Practical Machine Learning Course Project

Marty

Friday, August 21, 2015

Bottom Line Up Front (BLUF)

For this project I was given a dataset consisting of 19622 instances of 160 variables relating to personal activity monitoring. One of these variables was the "classe" variable, which depicted the manner in which the exercise was done (i.e., how well the subject did the exercise). After pre-processing the data, I ran some basic models to practice using machine learning algorithms on a larger dataset. The first model I ran (a random forest procedure) actually resulted in over a 95% out-of-sample accuracy rate, so I decided to try this model at predicting the test cases. It turned out that all 20 test cases were predicted accurately. I looked at some other machine learning algorithms, but none did any better, nor did adding more folds to the cross-validation procedure. In the end, a 95% accuracy rate and a perfect prediction score for the test cases seemed pretty good.

Pre-Processing the Data

The original training dataset consisted of 19622 rows (instances) and 160 columns (variables). I imported the data into Excel to do some exploratory data analysis and quickly discerned that 100 of the variables had significant amounts of missing data. Looking at the test set (which had 20 rows and 159 variables; it was just missing the *classe* variable), I confirmed that it too had 100 variables with significant missing data. So I removed these 100 variables along with the first 5 columns (which just contained date and time window information) and was left with 55 usable variables. I exported the results into a .csv file which I then imported into `R` for analysis; this file is called *JustData.csv.* Once I had the data imported, I split it into a training set and a testing set. To start, I only included 10% of the data in the training set (as opposed to 70% as suggested by the course slides) so that my training algorithms wouldn't take forever to run. I originally intended to increase this in order to get a better fit, but it turned out that I got a really good fit using just 10% of the data, so I never bothered going back and increasing the size of the training set. The code to accomplish all this is below:

```r
# load the appropriate packages

library(caret)
library(randomForest)
library(gbm)

# load just the data with columns
#  I did my adjusting in Excel and then saved this as a .csv file

JustDataCols <- read.csv("JustData.csv")

# partition the data

inTrain <- createDataPartition(y = JustDataCols$classe, p = 0.1, list = FALSE)
training <- JustDataCols[inTrain,]
testing <- JustDataCols[-inTrain,]
```

Random Forest Algorithm from Caret

The caret package contains lots of algorithms that allow for classification prediction, so that is what I used to do my analysis. Another nice thing about the caret package is that it handles cross-validation for you, plus it allows you to specify the tuning parameters. Again, I was looking at processing speed, so I selected just a k-fold cross-validation procedure with k = 3 as my initial try. It turned out that this gave me really good results, so I never bothered trying more folds. After setting up the tuning parameters, I went ahead and trained my model using the random forest algorithm on the training set, then predicted the outcomes on the testing set, and finally calculated an overall out-of-sample accuracy rate. Here's the code to do that:

```r
fitControl <- trainControl(method = "cv",
                           number = 3)

ptm <- proc.time()
RFFit <- train(training$classe ~ .,
               data = training,
               method = "rf",
               trControl = fitControl,
               verbose = FALSE)
elapsed <- proc.time() - ptm
RFTime <- round(elapsed[3], 2)
RFPred <- predict(RFFit, testing)
RFAcc <- 100*round(confusionMatrix(testing$classe, RFPred)$overall['Accuracy'], 4)
```

This algorithm resulted in an out-of-sample prediction accuracy of 96.94% and took 38.9 seconds to train the model. Not bad considering I only used 10% of the data to train the model and 3 folds in my k-fold cross validation.

Boosted Tree Model Algorithm from Caret

Although the random forest method gave me really good predictions (perfect results for the 20 test cases as a matter-of-fact), I decided to try one more prediction algorithm; namely boosted tree models (gbm in caret). Boosted tree models are even more computationally intensive, so I kept the 10% training set size. I didn't mess with the tuning parameters this time and just went ahead and trained my model using the gbm algorithm on the training set, then predicted the outcomes on the testing set, and finally calculated an overall out-of-sample accuracy rate. Here's the code to do that:

```r
ptm <- proc.time()
GBMFit <- train(training$classe ~ .,
                data = training,
                method = "gbm",
                verbose = FALSE)
```

```
## Loading required package: plyr
```

```
## Warning: package 'plyr' was built under R version 3.1.2
```

```r
elapsed <- proc.time() - ptm
GBMTime <- round(elapsed[3], 2)
GBMPred <- predict(GBMFit, testing)
GBMAcc <- 100*round(confusionMatrix(testing$classe, GBMPred)$overall['Accuracy'], 4)
```

This algorithm resulted in an out-of-sample prediction accuracy of 95.98% and took 219.68 seconds to train the model. No real difference in out-of-sample accuracy, but a significant increase in training time. In the end, I didn't think it was worth pursuing boosted tree models more because they didn't give me any better results and took a lot of time to tell me that!

Random Forest Model Directly from RandomForest Package

Algorithm processing time was a major issue with this exercise, so I took some advice from the discussion forums and decided to forego using the caret package and try running the random forest algorithm directly from its base package. This method avoids all the overhead that caret brings with it but also doesn't give me access to all the control (especially with respect to cross-validation) that caret provides. Since I just wanted to see how fast the algorithm would run, I didn't concern myself with that and just used the randomforest function on the training set, then predicted the outcomes on the testing set, and finally calculated an overall out-of-sample accuracy rate. Here's the code to do that:

```
ptm <- proc.time()
RandForFit <- randomForest(training$classe ~ ., data = training)
elapsed <- proc.time() - ptm
RandForTime <- round(elapsed[3], 2)
RandForPred <- predict(RandForFit, testing)
RandForAcc <- 100*round(confusionMatrix(testing$classe, RandForPred)$overall['Accuracy'], 4)
```

This algorithm resulted in an out-of-sample prediction accuracy of 96.31% and took a mere 4.92 seconds to train the model. The accuracy was basically the same as that from the caret package's random forest algorithm, but the processing speed was **much** faster! So, if I really have a big model and want fast processing speed, I would probably code up the cross-validation myself and then run the algorithm directly from the base package.

Predicting Outcomes on the Test Set

We were given a *true* test set (where we didn't know the value of the *classe* variable) and asked to run our best model on it to predict 20 cases. The code to do that, and put the results in individual text files, is below:

```
# Make predictions on the Test Set

TestSetData <- read.csv("TestSet.csv")
TestSetPred <- predict(RFFit, TestSetData)
TestSetPred
```

```
##  [1] B A B A A E D B A A B C B A E E A B B B
## Levels: A B C D E
```

```
# Function for writing the files

pml_write_files = function(x){
  n = length(x)
  for(i in 1:n){
    filename = paste0("problem_id_",i,".txt")
    write.table(x[i],file=filename,quote=FALSE,row.names=FALSE,col.names=FALSE)
  }
}

pml_write_files(TestSetPred)
```

Once the 20 predictions were made, I loaded them on the assignment web-site and found out that all 20 were correct!

Conclusion

This project gave me a chance to try out different machine learning algorithms in a practical exercise to predict how well 20 personal activity monitored exercises were conducted. I ran two models, random forests and bosted random forests, on a training set of data and then calculated the out-of-sample accuracy rates on a testing set of data. In the end, I achieved an accuraccy rate of over 95% and predicted all 20 unknown cases perfectly. There are dozens, if not hundreds, of different prediction algorithms out there, but random forests seem to be among the most popular and most accurate. My 95% accuracy rate and perfect prediction result on the test data seems to support this notion of random forest's popularity and accuracy.