the gold standard in dealing with imputation, MICE, was not the favored approach in this case due to the nature of the data. Instead, an alternative algorithm based on spectral regularization for large incomplete matrices—softImpute—was used. The inspiration behind the softImpute algorithm is to provide a simple and efficient convex algorithm that minimizes reconstruction error on the nuclear norm, providing low-rank solutions for large scale matrix problems. SoftImpute iteratively replaces the missing elements with those obtained from those obtained with a soft-thresholded SVD, allowing one to compute an entire regularization path of solutions on a grid of values of the regularization parameter. Thus, softImpute was used to impute the missing values. After all the data cleaning was concluded, there were 130 feature/predictor (x) variables in both the training and test data, and 1 column of target variable (y) in the training set. Moreover, the training data had 40,000 observations and test data had 10,000 observations. The addition of dummy variables caused the number of features to expand from 100 to 129.

**Data Cleaning Procedure: Used on Training and Test Data**

* Examine data types: convert “x41” and “x45” to numeric
* Create dummy variables to encode “x34”, “x35”, “x68”
* Impute-fill the missing feature values using softImpute from fancyimpute
* Create 3 numpy arrays from our data frames, “test\_data” from our test.csv these are the X features we will be trying to predict, “train” all our X features from our training.csv, “y” our labels we will be predicting.
* We create our models based on our training data so do a train-test-split of:
  + X\_train, X\_test, y\_train, y\_test= train\_test\_split (train, y, with test size=0.3 and random\_state=42)
  + A scaler was not chosen to be used since predictive power was the priority, this choice does affect hyperparameter tuning speed.

**Model overview**: There are 5 numpy arrays utilized throughout. X\_train and y\_train, train the predictive models. The models are scored on X\_test, and y\_test to determine their predictive abilities. These models are then used to predict the probability of class label (y=0 or y=1) for the data in the test\_data array.

**Classification Approaches**

Intuitively, it is known that for this type of data, logistic regression and linear discriminant analysis will act as good algorithms to predict the class label of y. Thus, both algorithms were run in an out-of-the-box fashion, with no hyperparameter tuning, to get a baseline expectation on how well the features serve as predictors of the target and to have a baseline. The score for logistic regression and LDA, were approximately 0.889 and 0.890 respectively. However, given that the goal of the task is to ensure the highest predictive power on our test.csv, the decision to use as many features as needed was taken. Due to this, and the nature of LDA and logistic regression to overfit, and increase bias and variance, the route of using ensemble methods was taken. Specifically, random forest classification and gradient boosting classification.

Random forests is an ensemble method that uses rules of randomly created decision trees to predict outcomes. Procedurally in random forests, one randomly selects *k* features out of *m* total features such that *k<<m*. Then among these features calculate the node *x* using the best split point. Split node *x* into its children nodes using the best split point. Random forests recursively iterates these 3 steps until *i* number of nodes has been reached. Random forests then builds the “forest” by repeating all the previous steps mentioned *n* times such that there now is a “forest” with *n* trees.

Gradient Boosting is a later derived application of AdaBoost. Boosting is essentially making accurate prediction rules by combining weak rules. AdaBoost works by weighing observations, specifically by putting more weight on difficult to classify instances. New, weak learners are sequentially added. A weak learner is a decision tree containing only a single split. Predictions are based on majority vote of weak learners. Gradient boosting makes an optimization problem out of boosting where the objective function is to minimize the loss function by adding weak learners through gradient descent. Gradient boosting machines essentially have 3 elements: a loss function, weak learners, and additive model. The loss function depends on the model, in the case of the classification done here friedman\_mse was used. In the model used, weak learners were also ensured by greedily manipulating parameters such as split points, minimum leafs etc. Then an additive model was added and a gradient descent function is used to minimize the loss function regarding: tree constraints, shrinkage, and penalized learning.

Random forests and gradient boosting machines were chosen as prime candidates to handle this predictive problem due to their ability to generalize well with many features, and be less subject to overfitting and bias. The main challenge was to properly tune these algorithms. The solution used for hyperparameter tuning was grid search cross validation. Although tuning both classifiers was time consuming (about 1.5 hours per each classifier), the classifiers performed better than logistic regression and LDA, had high AUCs, and are more robust. In summary the approach for each classifier was to create pipeline where each classifier would have a defined parameter grid, and GridSearchCV was conducted on the pipeline containing the parameter grid. The resulting tuned models were then fit the X\_train and y\_train and scored on X\_test and y\_test. Additionally, a ROC plot was made for each classifier and metrics such as accuracy, precision, recall, f1-score, and support were calculated for both classifiers. Lastly, each classifier was then used to predict the probability of the test\_data belonging to either class 0 or class 1 in y. The results of these probabilities were exported to CSVs called “results1.csv” and “results2.csv.”

**Results: Random Forests**

A pipeline was set up that would take logistic regression and random forests as classifiers, then tune the models with GridSearchCV over a defined parameter grid, and fit the best resulting model to the data. For logistic regression the parameters tuned were: penalty, C, and solver, and for random forests the parameters tuned were n\_estimators and max\_features. The pipeline chose random forest as the best classifier “best\_clf” had an accuracy of 0.902 and an AUC of 0.966, which were significant improvements over the out-of-the-box logistic regression and LDA. A more comprehensive classification including precision recall f-1 score and support for each class is included in the Jupyter notebook. The results were exported to a csv named “Results1.csv.”

**Results: Gradient Boosting Classifier**

A pipeline was set up that would take LDA classification and gradient boosting machines as classifiers, then tune the models with GridSearchCV over a defined parameter grid, and fit the best resulting model to the data. For LDA the parameters tuned were: solver, n\_components, and shrinkage. Meanwhile for the gradient boosting machine the parameters tuned were: learning rate, max depth, min sample split, min sample leaf, n\_estimators and max\_features. The pipeline GradientBoostingClassification as the best classifier “best\_clf2” had an accuracy of 0.901 and an AUC of 0.968, which were significant improvements over the out-of-the-box logistic regression and LDA. A more comprehensive classification including precision recall f-1 score and support for each class is included in the Jupyter notebook. The results were exported to a csv named “Results2.csv.”

**Future Methods to Explore & Recommendations**

The main challenge in this exercise was having to deal with computationally exhaustive hyperparameter tuning methods. This challenge was accepted due to GridSearchCV being a very effective tuning algorithm, and given that the purpose was to have high predictive power, the benefit of high predictive power outweighed the computationally intensive cost. In the future, Bayesian optimization methods and neural networks should be used for more efficient hyperparameter tuning. Or one could simply instantiate an instance of Spark and conduct hyperparameter tuning with Spark/PySpark. Also, if prediction on test data becomes less of a priority standardizing the training data would also speed up grid search. Additionally, one should study the difference in the predicative power of the algorithm if MICE or imputation via k-NN is done instead of softImpute. Another valid analysis would be to reduce the dimensionality of the model and create a simpler model using Gradient boosting regression for feature ranking or logistic regression with L1 penalty and omit all zero coefficients. It would also be interesting to see how these models would perform compared to something like H2O’s Auto-ML. In conclusion, both classifiers created accurately predicted the hold out training examples with +90% and both had AUC values above 0.96.

**Conclusion**

Both random forest and gradient boosting classifier nearly had identical accuracy, and AUC. Random forest had higher accuracy score, while gradient boosting had higher AUC. For the purposes, of this exercise AUC is more valuable thus if one algorithm had to be chosen one should chose the gradient boosting classifier, or classifier 2 (“best\_clf2”).

**Figures: Shown Below**

**Classifier 1:**

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A close up of a map

Description automatically generated

**Classifier 2:**

**A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated**

A close up of a map

Description automatically generated