Predicting Type of Exercise Using Motion Data

Jerry C., Prctical Machine Learning - Course Project

Background

For this project, we are provided with data collected from "wearable technology" equipment worn by enthusiasts that record their movements via accelerometers on belts, forearms, arms and dumbells. The training set consists of multiple observations, different independent variables and one variable that identifies one of five different ways the dumbbells were lifted (variable `classe`). The goal is to use this training data set to build a model that can then be used to predict the type of dumbbell lift given data on the other independent variables for twenty observations in a test set.

The data are provided from http://groupware.les.inf.puc-rio.br/har.

Initial Set Up

First, we will do some basic set up: load required libraries, load the train and test sets and set the seed for reproducibility.

```
library(caret)
```

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

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

```
## Warning: package 'ggplot2' was built under R version 3.1.2
```

```
library(rpart)
library(randomForest)
```

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

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
```

```
orig_train <- read.csv("pml-training.csv", na.strings = c("NA", "#DIV/0!", ""))
orig_test <- read.csv("pml-testing.csv", na.strings = c("NA", "#DIV/0!", ""))

set.seed(2015)
```

Data Cleaning

There are a couple of things that we can do to clean up the data. First, we can delete the first column which is just the row index.

```
orig_train <- orig_train[c(-1)]
orig_test <- orig_test[c(-1)]
```

Then we can search for zero or near zero variance predictors, which (as their names suggest) are variables that have either unique or relatively few values and are therefore not likely to be useful predictors.

```
nzv_var <- nearZeroVar(orig_train, saveMetrics = TRUE)
nzv_var <- subset(nzv_var, nzv_var$nzv == "TRUE")
```

The function identified thirty-six variables that are near zero variance predictors; we will remove those variables from our data set.

```
nzv_var_string <- names(orig_train) %in% c("new_window", "kurtosis_yaw_belt", "skewness_yaw_belt", "ampl

orig_train <- orig_train[!nzv_var_string]
orig_test <- orig_test[!nzv_var_string]
```

A third way to clean up the data set is to eliminate variables that have too many observations that are NA (e.g., more than 50%).

```
orig_train2 <- orig_train
orig_test2 <- orig_test

for(i in 1:length(orig_train)){
   if(sum(is.na(orig_train[, i])) / nrow(orig_train) >= 0.50){
       for(j in 1:length(orig_train2)){
           if(length(grep(names(orig_train[i]), names(orig_train2)[j]) == 1)){
               orig_train2 <- orig_train2[, -j]
               orig_test2 <- orig_test2[, -j]
           }
       }
   }
}
```

Finally, let’s convert all the relevant variables to a single data type, and get rid of the first few columns that don’t contain any predictive data.

```
orig_train2[, 5:57] <- sapply(orig_train2[, 5:57], as.numeric)
orig_test2[, 5:57] <- sapply(orig_test2[, 5:57], as.numeric)

orig_train2 <- orig_train2[, 5:58]
orig_test2 <- orig_test2[5:57]
```

Building the Training and Cross Validation Sets

Now we can split the training set into both a training set and a smaller test set.

```
inTrain <- createDataPartition(orig_train2$classe, p = 0.75)[[1]]
training <- orig_train2[inTrain, ]
testing <- orig_train2[-inTrain, ]
dim(training)
```

```
## [1] 14718    54
```

```
dim(testing)
```

```
## [1] 4904   54
```

Modeling

The first method that we will try is the decision tree. The idea is to build a model off of the training data and then test its accuracy on the cross validation data (named here as `testing`). The out of sample error rate for this method will likely be higher than other methods, but we can't say for sure how high it will be.

```
fitDT <- train(classe ~., data = training, method = "rpart")
predDT <- predict(fitDT, testing)
confusionMatrix(predDT, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1247  374  388  352   88
##          B   23  337   30  132   67
##          C  122  238  437  285  193
##          D    0    0    0    0    0
##          E    3    0    0   35  553
##
## Overall Statistics
##
##                Accuracy : 0.5249
##                  95% CI : (0.5108, 0.5389)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3807
##  Mcnemar's Test P-Value : < 2.2e-16
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.8939  0.35511  0.51111   0.0000   0.6138
## Specificity            0.6575  0.93628  0.79304   1.0000   0.9905
## Pos Pred Value         0.5092  0.57216  0.34275      NaN   0.9357
## Neg Pred Value         0.9397  0.85817  0.88482   0.8361   0.9193
## Prevalence             0.2845  0.19352  0.17435   0.1639   0.1837
## Detection Rate         0.2543  0.06872  0.08911   0.0000   0.1128
## Detection Prevalence   0.4994  0.12011  0.25999   0.0000   0.1205
## Balanced Accuracy      0.7757  0.64570  0.65207   0.5000   0.8021
```

The accuracy rate of 52.49% is not awful, but it's certainly not the method to use if a grade depended on it. It's more or less a coin toss.

The second method we will try is random forest. This method is similar to the decision tree method, but more rigorous; it's like running decision tree many, many times and then taking the average of those runs. The out of sample error rate for this method should be much lower.

```r
fitRF <- randomForest(classe ~., data = training)
predRF <- predict(fitRF, testing)
confusionMatrix(predRF, testing$classe)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1394    2    0    0    0
##          B    0  947    1    0    0
##          C    0    0  854    4    0
##          D    0    0    0  800    1
##          E    1    0    0    0  900
##
## Overall Statistics
##
##                Accuracy : 0.9982
##                  95% CI : (0.9965, 0.9992)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9977
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9993   0.9979   0.9988   0.9950   0.9989
## Specificity            0.9994   0.9997   0.9990   0.9998   0.9998
## Pos Pred Value         0.9986   0.9989   0.9953   0.9988   0.9989
## Neg Pred Value         0.9997   0.9995   0.9998   0.9990   0.9998
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2843   0.1931   0.1741   0.1631   0.1835
## Detection Prevalence   0.2847   0.1933   0.1750   0.1633   0.1837
## Balanced Accuracy      0.9994   0.9988   0.9989   0.9974   0.9993
```

As expected, the accuracy is 99.82%. So a bit better than the decision tree method.

Note: The confusionMatrix output shows up normally on my local instance, but for some reason is lost upon upload to Github. To view the local output, please use the pdf version included in this repo.

Predicting the Test Data

Having seen the accuracy of random forest method, let's use it to predict the outcomes on our actual test data set.

```r
predRF_new <- predict(fitRF, orig_test2)
```

We can use the code provided to generate the text files for submission.

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

pml_write_files(predRF_new)
```