title: "PML Course Project"
author: "aoshotse"
date: "Sunday, October 26, 2014"
output: html_document

Introduction to Project

In this project, the focus is human activity recognition (HAR), whereby data is collected on human movement using such things as gyroscopes, accelerometers, etc. Possible application include context-aware systems, elderly monitoring, workout support, etc.

The data set used in this project concerns weight lifting exercises and how well participants performed the exercises. Participants performed 10 repetitions of Dumbbell Bicep Curls in five different classes:

A: Perfect form.
B: throwing elbows out
C: lifting halfway
D: lowering halfway
E: throwing hips out

The goal of this project is to take the data set containing the features of the classes of activities performed and build a machine learning model that can predict what class a new activity belongs to. That is, given a test set of new data with no class label, our model should be able to "predict the manner in which they did the exercise."

For further readings on the data set used: *Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013. Read more: http://groupware.les.inf.puc-rio.br/har#ixzz3HHeqeN9E*

Loading the Data

This bit of code loads the test and training data in from the working directory and loads all necessary libraries. The test set is untouched until the prediction by the final model.

```
setwd("C:/Users/BigAbe/Desktop/Work/MOOC/Practical Machine Learning/Course Project/AOPracticalMachineLearning")

#####################################
### Load Packages and Train Data ####
#####################################
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.1.1
```

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

```
## Warning: package 'ggplot2' was built under R version 3.1.1
```

```
library(ggplot2)
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 3.1.1
```

```
## Rattle: A free graphical interface for data mining with R.
## Version 3.3.0 Copyright (c) 2006-2014 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
library(ipred)
```

```
## Warning: package 'ipred' was built under R version 3.1.1
```

```
library(plyr)
```

```
## Warning: package 'plyr' was built under R version 3.1.1
```

```
library( GGally )
```

```
## Warning: package 'GGally' was built under R version 3.1.1
```

```
train1 <- read.csv("pml-training.csv")
thetest <- read.csv("pml-testing.csv") ## untouched until model is complete
```

Cleaning the Training Data

This code chunk scrubs the data set of all variables with greater than 50% missing values and preserves variable types of the original data set (except converts timestamp variable to date format).

The rationale for this is that variables with too many missing observations will not contribute much to any prediction models later developed, and may cause considerable increase in processing time as well as contribute noise to the data.

```
#########################
### Clean Train Data ####
#########################

## Convert Variable Types
train1$user_name <- as.factor(train1$user_name)
train1$new_window <- as.factor(train1$new_window)
train1$cvtd_timestamp <- as.Date(train1$cvtd_timestamp, format="%m/%d/%Y %H:%M")
train1$classe <- as.factor(train1$classe)

## Function to Remove Variables with >= 50% Missng Values
sumnans <- function(x){
  train2 <- 1:length(x[,1]) ## was train1
    j = 1
    namer <- vector()
    for (i in 1:length(colnames(x))){
        if (sum(is.na(x[,i])) <= 0.5*length(x[,1]) ) {
            j = j + 1
            train2 <- cbind(train2, x[,i])
            namer <- rbind(namer, colnames(x[i]))
            }
    }

    train2 <- train2[,-1]
    colnames(train2) <- namer
    return(train2)
}

train2 <- sumnans(train1)
train2 <- as.data.frame(train2)
# train2$user_name <- train1$user_name
# train2$new_window <- train1$new_window
# train2$cvtd_timestamp <- train1$cvtd_timestamp
train2$classe <- as.factor(train1$classe)
train2 <- na.omit(train2)
```

Repartitioning and Further Preprocessing (Prinicipal Components Analysis)

At this point, the training data set is re-partitioned into a smaller training set (train3, trainv) and a test set (test0, testv). This is so that the model can be evaluated on a "pre-test" set before used on the actual test set.

In addition to this, the new training set and test set are both preprocessed into principal components (linear combinations of variables that capture their entire variation) in an attempt to capture the complete variation in the data set with as few variables as possible. The advantages of this are that the data is less noisy when selecting only the top ranked principal components. Also, with such few resulting variables, any prediction models can work through the data at a much faster speed since the volume of data is greatly reduced with PCA.

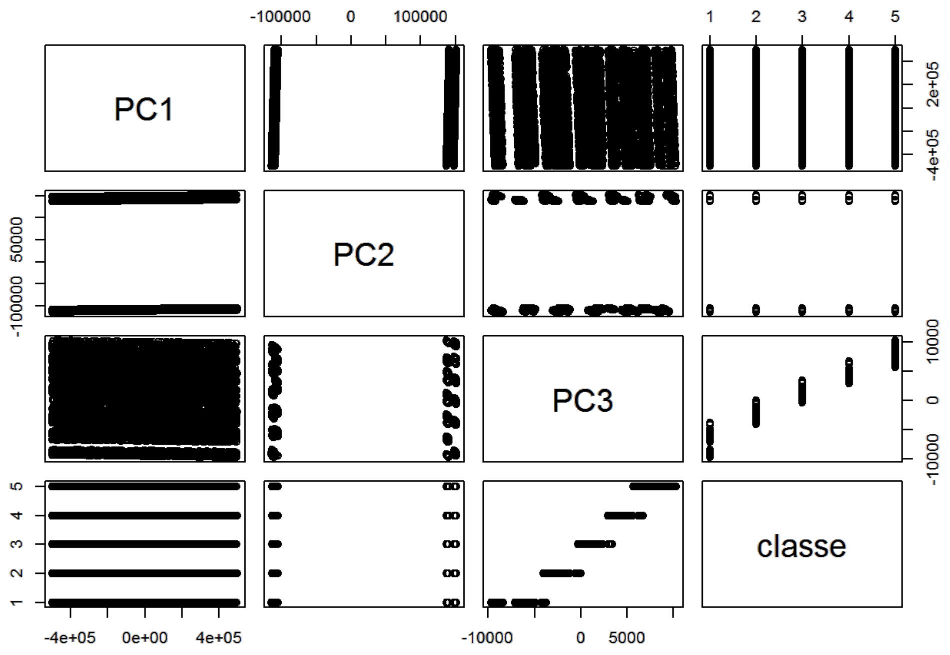
```
###########################################
### Re-Partition into Test0 and Train3 ####
###########################################
inTrain = createDataPartition(train2$classe, p = 0.85, list=FALSE)
train3 <- train2[inTrain,]
test0 <- train2[-inTrain,]
trainv <- train3[,-93]
testv <- test0[,-93]

############################
## Preprocessing with PCA ##
############################
## PCA pure train data set
trainz <- prcomp(trainv, tresh=0.9)
ztrain <- as.data.frame(trainz$x[,1:3])
ztrain$classe <- train3$classe

## PCA pure test data set
testz <- prcomp(testv, tresh=0.9)
ztest <- as.data.frame(testz$x[,1:3])
ztest$classe <- test0$classe
```

Plotting Predictors

At this point, it may be a good idea to take a look at how our resulting principal components relate to one another and relate to the response variable, "classe" (for class of activity). The following code plots the variables in a pairs plot. As is evident, the PC's seem to be excellent predictors of the class, as they are stratified along class categories. Also, they are not at all highly correlated with one another, so there is no redundancy in their predictive usefulness.

Just out of curiosity, I decided to make a linear model of the PC's as predictor variables and a numeric version of the classe variable as a response. The summary of the GLM shows high significance in the estimation of the coefficients. I would take this as further indication of the predictive strength of the chosen PC's.

```
##########################
## Plotting Predictors ##
##########################
pairs(ztrain)
```



```
gg <- glm(I(as.numeric(classe)) ~., data=ztrain)
summary(gg)
```

```
##
## Call:
## glm(formula = I(as.numeric(classe)) ~ ., data = ztrain)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -0.8438  -0.2434  -0.0626   0.3532   0.7495
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.80e+00   3.30e-03  847.75  < 2e-16 ***
## PC1         1.06e-07   1.14e-08    9.28  < 2e-16 ***
## PC2         2.07e-07   2.62e-08    7.91  2.7e-15 ***
## PC3         2.52e-04   5.82e-07  432.97  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.1217)
##
##     Null deviance: 24186.9  on 11179  degrees of freedom
## Residual deviance:  1359.8  on 11176  degrees of freedom
## AIC: 8184
##
## Number of Fisher Scoring iterations: 2
```

Cross Validation

Taking the second train set (the one that is a result of further partitioning the first training set), 4-fold cross validation was done so as to be able to estimate an "out-of-sample-error-rate". The method used for prediction was the *random forest* technique. The error rates for each fold are stored in a variable called **er.** The mean of this variable is the out-of-sample-error-rate for this study.

```
#######################################
### 4-Fold Cross Validation Ztrain ####
#######################################
## Data set shuffled and split into 4 sets
## Model tested in 4 "folds"
## Out of sample error rate averaged bewteen them
train22 <- ztrain[sample(nrow(ztrain)),]
set1 <- train22[1:2795,]
set2 <- train22[2796:5590,]
set3 <- train22[5591:8385,]
set4 <- train22[8386:11180,]
sets <- list(set1, set2, set3, set4)
ooser <- c(0,0,0,0,0)
for (i in 1:length(sets)){
  x <- as.data.frame(sets[i])
  v <- sets[-i]
  cc <- as.data.frame(v[1])
    cc <- rbind(cc, as.data.frame(v[2]))
    cc <- rbind(cc, as.data.frame(v[3]))
    fit <- train(classe~.,method="rf", data=cc)
    pred2 <- predict(fit, newdata=x)
    ooser[i] <- 1 - confusionMatrix(x$classe, pred2)$overall[1]
    ## print(ooser[i])
}
```

```
## Loading required package: randomForest
```

```
## Warning: package 'randomForest' was built under R version 3.1.1
```

```
## randomForest 4.6-10
## Type rfNews() to see new features/changes/bug fixes.
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
##
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
##
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
##
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

Testing on Pre-Test Set

Next, the second training set is tested against the pre-test set using the same random forest model. The error for this prediction is added to the er vector, and the final average is given below as the expected out of sample error rate based on this prediction model.

```
#####################################
## Test Model on in-Train Test Set ##
#####################################
fit1 <- train(classe~.,method="rf", data=ztrain)
```

```
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

```
pred11 <- predict(fit1, newdata=ztest)
ooser[5] <- 1 - confusionMatrix(ztest$classe, pred11)$overall[1]
er <- 100 * mean(ooser)
er ## expected Out of Sample error rate ##
```

```
## [1] 19.75
```

Preparing Final Train Data and Final Test Data for Prediction

At this stage, the entire training test data set (including removal of missing values and preprocessing into PC's) is put together into one data set (thetrain) and the untouched final test set loaded in as "thetest" is preprocessed in the same manner that the train set was processed in.

```
#############################################
## Prepare Final Train Data for Predicting ##
#############################################
thetrain <- rbind(ztrain, ztest)


###############################################
## Preprocess Final Test Data for Predicting ##
###############################################
## Reduce test set to same variables as train set
#thetest3 <- subset(thetest, select=(names(testv)))
## Convert data types
#thetest3$user_name <- as.numeric(thetest3$user_name)
#thetest3$new_window <- as.numeric(thetest3$new_window)
#thetest3$cvtd_timestamp <- as.numeric(thetest3$cvtd_timestamp)
thetest2 <- sumnans(thetest)
thetest2 <- as.data.frame(thetest2)
thetest2 <- na.omit(thetest2)
## Perform PCA
thetest3 <- prcomp(thetest2, tresh=0.9)
thetest3 <- as.data.frame(thetest3$x[,1:3])
```

Final Prediction

The final prediction of the model used in this project is coded below.

```
######################
## Final Prediction ##
```

```
######################
fit2 <- train(classe~.,method="rf", data=thetrain)
```

```
## note: only 2 unique complexity parameters in default grid. Truncating the grid to 2 .
```

```
pred22 <- predict(fit2, newdata=thetest3) ## Final predictions
```