Modeling in Revolution R Enterprise Module 8: Decision Forests

Random Forest

A Decision Forest (Random Forest) is an ensemble of decision trees. In ScaleR, each tree is fitted to a bootstrap sample of the original data, which leaves above a third of the data unused in the fitting of each tree. Each data point in the original data is then fed through each of the trees for which it was unused.

- The Decision Forest prediction for that data point is the statistical mode of the individual tree predictions - that is, the majority prediction.
    - Note that the above applies to classification. For regression, the prediction is the mean of the individual predictions.

Random Forest

Building one decision tree to its maximal depth/maximal complexity without any pruning can over fit the training data and perform poorly on new data. Combining many (hundreds) such trees, each using different variables/observations and overfitting the data differently, will lead to a final model (a forest of trees) that is less biased and more robust.

The final decision of the forest will be driven by the majority with each tree bearing equal weight. Unlike individual decision trees, decision forests are not prone to overfitting, and are consistently shown to be among the best machine learning algorithms.

rxDForest

The rxDForest fists a classification or regression decision forest to data in ScaleR. To create the forest, you have to specify the number of trees using the nTree argument, and the number of variables to consider for splitting in each tree using the mTry argument.

```
rxDForest(formula, data, nTree, mTry, ...)
```

- In most cases, you will also want to specify the maximum depth to grow the individual trees:
  - Greater depth results in greater accuracy, but results in significantly longer fitting times.

Example: Random Forest

Let’s repeat the same example from the last module, only this time we will construct a random forest model.

Again, let’s speed up the computation time by defining a complexity parameter equal to 0.01.

However, we also have to specify the number of trees, nTree, which we will set at 500.

Further, we will define the number of variables to consider for splitting in each tree, mTry, to two (the number of response variables). Finally, we will define a seed for potential reproducibility:

```
infile <- file.path("data", "BankXDF.xdf")
BankDS <- RxXdfData(file = infile)

Forest <- rxDForest(housing ~ balance + age, data = BankDS, seed = 10, cp = 0.01,
    nTree = 500, mTry = 2, overwrite = TRUE)
```

Example: Random Forest

```
Forest
```

```
##
## Call:
## rxDForest(formula = housing ~ balance + age, data = BankDS, overwrite = TRUE,
##     cp = 0.01, nTree = 500, mTry = 2, seed = 10)
##
##
...
```

Example: Interpretation

The OOB (out-of-bag) error estimate is calculated using the observations that are not included in the training set. The result indicates that the model will lead to erroneous conclusions 39.57% of the time when applied to new data.

The confusion matrix, located below the OOB, compares predicted values (columns) and actual training observations (rows).

- The model predicts that the client owns a home when he or she does not approximately 7% of the time.
- The model predicts that the client does not own a home when he or she does approximately 80% of the time.

Exercise: Random Forest

We will repeat the last exercise in the previous module, only this time constructing a random forest rather than a random tree. Classifying whether or not a client will subscribe to a term deposit (y) given the duration of the advertising phone call (duration) and the client’s age (age), construct a random forest determining the OOB rate and the confusion matrix.

- As with the above example, using a complexity parameter equal to 0.01
- Set the number of trees, nTrees, equal to 500
- Define the number of variables for splitting in each tree to 2.

Note: it is okay if your results differ slightly from the solution; otherwise, set your seed to 10 if you want to match.

Exercise: Solution

```
Forest2 <- rxDForest(y ~ duration + age, data = BankDS, seed = 10, cp = 0.01,
    nTree = 500, mTry = 2, overwrite = TRUE)
```

```
## Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.002 seconds
## Rows Read: 10000, Total Rows Processed: 20000, Total Chunk Time: 0.006 seconds
## Rows Read: 10000, Total Rows Processed: 30000, Total Chunk Time: 0.007 seconds
## Rows Read: 10000, Total Rows Processed: 40000, Total Chunk Time: 0.006 seconds
## Rows Read: 5211, Total Rows Processed: 45211, Total Chunk Time: 0.007 seconds
## Rows Read: 10000, Total Rows Processed: 10000, Total Chunk Time: 0.002 seconds
...
```

```
Forest2
```

```
##
## Call:
## rxDForest(formula = y ~ duration + age, data = BankDS, overwrite = TRUE,
##     cp = 0.01, nTree = 500, mTry = 2, seed = 10)
##
##
...
```

Exercise: Solution

The OOB (out-of-bag) error estimate indicates that the model will lead to erroneous conclusions 11.1% of the time when applied to new data.

The confusion matrix indicates:

- The model predicts that the client will not subscribe to a term deposit when he or she does approximately 2% of the time.
- The model predicts that the client will subscribe to a term deposit when he or she does not approximately 80% of the time.

Pruning

Both the rxDTree and rxDForest functions have pruning capability as part of the function call.

- One option is specifying cp, which is a numeric scalar that defines the complexity parameter. Any split that does not decrease the overall lack-of-fit by at least cp is not attempted, and by default it is set to 0.
- Alternately, pruneCp is another optional complexity parameter for pruning. By default it is set to 0, but if pruneCp is greater than 0, then prune.rxDTree is called on the completed tree with the specified pruneCp and the pruned tree is returned.

The difference between pruneCp and cp is that the latter determines which splits are considered in growing the tree.

Optimization of Tree-Building Parameters

As you may have experienced, the Tree and Forest functions are computationally intense. Here are some tips to speed up your processing time:

1. Prune your models using either the cp parameter to reject splits that do not meet the lack-of-fit standard while computing the model.
2. Partitioning your data prior to execution can improve computation time.
3. Decide whether a Tree or Random Forest model will better suit your needs.

Recap

Let’s review some of the concepts covered in this module:

- How does a random forest differ from a random tree?
- What is the OOB?
  - What does it stand for?
- How do you interpret the confusion matrix?

Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com, 1.855.GET.REVO, Twitter: @Revo-lutionR