Pratical Machine Learning - Course Project

Introduction

Using devices such as Jawbone Up, Nike FuelBand, and Fitbit it is now possible to collect a large amount of data about personal activity. 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, the goal is creating a model to predict the manner in which they did the exercise. To do so, we are going to use the data from the study about Qualitative Activity Recognition of Weight Lifting Exercises [1] that register the accelerometers on the belt, forearm, arm, and dumbell of 6 participants. In this study, the participants were asked to perform barbell lifts correctly and incorrectly in 5 different ways. More information about this data is available on and more details about it can be access by this website http://groupware.les.inf.puc-rio.br/har, in the section on the Weight Lifting Exercise Dataset.

Assignment

As said previously, the goal of your project is to predict the manner in which they did the exercise. This is the "classe" variable in the training set. To do so, it is allowed to use any of the other variables to predict with. This paper must report how built the created model, how were used the cross validation, what is the expected out of sample error, and the cause of the choices maded. After that, it must present the prediction result of the model over 20 different test cases.

```
options(scipen=999)           # make the number printer more readable
Sys.setenv(LANG = "en")      # show messages on english
Sys.setenv(LANGUAGE = "en") # show messages on english
rm(list=ls())                # remove other data from env, if any
set.seed(123)                # set a seed to ensure get always the same results
```

Data

```
trainDataLink <- 'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv'
trainDataFile <- 'pml-training.csv';
testDataLink  <- 'https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv'
testDataFile  <- 'pml-testing';
```

The data for this assignment is divided among trainng data and test data:

Loading Libraries

```
# loading required libraries
# install.packages(c("knitr","ggplot2","data.table","caret","caretEnsemble",
#   "doParallel","e1071","rpart","rpart.plot","rattle","gridExtra"), dependencies = TRUE)
library('knitr')
library('ggplot2')
library('data.table')
library('caret')
```

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

```r
library('devtools')
library('doParallel')
```

```
## Loading required package: foreach
## Loading required package: iterators
## Loading required package: parallel
```

```r
library('e1071')
library('rattle')
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.2.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```r
library('rpart')
library('rpart.plot')
library('gridExtra')
library('ggplotify')
library('cowplot')
```

```
##
## Attaching package: 'cowplot'
## The following object is masked from 'package:ggplot2':
##
##     ggsave
```

```r
library('kableExtra')
library('ggRandomForests')
```

```
## Loading required package: randomForestSRC
##
##  randomForestSRC 2.8.0
##
##  Type rfsrc.news() to see new features, changes, and bug fixes.
##
##
## Attaching package: 'randomForestSRC'
## The following objects are masked from 'package:e1071':
##
##     impute, tune
##
## Attaching package: 'ggRandomForests'
## The following object is masked from 'package:randomForestSRC':
##
##     partial.rfsrc
```

```r
registerDoParallel(cores=4)
```

Loading and preprocessing the data

Download the data and load it into data.table.

```
if ( !file.exists(trainDataFile) ) {
  download.file(trainDataLink, trainDataFile)
}
if ( !file.exists(testDataFile) ) {
  download.file(testDataLink, testDataFile)
}
trainData <- read.csv( trainDataFile, na.strings=c('#DIV/0!', '', 'NA'), stringsAsFactors = FALSE)
testData  <- read.csv( testDataFile,  na.strings=c('#DIV/0!', '', 'NA'), stringsAsFactors = FALSE)
```

Removing Null Columns

Remove all columns with 97% or more rows with null values.

```
MAX_PERCENT_OF_NA_VALUES = 0.97
fields <- names(trainData)
size <-nrow(trainData)
fieldsToRemove <- c()

for(field in fields) {
  column <- trainData[[field]]
  percentOfNA <- ( (length(column[is.na(column)])) / size )
  if( percentOfNA >= MAX_PERCENT_OF_NA_VALUES ) {
    fieldsToRemove[length(fieldsToRemove)+1] <- field;
  }
}
print(paste("removing these fields for having to many empty values: (",paste(fieldsToRemove, collapse =
```

```
## [1] "removing these fields for having to many empty values: ( kurtosis_roll_belt, kurtosis_picth_bel
```

```
# use the same field list to train and test to avoid different field list
trainData <- trainData[ , !(colnames(trainData) %in% fieldsToRemove)]
testData  <- testData[  , !(colnames(testData)  %in% fieldsToRemove)]
```

Replacing Null values by the Median

Replace remaing null values by the median or the most common value.

```
fields <- names(trainData)
commonValues <- c()
for(field in fields) {
  column <- trainData[[field]]
  if(!is.character(column)){
    new_value <- median(column)
  } else {
    new_value <- names(sort(table(column),decreasing=TRUE)[1])
  }
  commonValues[field] <- new_value;
}
replaceNullByCommonValues <- function(dataframe,commonValues) {
  for(field in fields) {
    column <- dataframe[[field]]
    totalNull <- length(column[is.na(column)])
    if(totalNull > 0) {
```

```
      commonValue <- commonValues[field]
      print(paste("replacing",totalNull," null values on",field,"by",commonValue))
      column[is.na(column)] <- commonValue
      dataframe[[field]] <- column
    }
  }
  return(dataframe)
}
trainData <- replaceNullByCommonValues(trainData,commonValues)
testData  <- replaceNullByCommonValues(testData,commonValues)
```

```
## Warning in is.na(column): is.na() applied to non-(list or vector) of type
## 'NULL'
```

```
print("all the null values where replaced by the common values")
```

```
## [1] "all the null values where replaced by the common values"
```

Remove Near Zero Variance Columns

```
nearZeroVarFields <- nearZeroVar(trainData, names = TRUE)
trainData <- trainData[ , !(colnames(trainData) %in% nearZeroVarFields)]
if( length(nearZeroVarFields) > 0 ) {
  cat(paste("removing these fields for having near zero variance: (",paste(nearZeroVarFields, collapse =
} else {
  cat("all fields have a acceptable variance")
}
```

```
## removing these fields for having near zero variance: ( new_window )
```

Remove Id and Time columns

The goal is detect if the exercise is being done correctly or not based on the detected device data. In this goal, when the data was collected or who is the user should not affect the result.

```
fieldsToRemove <- c("X", "user_name", "raw_timestamp_part_1", "raw_timestamp_part_2", "cvtd_timestamp",
trainData <- trainData[ , !(colnames(trainData) %in% fieldsToRemove)]
cat("id columns removed")
```

```
## id columns removed
```

Separate Validation Data

```
set.seed(123)
targetFields <- c("classe")

trainDataRows                <- createDataPartition(y = trainData$classe, p = 0.9, list = FALSE)

trainDataset                 <- trainData[ trainDataRows,]
trainDatasetPredictors       <- trainData[ trainDataRows, !(colnames(trainData) %in% targetFields)]
trainDatasetTarget           <- factor(trainData[ trainDataRows, ]$classe)
```

```r
validationDataset           <- trainData[-trainDataRows,]
validationDatasetPredictors <- trainData[-trainDataRows, !(colnames(trainData) %in% targetFields)]
validationDatasetTarget     <- factor(trainData[-trainDataRows, ]$classe)

cat(paste("train dataset have ",nrow(trainDataset),"rows"),"\n")
```

```
## train dataset have  17662 rows
```

```r
cat(paste("validation dataset have ",nrow(validationDataset),"rows"),"\n")
```

```
## validation dataset have  1960 rows
```

```r
cat(paste("test dataset have ",nrow(testData),"rows"),"\n")
```

```
## test dataset have  20 rows
```

Creating Models

```r
set.seed(123)
folds = 3
modelTrainControl <- trainControl(
  method = "cv",              # for "cross-validation"
  number = folds,             # number of k-folds
  returnResamp = 'final',
  classProb = TRUE,
  returnData = FALSE,
  savePredictions = FALSE,
  verboseIter = TRUE,
  allowParallel = TRUE,
  index=createFolds(trainDataset$classe,k=folds)
)

preProcess=c("pca","center","scale")
modelFitBag                  <- train(classe ~ ., data = trainDataset, method = "treebag",
                                      preProcess = preProcess, trControl=modelTrainControl)
```

```
## Aggregating results
## Fitting final model on full training set
```

```r
modelKNearestNeighbor        <- train(classe ~ ., data = trainDataset, method = "knn",
                                      preProcess = preProcess, trControl=modelTrainControl)
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting k = 5 on full training set
```

```r
modelRecursivePartition      <- train(classe ~ ., data = trainDataset, method = "rpart",
                                      preProcess = preProcess, trControl=modelTrainControl)
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting cp = 0.0331 on full training set
```

```r
modelGradientBoostingMachine <- train(classe ~ ., data = trainDataset, method = "gbm",
                                      preProcess = preProcess, trControl=modelTrainControl)
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting n.trees = 150, interaction.depth = 3, shrinkage = 0.1, n.minobsinnode = 10 on full training s
## Iter   TrainDeviance   ValidDeviance   StepSize   Improve
##      1        1.6094            -nan     0.1000    0.1276
##      2        1.5310            -nan     0.1000    0.0926
##      3        1.4727            -nan     0.1000    0.0735
##      4        1.4271            -nan     0.1000    0.0581
##      5        1.3906            -nan     0.1000    0.0502
##      6        1.3584            -nan     0.1000    0.0460
##      7        1.3299            -nan     0.1000    0.0435
##      8        1.3014            -nan     0.1000    0.0353
##      9        1.2781            -nan     0.1000    0.0338
##     10        1.2560            -nan     0.1000    0.0269
##     20        1.1055            -nan     0.1000    0.0186
##     40        0.9337            -nan     0.1000    0.0088
##     60        0.8294            -nan     0.1000    0.0055
##     80        0.7509            -nan     0.1000    0.0047
##    100        0.6883            -nan     0.1000    0.0036
##    120        0.6342            -nan     0.1000    0.0023
##    140        0.5877            -nan     0.1000    0.0019
##    150        0.5685            -nan     0.1000    0.0017
```

```
modelRandomForest            <- train(classe ~ ., data = trainDataset, method = "rf",
                                      preProcess = preProcess, trControl=modelTrainControl)
```

```
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 2 on full training set
```

```
allModels <- list(
  modelFitBag,
  modelKNearestNeighbor,
  modelRecursivePartition,
  modelGradientBoostingMachine,
  modelRandomForest
)
names(allModels) <- sapply(allModels, function(x) x$method)
sort(sapply(allModels, function(x) x$results$Accuracy[length(x$results$Accuracy)]),decreasing = TRUE)
```

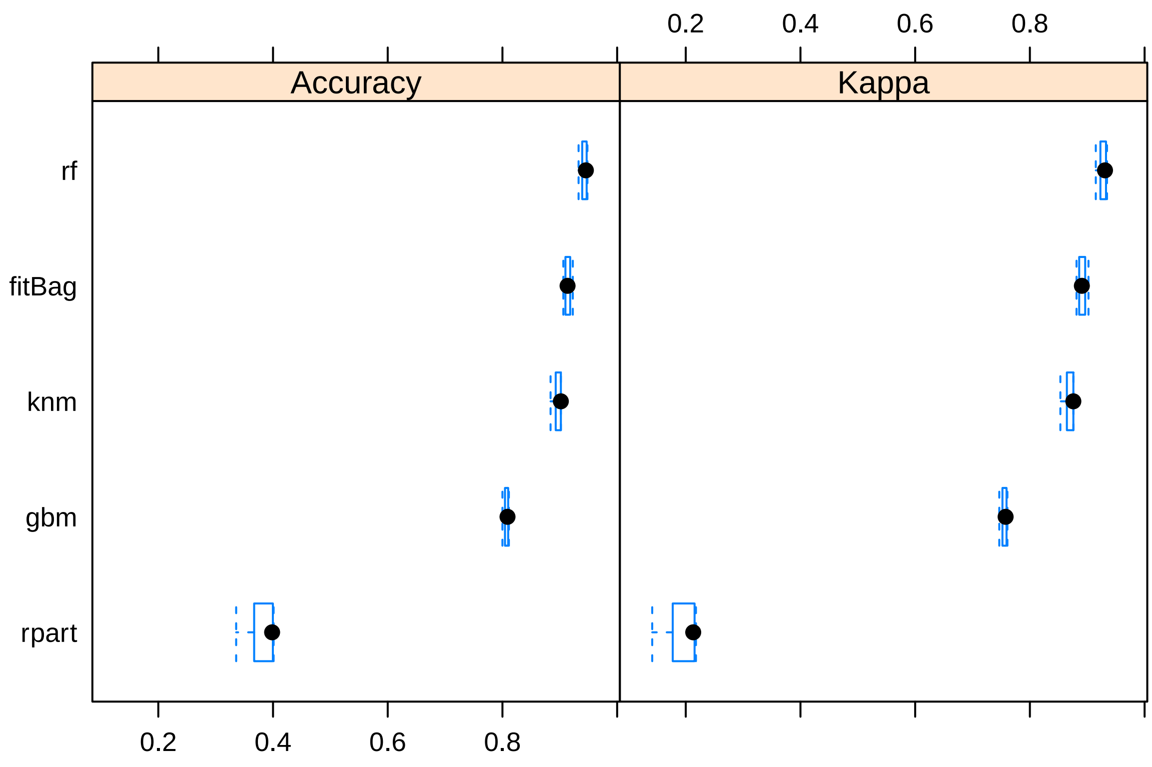
```
##        rf   treebag       knn       gbm     rpart
## 0.9290286 0.9141377 0.8611988 0.8066752 0.2843393
```

```
summaryModels <- resamples(
  list(
    fitBag=modelFitBag,
    knm=modelKNearestNeighbor,
    rpart=modelRecursivePartition,
    gbm=modelGradientBoostingMachine,
    rf=modelRandomForest
  )
)
summary(summaryModels)
```

```
##
## Call:
```

```
## summary.resamples(object = summaryModels)
##
## Models: fitBag, knm, rpart, gbm, rf
## Number of resamples: 3
##
## Accuracy
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## fitBag 0.9061491 0.9098511 0.9135530 0.9141377 0.9181320 0.9227111    0
## knm    0.8838967 0.8928655 0.9018342 0.8958778 0.9018684 0.9019025    0
## rpart  0.3357398 0.3670886 0.3984375 0.3784101 0.3997453 0.4010532    0
## gbm    0.7999830 0.8044158 0.8088485 0.8066752 0.8100213 0.8111942    0
## rf     0.9327331 0.9391077 0.9454823 0.9421355 0.9468366 0.9481909    0
##
## Kappa
##             Min.   1st Qu.    Median      Mean   3rd Qu.      Max. NA's
## fitBag 0.8812924 0.8859546 0.8906168 0.8913622 0.8963971 0.9021774    0
## knm    0.8531573 0.8644671 0.8757769 0.8682613 0.8758133 0.8758498    0
## rpart  0.1415161 0.1772751 0.2130341 0.1907712 0.2153988 0.2177635    0
## gbm    0.7466467 0.7521686 0.7576904 0.7550530 0.7592561 0.7608218    0
## rf     0.9148974 0.9229492 0.9310009 0.9267757 0.9327148 0.9344288    0
```

```r
bwplot(summaryModels)
```



Confusion Matrix for each Model

Heading

```
getAccuracy <- function(model) {
  return(model$result$Accuracy[length(model$result$Accuracy)])
}

allModels.prediction <- list()
allModels.accuracy <- list()
allModels.plot <- list()
test <- ggplot(mtcars, aes(mpg, hp)) + geom_point()

printConfusion <- function(currentConfusionMatrix, modelName) {
  confusionMatrixAsDataFrame <- data.frame(currentConfusionMatrix$table)
  confmatrix_df <- data.frame(currentConfusionMatrix$table)
  plotConfSquares <- ggplot(confmatrix_df) + geom_tile(aes(x=Prediction, y=Reference, fill=Freq))

  cat("### Model ",modelName,"\n")

  currentModel <- allModels[modelName]

  label <- paste('Confusion Matrix of model',modelName)
  cat(paste0("#### ",label,"\n\n"))
  cat(paste0(kable(currentConfusionMatrix$table,digits = 4) %>%
    kable_styling(bootstrap_options = "striped", full_width = F, position = "center"),collapse="\n"))
  cat('\n\n')

  cat('\n')
  cat("#### Overall Statistics\n\n")
  cat("|                           |                                            |\n")
  cat("|--------------------------:|:-------------------------------------------|\n")
  cat("| Accuracy:                 |",
      currentConfusionMatrix$overall[["Accuracy"]],"|\n")
  cat("| 95% CI:                   | (",
      currentConfusionMatrix$overall[["AccuracyLower"]],",",
      currentConfusionMatrix$overall[["AccuracyLower"]],")","|\n")
  cat("| No Information Rate:      |",
      currentConfusionMatrix$overall[["AccuracyPValue"]],"|\n")
  cat("| Kappa:                    |",
      currentConfusionMatrix$overall[["Kappa"]],"|\n")
  cat("| Mcnemar's Test P-Value:   |",
      currentConfusionMatrix$overall[["McnemarPValue"]],"|\n")
  cat('\n')

  label <- paste('Statistics by Class of model',modelName)
  cat('\n')
  cat(paste0("#### ",label,"\n\n"))
  cat(paste0(knitr::kable(currentConfusionMatrix$byClass,digits = 4) %>%
    kable_styling(bootstrap_options = "striped", full_width = F, position = "left", font_size = 11),col
  cat('\n\n')

  print(plotConfSquares)
```

```
  cat('\n\n')
  if(modelName=="rpart"){
    fancyRpartPlot(currentModel$rpart$finalModel)
  }
  if(modelName=="rf"){
    randomForestError <- gg_error(currentModel$rf$finalModel)
    print(plot(randomForestError))
  }
  cat('\n\n')
}


for(modelName in names(allModels)) {
  set.seed(123)
  currentModel <- allModels[modelName]
  predictedClasse <- predict(currentModel,validationDataset)
  allModels.prediction[[modelName]] <- predictedClasse
  allModels.accuracy[[modelName]] <- getAccuracy(currentModel[[modelName]])

  currentConfusionMatrix <- confusionMatrix(predictedClasse[[modelName]], as.factor(validationDataset$cl
  printConfusion(currentConfusionMatrix, modelName)
}
```

Model treebag

Confusion Matrix of model treebag

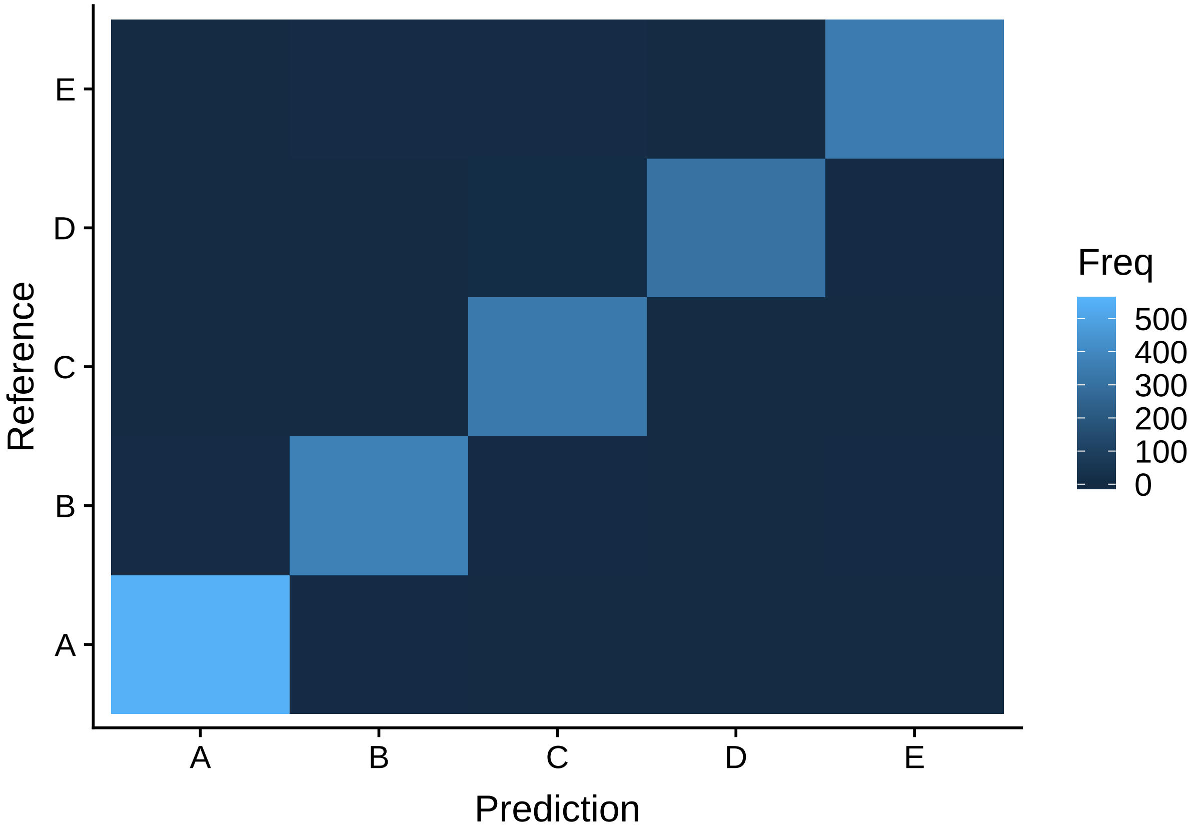
	A	B	C	D	E
A	551	7	1	1	1
B	2	367	3	0	6
C	0	2	334	11	6
D	4	1	4	307	3
E	1	2	0	2	344

Overall Statistics

Accuracy:	0.9709184
95% CI:	(0.962484 , 0.962484)
No Information Rate:	0
Kappa:	0.9632081
Mcnemar's Test P-Value:	0.05097757

Statistics by Class of model treebag

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence
Class: A	0.9875	0.9929	0.9822	0.9950	0.9822	0.9875	0.9848	0.2847
Class: B	0.9683	0.9930	0.9709	0.9924	0.9709	0.9683	0.9696	0.1934
Class: C	0.9766	0.9883	0.9462	0.9950	0.9462	0.9766	0.9612	0.1745
Class: D	0.9564	0.9927	0.9624	0.9915	0.9624	0.9564	0.9594	0.1638
Class: E	0.9556	0.9969	0.9857	0.9901	0.9857	0.9556	0.9704	0.1837



Model knn

Confusion Matrix of model knn

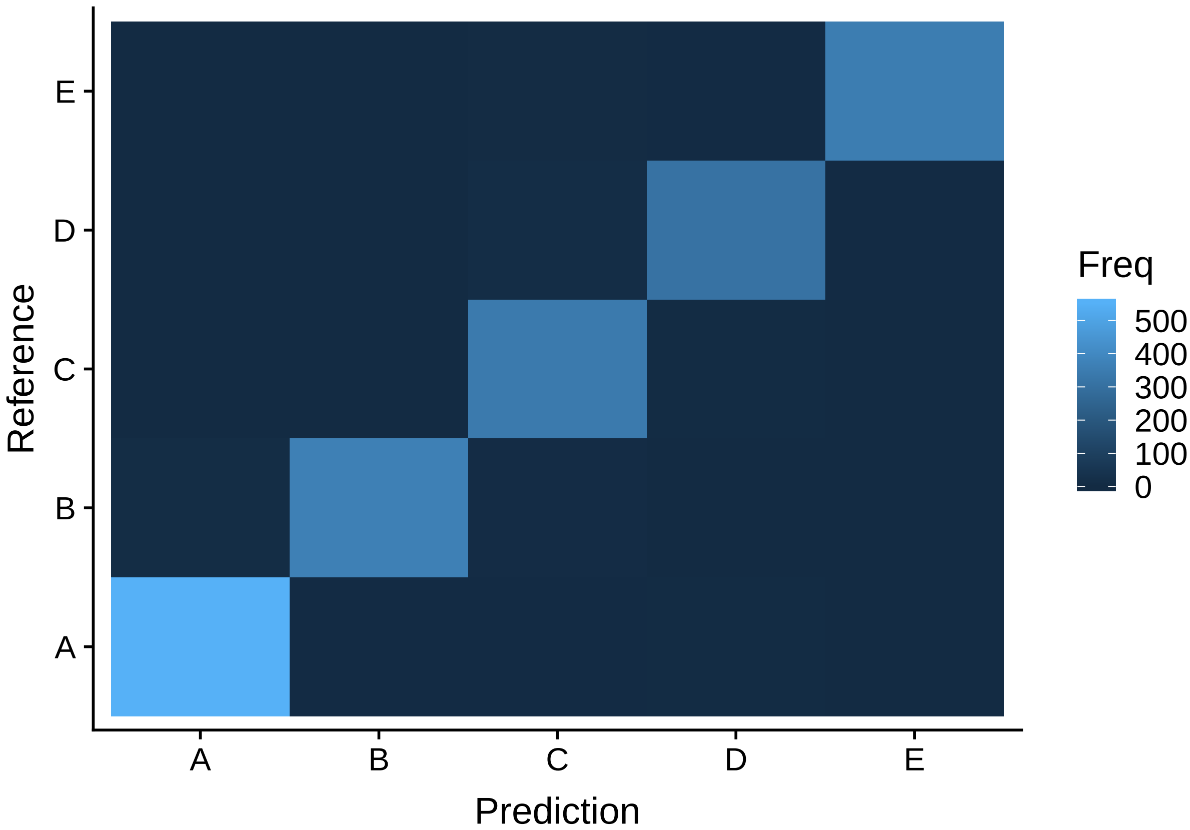
	A	B	C	D	E
A	550	8	0	1	1
B	2	363	1	0	1
C	2	7	338	11	5
D	3	0	3	307	2
E	1	1	0	2	351

Overall Statistics

Accuracy:	0.9739796
95% CI:	(0.9659286 , 0.9659286)
No Information Rate:	0
Kappa:	0.9670856
Mcnemar's Test P-Value:	NaN

Statistics by Class of model knn

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence
Class: A	0.9857	0.9929	0.9821	0.9943	0.9821	0.9857	0.9839	0.2847
Class: B	0.9578	0.9975	0.9891	0.9900	0.9891	0.9578	0.9732	0.1934
Class: C	0.9883	0.9845	0.9311	0.9975	0.9311	0.9883	0.9589	0.1745
Class: D	0.9564	0.9951	0.9746	0.9915	0.9746	0.9564	0.9654	0.1638
Class: E	0.9750	0.9975	0.9887	0.9944	0.9887	0.9750	0.9818	0.1837



Model rpart

Confusion Matrix of model rpart

	A	B	C	D	E
A	507	221	318	154	147
B	0	0	0	0	0
C	0	0	0	0	0
D	40	76	17	123	56
E	11	82	7	44	157

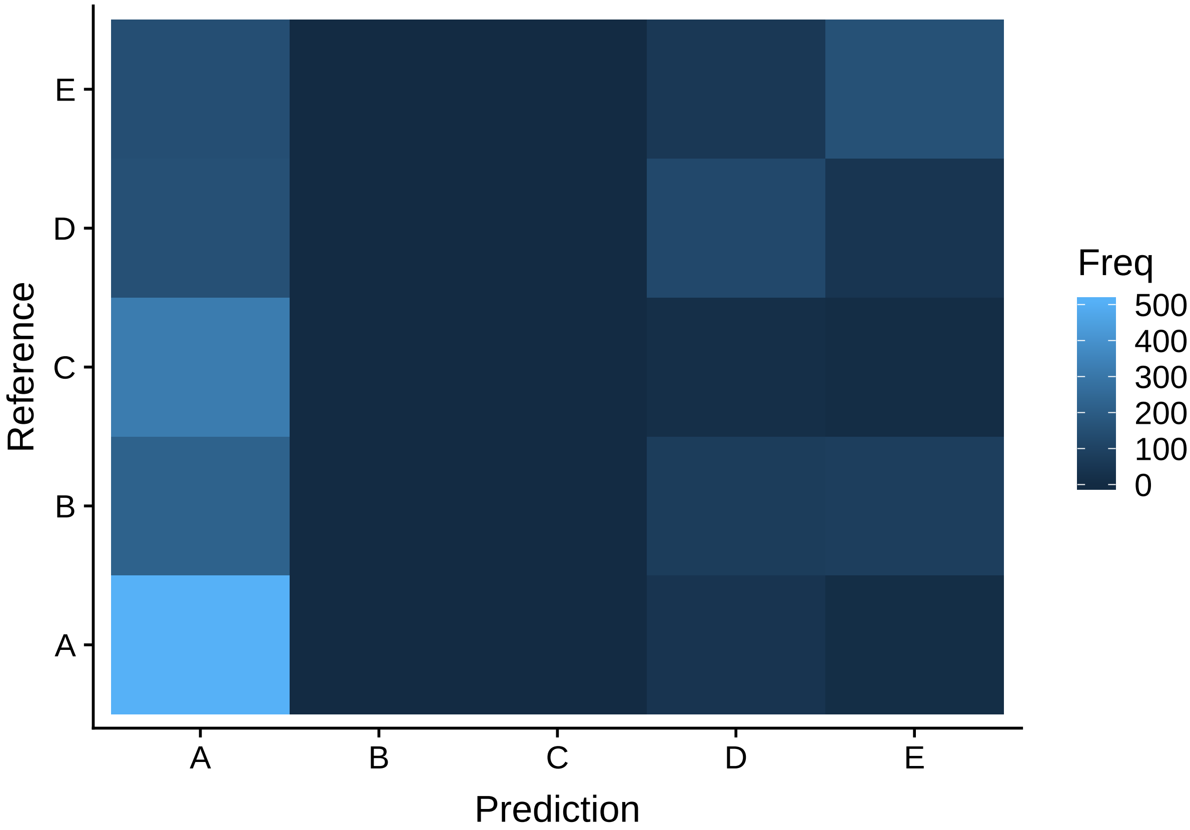
Overall Statistics

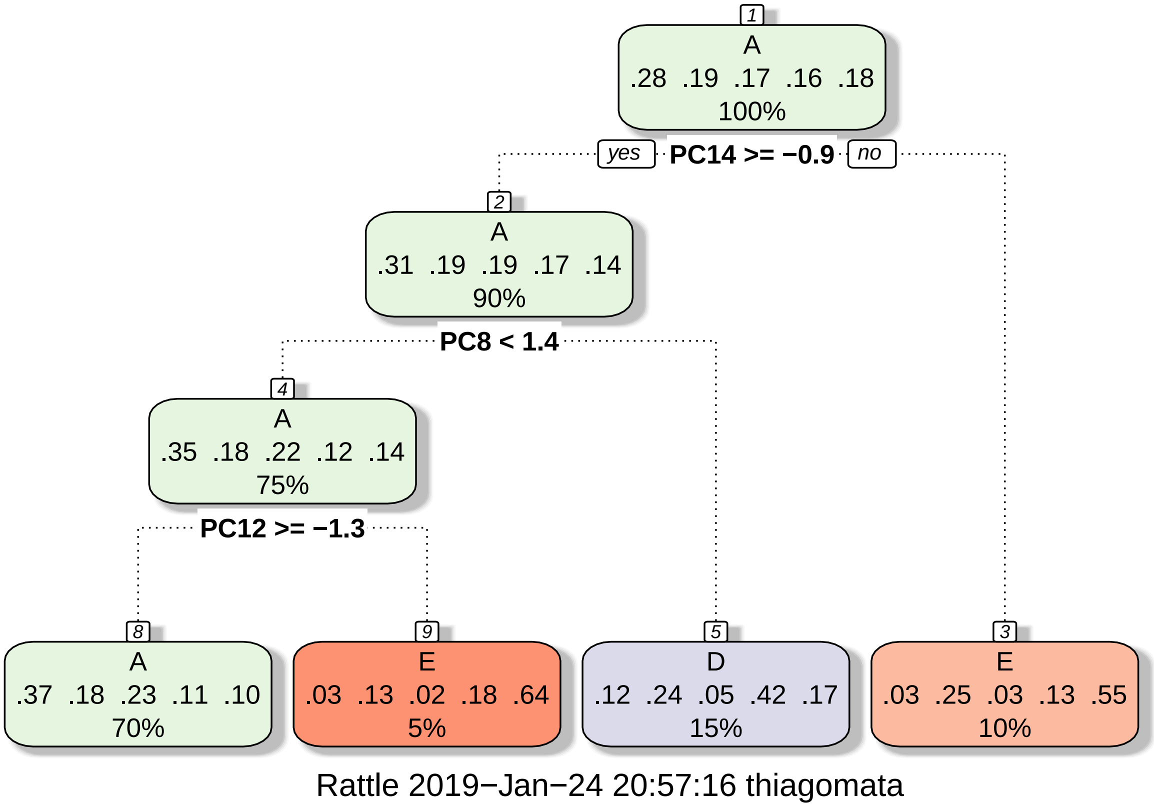
Accuracy:	0.4015306
95% CI:	(0.3797345 , 0.3797345)
No Information Rate:	0.0000000000000000000000000009335083
Kappa:	0.2021134

Mcnemar's Test P-Value: NaN

Statistics by Class of model rpart

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence
Class: A	0.9086	0.4009	0.3764	0.9168	0.3764	0.9086	0.5323	0.2847
Class: B	0.0000	1.0000	NaN	0.8066	NA	0.0000	NA	0.1934
Class: C	0.0000	1.0000	NaN	0.8255	NA	0.0000	NA	0.1745
Class: D	0.3832	0.8847	0.3942	0.8799	0.3942	0.3832	0.3886	0.1638
Class: E	0.4361	0.9100	0.5216	0.8776	0.5216	0.4361	0.4750	0.1837





Model gbm

Confusion Matrix of model gbm

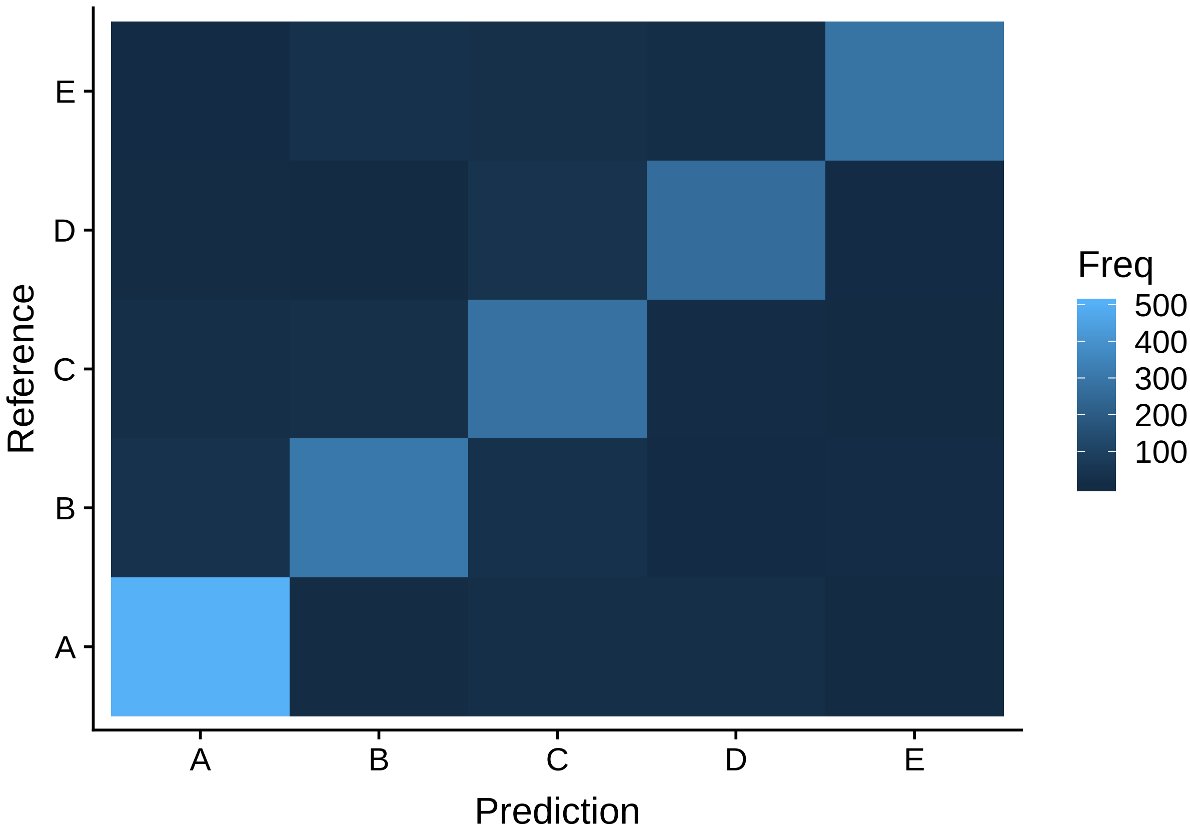
	A	B	C	D	E
A	502	32	21	11	6
B	11	302	26	4	28
C	20	29	279	36	25
D	20	6	9	264	17
E	5	10	7	6	284

Overall Statistics

Accuracy:	0.8321429
95% CI:	(0.8148481 , 0.8148481)
No Information Rate:	0
Kappa:	0.7875336
Mcnemar's Test P-Value:	0.0000000561395

Statistics by Class of model gbm

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence
Class: A	0.8996	0.9501	0.8776	0.9597	0.8776	0.8996	0.8885	0.2847
Class: B	0.7968	0.9564	0.8140	0.9515	0.8140	0.7968	0.8053	0.1934
Class: C	0.8158	0.9320	0.7172	0.9599	0.7172	0.8158	0.7633	0.1745
Class: D	0.8224	0.9683	0.8354	0.9653	0.8354	0.8224	0.8289	0.1638
Class: E	0.7889	0.9825	0.9103	0.9539	0.9103	0.7889	0.8452	0.1837



Model rf

Confusion Matrix of model rf

	A	B	C	D	E
A	557	7	0	0	0
B	0	370	1	0	1
C	0	2	340	11	2
D	1	0	1	309	1
E	0	0	0	1	356

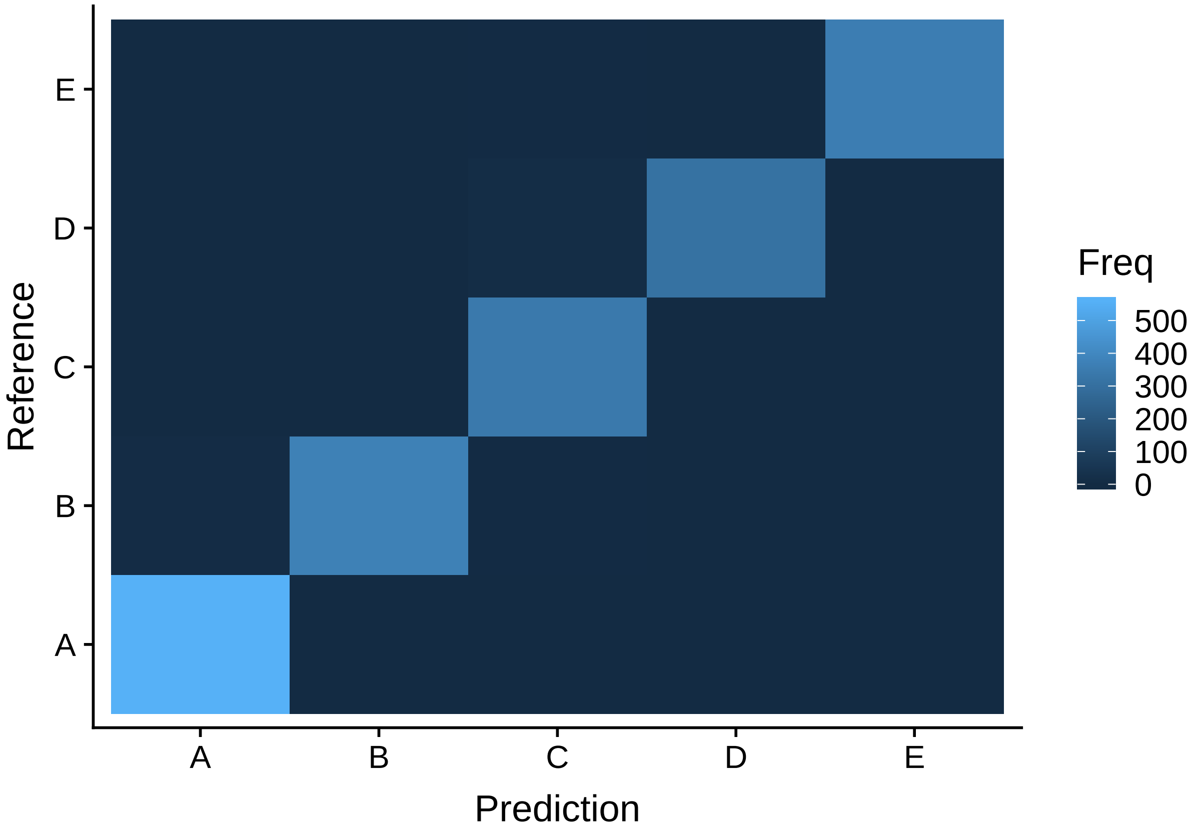
Overall Statistics

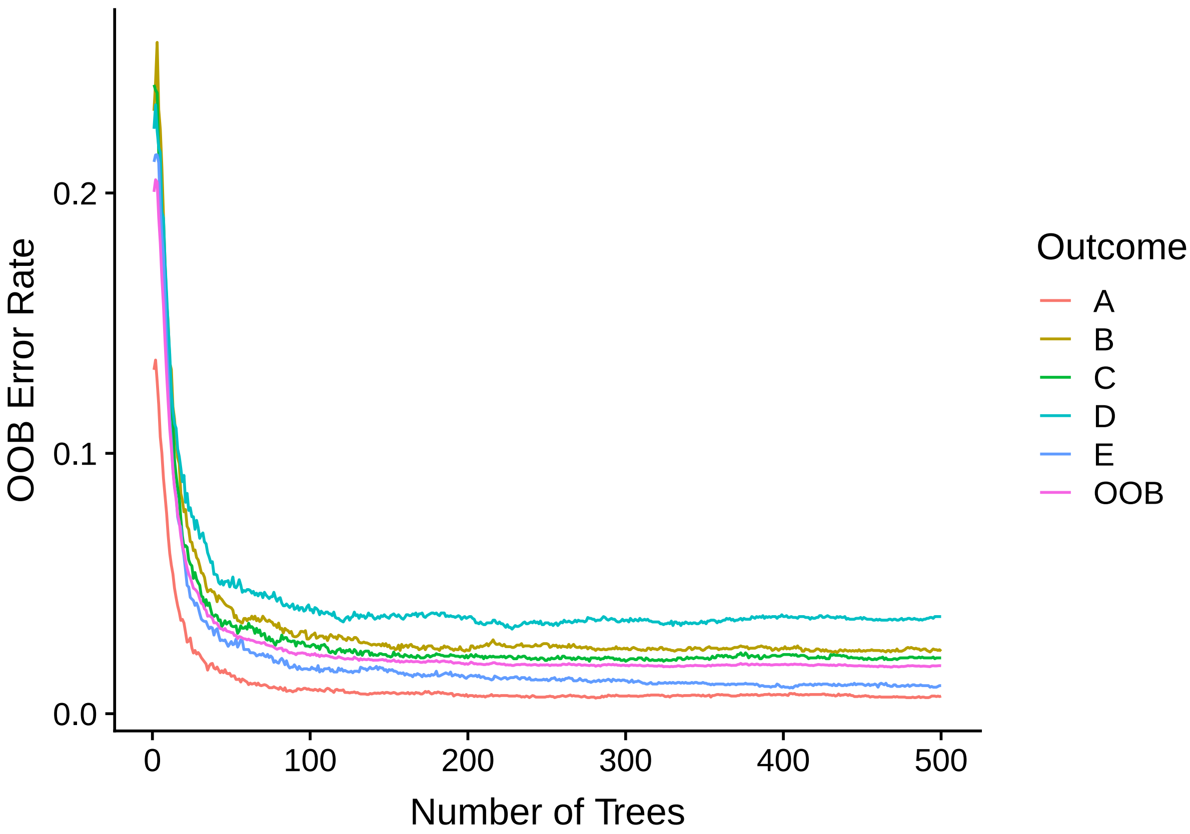
Accuracy:	0.9857143
95% CI:	(0.9794189 , 0.9794189)
No Information Rate:	0
Kappa:	0.9819225

Mcnemar's Test P-Value:	NaN

Statistics by Class of model rf

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence
Class: A	0.9982	0.9950	0.9876	0.9993	0.9876	0.9982	0.9929	0.2847
Class: B	0.9763	0.9987	0.9946	0.9943	0.9946	0.9763	0.9854	0.1934
Class: C	0.9942	0.9907	0.9577	0.9988	0.9577	0.9942	0.9756	0.1745
Class: D	0.9626	0.9982	0.9904	0.9927	0.9904	0.9626	0.9763	0.1638
Class: E	0.9889	0.9994	0.9972	0.9975	0.9972	0.9889	0.9930	0.1837





Voting Mechanism

Now, let's combine all the models that have a accuracy bigger or equal than the minimal 80% and make them vote using the accuracy of each model as weight.

```
getVotingScore <- function(predictions,value) {
    return(
      ifelse(predictions$predFitBag                      == value, predictions$accuracyFitBag, 0) +
      ifelse(predictions$predictKNearesNeighbor          == value, predictions$accuracyKNearesNeighbor, 0
      ifelse(predictions$predictRecursivePartition       == value, predictions$accuracyRecursivePartitio
      ifelse(predictions$predictGradientBoostingMachine == value, predictions$accuracyGradientBoostingMa
      ifelse(predictions$predictRandomForest             == value, predictions$accuracyRandomForest, 0) +
      0
    )
}

voting <- function(data) {

  # Any model with worse accuracy than this, should not be considered on the voting ( score = 0 )
  MIN_ACCURACY = 0.8

  # FIT BAG
  predFitBag                    <- predict(modelFitBag, data)
  accuracyFitBag                <- ifelse(getAccuracy(modelFitBag) > MIN_ACCURACY,getAccuracy(modelGra
  # GRADIENT BOOSTING MACHINE
```

```
predictGradientBoostingMachine  <- predict(modelGradientBoostingMachine, data)
accuracyGradientBoostingMachine <- ifelse(getAccuracy(modelGradientBoostingMachine) > MIN_ACCURACY,ge
# K NEARES NEIGHBOR
predictKNearesNeighbor          <- predict(modelKNearestNeighbor, data)
accuracyKNearesNeighbor         <- ifelse(getAccuracy(modelKNearestNeighbor) > MIN_ACCURACY,getAccura
# RECURSIVE PARTITION
predictRecursivePartition       <- predict(modelRecursivePartition, data)
accuracyRecursivePartition      <- ifelse(getAccuracy(modelRecursivePartition) > MIN_ACCURACY,getAccu
# RANDOM FOREST
predictRandomForest             <- predict(modelRandomForest, data)
accuracyRandomForest            <- ifelse(getAccuracy(modelRandomForest) > MIN_ACCURACY,getAccuracy(m

sumAccuracy <-
  accuracyFitBag +
  accuracyGradientBoostingMachine +
  accuracyKNearesNeighbor +
  accuracyRecursivePartition +
  accuracyRandomForest +
  0

predictions <- data.frame(
  predFitBag,
  accuracyFitBag,
  predictKNearesNeighbor,
  accuracyKNearesNeighbor,
  predictRecursivePartition,
  accuracyRecursivePartition,
  predictGradientBoostingMachine,
  accuracyGradientBoostingMachine,
  predictRandomForest,
  accuracyRandomForest
)
scoreA <-getVotingScore(predictions,'A') / sumAccuracy
scoreB <-getVotingScore(predictions,'B') / sumAccuracy
scoreC <-getVotingScore(predictions,'C') / sumAccuracy
scoreD <-getVotingScore(predictions,'D') / sumAccuracy
scoreE <-getVotingScore(predictions,'E') / sumAccuracy

votingData <- data.frame(
  scoreA,
  scoreB,
  scoreC,
  scoreD,
  scoreE
)

votingData[ is.na(votingData$scoreA), "scoreA" ]  <- c(0)
votingData[ is.na(votingData$scoreB), "scoreB" ]  <- c(0)
votingData[ is.na(votingData$scoreC), "scoreC" ]  <- c(0)
votingData[ is.na(votingData$scoreD), "scoreD" ]  <- c(0)
votingData[ is.na(votingData$scoreE), "scoreE" ]  <- c(0)

votingData$maxScore <- apply(votingData,1,max)
```

```r
  votingData$voted <- c()
  votingData[votingData$scoreA == votingData$maxScore, "voted"] <- 'A'
  votingData[votingData$scoreB == votingData$maxScore, "voted"] <- 'B'
  votingData[votingData$scoreC == votingData$maxScore, "voted"] <- 'C'
  votingData[votingData$scoreD == votingData$maxScore, "voted"] <- 'D'
  votingData[votingData$scoreE == votingData$maxScore, "voted"] <- 'E'
  votingData$classe <- data$classe
  return(votingData)
}
votedValidation <- voting(validationDataset)
votingConfusionMatrix <- confusionMatrix(as.factor(votedValidation$voted),as.factor(validationDataset$cl
allModels.accuracy["voting"] <- votingConfusionMatrix$overall['Accuracy']
printConfusion(votingConfusionMatrix, "voting")
```

Model voting

Confusion Matrix of model voting

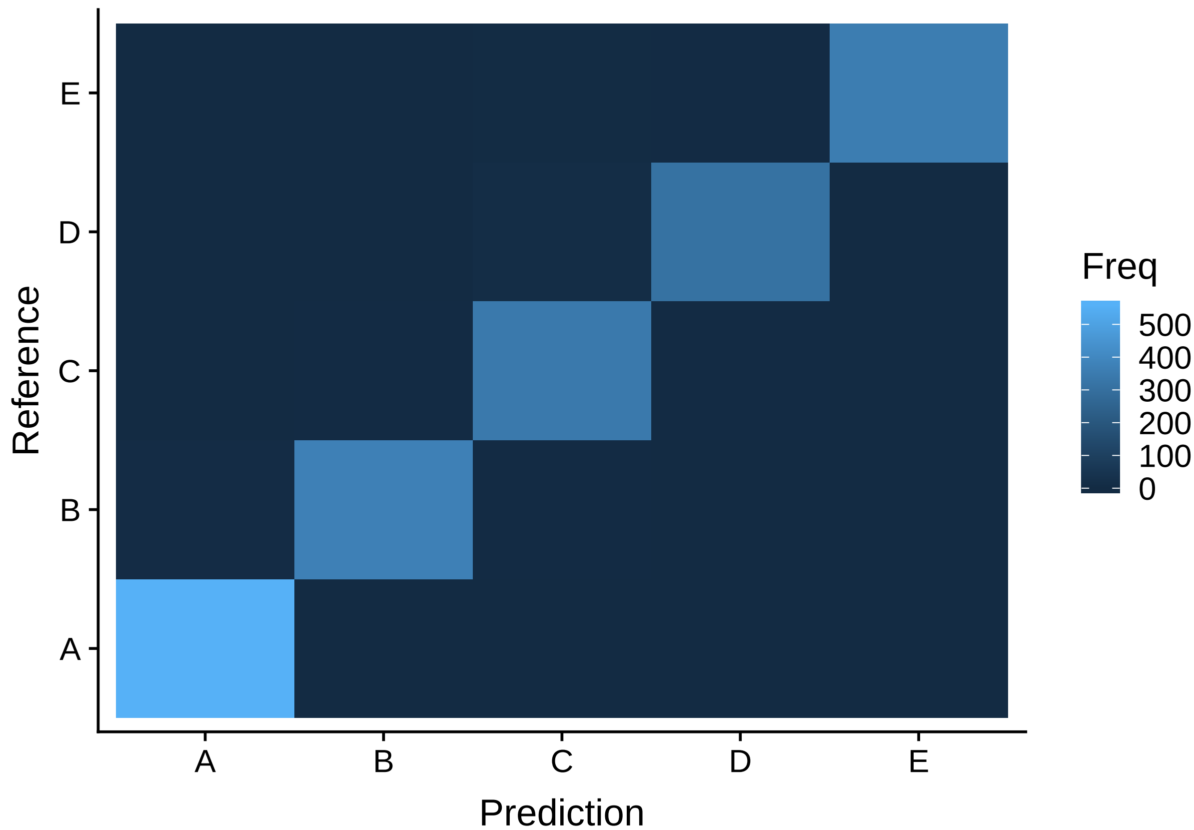
	A	B	C	D	E
A	557	7	0	0	0
B	0	369	2	0	1
C	0	2	338	11	3
D	1	0	2	309	2
E	0	1	0	1	354

Overall Statistics

Accuracy:	0.9831633
95% CI:	(0.9764356 , 0.9764356)
No Information Rate:	0
Kappa:	0.9786948
Mcnemar's Test P-Value:	NaN

Statistics by Class of model voting

	Sensitivity	Specificity	Pos Pred Value	Neg Pred Value	Precision	Recall	F1	Prevalence
Class: A	0.9982	0.9950	0.9876	0.9993	0.9876	0.9982	0.9929	0.2847
Class: B	0.9736	0.9981	0.9919	0.9937	0.9919	0.9736	0.9827	0.1934
Class: C	0.9883	0.9901	0.9548	0.9975	0.9548	0.9883	0.9713	0.1745
Class: D	0.9626	0.9969	0.9841	0.9927	0.9841	0.9626	0.9732	0.1638
Class: E	0.9833	0.9988	0.9944	0.9963	0.9944	0.9833	0.9888	0.1837



Choosing the final Model

The voting process show a good result and because it is combining different approachs it is more hard to have the same type of overfitting.

Show some samples of prediction on Test Data

```
predictSamples <- voting(testData)
knitr::kable(predictSamples,caption='predicting classe of the test data based on the model',align = "c")
  kable_styling(bootstrap_options = c("striped", "hover", "condensed"))
```

```
cat(predictSamples$voted)
```

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

Conclusion

The final model used is a voting from the best models created from the data. The accuracy of the voting model on the test data was very good but not as good as on the validation dataset, as expected.

Table 7: predicting classe of the test data based on the model

scoreA	scoreB	scoreC	scoreD	scoreE	maxScore	voted
0.000000	0.7629920	0.2370080	0	0	0.762992	B
1.000000	0.0000000	0.0000000	0	0	1.000000	A
0.474016	0.2729565	0.2530275	0	0	0.474016	A
1.000000	0.0000000	0.0000000	0	0	1.000000	A
1.000000	0.0000000	0.0000000	0	0	1.000000	A
0.000000	0.0000000	0.0000000	0	1	1.000000	E
0.000000	0.0000000	0.0000000	1	0	1.000000	D
0.000000	1.0000000	0.0000000	0	0	1.000000	B
1.000000	0.0000000	0.0000000	0	0	1.000000	A
1.000000	0.0000000	0.0000000	0	0	1.000000	A
0.474016	0.5259840	0.0000000	0	0	0.525984	B
0.000000	0.0000000	1.0000000	0	0	1.000000	C
0.000000	1.0000000	0.0000000	0	0	1.000000	B
1.000000	0.0000000	0.0000000	0	0	1.000000	A
0.000000	0.0000000	0.0000000	0	1	1.000000	E
0.000000	0.0000000	0.0000000	0	1	1.000000	E
1.000000	0.0000000	0.0000000	0	0	1.000000	A
0.000000	1.0000000	0.0000000	0	0	1.000000	B
0.000000	1.0000000	0.0000000	0	0	1.000000	B
0.000000	1.0000000	0.0000000	0	0	1.000000	B

Biografy

[1] Velloso, E.; Bulling, A.; Gellersen, H.; Ugulino, W.; Fuks, H. Qualitative Activity Recognition of Weight Lifting Exercises. Proceedings of 4th International Conference in Cooperation with SIGCHI (Augmented Human '13) . Stuttgart, Germany: ACM SIGCHI, 2013.