setwd("G:/Advanced health/Project")

health<- Project\_1

summarizeColumns(health)

#Consider "?" missing value

levels(health$lymph\_node) [[1]] <- NA

par(mfrow=c(2,2))

for(i in 3:34){

hist(health[,i], main = colnames(health)[i], breaks = 100,

xlab = colnames(health)[i] )}

featurePlot(x = health[, 4:7],

y = health$Outcome ,

plot = "box",

## Add a key at the top

scales = list(y = list(relation="free"),

x = list(rot = 90)),

layout = c(4,1 ),

auto.key = list(columns = 2))

featurePlot(x = health[, 8:11],

y = health$Outcome ,

plot = "box",

## Add a key at the top

scales = list(y = list(relation="free"),

x = list(rot = 90)),

layout = c(4,1 ),

auto.key = list(columns = 2))

featurePlot(x = health[, 12:15],

y = health$Outcome ,

plot = "box",

## Add a key at the top

scales = list(y = list(relation="free"),

x = list(rot = 90)),

layout = c(4,1 ),

auto.key = list(columns = 2))

featurePlot(x = health[, 16:19],

y = health$Outcome ,

plot = "box",

## Add a key at the top

scales = list(y = list(relation="free"),

x = list(rot = 90)),

layout = c(4,1 ),

auto.key = list(columns = 2))

featurePlot(x = health[, 20:23],

y = health$Outcome ,

plot = "box",

## Add a key at the top

scales = list(y = list(relation="free"),

x = list(rot = 90)),

layout = c(4,1 ),

auto.key = list(columns = 2))

featurePlot(x = health[, 24:27],

y = health$Outcome ,

plot = "box",

## Add a key at the top

scales = list(y = list(relation="free"),

x = list(rot = 90)),

layout = c(4,1 ),

auto.key = list(columns = 2))

featurePlot(x = health[, 28:31],

y = health$Outcome ,

plot = "box",

## Add a key at the top

scales = list(y = list(relation="free"),

x = list(rot = 90)),

layout = c(4,1 ),

auto.key = list(columns = 2))

featurePlot(x = health[, 32:33],

y = health$Outcome ,

plot = "box",

## Add a key at the top

scales = list(y = list(relation="free"),

x = list(rot = 90)),

layout = c(4,1 ),

auto.key = list(columns = 2))

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#1. Radius\_mean

health\_cd<- capLargeValues(health, target = "Outcome", cols = c("Radius\_mean"),threshold = 25)

#2. Texture\_mean

health\_cd\_1<- capLargeValues(health\_cd, target = "Outcome", cols = c("Texture\_mean"),threshold = 32)

#3. parimeter\_mean

health\_cd\_2<- capLargeValues(health\_cd\_1, target = "Outcome", cols = c("parimeter\_mean"),threshold = 165)

#4. area\_mean

health\_cd\_3<- capLargeValues(health\_cd\_2, target = "Outcome", cols = c("area\_mean"),threshold = 1800)

#5. smoothness\_mean

health\_cd\_4<- capLargeValues(health\_cd\_3, target = "Outcome", cols = c("smoothness\_mean"),threshold = 0.13)

#6. compactness\_mean

health\_cd\_5<- capLargeValues(health\_cd\_4, target = "Outcome", cols = c("compactness\_mean"),threshold = 0.25)

#7. concavity\_mean

health\_cd\_6<- capLargeValues(health\_cd\_5, target = "Outcome", cols = c("concavity\_mean"),threshold = 0.32)

#8. concavepoints\_mean

health\_cd\_7<- capLargeValues(health\_cd\_6, target = "Outcome", cols = c("concavepoints\_mean"),threshold = 0.15)

#9. symmetry\_mean

health\_cd\_8<- capLargeValues(health\_cd\_7, target = "Outcome", cols = c("symmetry\_mean"),threshold = 0.26)

#10. fractaldimensions\_mean

health\_cd\_9<- capLargeValues(health\_cd\_8, target = "Outcome", cols = c("fractaldimesnsions\_mean"),threshold = 0.082)

health\_cd\_10<- capLargeValues(health\_cd\_9, target = "Outcome" , cols = c("Radius\_SE"),threshold = 1.4)

#12. Texture\_SE

health\_cd\_11<- capLargeValues(health\_cd\_10, target = "Outcome", cols = c("Texture\_SE"),threshold = 2.5)

#13. parimeter\_SE

health\_cd\_12<- capLargeValues(health\_cd\_11, target = "Outcome", cols = c("parimeter\_SE"),threshold = 9.5)

#14. area\_SE

health\_cd\_13<- capLargeValues(health\_cd\_12, target = "Outcome", cols = c("area\_SE"),threshold = 175)

#15. smoothness\_SE

health\_cd\_14<- capLargeValues(health\_cd\_13, target = "Outcome", cols = c("smoothness\_SE"),threshold = 0.015)

#16. compactness\_SE

health\_cd\_15<- capLargeValues(health\_cd\_14, target = "Outcome", cols = c("compactness\_SE"),threshold = 0.06)

#17. concavity\_SE

health\_cd\_16<- capLargeValues(health\_cd\_15, target = "Outcome", cols = c("concavity\_SE"),threshold = 0.08)

#18. concavepoints\_SE

health\_cd\_17<- capLargeValues(health\_cd\_16, target = "Outcome", cols = c("concavepoints\_SE"),threshold = 0.0280)

#19. symmetry\_SE

health\_cd\_18<- capLargeValues(health\_cd\_17, target = "Outcome", cols = c("symmetry\_SE"),threshold = 0.036)

#20. fractaldimensions\_SE

health\_cd\_19<- capLargeValues(health\_cd\_18, target = "Outcome", cols = c("fractaldimesnsions\_SE"),threshold = 0.008)

#21. Radius\_worst

health\_cd\_20<- capLargeValues(health\_cd\_19, target = "Outcome", cols = c("Radius\_worst"),threshold = 32)

#22. Texture\_worst

health\_cd\_21<- capLargeValues(health\_cd\_20, target = "Outcome", cols = c("Texture\_worst"),threshold = 46)

#24. area\_worst

health\_cd\_22<- capLargeValues(health\_cd\_21, target = "Outcome", cols = c("area\_worst"),threshold = 3200)

#25. smoothness\_worst

health\_cd\_23<- capLargeValues(health\_cd\_22, target = "Outcome", cols = c("smoothness\_worst"),threshold = 0.20)

#26. compactness\_worst

health\_cd\_24<- capLargeValues(health\_cd\_23, target = "Outcome", cols = c("compactness\_worst"),threshold = 0.8)

#27. concavity\_worst

health\_cd\_25<- capLargeValues(health\_cd\_24, target = "Outcome", cols = c("concavity\_worst"),threshold = 1.0)

#29. symmetry\_worst

health\_cd\_27<- capLargeValues(health\_cd\_25, target = "Outcome", cols = c("symmetry\_worst"),threshold = 0.5)

#30. fractaldimensions\_worst

health\_cd\_28<- capLargeValues(health\_cd\_27, target = "Outcome", cols = c("fractaldimesnsions\_worst"),threshold = 0.17)

health\_cd\_29<- capLargeValues(health\_cd\_28, target = "Outcome", cols = c("tumor\_size"),threshold = 8)

health\_out <- health\_cd\_29

featurePlot(x = health\_out[, 4:33],

y = health\_out$Outcome ,

plot = "box",

## Add a key at the top

scales = list(y = list(relation="free"),

x = list(rot = 90)),

layout = c(4,1 ),

auto.key = list(columns = 2))

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

#Pre-processing (Missing values, scaling, Centered)

health\_imp1<- impute(health\_out, target = "Outcome", classes = list(factor = imputeMode(),integer = imputeMean()),

cols =list("lymph\_node"), dummy.classes = c("integer", "factor"), dummy.type = "factor")

health\_scaled<- health\_imp1$data

health\_scaled$lymph\_node<-as.numeric(health\_scaled$lymph\_node)

health\_1 <- model.matrix(Outcome~.-1 , data = health\_scaled)

normalizeFeatures(health\_scaled, target = "Outcome", method = "standardize")

preProcess(health\_scaled, method = c("BoxCox","knnImpute"))

nzv<-nearZeroVar(health\_1, saveMetrics = TRUE)

healthCor<- cor(health\_1)

abs(healthCor) > .85

sum(abs(healthCor[upper.tri(healthCor)]) > .85)

health\_scaled$area\_mean <- NULL

health\_scaled$Radius\_SE <- NULL

health\_scaled$Radius\_worst <- NULL

health\_scaled$concavepoints\_mean <- NULL

health\_scaled$area\_worst <- NULL

health\_scaled$lymph\_node.dummy<- NULL

health\_scaled$Radius\_mean<- NULL

#Splitting the data

trainIndex<- createDataPartition(health\_scaled$Outcome, p = 0.8,list = FALSE, times = 1)

training<- health\_scaled[trainIndex,]

testing<- health\_scaled[-trainIndex,]

#Feature Selection

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Control<- trainControl(method = "repeatedcv", number= 5, repeats= 3)

model<- train(Outcome~.-1,data = training,method = "lvq", trControl = Control)

Importance<- varImp(model, scale = FALSE)

plot(Importance)

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

fitcontrol <- trainControl(method = "repeatedcv", number= 5, summaryFunction = twoClassSummary,

classProbs = TRUE, allowParallel = TRUE)

# Here 10 times cross fold and this will be repeated 10 times

gbmgrid<- expand.grid(interaction.depth = c(1,2), n.trees = c(10,20), shrinkage = c(0.1,0.01),

n.minobsinnode = 10)

set.seed(123)

registerDoParallel(2)

getDoParWorkers()

gbmfit<- train(Outcome~.-1,data = training,method = "gbm",

metric = "ROC", trControl = fitcontrol, tuneGrid = gbmgrid, verbose = FALSE

)

res<- gbmfit

gbm.pred<- predict(gbmfit,testing)

confusionMatrix(gbm.pred, testing$Outcome,positive = "R")

#xGBoost

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

set.seed(12345)

registerDoParallel(2, core = 2)

getDoParWorkers()

xgb.grid <- expand.grid(nrounds = 50, #the maximum number of iterations

max\_depth = c(2,6,10), eta = c(0.1,0.4),

gamma = 0, colsample\_bytree = .7, min\_child\_weight = 1,

subsample= c(0.8,0.1))

xgb.tune<- train(Outcome~.-1,data = training,method = "xgbTree",

metric = "ROC", trControl = fitcontrol, tuneGrid = xgb.grid)

xgb.pred<- predict(xgb.tune, testing)

confusionMatrix(xgb.pred, testing$Outcome, positive = "R")

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

trainTask<- makeClassifTask(id= "Breast\_cancer" ,data = training, target = "Outcome",

positive = "N")

testTask<- makeClassifTask(id= "Breast\_cancer" ,data = testing, target = "Outcome",

positive = "N")

#Logistic Regression

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

logistic.learner<- makeLearner("classif.logreg", predict.type = "response")

cv.logistic<- crossval(learner = logistic.learner, task= trainTask, iters = 3,

stratify = TRUE, measures = acc, show.info = F)

cv.logistic$aggr

mod\_log<- mlr::train(logistic.learner, trainTask)

getLearnerModel(mod\_log)

names(mod\_log)

mod\_log$learner

fv<- generateFilterValuesData(trainTask, method = c("information.gain", "chi.squared"))

plotFilterValues(fv)

logpredict<- predict(mod\_log, testTask)

plotLearnerPrediction(ran\_classif.lrn, task = classif.task, cv = 0, features = c("Petal.Length", "Petal.Width"))

#R part

#\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

library(rpart)

library(rpart.plot)

library("rattle")

install.packages("fancyRpartPlot")

library(fancyRpartPlot)

rpart\_classLearner<- makeLearner("classif.rpart", predict.type = "response")

getParamSet("classif.rpart")

set\_cv<- makeResampleDesc("CV", iters = 3L)

gs<- makeParamSet(

makeIntegerParam("minsplit", lower = 2, upper = 5),

makeIntegerParam("minbucket", lower = 1, upper = 5),

makeNumericParam("cp", lower = 0.001, upper = 0.2)

)

gscontrol<- makeTuneControlGrid()

stune<- tuneParams(learner = rpart\_classLearner, resampling = set\_cv, task= trainTask, par.set= gs,

control = gscontrol, measures = acc)

stune$x

t.tree<- setHyperPars(rpart\_classLearner, par.vals = stune$x)

t.part<- mlr::train(t.tree, trainTask)

getLearnerModel(t.part)

tpmodel <- predict(t.part, testTask)

set.seed(3456)

library(e1071)

fit1 <- naiveBayes(training\_filter, training\_filter$Outcome, type="raw")

pred1 <- predict(fit1, testing\_filter, type="class")

confusionMatrix(pred1, testing\_filter$Outcome)

svm.model\_1<- svm(Outcome~.-7, data = training\_filter)

svm.pred\_1 <- predict(svm.model\_1, testing\_filter[,-6])

t1<- table(testing\_filter$Outcome, svm.pred\_1)