PML Sainath Kumar

Sai

July 29, 2016

```
library(caret)
```

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

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

```
library(ISLR)
library(ggplot2)
library(Hmisc)
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
## Loading required package: Formula
```

```
##
## Attaching package: 'Hmisc'
```

```
## The following objects are masked from 'package:base':
##
##     format.pval, round.POSIXt, trunc.POSIXt, units
```

```
library(splines)
library(rpart)
library(ElemStatLearn)
```

Load the training and testing data sets

```
setwd("C:/Users/IBM_ADMIN/Documents/Coursera/Practical Machine LEarning")
train <-read.csv("pml-training.csv",header=TRUE)
test <-read.csv("pml-testing.csv",header=TRUE)
dim(train)
```

```
## [1] 19622   160
```

Analyze the data set

Based on initial look at the data the data has blanks , NA ,#DIV/0 First step would be to cleanse the data Also at first glance the first 7 columns do not seem like the ones that would have effect on the prediction outcome .

```
train <-read.csv("pml-training.csv",header=TRUE,na.strings=c("NA","#DIV/0!",""))
test <-read.csv("pml-testing.csv",header=TRUE,na.strings=c("NA","#DIV/0!",""))
train <- train[,8:160]
```

Identifying and removing near zero variance variables

```
train_var <- nearZeroVar(train, saveMetrics=TRUE)
train_var1 <- train[,train_var$nzv==FALSE]

train_1 <- train_var1
for(i in 1:length(train_var1)) {
  if( sum( is.na( train_var1[, i] ) ) /nrow(train_var1) >= .6) {
    for(j in 1:length(train_1)) {
      if( length( grep(names(train_var1[i]), names(train_1)[j]) ) == 1)  {
        train_1 <- train_1[ , -j]
      }
    }
  }
}
train_var1 <- train_1
rm(train_1)
dim(train_var1)
```

```
## [1] 19622    53
```

Applying the same transformations to the testing set as well

#Since here we have removed the near zero variables and NA variables testing data should also have only these variables

```
train_variables <- colnames(train_var1)
train_variables_1 <- colnames(train_var1[, -53])  # remove the classe column
test <- test[train_variables_1]
dim(test)
```

```
## [1] 20 52
```

Partitioning the training data

```
intrain <- createDataPartition(train_var1$classe, p=0.7, list=FALSE)
train_fin <- train_var1[intrain, ]
valid_fin <- train_var1[-intrain, ]
dim(train_fin)
```

```
## [1] 13737    53
```

```
dim(valid_fin)
```

```
## [1] 5885   53
```

Using basic classification tree algorithm to predict the classe

```
modfit <- train(train_fin$classe ~ ., data = train_fin, method="rpart")
print(modfit)
```

```
## CART
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.03255010  0.5180528  0.37114903
##   0.05852236  0.4133046  0.20161251
##   0.11545112  0.3352409  0.07439787
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.0325501.
```

```
predict_1 <- predict(modfit, newdata=valid_fin)
print(confusionMatrix(predict_1,valid_fin$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1520  438  460  409  157
##          B   20  400   34  172  149
##          C  132  301  532  383  292
##          D    0    0    0    0    0
##          E    2    0    0    0  484
##
## Overall Statistics
##
##                Accuracy : 0.4989
##                  95% CI : (0.486, 0.5118)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3463
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9080  0.35119   0.5185   0.0000  0.44732
## Specificity            0.6523  0.92099   0.7720   1.0000  0.99958
## Pos Pred Value         0.5094  0.51613   0.3244      NaN  0.99588
## Neg Pred Value         0.9469  0.85538   0.8836   0.8362  0.88924
## Prevalence             0.2845  0.19354   0.1743   0.1638  0.18386
## Detection Rate         0.2583  0.06797   0.0904   0.0000  0.08224
## Detection Prevalence   0.5071  0.13169   0.2787   0.0000  0.08258
## Balanced Accuracy      0.7802  0.63609   0.6452   0.5000  0.72345
```

the accuracy of the prediction deteriorated further less that 50% therefore this would not be a good model to predict Do a check if preprocessing the data will improve the prediction results

```
modfit_1 <- train(train_fin$classe ~ ., data = train_fin, method="rpart", preProcess
=c("center", "scale"))
print(modfit_1)
```

```
## CART
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (52), scaled (52)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.03255010  0.5101534  0.36102021
##   0.05852236  0.4111651  0.20130656
##   0.11545112  0.3364399  0.07962087
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was cp = 0.0325501.
```

```
predict_1 <- predict(modfit_1, newdata=valid_fin)
print(confusionMatrix(predict_1,valid_fin$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1520  438  460  409  157
##          B   20  400   34  172  149
##          C  132  301  532  383  292
##          D    0    0    0    0    0
##          E    2    0    0    0  484
##
## Overall Statistics
##
##                Accuracy : 0.4989
##                  95% CI : (0.486, 0.5118)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3463
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9080  0.35119   0.5185   0.0000  0.44732
## Specificity            0.6523  0.92099   0.7720   1.0000  0.99958
## Pos Pred Value         0.5094  0.51613   0.3244      NaN  0.99588
## Neg Pred Value         0.9469  0.85538   0.8836   0.8362  0.88924
## Prevalence             0.2845  0.19354   0.1743   0.1638  0.18386
## Detection Rate         0.2583  0.06797   0.0904   0.0000  0.08224
## Detection Prevalence   0.5071  0.13169   0.2787   0.0000  0.08258
## Balanced Accuracy      0.7802  0.63609   0.6452   0.5000  0.72345
```

Preprocessing does not improve the accruacy further therfore proceeding to use other algrithms to build the model

Random forests to be used to build further models

Lines Commented for Kniting as RF take long time to process. Please see Original R code for details

```
modfit_2 <- train(train_fin$classe ~ ., data = train_fin, method="rf", preProcess=c(
"center", "scale"))
```

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

```
## randomForest 4.6-12
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:Hmisc':
##
##     combine
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
print(modfit_2)
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## Pre-processing: centered (52), scaled (52)
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 13737, 13737, 13737, 13737, 13737, 13737, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy   Kappa
##    2    0.9885668  0.9855352
##   27    0.9882691  0.9851590
##   52    0.9817997  0.9769746
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was mtry = 2.
```

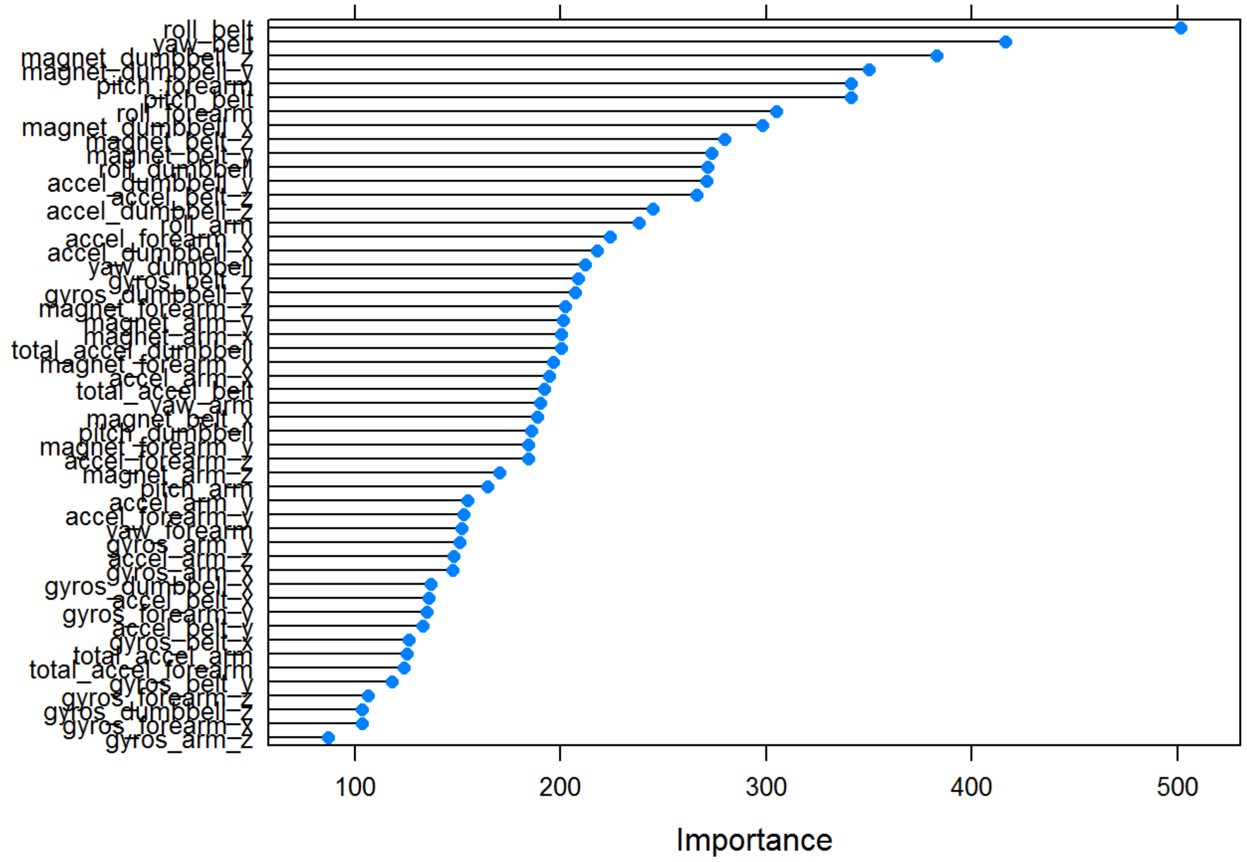
```
 predict_2 <- predict(modfit_2, newdata=valid_fin)
 print(confusionMatrix(predict_2,valid_fin$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    9    0    0    0
##          B    2 1129    8    0    0
##          C    0    1 1018   18    0
##          D    0    0    0  946    4
##          E    0    0    0    0 1078
##
## Overall Statistics
##
##                Accuracy : 0.9929
##                  95% CI : (0.9904, 0.9949)
##     No Information Rate : 0.2845
##     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            0.9988   0.9912   0.9922   0.9813   0.9963
## Specificity            0.9979   0.9979   0.9961   0.9992   1.0000
## Pos Pred Value         0.9946   0.9912   0.9817   0.9958   1.0000
## Neg Pred Value         0.9995   0.9979   0.9983   0.9964   0.9992
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1918   0.1730   0.1607   0.1832
## Detection Prevalence   0.2856   0.1935   0.1762   0.1614   0.1832
## Balanced Accuracy      0.9983   0.9946   0.9941   0.9903   0.9982
```

```
predict_2 <- predict(modfit_2, newdata=valid_fin)
print(confusionMatrix(predict_2,valid_fin$classe))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    9    0    0    0
##          B    2 1129    8    0    0
##          C    0    1 1018   18    0
##          D    0    0    0  946    4
##          E    0    0    0    0 1078
##
## Overall Statistics
##
##                Accuracy : 0.9929
##                  95% CI : (0.9904, 0.9949)
##     No Information Rate : 0.2845
##     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            0.9988   0.9912   0.9922   0.9813   0.9963
## Specificity            0.9979   0.9979   0.9961   0.9992   1.0000
## Pos Pred Value         0.9946   0.9912   0.9817   0.9958   1.0000
## Neg Pred Value         0.9995   0.9979   0.9983   0.9964   0.9992
## Prevalence             0.2845   0.1935   0.1743   0.1638   0.1839
## Detection Rate         0.2841   0.1918   0.1730   0.1607   0.1832
## Detection Prevalence   0.2856   0.1935   0.1762   0.1614   0.1832
## Balanced Accuracy      0.9983   0.9946   0.9941   0.9903   0.9982
```

```
print(plot(varImp(modfit_2, scale = FALSE)))
```



Applying the model to the final test data for 20 cases Run against 20 testing set provided by Professor Leek.

```
print(predict(modfit_2, newdata=test))
```

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

```
Results <- data.frame(predicted=predict(modfit_2, newdata=test))
print(Results)
```

```
##    predicted
## 1          B
## 2          A
## 3          B
## 4          A
## 5          A
## 6          E
## 7          D
## 8          B
## 9          A
## 10         A
## 11         B
## 12         C
## 13         B
## 14         A
## 15         E
## 16         E
## 17         A
## 18         B
## 19         B
## 20         B
```

Conclusion:

Based on the RF model the test cases have been predicted as above.The accuracy for the RF model is relatively higher and would certainly provide more accurate results Additional Analysis and further explanation: I would like to have done additional exploratory data analysis to reduce the number of predictors down further. PCA- Principal component analysis as well as using concepts from Regularized regression and combining predictor chapter.