MachineLearningProject

NEO

12/13/2019

Executive Summary

This is the project for week four of the Machine Learning class that is part of Coursera's Data Science suite. This project will follow steps for machine learning:

1.Define Problem. 2.Load Data, Sumamrize and Prepare Data. 3.Visualizing the dataset. 4.Evaluate Algorithms. 5.Making some predictions. 6.Improve Results. 7.Present Results.

Background

This report describes model building, cross validation, what the expected out of sample error is, and why I made the choices I did. The prediction model is then used to predict 20 different test cases.

The Human Activity Recognition (HAR) data used in this project and more information is available from: http://web.archive.org/web/20161224072740/http:/groupware.les.inf.puc-rio.br/har. Participantds were asked to perform barbell lifts correctly and incorrectly in 5 different ways. Data from accelerometers on the belt, forearm, arm, and dumbell of 6 participants and predict the manner in which persons did the exercise.

The libraries included in this project are listed here in the librarySetUp section of R code.

```
library(dplyr)
library(data.table)
library(caret)
library(rpart);
library(ggplot2)
library(randomForest);
library(R.utils)
```

The code for setting up the local analysis directory, retieving data file and reading in the dataset described and documented below. First, a data directory in the users working directory is created if it does not exist.

```
if(!file.exists('./data')) dir.create('./data')
```

Data for the analysis is retrieved using R script. The code for this is included in the report below for purposes of reproduction of this analyisis.

```
#
if(!file.exists('./data/pml-testing.csv')) {
 fileURL<-'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv'
 download.file(fileURL, destFile = './data/pml-testing.csv')
}
if(!file.exists('./data/pml-training.csv')) {
 fileURL<-'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv'
 download.file(fileURL, destFile = './data/pml-training.csv')
}
```

#Step 1. Load the data

```
      fileName = './data/pml-training.csv'
      pmlTrnData <- read.csv(fileName, na.strings = c("NA", ""), strip.white = TRUE,
                             stringsAsFactors=FALSE)
# Uncomment the STR function to get a look at the data.
##       str(pmlTrnData)
#
      fileName = './data/pml-testing.csv'
      pmlTstData <- read.csv(fileName, na.strings = c("NA", ""), strip.white = TRUE,
                             stringsAsFactors=FALSE)
```

#Step 2. Preprocess the Data and Create Training and Validation Data Sets Clean the data and remove data columns that have no data. This will remove all attributes containing NA: Next, on both the training and the testing data, removal of first 7 of the 60 data frame attributes since they are not pertainent attributes or predictors.

```
      pmlTraining <- pmlTrnData[ , colSums(is.na(pmlTrnData))==0]
      pmlTesting  <- pmlTstData[ , colSums(is.na(pmlTstData))==0]

      pmlTrn <- pmlTraining[ , -c(1:7)]
      pmlTst <- pmlTesting[ , -c(1:7)]
# Show the dimension of the data set
      dim(pmlTrn)
```

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

```
      dim(pmlTst)
```

```
## [1] 20 53
```

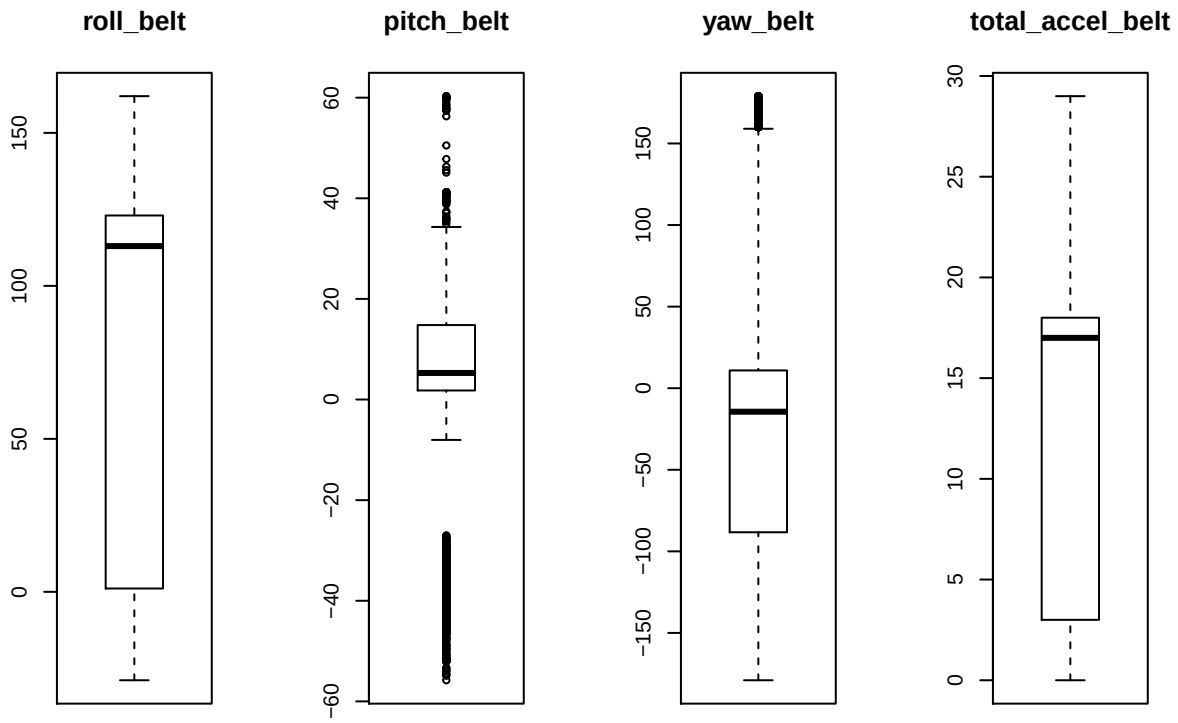
```
# Create Normalize Function and apply to training and testing set
#       normalize <- function(x) {
#               num <- x - min(x)
#               denom <- max(x) - min(x)
#               return (num/denom)
#       }
#
# Normalize the data (minus the classe/problem_id)
#       tstNorm<-as.data.frame(lapply(pmlTst[1:52], normalize))
#       trnNorm<-as.data.frame(lapply(pmlTrn[1:52], normalize))
# Add problem_id/classe back to normalized data
#       tstNorm$problem_id <- pmlTst$problem_id
#           trnNorm$classe <- pmlTrn$classe
# Get understanding of the classe variable in the training data set
          levels (as.factor(pmlTrn$classe))
```

```
## [1] "A" "B" "C" "D" "E"
```

```
# Split trnNorm into training and Validation at a 70%/30% rate
         inTrain <- createDataPartition(pmlTrn$classe, p = 0.7, list = FALSE)
        training <- pmlTrn[inTrain, ]
      validation <- pmlTrn[-inTrain, ]
```

Step 3. Visualize the data

```
# For this paper- ontly the first 4 attributes are plotted
        x <- training[,1:4]
        y <-training[,53]
        par(mfrow=c(1,4))
        for(i in 1:4) { boxplot(x[,i], main=names(training)[i])
        }
```



The dim function on the training data shows 52 attributes along with "classe" (53 col total) with 19622 rows, and the testing data has 52 similar attributes along with "problem_id" (53 col total) and 20 rows. The unique function shows that the classe variable has values: "A" "B" "C" "D" "E".

Due to the size of the data set and attributes, the machine learning methods of Trees and Random Forests will by applied. The best approach will be used to predict using the testing data set. Repeated cross validation is used in the trainControl specification, again in an exploratory fashion, and has yielded good results.

#Step 4. Build Model Trees

```
# Set random seed for reproducibility
       set.seed(1234)
# Assign trainControl attributes to limit the number used in the classification. I will use 5-fold repe
       ctrl<- trainControl(method="cv",number=10)
       modFitTree <- train(classe ~ ., data=training, method="rpart", trControl=ctrl)
       modFitTree
```

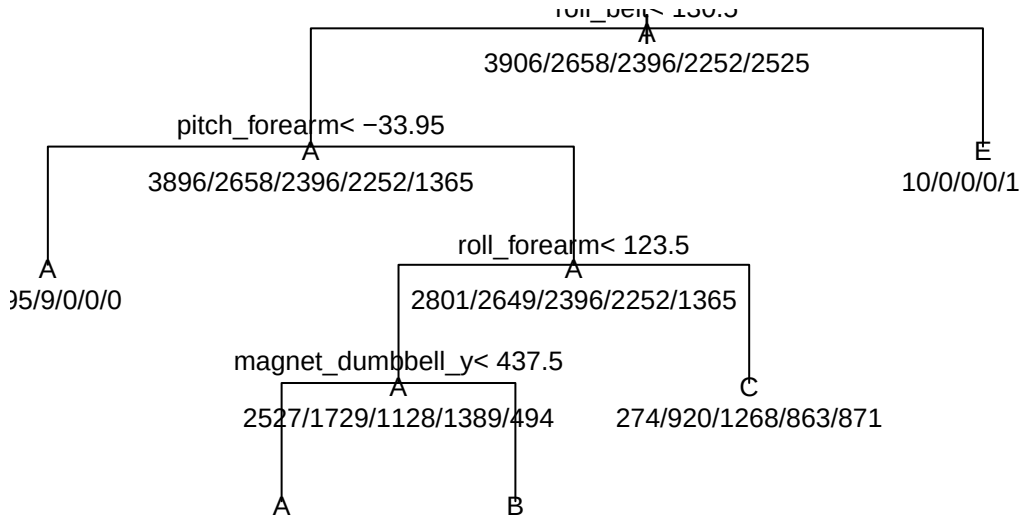
```
## CART
##
```

```
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12363, 12364, 12362, 12364, 12363, 12363, ...
## Resampling results across tuning parameters:
##
##   cp          Accuracy   Kappa
##   0.04323060  0.4910805  0.33530375
##   0.05055437  0.4487130  0.26614715
##   0.11697691  0.3144782  0.04591119
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was cp = 0.0432306.
```

```
print(modFitTree$finalModel)
```

```
## n= 13737
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 13737 9831 A (0.28 0.19 0.17 0.16 0.18)
##    2) roll_belt< 130.5 12567 8671 A (0.31 0.21 0.19 0.18 0.11)
##      4) pitch_forearm< -33.95 1104    9 A (0.99 0.0082 0 0 0) *
##      5) pitch_forearm>=-33.95 11463 8662 A (0.24 0.23 0.21 0.2 0.12)
##       10) roll_forearm< 123.5 7267 4740 A (0.35 0.24 0.16 0.19 0.068)
##         20) magnet_dumbbell_y< 437.5 5969 3506 A (0.41 0.18 0.18 0.17 0.059) *
##         21) magnet_dumbbell_y>=437.5 1298  626 B (0.049 0.52 0.029 0.3 0.11) *
##       11) roll_forearm>=123.5 4196 2928 C (0.065 0.22 0.3 0.21 0.21) *
##    3) roll_belt>=130.5 1170   10 E (0.0085 0 0 0 0.99) *
```

```
# Plot Resulting Tree
plot(modFitTree$finalModel, uniform=TRUE, main="Classification Tree")
text(modFitTree$finalModel, use.n=TRUE, all=TRUE, cex=.8)
```




```
# Now predict using modelFitTrees using the validation Data
       ans <- predict(modFitTree,newdata=validation)

           summary(ans)
```

```
##    A    B    C    D    E
## 3124  561 1725    0  475
```

```
#  Result from confusionMatrix
       classe<-as.factor(validation$classe)
       confTree <- confusionMatrix(classe,ans)
       confTree
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1516   22  132    0    4
##          B  507  287  345    0    0
##          C  495   17  514    0    0
##          D  438  164  362    0    0
##          E  168   71  372    0  471
##
## Overall Statistics
##
##                Accuracy : 0.4737
##                  95% CI : (0.4609, 0.4866)
```

```
##     No Information Rate : 0.5308
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.3117
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.4853  0.51159  0.29797       NA  0.99158
## Specificity            0.9428  0.83997  0.87692   0.8362  0.88706
## Pos Pred Value         0.9056  0.25198  0.50097       NA  0.43530
## Neg Pred Value         0.6181  0.94227  0.75077       NA  0.99917
## Prevalence             0.5308  0.09533  0.29312   0.0000  0.08071
## Detection Rate         0.2576  0.04877  0.08734   0.0000  0.08003
## Detection Prevalence   0.2845  0.19354  0.17434   0.1638  0.18386
## Balanced Accuracy      0.7140  0.67578  0.58745       NA  0.93932
```

Accuracy was used to select the optimal model using the largest value. The final value used for the model was cp = 0.03387244.

Next, predict using modelFitTrees and validation data with ans<-predict(modFitTree,newdata=validation)

The overall statistics as listed above shows ~.5 accuracy, or about the flip of a coin.

Step4b. KNN approach

```
      set.seed(1234)
      modFitKnn <- train(classe ~ ., data=training, method="knn", trControl=ctrl)
      modFitKnn
```

```
## k-Nearest Neighbors
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12363, 12364, 12362, 12364, 12363, 12363, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.8972837  0.8700647
##   7  0.8729693  0.8392569
##   9  0.8573901  0.8194962
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
# Use the model for prediction
      ansKnn<-predict(modFitKnn,newdata=validation)
      summary(ansKnn)
```

```
##    A    B    C    D    E
```

```
## 1725 1090 1070  986 1014
#
```

Step4c. Ranger Random Forest approach

Here, research led to the 'ranger' package for a fast implementation of Random Forest (Breiman 2001) for high dimensional data. The package is automatically installed when running the script.

```
# Set random seed for reproducibility
       set.seed(1234)
      modFitRRF <- train(classe ~ ., data=training,method="ranger", trControl=ctrl)
      modFitRRF
```

```
## Random Forest
##
## 13737 samples
##    52 predictor
##     5 classes: 'A', 'B', 'C', 'D', 'E'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 12363, 12364, 12362, 12364, 12363, 12363, ...
## Resampling results across tuning parameters:
##
##   mtry  splitrule   Accuracy   Kappa
##    2    gini        0.9917009  0.9895010
##    2    extratrees  0.9903908  0.9878440
##   27    gini        0.9915556  0.9893173
##   27    extratrees  0.9940303  0.9924485
##   52    gini        0.9848582  0.9808458
##   52    extratrees  0.9945401  0.9930934
##
## Tuning parameter 'min.node.size' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were mtry = 52, splitrule = extratrees
##  and min.node.size = 1.
```

```
# Use the model for prediction
       ans2<-predict(modFitRRF,newdata=validation)
       summary(ans2)
```

```
##    A    B    C    D    E
## 1678 1137 1030  960 1080
```

```
# Check accuracy using the confusion matrix.
       confRRF<-confusionMatrix(classe,ans2)
       confRRF
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1672    2    0    0    0
##          B    6 1131    2    0    0
##          C    0    4 1022    0    0
##          D    0    0    4  959    1
```

```
##          E    0    0    2    1 1079
##
## Overall Statistics
##
##                Accuracy : 0.9963
##                  95% CI : (0.9943, 0.9977)
##     No Information Rate : 0.2851
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9953
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9964   0.9947   0.9922   0.9990   0.9991
## Specificity            0.9995   0.9983   0.9992   0.9990   0.9994
## Pos Pred Value         0.9988   0.9930   0.9961   0.9948   0.9972
## Neg Pred Value         0.9986   0.9987   0.9984   0.9998   0.9998
## Prevalence             0.2851   0.1932   0.1750   0.1631   0.1835
## Detection Rate         0.2841   0.1922   0.1737   0.1630   0.1833
## Detection Prevalence   0.2845   0.1935   0.1743   0.1638   0.1839
## Balanced Accuracy      0.9980   0.9965   0.9957   0.9990   0.9992
```

Ranger Random Forest.

The accuracy for the Ranger implementation of Random Forest is very good at .997.

Step 5. Prediction with Ranger Random Forest

Finally, using the Ranger Random Forest to predict against the normalized testing data:

```
      ans3<-predict(modFitRRF,pmlTst)
      ans3
```

```
##  [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
```

```
      summary(ans3)
```

```
## A B C D E
## 7 8 1 1 3
```

Discussion of Findings

Three ML approaches to model building were tried, rpart (Trees) , KNN and Ranger Random forest. A judgement call was made to normalize the data, while this was not rigoursly tested against non-normalized data, it appears to be a sound call. The application of simple Trees yielded a prediction capability similar to a coin toss, and thus not a preferred implementation of ML. Knn greatly improved results with accuracy ~.92%. The Random Forest approach was then applied using the Ranger package, and the accuracy went to 99.7%. The Ranger package is a rapid implementation of RF, and was implemented for time saving. This model was ultimately used to predict using the testing data.

References

Breiman, L. (2001). Random forests. Mach Learn, 45:5-32. https://doi.org/10.1023/A:1010933404324.

Ugulino, W.; Cardador, D.; Vega, K.; Velloso, E.; Milidiu, R.; Fuks, H. Wearable Computing: Accelero