Predictive Analytics in-class exercise on Cancer Detection

```
# load the required libraries
library("readxl") # used to read excel files
library("dplyr") # used for data munging
library("FNN") # used for knn regression (knn.reg function)
library("caret") # used for various predictive models
library("class") # for using confusion matrix function
library("rpart.plot") # used to plot decision tree
library("rpart")  # used for Regression tree
library("glmnet") # used for Lasso and Ridge regression
library('NeuralNetTools') # used to plot Neural Networks
library("PRROC") # top plot ROC curve
library("ROCR") # top plot lift curve
library("tidyverse")
```

1. Classification

1.1 Data loading and transformation

Please download the Breast Cancer data set from the below mentioned links

Data: https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data (https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.data)

Description: https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.names (https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/wdbc.names)

```
# Load the Breast Cancer data set

cancer_data = read_csv("wdbc.data", col_names =  FALSE)
```

```
##
## -- Column specification --------------------------------------------------------
-------------------------------------------------------------------------------
## cols(
##   .default = col_double(),
##   X2 = col_character()
## )
## i Use `spec()` for the full column specifications.
```

```
# create Y and X data frames
cancer_y = cancer_data %>% pull("X2") %>% as.factor()
# exclude X1 since its a row number
cancer_x = cancer_data %>% select(-c("X1", "X2"))
```

Create a function that normalizes columns since scale for each column might be different.

```
# function to normalize data (0 to 1)
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
}
```

```
# Normalize x variables since they are at different scale
cancer_x_normalized <- as.data.frame(lapply(cancer_x, normalize))
```

Create Training and Testing data sets

```
# 75% of the data is used for training and rest for testing
smp_size <- floor(0.75 * nrow(cancer_x_normalized))

# randomly select row numbers for training data set
train_ind <- sample(seq_len(nrow(cancer_x_normalized)), size = smp_size)

# creating test and training sets for x
cancer_x_train <- cancer_x_normalized[train_ind, ]
cancer_x_test <- cancer_x_normalized[-train_ind, ]

# creating test and training sets for y
cancer_y_train <- cancer_y[train_ind]
cancer_y_test <- cancer_y[-train_ind]

# Create an empty data frame to store results from different models
clf_results <- data.frame(matrix(ncol = 5, nrow = 0))
names(clf_results) <- c("Model", "Accuracy", "Precision", "Recall", "F1")

# Create an empty data frame to store TP, TN, FP and FN values
cost_benefit_df <- data.frame(matrix(ncol = 5, nrow = 0))
names(cost_benefit_df) <- c("Model", "TP", "FN", "FP", "TN")
```

Cross validation

It is a technique to use same training data but some portion of it for training and rest for validation of model. This technique reduces chances of overfitting

Hyperparamter tuning

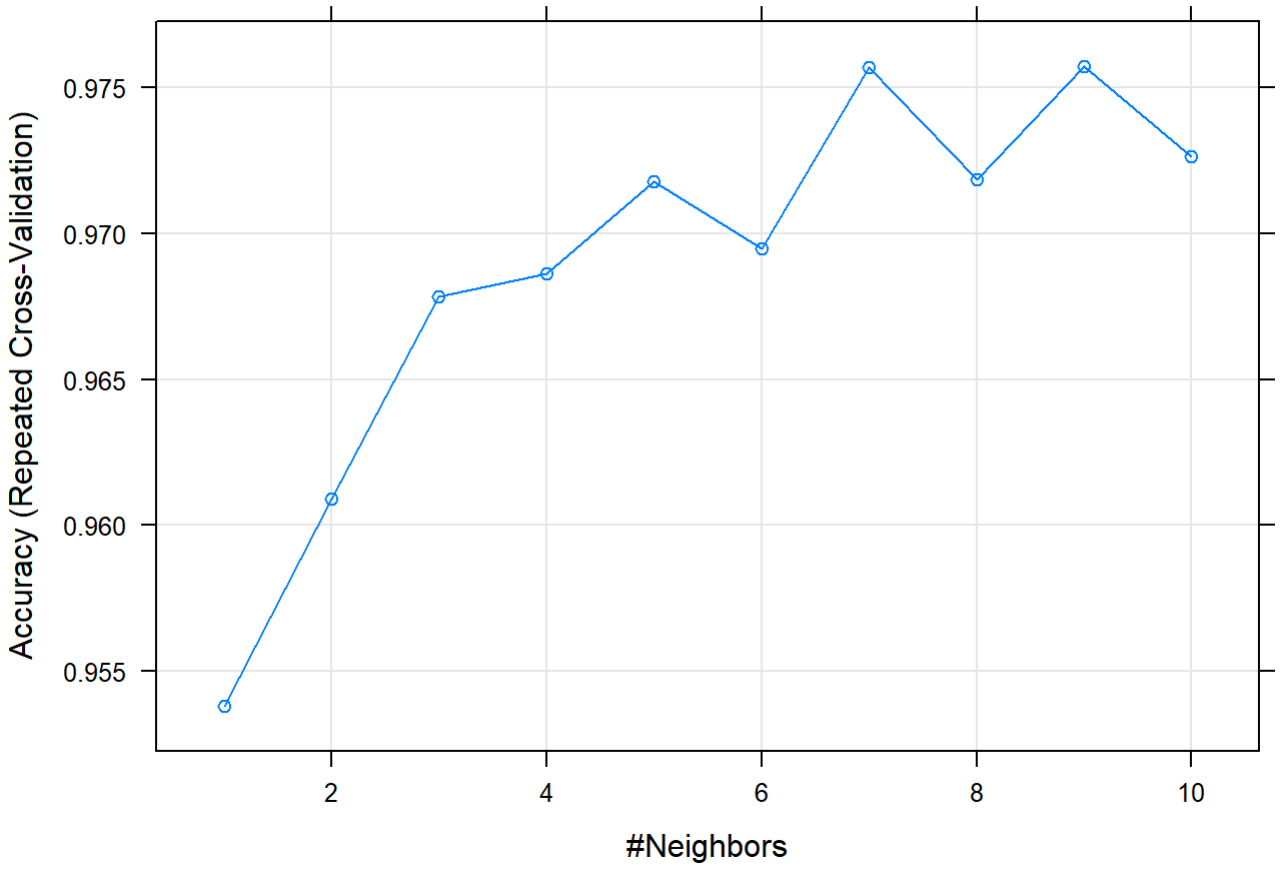
We provide a list of hyperparameters to train the model. This helps in identifying best set of hyperparameters for a given model like Decision tree. **train** function in caret library automatically stores the information of the best model and its hyperparameters.

1.2 KNN Classification

```
# Cross validation
cross_validation <- trainControl(## 10-fold CV
                                 method = "repeatedcv",
                                 number = 10,
                                 ## repeated three times
                                 repeats = 3)
# Hyperparamter tuning
# k = number of nrearest neighbours
Param_Grid <-  expand.grid( k = 1:10)

# fit the model to training data
knn_clf_fit <- train(cancer_x_train,
                     cancer_y_train,
                     method = "knn",
                     tuneGrid = Param_Grid,
                     trControl = cross_validation )

# check the accuracy for different models
knn_clf_fit
```

```
## k-Nearest Neighbors
##
## 426 samples
##  30 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 384, 383, 383, 383, 384, 383, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9537819  0.9000611
##    2  0.9609072  0.9150567
##    3  0.9678470  0.9301576
##    4  0.9686222  0.9321461
##    5  0.9717968  0.9390653
##    6  0.9694712  0.9339730
##    7  0.9757097  0.9474158
##    8  0.9718522  0.9392110
##    9  0.9757466  0.9475513
##   10  0.9726458  0.9407528
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```
# Plot accuracies for different k values
plot(knn_clf_fit)
```




```
# print the best model
print(knn_clf_fit$finalModel)
```

```
## 9-nearest neighbor model
## Training set outcome distribution:
##
##   B   M
## 269 157
```

```
# Predict on test data
knnPredict <- predict(knn_clf_fit, newdata = cancer_x_test)
```

```
# Print Confusion matrix, Accuracy, Sensitivity etc
confusionMatrix(knnPredict, cancer_y_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 88  5
##          M  0 50
##
##                Accuracy : 0.965
##                  95% CI : (0.9203, 0.9886)
##     No Information Rate : 0.6154
##     P-Value [Acc > NIR] : < 2e-16
##
##                   Kappa : 0.9249
##
##  Mcnemar's Test P-Value : 0.07364
##
##             Sensitivity : 1.0000
##             Specificity : 0.9091
##          Pos Pred Value : 0.9462
##          Neg Pred Value : 1.0000
##              Prevalence : 0.6154
##          Detection Rate : 0.6154
##    Detection Prevalence : 0.6503
##       Balanced Accuracy : 0.9545
##
##        'Positive' Class : B
##
```

```
# Add results into clf_results dataframe
x1 <- confusionMatrix(knnPredict, cancer_y_test)[["overall"]]
y1 <- confusionMatrix(knnPredict, cancer_y_test)[["byClass"]]

clf_results[nrow(clf_results) + 1,] <-  list(Model = "KNN",
                                             Accuracy = round (x1[["Accuracy"]],3),
                                            Precision = round (y1[["Precision"]],3),
                                            Recall = round (y1[["Recall"]],3),
                                            F1 = round (y1[["F1"]],3))
# Print Accuracy and F1 score

cat("Accuarcy is ", round(x1[["Accuracy"]],3), "and F1 is ", round (y1[["F1"]],3)  )
```

```
## Accuarcy is  0.965 and F1 is  0.972
```

```
# Add results into cost_benefit_df dataframe for cost benefit analysis
a1 <- confusionMatrix(knnPredict, cancer_y_test)

cost_benefit_df[nrow(cost_benefit_df) + 1,] <-  list(Model = "KNN",
                                             TP = a1[["table"]][1],
                                             FN = a1[["table"]][2],
                                             FP = a1[["table"]][3],
                                             TN = a1[["table"]][4])
```

1.3 Decision Tree Classification

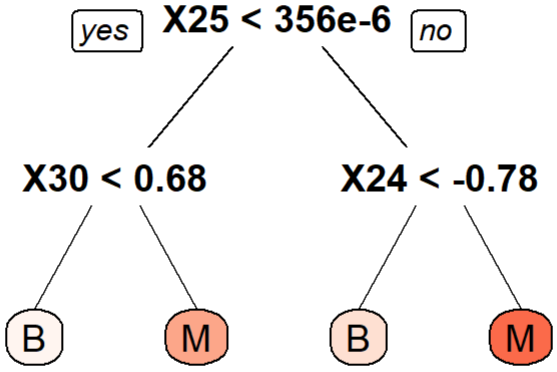
```
# Cross validation
cross_validation <- trainControl(## 10-fold CV
                                 method = "repeatedcv",
                                 number = 10,
                                 ## repeated three times
                                 repeats = 3)
# Hyperparamter tuning
# maxdepth =  the maximum depth of the tree that will be created or
# the length of the longest path from the tree root to a leaf.

Param_Grid <-  expand.grid(maxdepth = 2:10)

dtree_fit <- train(cancer_x_train,
                   cancer_y_train,
                   method = "rpart2",
                   # split - criteria to split nodes
                   parms = list(split = "gini"),
                  tuneGrid = Param_Grid,
                   trControl = cross_validation,
                  # preProc -  perform listed pre-processing to predictor dataframe
                   preProc = c("center", "scale"))

# check the accuracy for different models
dtree_fit
```

```
## CART
##
## 426 samples
##  30 predictor
##   2 classes: 'B', 'M'
##
## Pre-processing: centered (30), scaled (30)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 383, 384, 384, 383, 385, 383, ...
## Resampling results across tuning parameters:
##
##   maxdepth  Accuracy   Kappa
##    2        0.9233031  0.8374402
##    3        0.9240598  0.8376756
##    4        0.9232855  0.8361082
##    5        0.9232855  0.8361082
##    6        0.9232855  0.8361082
##    7        0.9232855  0.8361082
##    8        0.9232855  0.8361082
##    9        0.9232855  0.8361082
##   10        0.9232855  0.8361082
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was maxdepth = 3.
```

```
# print the final model
dtree_fit$finalModel
```

```
## n= 426
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 426 157 B (0.63145540 0.36854460)
##   2) X25< 0.0003560704 266  13 B (0.95112782 0.04887218)
##     4) X30< 0.6795072 255   4 B (0.98431373 0.01568627) *
##     5) X30>=0.6795072 11   2 M (0.18181818 0.81818182) *
##   3) X25>=0.0003560704 160  16 M (0.10000000 0.90000000)
##     6) X24< -0.7812289 17   6 B (0.64705882 0.35294118) *
##     7) X24>=-0.7812289 143   5 M (0.03496503 0.96503497) *
```

```
# Plot decision tree
prp(dtree_fit$finalModel, box.palette = "Reds", tweak = 1.2)
```




```
# Predict on test data
dtree_predict <- predict(dtree_fit, newdata = cancer_x_test)
```

```
# Print Confusion matrix, Accuarcy, Sensitivity etc
confusionMatrix(dtree_predict,  cancer_y_test )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 84  9
##          M  4 46
##
##                Accuracy : 0.9091
##                  95% CI : (0.8496, 0.9507)
##     No Information Rate : 0.6154
##     P-Value [Acc > NIR] : 1.775e-15
##
##                   Kappa : 0.8046
##
##  Mcnemar's Test P-Value : 0.2673
##
##             Sensitivity : 0.9545
##             Specificity : 0.8364
##          Pos Pred Value : 0.9032
##          Neg Pred Value : 0.9200
##              Prevalence : 0.6154
##          Detection Rate : 0.5874
##    Detection Prevalence : 0.6503
##       Balanced Accuracy : 0.8955
##
##        'Positive' Class : B
##
```

```
# Add results into clf_results dataframe
x2 <- confusionMatrix(dtree_predict,  cancer_y_test )[["overall"]]
y2 <- confusionMatrix(dtree_predict,  cancer_y_test )[["byClass"]]

clf_results[nrow(clf_results) + 1,] <-  list(Model = "Decision Tree",
                                             Accuracy = round (x2[["Accuracy"]],3),
                                            Precision = round (y2[["Precision"]],3),
                                            Recall = round (y2[["Recall"]],3),
                                            F1 = round (y2[["F1"]],3))

# Print Accuracy and F1 score

cat("Accuarcy is ", round(x2[["Accuracy"]],3), "and F1 is ", round (y2[["F1"]],3)  )
```

```
## Accuarcy is  0.909 and F1 is  0.928
```

```
# Add results into cost_benefit_df dataframe for cost benefit analysis
a2 <- confusionMatrix(dtree_predict,  cancer_y_test )

cost_benefit_df[nrow(cost_benefit_df) + 1,] <-  list(Model = "Decision Tree",
                                                TP = a2[["table"]][1],
                                                FN = a2[["table"]][2],
                                                FP = a2[["table"]][3],
                                                TN = a2[["table"]][4])
```

1.4 Logistic regression

Convert categorical outcome into numerical.

```
cancer_y_train_l <- ifelse(cancer_y_train =="B", 1, 0)
cancer_y_test_l <- ifelse(cancer_y_test =="B", 1, 0)
```

```
glm_fit <- train(cancer_x_train,
                 cancer_y_train_l,
                 method = "glm",
                 family = "binomial",
                 preProc = c("center", "scale"))
```

```
# Predict on test data
glm_predict <- predict(glm_fit, newdata = cancer_x_test)
```

convert probability outcome into categorical outcome

```
y_pred_num <- ifelse(glm_predict > 0.5, 1, 0)
```

```
# Print Confusion matrix, Accuarcy, Sensitivity etc
confusionMatrix(as.factor(y_pred_num), as.factor(cancer_y_test_l), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  0  1
##          0 50  5
##          1  5 83
##
##                Accuracy : 0.9301
##                  95% CI : (0.8752, 0.966)
##     No Information Rate : 0.6154
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8523
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9432
##             Specificity : 0.9091
##          Pos Pred Value : 0.9432
##          Neg Pred Value : 0.9091
##              Prevalence : 0.6154
##          Detection Rate : 0.5804
##    Detection Prevalence : 0.6154
##       Balanced Accuracy : 0.9261
##
##        'Positive' Class : 1
##
```

```
# Add results into clf_results dataframe
x3 <- confusionMatrix(as.factor(y_pred_num), as.factor(cancer_y_test_1), positive = "1")[["overa
ll"]]
y3 <- confusionMatrix(as.factor(y_pred_num), as.factor(cancer_y_test_1),positive = "1")[["byClas
s"]]

clf_results[nrow(clf_results) + 1,] <-  list(Model = "Logistic Regression",
                                             Accuracy = round (x3[["Accuracy"]],3),
                                            Precision = round (y3[["Precision"]],3),
                                            Recall = round (y3[["Recall"]],3),
                                            F1 = round (y3[["F1"]],3))

# Print Accuracy and F1 score
cat("Accuarcy is ", round(x3[["Accuracy"]],3), "and F1 is ", round (y3[["F1"]],3)  )
```

```
## Accuarcy is  0.93 and F1 is  0.943
```

```
# Add results into cost_benefit_df dataframe for cost benefit analysis
a3 <- confusionMatrix(as.factor(y_pred_num), as.factor(cancer_y_test_l))

#be careful about accurately pickign up the TP, FN, FP and TN
cost_benefit_df[nrow(cost_benefit_df) + 1,] <-  list(Model = "Logistic Regression",
                                                     TP = a3[["table"]][4],
                                                     FN = a3[["table"]][3],
                                                     FP = a3[["table"]][2],
                                                     TN = a3[["table"]][1])
```

1.5 XGBoost classification

```
XG_clf_fit <- train(cancer_x_train,
                    cancer_y_train,
                    method = "xgbTree",
                    preProc = c("center", "scale"))
```

```
# print the final model
XG_clf_fit$finalModel
```

```
## ##### xgb.Booster
## raw: 27.6 Kb
## call:
##   xgboost::xgb.train(params = list(eta = param$eta, max_depth = param$max_depth,
##     gamma = param$gamma, colsample_bytree = param$colsample_bytree,
##     min_child_weight = param$min_child_weight, subsample = param$subsample),
##     data = x, nrounds = param$nrounds, objective = "binary:logistic")
## params (as set within xgb.train):
##   eta = "0.4", max_depth = "1", gamma = "0", colsample_bytree = "0.6", min_child_weight =
"1", subsample = "0.5", objective = "binary:logistic", validate_parameters = "TRUE"
## xgb.attributes:
##   niter
## callbacks:
##   cb.print.evaluation(period = print_every_n)
## # of features: 30
## niter: 100
## nfeatures : 30
## xNames : X3 X4 X5 X6 X7 X8 X9 X10 X11 X12 X13 X14 X15 X16 X17 X18 X19 X20 X21 X22 X23 X24 X25
X26 X27 X28 X29 X30 X31 X32
## problemType : Classification
## tuneValue :
##     nrounds max_depth eta gamma colsample_bytree min_child_weight subsample
## 56     100         1 0.4     0              0.6                1       0.5
## obsLevels : B M
## param :
##  list()
```

```
# Predict on test data
XG_clf_predict <- predict(XG_clf_fit,cancer_x_test)
```

```
# Print Confusion matrix, Accuracy, Sensitivity etc
confusionMatrix(XG_clf_predict,  cancer_y_test )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 84  4
##          M  4 51
##
##                Accuracy : 0.9441
##                  95% CI : (0.8927, 0.9755)
##     No Information Rate : 0.6154
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.8818
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9545
##             Specificity : 0.9273
##          Pos Pred Value : 0.9545
##          Neg Pred Value : 0.9273
##              Prevalence : 0.6154
##          Detection Rate : 0.5874
##    Detection Prevalence : 0.6154
##       Balanced Accuracy : 0.9409
##
##        'Positive' Class : B
##
```

```
# Add results into clf_results dataframe
x4 <- confusionMatrix(XG_clf_predict,  cancer_y_test )[["overall"]]
y4 <- confusionMatrix(XG_clf_predict,  cancer_y_test )[["byClass"]]

clf_results[nrow(clf_results) + 1,] <-  list(Model = "XG Boost",
                                             Accuracy = round (x4[["Accuracy"]],3),
                                            Precision = round (y4[["Precision"]],3),
                                            Recall = round (y4[["Recall"]],3),
                                            F1 = round (y4[["F1"]],3))

# Print Accuracy and F1 score
cat("Accuarcy is ", round(x4[["Accuracy"]],3), "and F1 is ", round (y4[["F1"]],3)  )
```

```
## Accuarcy is  0.944 and F1 is  0.955
```

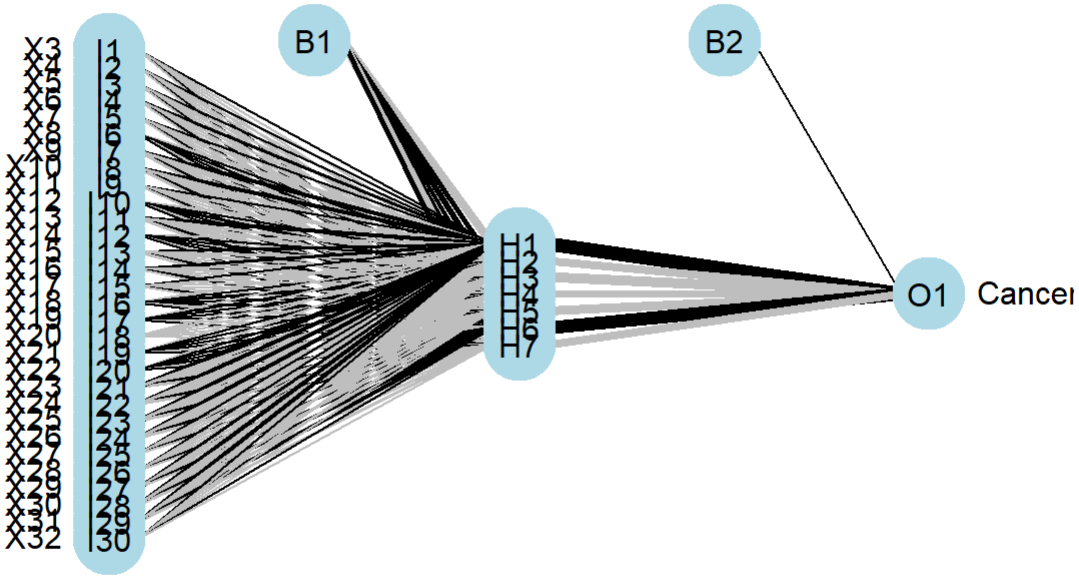
```
# Add results into cost_benefit_df dataframe for cost benefit analysis
a4 <- confusionMatrix(XG_clf_predict,  cancer_y_test )

cost_benefit_df[nrow(cost_benefit_df) + 1,] <-  list(Model = "XG Boost",
                                                     TP = a4[["table"]][1],
                                                     FN = a4[["table"]][2],
                                                     FP = a4[["table"]][3],
                                                     TN = a4[["table"]][4])
```

1.6 Neural Network classification

```
# Try different combinations of parameters like
# decay (prevents the weights from growing too large,)
# and size of Hidden Layers
my.grid <- expand.grid(.decay = c(0.5, 0.1), .size = c(5, 7))

# stepmax is maximum steps for the training of the neural network
# threshold is set to 0.01, meaning that if the change in error during an iteration is
# less than 1%, then no further optimization will be carried out by the model
nn_clf_fit <- train(cancer_x_train,
                    cancer_y_train,
                    method = "nnet",
                    trace = F,
                    tuneGrid = my.grid,
                    linout = 0,
                    stepmax = 100,
                    threshold = 0.01 )
print(nn_clf_fit)
```

```
## Neural Network
##
## 426 samples
##  30 predictor
##   2 classes: 'B', 'M'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 426, 426, 426, 426, 426, 426, ...
## Resampling results across tuning parameters:
##
##   decay  size  Accuracy   Kappa
##   0.1    5     0.9725320  0.9403657
##   0.1    7     0.9727969  0.9409384
##   0.5    5     0.9699752  0.9345933
##   0.5    7     0.9697218  0.9340257
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 7 and decay = 0.1.
```

```
# Plot Neural Network
plotnet(nn_clf_fit$finalModel, y_names = "Cancer Type")
```




```
# Predict on test data
nn_clf_predict <- predict(nn_clf_fit,cancer_x_test)
```

```
# Print Confusion matrix, Accuarcy, Sensitivity etc
confusionMatrix(nn_clf_predict,  cancer_y_test)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  B  M
##          B 86  3
##          M  2 52
##
##                Accuracy : 0.965
##                  95% CI : (0.9203, 0.9886)
##     No Information Rate : 0.6154
##     P-Value [Acc > NIR] : <2e-16
##
##                   Kappa : 0.9259
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.9773
##             Specificity : 0.9455
##          Pos Pred Value : 0.9663
##          Neg Pred Value : 0.9630
##              Prevalence : 0.6154
##          Detection Rate : 0.6014
##    Detection Prevalence : 0.6224
##       Balanced Accuracy : 0.9614
##
##        'Positive' Class : B
##
```

```
# Add results into clf_results dataframe
x5 <- confusionMatrix(nn_clf_predict,  cancer_y_test)[["overall"]]
y5 <- confusionMatrix(nn_clf_predict,  cancer_y_test)[["byClass"]]

clf_results[nrow(clf_results) + 1,] <-  list(Model = "Neural Network",
                                             Accuracy = round (x5[["Accuracy"]],3),
                                            Precision = round (y5[["Precision"]],3),
                                            Recall = round (y5[["Recall"]],3),
                                            F1 = round (y5[["F1"]],3))

# Print Accuracy and F1 score
cat("Accuarcy is ", round(x5[["Accuracy"]],3), "and F1 is ", round (y5[["F1"]],3)  )
```

```
## Accuarcy is  0.965 and F1 is  0.972
```

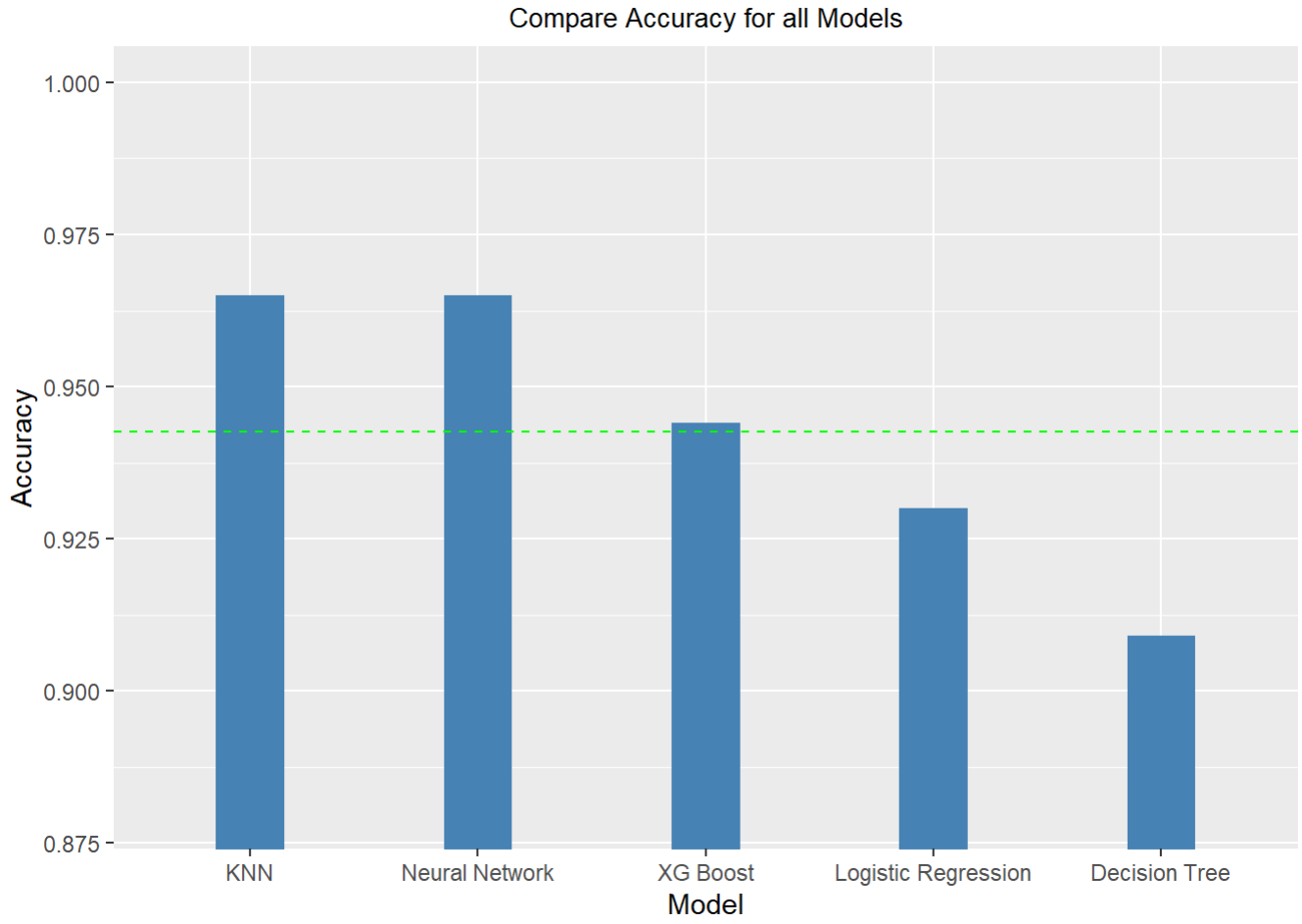
```
# Add results into cost_benefit_df dataframe for cost benefit analysis
a5 <- confusionMatrix(nn_clf_predict,  cancer_y_test)

cost_benefit_df[nrow(cost_benefit_df) + 1,] <-  list(Model = "Neural Network",
                                             TP = a5[["table"]][1],
                                             FN = a5[["table"]][2],
                                             FP = a5[["table"]][3],
                                             TN = a5[["table"]][4])
```

Compare Accuracy for all Classification models

```
print(clf_results)
```

```
##                 Model Accuracy Precision Recall    F1
## 1                 KNN    0.965     0.946  1.000 0.972
## 2       Decision Tree    0.909     0.903  0.955 0.928
## 3 Logistic Regression    0.930     0.943  0.943 0.943
## 4             XG Boost    0.944     0.955  0.955 0.955
## 5      Neural Network    0.965     0.966  0.977 0.972
```

```
# Plot accuracy for all the Classification Models

ggplot(clf_results %>% arrange(desc(Accuracy)) %>%
       mutate(Model=factor(Model, levels=Model) ),
       aes(x = Model, y = Accuracy)) +
  geom_bar(stat = "identity" , width=0.3, fill="steelblue") +
  coord_cartesian(ylim = c(0.88, 1)) +
  geom_hline(aes(yintercept = mean(Accuracy)),
             colour = "green",linetype="dashed") +
  ggtitle("Compare Accuracy for all Models") +
  theme(plot.title = element_text(color="black", size=10, hjust = 0.5))
```



1.7 Cost Benefit analysis

A model with high accuracy need not be the most profitable one. We can assign different costs to True Positive (TP), True Negative (TN), False Positive (FP), False Negative (FN) and evaluate each model and figure out which one is the most profitable model.

For this exercise lets assume that:

benefit_TP = benefit for correctly predicting the cell type to be benign = $1000 benefit_TN = benefit for correctly predicting the cell type to be malignant = $4000 (so that you have a shot at curing it) cost_FP = cost of incorrectly predicting a cancer cell as B= $5000 as it could lead to no further screening and eventual death cost_FN= cost of incorrectly predicting a cancer cell as M= $200 (cost of additional test that would clarify the situation

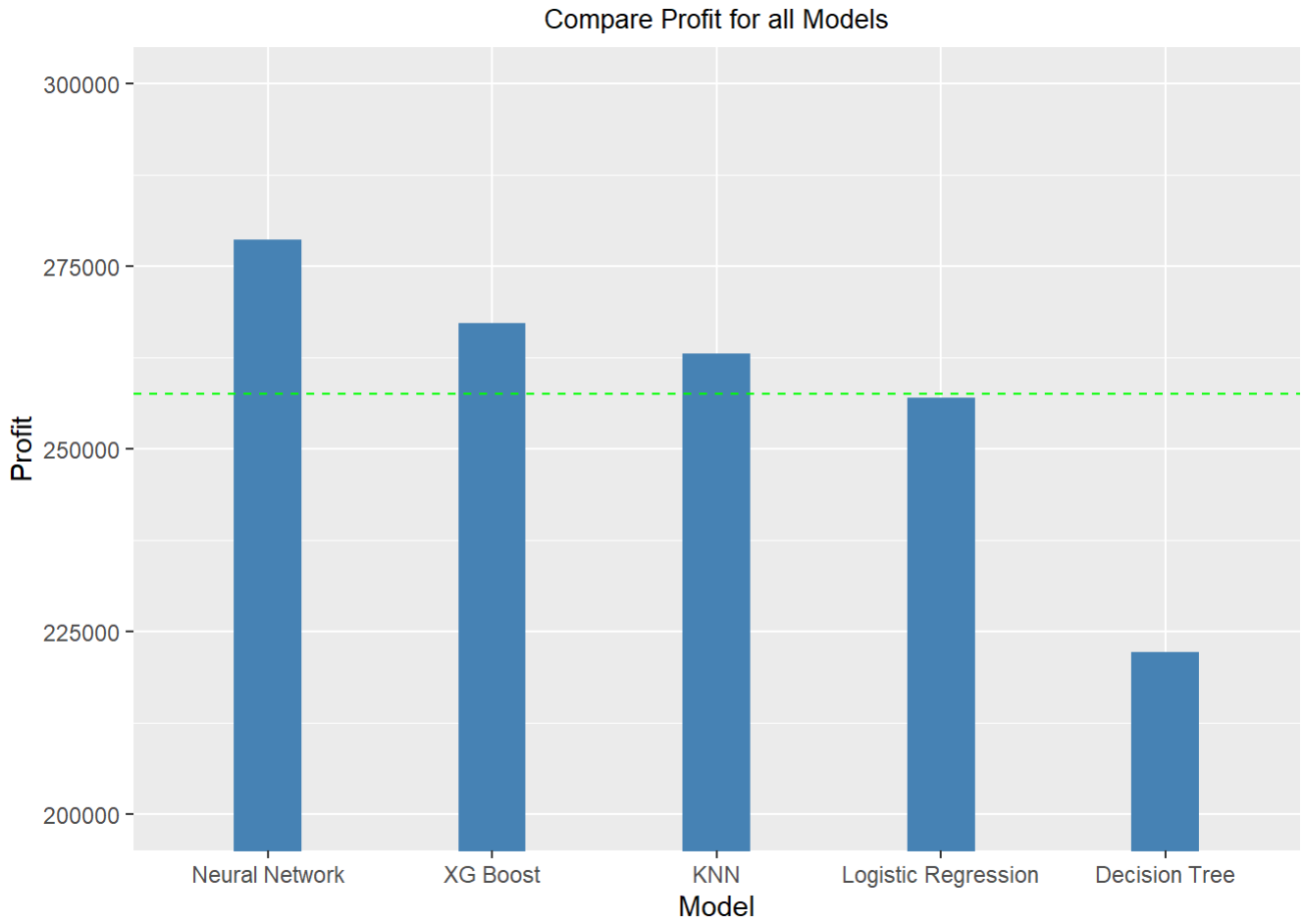
```
benefit_TP = 1000
benefit_TN = 4000
cost_FN = -200
cost_FP = -5000

cost_benefit_df <- cost_benefit_df %>%
                    mutate(Profit = (benefit_TP * TP) + (benefit_TN * TN) +
                                    (cost_FP * FP) + (cost_FN * FN))
```

Compare Profit for all Classification models

```
print(cost_benefit_df)
```

```
##                 Model TP FN FP TN Profit
## 1                 KNN 88  0  5 50 263000
## 2       Decision Tree 84  4  9 46 222200
## 3 Logistic Regression 83  5  5 50 257000
## 4            XG Boost 84  4  4 51 267200
## 5      Neural Network 86  2  3 52 278600
```

```
# Plot Profit for all the Classification Models

ggplot(cost_benefit_df %>% arrange(desc(Profit)) %>%
       mutate(Model=factor(Model, levels=Model) ),
       aes(x = Model, y = Profit)) +
  geom_bar(stat = "identity" , width=0.3, fill="steelblue") +
  coord_cartesian(ylim = c(200000, 300000)) +
  geom_hline(aes(yintercept = mean(Profit)),
             colour = "green",linetype="dashed") +
  ggtitle("Compare Profit for all Models") +
  theme(plot.title = element_text(color="black", size=10, hjust = 0.5))
```



1.8 ROC and Lift curves for all models

ROC curve - It is a performance measurement for classification problem at various thresholds settings. It tells how much a model is capable of distinguishing between classes.

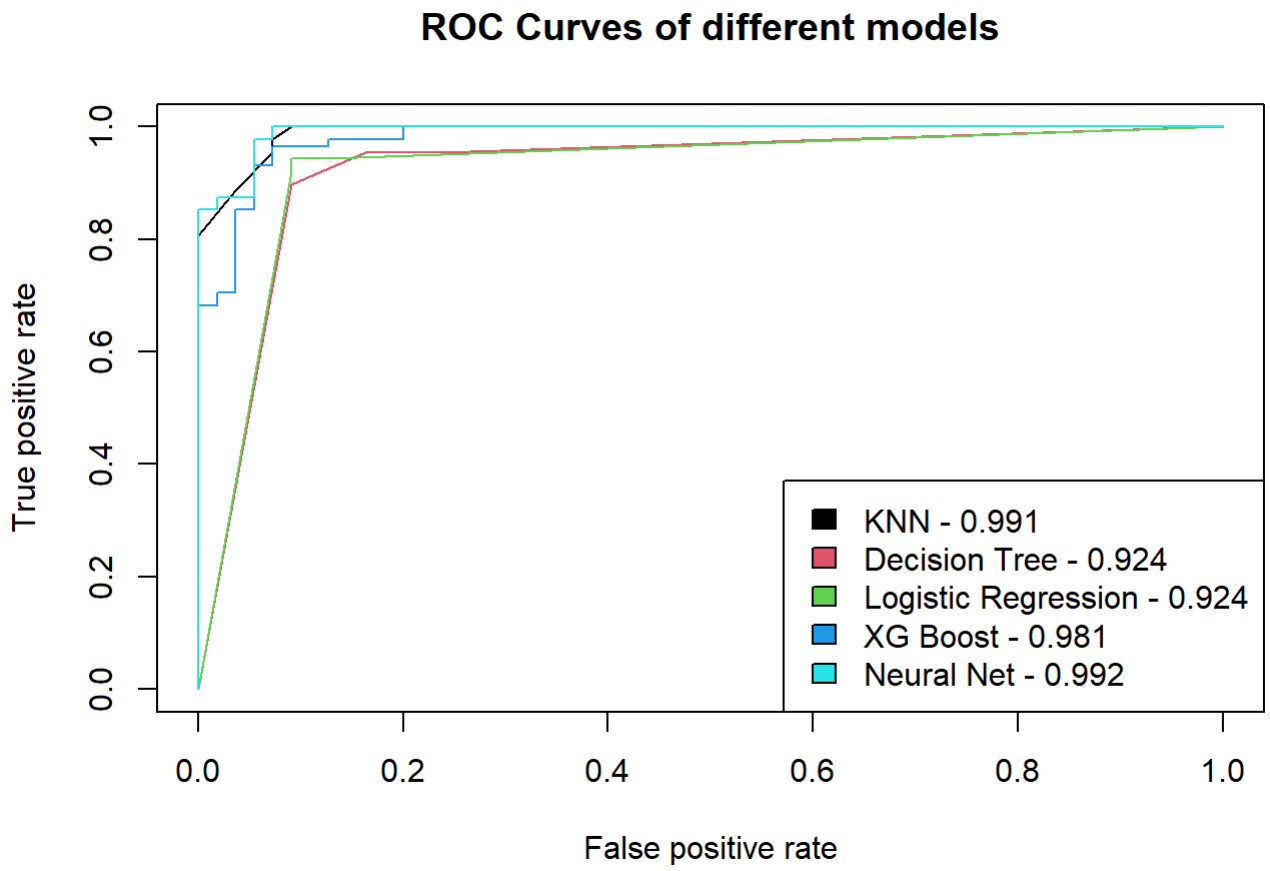
Y axis - True Positive rate or Sensitivity = (TP / TP + FN)

X axis - False Positive rate or (1 - specificity) = (FP / TN + FP)

AUC - Area under ROC curve. Higher the AUC, better the model is at predicting 0s as 0s and 1s as 1s.

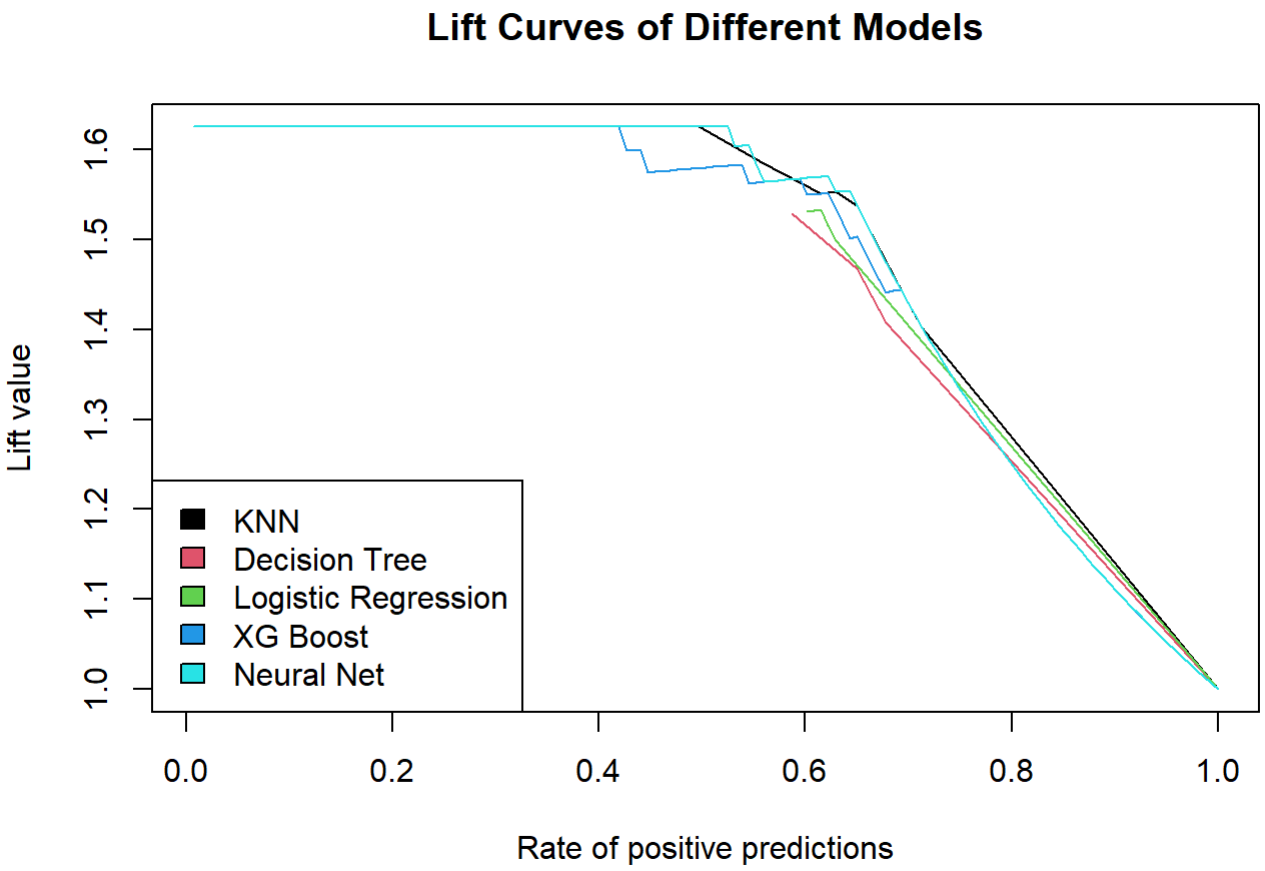
Lets Plot ROC curves for all the Models. The more “up and to the left” the ROC curve of a model is, the better the model. Also, higher the Area under curve, the better the model.

```
# Predict probabilities of each model to plot ROC curve
knnPredict_prob <- predict(knn_clf_fit, newdata = cancer_x_test, type = "prob")
dtree_prob <- predict(dtree_fit, newdata = cancer_x_test, type = "prob")
XG_boost_prob <- predict(XG_clf_fit, newdata = cancer_x_test, type = "prob")
nn_clf_prob <- predict(nn_clf_fit, newdata = cancer_x_test, type = "prob")

# List of predictions
preds_list <- list(knnPredict_prob[,1], dtree_prob[,1],
                   glm_predict, XG_boost_prob[,1], nn_clf_prob[,1] )

# List of actual values (same for all)
m <- length(preds_list)
actuals_list <- rep(list(cancer_y_test_l), m)

# Plot the ROC curves
pred <- prediction(preds_list, actuals_list)
rocs <- performance(pred, "tpr", "fpr")

# calculate AUC for all models
AUC_models <- performance(pred, "auc")
auc_knn = round(AUC_models@y.values[[1]], 3)
auc_dt = round(AUC_models@y.values[[2]], 3)
auc_lr = round(AUC_models@y.values[[3]], 3)
auc_xg = round(AUC_models@y.values[[4]], 3)
auc_nn = round(AUC_models@y.values[[5]], 3)

# Plot the ROC curves
plot(rocs, col = as.list(1:m), main = "ROC Curves of different models")
legend(x = "bottomright",
       legend = c( paste0("KNN - ", auc_knn),
                   paste0("Decision Tree - ", auc_dt),
                   paste0("Logistic Regression - ", auc_lr),
                   paste0("XG Boost - ", auc_xg),
                   paste0("Neural Net - ", auc_nn)), fill = 1:m)
```



Lift curve - Lift is a measure of the effectiveness of a predictive model calculated as the ratio between the results obtained with and without the predictive model. The lift chart shows how much more likely we are to predict the correct outcome than a random guess.

```
lifts <- performance(pred, "lift", "rpp")

# Plot the Lift curves
plot(lifts, col = as.list(1:m), main = "Lift Curves of Different Models")
legend(x = "bottomleft",
       legend = c( "KNN",
                   "Decision Tree",
                   "Logistic Regression",
                   "XG Boost",
                   "Neural Net"), fill = 1:m)
```


Lift Curves of Different Models
Lift value
1.0
1.1
1.2
1.3
1.4
1.5
1.6
KNN
Decision Tree
Logistic Regression
XG Boost
Neural Net
0.0
0.2
0.4
0.6
0.8
1.0
Rate of positive predictions