Will a Customer Open a Bank Account?

Code

Below UCI dataset is a “Bank Marketing” campaign that contains records of calls made by a Portugese bank to its clients, including client and campaign attributes. Analysis is based on classfication methods to understand if a customer is going to open a deposit account or not.

Data Cleaning and Preparation

Load all relevant libraries.

Hide

Hide

```
library(readr)
library(ggplot2)
library(lattice)
library(plyr)
library(dplyr)
library(caret)
library(mlbench)
library(foreign)
library(ggplot2)
library(reshape)
library(scales)
library(e1071)
library(MASS)
library(klaR)
library(C50)
library(kernlab)
library(nnet)
```

Read the data set on bank clients. Here, analysis is based on the smaller dataset that represents randomly selected 10% of the entire dataset, so that computationally demanding algorithms (eg: SVM) can be performed faster.

Hide

Hide

```
bank <- read_delim("~/Documents/homework/ITM_6285/bank-additional.csv",";",escape_dou
ble = FALSE, trim_ws = TRUE)
bank <- subset(bank, select = -c(duration))
```

There are 20 attributes in the dataset. Since duration has a high correlation with the target variable, variable named ‘duration’ is removed from the dataset. Here is a breakdown of all 20 variables in the dataset along with variable data type. The target variable is y which has two values: ‘yes’ (customer opens a bank account)

and ‘no’ (customer does not open an account).

To get an understanding of the data, lets visualize a few variables.

Hide

Hide

```
table(bank$y)
```

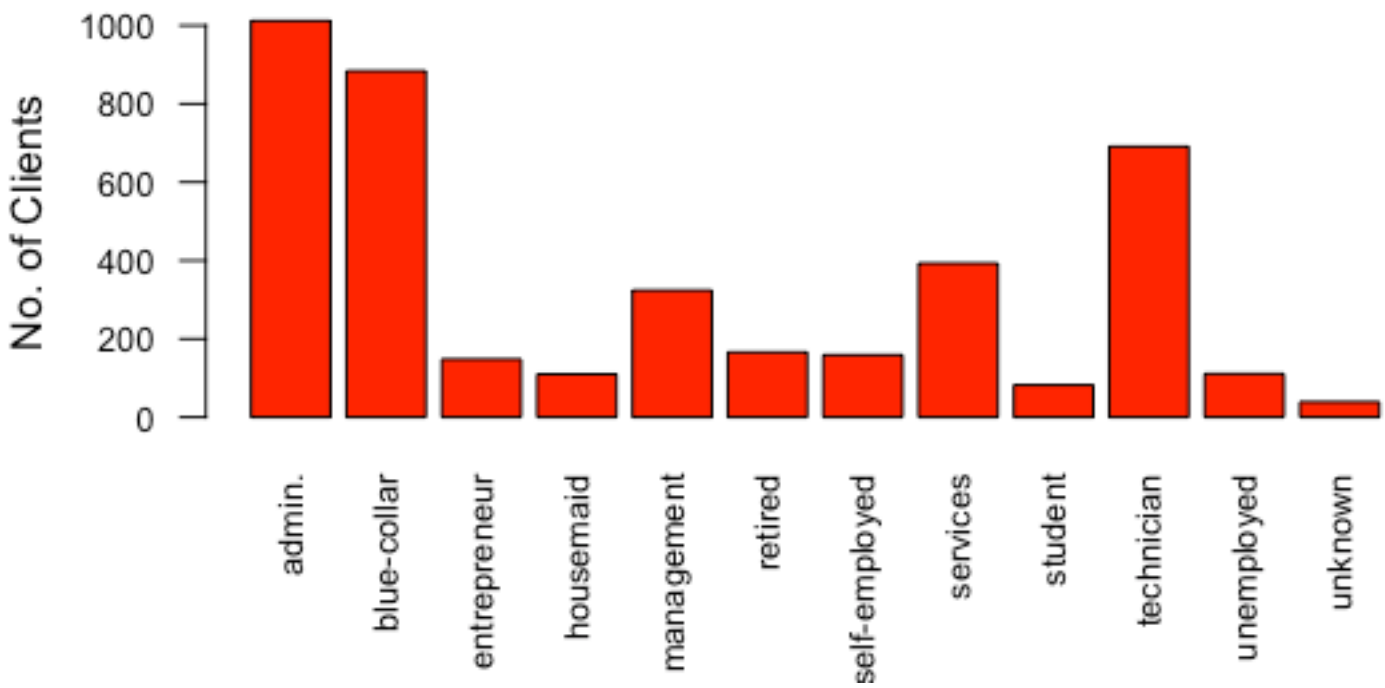
```
  no  yes
3668  451
```

The dataset contains 3668 ‘no’ responses and 451 ‘yes’ responses. Below is the distribution by occupation and age.

Hide

Hide

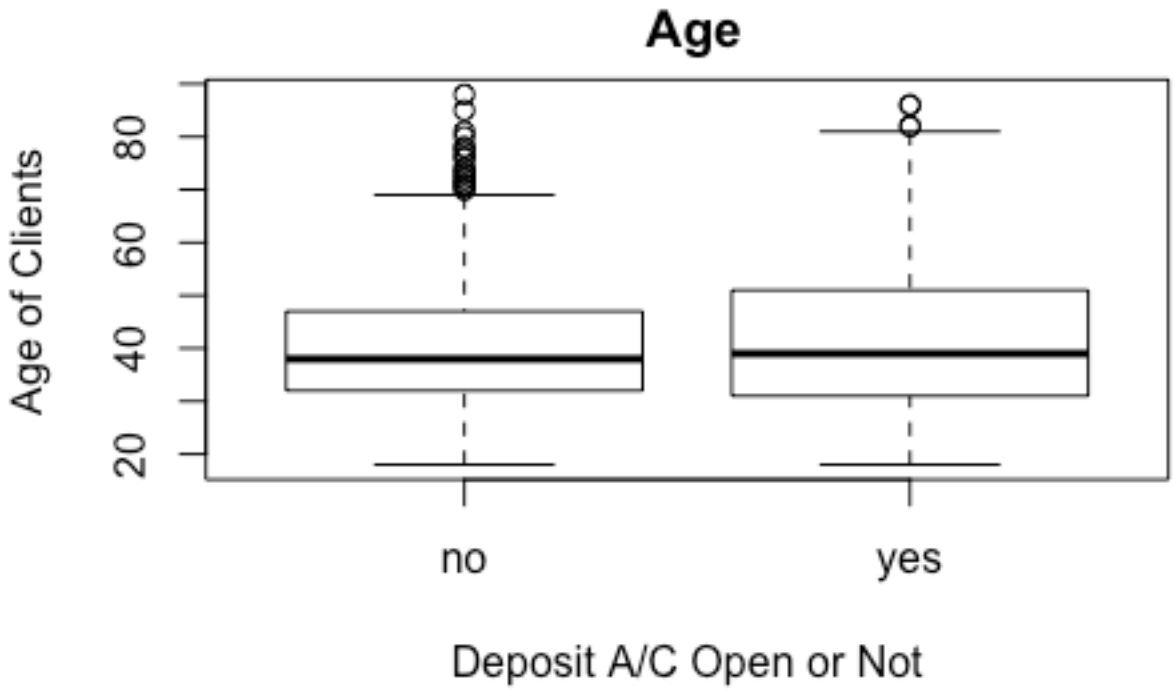
```
barplot(table(bank$job),col="red",ylab="No. of Clients",las=2,main="Job",cex.names =
0.8,cex.axis = 0.8)
```



Hide

Hide

```
boxplot(bank$age~bank$y, main=" Age",ylab="Age of Clients",xlab="Deposit A/C Open or
Not")
```



Splitting Data for Testing and Training

Now the dataset of 4119 observations are splitted into training and test data. We use stratified sampling to split the data, so that distribution of the outcome within traning and testing datasets is preserved. We split the data with 75% (or 3090) of observations is used for training the model and 25% (or 1029) of observations is used to test the prediction outcome from the classifier model.

Hide

Hide

```
set.seed(123456)
TrainingDataIndex <- createDataPartition(bank$y, p=0.75, list = FALSE)
train <- bank[TrainingDataIndex,]
test <-bank[-TrainingDataIndex,]
prop.table(table(train$y))
```

```
       no       yes 
0.8902913 0.1097087 
```

Hide

Hide

```
nrow(train)
```

```
[1] 3090
```

Hide

Hide

```
prop.table(table(test$y))
```

```
       no       yes
0.8911565 0.1088435
```

Hide

Hide

```
nrow(test)
```

```
[1] 1029
```

Thus, stratified sampling has enabled to maintain the distribution with about 89% of clients have responded ‘no’ to opening a deposit in both testing and training data set.

1. Classification Methods

a) Decision Tree

Training the Model

After partitioning the data to train and test, use a 10 fold cross validation repeated 5 times to evaluate the model.

Hide

Hide

```
TrainingParameters <- trainControl(method = "cv", number = 10, repeats = 5)
```

Then create the decision tree using the C5.0 algorithm.

Hide

Hide

```
DecTreeModel <- train(y ~ ., data = train,
                      method = "C5.0",
                      trControl= TrainingParameters,
                      na.action = na.omit)
```

Lets take a look.

Hide

Hide

```
DecTreeModel
```

```
C5.0

3090 samples
  19 predictor
   2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 2781, 2781, 2781, 2782, 2781, 2781, ...
Resampling results across tuning parameters:

  model  winnow  trials  Accuracy   Kappa
  rules  FALSE    1      0.9009644  0.2583583
  rules  FALSE   10      0.8954690  0.2759816
  rules  FALSE   20      0.8957927  0.2839006
  rules   TRUE    1      0.8974056  0.2271179
  rules   TRUE   10      0.8944908  0.2386011
  rules   TRUE   20      0.8944908  0.2458461
  tree   FALSE    1      0.9009644  0.2583583
  tree   FALSE   10      0.8961173  0.2716831
  tree   FALSE   20      0.8957927  0.2820709
  tree    TRUE    1      0.8974056  0.2271179
  tree    TRUE   10      0.8948145  0.2395735
  tree    TRUE   20      0.8951381  0.2431570

Accuracy was used to select the optimal model using  the largest value.
The final values used for the model were trials = 1, model = rules and winnow = FALSE
.
```

Hide

Hide

```
summary(DecTreeModel)
```

```
Call:
C5.0.default(x = structure(c(30, 39, 25, 38, 47, 32, 32, 41, 31, 35, 25, 36, 29, 27,
46, 45, 50, 39,
 = c("subset", "bands", "winnow", "noGlobalPruning", "CF", "minCases", "fuzzyThreshol
d",
 "sample", "earlyStopping", "label", "seed")))


C5.0 [Release 2.07 GPL Edition]     Thu Mar 16 20:18:59 2017
-------------------------------

Class specified by attribute `outcome'

Read 3090 cases (53 attributes) from undefined.data
```

```
Rules:

Rule 1: (2816/207, lift 1.0)
    nr.employed > 5023.5
    ->  class no  [0.926]

Rule 2: (2977/266, lift 1.0)
    poutcomesuccess <= 0
    ->  class no  [0.910]

Rule 3: (78/17, lift 7.1)
    poutcomesuccess > 0
    nr.employed <= 5023.5
    ->  class yes  [0.775]

Default class: no


Evaluation on training data (3090 cases):

           Rules
     ----------------
       No      Errors

        3  295( 9.5%)   <<


      (a)   (b)    <-classified as
     ----  ----
     2734    17    (a): class no
      278    61    (b): class yes


    Attribute usage:

     98.87% poutcomesuccess
     93.66% nr.employed


Time: 0.0 secs
```

For instance, Rule 1 shows that when the number of employees in a quarter is greater than 5023, it was assigned the class ‘no’ (client does not want to open a bank account) 2816 times and out of 2816 times, the model incorrectly assigned ‘no’ 207 times.

Based on the training data confusion matrix, 9.5% of observations were assigned an incorrect class variable.

Testing the Model

Hide

Hide

```
DTPredictions <-predict(DecTreeModel, test, na.action = na.pass)
confusionMatrix(DTPredictions, test$y)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  no yes
       no  911  97
       yes   6  15

               Accuracy : 0.8999
                 95% CI : (0.8799, 0.9176)
    No Information Rate : 0.8912
    P-Value [Acc > NIR] : 0.1984

                  Kappa : 0.198
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9935
            Specificity : 0.1339
         Pos Pred Value : 0.9038
         Neg Pred Value : 0.7143
             Prevalence : 0.8912
         Detection Rate : 0.8853
   Detection Prevalence : 0.9796
      Balanced Accuracy : 0.5637

       'Positive' Class : no
```

Based on confusion matrix for test data, using the decision tree model we have correctly classified 911 + 15 = 926 observations and misclassified 6 + 97 = 103 representing a 90% accuracy.

b) Naive Bayes

Training the Model

The next machine learning method used to predict if a customer opens a bank account is Naive Bayes method. The Naive Bayes method assumes independece among each 19 variables, i.e. the algorithm assumes that attributes such as job and education are independent from each other in predicting whether a customer will open a bank account or not.

Hide

Hide

```
set.seed(100)
TrainingDataIndex <- createDataPartition(bank$y, p=0.75, list = FALSE)
train <- bank[TrainingDataIndex,]
test <-bank[-TrainingDataIndex,]
NBModel <- train(train[,-20], train$y, method = "nb",trControl= trainControl(method =
"cv", number = 10, repeats = 5))
NBModel
```

After invoking the Naive Bayes method using training data set, lets feed test data to the model.

Testing the model

Below confusion matrix by class y shows that there is 89% accuracy in classification per Naive Bayes method.

Hide

Hide

```
NBPredictions <-predict(NBModel, test)
confusionMatrix(NBPredictions, test$y)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  no yes
      no  867  66
      yes  50  46

               Accuracy : 0.8873
                 95% CI : (0.8663, 0.9059)
    No Information Rate : 0.8912
    P-Value [Acc > NIR] : 0.6775

                  Kappa : 0.38
 Mcnemar's Test P-Value : 0.1637

            Sensitivity : 0.9455
            Specificity : 0.4107
         Pos Pred Value : 0.9293
         Neg Pred Value : 0.4792
             Prevalence : 0.8912
         Detection Rate : 0.8426
   Detection Prevalence : 0.9067
      Balanced Accuracy : 0.6781

       'Positive' Class : no
```

c) Suppor Vector Machines (SVM)

SVM is another classification method that can be used to predict if a client falls into either ‘yes’ or ‘no’ class.

Training the model

As before, create a prediction model using svmPoly method.

Hide

Hide

```
set.seed(120)
TrainingDataIndex <- createDataPartition(bank$y, p=0.75, list = FALSE)
train <- bank[TrainingDataIndex,]
test <-bank[-TrainingDataIndex,]
svm_model <- train(y~., data = train,
                   method = "svmPoly",
                   trControl= trainControl(method = "cv", number = 10, repeats = 5),
                   tuneGrid = data.frame(degree = 1,scale = 1,C = 1))
svm_model
```

```
Support Vector Machines with Polynomial Kernel

3090 samples
  19 predictor
   2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 2781, 2781, 2781, 2781, 2781, 2781, ...
Resampling results:

  Accuracy   Kappa
  0.9000019  0.2830744

Tuning parameter 'degree' was held constant at a value of 1
Tuning parameter 'scale' was held constant
 at a value of 1
Tuning parameter 'C' was held constant at a value of 1
```

After using polynomial kernal function to build a model, lets use test data to predict the accuracy of the model.

Testing the model

Hide

Hide

```
SVMPredictions <-predict(svm_model, test, na.action = na.pass)
confusionMatrix(SVMPredictions, test$y)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  no yes
       no  902  92
       yes  15  20

               Accuracy : 0.896
                 95% CI : (0.8757, 0.914)
    No Information Rate : 0.8912
    P-Value [Acc > NIR] : 0.33

                  Kappa : 0.2323
 Mcnemar's Test P-Value : 2.024e-13

            Sensitivity : 0.9836
            Specificity : 0.1786
         Pos Pred Value : 0.9074
         Neg Pred Value : 0.5714
             Prevalence : 0.8912
         Detection Rate : 0.8766
   Detection Prevalence : 0.9660
      Balanced Accuracy : 0.5811

       'Positive' Class : no
```

As evident, the SVM classfication method gives a 89.6% accuracy predicting only 15 instances of false positives.

d) Neural Network

Neural netowrks attempt to mimic the learning pattern of natural biological neural network. Lets use a neural network method to understand a customer's decision to open a bank account.

Training the model

Hide

Hide

```
set.seed(80)
TrainingDataIndex <- createDataPartition(bank$y, p=0.75, list = FALSE)
train <- bank[TrainingDataIndex,]
test <-bank[-TrainingDataIndex,]
nnmodel <- train(train[,-20], train$y, method = "nnet",
                 trControl= trainControl(method = "cv", number = 10, repeats = 5))
nnmodel
```

After training the model using nnet method, use a confusion matrix to evaluate the performanc eof the model on test data.

Testing the model

Hide

Hide

```
nnetpredictions <-predict(nnmodel, test, na.action = na.pass)
confusionMatrix(nnetpredictions, test$y)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  no yes
       no  898  76
       yes  19  36

               Accuracy : 0.9077
                 95% CI : (0.8883, 0.9247)
    No Information Rate : 0.8912
    P-Value [Acc > NIR] : 0.04685

                  Kappa : 0.3872
 Mcnemar's Test P-Value : 9.166e-09

            Sensitivity : 0.9793
            Specificity : 0.3214
         Pos Pred Value : 0.9220
         Neg Pred Value : 0.6545
             Prevalence : 0.8912
         Detection Rate : 0.8727
   Detection Prevalence : 0.9466
      Balanced Accuracy : 0.6504

       'Positive' Class : no
```

Based on the confusion matrix, there are only 19 instances of false positives. The neural network has a 90.8% accuracy.

2. Model Evaluation

We created four models above to classify whether a cutomer would open a bank account or not. Lets build some key performance indicators to understand which model is the most successful in predicting the customer's decision.

The typically used performance metrics are:

a. precision: success rate in identifying whether a customer did not subscibe to the deposit account
b. recall: proportion of clients correctly or incorrectly predicted to unsubscribe to an account

The classification goal is to predict whether or not customers will subscribe to a term deposit. Here the positive class is 'no' or that a customer does not subscribe to a deposit. Thus, it is important to choose a model with a low *recall*, i.e. the model that should contain a lower proportion of true positives (customers that did not subscribe to the deposit) out of total actual positives. If the bank aggressively determines those customers that do not subscribe to the bank account, the bank will lose some customers.

In order to illustrate recall and precision for each model, lets compute the weighted F-measure. The R output of the Confusion Matrix of each model already calculates recall and precision indicated by *sensitivity* and *Pos Pred Value* respectively. Thus, we can compute weighted F-measure (giving equal weights to reall and precision) as below. We collect *sensitivity* and *Pos Pred Value* from confusion matrix to compute F-measure for each model.

Hide

Hide

```
model = c("dec","nb","svm","nn")
recall = c(0.9935,0.9466,0.9836,0.9793)
precision = c(0.9038,0.9175,0.9074,0.9220)
fmeasure <- 2 * precision * recall / (precision + recall)
eval_table = data.frame(model,recall,precision,fmeasure)
eval_table
```

Based on the above table, Naive Bayes method is the recommended classification method as it contains lowest recall. We do not want a model that aggressively classifies a customer response as 'no', we want more customers to open a bank account.

3. Effect of PCA on Classification Performance

Since the bank dataset on telephone calls contains multiple variables, we can perform a principal component analysis (PCA), a dimensionality reduction technique, to reduce some of the variables with less variance, such that we can improve the model performances by focusing only on those attributes with relatively high variance.

As before, we will partition the data to test and training and perform each classification method to predict whether or not a customer will open a bank account. The pca function in *caret* package in R is used to perform dimensionality reduction which will exclude all categorical variables in the bank dataset.

Hide

Hide

```
TrainingDataIndex <- createDataPartition(bank$y, p=0.75, list = FALSE)
trainingData <- bank[TrainingDataIndex,]
testData <- bank[-TrainingDataIndex,]
```

Decision Tree

The decision tree model uses PCA to predict the class variable with an accuracy of 89.3%. This is slightly lower than the accuracy produced without performing dimensionalty reduction (89.9%). However, this model **DecTreeModel2** contains a higher precision, 91.3% compared to 90.4% of **DTPredictions**.

Hide

Hide

```
set.seed(30)
DecTreeModel2 <- train(trainingData[,-20], trainingData$y,
                       method = "C5.0",
                       trControl= trainControl(method = "cv", number = 10),
                       preProcess = c("pca"),
                       na.action = na.omit)
DTPredictions2 <-predict(DecTreeModel, testData, na.action = na.pass)
confusionMatrix(DTPredictions2, testData$y)
```

Naive Bayes

With PCA, naive bayes method produces a higher accuracy of 88.6% compared to the accuracy produced with PCA, 87.7%, thus this model predict a higher true negative rate (customers identified as opening a bank account) compared to the model without PCA. This model produces the same recall in comparison to the naive bayes model without PCA. The specificity is significantly higher than that from without PCA (39% versus 30%). Specificity is instances of true negative (44) as a proportion of true negative and false positive (44 + 68). In the banking campaigns, we want to minimize false positives, i.e. identifying class variable as ‘no’ when a customer actually wants to a bank account.

Hide

Hide

```
NBPredictions2 <-predict(NBModel2, testData, na.action = na.pass)
confusionMatrix(NBPredictions2, testData$y)
```

Support Vector Machine

When using PCA with SVM polynomial model, the accuracy improved from 89.6% to 90.3%. However, the model using PCA produced higher false positives (the model predicted a ‘no’ when a customer subscibed to an account) and thus SVM using PCA produced a higher precision, 91.1% versus 90.7%.

Hide

Hide

```
set.seed(40)
SVModel2 <- train(y ~ ., data = trainingData,
                  method = "svmPoly",
                  preProcess = c("pca"),
                  trControl= trainControl(method = "cv", number = 10),
                  tuneGrid = data.frame(degree = 1,
                                        scale = 1,
                                        C = 1))
SVpredictions2 <-predict(SVModel2, testData, na.action = na.pass)
confusionMatrix(SVpredictions2, testData$y)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  no yes
       no  901  97
       yes  16  15

               Accuracy : 0.8902
                 95% CI : (0.8695, 0.9086)
    No Information Rate : 0.8912
    P-Value [Acc > NIR] : 0.5647

                  Kappa : 0.1707
 Mcnemar's Test P-Value : 5.241e-14

            Sensitivity : 0.9826
            Specificity : 0.1339
         Pos Pred Value : 0.9028
         Neg Pred Value : 0.4839
             Prevalence : 0.8912
         Detection Rate : 0.8756
   Detection Prevalence : 0.9699
      Balanced Accuracy : 0.5582

       'Positive' Class : no
```

Neural Networks

When using PCA for neural network, the model produces a significantly lower accuracy 86% compared with 91% from neural network model without PCA. However, this model with PCA contains lower number of false positives and as such neural network classification with PCA is recommended over the neural network without PCA.

Hide

Hide

```
set.seed(30)
NNModel2 <- train(trainingData[,-20], trainingData$y,
                  method = "nnet",
                  preProcess = c("pca"),
                  trControl= trainControl(method = "cv", number = 10),
                  tuneGrid = data.frame(size = 5,
                  decay = 0))
```

Hide

Hide

```
NNpredictions2 <-predict(NNModel2, testData, na.action = na.pass)
confusionMatrix(NNpredictions2, testData$y)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  no yes
       no  917 112
       yes   0   0

               Accuracy : 0.8912
                 95% CI : (0.8705, 0.9095)
    No Information Rate : 0.8912
    P-Value [Acc > NIR] : 0.5251

                  Kappa : 0
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.8912
         Neg Pred Value :    NaN
             Prevalence : 0.8912
         Detection Rate : 0.8912
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : no
```