Assignment_ML2

```r
chooseCRANmirror(graphics = getOption("menu.graphics"), ind = 79,
                 local.only = FALSE)
install.packages("caret")
```

```
## Installing package into 'C:/Users/ibeme/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)

## package 'caret' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'caret'

## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying C:
## \Users\ibeme\Documents\R\win-library\4.1\00LOCK\caret\libs\x64\caret.dll to C:
## \Users\ibeme\Documents\R\win-library\4.1\caret\libs\x64\caret.dll: Permission
## denied

## Warning: restored 'caret'

##
## The downloaded binary packages are in
##  C:\Users\ibeme\AppData\Local\Temp\RtmpEDEeqo\downloaded_packages
```

```r
install.packages("ISLR")
```

```
## Installing package into 'C:/Users/ibeme/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)

## package 'ISLR' successfully unpacked and MD5 sums checked
##
## The downloaded binary packages are in
##  C:\Users\ibeme\AppData\Local\Temp\RtmpEDEeqo\downloaded_packages
```

```r
install.packages("class")
```

```
## Installing package into 'C:/Users/ibeme/Documents/R/win-library/4.1'
## (as 'lib' is unspecified)

## package 'class' successfully unpacked and MD5 sums checked

## Warning: cannot remove prior installation of package 'class'
```

```
## Warning in file.copy(savedcopy, lib, recursive = TRUE): problem copying C:
## \Users\ibeme\Documents\R\win-library\4.1\00LOCK\class\libs\x64\class.dll to C:
## \Users\ibeme\Documents\R\win-library\4.1\class\libs\x64\class.dll: Permission
## denied

## Warning: restored 'class'

##
## The downloaded binary packages are in
##  C:\Users\ibeme\AppData\Local\Temp\RtmpEDEeqo\downloaded_packages
```

```
library(class)
library(caret)
```

```
## Loading required package: ggplot2

## Loading required package: lattice
```

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

KNN Algorithm

1.Classification of customer using K=1

Data Exploration

- Imported data from UniversalBank CSV file.
- Eliminated ID,ZIPCODE Variables from the Dataset.
- Convert Education column into Dummy Variables (Education_1,Education2,Education3) and added these dummy variabes to the dataset and dropped the original education variable.

```
# Importing the file
Universal_data<- read.csv("UniversalBank.csv")

#Eliminating variables [ID & Zipcode] from the dataset.

Universal_sub_data <- Universal_data[c(-1,-5)]


#Converting Education Categorical variables into dummy Variables
library(psych)
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```r
dummy_Education <- as.data.frame(dummy.code(Universal_sub_data$Education))
names(dummy_Education) <- c("Education_1", "Education_2","Education_3")

#Eliminating Education variable from Dataset

Universal_new_data_without_Education = subset(Universal_sub_data,select = - c(Education))
Universal_new_data <- cbind(Universal_new_data_without_Education,dummy_Education)


Universal_new_data$Personal.Loan <- as.factor(Universal_new_data$Personal.Loan)
Universal_new_data$CCAvg <- as.integer(Universal_new_data$CCAvg)
```

Data Partitioning

- Data is split-ted into training(60%)[3000] and validation(40%)[2000] data

```r
# Splitting the data into training(60%) and validation(40%)

set.seed(123)
train_index <- createDataPartition(Universal_new_data$Personal.Loan,p=0.6,list = FALSE)
train_data <- Universal_new_data[train_index,] #3000 Observations
validation_data <- Universal_new_data[-train_index,] #2000 Observations
```

Generating the Test Data

- Created a dataframe for the given test data in the question1

```r
test_data<-data.frame(Age = 40, Experience = 10, Income = 84, Family = 2, CCAvg = 2, Education_1 = 0, Ed
test_data
```

```
##   Age Experience Income Family CCAvg Education_1 Education_2 Education_3
## 1  40         10     84      2     2           0           1           0
##   Mortgage Securities.Account CD.Account Online CreditCard
## 1        0                  0          0      1          1
```

Data Normalization

- Normalize the training data using preProcess function.
- Apply the normalized model on the training, validation data and test data using Predict Function.

```r
# Copy the original data
train.norm.df <- train_data
valid.norm.df <- validation_data
test.norm.df <- test_data
traval.norm.df <- Universal_new_data #(Training + Validation data)
```

```r
#Use preProcess() function to normalize numerical columns from the dataset

Values_z_normalised <- preProcess(train_data[,-7],method = c("center","scale"))

#Applying the normalized model on the training, validation and test data

train.norm.df[,-7] <-  predict(Values_z_normalised,train_data[,-7])
valid.norm.df[,-7] <-  predict(Values_z_normalised,validation_data[,-7])
traval.norm.df[,-7] <-  predict(Values_z_normalised,Universal_new_data[,-7])
test.norm.df <- predict(Values_z_normalised, test_data)

#summary(train.norm.df)
#var(train.norm.df[,-7])
#summary(valid.norm.df)
#var(valid.norm.df[,-7])
```

Modeling k-NN

- Performing Knn() function on training and validation using K=1.
- Calculating for Prediction probability and mean.
- Created a table for the the actual(Personal.loan) and predicted model(Model.k.1).

```r
set.seed(123)

Model.k.1<- knn(train=train.norm.df[,-7],test=valid.norm.df[,-7],cl= train.norm.df[,7],k=1,prob= TRUE)

actual=valid.norm.df[,7]
Prediction_prob =attr(Model.k.1,"prob")
head(Prediction_prob)
```

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

```r
table(Model.k.1, actual)
```

```
##          actual
## Model.k.1    0    1
##         0 1789   56
##         1   19  136
```

```r
mean(Model.k.1 == actual)
```

```
## [1] 0.9625
```

Classifying the customer using the k=1 [Performing KNN classification on test data]

- Before predicting the data, combined the training and validation data and renormalised the data and apply it on test data
- The knn model using k=1 predicted that the test data will be classified as ‘0’ with probability ‘1’(Loan will be denied)

```
# Renormalizing the (training+validation) data
set.seed(123)
Values_z_normalised2 <- preProcess(traval.norm.df[,-7], method = c("center","scale"))

traval.norm.df[,-7] <-  predict(Values_z_normalised2,Universal_new_data[,-7])
test.norm.df<- predict(Values_z_normalised2,test_data)


Prediction_test <- knn(train= traval.norm.df[,-7],test=test.norm.df,cl=traval.norm.df[,7],k=1,prob=TRUE)

head(Prediction_test)
```

```
## [1] 0
## Levels: 0 1
```

```
Prediction_test_prob<-attr(Prediction_test,"prob")

Prediction_test_prob
```

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

2.Choice of the best K that balances between overfitting and ignorning the predictor information

- To determine k, we use the performance on the validation set.Here, we will vary the value of k from 1 to 15.
- initialize a data frame with two columns: k, and accuracy.
- Computed knn for different K on validation and plotted a graph for K and corresponding accuracy.
- The best choice of K which also balances the model from over fitting is k=3 with accuracy(96.30%).

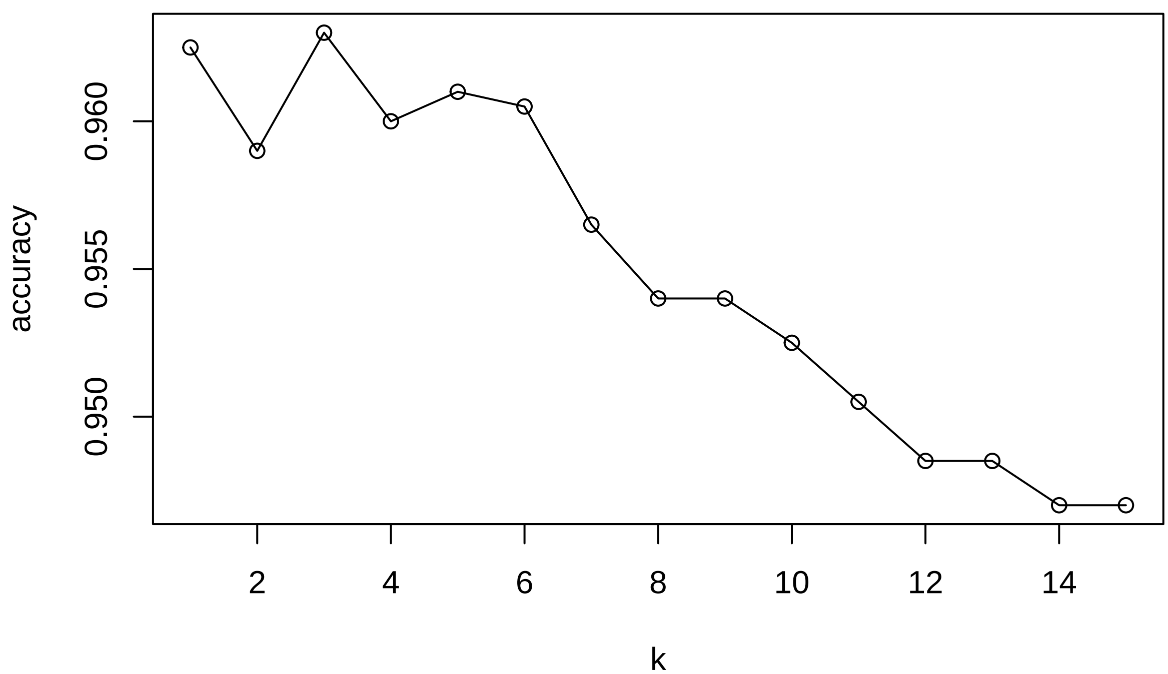
```
library(caret)
set.seed(123)
accuracy.df <- data.frame(k = seq(1, 15, 1), accuracy = rep(0, 15))

# compute knn for different k on validation.
for(i in 1:15) {
  knn.pred <- knn(train.norm.df[,-7], valid.norm.df[,-7],
                  cl = train.norm.df[, 7], k = i)
  accuracy.df[i, 2] <- confusionMatrix(knn.pred,valid.norm.df[,7])$overall[1]
}
accuracy.df
```

```
##     k accuracy
## 1   1   0.9625
## 2   2   0.9590
## 3   3   0.9630
## 4   4   0.9600
## 5   5   0.9610
```

```
## 6   6   0.9605
## 7   7   0.9565
## 8   8   0.9540
## 9   9   0.9540
## 10 10   0.9525
## 11 11   0.9505
## 12 12   0.9485
## 13 13   0.9485
## 14 14   0.9470
## 15 15   0.9470
```

```
plot(accuracy.df,type="o")
```



3.Confusion matrix for the validation results using best K

- From above question it is inferred that k=3 is the best choice of k.
- performing knn function on the validation data using the best k=3.
- computed confusion matrix with highest sensitivity and moderate specificity.

```
Model.k.3 <- knn(train.norm.df[,-7], valid.norm.df[,-7],
                  cl = train.norm.df[,7], k = 3,prob = TRUE)
confusionMatrix(Model.k.3,valid.norm.df[,7])
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1805   71
##          1    3  121
##
##                Accuracy : 0.963
##                  95% CI : (0.9538, 0.9708)
##     No Information Rate : 0.904
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.7467
##
##  Mcnemar's Test P-Value : 6.776e-15
##
##             Sensitivity : 0.9983
##             Specificity : 0.6302
##          Pos Pred Value : 0.9622
##          Neg Pred Value : 0.9758
##              Prevalence : 0.9040
##          Detection Rate : 0.9025
##    Detection Prevalence : 0.9380
##       Balanced Accuracy : 0.8143
##
##        'Positive' Class : 0
##
```

4.Classifying the customer using the best K

- Performed KNN model on the test data from question one using the best k value (k=3).
- The knn model using k=1 predicted that the test data will be classified as '1' (Loan will be accepted).

```
Model.Best.k <-knn(train=train.norm.df[,-7],test=test_data,cl=train.norm.df[,7],k=3,prob=TRUE)
head(Model.Best.k)
```

```
## [1] 1
## Levels: 0 1
```

5.RePartitioning the data

- Splitted data into Training data(50%),Validation Data(30%) and test data(20%)
- Normalize the training data using preProcess function
- Apply the normalized model on the training, validation data and test data using Predict Function
- Performing Knn() function on training and validation using K=3
- Calculating for Prediction probability and mean
- Created a table for the the actual(Personal.loan) and predicted model(Model.k.1)

Accuracy of the Knn models for traning,validation and test datasets for k=3 Train_Knn= 97.6% (I tried to understand on how model reacts to the same data it got trained.) Valid_Knn = 95.2% test_knn = 96.9% It is known that the larger the model , more unlikely it will overfit. The model performed better in the test data set as it got enough data to learn from ie, 80% of the data(Training and validation), whereas validation model learned from 50% of the training data.More the data ,better the accuracy.

```
## Data Splitting (50% Training Data and 30% for validation data and 20% test data)
set.seed(123)
str(Universal_new_data)
```

```
## 'data.frame':    5000 obs. of  14 variables:
##  $ Age               : int  25 45 39 35 35 37 53 50 35 34 ...
##  $ Experience        : int  1 19 15 9 8 13 27 24 10 9 ...
##  $ Income            : int  49 34 11 100 45 29 72 22 81 180 ...
##  $ Family            : int  4 3 1 1 4 4 2 1 3 1 ...
##  $ CCAvg             : int  1 1 1 2 1 0 1 0 0 8 ...
##  $ Mortgage          : int  0 0 0 0 0 155 0 0 104 0 ...
##  $ Personal.Loan     : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 2 ...
##  $ Securities.Account: int  1 1 0 0 0 0 0 0 0 0 ...
##  $ CD.Account        : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Online            : int  0 0 0 0 0 1 1 0 1 0 ...
##  $ CreditCard        : int  0 0 0 0 1 0 0 1 0 0 ...
##  $ Education_1       : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ Education_2       : num  0 0 0 0 0 0 0 1 0 1 ...
##  $ Education_3       : num  0 0 0 1 1 1 1 0 1 0 ...
```

```
test_index1 <- createDataPartition(Universal_new_data$Personal.Loan,p=0.2,list = FALSE)
Test_Data2 <- Universal_new_data[test_index1,]# 1000 Rows (Test data)
train_vali_data <- Universal_new_data[-test_index1,]

train_index2 <- createDataPartition(train_vali_data$Personal.Loan,p=0.625,list = FALSE)
train_data2 <- train_vali_data[train_index2,] #2500 Rows (Training data)
validation_data2 <- train_vali_data[-train_index2,]#1500 Rows (validation data)

 NROW(Test_Data2)
```

```
## [1] 1000
```

```
 NROW(train_data2)
```

```
## [1] 2500
```

```
 NROW(validation_data2)
```

```
## [1] 1500
```

```
 # Data Normalization

# Copy the original data
train.norm.df2 <- train_data2
valid.norm.df2 <- validation_data2
```

```r
train_vali.norm.df <- train_vali_data
test.norm.df2 <-Test_Data2

#Use preProcess() function to normalize numerical columns from the Universal_new_data dataset

Values_z_normalised_repartition <- preProcess(train_data2[,-7],method = c("center","scale"))


train.norm.df2[,-7] <- predict(Values_z_normalised_repartition,train_data2[,-7])
valid.norm.df2[,-7] <- predict(Values_z_normalised_repartition,validation_data2[,-7])
train_vali.norm.df[,-7] <- predict(Values_z_normalised2,train_vali_data[,-7])
test.norm.df2[,-7] <-predict(Values_z_normalised_repartition,Test_Data2[,-7])


#summary(train.norm.df2)
#var(train.norm.df2[, c(1:3,5,7)])
#summary(valid.norm.df2)
#var(valid.norm.df2[, c(1:3,5,7)])

## Modeling k-NN for validation data

set.seed(123)
train_knn_3<- knn(train.norm.df2[,-7],train.norm.df2[,-7],cl=train.norm.df2[,7],k=3,prob=TRUE)
valid_knn_3<- knn(train.norm.df2[,-7],valid.norm.df2[,-7],cl= train.norm.df2[,7],k=3,prob= TRUE)
#print(ModelNew.k.1)
head(train_knn_3)
```

```
## [1] 0 0 0 0 0 1
## Levels: 0 1
```

```r
head(valid_knn_3)
```

```
## [1] 0 0 0 0 0 0
## Levels: 0 1
```

```r
actual= valid.norm.df2[,7]
mean(valid_knn_3==actual)
```

```
## [1] 0.952
```

```r
class_prob = attr(valid_knn_3,"prob")
head(class_prob)
```

```
## [1] 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 0.6666667
```

```r
# Knn for test data

Values_z_normalised3<- preProcess(train_vali_data[,-7], method = c("center","scale"))

train_vali.norm.df[,-7] <- predict(Values_z_normalised3,train_vali_data[,-7])
```

```
test.norm.df2[,-7]<- predict(Values_z_normalised3,Test_Data2[,-7])


test_knn_3<- knn(train_vali.norm.df[,-7],test.norm.df2[,-7],cl=train_vali.norm.df[,7],k=3)
#print(Model_new3)

head(test_knn_3)
```

```
## [1] 0 0 1 0 0 0
## Levels: 0 1
```

```
actual= test.norm.df2[,7]
mean(test_knn_3==actual)
```

```
## [1] 0.969
```

Including Confusion Matrix

Accuracy of the Knn models for traning,validation and test datasets for k=3 Train_Knn= 97.6% (I tried to understand on how model reacts to the same data it got trained.) Valid_Knn = 95.2% test_knn = 96.9% It is known that the larger the model , more unlikely it will overfit. The model performed better in the test data set as it got enough data to learn from ie, 80% of the data(Training and validation), whereas validation model learned from 50% of the training data.More the data ,better the accuracy.

```
confusionMatrix(train_knn_3,as.factor(train.norm.df2[,7]),positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 2257   56
##          1    3  184
##
##                Accuracy : 0.9764
##                  95% CI : (0.9697, 0.982)
##     No Information Rate : 0.904
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8491
##
##  Mcnemar's Test P-Value : 1.289e-11
##
##             Sensitivity : 0.7667
##             Specificity : 0.9987
##          Pos Pred Value : 0.9840
##          Neg Pred Value : 0.9758
##              Prevalence : 0.0960
##          Detection Rate : 0.0736
##    Detection Prevalence : 0.0748
##       Balanced Accuracy : 0.8827
##
```

```
##        'Positive' Class : 1
##
```

```
confusionMatrix(valid_knn_3,as.factor(valid.norm.df2[,7]),positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 1342   58
##          1   14   86
##
##                Accuracy : 0.952
##                  95% CI : (0.9399, 0.9623)
##     No Information Rate : 0.904
##     P-Value [Acc > NIR] : 3.516e-12
##
##                   Kappa : 0.6797
##
##  Mcnemar's Test P-Value : 4.029e-07
##
##             Sensitivity : 0.59722
##             Specificity : 0.98968
##          Pos Pred Value : 0.86000
##          Neg Pred Value : 0.95857
##              Prevalence : 0.09600
##          Detection Rate : 0.05733
##    Detection Prevalence : 0.06667
##       Balanced Accuracy : 0.79345
##
##        'Positive' Class : 1
##
```

```
confusionMatrix(test_knn_3,as.factor(test.norm.df2[,7]),positive = '1')
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction   0   1
##          0 903  30
##          1   1  66
##
##                Accuracy : 0.969
##                  95% CI : (0.9563, 0.9788)
##     No Information Rate : 0.904
##     P-Value [Acc > NIR] : 1.027e-15
##
##                   Kappa : 0.7935
##
##  Mcnemar's Test P-Value : 4.932e-07
##
##             Sensitivity : 0.6875
##             Specificity : 0.9989
```

```
##          Pos Pred Value : 0.9851
##          Neg Pred Value : 0.9678
##              Prevalence : 0.0960
##          Detection Rate : 0.0660
##    Detection Prevalence : 0.0670
##       Balanced Accuracy : 0.8432
##
##        'Positive' Class : 1
##
```

Note that the `echo = FALSE` parameter was added to the code chunk to prevent printing of the R code that generated the plot.