Intro to Ensemble Learning in R

19 JAN 2012 on machine learning and r

Introduction

This post incorporates parts of yesterday's post about bagging. If you are unfamiliar with bagging, I suggest that you read it before continuing with this article.

I would like to give a basic overview of ensemble learning. Ensemble learning involves combining multiple predictions derived by different techniques in order to create a stronger overall prediction. For example, the predictions of a random forest, a support vector machine, and a simple linear model may be combined to create a stronger final prediction set. The key to creating a powerful ensemble is model diversity. An ensemble with two techniques that are very similar in nature will perform more poorly than a more diverse model set.

Some ensemble learning techniques, such as Bayesian model combination and stacking, attempt to weight the models prior to combining them. I will save the discussion of how to use these techniques for another day, and will instead focus on simple model combination.

Initial Setup

We will begin with similar variables to those that I used yesterday in the introduction to bagging. However, I have altered x2 and x3 to introduce distinct nonlinear tendencies, in order to evaluate the performance of nonlinear learning techniques.

```
set.seed(10)
y<-c(1:1000)
x1<-c(1:1000)*runif(1000,min=0,max=2)
x2<-(c(1:1000)*runif(1000,min=0,max=2))^2
x3<-log(c(1:1000)*runif(1000,min=0,max=2))

lm_fit<-lm(y~x1+x2+x3)
summary(lm_fit)
```

As you can see, the R-squared value is now .6658, indicating that the predictors have less of a linear correlation to the dependent variable than the predictors from the bagging intro yesterday.

```
set.seed(10)
all_data<-data.frame(y,x1,x2,x3)
```

```
positions <-
sample(nrow(all_data),size=floor((nrow(all_data)/4)*3))
training<- all_data[positions,]
testing<- all_data[-positions,]

lm_fit<-lm(y~x1+x2+x3,data=training)
predictions<-predict(lm_fit,newdata=testing)
error<-sqrt((sum((testing$y-predictions)^2))/nrow(testing))
error
```

Dividing the data into training and testing sets and using a simple linear model to make predictions about the testing set yields a root mean squared error of 177.36.

```
library(foreach)
length_divisor<-6
iterations<-5000
predictions<-foreach(m=1:iterations,.combine=cbind) %do% {
training_positions <- sample(nrow(training),
size=floor((nrow(training)/length_divisor)))
train_pos<-1:nrow(training) %in% training_positions
lm_fit<-lm(y~x1+x2+x3,data=training[train_pos,])
predict(lm_fit,newdata=testing)
}
predictions<-rowMeans(predictions)
error<-sqrt((sum((testing$y-predictions)^2))/nrow(testing))
error
```

Creating 5000 simple linear models and averaging the results creates a prediction error of 177.2591, which is marginally superior to the initial results.

Creating the First Ensemble

To create our initial ensemble, we will use the linear model, and a random forest. A random forest can be a powerful learning technique. It creates multiple decision trees from randomized sets of predictors and observations. It will be less useful here, as we only have three predictors, but will still illustrate the general point.

Now, we will test the efficacy of a random forest on our data:

```
library(randomForest)
rf_fit<-randomForest(y~x1+x2+x3,data=training,ntree=500)
predictions<-predict(rf_fit,newdata=testing)
error<-sqrt((sum((testing$y-predictions)^2))/nrow(testing))
error
```

My error value came to 134.98 with the above code. As you can see, random forests can make much better predictions than linear models. In this case, the linear model could not deal with the nonlinear predictors, whereas the random forest could. One should beware the trap of overfitting, however, as while random forests are much less prone to it than a single decision tree, it can still occur in very noisy data.

Note that a random forest already incorporates the idea of bagging into the basic algorithm, so we will gain little to nothing by running a random forest through our bagging function. What we will do instead is this:

```
length_divisor<-6
iterations<-5000
predictions<-foreach(m=1:iterations,.combine=cbind) %do% {
training_positions <- sample(nrow(training),
size=floor((nrow(training)/length_divisor)))
train_pos<-1:nrow(training) %in% training_positions
lm_fit<-lm(y~x1+x2+x3,data=training[train_pos,])
predict(lm_fit,newdata=testing)
}
lm_predictions<-rowMeans(predictions)

library(randomForest)
rf_fit<-randomForest(y~x1+x2+x3,data=training,ntree=500)
rf_predictions<-predict(rf_fit,newdata=testing)
predictions<-(lm_predictions+rf_predictions)/2
error<-sqrt((sum((testing$y-predictions)^2))/nrow(testing))
error
```

This combines the results of the linear model and the random forest using equal weights. Not surprisingly, considering that the linear model is weaker than the random forest, we end up with an error of 147.97. While this is not a fantastic result, it does illustrate ensemble learning.

Improving the Performance of Our Ensemble

From here, we have two options. We can either combine the predictions of the linear model and the random forest in different ratios until we achieve a better result(I would do this with caution in the real world, and use one hold out set to find the optimal weights, and test the weightings on another hold out set, as this can lead to overfitting if done improperly), or we can replace the linear model with a better one. We will try both, but first we will try combining the results in different ratios. Our prior knowledge of the error rates tells us to use a small ratio for the linear model.

```
predictions<-(lm_predictions+rf_predictions*9)/10
error<-sqrt((sum((testing$y-predictions)^2))/nrow(testing))
error
```

This results in an error rate of 136.23, which is not an improvement on the random Forest alone. Next, we replace the linear model with a support vector machine(svm) from the e1071 package, which provides an R interface to libSVM. A support vector machine is based on some complicated mathematics, but is basically a stronger machine learning technique that can pick up nonlinear tendencies in the data, depending on what kind of kernel is used.

```
library(e1071)
svm_fit<-svm(y~x1+x2+x3,data=training)
svm_predictions<-predict(svm_fit,newdata=testing)
error<-sqrt((sum((testing$y-
svm_predictions)^2))/nrow(testing))
error
```

The error when using an svm is 129.87, which is superior to both the random forest and the linear models. Next we will try using the svm with our bagging function.

```
length_divisor<-6
iterations<-5000
predictions<-foreach(m=1:iterations,.combine=cbind) %do% {
training_positions <- sample(nrow(training),
size=floor((nrow(training)/length_divisor)))
train_pos<-1:nrow(training) %in% training_positions
svm_fit<-svm(y~x1+x2+x3,data=training[train_pos,])
predict(svm_fit,newdata=testing)
}
svm2_predictions<-rowMeans(predictions)
error<-sqrt((sum((testing$y-
svm2_predictions)^2))/nrow(testing))
error
```

The error with 5000 iterations of an svm model is 141.14. In this case, it appears that the svm performs better without bagging techniques. It may be that there is too little data for it to be effective when used like this. However, the time it takes an svm increases exponentially(I believe) with more observations, so sometimes various techniques, including reduction in the number of observations or features, will need to be performed to improve svm performance to tolerable levels. Going forward, we will use the results of the single svm for the rest of this article.

```
predictions<-(svm_predictions+rf_predictions)/2
error<-sqrt((sum((testing$y-predictions)^2))/nrow(testing))
error
```

When we equally combine the svm predictions from the single model with the random forest predictions, we get an error rate of 128.8, which is superior to either the svm model alone, or the random forest model alone.

```
predictions<-(svm_predictions*2+rf_predictions)/3
error<-sqrt((sum((testing$y-predictions)^2))/nrow(testing))
error
```

If we tweak the ratios to emphasize the stronger svm model, we lower our error to 128.34.

Conclusion

As we have seen, ensemble learning can outperform any one single model when used properly. A good exercise to continue on from here would be to see if other models can improve performance when added to the ensemble. Please feel free to email me or leave a comment if you see something incorrect or have any questions.

Vik Paruchuri

Data scientist and developer in Somerville, MA. Working on DataQuest. Get in touch @vikparuchuri, or read more about me.

Share this post