R Notebook

```r
if(!require("caret")){install.packages("caret")}
if(!require("tidyverse")){install.packages("tidyverse")}
if(!require("ISLR2")){install.packages("ISLR2")}
if(!require("boot")){install.packages("boot")}
if(!require("MASS") ){ install.packages("MASS") }
if(! require("leaps") ){ install.packages("leaps") }
if(! require("glmnet") ){ install.packages("glmnet") }
if(! require("pls") ){ install.packages("pls") }
if(!require("lmvar")) {install.packages("lmvar")}
if(!require("splines")) {install.packages("splines")}
library(boot)
library(caret)
library(tidyverse)
library(ISLR2)
library(MASS)
library(leaps)
library(glmnet)
library(pls)
library(lmvar)
library(splines)
library(tidyverse)
library(magrittr)
library(ISLR)
library(caret)
library(e1071)
library(MASS)
library(neuralnet)
library(tensorflow)
library(keras)
library(quantmod)
library(dplyr)
```

```r
data <- read.csv("data.csv")
data$Net.Income.Flag <- NULL
data <- data[, !names(data) %in% c("ROA.B..before.interest.and.depreciation.after.tax",
           "ROA.C..before.interest.and.depreciation.before.interest",
           "Persistent.EPS.in.the.Last.Four.Seasons",
           "Operating.Profit.Per.Share..Yuan...",
           "Current.Liabilities.Equity",
           "Current.Liability.to.Equity",
           "Operating.Gross.Margin",
           "Pre.tax.net.Interest.Rate",
           "Operating.Profit.Rate")]
```

```
#Scale
library(corrgram)
data$Bankrupt. <- as.factor(data$Bankrupt.)
head(data)
```

```
set.seed(1000)

splitSample <- sample(1:2, size=nrow(data), prob=c(0.9,0.1), replace = TRUE)

train_set <- data[splitSample==1,]
valid_set <- data[splitSample==2,]
```

```
bankrupts = 0
for (i in train_set$Bankrupt.){
  if (as.integer(i) == "1"){
    bankrupts = bankrupts + 1
  }
}
print("Training Set:")
```

```
## [1] "Training Set:"
```

```
paste("Bankrupts:", bankrupts)
```

```
## [1] "Bankrupts: 198"
```

```
paste("No Bankrupts:", length(train_set[,1]) - bankrupts)
```

```
## [1] "No Bankrupts: 5970"
```

```
print("")
```

```
## [1] ""
```

```
bankrupts.2 = 0
for (i in valid_set$Bankrupt.){
  if (as.integer(i) == "1"){
    bankrupts.2 = bankrupts.2 + 1
  }
}
print("Valid Set:")
```

```
## [1] "Valid Set:"
```

```
paste("Bankrupts:", bankrupts.2)
```

```
## [1] "Bankrupts: 22"
```

```r
paste("No Bankrupts:", length(valid_set[,1]) - bankrupts.2)
```

```
## [1] "No Bankrupts: 629"
```

```r
#Standardize sets
count = 2
check = 0
while(count < length(data[3,]) && check == 0){
  count = count + 1
  if(colnames(data[1,])[count] == "Liability.Assets.Flag"){
    check = 1
  }
}
```

```r
for (i in 2:length(data[3,])){
  if(i != count){
    means = mean(train_set[,i])
    sds = sd(train_set[,i])
    valid_set[,i] = (valid_set[,i]-means)/sds
  }
}

train_liability <- train_set[,count]

total_col <- length(data[3,])-1

train_set[,count] <- NULL

train_set[,2:total_col] <- scale(train_set[,2:total_col])

train_set[,total_col+1] <- train_liability

valid_liability <- valid_set[,count]
valid_set[,count] <- NULL
valid_set[,total_col+1] <- valid_liability

colnames(train_set)[total_col+1] <- "Liability.Assets.Flag"
colnames(valid_set)[total_col+1] <- "Liability.Assets.Flag"
```

```r
#10-fold CV
control <- trainControl(method="cv", number=10)
```

```r
#Logistic Regression
fit.glm <- caret::train(Bankrupt. ~ .,
                  data = train_set,
                  method = "glm",
                  family = "binomial",
                  trControl = control)
```

```r
#Random Forest
library(randomForest)
```

```
train_x <- train_set[,2:total_col+1]
train_y <- train_set$Bankrupt.

tune <- tuneRF(train_x, train_y, ntreeTry = 500, plot = FALSE)
```

```
## mtry = 9  OOB error = 2.85%
## Searching left ...
## mtry = 5     OOB error = 2.95%
## -0.03409091 0.05
## Searching right ...
## mtry = 18    OOB error = 2.85%
## 0 0.05
```

```
mtry = 0

min <- tune[1,2]

rf_count <- 1
mtry <- tune[1,1]

for (i in length(tune[,2])){
  if(tune[i,2] < min){
    mtry <- tune[i,1]
  }
}
```

```
tunegrid <- expand.grid(.mtry=c(9))
fit.rf<- caret::train(Bankrupt.~., data=train_set, method = 'rf', metric = 'Accuracy', tuneGrid = tuneg
fit.rf
```

```
## Random Forest
##
## 6168 samples
##   85 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 5551, 5551, 5551, 5552, 5551, 5551, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9704935  0.2481772
##
## Tuning parameter 'mtry' was held constant at a value of 9
```

```
control.radial <- trainControl(method="cv", number=10)
fit.svm.radial <- caret::train(Bankrupt.~., data = train_set, method="svmRadial", trControl=control.radi
```

```
fit.svm.radial$bestTune
```

```r
#KNN
fit.knn <- caret::train(Bankrupt.~., data=train_set, method="knn", trControl=control)
fit.knn$bestTune
```

```r
#SVM Sigmoid
cost <- c(0.001, 0.01, 0.1, 1, 10, 1/dim(train_set))
gamma <- c(0.01, 0.1, 0.25, 0.5, 1)
sigmoid_svms_accu_mean <- data.frame(
  cost = 0,
  gamma = 0,
  accuracy = 0
)
```

```r
j = 1
for(c in cost){
    for (g in gamma){
      sigmoid_svms_accu <- c()
      for (i in length(10)){

        splitSample <- sample(1:2, size=nrow(train_set), prob=c(0.9,0.1), replace = TRUE)
        train <- train_set[splitSample==1,]
        test <- train_set[splitSample==2,]
        sigmoid_svms <- svm(Bankrupt. ~., data = train, kernel = "sigmoid", cost = c, gamma = g)
        predict.svms <- predict(sigmoid_svms, test)
        cm.svms <- confusionMatrix(predict.svms, test$Bankrupt.)
        sigmoid_svms_accu[i] <- cm.svms$overall['Accuracy']
      }
      new_row = list(cost = c, gamma= g, accuracy = mean(sigmoid_svms_accu))
      sigmoid_svms_accu_mean = rbind(sigmoid_svms_accu_mean,new_row)
      j = j +1
  }
}
```

```r
max_accuracy = 0
index = 1
j = 1
for(i in sigmoid_svms_accu_mean$accuracy){
  if(i > max_accuracy){
    max_accuracy = i
    index = j
  }
  j = j+1
}

#sigmoid_svms_accu_mean[which.max(sigmoid_svms_accu_mean$accuracy),]
#which.max(sigmoid_svms_accu_mean$accuracy)

sigmoid_svms <- svm(Bankrupt. ~., data = train_set, kernel = "sigmoid", cost = sigmoid_svms_accu_mean[i
```

```r
# Define grid of hyperparameters
xgboost.grid <- expand.grid(
  nrounds = c(100),
  max_depth = c(2, 4, 6),
```

```
  eta = c(0.01, 0.05, 0.1),
  gamma = c(0, 0.1, 0.2),
  colsample_bytree = c(0.5, 0.75, 1),
  min_child_weight = c(1, 3, 5),
  subsample = c(0.5, 0.75, 1)
)

# Define cross-validation scheme
ctrl <- trainControl(
  method = "cv",
  number = 1,
  verboseIter = FALSE,
  allowParallel = TRUE
  # numberParallel = 4
)

# Train the model using cross-validation
invisible({capture.output({fit.xgb_model <- caret::train(
  Bankrupt. ~ .,
  data = train_set,
  method = "xgbTree",
  trControl = ctrl,
  tuneGrid = xgboost.grid,
  nthread = 4
)
})})
```

```
#Validation Tests

#                            Actually Positive       Actually Negative
#                            --------------------    --------------------
#    Predicted positive      True positives (TP)     False positives (FP)
#    Predicted negative      False negatives (FN)    True negatives (TN)

#SENSITIVITY = TP/(TP+FN)
#SPECIFICITY = TN/(TN+FP)
predict.glm <- predict(fit.glm, valid_set)
cm.glm <- confusionMatrix(predict.glm, valid_set$Bankrupt., positive = "0")
predictions.rf <- predict(fit.rf, valid_set)
cm.rf <- confusionMatrix(predictions.rf, valid_set$Bankrupt.)
predict.svm <- predict(fit.svm.radial, valid_set)
cm.svm <- confusionMatrix(predict.svm, valid_set$Bankrupt.)
predict.knn <- predict(fit.knn, valid_set)
cm.knn <- confusionMatrix(predict.knn, valid_set$Bankrupt.)

predict.svm.sigmoid <- predict(sigmoid_svms, valid_set)
cm.svm.sigmoid <- confusionMatrix(predict.svm.sigmoid, valid_set$Bankrupt.)

predict.xgb <- predict(fit.xgb_model, valid_set)
cm.xgb <- confusionMatrix(predict.xgb, valid_set$Bankrupt.)
```

```
print("LR")
```

```
## [1] "LR"
```

```
cm.glm$overall['Accuracy']
```

```
##  Accuracy
## 0.9677419
```

```
cm.glm$byClass['Sensitivity']
```

```
## Sensitivity
##   0.9968203
```

```
cm.glm$byClass['Specificity']
```

```
## Specificity
##   0.1363636
```

```
print("KNN")
```

```
## [1] "KNN"
```

```
cm.knn$overall['Accuracy']
```

```
##  Accuracy
## 0.9708141
```

```
cm.knn$byClass['Sensitivity']
```

```
## Sensitivity
##           1
```

```
cm.knn$byClass['Specificity']
```

```
## Specificity
##   0.1363636
```

```
print("RF")
```

```
## [1] "RF"
```

```
cm.rf$overall['Accuracy']
```

```
##  Accuracy
## 0.9708141
```

```r
cm.rf$byClass['Sensitivity']
```

```
## Sensitivity
##   0.9984102
```

```r
cm.rf$byClass['Specificity']
```

```
## Specificity
##   0.1818182
```

```r
print("SVM Radial")
```

```
## [1] "SVM Radial"
```

```r
cm.svm$overall['Accuracy']
```

```
##  Accuracy
## 0.9662058
```

```r
cm.svm$byClass['Sensitivity']
```

```
## Sensitivity
##           1
```

```r
cm.svm$byClass['Specificity']
```

```
## Specificity
##           0
```

```r
print("SVM Sigmoidal")
```

```
## [1] "SVM Sigmoidal"
```

```r
cm.svm.sigmoid$overall['Accuracy']
```

```
##  Accuracy
## 0.9662058
```

```r
cm.svm.sigmoid$byClass['Sensitivity']
```

```
## Sensitivity
##           1
```

```r
cm.svm.sigmoid$byClass['Specificity']
```

```
## Specificity
##           0
```

```r
print("XGBoost")
```

```
## [1] "XGBoost"
```

```r
cm.xgb$overall['Accuracy']
```

```
##  Accuracy
## 0.9677419
```

```r
cm.xgb$byClass['Sensitivity']
```

```
## Sensitivity
##           1
```

```r
cm.xgb$byClass['Specificity']
```

```
## Specificity
##  0.04545455
```

```r
print("Logistic Regression: ")
```

```
## [1] "Logistic Regression: "
```

```r
cm.glm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 627  19
##          1   2   3
##
##                Accuracy : 0.9677
##                  95% CI : (0.9511, 0.9799)
##     No Information Rate : 0.9662
##     P-Value [Acc > NIR] : 0.4701816
##
##                   Kappa : 0.2124
##
##  Mcnemar's Test P-Value : 0.0004803
##
##             Sensitivity : 0.9968
##             Specificity : 0.1364
##          Pos Pred Value : 0.9706
##          Neg Pred Value : 0.6000
##              Prevalence : 0.9662
##          Detection Rate : 0.9631
##    Detection Prevalence : 0.9923
##       Balanced Accuracy : 0.5666
##
##        'Positive' Class : 0
##
```

```r
print("")
```

```
## [1] ""
```

```r
print("KNN Regression: ")
```

```
## [1] "KNN Regression: "
```

```r
cm.knn
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 629  19
##          1   0   3
##
##                Accuracy : 0.9708
##                  95% CI : (0.9548, 0.9823)
##     No Information Rate : 0.9662
##     P-Value [Acc > NIR] : 0.3022
##
##                   Kappa : 0.2338
##
##  Mcnemar's Test P-Value : 3.636e-05
##
##             Sensitivity : 1.0000
##             Specificity : 0.1364
##          Pos Pred Value : 0.9707
##          Neg Pred Value : 1.0000
##              Prevalence : 0.9662
##          Detection Rate : 0.9662
##    Detection Prevalence : 0.9954
##       Balanced Accuracy : 0.5682
##
##        'Positive' Class : 0
##
```

```r
print("")
```

```
## [1] ""
```

```r
print("Random Forest: ")
```

```
## [1] "Random Forest: "
```

```r
cm.rf
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 628  18
##          1   1   4
##
##                Accuracy : 0.9708
##                  95% CI : (0.9548, 0.9823)
##     No Information Rate : 0.9662
##     P-Value [Acc > NIR] : 0.3022239
##
##                   Kappa : 0.2874
##
##  Mcnemar's Test P-Value : 0.0002419
##
##             Sensitivity : 0.9984
##             Specificity : 0.1818
##          Pos Pred Value : 0.9721
##          Neg Pred Value : 0.8000
##              Prevalence : 0.9662
##          Detection Rate : 0.9647
##    Detection Prevalence : 0.9923
##       Balanced Accuracy : 0.5901
##
##        'Positive' Class : 0
##
```

```r
print("")
```

```
## [1] ""
```

```r
print("Support Vector Machine Radial: ")
```

```
## [1] "Support Vector Machine Radial: "
```

```r
cm.svm
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 629  22
##          1   0   0
##
##                Accuracy : 0.9662
##                  95% CI : (0.9493, 0.9787)
##     No Information Rate : 0.9662
##     P-Value [Acc > NIR] : 0.5564
##
##                   Kappa : 0
##
```

```
##  Mcnemar's Test P-Value : 7.562e-06
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9662
##          Neg Pred Value :    NaN
##              Prevalence : 0.9662
##          Detection Rate : 0.9662
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

```
print("")
```

```
## [1] ""
```

```
print("Support Vector Machine Sigmoidal: ")
```

```
## [1] "Support Vector Machine Sigmoidal: "
```

```
cm.svm.sigmoid
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 629  22
##          1   0   0
##
##                Accuracy : 0.9662
##                  95% CI : (0.9493, 0.9787)
##     No Information Rate : 0.9662
##     P-Value [Acc > NIR] : 0.5564
##
##                   Kappa : 0
##
##  Mcnemar's Test P-Value : 7.562e-06
##
##             Sensitivity : 1.0000
##             Specificity : 0.0000
##          Pos Pred Value : 0.9662
##          Neg Pred Value :    NaN
##              Prevalence : 0.9662
##          Detection Rate : 0.9662
##    Detection Prevalence : 1.0000
##       Balanced Accuracy : 0.5000
##
##        'Positive' Class : 0
##
```

```
print("")
```

```
## [1] ""
```

```
print("XGBoost: ")
```

```
## [1] "XGBoost: "
```

```
cm.xgb
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 629  21
##          1   0   1
##
##                Accuracy : 0.9677
##                  95% CI : (0.9511, 0.9799)
##     No Information Rate : 0.9662
##     P-Value [Acc > NIR] : 0.4702
##
##                   Kappa : 0.0843
##
##  Mcnemar's Test P-Value : 1.275e-05
##
##             Sensitivity : 1.00000
##             Specificity : 0.04545
##          Pos Pred Value : 0.96769
##          Neg Pred Value : 1.00000
##              Prevalence : 0.96621
##          Detection Rate : 0.96621
##    Detection Prevalence : 0.99846
##       Balanced Accuracy : 0.52273
##
##        'Positive' Class : 0
##
```

```
rmarkdown::render("ST694Project copy.Rmd", output_format = "pdf_document")
```