Weight Lifting Exercise Data Analysis

Introduction

Today we can collect a large amount of personal activity data inexpensively, with a view analyse the same and to improve one's health. In this regard it is noted that most such analysis focus on how much activity is done rather than how well the activity is done.

The analysis in this note seeks to analyse the weight lifting exercise data of 6 individuals so that we may develop a model to predict whether the exercises are being done correctly. IN this data set the dependant variable is “classe” with 5 values A,B,C,D,E - where A represents the correct way of dong the said activity. The dependant variables are data from data from accelerometers on the belt, forearm, arm, and dumbell.

Reading Data

1.Assuming that we have set directory with the respective training and testing data sets, we read the respective files as below

```
train = read.csv("pml-training.csv")  # Reading training
file in working directory
test = read.csv("pml-testing.csv")  # Reading testing
file in working directory
```

Exploring Data

From the exploration below we ascertain the following: 1. There are 19622 obs and 160 columns in the training data set and 20 obs and 160 columns in the testing data set. 2. In the training data set we have 67 columns that have NA values in 19216 (out of 19622) observations 3. In the testing data set we have 100 columns that have NA values in all 20 observations 4. The first six columns are not sensor data 5. The predictor variable in training set is “classe” 6 An extra variable in testing set is “problem_id”

```
str(train)
str(test)
summary(train)
summary(test)
dim(train)
dim(test)
f = apply(train, 2, function(x) length(which(is.na(x))))
a = apply(test, 2, function(x) length(which(is.na(x))))
table(f)
table(a)
```

Pre-processing data

1. In order that the model developed on the training set be applicable on testing set in terms of predictors available, we drop columns in both testing and training set that have 100% NAs in testing set.
2. In addition we drop the first six columns that are not sensor data from both training and testing set
3. We then check for NAs in the training set (with reduced columns) in case there are predictors with no data.

```
f = as.vector(apply(test, 2, function(x)
length(which(is.na(x))) == 20))  # Tag Column with NA
g = as.vector(names(test))  # Get full list of columns
h = data.frame(cbind(g, f))  # Get data frame of columns
and NA tags 9 TRUE)
i = as.vector(subset(h, h[, 2] == T)[, 1])  # Columns to
be dropped
j = as.vector(subset(h, h[, 2] == F)[, 1])  # Columns to
be retained
test = test[, c(j)]  # Create testing set with required
columns
drops <- head(names(test))  # further columns dropped
that are not sensor related
test = test[, !(names(test) %in% drops)]  # final testing
file with required columns

train = train[, !(names(train) %in% drops)]  # dropping
the 6 columns that are not sensor related
train = train[, !(names(train) %in% i)]  # dropping
columns that were not having any values in the testing
set

a = as.vector(apply(train, 2, function(x)
length(which(is.na(x)))))  # Check remaining columns in
training set have values in all columns
```

Splitting training set into train and test set

1. Splitting the training set into train and test so that we can derive the out of bag error by validating the model on the t.test data.

```
library(caTools)
set.seed(12345)
split = sample.split(train$classe, SplitRatio = 0.7)
t.train = subset(train, split == TRUE)
t.test = subset(train, split == FALSE)
```

Model Selection

1. We decide to use the random forest method with cross validation since (a) The independant variable is a factor variable (b) With a large number of predictors it is preferable to adopt non-linear methods (3) Among non-linear methods it is expected that RF with CV will give the desired objective of high predictive power.

2. OOB rate and accuracy results are very good. Refer confusion matrix.

```
library(caret)
```

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

```
library(randomForest)
```

```
## randomForest 4.6-7
## Type rfNews() to see new features/changes/bug fixes.
```

```
library(useful)
set.seed(12345)
mod.rf = randomForest(classe ~ ., data = t.train,
trControl = trainControl(method = "cv",
    number = 10))
mod.rf
```

```
##
## Call:
##  randomForest(formula = classe ~ ., data = t.train,
trControl = trainControl(method = "cv",      number =
10))
##               Type of random forest: classification
##                     Number of trees: 500
## No. of variables tried at each split: 7
##
##         OOB estimate of  error rate: 0.35%
## Confusion matrix:
##      A    B    C    D    E class.error
## A 3906    0    0    0    0    0.000000
## B    7 2649    2    0    0    0.003386
## C    0   11 2381    3    0    0.005846
## D    0    0   18 2232    1    0.008441
## E    0    0    0    6 2519    0.002376
```

```
table(t.train$classe)
```

```
##
##    A    B    C    D    E
## 3906 2658 2395 2251 2525
```

Validating Model on t.test data

1. The prediction accuracy on t.test data is 99.7%.

```
# Make predictions
pred = predict(mod.rf, newdata = t.test)
table(t.test$classe, pred)
```

```
##    pred
##       A    B    C    D    E
##   A 1674    0    0    0    0
##   B    2 1137    0    0    0
##   C    0    6 1021    0    0
##   D    0    0    9  956    0
##   E    0    0    0    2 1080
```

```
(1674 + 1138 + 1021 + 957 + 1080)/(1674 + 1138 + 1021 +
957 + 1080 + 2 + 8 +
    6 + 1)
```

```
## [1] 0.9971
```

Making Predictions test data

1. We now use model to predict “classe” variable for the 20 observations in the test data

```
# Make predictions
predtest = predict(mod.rf, newdata = test)
write.csv(predtest, "predtest.csv")
```