Predicting Exercise Method

Friday, March 20, 2015

Overview

The goal of this project is to predict the manner in which the people have done the exercise after training a model using the training data.

Read in the data

We first download and read in the data.

```
library(caret)

train_url = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"
#download.file(train_url,destfile=".//pml-training.csv")

test_url  = "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"
#download.file(test_url,destfile=".//pml-testing.csv")

train = read.csv(".//pml-training.csv")
test  = read.csv(".//pml-testing.csv")
```

Data Cleansing

To start with there are a possible 159 predictor variables in the dataset. On exploring the data it was found that there are a lot of variables with NA values and also lot of variables with mostly spaces. The folowing code identifies such variables and removes them from the dataset thus reducing the dimensions. After removing the sparse columns we get 52 predictors.

```
#Check nr. of predictors
colnames = names(train)

#Detect columns with NA's
x=c()
n = length(colnames)
for (i in 1:n)
{
  t = summary(train[,i])
  m = t["NA's"]
  if (!is.na(m)) {print(paste(i,"-", colnames[i], " NAs =", m)); x=c(x,i)}
}
```

```
## [1] "18 - max_roll_belt  NAs = 19216"
## [1] "19 - max_picth_belt  NAs = 19216"
## [1] "21 - min_roll_belt  NAs = 19216"
## [1] "22 - min_pitch_belt  NAs = 19216"
## [1] "24 - amplitude_roll_belt  NAs = 19216"
## [1] "25 - amplitude_pitch_belt  NAs = 19216"
## [1] "27 - var_total_accel_belt  NAs = 19216"
## [1] "28 - avg_roll_belt  NAs = 19216"
## [1] "29 - stddev_roll_belt  NAs = 19216"
## [1] "30 - var_roll_belt  NAs = 19216"
## [1] "31 - avg_pitch_belt  NAs = 19216"
## [1] "32 - stddev_pitch_belt  NAs = 19216"
## [1] "33 - var_pitch_belt  NAs = 19216"
## [1] "34 - avg_yaw_belt  NAs = 19216"
## [1] "35 - stddev_yaw_belt  NAs = 19216"
## [1] "36 - var_yaw_belt  NAs = 19216"
## [1] "50 - var_accel_arm  NAs = 19216"
## [1] "51 - avg_roll_arm  NAs = 19216"
## [1] "52 - stddev_roll_arm  NAs = 19216"
## [1] "53 - var_roll_arm  NAs = 19216"
## [1] "54 - avg_pitch_arm  NAs = 19216"
## [1] "55 - stddev_pitch_arm  NAs = 19216"
## [1] "56 - var_pitch_arm  NAs = 19216"
## [1] "57 - avg_yaw_arm  NAs = 19216"
## [1] "58 - stddev_yaw_arm  NAs = 19216"
## [1] "59 - var_yaw_arm  NAs = 19216"
## [1] "75 - max_roll_arm  NAs = 19216"
## [1] "76 - max_picth_arm  NAs = 19216"
## [1] "77 - max_yaw_arm  NAs = 19216"
## [1] "78 - min_roll_arm  NAs = 19216"
## [1] "79 - min_pitch_arm  NAs = 19216"
## [1] "80 - min_yaw_arm  NAs = 19216"
## [1] "81 - amplitude_roll_arm  NAs = 19216"
## [1] "82 - amplitude_pitch_arm  NAs = 19216"
## [1] "83 - amplitude_yaw_arm  NAs = 19216"
## [1] "93 - max_roll_dumbbell  NAs = 19216"
## [1] "94 - max_picth_dumbbell  NAs = 19216"
## [1] "96 - min_roll_dumbbell  NAs = 19216"
## [1] "97 - min_pitch_dumbbell  NAs = 19216"
## [1] "99 - amplitude_roll_dumbbell  NAs = 19216"
## [1] "100 - amplitude_pitch_dumbbell  NAs = 19216"
## [1] "103 - var_accel_dumbbell  NAs = 19216"
## [1] "104 - avg_roll_dumbbell  NAs = 19216"
## [1] "105 - stddev_roll_dumbbell  NAs = 19216"
## [1] "106 - var_roll_dumbbell  NAs = 19216"
## [1] "107 - avg_pitch_dumbbell  NAs = 19216"
## [1] "108 - stddev_pitch_dumbbell  NAs = 19216"
## [1] "109 - var_pitch_dumbbell  NAs = 19216"
## [1] "110 - avg_yaw_dumbbell  NAs = 19216"
## [1] "111 - stddev_yaw_dumbbell  NAs = 19216"
## [1] "112 - var_yaw_dumbbell  NAs = 19216"
## [1] "131 - max_roll_forearm  NAs = 19216"
## [1] "132 - max_picth_forearm  NAs = 19216"
## [1] "134 - min_roll_forearm  NAs = 19216"
## [1] "135 - min_pitch_forearm  NAs = 19216"
## [1] "137 - amplitude_roll_forearm  NAs = 19216"
## [1] "138 - amplitude_pitch_forearm  NAs = 19216"
## [1] "141 - var_accel_forearm  NAs = 19216"
## [1] "142 - avg_roll_forearm  NAs = 19216"
## [1] "143 - stddev_roll_forearm  NAs = 19216"
## [1] "144 - var_roll_forearm  NAs = 19216"
## [1] "145 - avg_pitch_forearm  NAs = 19216"
## [1] "146 - stddev_pitch_forearm  NAs = 19216"
## [1] "147 - var_pitch_forearm  NAs = 19216"
## [1] "148 - avg_yaw_forearm  NAs = 19216"
## [1] "149 - stddev_yaw_forearm  NAs = 19216"
## [1] "150 - var_yaw_forearm  NAs = 19216"
```

```
#Since all of the columns containing NA's are sparce(19000+ records in each con
tain NA), remove them from the dataset
train=train[,-c(1:7,x)]
test=test[,-c(1:7,x)]

#Detect columns with spaces
colnames=names(train)
x=c()
n = length(colnames)
for (i in 1:n)
{
  t = as.character(train[,i])
  m = which(t == "")
  if (length(m)!=0) { x=c(x,i); print(paste(i,"-", colnames[i], " Spaces =", le
ngth(m)))}
}
```

```
## [1] "5 - kurtosis_roll_belt  Spaces = 19216"
## [1] "6 - kurtosis_picth_belt  Spaces = 19216"
## [1] "7 - kurtosis_yaw_belt  Spaces = 19216"
## [1] "8 - skewness_roll_belt  Spaces = 19216"
## [1] "9 - skewness_roll_belt.1  Spaces = 19216"
## [1] "10 - skewness_yaw_belt  Spaces = 19216"
## [1] "11 - max_yaw_belt  Spaces = 19216"
## [1] "12 - min_yaw_belt  Spaces = 19216"
## [1] "13 - amplitude_yaw_belt  Spaces = 19216"
## [1] "36 - kurtosis_roll_arm  Spaces = 19216"
## [1] "37 - kurtosis_picth_arm  Spaces = 19216"
## [1] "38 - kurtosis_yaw_arm  Spaces = 19216"
## [1] "39 - skewness_roll_arm  Spaces = 19216"
## [1] "40 - skewness_pitch_arm  Spaces = 19216"
## [1] "41 - skewness_yaw_arm  Spaces = 19216"
## [1] "45 - kurtosis_roll_dumbbell  Spaces = 19216"
## [1] "46 - kurtosis_picth_dumbbell  Spaces = 19216"
## [1] "47 - kurtosis_yaw_dumbbell  Spaces = 19216"
## [1] "48 - skewness_roll_dumbbell  Spaces = 19216"
## [1] "49 - skewness_pitch_dumbbell  Spaces = 19216"
## [1] "50 - skewness_yaw_dumbbell  Spaces = 19216"
## [1] "51 - max_yaw_dumbbell  Spaces = 19216"
## [1] "52 - min_yaw_dumbbell  Spaces = 19216"
## [1] "53 - amplitude_yaw_dumbbell  Spaces = 19216"
## [1] "67 - kurtosis_roll_forearm  Spaces = 19216"
## [1] "68 - kurtosis_picth_forearm  Spaces = 19216"
## [1] "69 - kurtosis_yaw_forearm  Spaces = 19216"
## [1] "70 - skewness_roll_forearm  Spaces = 19216"
## [1] "71 - skewness_pitch_forearm  Spaces = 19216"
## [1] "72 - skewness_yaw_forearm  Spaces = 19216"
## [1] "73 - max_yaw_forearm  Spaces = 19216"
## [1] "74 - min_yaw_forearm  Spaces = 19216"
## [1] "75 - amplitude_yaw_forearm  Spaces = 19216"
```

```
#Since all of the columns containing spaces are sparce(19000+ records in each c
ontain spaces), remove them from the dataset
train=train[,-c(x)]
test=test[,-c(x)]
```

Model Creation

The following models were tried and their findings are explained in the sections below:

Cartesian Tree

Cartesian Tree on the entire dataset without any pre-processing gave poor predictive accuracy. Hence was rejected.

```
modFit = train(classe~., method="rpart", data=train)
```

```
## Loading required package: rpart
```

```
confusionMatrix(train$classe,predict(modFit,newdata=train[,-53]))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 5080   81  405    0   14
##          B 1581 1286  930    0    0
##          C 1587  108 1727    0    0
##          D 1449  568 1199    0    0
##          E  524  486  966    0 1631
## 
## Overall Statistics
## 
##                Accuracy : 0.4956
##                  95% CI : (0.4885, 0.5026)
##     No Information Rate : 0.5209
##     P-Value [Acc > NIR] : 1
## 
##                   Kappa : 0.3407
##  Mcnemar's Test P-Value : NA
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.4970  0.50850  0.33040       NA  0.99149
## Specificity            0.9468  0.85310  0.88225   0.8361  0.89008
## Pos Pred Value         0.9104  0.33869  0.50468       NA  0.45218
## Neg Pred Value         0.6339  0.92145  0.78395       NA  0.99913
## Prevalence             0.5209  0.12889  0.26638   0.0000  0.08383
## Detection Rate         0.2589  0.06554  0.08801   0.0000  0.08312
## Detection Prevalence   0.2844  0.19351  0.17440   0.1639  0.18382
## Balanced Accuracy      0.7219  0.68080  0.60633       NA  0.94079
```

Pre-processing using PCA

Principal Component Anaysis identified Principal components, but a tree model created using the PCA predictors had very poor predictive accuracy. PCA was tried at 100%, 99% and 95% threshold levels. But all gave poor predictive accuracy and hence this effort was rejected.

```
prComp = prcomp(train[,-53])

preProc = preProcess(train[,-53],method="pca", pcaComp=2)
preProc
```

```
##
## Call:
## preProcess.default(x = train[, -53], method = "pca", pcaComp = 2)
##
## Created from 19622 samples and 52 variables
## Pre-processing: principal component signal extraction, scaled, centered
##
## PCA used 2 components as specified.
```

```
trainPC = predict(preProc,train[,-53])
trainPC = cbind(trainPC,classe=train[,53])
modFit = train(classe~., method="rpart", data=trainPC)
confusionMatrix(trainPC$classe,predict(modFit,newdata=trainPC[,-6]))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 3405  464  291  646  774
##          B 1295  822  289  475  916
##          C 1349  453  471  418  731
##          D  514  577   76 1230  819
##          E  706  382  190  695 1634
## 
## Overall Statistics
## 
##                Accuracy : 0.3854
##                  95% CI : (0.3786, 0.3922)
##     No Information Rate : 0.3705
##     P-Value [Acc > NIR] : 8.053e-06
## 
##                   Kappa : 0.2138
##  Mcnemar's Test P-Value : < 2.2e-16
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.4684  0.30467  0.35763  0.35508  0.33525
## Specificity            0.8239  0.82421  0.83879  0.87709  0.86622
## Pos Pred Value         0.6102  0.21649  0.13764  0.38246  0.45301
## Neg Pred Value         0.7248  0.88145  0.94778  0.86383  0.79769
## Prevalence             0.3705  0.13750  0.06712  0.17654  0.24839
## Detection Rate         0.1735  0.04189  0.02400  0.06268  0.08327
## Detection Prevalence   0.2844  0.19351  0.17440  0.16390  0.18382
## Balanced Accuracy      0.6462  0.56444  0.59821  0.61608  0.60073
```

Random Forest

Random Forest was finally selected since it gave a predictive accurancy of 97%. Random Forest could not run successfully on my laptop for the entire data. It would error out after a while citing memory allocation issues. It ran successfully for 60% of the training data. The remainder of the training data was used to test the model. Models were created using ntree=5 and 10 since anything above 10 would take more than 1 hr and the laptop would get scaringly hot. With ntree=10, the model gave an accuracy rate around 97% and was accepted since I could not try for better values due to laptop limitations.

```
#Splitting the training set into training and testing
inTrain = createDataPartition(y=train$classe, p=0.6, list=FALSE)
training = train[inTrain,]
testing = train[-inTrain,]

#Creating a random forest. Not running this again while report generation since it takes a long time. Just executing the final model in the section below
#modFit = train(classe~., method="rf", data=training, ntree=10, prox=T)
#Checking the confusion matrix
#confusionMatrix(training$classe,predict(modFit,newdata=training[,-53]))
```

Crossvalidation

Default Bootstrap reSampling with 25 repeats provided a good accuracy rate of 97.7%.

5 fold crossvalidation with 5 repeats provided and accurance of 98.56%.

10 fold cross-validation with 10 repeats provided and accurance of 98.64%. Submitted the predictions using this model because it provided the highest accuracy.

Although the 10 fold cross-validation model did not provide a significant improvement in accuracy over the 5 fold model compared to the time it took to get generated, it gave a better accuracy compared to the bootstrap resampling cross-validation.

```
#During report generation am running a 5 fold crossvalidation model since 10-fold model takes toooo long. The actual prediction was done with 10 fold cross-validation model.
fitControl <- trainControl(## 5-fold CV
  method = "repeatedcv",
  number = 5,
  ## repeated 5 times
  repeats = 5)
modFit = train(classe~., method="rf", data=training, trControl = fitControl, ntree=10, prox=T)
```

```
confusionMatrix(testing$classe,predict(modFit,newdata=testing[,-53]))
```

```
## Confusion Matrix and Statistics
## 
##           Reference
## Prediction    A    B    C    D    E
##          A 2221    6    0    3    2
##          B   21 1482   14    1    0
##          C    0   18 1344    6    0
##          D    0    1   19 1264    2
##          E    0    5    8    4 1425
## 
## Overall Statistics
## 
##                Accuracy : 0.986           
##                  95% CI : (0.9831, 0.9885)
##     No Information Rate : 0.2858          
##     P-Value [Acc > NIR] : < 2.2e-16       
## 
##                   Kappa : 0.9823          
##  Mcnemar's Test P-Value : NA              
## 
## Statistics by Class:
## 
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9906   0.9802   0.9704   0.9890   0.9972
## Specificity            0.9980   0.9943   0.9963   0.9967   0.9974
## Pos Pred Value         0.9951   0.9763   0.9825   0.9829   0.9882
## Neg Pred Value         0.9963   0.9953   0.9937   0.9979   0.9994
## Prevalence             0.2858   0.1927   0.1765   0.1629   0.1821
## Detection Rate         0.2831   0.1889   0.1713   0.1611   0.1816
## Detection Prevalence   0.2845   0.1935   0.1744   0.1639   0.1838
## Balanced Accuracy      0.9943   0.9872   0.9833   0.9928   0.9973
```

Expected Out of Sample Error

I expect the out of sample error to be less than 3% i.e on a sample of 20, I expect less than 1 to be wrong.FYI:I got 1 wrong when I predicted in first run. But in the 2nd run without any change of parameters it predicted correctly for the one that it had got wrong earlier.

Predicted Output

The predicted output is as follows:

```
predict(modFit,newdata=test[,-53])
```

```
##  [1] B A B A A E D B A A C C B A E E A B B B
## Levels: A B C D E
```