2. Perform the below given activities:

a. Create classification model using different random forest models

b. Verify model goodness of fit

c. Apply all the model validation techniques

d. Make conclusions

e. Plot importance of variables

library(randomForest)

library(mlbench)

library(caret)

# Load Dataset

data(Sonar)

dataset <- Sonar

x <- dataset[,1:60]

y <- dataset[,61]

|  |  |
| --- | --- |
|  | # Create model with default paramters  control <- trainControl(method="repeatedcv", number=10, repeats=3)  seed <- 7  metric <- "Accuracy"  set.seed(seed)  mtry <- sqrt(ncol(x))  tunegrid <- expand.grid(.mtry=mtry)  rf\_default <- train(Class~., data=dataset, method="rf", metric=metric, tuneGrid=tunegrid, trControl=control)  print(rf\_default) |

  Accuracy   Kappa      Accuracy SD  Kappa SD

  0.8138384  0.6209924  0.0747572    0.1569159

control <- trainControl(method="repeatedcv", number=10, repeats=3, search="random")

set.seed(seed)

mtry <- sqrt(ncol(x))

rf\_random <- train(Class~., data=dataset, method="rf", metric=metric, tuneLength=15, trControl=control)

print(rf\_random)

plot(rf\_random)

Resampling results across tuning parameters:

  mtry  Accuracy   Kappa      Accuracy SD  Kappa SD

  11    0.8218470  0.6365181  0.09124610   0.1906693

  14    0.8140620  0.6215867  0.08475785   0.1750848

  17    0.8030231  0.5990734  0.09595988   0.1986971

  24    0.8042929  0.6002362  0.09847815   0.2053314

  30    0.7933333  0.5798250  0.09110171   0.1879681

  34    0.8015873  0.5970248  0.07931664   0.1621170

  45    0.7932612  0.5796828  0.09195386   0.1887363

  47    0.7903896  0.5738230  0.10325010   0.2123314

  49    0.7867532  0.5673879  0.09256912   0.1899197

  50    0.7775397  0.5483207  0.10118502   0.2063198

  60    0.7790476  0.5513705  0.09810647   0.2005012

control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")

set.seed(seed)

tunegrid <- expand.grid(.mtry=c(1:15))

rf\_gridsearch <- train(Class~., data=dataset, method="rf", metric=metric, tuneGrid=tunegrid, trControl=control)

print(rf\_gridsearch)

plot(rf\_gridsearch)

Resampling results across tuning parameters:

  mtry  Accuracy   Kappa      Accuracy SD  Kappa SD

   1    0.8377273  0.6688712  0.07154794   0.1507990

   2    0.8378932  0.6693593  0.07185686   0.1513988

   3    0.8314502  0.6564856  0.08191277   0.1700197

   4    0.8249567  0.6435956  0.07653933   0.1590840

   5    0.8268470  0.6472114  0.06787878   0.1418983

   6    0.8298701  0.6537667  0.07968069   0.1654484

   7    0.8282035  0.6493708  0.07492042   0.1584772

   8    0.8232828  0.6396484  0.07468091   0.1571185

   9    0.8268398  0.6476575  0.07355522   0.1529670

  10    0.8204906  0.6346991  0.08499469   0.1756645

  11    0.8073304  0.6071477  0.09882638   0.2055589

  12    0.8184488  0.6299098  0.09038264   0.1884499

  13    0.8093795  0.6119327  0.08788302   0.1821910

  14    0.8186797  0.6304113  0.08178957   0.1715189

  15    0.8168615  0.6265481  0.10074984   0.2091663

# Algorithm Tune (tuneRF)

set.seed(seed)

bestmtry <- tuneRF(x, y, stepFactor=1.5, improve=1e-5, ntree=500)

print(bestmtry)

mtry  OOBError

5.OOB     5 0.1538462

7.OOB     7 0.1538462

10.OOB   10 0.1442308

15.OOB   15 0.1682692

control <- trainControl(method="repeatedcv", number=10, repeats=3, search="grid")

tunegrid <- expand.grid(.mtry=c(sqrt(ncol(x))))

modellist <- list()

for (ntree in c(1000, 1500, 2000, 2500)) {

set.seed(seed)

fit <- train(Class~., data=dataset, method="rf", metric=metric, tuneGrid=tunegrid, trControl=control, ntree=ntree)

key <- toString(ntree)

modellist[[key]] <- fit

}

# compare results

results <- resamples(modellist)

summary(results)

dotplot(results)

Models: 1000, 1500, 2000, 2500

Number of resamples: 30

Accuracy

      Min. 1st Qu. Median   Mean 3rd Qu.   Max. NA's

1000 0.600  0.8024 0.8500 0.8186  0.8571 0.9048    0

1500 0.600  0.8024 0.8095 0.8169  0.8571 0.9500    0

2000 0.619  0.8024 0.8095 0.8202  0.8620 0.9048    0

2500 0.619  0.8000 0.8095 0.8201  0.8893 0.9091    0

|  |  |
| --- | --- |
|  | customRF <- list(type = "Classification", library = "randomForest", loop = NULL)  customRF$parameters <- data.frame(parameter = c("mtry", "ntree"), class = rep("numeric", 2), label = c("mtry", "ntree"))  customRF$grid <- function(x, y, len = NULL, search = "grid") {}  customRF$fit <- function(x, y, wts, param, lev, last, weights, classProbs, ...) {    randomForest(x, y, mtry = param$mtry, ntree=param$ntree, ...)  }  customRF$predict <- function(modelFit, newdata, preProc = NULL, submodels = NULL)     predict(modelFit, newdata)  customRF$prob <- function(modelFit, newdata, preProc = NULL, submodels = NULL)     predict(modelFit, newdata, type = "prob")  customRF$sort <- function(x) x[order(x[,1]),]  customRF$levels <- function(x) x$classes |

control <- trainControl(method="repeatedcv", number=10, repeats=3)

tunegrid <- expand.grid(.mtry=c(1:15), .ntree=c(1000, 1500, 2000, 2500))

set.seed(seed)

custom <- train(Class~., data=dataset, method=customRF, metric=metric, tuneGrid=tunegrid, trControl=control)

summary(custom)

plot(custom)

mtry  ntree  Accuracy   Kappa      Accuracy SD  Kappa SD

   1    1000   0.8442424  0.6828299  0.06505226   0.1352640

   1    1500   0.8394805  0.6730868  0.05797828   0.1215990

   1    2000   0.8314646  0.6564643  0.06630279   0.1381197

   1    2500   0.8379654  0.6693773  0.06576468   0.1375408

   2    1000   0.8313781  0.6562819  0.06909608   0.1436961

   2    1500   0.8427345  0.6793793  0.07005975   0.1451269

   2    2000   0.8443218  0.6830115  0.06754346   0.1403497

   2    2500   0.8428066  0.6791639  0.06488132   0.1361329

   3    1000   0.8350216  0.6637523  0.06530816   0.1362839

   3    1500   0.8347908  0.6633405  0.06836512   0.1418106

   3    2000   0.8428066  0.6800703  0.06643838   0.1382763

   3    2500   0.8365296  0.6668480  0.06401429   0.1336583

   4    1000   0.8316955  0.6574476  0.06292132   0.1317857

   4    1500   0.8331241  0.6605244  0.07543919   0.1563171

   4    2000   0.8378860  0.6699428  0.07147459   0.1488322

   4    2500   0.8315368  0.6568128  0.06981259   0.1450390

   5    1000   0.8284343  0.6505097  0.07278539   0.1516109

   5    1500   0.8283622  0.6506604  0.07166975   0.1488037

   5    2000   0.8219336  0.6375155  0.07548501   0.1564718

   5    2500   0.8315440  0.6570792  0.07067743   0.1472716

   6    1000   0.8203391  0.6341073  0.08076304   0.1689558

   6    1500   0.8186797  0.6302188  0.07559694   0.1588256

   6    2000   0.8187590  0.6310555  0.07081621   0.1468780

   6    2500   0.8153463  0.6230495  0.07728249   0.1623253

   7    1000   0.8217027  0.6367189  0.07649651   0.1606837

   7    1500   0.8282828  0.6503808  0.06628953   0.1381925

   7    2000   0.8108081  0.6147563  0.07605609   0.1573067

   7    2500   0.8250361  0.6437397  0.07737756   0.1602434

   8    1000   0.8187590  0.6314307  0.08378631   0.1722251

   8    1500   0.8201876  0.6335679  0.07380001   0.1551340

   8    2000   0.8266883  0.6472907  0.06965118   0.1450607

   8    2500   0.8251082  0.6434251  0.07745300   0.1628087

   9    1000   0.8121717  0.6177751  0.08218598   0.1709987

   9    1500   0.8184488  0.6300547  0.08077766   0.1674261

   9    2000   0.8247980  0.6429315  0.07260439   0.1513512

   9    2500   0.8186003  0.6302674  0.07356916   0.1547231

  10    1000   0.8235209  0.6407121  0.07991334   0.1656978

  10    1500   0.8125541  0.6183581  0.06851683   0.1421993

  10    2000   0.8187518  0.6308120  0.08538951   0.1782368

  10    2500   0.8169336  0.6263682  0.07847066   0.1649216

  11    1000   0.8203463  0.6341158  0.07222587   0.1497558

  11    1500   0.8153463  0.6235878  0.09131621   0.1904418

  11    2000   0.8234416  0.6402906  0.07586609   0.1576765

  11    2500   0.8154906  0.6236875  0.07485835   0.1576576

  12    1000   0.8201948  0.6336913  0.08672139   0.1806589

  12    1500   0.8139105  0.6206994  0.08638618   0.1804780

  12    2000   0.8137590  0.6204461  0.07771424   0.1629707

  12    2500   0.8201876  0.6333194  0.07799832   0.1636237

  13    1000   0.8123232  0.6173280  0.09299062   0.1936232

  13    1500   0.8108802  0.6142721  0.08416414   0.1760527

  13    2000   0.8154257  0.6236191  0.08079923   0.1693634

  13    2500   0.8106566  0.6138814  0.08074394   0.1687437

  14    1000   0.8171645  0.6270292  0.08608806   0.1799346

  14    1500   0.8139033  0.6207263  0.08522205   0.1781396

  14    2000   0.8170924  0.6276518  0.08766645   0.1822010

  14    2500   0.8137590  0.6207371  0.08353328   0.1746425

  15    1000   0.8091486  0.6110154  0.08455439   0.1745129

  15    1500   0.8109668  0.6154780  0.08928549   0.1838700

  15    2000   0.8059740  0.6047791  0.08829659   0.1837809

  15    2500   0.8122511  0.6172771  0.08863418   0.1845635

num\_of\_samples = 1000  
x <- rgamma(num\_of\_samples, shape = 10, scale = 3)  
x <- x + rnorm(length(x), mean=0, sd = .1)

p1 <- hist(x,breaks=50, include.lowest=FALSE, right=FALSE)

a <- chisq.test(p1$counts, p=null.probs, rescale.p=TRUE, simulate.p.value=TRUE)

library('zoo')  
breaks\_cdf <- pgamma(p1$breaks, shape=10, scale=3)  
null.probs <- rollapply(breaks\_cdf, 2, function(x) x[2]-x[1])

num\_of\_samples = 100000  
y <- rgamma(num\_of\_samples, shape = 10, scale = 3)  
res <- CramerVonMisesTwoSamples(x,y)  
p-value = 1/6\*exp(-res)

num\_of\_samples = 100000  
y <- rgamma(num\_of\_samples, shape = 10, scale = 3)  
result = ks.test(x, y)

|  |  |  |  |
| --- | --- | --- | --- |
| REFERENCE DISTRIBUTION | CHI SQUARE TEST | KOLMOGOROV–SMIRNOV TEST | CRAMÉR–VON MISES CRITERION |
| Gamma(11,3) | 5e-4 | 2e-10 | 0.019 |
| N(30, 90) | 4e-5 | 2.2e-16 | 3e-3 |
| Gamme(10, 3) | .2 | .22 | .45 |

set.seed(101) # Set Seed so that same sample can be reproduced in future also

# Now Selecting 50% of data as sample from total 'n' rows of the data

sample <- sample.int(n = nrow(data), size = floor(.50\*nrow(data)), replace = F)

train <- data[sample, ]

test  <- data[-sample, ]

score = list()

LOOCV\_function = function(x,label){

for(i in 1:nrow(x)){

training = x[-i,]

model = #... train model on training

validation = x[i,]

pred = predict(model, validation[,setdiff(names(validation),label)])

score[[i]] = rmse(pred, validation[[label]]) # score/error of ith fold

}

return(unlist(score)) # returns a vector

}

library(caret)

data(iris)

# Define train control for k fold cross validation

train\_control <- trainControl(method="cv", number=10)

# Fit Naive Bayes Model

model <- train(Species~., data=iris, trControl=train\_control, method="nb")

# Summarise Results

print(model)

library(caret)

# Folds are created on the basis of target variable

folds <- createFolds(factor(data$target), k = 10, list = FALSE)

library(fpp)

library(forecast)

e <- tsCV(ts, Arima(x, order=c(2,0,0), h=1) #CV for arima model

sqrt(mean(e^2, na.rm=**TRUE**)) # RMSE

fold 1: training [1], test [2]

fold 2: training [1 2], test [3]

fold 3: training [1 2 3], test [4]

fold 4: training [1 2 3 4], test [5]

fold 5: training [1 2 3 4 5], test [6]

.

.

.

fold n: training [1 2 3 ….. n-1], test [n]

probs = clf.predict\_proba(x1)[:,1]

new\_df = pd.DataFrame({'id':train.id, 'probs':probs})

new\_df = new\_df.sort\_values(by = 'probs', ascending=False) # 30% validation set

val\_set\_ids = new\_df.iloc[1:np.int(new\_df.shape[0]\*0.3),1]

y = df['is\_train']; df.drop('is\_train', axis = 1, inplace = True)

# Xgboost parameters

xgb\_params = {'learning\_rate': 0.05,

'max\_depth': 4,

'subsample': 0.9,

'colsample\_bytree': 0.9,

'objective': 'binary:logistic',

'silent': 1,

'n\_estimators':100,

'gamma':1,

'min\_child\_weight':4}

clf = xgb.XGBClassifier(\*\*xgb\_params, seed = 10)

df = pd.concat([train, test], axis = 0)

train['is\_train'] = 1

test['is\_train'] = 0

train.drop(['target'], axis = 1, inplace = True)

**library**(tidyverse) **library**(caret)

# Load the data

data("swiss")

# Inspect the data

sample\_n(swiss, 3)

# Split the data into training and test set

set.seed(123)

training.samples <- swiss$Fertility %>% createDataPartition(p = 0.8, list = FALSE)

train.data <- swiss[training.samples, ]

test.data <- swiss[-training.samples, ]

# Build the model

model <- lm(Fertility ~., data = train.data)

# Make predictions and compute the R2, RMSE and MAE

predictions <- model %>% predict(test.data) data.frame( R2 = R2(predictions, test.data$Fertility), RMSE = RMSE(predictions, test.data$Fertility), MAE = MAE(predictions, test.data$Fertility))

RMSE(predictions, test.data$Fertility)/mean(test.data$Fertility)

# Define training control

train.control <- trainControl(method = "LOOCV")

# Train the model

model <- train(Fertility ~., data = swiss, method = "lm", trControl = train.control)

# Summarize the results

print(model)

# Define training control

set.seed(123)

train.control <- trainControl(method = "cv", number = 10)

# Train the model

model <- train(Fertility ~., data = swiss, method = "lm", trControl = train.control)

# Summarize the results

print(model)

# Define training control

set.seed(123)

train.control <- trainControl(method = "repeatedcv", number = 10, repeats = 3)

# Train the model

model <- train(Fertility ~., data = swiss, method = "lm", trControl = train.control)

# Summarize the results

print(model)

*## Initialize the environment*

**library**(caret)

## Loading required package: lattice

## Loading required package: ggplot2

**if** (!file.exists("data")) {

dir.create("data")

}

*## Download the Data*

trainURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-training.csv"

testURL <- "https://d396qusza40orc.cloudfront.net/predmachlearn/pml-testing.csv"

download.file(trainURL, "./data/train.csv", method = "curl")

download.file(testURL, "./data/test.csv", method = "curl")

downloadDate <- date()

*## Read the file into R*

trainData <- read.csv("./data/train.csv", header = TRUE)

testData <- read.csv("./data/test.csv", header = TRUE)

*# Select 8 random predictors*

set.seed(6969)

numPredictors <- ncol(trainData) - 1

s <- sample(1:numPredictors, 8)

Sample <- trainData[, s]

str(Sample)

## 'data.frame': 19622 obs. of 8 variables:

## $ min\_pitch\_dumbbell : num NA NA NA NA NA NA NA NA NA NA ...

## $ avg\_roll\_arm : num NA NA NA NA NA NA NA NA NA NA ...

## $ accel\_dumbbell\_x : int -234 -233 -232 -232 -233 -234 -232 -234 -232 -235 ...

## $ var\_roll\_belt : num NA NA NA NA NA NA NA NA NA NA ...

## $ magnet\_belt\_x : int -3 -7 -2 -6 -6 0 -4 -2 1 -3 ...

## $ stddev\_roll\_belt : num NA NA NA NA NA NA NA NA NA NA ...

## $ raw\_timestamp\_part\_2: int 788290 808298 820366 120339 196328 304277 368296 440390 484323 484434 ...

## $ accel\_dumbbell\_z : int -271 -269 -270 -269 -270 -269 -270 -272 -269 -270 ...

summary(Sample)

## min\_pitch\_dumbbell avg\_roll\_arm accel\_dumbbell\_x var\_roll\_belt

## Min. :-147 Min. :-167 Min. :-419.0 Min. : 0

## 1st Qu.: -92 1st Qu.: -38 1st Qu.: -50.0 1st Qu.: 0

## Median : -66 Median : 0 Median : -8.0 Median : 0

## Mean : -33 Mean : 13 Mean : -28.6 Mean : 8

## 3rd Qu.: 21 3rd Qu.: 76 3rd Qu.: 11.0 3rd Qu.: 0

## Max. : 121 Max. : 163 Max. : 235.0 Max. :201

## NA's :19216 NA's :19216 NA's :19216

## magnet\_belt\_x stddev\_roll\_belt raw\_timestamp\_part\_2 accel\_dumbbell\_z

## Min. :-52.0 Min. : 0 Min. : 294 Min. :-334.0

## 1st Qu.: 9.0 1st Qu.: 0 1st Qu.:252912 1st Qu.:-142.0

## Median : 35.0 Median : 0 Median :496380 Median : -1.0

## Mean : 55.6 Mean : 1 Mean :500656 Mean : -38.3

## 3rd Qu.: 59.0 3rd Qu.: 1 3rd Qu.:751891 3rd Qu.: 38.0

## Max. :485.0 Max. :14 Max. :998801 Max. : 318.0

## NA's :19216

*## Plot the spread of missing values in the Sample*

missingVar <- sapply(Sample, **function**(x) sum(is.na(x)))

dfmissing <- data.frame(variable = names(missingVar), missing = missingVar,

stringsAsFactors = FALSE)

dfmissing$variable <- factor(dfmissing$variable, levels = dfmissing$variable,

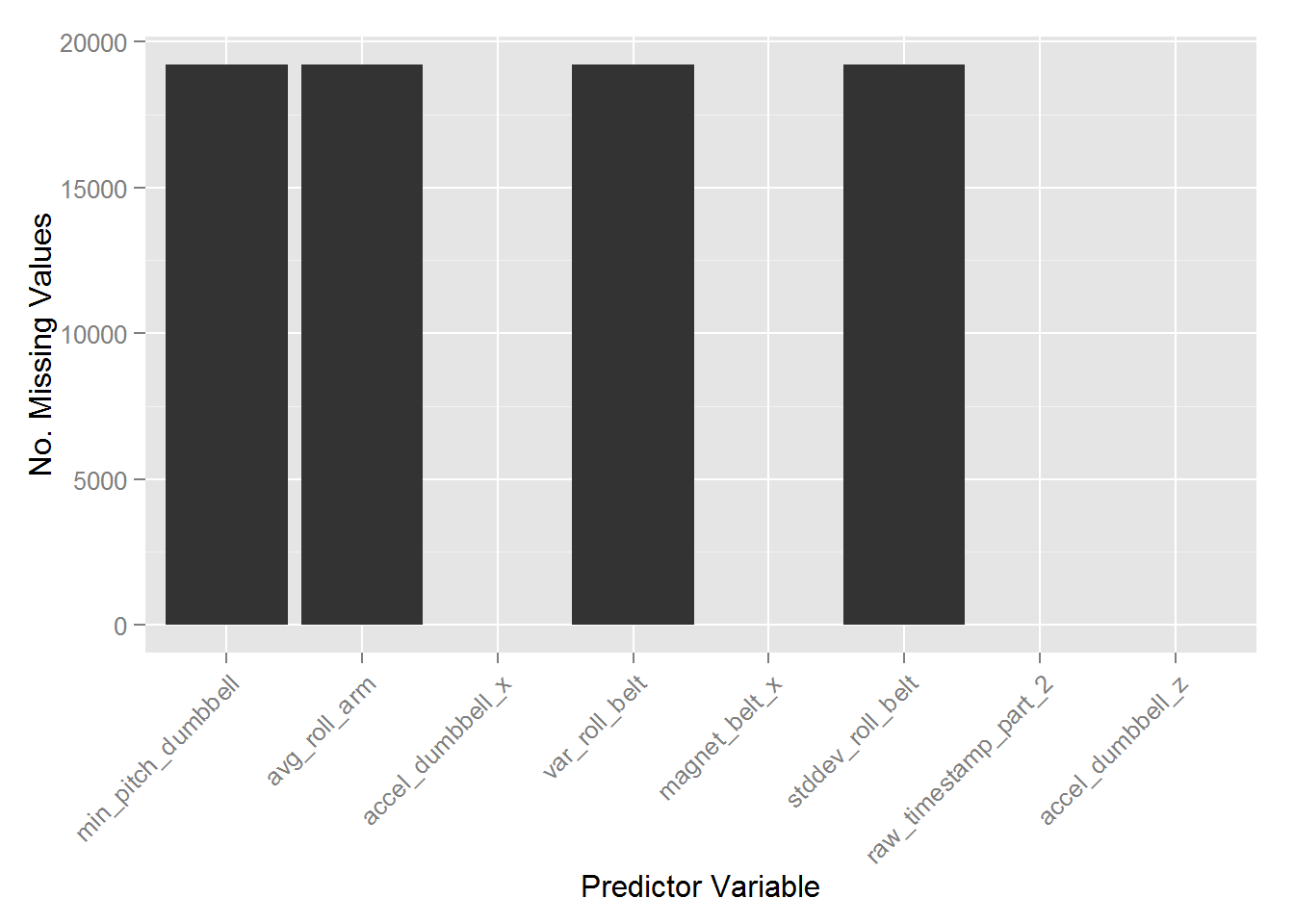
ordered = FALSE)

qplot(x = variable, y = missing, data = dfmissing, geom = "bar",

stat = "identity", position = "dodge") +

xlab("Predictor Variable") + ylab("No. Missing Values") +

theme(axis.text.x = element\_text(angle = 45, hjust = 1))



*## Remove unnecessary Columns*

removeCols <- c("X", "user\_name", "raw\_timestamp\_part\_1", "new\_window",

"num\_window", "raw\_timestamp\_part\_2", "cvtd\_timestamp")

trainData <- trainData[, !names(trainData) %in% removeCols]

The next step is is replace all the missing values with zero.

*## Chnage all the missing values to ZERO*

trainData[is.na(trainData)] <- 0

*## Ensure all the data that can be used as features (numeric class)*

classe <- trainData$classe

features <- sapply(trainData, is.numeric)

trainData <- cbind(trainData[, features], classe)

#### Pre-Processing the Data

Now the data has been cleaned, it must be pre-processed to ensure the best predictors are selected to train the model. This process involves finding correlated predictors and removing them. This process ensures that any **redundant** (**99%** similar) predictors are removed.

*## Find correlations in the data for further dimentionality reduction*

correlated <- findCorrelation(cor(trainData[, -120]), cutoff = 0.99)

trainData.reduced <- trainData[, -correlated]

*##Subset the data for cross-validation (60/40 Split)*

subSet <- createDataPartition(y = trainData.reduced$classe, p = 0.6,

list = FALSE)

trainSet <- trainData.reduced[subSet, ]

valSet <- trainData.reduced[-subSet, ]

**library**(randomForest)

*## Fit the Random Forest Model*

rfModel <- randomForest(classe ~ ., data = trainSet, mtry = 15, ntree = 2048,

keep.forest = TRUE, importance = TRUE, proximity = TRUE)

*## Plot the Important Variables*

varImpPlot(rfModel)

*## Plot the Model*

layout(matrix(c(1, 2), nrow = 1), width = c(4, 1))

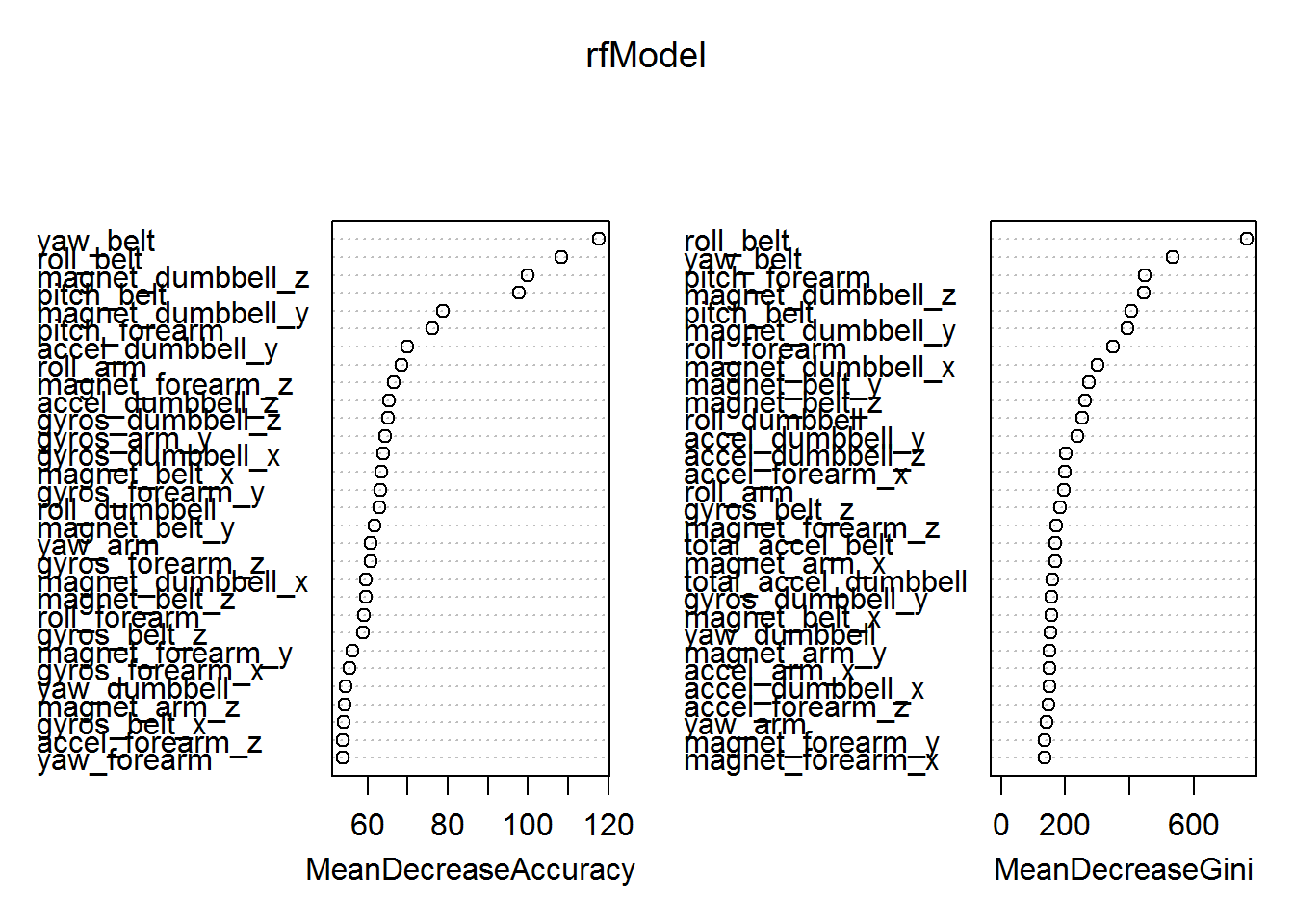
par(mar = c(5, 4, 4, 0))

plot(rfModel, log = "y")

par(mar = c(5, 0, 4, 2))

plot(c(0, 1), type = "n", axes = F, xlab = "", ylab = "")

legend("top", colnames(rfModel$err.rate), col = 1:6, cex = 0.8, fill = 1:6)



*## Plot the Model*

layout(matrix(c(1, 2), nrow = 1), width = c(4, 1))

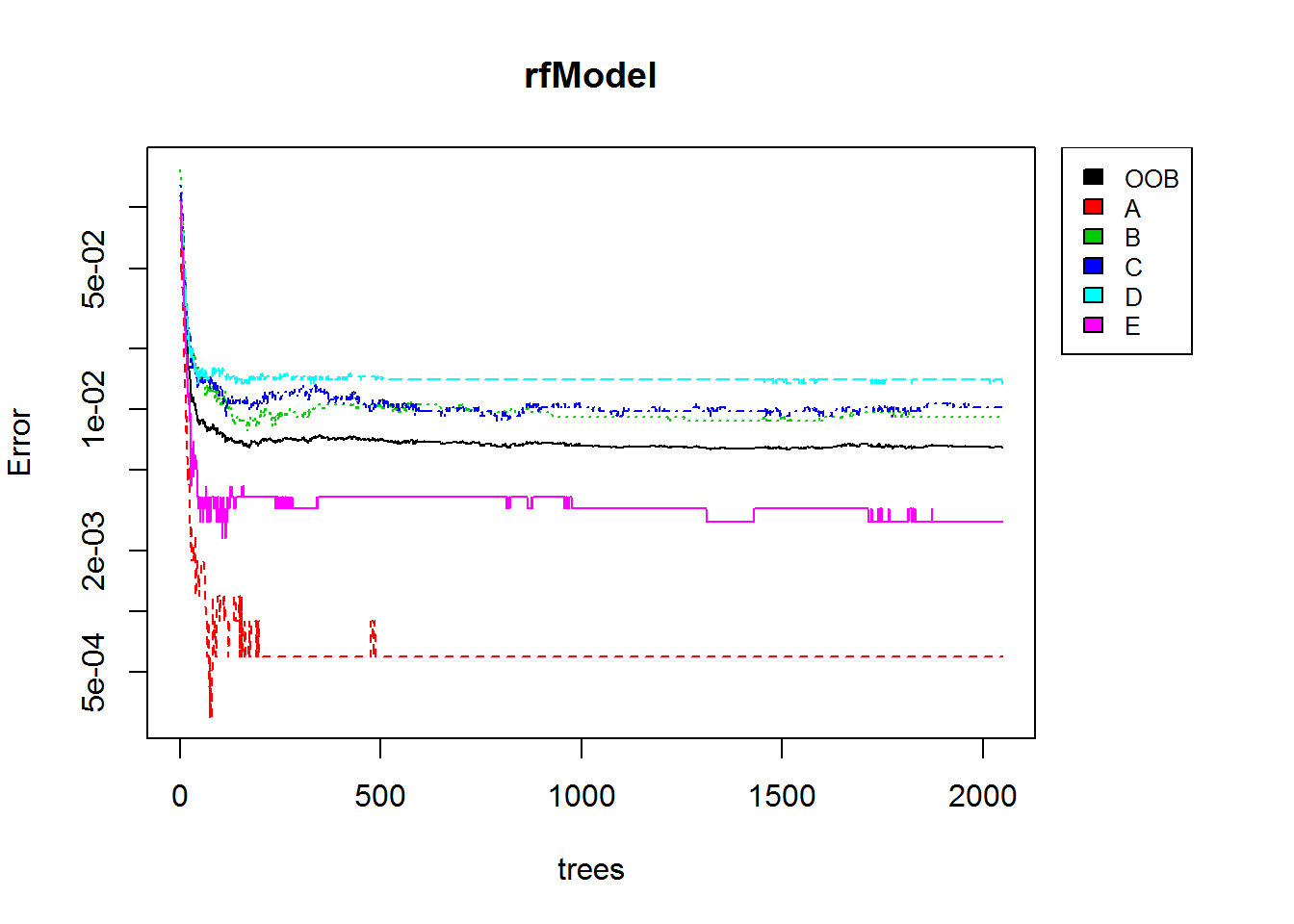
par(mar = c(5, 4, 4, 0))

plot(rfModel, log = "y")

par(mar = c(5, 0, 4, 2))

plot(c(0, 1), type = "n", axes = F, xlab = "", ylab = "")

legend("top", colnames(rfModel$err.rate), col = 1:6, cex = 0.8, fill = 1:6)



*## Predict on the Validation data set*

Predict <- predict(rfModel, valSet)

*## Confusion Matrix*

confusionMatrix(Predict, valSet$classe)

*# Apply Pre-processing methadology to the Test Data*

removeCols <- c("X", "user\_name", "raw\_timestamp\_part\_1", "new\_window",

"num\_window", "raw\_timestamp\_part\_2", "cvtd\_timestamp")

testData <- testData[, !names(testData) %in% removeCols]

testData[is.na(testData)] <- 0

classe <- testData$problem\_id

features <- sapply(testData, is.numeric)

testData <- cbind(testData[, features], classe)