Common Code

Vikas Goel, Srikanth Shetty, Nilima Ghosh

Tue Feb 15 18:00:00 2019

```
# All these common functions originally created by "Vikas Goel" in order to make code as modular
# as possible. After Project completion, this code will be available at my GitHub account
# "https://github.com/AeroVikas"
# Function names are self explanatory however enough code comments are provided wherever necessary
# In Future, Will add more algorithms like k-means, k-nearest neighbours and more.
# For any clarifiation, contact me at vikas.aero@gmail.com

################[   packages to be installed ]############################
#install.packages("caret")
#install.packages("deepnet")
#install.packages("devtools")
#install.packages("dplyr")
#install.packages("dummies")
#install.packages("e1071")
#install.packages("ggfortify")
#install.packages("ggplot2")
#install.packages("ggpubr")
#install.packages("grid")
#install.packages("gridExtra")
#install.packages("pROC")
#install.packages("randomForest")
#install.packages("reshape2")
#install.packages("rpart")
#install.packages("rpart.plot")
#install.packages("bindrcpp")
#install.packages("xgboost")

# KERAS requires special installation
#install.packages("keras)
#library(keras)
#install_keras() #Creates tensorflow enviornment and install related modules



################ Global Variables ##################################################################
listRocCurves<<-c() #List which will store the data for each model to be used in ROC curves
dfModelPerformance<<-data.frame() #Dataframe which will store the performance data for each model
i<<-1 #flag initialized by 1
CostFactorOfNewHiring <<- 4.0
CostFactorOfRetainingOldEmployee <<- 1.0
CostToRetainEmployee <<- 5 #1000$
options(warn=-1)
#####################################################################################################

#Single function for the familiarity with Data.
DataFamilarization <- function(data)
{
  # Check the dimensions of data
  dimen <- dim(data)
  print(paste("Size of data : Columns = ", dimen[2], "Rows = ", dimen[1])
)

  # View the first 6 rows of data
  head(data)

  # View the last 6 rows of data
  tail(data)

  # View a condensed summary of the data
  str(data)
```

```
  # Check the class of data
  class(data)

  # View the column names of data
  names(data)

  # View structure of lunch, the dplyr way
  library(dplyr)
  glimpse(data)

  # View a summary
  summary(data)

  #Plot
  boxplot(data)
  coef(data)
}

#############################################################################################
PrintCorrelationHeatmap <- function(data)
{
  #Correlation Matrix
  library(reshape2)
  library(ggplot2)
  melted_cormat<-melt(cor(data))
  ggplot(melted_cormat, aes(Var2, Var1, fill = value)) +

    geom_tile(color = "black") +

    scale_fill_gradient2(low = "red", high = "blue", limit = c(-1, 1), name = "Correlation") +

    theme(axis.text.x = element_text(angle = 10, hjust = .5))

}

#############################################################################################
TrainTestSplit = function(data, splitFactor = 0.7, train = TRUE
)
{
  #split between train and test
  set.seed(123
)
  trainData <- sample(1:nrow(data), splitFactor * nrow(data))
  if (train == TRUE)
{
    return(data[trainData,])
  } else
{
    return(data[-trainData,])
  }
}

#############################################################################################
GetConfusionMatix <- function(testLabelData, test, testprediction, modelType)
{
  #Prepare Confusion Matrix
  print(modelType)
  cm <- caret::confusionMatrix(as.factor(testprediction),
                               as.factor(testLabelData
),
                               dnn = c("Prediction", "Actual"
))
  class(cm)
  print (paste("Test data Confusion Matrix for ",modelType," Model:")
)
  print(cm)
```

```
  #Get Performance metrics
  accuracy <- paste(round(100*cm$overall["Accuracy"], digits = 2), "%"
)
  sensitivity <- paste(round(cm$byClass["Sensitivity"], digits = 4)
)
  specificity <- paste(round(cm$byClass["Specificity"], digits = 4)
)
  precision <- paste(round(cm$byClass["Precision"], digits = 4))
  recall <- paste(round(cm$byClass["Recall"], digits = 4)
)

  #Get Roc and Auc of a particular Model
  library(pROC)
  roc_obj <- roc(testLabelData, as.numeric(testprediction))
  areaUnderCurve <- round(auc(roc_obj), digits = 2
)
  falsePositive <- cm$table[2,1
]
  falseNegative <- cm$table[1,2
]
  costToCompany<-
    (CostFactorOfNewHiring*falseNegative)+(CostFactorOfRetainingOldEmployee*falsePositive)
  costToCompany<-paste("$", round(costToCompany * CostToRetainEmployee, digits = 2), "K"
)
  # Store model performance data into data frame so at the end of all models, we can print summary
  #df <- data.frame(i,modelType);  print(df)
  df <- data.frame(MODEL = modelType,
                   Accuracy = accuracy,

                   AUC=areaUnderCurve
,
                   Specificity=specificity
,
                   Precision=precision
,
                   Sensitivity_Recall=sensitivity
,
                   CostToCompany = costToCompany
)

  print (paste("Model Performance:")
)
  print(df)
  print
(
    paste
(
      "False Negative = "
,
      falseNegative,
      "   False Positive = "
,
      falsePositive,
      " Cost To Company = "
,
      costToCompany
    )
  )

  # Store model ROC curve data into list so at the end of all models, we can plot combined ROC curve
  listRocCurves[[i]] <<- roc_obj
  names(listRocCurves)[i] <<- modelType

  dfModelPerformance <<- rbind(df,dfModelPerformance)
  i <<- i + 1 #increment flag
}

#########################################################################################################
#This function does 2 things
```

```r
#1. print the performance metrices of each model in table format.
#2. plot combined ROC curve for all the mdoels.
CompareModelsAndPlotCombinedRocCurve <- function()
{
  #print(dfModelPerformance)
  library(gridExtra)
  library(grid)
  grid.newpage()
  grid.table(dfModelPerformance)
  #plot ROC Curve
  rocCurve <- ggroc( listRocCurves, alpha = 1, size = 1) +

    ggtitle("ROC[Receiver Operating Characteristics] curve") +

    theme(axis.text = element_text(colour = "blue")
)
  plot(rocCurve)
}

#########################################################################################################
#Function to normalize the data by dividing maximum of that data.
# This function to be used for data which is +ve so that it will transform in range 0-1
normalize_DivideByMax <- function(x)
{
  return(x/max(x))
}

#########################################################################################################
CreateLogisticRegressionModel <- function(trainlabel, testLabelData, IndependentVariables,
                                          train, test)
{

  modelType = "Logistic Regression"
  print(paste("Creating Model for ",modelType))

  formula <- as.formula(paste(trainLabel, paste(IndependentVariables), sep = " ~ ")
)
  model <- glm(formula, data = train, binomial()
)

  #Summary of Logistic Regression Model
  print(summary(model))

  ##Stepwise regression to take only relevant variables
  #modelStep<-step(model, direction = "both", trace = 1)
  #print(summary(modelStep))

  #Predictions using Logistic regression Model with test Dataset
  #considering 1 for prob >=.5 and 0 0therwise
  predictions <- predict(model, test)
  testprediction <- ifelse(predictions >= .5, 1, 0
)

  #Create confusion Matrix
  GetConfusionMatix(testLabelData, test, testprediction, modelType)
}

#########################################################################################################
CreateDecisionTreeModel <- function(trainlabel, testLabelData, IndependentVariables, train, test)
{
  modelType = "Decision Tree"
  print(paste("Creating Model for ", modelType))

  #Import Library
  library(rpart)
```

```
  #Fitting model
  formula <- as.formula(paste(trainLabel, paste(IndependentVariables), sep = " ~ ")
)
  model <- rpart::rpart(formula, data = train, method = 'class') #class to Fit a binary model

  #Summary of Model
  #summary(model)

  #plot
  library(rpart.plot)
  rpart.plot(model,type = 5,fallen.leaves = T,extra =
8
             ,cex = .58,trace = 1, main = "Decision Tree",cex.main = 1.5
,
             leaf.round = 1,prefix = "",branch.col = "blue",branch.lwd = 2,box.palette = "RdGn"
,
             nn = F, branch.lty = 1) #3 dotted branch lin
es

  #Predictions using Model with test Dataset
  predictions <- predict(model, test, type = "class"
)


  #Create confusion Matrix
  GetConfusionMatix(testLabelData, test, predictions, modelType)
}

##############################################################################################
CreateRandomForestModel <- function(trainlabel, testLabelData,
                                    IndependentVariables, train, test, numTree)
{
  modelType = "Random Forest"
  print(paste("Creating Model for ", modelType))

  #Import Library
  library(randomForest)

  #Fitting model
  formula <- as.formula(paste(trainLabel, paste(IndependentVariables), sep = " ~ ")
)
  model <- randomForest::randomForest(formula, data = train, importance = TRUE, ntree = numTree
)
  print(model)

  #Summary of Model
  summary(model)

  ## Look at variable importance:
  print(round(importance(model), 5)
)


  #Predictions using Model with test Dataset
  predictions <- predict(model, test)

  ##Append the predicted values in the test dataset and save the file
  # test[ ,(ncol(test)+1)] <- predictions
  # names(test)[(ncol(test))]<-paste("Predictions")
  # write.csv(test, file = "Data\\testWithPredictionsWithRandomForest.csv")
  #Create confusion Matrix
  GetConfusionMatix(testLabelData, test, predictions, modelType)
  # graphics.off()
  # par("mar")
  # par(mar=c(1,1,1,1))
  # plot(model)
  # print (paste("Minimum Error at Tree : ",which.min(model$err.rate[,1])))
```

```
  #Create Variable importance plot

  varImpPlot(model)
}

#############################################################################################################
CreateKernalSvmModel <- function(trainlabel, testLabelData, dependentVariables, train, test)
{
  modelType = "Kernal SVM"
  print(paste("Creating Model for ", modelType))

  #Guassuian Kernal SVM Model
  library(e1071)

  #Fitting model
  formula <- as.formula(paste(trainLabel, paste(dependentVariables), sep = " ~ ")
)
  model <- e1071::svm(formula = train$left ~ .
,
                      data = train, type = 'C-classification', kernel = 'radial
')

  #Summary of Model
  model

  #Predictions using Model with test Dataset
  predictions <- predict(model, test)

  #Create confusion Matrix
  GetConfusionMatix(testLabelData, test, predictions, modelType)
}

#############################################################################################################
CreateNaiveBayesModel <- function(trainlabel, testLabelData, dependentVariables, train, test)
{
  modelType = "Naive Bayes"
  print(paste("Creating Model for ", modelType))

  #Fitting the Naive Bayes Model
  library(e1071)
  formula <- as.formula(paste(trainLabel, paste(dependentVariables), sep = " ~ ")
)
  model <- e1071::naiveBayes(formula, data = train)
  summary(model)

  #Predictions using Model with test Dataset
  predictions <- predict(model, test)

  #Making the Confusion-Matrix.
  GetConfusionMatix(testLabelData, test, predictions, modelType)
}

#############################################################################################################
#Neural network model with deepnet Library.
# this method is optimized for HR-Analytics project and has 3 hidden layers with fixed neurons
# Need to be modified in order to use for some other dataset.
CreateDeepnetNNModel <- function(train,test,targetColumnNumber,hiddenLayers,numepochs)
{
  modelType = "Deepnet Neural Network"
  #numepochs =700
  #hiddenLayers=c(50, 30, 10)
  #targetColumnNumber=7
  library(keras)
  train_X <- as.matrix(train[,-targetColumnNumber])
  train_Y <- as.vector(train[, targetColumnNumber])
```

```
  train_Y_onehot = keras::to_categorical(train_Y)
  test_X <- as.matrix(test[,-targetColumnNumber])
  test_Y <- as.vector(test[, targetColumnNumber])
  test_Y_onehot = keras::to_categorical(test_Y)
  library(deepnet)

  #train and create the model with 3 layers of 50,30,10 neurons respectively
  model <- nn.train(train_X, train_Y_onehot, hidden =hiddenLayers , numepochs = numepochs)
  error <- nn.test(model, train_X, train_Y_onehot)
  print (paste("Train Accuracy of ", modelType, " is ", round(100*(1-error), digits = 2), "%")
)

  error <- nn.test(model, test_X, test_Y_onehot)
  print (paste("Test Accuracy of ", modelType, " is ", round(100*(1-error), digits = 2), "%")
)
  testoutput <- nn.predict(model, test_X)
  predict<- nn.predict(model,test_X)

  #predictions
  predictions <- ifelse(predict>0.5,1,0
)

  #Making the Confusion-Matrix.
  GetConfusionMatix(test$left, test, predictions[,2], modelType)
}

#################################################################################################
#Neural network model with Keras Library.
# this method is optimized for HR-Analytics project and has 4 hidden layers with fixed neurons
# Need to be modified in order to use for some other dataset.
CreateKerasNNModel <- function(train,test,targetColumnNumber,batchSize,numepochs,
                               validationSplit,lossFunction,errorMetrics)
{
  modelType = "Keras Neural Network"
  #targetColumnNumber=7
  #numepochs = 25
  #batchSize = 128
  #validationSplit = 0.2
  #lossFunction = "categorical_crossentropy"
  #errorMetrics = "accuracy"

  ## Installing / Loading Keras
  # install.packages("keras")

  library(keras)
  train_X <- as.matrix(train[,-targetColumnNumber])
  train_Y <- as.vector(train[, targetColumnNumber])
  train_Y_onehot = keras::to_categorical(train_Y)
  test_X <- as.matrix(test[,-targetColumnNumber])
  test_Y <- as.vector(test[, targetColumnNumber])
  test_Y_onehot = keras::to_categorical(test_Y)

  ## Creating the sequential model
  model = keras::keras_model_sequential() %>%

    layer_dense(units = 100, activation = "relu", input_shape = ncol(train_X)) %>
%
    layer_dense(units = 50 , activation = "relu") %>
%
    layer_dense(units = 25 , activation = "relu") %>
%
    layer_dense(units = 10 , activation = "relu") %>
%
    layer_dense(units = ncol(train_Y_onehot), activation = "softmax"
)

  model
```

```
  keras::compile(model, loss = lossFunction, optimizer = optimizer_rmsprop(), metrics = errorMetrics)
  history = keras::fit(model, train_X, train_Y_onehot,
                       epochs = numepochs, batch_size = batchSize, validation_split = validationSpli
t)
  plot(history)

  ## Predictions using test data
  predictions=keras::predict_classes(model, test_X)

  #Making the Confusion-Matrix.
  GetConfusionMatix(test$left, test, predictions, modelType)
}

#############################################################################################################
CreateXGBoostModel <- function(train, test,yActual,number = 10, classification = TRUE)

{
  modelType = "Extreme Gradient Boost"
  #library(tidyverse)
  library(caret)
  #library(xgboost)

  # Fit the model on the training set
  set.seed(123
)
  model <- train(satisfaction_level ~., data = train, method = "xgbTree"
,
                 trControl = trainControl("cv", number = number
))
  # Best tuning parameter
  model$bestTune

  #Variabe importance of Model
  varImp(model)

  # Make predictions on the test data
  predictions <- predict(model,test)

  if (classification == TRUE)
{
    # for classification : Compute model prediction accuracy rate
    print(mean(predictions == yActual))
  } else
{
    # Compute the average prediction error RMSE
    print(data.frame(RMSE = caret::RMSE(predictions, yActual),R2 = caret::R2(predictions, yActual)))
  }
}

#############################################################################################################
CreatePCA <- function(data,numComponents)
{
  print(paste("Creating Principle Components Analysis(PCA)")
)
  # Creating a data set of numeric variables as PCA is applicable on Numeric data
  #first create dummies(one hot encoding) for non-numeric Data
  library(dummies)
  dataWithDummies <- dummy.data.frame(data, sep = "."
)
  nums <- sapply(dataWithDummies, is.numeric)
  dataNumeric <-dataWithDummies[ , nums]

  #principal component analysis
  model <- prcomp(dataNumeric, scale. = T,center = T, rank. = 9
)
  print(summary(model))
```

```
  #str(model) #look at your PCA object.
  plot(model)
  #library(devtools)
  #install_github("vqv/ggbiplot")
  #library(ggbiplot)
  #ggbiplot(model, ellipse = TRUE, obs.scale = 1, var.scale = 1) +
  # scale_colour_manual(name = "Origin", values = c("forest green", "red3", "dark blue")) +
  # ggtitle("PCA") + theme_minimal() + theme(legend.position = "bottom")
  #take first 5 components
  PCADATA <- model$rotation[1:ncol(dataNumeric),1:numComponents]
  print(PCADATA)
  library(devtools)
  #install_github('sinhrks/ggfortify')
  library(ggfortify); library(ggplot2)
  autoplot(model,shape = FALSE, data=data,label=TRUE,label.size = 1
,
           loadings = TRUE, loadings.colour = 'blue',loadings.label = TRUE
,
           loadings.label.size = 4,loadings.label.colour="blue"
)
  # library(ggbiplot)
  # g <- ggbiplot(model, obs.scale = 1, var.scale = 1,
  #               groups = data.class, ellipse = TRUE, circle = TRUE)
  # g <- g + scale_color_discrete(name = '')
  # g <- g + opts(legend.direction = 'horizontal',
  #               legend.position = 'top')
  # print(g)
}

#############################################################################################################
#This is a very specialized function to get the Xi+Xi^2+Xi^3 polynomial
# Note: To be used very carefully for polynomial regression
# Limitation: it takes only one column as ignored column and need to be enhanced in case more
# columns to be ignored
GetSquareAndCubePolynomialInputs <- function(data, ignoredColumn)
{
  data_sq<- sapply(data[,ignoredColumn], function(x) x^2
)
  data_sq<- as.data.frame(data_sq)
  colnames(data_sq) <- paste(colnames(data_sq), "Square", sep = "_"
)
  data_cube<- sapply(data[,ignoredColumn], function(x) x^3
)
  data_cube<- as.data.frame(data_cube)
  colnames(data_cube) <- paste(colnames(data_cube), "Cube", sep = "_"
)
  data_poly<- cbind(data,data_sq,data_cube)
  return(data_poly)
}

#############################################################################################################
CreateStepwiseLinearRegressionModel <- function(data,targetColumnNumber,isPoly=FALSE)
{
  if(isPoly){
    data<- GetSquareAndCubePolynomialInputs(data,-1
)
  }
  #Split data into Train and Test
  train<-TrainTestSplit(data, splitFactor = 0.7, train = TRUE
)
  test<-TrainTestSplit(data, splitFactor = 0.7, train = FALSE
)
  model <- lm(satisfaction_level ~ ., data = train)
  print(summary(model))
  modelStep<-step(model, direction = "both", trace = 1
)
  print(summary(modelStep))
```

```
  anova(model)
  model$residuals
  #plot residuals
  plot(model$residuals, pch = 16, col = "red"
)

  #The Akaike's information criterion - AIC (Akaike, 1974)
  AIC(model)
  #Bayesian information criterion - BIC (Schwarz, 1978)
  BIC(model)

  confint(model)

  pred <- predict(model, test, interval = "predict"
)
  test_Y <- test[,targetColumnNumber]
  print(data.frame(RMSE = caret::RMSE(pred, test_Y),R2 = caret::R2(pred, test_Y)))
}
```