Practical_Machine_Learning_Project

Md Ahmed

August 9th, 2017

Project Report: Machine Learning Algorithms

We are given two sets of data collected from accelerometers placed on the belt, forearm, arm, and dumbell of 6 research study participants for this machine learning project. Training data stems from accelerometers with label identifying the quality of the activity the participant was doing. Testing data also comprised of accelerometer data without identifiable label(A-E).

The definitive instruction for this project is to use data to predict whether the exercise is being done properly or improperly based solely on accelerometer data measurements. The participants were instructed to perform the exercise either properly (Class A) or in a way which replicated 4 common weightlifting mistakes (Classes B, C, D, and E).

The question is, would we be able to predict appropriately each participants exercise manner by processing data gathered from classe(A-E) accelerometers? In that persuasion, we should apply some Machine Learning(ML) algorithms on 'trainData' and test them on given 'test dataset' for 'classe-level' based exercise manner prediction.

1. Project write up Sequence:

Here in drop down, I wrote the needed 'code' along with 'line-description' on each step of the process of ML-algorithms. I have used four machine learnig algorithms are Classification Tree, lda, gbm and random forest. I also used cross-validation with 'method' and 'k-folds' with number within all model. At the end of each ML-algorithm run, I presented the quantified 'accuracy rate'.

These findings would help us to analyse and predict the manner, in which participants did their exercise regime.

2. Data loading, visual overview and manipulation:

```
# Necessary library loaded
library(easypackages)
```

```
## Warning: package 'easypackages' was built under R version 3.3.3
```

```
suppressMessages(libraries("formattable", "dplyr", "tidyr", "ggplot2"))
```

```
## Warning: package 'formattable' was built under R version 3.3.3

## Warning: package 'dplyr' was built under R version 3.3.3

## Warning: package 'tidyr' was built under R version 3.3.3

## Warning: package 'ggplot2' was built under R version 3.3.3
```

```
# loading and reading data file from my desktop
trainDataSet <- read.csv("pml-training.csv", na.strings = c("", "NA"), header = TRUE)
testDataSet  <- read.csv("pml-testing.csv",  na.strings = c("", "NA"), header = TRUE)
```

```
# data dimension with row and columns
rbind ( trainDataSet = dim(trainDataSet), testDataSet = dim(testDataSet) )
```

```
##              [,1] [,2]
## trainDataSet 19622  160
## testDataSet     20  160
```

2.a. Row-Columnar percentile presentation of classe(A-E) variables by each user

This columnar overview rendered in a 100% scale, which displays, how each user did their exercise regime(A-E), in what percentage of the total workout sequence.

```
# percentile projection of classe elements by user name
trainDataSet %>% count(classe, user_name) %>% group_by(user_name) %>% mutate(n=percent(n/sum(n),0))%>% s
```

user_name

A

B

C

D

E

adelmo

30%

20%

19%

13%

18%

carlitos

27%

22%

16%

16%

20%

charles

25%

21%

15%

18%

20%

eurico

28%

19%

16%

19%

18%

jeremy

35%

14%

19%

15%

17%

pedro

25%

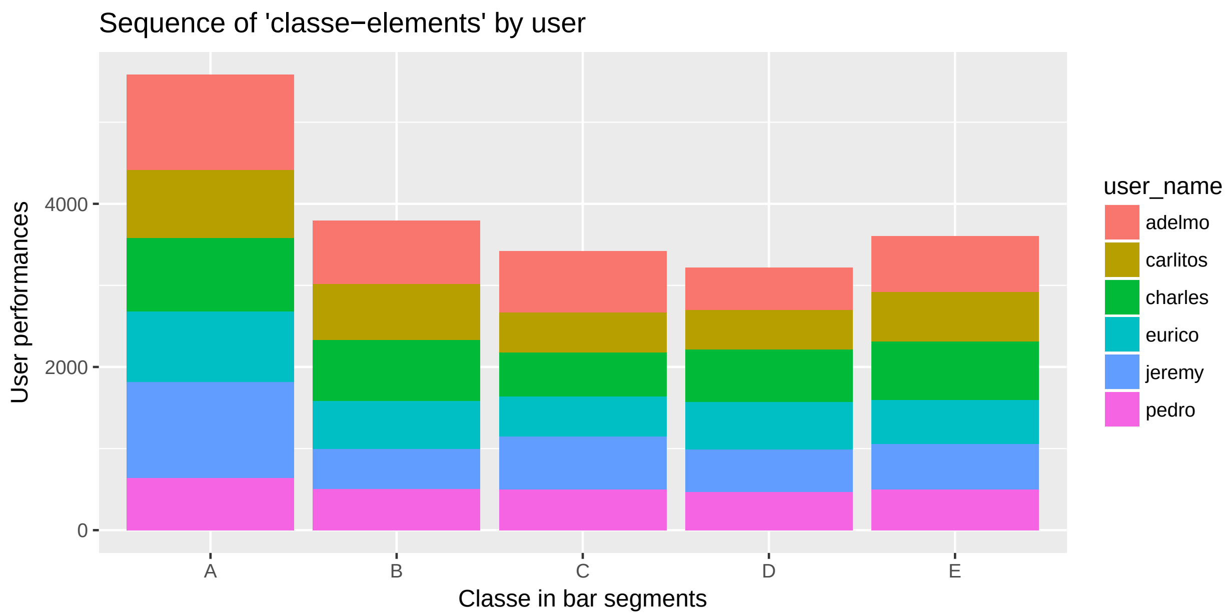
19%

19%

18%

19%

```
ggplot(trainDataSet, aes(x=classe, fill=user_name)) + geom_bar() + xlab("Classe in bar segments") + yla
```



Plot analysis: In this plots we can see that the all participants did 'Classe A' the most number of times and then slowly down to (B-E) pattern. They all started doing biceps curls the proper way (Class A), then proceeded with Class B, C to E. This plot gives us a percentile representation of each classe variable by each user which projects a visible exercise manner.

3. DataSet Partition and Exploratory data Cleaning:

```
suppressMessages(library(caret))
```

```
## Warning: package 'caret' was built under R version 3.3.3
```

```
# Create Data Partition with 0.75 is training and 0.25 test dataset
inTrain <- createDataPartition(trainDataSet$classe, p=0.75, list=FALSE)
TrainSet <- trainDataSet[inTrain, ]
TestSet  <- trainDataSet[-inTrain,]

# quick data-dimension after data partition
rbind ( TrainSet = dim(TrainSet), TestSet = dim(TestSet) )
```

```
##           [,1] [,2]
## TrainSet 14718  160
## TestSet   4904  160
```

```
#> **Note: some machine learning algoriths do not accept 'NA' values inside the DataSet.So we will do s

# checking number of columns have 'NA' values with percentile projeciton in a table
table (NA_Value_Percent <- round(colMeans(is.na(TrainSet)), 2))
```

```
##
##    0 0.98
##   60  100
```

Note: We see that 100-variables have more than 98 percent data with “NA” input ‘filled-in’ and only 60-variables have complete data set. Variables with 98% data is ‘NA’ doesn’t make any quantifiable effect in decision making anlytic processes.

```
# so we'd eliminate all variable-columns, where more than 96% of the input are 'NA'
All_NA_columns <- sapply(TrainSet, function(x) mean(is.na(x))) > 0.96

# removing columns with 96% 'NA' only input from both 'Train and Test' dataset
TrainSet <- TrainSet[, All_NA_columns == FALSE]
TestSet  <- TestSet [, All_NA_columns == FALSE]

# a quick view of how many 'variable-columns' left after 'NA-elimination' process
rbind(TrainSet = dim(TrainSet), TestSet = dim(TestSet))
```

```
##           [,1] [,2]
## TrainSet 14718   60
## TestSet   4904   60
```

3.a. Covariates variation check

```
# covariates variability check by setting 'saveMetrics = TRUE', return a data frame with predictor info
nzv <- nearZeroVar(TrainSet, saveMetrics = TRUE)
head(nzv)
```

```
##                      freqRatio percentUnique zeroVar   nzv
## X                     1.000000  100.00000000   FALSE FALSE
## user_name             1.116610    0.04076641   FALSE FALSE
## raw_timestamp_part_1  1.133333    5.68691398   FALSE FALSE
## raw_timestamp_part_2  1.000000   88.72808806   FALSE FALSE
## cvtd_timestamp        1.001826    0.13588803   FALSE FALSE
## new_window           45.872611    0.01358880   FALSE  TRUE
```

Analsis: We see that most of the near-zero-variables(nzv) are ‘false’, so we don’t need to eliminate any covariates.For further Simplification we will remove some unwarranted columns (‘row-index’ to ‘not-relevant’) from the dataset.

```
TrainSet <- TrainSet[, -(1:7)]
TestSet  <- TestSet [, -(1:7)]

# final dataSet dimension after all irrelevant column elimination
rbind ( TrainSet = dim(TrainSet), TestSet = dim(TestSet))
```

```
##           [,1] [,2]
## TrainSet 14718   53
## TestSet   4904   53
```

4. Machine Learning Algorithms with Cross Validation:

Here, I have used multiple Machine Learning algorithim in searching for high level model accuracy. I have used four algorithms Decision Tree, Linear Discriminant Analysis(lda), Gradient Boosting Method(gbm) and Random Forest(rf) to validate my search. Cross validation processes were included in ‘trainControl’ method with number of folds added. I used parallel-processing feature to reduce ‘data-processing’ time with ‘gbm’ and ‘rf’ model. I also used a confusion Matrix plot to visualize the level of accuracy of the classe variables with ‘rf’ model-algorithm only.

Model.01: Decision (Classification) Tree

```
# setting seed and loading library 'rattle' for decision tree
suppressMessages(library(rattle));set.seed(666)
```

```
## Warning: package 'rattle' was built under R version 3.3.3
```

```
# designing the tree using 'rpart' method
control_dt <- trainControl(method="cv", number = 10)
model_Tree <- train(classe~., method = "rpart", data = TrainSet, trControl = control_dt)
```

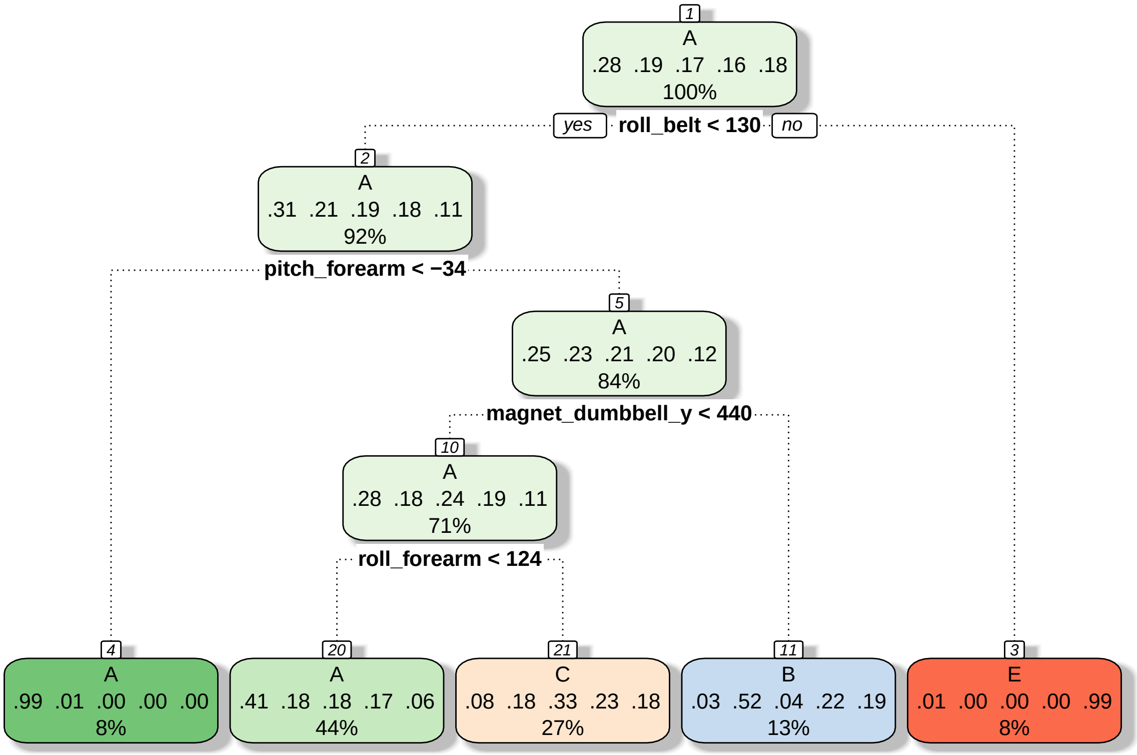
```
## Loading required package: rpart
```

```
## Warning: package 'rpart' was built under R version 3.3.3
```

```
# displaying 'model_Tree' node and leaf detail
print(model_Tree$finalModel, digits = 4)
```

```
## n= 14718
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 14718 10530 A (0.28 0.19 0.17 0.16 0.18)
##    2) roll_belt< 130.5 13469  9294 A (0.31 0.21 0.19 0.18 0.11)
##      4) pitch_forearm< -33.95 1161     7 A (0.99 0.006 0 0 0) *
##      5) pitch_forearm>=-33.95 12308  9287 A (0.25 0.23 0.21 0.2 0.12)
##       10) magnet_dumbbell_y< 439.5 10417  7461 A (0.28 0.18 0.24 0.19 0.11)
##         20) roll_forearm< 123.5 6478  3819 A (0.41 0.18 0.18 0.17 0.061) *
##         21) roll_forearm>=123.5 3939  2645 C (0.075 0.18 0.33 0.23 0.18) *
##       11) magnet_dumbbell_y>=439.5 1891   906 B (0.034 0.52 0.042 0.22 0.19) *
##    3) roll_belt>=130.5 1249    10 E (0.008 0 0 0 0.99) *
```

```
# visualizing the decision tree with all detail 'leaf-palletes'
fancyRpartPlot(model_Tree$finalModel)
```



Rattle 2017−Aug−09 13:08:21 paralax11

```
# running the 'rpart' model on 'TestSet' data and measure model accuracy rate
Test_pred <- predict(model_Tree, newdata = TestSet)
confusionMatrix(Test_pred, TestSet$classe)$overall['Accuracy']
```

```
## Accuracy
## 0.487969
```

Upshot: The accuracy rate with 'rpart' model on 'TestSet' data is 0.490, which is significantly lower and needs newer model exploration.

Model.02: Linear Discriminant Analysis (lda)

```
suppressMessages(library(MASS));set.seed(459)

# setting 'trainControl' feature for the 'lda' model with 8-fold cross-validation method
control_lda <- trainControl(method="cv", number = 10)
model_lda  <- train(classe~., trControl = control_lda, method="lda", data=TrainSet)

# using predict method to verify the model with 'TestSet' data and display model accuracy
lda_pred <- predict(model_lda, TestSet)
confusionMatrix(lda_pred, TestSet$classe)$overall['Accuracy']
```

```
##  Accuracy
## 0.7071778
```

Upshot: ‘lda’ model accuracy rate now rose up to at 0.70 on ‘TestSet’ data.

Model.03: Gradient Boosting Method (gbm)

Note: ‘gbm’ and Random Forest(rf) models are computationally intensive, I have decided to use parallel processing to reduce computation timing. Parallel processing gave me a significant reduction(almost 60%, about 12 minutes) of time savings in ML-code processing.

```
# all necessary library for 'gbm' model including (parallel and doParallel) for faster processing
suppressMessages(libraries("gbm", "plyr", "dplyr", "doParallel"));set.seed(9515)
```

```
## Warning: package 'gbm' was built under R version 3.3.3

## Warning: package 'survival' was built under R version 3.3.3

## Warning: package 'doParallel' was built under R version 3.3.3

## Warning: package 'foreach' was built under R version 3.3.3

## Warning: package 'iterators' was built under R version 3.3.3
```

```
# leaving a single core fo the operating system and registering the cluster
cluster <- makeCluster(detectCores() - 1)
registerDoParallel(cluster)

#> ** Note: 'trainControl' with repeated-cross-validation method, number specifies number of folds for

control_gbm <- trainControl(method = "repeatedcv", number = 10, allowParallel = TRUE)
model_gbm <- train(classe~., preProcess= c("center", "scale"), trControl = control_gbm, method="gbm", da

# applying 'gbm' model on 'TestSet' data
gbm_pred <- predict(model_gbm, TestSet)

# confusion Matrix summary statistics with model 'accuracy' rate
print(confusionMatrix(gbm_pred, TestSet$classe), digits = 4)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1375   16    0    1    0
##          B   13  900   26    1   10
##          C    5   32  818   24    5
##          D    2    0   11  772    9
##          E    0    1    0    6  877
##
## Overall Statistics
##
##                Accuracy : 0.967
##                  95% CI : (0.9616, 0.9718)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
```

```
##                   Kappa : 0.9582
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            0.9857   0.9484   0.9567   0.9602   0.9734
## Specificity            0.9952   0.9874   0.9837   0.9946   0.9983
## Pos Pred Value         0.9878   0.9474   0.9253   0.9723   0.9921
## Neg Pred Value         0.9943   0.9876   0.9908   0.9922   0.9940
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2804   0.1835   0.1668   0.1574   0.1788
## Detection Prevalence   0.2838   0.1937   0.1803   0.1619   0.1803
## Balanced Accuracy      0.9904   0.9679   0.9702   0.9774   0.9858
```

```r
# confusionMatrix(gbm_pred, TestSet$classe)$StatisticsbyClass
confusionMatrix(gbm_pred, TestSet$classe)$overall['Accuracy']
```

```
##  Accuracy
## 0.9669657
```

Upshot: There is a considerable accuracy rate increase up to (0.963) compare to 'lda' model (0.701).

Model.04: Random Forest (rf)

```r
# loading library, setting seed and 'registering-parallel-processing'
suppressMessages(library(randomForest));set.seed(969)
```

```
## Warning: package 'randomForest' was built under R version 3.3.3
```

```r
registerDoParallel(cluster)

# setting control feature with method 'repeatedcv' and adding parallel processing cluster
Control_Rfo <- trainControl(method = "repeatedcv", number = 9, allowParallel = TRUE)

# running 'rf' model with proprocessing method and predefined control feature
model_Rfo  <- train(classe~., method = "rf", preProcess=c("center", "scale"),  data=TrainSet, trControl

# Evaluating the model on 'TestSet' data and calculating confusionMatrix
Rfo_pred <- predict(model_Rfo, TestSet)
confusion_Rfo <- confusionMatrix(Rfo_pred, TestSet$classe)

# confusion Matrix summary statistics with 'accuracy' rate
print(confusionMatrix(Rfo_pred, TestSet$classe), digits = 4 )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    A    B    C    D    E
##          A 1395    3    0    0    0
##          B    0  943   14    0    0
##          C    0    3  839   15    0
##          D    0    0    2  789    2
##          E    0    0    0    0  899
##
```

```
## Overall Statistics
##
##                Accuracy : 0.992
##                  95% CI : (0.9891, 0.9943)
##     No Information Rate : 0.2845
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.9899
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: A Class: B Class: C Class: D Class: E
## Sensitivity            1.0000   0.9937   0.9813   0.9813   0.9978
## Specificity            0.9991   0.9965   0.9956   0.9990   1.0000
## Pos Pred Value         0.9979   0.9854   0.9790   0.9950   1.0000
## Neg Pred Value         1.0000   0.9985   0.9960   0.9964   0.9995
## Prevalence             0.2845   0.1935   0.1743   0.1639   0.1837
## Detection Rate         0.2845   0.1923   0.1711   0.1609   0.1833
## Detection Prevalence   0.2851   0.1951   0.1748   0.1617   0.1833
## Balanced Accuracy      0.9996   0.9951   0.9884   0.9902   0.9989
```

```
confusion_Rfo$overall['Accuracy']
```

```
##  Accuracy
## 0.9920473
```

```
if(FALSE){
# ploting the 'confusion Matrix' of "Random Forest" model for classe-steps verification
plot.03 <- plot(confusion_Rfo$table, col = confusion_Rfo$byClass, main = paste("Random Forest Model Accu
            round(confusion_Rfo$overall['Accuracy'], 4)))
}
```

Upshot: Random forest model by far is predicting the best ‘accuracy rate’ 0.9955 with least ‘out-of-sample error’ is 0.004 rate.

Out-Of-Sample error calculation:

Random Forest Model out of sample error:(1 - 0.9955139) = **0.005**

Gradient Boosting Model out of sample error:(1- 0.9665579) = **0.040**

Linear Discriminant Analysis out of sample error:(1- 0.694739) = 0.305

Classification or Decision tree out of sample error:(1- 0.4912316) = 0.508

Note: Every single time running these algorithms produces slightly different accuracy rates and tree pallets.

Applying ML-models on 20 test-case data set:

Applying only three machine learning(‘rf’,‘gbm’,‘lda’) algorithm model on to the 20 test-cases (‘testDataSet’) dataset, provided with the project instruction for level-based prediction.

```
print(predict(model_Rfo, newdata = testDataSet))
```

```
##  [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
```

```
print(predict(model_gbm, newdata = testDataSet))
```

```
##  [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
```

Analysis: Remarkably 'random-forest' and 'gbm' model both made exact same 'level' of prediction on 'testDataSet', which proves high level of accuracy proximity.

```
print(predict(model_lda, newdata = testDataSet))
```

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

```
# finally folding the parallel-processing cluster
stopCluster(cluster)
# forcing 'R' to return single threading process
registerDoSEQ()
```