Universal Bank - Promotion Targeting

10/15/2020

```r
# 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("skimr")
```

1. Classification

1.1 Data loading and transformation

The Excel file has three sheets. There is the data dictionary, 2000 rows of past data to train a model, and 500 rows of new (unlabeled) data to identify which top 50 people to target.

```r
# Load the Universal Bank  data set
# skip -> zip code column is skipped. We skipped it because it is not a number and there are too many factors (too many zi codes)
bank1 <- read_excel("CL-bank-training-testingPredicting.xlsx",
                    sheet = "historicalDataFromMktingPromo",
                    col_types = c("numeric", "numeric", "numeric",
                                  "numeric", "skip", "numeric", "numeric",
                                  "numeric", "numeric", "numeric",
                                  "numeric", "numeric", "numeric",
                                  "text"))

skim(bank1)
```

Data summary

Name	bank1
Number of rows	2001
Number of columns	13
_______________________	
Column type frequency:	
character	1
numeric	12
________________________	
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
PersonalLoan	0	1	1	1	0	2	0

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
ID	0	1	1001.00	577.78	1	501.0	1001.0	1501.0	2001	▇▇▇▇▇
Age	0	1	45.40	11.47	23	35.0	45.0	55.0	67	▃▇▇▇▅
Experience	0	1	20.18	11.47	-2	10.0	20.0	30.0	42	▃▇▇▇▅
Income	0	1	74.49	46.48	8	39.0	64.0	99.0	205	▇▇▃▂▂
Family	0	1	2.41	1.16	1	1.0	2.0	4.0	4	▇▅▁▅▆
CCAvg	0	1	1.96	1.79	0	0.7	1.5	2.5	10	▇▃▁▁▁
Education	0	1	1.87	0.84	1	1.0	2.0	3.0	3	▇▁▅▁▆
Mortgage	0	1	56.78	99.17	0	0.0	0.0	104.0	617	▇▂▁▁▁
SecuritiesAccount	0	1	0.11	0.32	0	0.0	0.0	0.0	1	▇▁▁▁▁
CDAccount	0	1	0.06	0.24	0	0.0	0.0	0.0	1	▇▁▁▁▁
Online	0	1	0.61	0.49	0	0.0	1.0	1.0	1	▅▁▁▁▇
CreditCard	0	1	0.30	0.46	0	0.0	0.0	1.0	1	▇▁▁▁▃

```
#notice, we imported the Outcome "PersonalLoan" column as text.
#this will make it easier to convert the y column to be of the type factor, which tells R to do binary classifaction
#you have to be careful if you import an outcome that is 0 or 1 and you intend it to be factor
bank1$PersonalLoan <- as.factor(bank1$PersonalLoan)

# create Y and X data frames
#we will need the y column as a vector (X to be a dataframe)
# dplyr allows us to do this by using 'pull' instead of select
bank1_y = bank1 %>% pull("PersonalLoan")

# exclude column 1 since its a ID
bank1_x = bank1 %>% select(-c("ID", "PersonalLoan"))
```

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
bank1_x_normalized <- as.data.frame(lapply(bank1_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(bank1_x_normalized))

# randomly select row numbers for training data set
set.seed(12345)
train_ind <- sample(seq_len(nrow(bank1_x_normalized)), size = smp_size)

# creating test and training sets for x
bank1_x_train <- bank1_x_normalized[train_ind, ]
bank1_x_test <- bank1_x_normalized[-train_ind, ]

# creating test and training sets for y
bank1_y_train <- bank1_y[train_ind]
bank1_y_test <- bank1_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 nearest neighbors
#try k from 1 to 15
Param_Grid <-  expand.grid( k = 1:15)

#for every k from 1: 15 it will do 10 fold cross validation, 3 times

# fit the model to training data
knn_clf_fit <- train(bank1_x_train,
                     bank1_y_train,
                     method = "knn",
                     tuneGrid = Param_Grid,
                     trControl = cross_validation )



#you can see the sampling process
knn_clf_fit$resample %>% arrange(Resample)
```

Accuracy <dbl>	Kappa <dbl>	Resample <chr>

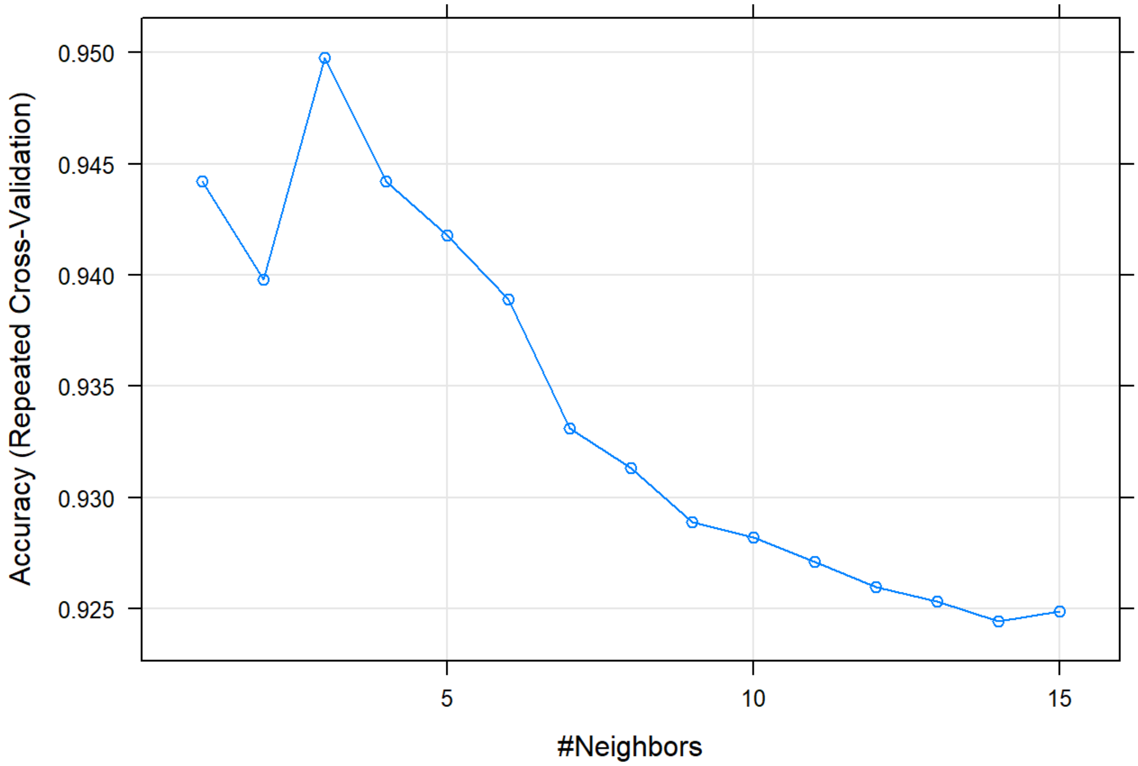
Accuracy <dbl>	Kappa <dbl>	Resample <chr>
0.9333333	0.6069182	Fold01.Rep1
0.9533333	0.7169811	Fold01.Rep2
0.9600000	0.7647674	Fold01.Rep3
0.9466667	0.6661102	Fold02.Rep1
0.9533333	0.6967071	Fold02.Rep2
0.9530201	0.6965377	Fold02.Rep3
0.9536424	0.7171528	Fold03.Rep1
0.9533333	0.7336377	Fold03.Rep2
0.9602649	0.7783757	Fold03.Rep3
0.9403974	0.6363393	Fold04.Rep1

1-10 of 30 rows Previous **1** 2 3 Next

```
# check the accuracy for different models
knn_clf_fit
```

```
## k-Nearest Neighbors
##
## 1500 samples
##   11 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1350, 1350, 1349, 1349, 1350, 1350, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9442227  0.6865960
##    2  0.9397827  0.6528644
##    3  0.9497814  0.6936494
##    4  0.9442287  0.6566880
##    5  0.9417812  0.6360917
##    6  0.9388938  0.6179154
##    7  0.9331144  0.5711722
##    8  0.9313352  0.5550822
##    9  0.9288847  0.5412887
##   10  0.9282136  0.5298049
##   11  0.9271040  0.5182183
##   12  0.9259810  0.5043349
##   13  0.9253232  0.4924183
##   14  0.9244313  0.4819031
##   15  0.9248757  0.4823591
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 3.
```

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




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

```
## 3-nearest neighbor model
## Training set outcome distribution:
##
##    0    1
## 1335  165
```

```
# Predict on test data
knnPredict <- predict(knn_clf_fit, newdata = bank1_x_test)
```

```
# Print Confusion matrix, Accuarcy, Sensitivity etc
confusionMatrix(knnPredict, bank1_y_test, positive="1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 452  19
##          1   2  28
##
##                Accuracy : 0.9581
##                  95% CI : (0.9366, 0.9739)
##     No Information Rate : 0.9062
##     P-Value [Acc > NIR] : 8.195e-06
##
##                   Kappa : 0.7058
##
##  Mcnemar's Test P-Value : 0.0004803
##
##             Sensitivity : 0.59574
##             Specificity : 0.99559
##          Pos Pred Value : 0.93333
##          Neg Pred Value : 0.95966
##              Prevalence : 0.09381
##          Detection Rate : 0.05589
##    Detection Prevalence : 0.05988
##       Balanced Accuracy : 0.79567
##
##        'Positive' Class : 1
##
```

```
# Add results into clf_results dataframe
x1 <- confusionMatrix(knnPredict, bank1_y_test, positive="1")[["overall"]]
y1 <- confusionMatrix(knnPredict, bank1_y_test, positive="1")[["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  0.958 and F1 is  0.727
```

```
# Add results into cost_benefit_df dataframe for cost benefit analysis
a1 <- confusionMatrix(knnPredict, bank1_y_test)

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

1.3 Decision Tree Classification

```r
# 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.


#lets see what hyper parameters are needed /available to tune the decision tree using rpart2 package
modelLookup("rpart2")
```

	model	parameter	label	forReg	forClass	probModel
	<chr>	<chr>	<chr>	<lgl>	<lgl>	<lgl>
1	rpart2	maxdepth	Max Tree Depth	TRUE	TRUE	TRUE

1 row

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


dtree_fit <- train(bank1_x_train,
                   bank1_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
##
## 1500 samples
##   11 predictor
##    2 classes: '0', '1'
##
## Pre-processing: centered (11), scaled (11)
## Resampling: Cross-Validated (10 fold, repeated 3 times)
## Summary of sample sizes: 1350, 1350, 1350, 1350, 1349, 1350, ...
## Resampling results across tuning parameters:
##
##   maxdepth  Accuracy   Kappa
##    2        0.9564542  0.7467948
##    3        0.9711151  0.8444781
##    4        0.9773404  0.8758976
##    5        0.9775626  0.8772363
##    6        0.9768929  0.8759013
##    7        0.9768929  0.8759013
##    8        0.9768929  0.8762019
##    9        0.9768929  0.8762019
##   10        0.9768929  0.8762019
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was maxdepth = 5.
```

```
dtree_fit$resample
```

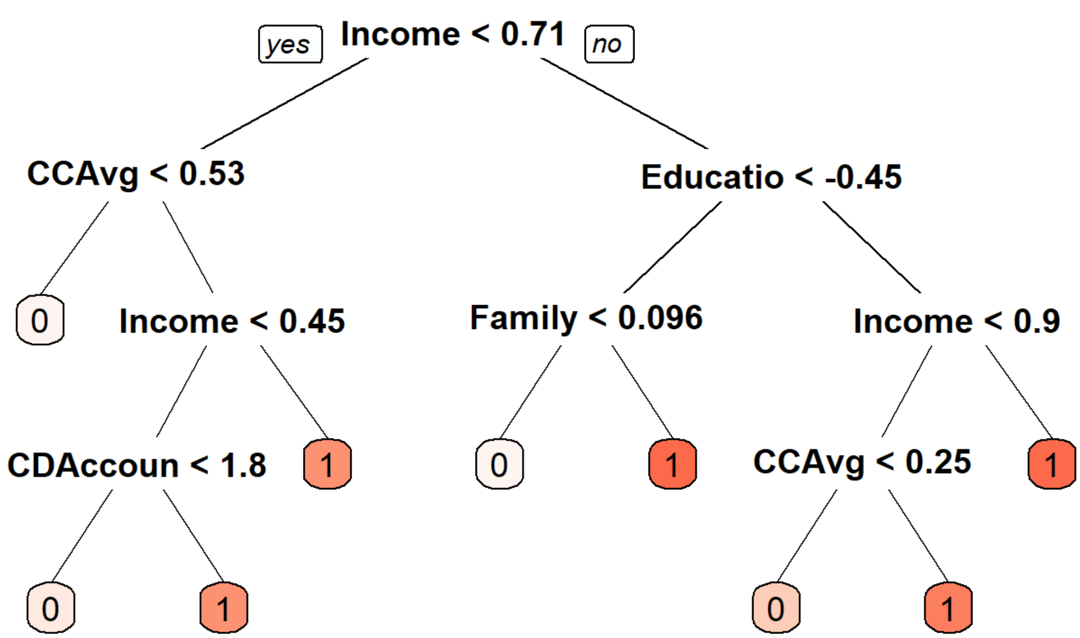
Accuracy <dbl>	Kappa <dbl>	Resample <chr>
0.9735099	0.8602499	Fold06.Rep3
0.9731544	0.8426610	Fold02.Rep3
0.9733333	0.8519250	Fold07.Rep3
0.9733333	0.8601399	Fold08.Rep2
0.9865772	0.9259075	Fold03.Rep3
0.9866667	0.9300699	Fold03.Rep2
0.9536424	0.7338202	Fold04.Rep2
0.9800000	0.8920863	Fold09.Rep2
0.9800000	0.8978665	Fold08.Rep3
0.9798658	0.8855314	Fold09.Rep1

1-10 of 30 rows   Previous **1** 2 3 Next

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

```
## n= 1500
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 1500 165 0 (0.89000000 0.11000000)
##    2) Income< 0.7130012 1154  21 0 (0.98180243 0.01819757)
##      4) CCAvg< 0.5290213 1070   0 0 (1.00000000 0.00000000) *
##      5) CCAvg>=0.5290213 84  21 0 (0.75000000 0.25000000)
##       10) Income< 0.4531979 67  11 0 (0.83582090 0.16417910)
##         20) CDAccount< 1.792241 60   6 0 (0.90000000 0.10000000) *
##         21) CDAccount>=1.792241 7   2 1 (0.28571429 0.71428571) *
##       11) Income>=0.4531979 17   7 1 (0.41176471 0.58823529) *
##    3) Income>=0.7130012 346 144 0 (0.58381503 0.41618497)
##      6) Education< -0.4468338 211  25 0 (0.88151659 0.11848341)
##       12) Family< 0.09554566 183   0 0 (1.00000000 0.00000000) *
##       13) Family>=0.09554566 28   3 1 (0.10714286 0.89285714) *
##      7) Education>=-0.4468338 135  16 1 (0.11851852 0.88148148)
##       14) Income< 0.8970285 27  11 0 (0.59259259 0.40740741)
##         28) CCAvg< 0.2508981 20   5 0 (0.75000000 0.25000000) *
##         29) CCAvg>=0.2508981 7   1 1 (0.14285714 0.85714286) *
##       15) Income>=0.8970285 108   0 1 (0.00000000 1.00000000) *
```

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




```r
# Predict on test data
dtree_predict <- predict(dtree_fit, newdata = bank1_x_test)
dtree_predict_prob <- predict(dtree_fit, newdata = bank1_x_test, type = "prob")
```

```r
# Print Confusion matrix, Accuarcy, Sensitivity etc
confusionMatrix(dtree_predict,  bank1_y_test, positive="1" )
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 450   3
##          1   4  44
##
##                Accuracy : 0.986
##                  95% CI : (0.9714, 0.9944)
##     No Information Rate : 0.9062
##     P-Value [Acc > NIR] : 8.166e-14
##
##                   Kappa : 0.9186
##
##  Mcnemar's Test P-Value : 1
##
##             Sensitivity : 0.93617
##             Specificity : 0.99119
##          Pos Pred Value : 0.91667
##          Neg Pred Value : 0.99338
##              Prevalence : 0.09381
##          Detection Rate : 0.08782
##    Detection Prevalence : 0.09581
##       Balanced Accuracy : 0.96368
##
##        'Positive' Class : 1
##
```

```
# Add results into clf_results dataframe
x2 <- confusionMatrix(dtree_predict,  bank1_y_test, positive="1")[["overall"]]
y2 <- confusionMatrix(dtree_predict,  bank1_y_test, positive="1" )[["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  0.986 and F1 is  0.926
```

```
# Add results into cost_benefit_df dataframe for cost benefit analysis
a2 <- confusionMatrix(dtree_predict,  bank1_y_test )

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

1.4 Logistic regression

Convert categorical outcome into numerical. Logistic regression cannot handle categorical variables

```
glm_fit <- train(bank1_x_train,
                 bank1_y_train,
                 method = "glm",
                 family = "binomial",
                 preProc = c("center", "scale"))

glm_fit$resample
```

Accuracy <dbl>	Kappa <dbl>	Resample <chr>
0.9320215	0.6177296	Resample01
0.9362477	0.6251097	Resample02
0.9420035	0.6658481	Resample03
0.9437387	0.6944529	Resample04
0.9337017	0.6298709	Resample05
0.9423423	0.7054042	Resample06
0.9500000	0.7412508	Resample07
0.9355433	0.6707209	Resample08
0.9346290	0.6067821	Resample09
0.9532374	0.7334513	Resample10

1-10 of 25 rows Previous **1** 2 3 Next

```
#notice that the model was estimated 25 times
glm_fit$resampledCM
```

cell1 <dbl>	cell2 <dbl>	cell3 <dbl>	cell4 <dbl>	parameter <chr>	Resample <chr>
485	11	27	36	none	Resample01
480	15	20	34	none	Resample02
498	9	24	38	none	Resample03
479	12	19	41	none	Resample04
471	17	19	36	none	Resample05
478	13	19	45	none	Resample06
468	11	16	45	none	Resample07
466	8	27	42	none	Resample08
496	7	30	33	none	Resample09
489	10	16	41	none	Resample10

1-10 of 25 rows Previous **1** 2 3 Next

```
# Predict on test data
glm_predict <- predict(glm_fit, newdata = bank1_x_test)
glm_predict_prob <- predict(glm_fit, newdata = bank1_x_test, type="prob")
```

convert probability outcome into categorical outcome

```
y_pred_num <- ifelse(glm_predict_prob[,2] > 0.5, 1, 0)
```

```
# Print Confusion matrix, Accuarcy, Sensitivity etc
confusionMatrix(as.factor(y_pred_num), as.factor(bank1_y_test), positive = "1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 448  12
##          1   6  35
##
##                Accuracy : 0.9641
##                  95% CI : (0.9438, 0.9786)
##     No Information Rate : 0.9062
##     P-Value [Acc > NIR] : 4.784e-07
##
##                   Kappa : 0.7759
##
##  Mcnemar's Test P-Value : 0.2386
##
##             Sensitivity : 0.74468
##             Specificity : 0.98678
##          Pos Pred Value : 0.85366
##          Neg Pred Value : 0.97391
##              Prevalence : 0.09381
##          Detection Rate : 0.06986
##    Detection Prevalence : 0.08184
##       Balanced Accuracy : 0.86573
##
##        'Positive' Class : 1
##
```

```
# Add results into clf_results dataframe
x3 <- confusionMatrix(as.factor(y_pred_num), as.factor(bank1_y_test), positive = "1")[["overall"]]
y3 <- confusionMatrix(as.factor(y_pred_num), as.factor(bank1_y_test), positive = "1")[["byClass"]]

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  0.964 and F1 is  0.795
```

```
# Add results into cost_benefit_df dataframe for cost benefit analysis
a3 <- confusionMatrix(as.factor(y_pred_num), as.factor(bank1_y_test))

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])
```

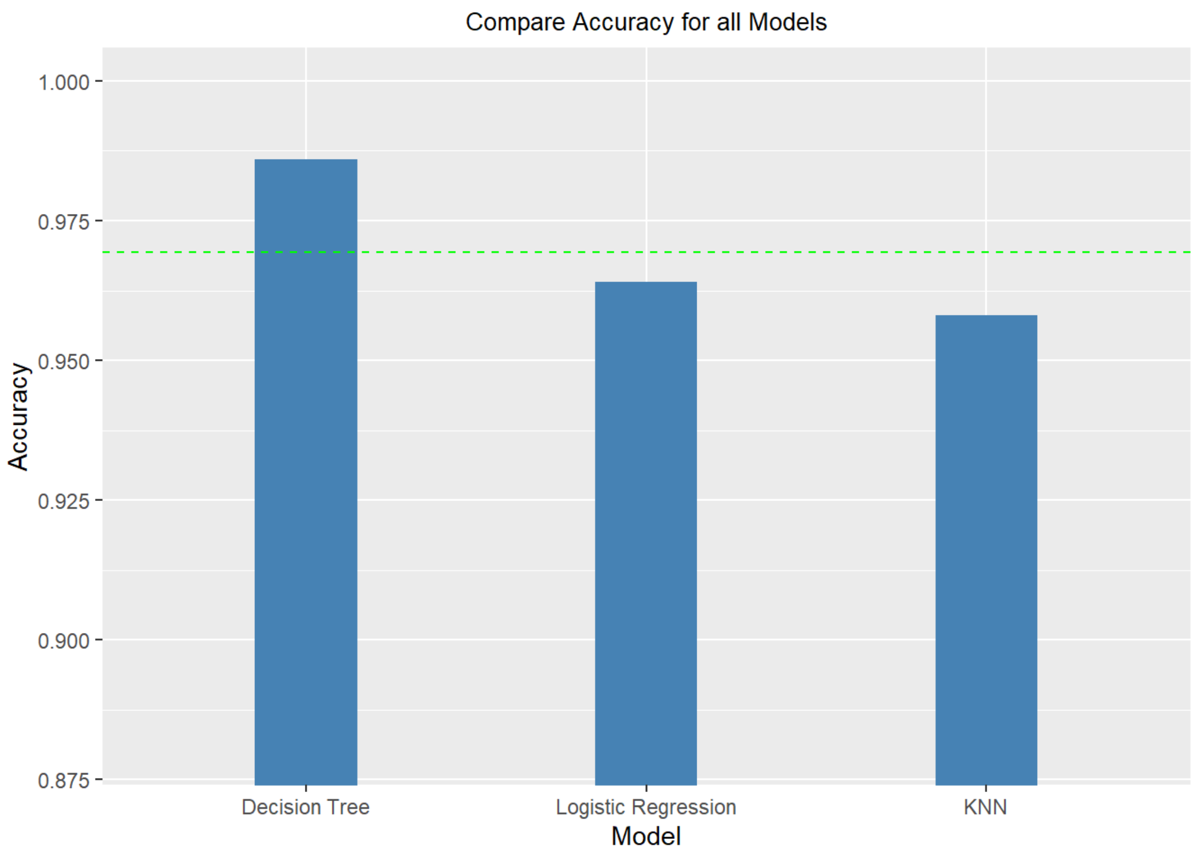
Compare Accuracy for all Classification models

```
print(clf_results)
```

```
##                 Model Accuracy Precision Recall    F1
## 1                 KNN    0.958     0.933  0.596 0.727
## 2       Decision Tree    0.986     0.917  0.936 0.926
## 3 Logistic Regression    0.964     0.854  0.745 0.795
```

```
# Plot accuracy for all the Classification Models

ggplot(clf_results[1:3,] %>% 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 of a true Positive is that you acquire a customer = 10000 benefit of a true negative is that you don’t target a customer, and maybe save some operational costs and even annoyance costs to cust. = 10 cost of a false negative is that you missed out on acquiring this customer = -8000 cost of a false positive is that you target and potentially annoy this customer = -8000 = -100

```
benefit_TP = 10000
benefit_TN = 10
cost_FN = -8000
cost_FP = -100

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

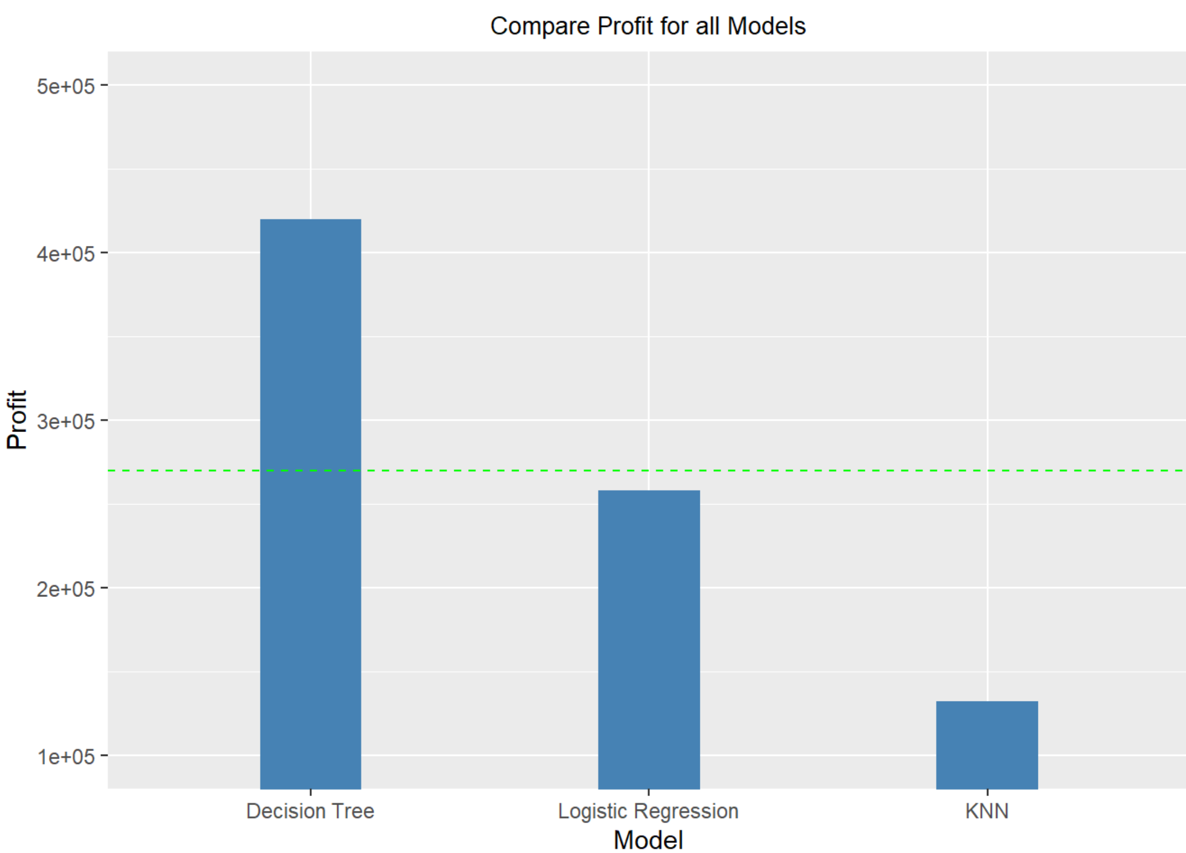
Compare Profit for all Classification models

```
print(cost_benefit_df)
```

```
##                 Model TP FN FP  TN Profit
## 1                 KNN 28 19  2 452 132320
## 2       Decision Tree 44  3  4 450 420100
## 3 Logistic Regression 35 12  6 448 257880
```

```
# Plot Profit for all the Classification Models

ggplot(cost_benefit_df[1:3,] %>% 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(100000, 500000)) +
    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))
```



#2 Scoring the target list ##2.1 Let us use the decision tree model (as it had highest accuracy)

```r
# #load new data that has not been scored

bank1.new <- read_excel("CL-bank-training-testingPredicting.xlsx",
                    sheet = "newDataForTargeting",
                    col_types = c("numeric", "numeric", "numeric",
                                  "numeric", "skip", "numeric", "numeric",
                                  "numeric", "numeric", "numeric",
                                  "numeric", "numeric", "numeric",
                                  "text"))

bank1.new_x <- bank1.new %>% select(-c("ID", "PersonalLoan"))

# Normalize x variables since they are at different scale
bank1.new_x_normalized <- as.data.frame(lapply(bank1.new_x, normalize))

#use the fitted tree model to score this new data
#note here, i want to rank order my targets by probability of responding
#hence we tell R type="prob" as opposed to type="class"
#latter would give 0, 1 output



dTreePredict.new <- predict(dtree_fit, newdata = bank1.new_x_normalized, type="prob")

#above results in 2 columns, probability of class 0 and probability of class 1 in the 2nd column
#we will extract the 2nd column

bank1.new$ProbAccepting <- dTreePredict.new[,2]

#sort this in descending order and add a column to the bank1.new data frame
#could have also used mutate here

bank1.new <- bank1.new %>% arrange(desc(ProbAccepting))

#lets count

successNum <-  bank1.new[1:100,] %>% filter(ProbAccepting>0.5) %>% tally()
# Print targeting accuracy

cat("If we target the top 100 people ", successNum[[1]], " out of 100 have a chance > 50% of converting","\n")
```

```
## If we target the top 100 people  46  out of 100 have a chance > 50% of converting
```

```r
cat("If we have no model, we get a 10% repsonse rate and should get 5 hits")
```

```
## If we have no model, we get a 10% repsonse rate and should get 5 hits
```

```r
#write this sorted target list as a CSV file
write.csv(x=bank1.new, file = "targetList1.csv", row.names = TRUE)
```

##2.2 Let us use the kNN model

Note that we will retrain the model for k=3 with the entire dataset. This will allow the new observations to find the 3 nearest among 2001 rows, as opposed to the smaller training set.

```
# simple control her with k=3
# make sure train() also records class probabilities, not just the 0 or 1 classification
simple_control <- trainControl(classProbs = TRUE)

param_fixed <-  expand.grid( k = 3)

# knn will throw an error with the O and 1 class labels, hence the command below
levels(bank1$PersonalLoan) = c("nonAcceptor", "Acceptor")

# fit the model to training data
knn_fullData_deployment_time <- train(bank1_x_normalized,
                     bank1$PersonalLoan,
                     method = "knn",
                     tuneGrid = param_fixed,
                     trControl = simple_control )



knnPredict.new <- predict(knn_fullData_deployment_time, newdata = bank1.new_x_normalized, type="prob")

#above results in 2 columns, probability of class 0 and probability of class 1 in the 2nd column
#we will extract the 2nd column

bank1.new$ProbAccepting <- knnPredict.new[,2]

#sort this in descending order and add a column to the bank1.new data frame
#could have also used mutate here

bank1.new <- bank1.new %>% arrange(desc(ProbAccepting))

#lets count

successNum <-  bank1.new[1:50,] %>% filter(ProbAccepting>0.5) %>% tally()
# Print targeting accuracy

cat("If we target the top 50 people ", successNum[[1]], " out of 50 have a chance > 50% of converting","\n")
```

```
## If we target the top 50 people  29  out of 50 have a chance > 50% of converting
```

```
cat("If we have no model, we get a 10% repsonse rate and should get 5 hits")
```

```
## If we have no model, we get a 10% repsonse rate and should get 5 hits
```

```
#we can end by writing this sorted target list as a CSV file
write.csv(x=bank1.new, file = "targetList1-knn.csv", row.names = TRUE)
```