Practical Machine Learning

Christian Peikert

19 Juli 2018

Background

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity relatively inexpensively. These type of devices are part of the quantified self movement – a group of enthusiasts who take measurements about themselves regularly to improve their health, to find patterns in their behavior, or because they are tech geeks. One thing that people regularly do is quantify how much of a particular activity they do, but they rarely quantify how well they do it. In this project, your goal will be to use data from accelerometers on the belt, forearm, arm, and dumbbell of 6 participants. They were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information is available from the website here: http://groupware.les.inf.puc-rio.br/har (see the section on the Weight Lifting Exercise Dataset).

Data loading

```
pml_training <-
  read.csv(
    "data/pml-training.csv",
    sep = ',',
    header = TRUE,
    row.names = NULL,
    check.names = FALSE,
    stringsAsFactors = FALSE
  )

pml_testing <-
  read.csv(
    "data/pml-testing.csv",
    sep = ',',
    header = TRUE,
    row.names = NULL,
    check.names = FALSE,
    stringsAsFactors = FALSE
  )

# dim(pml_training)
# dim(pml_testing)
# str(pml_training)
# str(pml_testing)

colnames(pml_training)[1] <- 'id_row'
colnames(pml_testing)[1] <- 'id_row'
pml_training$id_row <- NULL
pml_testing$id_row <- NULL
```

The training data set contains 19622 and 158 columns. The column “classe” contain the correct classification which should be achieved.

5580, 3797, 3422, 3216, 3607

Class B to E are uniformly represented, however class A has a much higher frequency.

Typo crrection

```
colnames(pml_training) <- sapply(colnames(pml_training),function(x)str_replace(x,'picth','pitch'))
colnames(pml_testing) <- sapply(colnames(pml_testing),function(x)str_replace(x,'picth','pitch'))
```

Correction of the colname typo 'picth' to 'pitch'

Data cleaning

```
# compare valible cols in train and test set
datasetColumnIntersection <- intersect(colnames(pml_training),colnames(pml_testing))
datasetColumnOutersection <- sort(c(setdiff(colnames(pml_training), colnames(pml_testing)),
                                    setdiff(colnames(pml_testing), colnames(pml_training))))

# assignment of columns to there belonging measuring instruments
accelerometers <- c("belt", "forearm", "arm","dumbbell")
accelerometers_cols  <- lapply(accelerometers,function(x)colnames(pml_training)[grep(paste0('([^A-Za-z]
names(accelerometers_cols) <- accelerometers

# identifcation columns that can be found for all four instruments
accelerometers_cols_temp <- accelerometers_cols
for(n in names(accelerometers_cols_temp)){
  accelerometers_cols_temp[[n]] <- sapply(accelerometers_cols_temp[[n]], function(x){
    m <- str_match(x,paste0('(.*)','_(',n,')','([^A-Za-z\\d]{0,1})','(.*)'))
    if(any(!is.na(m)))paste(m[c(2)],collapse = '')
  })
}
v <- Vennerable::Venn(accelerometers_cols_temp)
intersectionSets <- attr(v,'IntersectionSets')


cols <- unique(unlist(sapply(intersectionSets$`1111`,function(x)colnames(pml_training)[startsWith(colnam

accelerometers_cols_clean  <- lapply(accelerometers,function(x)cols[grep(paste0('([^A-Za-z]|^)',x,'([^A
names(accelerometers_cols_clean) <- accelerometers


# cleaning columns and converting features to numeric
pml_training_cleaned <- pml_training
pml_training_cleaned[unlist(accelerometers_cols_clean)] <- sapply(pml_training_cleaned[unlist(accelerome

# validValuesPerRow <- apply(pml_training_cleaned[unlist(accelerometers_cols_clean)],1,function(x)sum(!
# table(validValuesPerRow)

# identification of columns containing less missing values
validValuesPerCol <-  apply(pml_training_cleaned[unlist(accelerometers_cols_clean)],2,function(x)sum(!is
table(validValuesPerCol)
```

```
## validValuesPerCol
##     0   321   322   323   326   328   329   374   395   396   397   401
##     6     1     4     1     1     1     1     2     2     4     1     4
##   402   404   406 19622
```

```
##     1     1    63    52
```

```
valid_cols <- sort(names(validValuesPerCol)[validValuesPerCol == max(as.numeric(names(table(validValues

for(n in names(accelerometers_cols)){
  accelerometers_cols[[n]] <- accelerometers_cols[[n]][accelerometers_cols[[n]] %in% valid_cols]
}

# reducing the columns to the bedefined set
# dim(pml_training_cleaned)
pml_training_cleaned <- pml_training_cleaned[c('classe',unlist(accelerometers_cols))]
pml_training_cleaned <- pml_training_cleaned[complete.cases(pml_training_cleaned),]
# dim(pml_training_cleaned)
classe_index_column <- 1

# converting classe column to factor
classes <- pml_training_cleaned$classe
classes <- as.factor(classes)
levels(classes) <- 1:length(levels(classes))
# classes <- as.numeric(classes)
pml_training_cleaned$classe <- classes


# test for near zero variance of columns
nzv_control <- nearZeroVar(pml_training_cleaned[,-1])

`%ni%` <- Negate(`%in%`)
```

First I quickly checked the intersection of columns of the training and test set just to get sure that we will not focus on feature that are not available in the test set. As aspected "classe" is missing in the test set. In addition the column "problem_id" is also unique. Next I compared the different columns which belong to the four measuring instruments and defined a set of columns that are available for all four instruments.

In further check which rows contain how many valid values.

Finally, we got a 19622, 53 data.frame containing 4*13 numeric columns and the 'classe' as factor.

Creation of Sets

```
partition <- createDataPartition(y=pml_training_cleaned[,classe_index_column], p=0.75, list=FALSE)

trainingSet <- pml_training_cleaned[partition,]
testingSet <- pml_training_cleaned[partition,]
testing20cases <- pml_testing[,unlist(accelerometers_cols)]

# dim(trainingSet)
# dim(testingSet)
# dim(testing20cases)
```

Three sets were defined for the further processing. Training data were split in 75% train and 25% test data. The third data frame contains the to predict observations for the quiz.

Data.frames:

trainingSet: rows= 14718, columns= 53

testingSet: rows= 14718, columns= 53

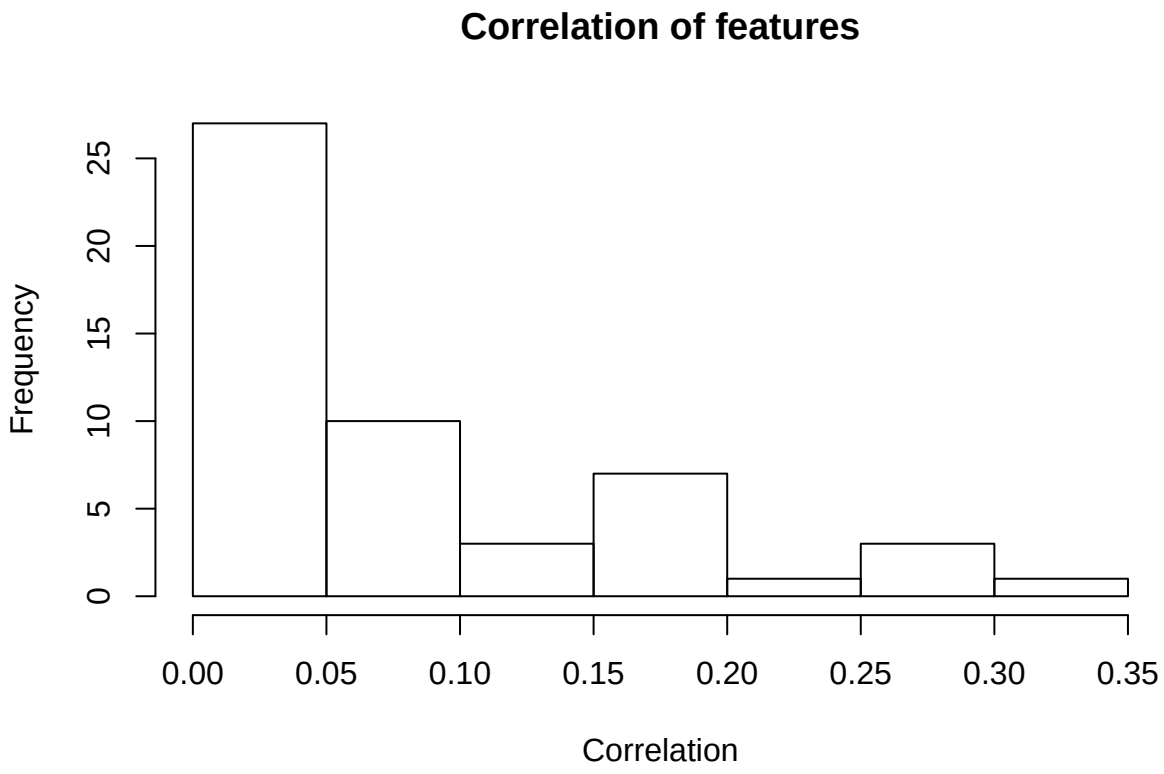
testing20cases: rows= 20, columns= 52

Correlation

```
featureCor <- abs(cor(subset(trainingSet,select = names(trainingSet) %ni% 'classe'),as.numeric(training

cat("In the histogram of correlation you can the that the features obviously only show weak correlation
```

In the histogram of correlation you can the that the features obviously only show weak correlation.

```
h1  <- hist(featureCor, main = 'Correlation of features',xlab='Correlation')
```




```
show(h1)
```

$breaks [1] 0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35

$counts [1] 27 10 3 7 1 3 1

$density [1] 10.3846154 3.8461538 1.1538462 2.6923077 0.3846154 1.1538462 [7] 0.3846154

$mids [1] 0.025 0.075 0.125 0.175 0.225 0.275 0.325

$xname [1] “featureCor”

$equidist [1] TRUE

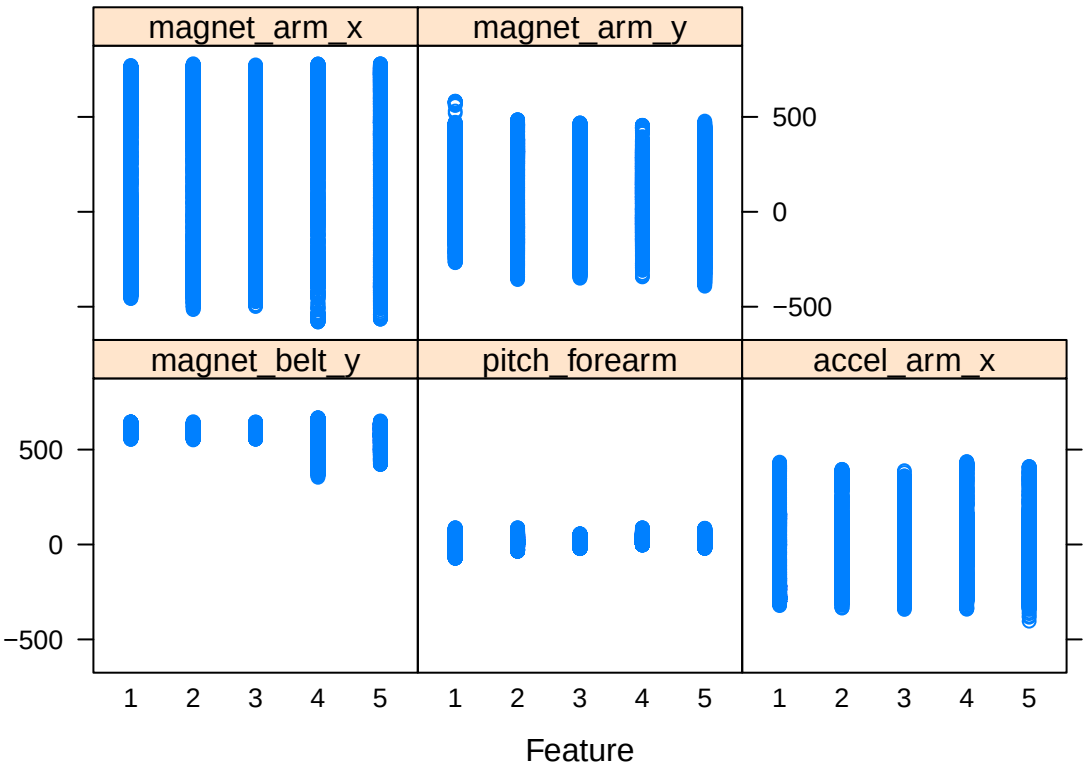
attr(,“class”) [1] “histogram”

```
bestCorrelations <- subset(as.data.frame(as.table(featureCor)), abs(Freq)>0.3)
```

```
coi <- rownames(featureCor)[which(featureCor > 0.20)]


cat("In a strip featurePlot we can see that all feature show the same distribution among the 5 classe's
```

In a strip featurePlot we can see that all feature show the same distribution among the 5 classe's

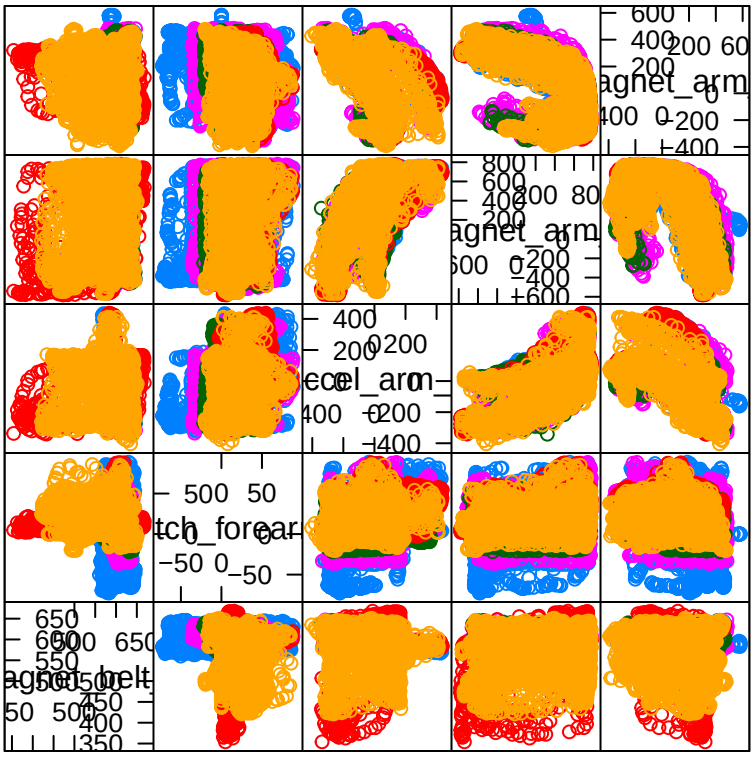
```
f1 <- featurePlot(x = subset(trainingSet,select = coi),
                  y = trainingSet[,classe_index_column],
                  plot = "strip"
)
show(f1)
```




```
cat("By the pairs featurePlot of the highest correlating features it is not possible to identify a clea
```

By the pairs featurePlot of the highest correlating features it is not possible to identify a clear feature for classification

```
f2 <- featurePlot(x = subset(trainingSet,select = coi),
                  y = trainingSet[,classe_index_column],
                  plot = "pairs"
)
show(f2)
```

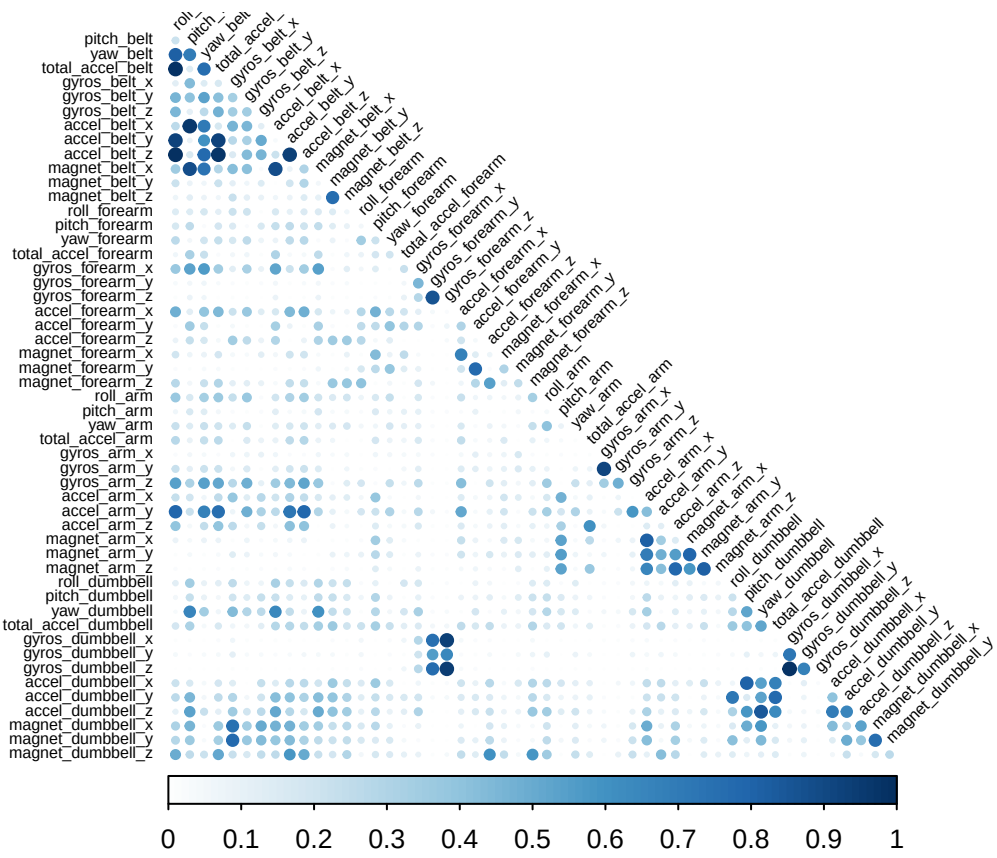


Scatter Plot Matrix

```
correlationMatrix <- cor(subset(trainingSet,select = names(trainingSet) %ni% 'classe'))
correlationMatrix <-  abs(correlationMatrix)
highlyCorrelated <- findCorrelation(correlationMatrix, cutoff=0.8)
excludeColumns <- c(highlyCorrelated, classe_index_column)
cat("Here you can see the plot of correction")
```

Here you can see the plot of correction

```
corrplot(correlationMatrix
         ,method=c("circle")
         ,type="lower"
         # ,order="hclust"
         ,hclust.method = "ward.D2"
         ,tl.cex=0.50
         ,tl.col="black"
         ,tl.srt = 45
         ,diag = F
         # ,cl.lim=c(-1,1)
         ,cl.lim=c(0,1)
         ,addgrid.col = NA
)
```



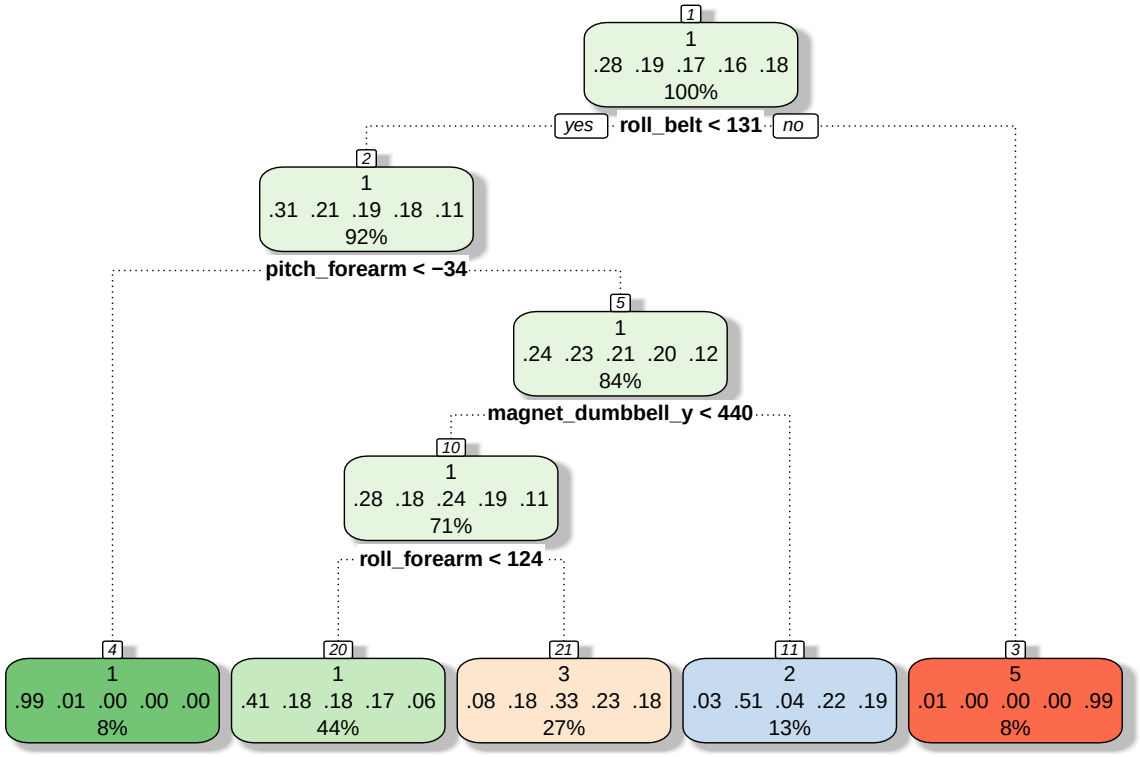

```
# c2 <- corrplot(correlationMatrix[highlyCorrelated,highlyCorrelated]
#          ,method=c("circle")
#          ,type="lower"
#          ,order="hclust"
#          ,hclust.method = "ward.D2"
#          ,tl.cex=0.50
#          ,tl.col="black"
#          ,tl.srt = 45
#          ,diag = F
#          # ,cl.lim=c(-1,1)
#           ,cl.lim=c(0,1)
#          ,addgrid.col = NA
#          )
```

```
df_temp <- subset(trainingSet,select = names(trainingSet) %ni% 'classe')
res.pca = PCA(X = df_temp
              ,scale.unit = FALSE #  is importent, data are not scale correctly
              ,ncp = ncol(df_temp) # number of final principal component
              ,graph = FALSE)

fv1 <- fviz_eig(res.pca, addlabels = TRUE, ylim = c(0, 50))
fv2 <- fviz_pca_ind(res.pca, repel = TRUE, label="none", habillage = trainingSet$classe)
```

Also a reduction of dimantion by PCA doesn't allow us a good classifcation of data.

decision tree

```
# library('rpart')
# library('rpart.plot')
# dTreeModel <- rpart(classe ~ ., data = trainingSet, method= 'class')
# fancyRpartPlot(dTreeModel)

if(F){
  dTreeModel <- train(classe ~ ., data = trainingSet, method= 'rpart')
  saveRDS(dTreeModel,"dTreeModel.RDS")
}else{
  dTreeModel <- readRDS("dTreeModel.RDS")
}
# plot(dTreeModel$finalModel, uniform = TRUE, main="Classification Tree")
# text(dTreeModel$finalModel, use.n=TRUE, all= TRUE, cex=0.8)
fp <- fancyRpartPlot(dTreeModel$finalModel)
```



Rattle 2018−Jul−27 17:30:55 cpeikert

```
cat('Here you can see the result of the trained decision tree!')
```

```
## Here you can see the result of the trained decision tree!
```

```
show(fp)
```

```
## NULL
```

```
dTreePrediction <- predict(dTreeModel$finalModel,testingSet,type='class')
dTreeValidation <-  confusionMatrix(dTreePrediction, testingSet$classe)
dTreeValidation
```

```
## Confusion Matrix and Statistics
```

```
##
##           Reference
## Prediction    1    2    3    4    5
##          1 3806 1175 1178 1070  408
##          2   63  980   78  430  372
##          3  306  693 1311  912  712
##          4    0    0    0    0    0
##          5   10    0    0    0 1214
##
## Overall Statistics
##
##                Accuracy : 0.4967
##                  95% CI : (0.4886, 0.5049)
##     No Information Rate : 0.2843
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.3424
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: 1 Class: 2 Class: 3 Class: 4 Class: 5
## Sensitivity            0.9094  0.34410  0.51071   0.0000  0.44863
## Specificity            0.6363  0.92056  0.78413   1.0000  0.99917
## Pos Pred Value         0.4984  0.50962  0.33325      NaN  0.99183
## Neg Pred Value         0.9465  0.85401  0.88353   0.8361  0.88943
## Prevalence             0.2843  0.19350  0.17441   0.1639  0.18386
## Detection Rate         0.2586  0.06659  0.08907   0.0000  0.08248
## Detection Prevalence   0.5189  0.13066  0.26729   0.0000  0.08316
## Balanced Accuracy      0.7729  0.63233  0.64742   0.5000  0.72390
```

```
dTreeValidationAcc <- as.numeric(dTreeValidation$overall['Accuracy'])
cat('Howevery the model have only a weak accuracy (',dTreeValidationAcc,')')
```

```
## Howevery the model have only a weak accuracy ( 0.4967387 )
```

random Forest

```
if(F){
  rForestModel_train <- train(classe ~ ., data = trainingSet, method= 'rf', prox=TRUE)#saveRDS(rForestM
}else{
  rForestModel_train <- readRDS("rForestModel_train.RDS")
}
rForestModel_train
```

```
## Random Forest
##
## 14718 samples
##    52 predictor
##     5 classes: '1', '2', '3', '4', '5'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 14718, 14718, 14718, 14718, 14718, 14718, ...
## Resampling results across tuning parameters:
```

```
##
##   mtry  Accuracy   Kappa
##    2    0.9893045  0.9864706
##   27    0.9893639  0.9865462
##   52    0.9765902  0.9703879
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 27.
```

```
rForest_trainPrediction <- predict(rForestModel_train,testingSet)
rForest_trainValidation <-  confusionMatrix(rForest_trainPrediction, testingSet$classe)

rForest_trainPrediction20cases <- predict(rForestModel_train,testing20cases)

rForest_trainValidationAcc <- rForestModel_train$results$Accuracy[rForestModel_train$results$mtry == as

cat('The random forest model trained by the carrot::train function results in a accuracy of',rForest_tra
```

```
## The random forest model trained by the carrot::train function results in a accuracy of 0.9893639 .
```

```
if(F){
  rForestModel_randomFores <- randomForest(
    x=trainingSet[, -classe_index_column],
    y=trainingSet$classe,
    xtest=testingSet[, -classe_index_column],
    ytest=testingSet$classe,
    ntree=100,
    keep.forest=TRUE,
    proximity=TRUE)
  saveRDS(rForestModel_randomFores,"rForestModel_randomForest.RDS")
}else{
  rForestModel_randomForest <- readRDS("rForestModel_randomForest.RDS")
}
rForest_randomForestPrediction <- predict(rForestModel_randomForest,testingSet)
rForest_randomForestValidation <-  confusionMatrix(rForest_randomForestPrediction, testingSet$classe)
rForest_randomForestPrediction20cases <- predict(rForestModel_randomForest,testing20cases)

rForestrForestModel_randomForestValidationAcc <- 1-sum(rForestModel_randomForest$confusion[,'class.erro

cat('The random forest model trained by the randomForest::randomForest function results in a accuracy o
```

```
## The random forest model trained by the randomForest::randomForest function results in a accuracy of
```

```
saveRDS(rForest_randomForestPrediction20cases,"rForest_randomForestPrediction.RDS") # all correct
```

Linear discriminant Analysis and naive Bayes

```
if(F){
  ldaModel <- train(classe ~ ., data = trainingSet, method= 'lda')
  saveRDS(ldaModel,"ldaModel.RDS")
}else{
  ldaModel <- readRDS("ldaModel.RDS")
}
# nbModel <- train(classe ~ ., data = trainingSet, method= 'nb')
# saveRDS(nbModel,"nbModel.RDS")
```

```
if(F){
  trainCtrl <- trainControl(method   =   "repeatedcv",
                            repeats   =   3,
                            number = 5,
                            classProbs   =   TRUE,
                            savePredictions = 'final',
                            summaryFunction   =   twoClassSummary)

  colnames(trainingSet)
  temp_trainingSet <- trainingSet
  temp_trainingSet$classe <- as.factor(paste0('c',temp_trainingSet$classe))
  levels(temp_trainingSet$classe) <- paste0('c',levels(trainingSet$classe))
  nbModel <- train(classe ~ ., data = temp_trainingSet,
                   method   =   'nb',
                   # trControl   =   trainCtrl,
                   metric   =   "ROC", #for classification
                   preProc = c("center", "scale"),
                   tuneLength = 10)
  nbModel <- train(classe ~ ., data = temp_trainingSet, method= 'nb')


  saveRDS(nbModel,"nbModel.RDS")
}else{
  nbModel <- readRDS("nbModel.RDS")
}

ldaPrediction <- predict(ldaModel,testingSet)
nbPrediction <- predict(nbModel,testingSet)
compareLdaNb <- table(ldaPrediction,nbPrediction)

ldaValidation <- confusionMatrix(ldaPrediction, testingSet$classe)

temp_testingSet <- testingSet
temp_testingSet$classe <- as.factor(paste0('c',temp_testingSet$classe))
levels(temp_testingSet$classe) <- paste0('c',levels(testingSet$classe))
nbValidation <- confusionMatrix(nbPrediction, temp_testingSet$classe)

ldaModelValidationAcc <- ldaModel$results$Accuracy
nbModelValidationAcc <- nbModel$results$Accuracy[nbModel$results$usekernel==nbModel$bestTune$usekernel]

cat('The linear discriminant analysis Model results in an accuracy of',ldaModelValidationAcc,'and the m
```

```
## The linear discriminant analysis Model results in an accuracy of 0.6988645 and the model of naive ba
```

Boosting

```
if(F){
  boostingModel <- train(classe ~ ., data = trainingSet, method= 'gbm', verbose=FALSE)
  saveRDS(boostingModel,"boostingModel.RDS")
}else{
  boostingModel <- readRDS("boostingModel.RDS")
}
boostingPrediction <- predict(boostingModel,testingSet)
```

```
print(boostingModel)
```

```
## Stochastic Gradient Boosting
##
## 14718 samples
##    52 predictor
##     5 classes: '1', '2', '3', '4', '5'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 14718, 14718, 14718, 14718, 14718, 14718, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7489860  0.6818796
##   1                  100      0.8161624  0.7673608
##   1                  150      0.8501164  0.8103405
##   2                   50      0.8510081  0.8112724
##   2                  100      0.9033313  0.8776752
##   2                  150      0.9291536  0.9103657
##   3                   50      0.8930411  0.8646106
##   3                  100      0.9393173  0.9232295
##   3                  150      0.9576039  0.9463705
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
boostingValidation <-  confusionMatrix(boostingPrediction, testingSet$classe)
boostingModel
```

```
## Stochastic Gradient Boosting
##
## 14718 samples
##    52 predictor
##     5 classes: '1', '2', '3', '4', '5'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 14718, 14718, 14718, 14718, 14718, 14718, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  Accuracy   Kappa
##   1                   50      0.7489860  0.6818796
##   1                  100      0.8161624  0.7673608
##   1                  150      0.8501164  0.8103405
##   2                   50      0.8510081  0.8112724
##   2                  100      0.9033313  0.8776752
##   2                  150      0.9291536  0.9103657
##   3                   50      0.8930411  0.8646106
##   3                  100      0.9393173  0.9232295
```

```
##   3                  150      0.9576039  0.9463705
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were n.trees = 150,
##  interaction.depth = 3, shrinkage = 0.1 and n.minobsinnode = 10.
```

```
boostingValidationAcc <- boostingModel$results$Accuracy[
  boostingModel$results$n.trees ==  boostingModel$bestTune$n.trees &
    boostingModel$results$interaction.depth ==  boostingModel$bestTune$interaction.depth
  ]
cat('The gradient boosting machine achieved an accuracy of',boostingValidationAcc,'.')
```

```
## The gradient boosting machine achieved an accuracy of 0.9576039 .
```

```
accuracy_list <- c(
  rForest_train = rForest_trainValidationAcc,
  rForestrForestModel_randomForest = rForestrForestModel_randomForestValidationAcc,
  lda = ldaModelValidationAcc,
  nb = nbModelValidationAcc,
  boosting = boostingValidationAcc
)

accuracy_list < sort(accuracy_list, decreasing = T)
```

```
##                     rForest_train rForestrForestModel_randomForest
##                             FALSE                             FALSE
##                               lda                                nb
##                              TRUE                             FALSE
##                          boosting
##                             FALSE
```

```
cat('The random Forest Model performed best of all tested methods.')
```

```
## The random Forest Model performed best of all tested methods.
```

```
accuracy_list[1]
```

```
## rForest_train
##     0.9893639
```

```
letters <- c('A','B','C','D','E')
final_prediction <- letters[rForest_trainPrediction20cases]
cat('For the quiz the predicted classes are:',final_prediction)
```

```
## For the quiz the predicted classes are: B A B A A E D B A A B C B A E E A B B B
```