Predictions on Weight Lifting Exercises

Simon Coplan

Sunday, April 26, 2015

Introduction I decided to use a random forest model in order to predict the classe variable based on significant sensor variables. I thought about using a tree algorithm but looking at the data set and possible number of predictors, I concluded that a random forest, although somewhat time consuming, would be better.

Analysis I loaded the training and test sets into R; I already had downloaded the two csv files and put them in my working directory.

In order to find the variables to use, I examined the training data set and found many superfluous variables. I decided to use the more obvious quantitative variables associated with roll, pitch, yaw, accelleration, and gyros for each sensor location: belt, arm, dumbbell, and forearm.

I assigned each sensor variables to individual data sets, and used the findCorrelation function from the caret package on each data set to eliminate variables that were correlated to one another. Once I removed intra-sensor correlated variables, I combined all the sensor variables into one data set and removed inter-sensor correlated variables. Finally, I created a data set containing all of the chosen sensor variables and the outcome variable classe.

```
library(plyr)
library(dplyr)
library(caret)
library(randomForest)

pmltraining<-read.csv(file = "pml-training.csv")
pmltesting<-read.csv(file = "pml-testing.csv")

set.seed(2009)

pmltrainbelt<-select(pmltraining, roll_belt, pitch_belt, yaw_belt, gyros_belt_x, gyros_belt_y, gyros_bel

pmltrainarm<-select(pmltraining, roll_arm, pitch_arm, yaw_arm, gyros_arm_x, gyros_arm_y, gyros_arm_z, ac

pmltraindumbbell<-select(pmltraining, roll_dumbbell, pitch_dumbbell, yaw_dumbbell, gyros_dumbbell_x, gyr

pmltrainforearm<-select(pmltraining, roll_forearm, pitch_forearm, yaw_forearm, gyros_forearm_x, gyros_fo

pmltrainbelt<-select(pmltrainbelt, -findCorrelation(cor(pmltrainbelt)))

pmltrainarm<-select(pmltrainarm, -findCorrelation(cor(pmltrainarm)))

pmltraindumbbell<-select(pmltraindumbbell, -findCorrelation(cor(pmltraindumbbell)))

pmltrainsensors<-cbind(pmltrainbelt, pmltrainarm, pmltraindumbbell, pmltrainforearm)

pmltrainsensors<-select(pmltrainsensors, -findCorrelation(cor(pmltrainsensors)))

pmltraintotal<-cbind(pmltrainsensors, classe = pmltraining[,160])

pmltraintotal<-as.data.frame(pmltraintotal)
```

After multiple attempts, I found that my computer did not have enough memory to use the entire training data set to create the prediction model. So, I used caret's createDataPartition function to create a "pre-testing" and "pre-training" data set within the original training set, using 50% of the original training data to create my random forest model. 50% was arbitrarily chosen, but it worked in the randomForest function and didn't shoot back any errors.

This unfortunate but necessary diversion also gave me the change to see how a model would work on a "test" data set (remember, I created a pre-testing set using 50% the original training data).

```
intrain<-createDataPartition(y = pmltraintotal$classe, p = .5, list = F)

pmltraintotalTRAIN<-pmltraintotal[intrain,]

pmltraintotalTEST<-pmltraintotal[-intrain,]

pmlmodelrf<-randomForest(classe~., data = pmltraintotalTRAIN,importance = TRUE, proximity = TRUE)
```

After creating the randomForest model, I wanted to see how well the model would predict on the classe variable. So, I used half of the original training set as a test set (the code for which is shown and explained above) and predicted the classe variable. Shown below is the confusion matrix, from which I found my out of sample error.

```
trainTESTprediction<-predict(pmlmodelrf, newdata = pmltraintotalTEST)

confusionMatrix(trainTESTprediction, pmltraintotalTEST$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 2786   18    0    0    0
##          B    2 1869   25    0    0
##          C    2   11 1685   25    4
##          D    0    0    1 1580    3
##          E    0    0    0    3 1796
##
## Overall Statistics
##
##                Accuracy : 0.9904
##                  95% CI : (0.9883, 0.9922)
##     No Information Rate : 0.2844
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9879
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9986   0.9847   0.9848   0.9826   0.9961
## Specificity            0.9974   0.9966   0.9948   0.9995   0.9996
## Pos Pred Value         0.9936   0.9858   0.9757   0.9975   0.9983
## Neg Pred Value         0.9994   0.9963   0.9968   0.9966   0.9991
## Prevalence             0.2844   0.1935   0.1744   0.1639   0.1838
## Detection Rate         0.2840   0.1905   0.1718   0.1611   0.1831
## Detection Prevalence   0.2858   0.1933   0.1760   0.1615   0.1834
## Balanced Accuracy      0.9980   0.9907   0.9898   0.9910   0.9979
```

I assumed that the out of sample error was 1-Accuracy, which leads me with an out of sample error of approximately 1-.9904 = .0096 (which is decent)

Finally, I subsetted the true testing data set to match the training set variables, and used the predict function to predict the classe (you'll notice that I used the random forest model that only used half of the original training data; remember, this was because my computer did not have enough memory to use all of the . I am not showing the code, but wanted to explain that I did the prediction on the testing data.