Random Forest

36-290 – Statistical Research Methodology

Week 8 Tuesday – Fall 2021

Motivation

We have previously learned about decision trees. While trees are interpretable, they do have issues:

- They are highly variable. For instance, if you split a dataset in half and grow trees for each half, they can look very different.
- (Related:) They do not generalize as well as other models, i.e., they tend to have higher test-set MSE values.

To counteract these issues, one can utilize *bootstrap aggregation*, or *bagging*. Let's break this down:

- bootstrap: sample the training data *with replacement*
- aggregation: aggregate many trees, each constructed with a different bootstrap sample of the original training set

Bagging: Algorithm

```
Specify number of trees: n

For each tree 1->n:
  - construct a bootstrap sample from the training data
  - grow a deep and unpruned tree (i.e., overfit!)

Pass test datum through all n trees:
  - if regression: overall prediction is average of those for each tree
  - if classification: by default, predicted class for each tree is the
                       most represented class in the leaf; overall prediction
                       is majority vote
```

Bagging improves prediction accuracy at the expense of interpretability: one can "read" a single tree, but if you have 500 trees, what can you do?

$\Rightarrow$ You can use a metric dubbed *variable importance*, which is the average improvement in, e.g., RSS or Gini when splits are made on a particular predictor variable. The larger the average improvement, the more important the predictor variable.

Random Forest: Algorithm

The random forest algorithm *is* the bagging algorithm, with a tweak.

For each bootstrapped sampled dataset, we randomly select a subset of the predictor variables, and we build the tree using only those variables. By default, $m = \sqrt{p}$. (If $m = p$, then we recover the bagging algorithm.)

$\Rightarrow$ Selecting a variable subset for each tree allows us to get around the issue that if there is a dominant predictor variable, the first split is (almost always) made on that predictor variable. (Subsetting acts to "decorrelate" the different trees.)

Random Forest: Variable Importance

	Regression	Classification	Preferred?
%IncMSE	YES	NO	✓
IncNodePurity	YES	NO	
MeanDecreaseGini	NO	YES	
MeanDecreaseAccuracy	NO	YES	✓

In order to get the preferred plots, one needs to specify `importance=TRUE` as an argument in the call to `randomForest()`.

The "percent increase in mean-squared error" algorithm:

- Grow forest. Using "out-of-bag" (or OOB) data, compute the MSE. Call this MSE.min. (Out-of-bag data are the data that are left out of tree building, for each tree in the forest.)
- For each predictor variable in turn, randomly permute the data values, then repeat Step 1. Call this MSE.ii.
- %IncMSE for the ii^th variable is 100*(MSE.ii-MSE.min)/(MSE.min). The percentage increase does not depend on sample size.

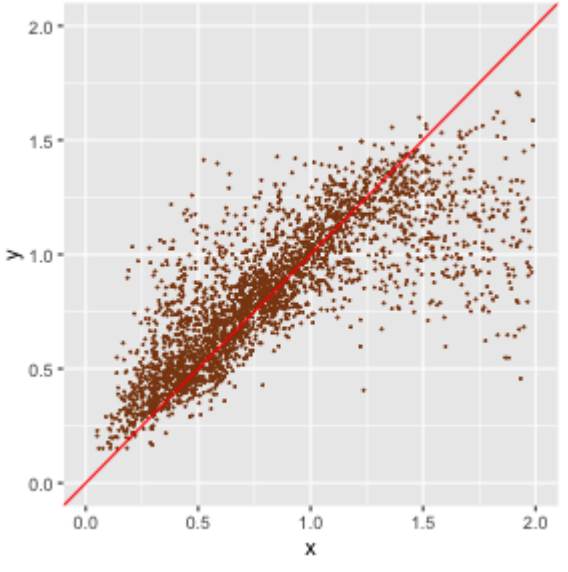
The mean decrease in accuracy is computed similarly:

- How many more OOB observations are misclassified if we randomly permute the data in the named data frame column, as opposed to not permuting the data?
- However, unlike for %IncMSE, the magnitude of the decrease in accuracy will depend on sample size.

Random Forest: Regression Example

Our example dataset contains 10000 galaxies, of which 7000 will be used for training the random forest model and 3000 will be used to assess it. The predictors are u-, g-, r-, i-, z-, and y-band magnitudes (brightnesses from the near-ultraviolet through the optical to the near-infrared) and the response is redshift, a measure of the distance of the galaxies from Earth. Note: there is extreme multicollinearity in this dataset, but because we are trying to predict redshift, it doesn't matter.

```
suppressMessages(library(tidyverse)) ; suppressMessages(library(randomForest))
rf.out = randomForest(resp.train~.,data=pred.train,importance=TRUE)
resp.pred = predict(rf.out,newdata=pred.test)
ggplot(data=data.frame("x"=resp.test,"y"=resp.pred),mapping=aes(x=x,y=y)) +
  geom_point(size=0.1,color="saddlebrown") + xlim(0,2) + ylim(0,2) +
  geom_abline(intercept=0,slope=1,color="red")
```

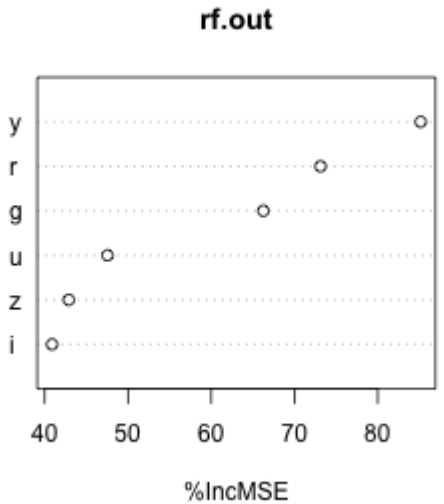


Random Forest: Regression Example

```
round(mean((resp.pred-resp.test)^2),3)
```

```
## [1] 0.069
```

```
varImpPlot(rf.out,type=1)  # type=1 means "only show %IncMSE plot
```



Variable importance plots provide the extent to which we can do inference with a random forest model. The plot above indicates that y-, r-, and g-band magnitudes are the more important variables for predicting redshift...however, note the extent of the x axis (the minimum value is not zero, so the i band contains far more information than you would initially think).

Random Forest: Classification

There is no difference in how random forest is run for classification analyses, except in how one generates predictions.

If you are content to simply predict by majority vote (i.e., use a class-separation threshold of 0.5), then you could do the following:

```
resp.pred = predict(rf.out,newdata=pred.test,type="response")
```

This will output classes directly. On the other hand, if you want to specify the class-separation threshold:

```
resp.prob = predict(rf.out,newdata=pred.test,type="prob")[,2]
resp.pred = ifelse(resp.prob>0.5,"<class 1>","<class 0>")
```

Why the `[,2]`? Because unlike `predict.glm()`, `predict.randomForest()` returns a matrix of probabilities. If there are two levels to the response variable, then `[,2]` gives the probability that the test-set object is associated with class 1.