Project Practical Machine Learning

Parama Bhattacharya

Sunday, December 27, 2015

Background

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement-a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Data Preparation

In this section, load the data and the 20 cases that will be submitted to coursera.

```
rm(list = ls())
if (!file.exists("pml-training.csv")) {
  download.file("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv", destfile = "pml-
}
if (!file.exists("pml-testing.csv")) {
  download.file("http://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv", destfile = "pml-te
}
submit <- read.csv("pml-testing.csv", sep = ",", na.strings = c("", "NA"))
data <- read.csv("pml-training.csv", sep = ",", na.strings = c("", "NA"))
```

Data Cleansing

Here, remove columns containing NAs and remove features that are not in the submit set. The features containing NAs are the variance, mean and stddev(standard deviation) within each window for each feature. Since the submit dataset has no time-dependence, these values are useless and can be disregarded. Also remove the first 7 features since they are related to the time-series and are not numeric.

```
# Remove columns full of NAs.
features <- names(submit[,colSums(is.na(submit)) == 0])[8:59]
# Only use features used in submit cases.
data <- data[,c(features,"classe")]
submit <- submit[,c(features,"problem_id")]
```

Bootstrap

Next, withhold 25% of the dataset for testing after the final model is constructed.

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.1.3
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 3.1.3
```

```
## Loading required package: ggplot2
```

```
## Warning: package 'ggplot2' was built under R version 3.1.3
```

```
set.seed(916)
inTrain = createDataPartition(data$classe, p = 0.75, list = F)
training = data[inTrain,]
testing = data[-inTrain,]
```

Feature Selection

Some features may be highly correlated. The PCA method mixes the final features into components that are difficult to interpret; instead, drop features with high correlation (>90%).

```
outcome = which(names(training) == "classe")
highCorrCols = findCorrelation(abs(cor(training[,-outcome])),0.90)
highCorrFeatures = names(training)[highCorrCols]
training = training[,-highCorrCols]
outcome = which(names(training) == "classe")
```

The features with high correlation are accel_belt_z, roll_belt, accel_belt_y, accel_belt_x, gyros_arm_y, gyros_forearm_z, and gyros_dumbbell_x.

Feature Significance

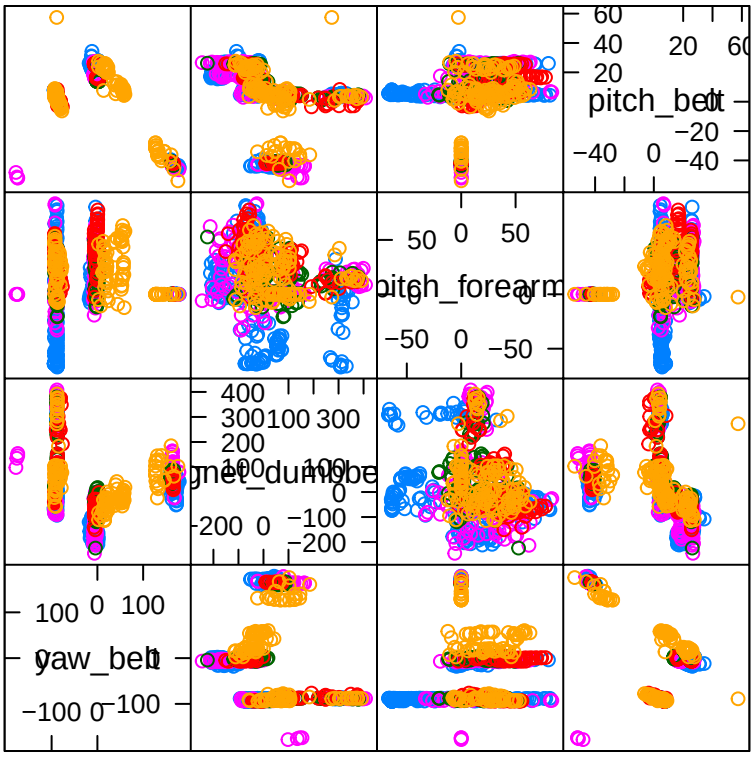
The random forest method reduces overfitting and is good for nonlinear features. First, to see if the data is nonlinear, I use the random forest to discover the most important features. The feature plot for the 4 most important features is shown.

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.1.3
```

```
## randomForest 4.6-12
## Type rfNews() to see new features/changes/bug fixes.
```

```
fsRF = randomForest(training[,-outcome], training[,outcome], importance = T)
rfImp = data.frame(fsRF$importance)
impFeatures = order(-rfImp$MeanDecreaseGini)
inImp = createDataPartition(data$classe, p = 0.05, list = F)
featurePlot(training[inImp,impFeatures[1:4]],training$classe[inImp], plot = "pairs")
```



Scatter Plot Matrix

The most important features are:

pitch_belt yaw_belt total_accel_belt gyros_belt_x

Train the Prediction Model

Train using the random forest and k-nearest neighbors for comparison.

```
ctrlKNN = trainControl(method = "adaptive_cv")
modelKNN = train(classe ~ ., training, method = "knn", trControl = ctrlKNN)
ctrlRF = trainControl(method = "oob")
modelRF = train(classe ~ ., training, method = "rf", ntree = 200, trControl = ctrlRF)
resultsKNN = data.frame(modelKNN$results)
resultsRF = data.frame(modelRF$results)
```

The random forest will give a larger accuracy compared to k-nearest neighbors. Give the confusion matrix between the KNN and RF models to see how much they agree on the test set, then compare each model using the test set outcomes.

```
fitKNN = predict(modelKNN, testing)
fitRF = predict(modelRF, testing)
```

KNN Versus RF

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1358    8   11   17    8
##          B   41  832   28   23   25
##          C   14   29  781   22    9
##          D    9    3   55  722   14
##          E   13   36   23   38  785
##
## Overall Statistics
##
##                Accuracy : 0.9131
##                  95% CI : (0.9049, 0.9209)
##     No Information Rate : 0.2926
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.89
##  Mcnemar's Test P-Value : 4.829e-12
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9463   0.9163   0.8697   0.8783   0.9334
## Specificity            0.9873   0.9707   0.9815   0.9802   0.9729
## Pos Pred Value         0.9686   0.8767   0.9135   0.8991   0.8771
## Neg Pred Value         0.9780   0.9808   0.9711   0.9756   0.9860
## Prevalence             0.2926   0.1852   0.1831   0.1676   0.1715
## Detection Rate         0.2769   0.1697   0.1593   0.1472   0.1601
## Detection Prevalence   0.2859   0.1935   0.1743   0.1637   0.1825
## Balanced Accuracy      0.9668   0.9435   0.9256   0.9293   0.9532
```

KNN Versus Test Set

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1354    5   11   17    8
##          B   46  832   26   23   22
##          C   13   32  780   19   11
##          D    9    3   58  725    9
##          E   13   36   23   38  791
##
## Overall Statistics
##
##                Accuracy : 0.9139
##                  95% CI : (0.9057, 0.9217)
##     No Information Rate : 0.2926
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8911
##  Mcnemar's Test P-Value : < 2.2e-16
##
```

```
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9436   0.9163   0.8686   0.8820   0.9405
## Specificity            0.9882   0.9707   0.9813   0.9806   0.9729
## Pos Pred Value         0.9706   0.8767   0.9123   0.9017   0.8779
## Neg Pred Value         0.9769   0.9808   0.9709   0.9763   0.9875
## Prevalence             0.2926   0.1852   0.1831   0.1676   0.1715
## Detection Rate         0.2761   0.1697   0.1591   0.1478   0.1613
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9659   0.9435   0.9249   0.9313   0.9567
```

RF Versus Test Set

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    0    0    0    0
##          B    7  941    1    0    0
##          C    0    8  844    3    0
##          D    0    0   10  794    0
##          E    0    0    0    6  895
##
## Overall Statistics
##
##                Accuracy : 0.9929
##                  95% CI : (0.9901, 0.995)
##     No Information Rate : 0.2859
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.991
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9950   0.9916   0.9871   0.9888   1.0000
## Specificity            1.0000   0.9980   0.9973   0.9976   0.9985
## Pos Pred Value         1.0000   0.9916   0.9871   0.9876   0.9933
## Neg Pred Value         0.9980   0.9980   0.9973   0.9978   1.0000
## Prevalence             0.2859   0.1935   0.1743   0.1637   0.1825
## Detection Rate         0.2845   0.1919   0.1721   0.1619   0.1825
## Detection Prevalence   0.2845   0.1935   0.1743   0.1639   0.1837
## Balanced Accuracy      0.9975   0.9948   0.9922   0.9932   0.9993
```

Making the final Test Set Prediction

```
# predict on test set

finPred <- predict(modelRF, submit)
```

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