Intrusion Detection: README Document

Vamshidhar Pandrapagada

April 23, 2017

Overview

The goal of this project on model optimization using ensemble methods is:

1. To detect intrusion from a complex, unbalanced data set that has ~500 thousand observations, 42 features, 23 classes; with highly unequal distribution of observations across different classes.
2. To design and optimize an ensemble classifier using bagging, boosting, and random forest techniques.
3. To validate and evaluate the performance metrics relevant to the problem.

Exploratory Data Analysis

The initial data set contains 494020 observations, 42 features and 23 unique classes. EDA will help us analyze how our Classes are distributed over the entire data set. This gives us a general idea on how we split the data into Training, testing and validation data sets. The idea here is, if any one of the classes in the training data is very rare, and when we draw bootstrap samples on it, we may encounter samples in which the rare class may never appear. When that happens , the model we build will not be able to make predictions when it encounters an observation with a value for the class it has never seen before.

After Visual analysis:

- It is evident that that classes are not evenly distributed.
- The data range is very wide and the mass of the distribution of classes is heavily concentrated on one side and it is very difficult to see the actual numbers of the classes which are very low.
- For this exercise, we have chosen to build a predictive model on the top 3 classes (attack types), so that when bootstrap samples are taken, the probability of every class being included in the sample is high.

Remove unnecessary features from the Data set

Another advantage of EDA is, it will help us understand the data better. As mentioned above, the data set contains 42 observations and some of them may not add any value to the model we build. Looking at the data set, we have decided to filter out the following features.

1. protocol_type
2. service
3. flag
4. is_host_login
5. num_outbound_cmds

Split Data in Training and Testing Data sets

caret package provides a method **createDataPartition** which creates a series of test and training partitions on the main data set using bootstrap samples, while making sure there is similar distribution of classes on both the sets. This can be seen in the implementation and we have used *ggplot* function to visually prove the distribution.

Building Ensemble Models (Process Flow)

We attempt to solve this problem using 3 different ensemble techniques: Bagging, Boosting and Random Forest. 3 different models are used to compare the performance and prediction accuracy for each model.

1.Bagging:

Following steps will walk you through the details of the implementation:

1a. Build the Model:

1. Set the seed and collect the bootstrap samples. It is always a good practice to set the seed before doing any randomized operation. This is a useful technique which helps to replicate the issue should there be any issue or bug in the implementation.

- a. Set the seed
- b. Collect the bootstrap samples, each sample equal to the number of rows in the training data Set.
- c. Number of samples taken in this use case is 11. This means we build 11 models with 11 different bootstrap observation samples. Again, always a better option to chose an odd number of samples so that there is no tie while taking the majority vote in a classification model.

2. Build decision tree models based on the bootstrap samples.

- a. In this exercise, Decision tree implementation **C5.0** method from **C50 Package** is used.
- b. C5.0 method , instead of giving a probability while predicting, it preserves the actual class name as predicted value.

3. Collect Out of the Bag samples.

- a. Observations which were not chosen by the bootstrap sample at that iteration are called as Out of the Bag Samples.
- b. OOB samples are collected for all 11 iterations. These observations are used to record the accuracy of the model.

1b. Model Error Evaluation:

4. Build a function for Predictions using *predict* method. This method is called for every tree-OOB combination (11 times) and predictions for all 11 models are stored in a list. Plot the variations in OOB accuracy across different models.
5. Using the predictions from Step 4, Compute the predicted value using majority vote across 11 different models. Calculate mean accuracy by comparing the prediction with actual Value.
6. Calculate Test Accuracy.

- a. Similarly, we record test accuracy by calling predict function for all 11 trees and testing data set combinations.
- b. Finally we calculate the mean accuracy by taking the MAX VOTE logic and print classification table to record the predicted observations.

2.Boosting:

There are several types of Boosting techniques. But in this example, we will use **AdaBoost** which is short for **Adaptive Boosting**. *adabag* package provides an implementation of **AdaBoost** algorithm.

2a. Build the Model:

1. Use *boosting* method from *adabag* to perform boosting. This method Fits the AdaBoost.M1 (Freund and Schapire, 1996) and SAMME (Zhu et al., 2009) algorithms using classification trees as single classifiers.
2. The following parameters are set before calling *boosting* method.

- a. Uses boosting method from *adabag* package.
- b. *mfinal* is the used parameter to specify the number of trees.
- c. *coeflearn* If 'Freund' , error function alpha=ln((1-err)/err) is used.
- d. *boos* if TRUE (by default), a bootstrap sample of the training set is drawn using the weights for each observation on that iteration. If FALSE, every observation is used with its weights.
- c. *control* is rpart control which uses a decision tree fit with a maximum depth of 3.

2b. Model Error Evaluation:

3. Use *predict.boosting* function to predict the results on testing set.

- a. Confusion Matrix is printed.

4. Calculate error evolution to see the variance in error while boosting over iterations. *adabag* package provides a method call *errorevol* which allows the user to estimate the errors in accordance to the number of iterations. Error evolution can be performed on both Training and Testing data sets.

3.Random Forest:

randomForest package provides an implementation for randomForest algorithm for classification and regression.

3a. Build the Model:

1. The following parameters are set before calling randomForest method.

- a. *ntree* is the parameter used to specify the number of trees we build in the random Forest.
- b. *mtry* is the number of features sampled for use at each node for splitting.
- c. We have 37 features in our data set. Square root of 37 is ~ 6. The model is fit using *mtry* value as 6.
- d. Model is fit using *randomForest* method.

3b. Model Error Evaluation:

2. Predict and Test Accuracy

- a. Use *predict* function predict the values on Testing data set.
- b. Print the confusion matrix table to print stats and gather accuracy.

3. Observe Variable importance

- a. Random Forest model allows us to examine the variable importance. This can be done using *importance* function.
- b. Use *plot* function to plot the mean square error of the forest object. This plot shows error variance for each tree that is fit inside the forest.

- c. Use *ggplot* function to obtain the plot of variable importance. Optionally, we can also use *varImpPlot* to plot a very basic version.

Conclusion

For this particular data set, all the 3 models performed fairly well. But by looking at the accuracy of each model, Random Forest won by a very low margin with accuracy almost equal to one without any False Positives or False Negatives.
Also in general, out of all the state of art classifiers, Random Forest is a winner majority of the times. This is a very powerful algorithm based on bagging decision trees, with an exception that a different feature sample is randomly selected during the construction of every new tree. By doing this, the model reduces the correlation between trees, captures significant localized variations in the output and improves the degree of variance reduction in the final result.
Random forest also scales well with a larger number of input features.

References

1. Rui Miguel Forte (2015): Mastering Predictive Analytics with R
2. Nina Zumel, John Mount (2014): Practical Data Science with R
3. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani (2015): An Introduction to Statistical Learning.