Code

Term deposit subscription prediction using bank data

Hari Hara Priya Kannan

Introduction

The aim of this project is to build a model to predict whether an individual will subscribe to a term deposit or not. The data is sourced from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets/Bank+Marketing).

Buisness Problem: There has been a revenue decline for the Portuguese bank and they would like to know what actions to take. After investigation, we found out that the root cause is that their clients are not depositing as frequently as before. Knowing that term deposits allow banks to hold onto a deposit for a specific amount of time, so banks can invest in higher gain financial products to make a profit. In addition, banks also hold better chance to persuade term deposit clients into buying other products such as funds or insurance to further increase their revenues. As a result, the Portuguese bank would like to identify existing clients that have higher chance to subscribe for a term deposit and focus marketing effort on such clients.

Methodology

We considered three classifiers - Naive Bayes (NB), Decision Tree , and Support Vector Machine (SVM). We split the full data set into 75% training set and 25% test set. Each set resembled the full data by having the same proportion of target classes i.e. approximately 90 % of individuals reponding ‘no’ and 10% reposponding ‘yes’ in the target variable. For fine-tuning process, we ran a ten-folded cross-validation stratified sampling on each classifier. We also study the effect of Principal Component Analysis on each of the classifiers.

Classification Methods

Load the data

All the necessary library packages are imported.

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)
```

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

```
bank <- read_delim("C:\\Sem2\\Machine Learning\\bank-additional\\bank-additional.csv",";", escape
_double = FALSE, trim_ws = TRUE)
head(bank)
```

... <int>	job <chr>	marital <chr>	education <chr>	default <chr>	housi... <chr>	loan <chr>	contact <chr>	mo... <chr>	day_of <chr>
30	blue-collar	married	basic.9y	no	yes	no	cellular	may	fri
39	services	single	high.school	no	no	no	telephone	may	fri
25	services	married	high.school	no	yes	no	telephone	jun	wed
38	services	married	basic.9y	no	unknown	unknown	telephone	jun	fri
47	admin.	married	university.degree	no	yes	no	cellular	nov	mon
32	services	single	university.degree	no	no	no	cellular	sep	thu

6 rows | 1-10 of 21 columns

Hide

```
bank <- na.omit(bank)
bank[, sapply( bank, is.character )] <- sapply( bank[, sapply( bank, is.character )], trimws)
```

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

Hide

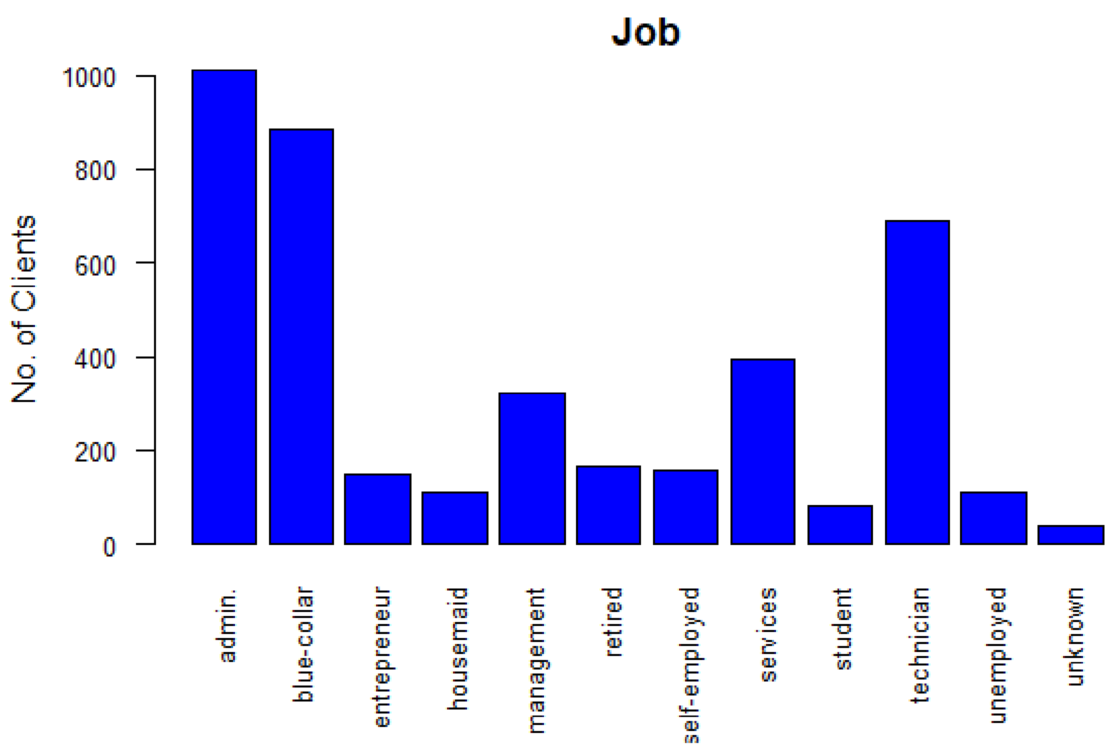
```
table(bank$y)
```

```
  no  yes
3668  451
```

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

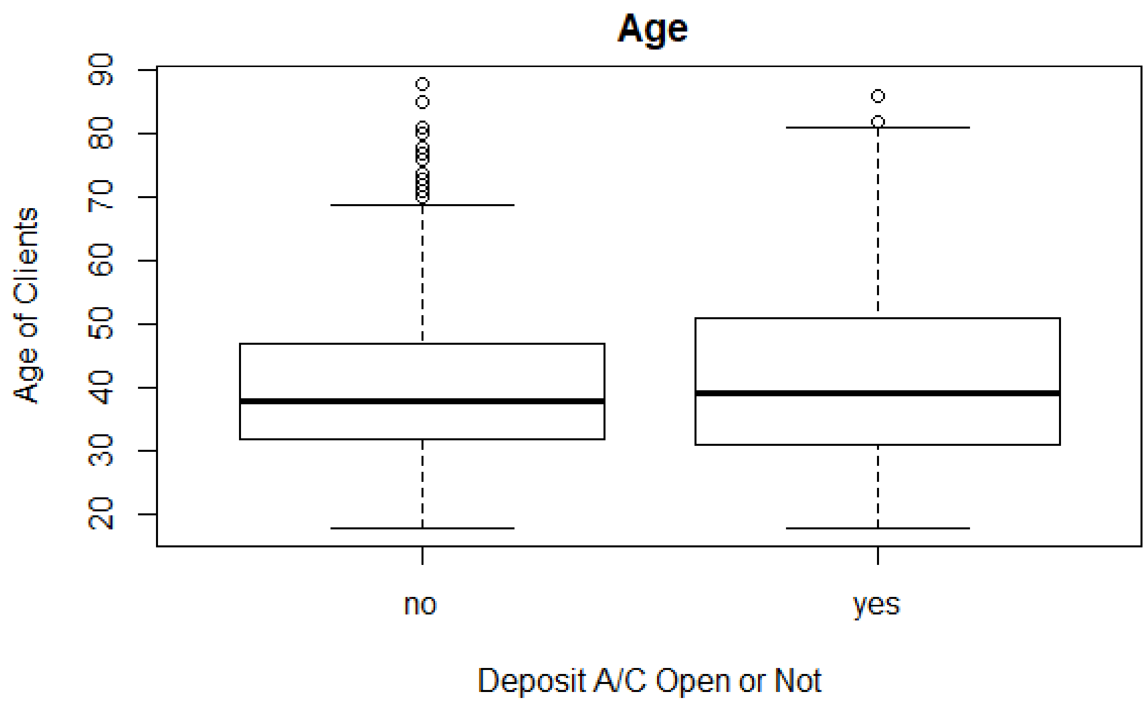
Hide

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



Hide

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



Splitting the dataset for training and testing

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

```
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

```
nrow(train)
```

```
[1] 3090
```

Hide

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

```
       no       yes
0.8911565 0.1088435
```

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.

Classification Methods

Decision Tree

Training the model

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

Hide

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

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

Hide

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

Let us take a look at the model.

Hide

```
DecTreeModel
```

```
C5.0

3090 samples
  20 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      0.9058282  0.3751454
  rules  FALSE   10      0.9158554  0.5038596
  rules  FALSE   20      0.9116514  0.4569425
  rules   TRUE    1      0.9051799  0.3708574
  rules   TRUE   10      0.9084151  0.4536081
  rules   TRUE   20      0.9100322  0.4580616
  tree   FALSE    1      0.9029156  0.4009655
  tree   FALSE   10      0.9097033  0.4473559
  tree   FALSE   20      0.9103548  0.4750974
  tree    TRUE    1      0.9045327  0.3887380
  tree    TRUE   10      0.9100353  0.4479091
  tree    TRUE   20      0.9106826  0.4656236

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were trials = 10, model = rules
 and winnow = FALSE.
```

Hide

```
summary(DecTreeModel)
```

```
Call:
(function (x, y, trials = 1, rules = FALSE, weights = NULL, control
 "winnow", "noGlobalPruning", "CF", "minCases",
 "fuzzyThreshold", "sample", "earlyStopping", "label", "seed")))


C5.0 [Release 2.07 GPL Edition]     Mon Jun 11 14:02:50 2018
-------------------------------

Class specified by attribute `outcome'

Read 3090 cases (54 attributes) from undefined.data

-----  Trial 0:  -----

Rules:

Rule 0/1: (2514/79, lift 1.1)
    duration <= 677
    poutcomesuccess <= 0
    nr.employed > 5076.2
    ->  class no  [0.968]

Rule 0/2: (2463/93, lift 1.1)
    duration <= 395
    poutcomesuccess <= 0
    ->  class no  [0.962]

Rule 0/3: (71/16, lift 7.0)
    duration > 395
    nr.employed <= 5076.2
    ->  class yes  [0.767]

Rule 0/4: (113/40, lift 5.9)
    poutcomesuccess > 0
    ->  class yes  [0.643]

Rule 0/5: (183/79, lift 5.2)
    duration > 677
    ->  class yes  [0.568]

Default class: no

-----  Trial 1:  -----

Rules:

Rule 1/1: (1782.4/67.6, lift 1.2)
    monthmar <= 0
    duration <= 395
    nr.employed > 5076.2
    ->  class no  [0.962]
```

```
Rule 1/2: (1147.6/48.8, lift 1.2)
    monthmar <= 0
    duration <= 163
    ->  class no  [0.957]

Rule 1/3: (2359.5/269.5, lift 1.1)
    age <= 57
    monthmar <= 0
    nr.employed > 5076.2
    ->  class no  [0.885]

Rule 1/4: (64.3/16.5, lift 3.6)
    monthmar > 0
    ->  class yes  [0.736]

Rule 1/5: (36.8/10.3, lift 3.5)
    age > 57
    duration > 395
    nr.employed > 5076.2
    ->  class yes  [0.708]

Rule 1/6: (417.6/160.3, lift 3.0)
    duration > 163
    nr.employed <= 5076.2
    ->  class yes  [0.615]

Default class: no

-----  Trial 2:  -----

Rules:

Rule 2/1: (2033.9/321.6, lift 1.2)
    monthoct <= 0
    duration <= 454
    cons.price.idx > 92.379
    ->  class no  [0.842]

Rule 2/2: (120.1/22, lift 1.1)
    age <= 25
    monthoct <= 0
    cons.price.idx > 92.379
    ->  class no  [0.811]

Rule 2/3: (2343.6/617.5, lift 1.0)
    educationbasic.6y <= 0
    previous <= 0
    ->  class no  [0.736]

Rule 2/4: (292.9/95.4, lift 2.4)
    age > 25
    jobretired <= 0
    educationbasic.6y <= 0
    monthoct <= 0
```

```
    duration > 454
    duration <= 697
    previous <= 0
    ->  class yes  [0.673]

Rule 2/5: (794.3/345.5, lift 2.0)
    monthoct <= 0
    duration > 454
    cons.price.idx > 92.379
    ->  class yes  [0.565]

Default class: no

-----  Trial 3:  -----

Rules:

Rule 3/1: (673.2/22, lift 1.5)
    duration <= 152
    euribor3m > 0.851
    ->  class no  [0.966]

Rule 3/2: (778.3/132.4, lift 1.3)
    age <= 74
    contacttelephone > 0
    duration <= 837
    ->  class no  [0.829]

Rule 3/3: (332.9/63.3, lift 1.2)
    educationbasic.9y > 0
    duration <= 837
    ->  class no  [0.808]

Rule 3/4: (1811.9/362.5, lift 1.2)
    age <= 74
    educationunknown <= 0
    duration <= 837
    euribor3m > 1.27
    ->  class no  [0.800]

Rule 3/5: (518.8/107.1, lift 1.2)
    jobblue-collar > 0
    duration <= 837
    ->  class no  [0.792]

Rule 3/6: (438.8/95.6, lift 1.2)
    age <= 74
    monthjul > 0
    duration <= 837
    ->  class no  [0.781]

Rule 3/7: (33.3/4.2, lift 2.5)
    duration > 152
    euribor3m > 1.51
```

```
    nr.employed <= 5099.1
    ->  class yes  [0.854]

Rule 3/8: (89.4/26.9, lift 2.0)
    duration <= 152
    euribor3m <= 0.851
    ->  class yes  [0.695]

Rule 3/9: (300/99.1, lift 2.0)
    duration > 837
    ->  class yes  [0.669]

Rule 3/10: (937.5/425, lift 1.6)
    jobblue-collar <= 0
    educationbasic.9y <= 0
    duration > 152
    nr.employed <= 5099.1
    ->  class yes  [0.547]

Default class: no

-----  Trial 4:  -----

Rules:

Rule 4/1: (567.9/80.4, lift 1.3)
    contacttelephone > 0
    monthmar <= 0
    duration <= 616
    pdays > 7
    cons.price.idx > 92.379
    ->  class no  [0.857]

Rule 4/2: (1256.4/237.9, lift 1.3)
    duration <= 228
    ->  class no  [0.810]

Rule 4/3: (61.2/11.8, lift 1.2)
    duration > 228
    pdays > 7
    cons.price.idx <= 92.379
    ->  class no  [0.798]

Rule 4/4: (110.1/24.2, lift 1.2)
    contacttelephone <= 0
    duration > 533
    duration <= 616
    pdays > 7
    ->  class no  [0.776]

Rule 4/5: (116.5/25.7, lift 1.2)
    jobself-employed > 0
    pdays > 7
    ->  class no  [0.775]
```

```
Rule 4/6: (1495.8/384.3, lift 1.2)
    monthmar <= 0
    day_of_weekwed <= 0
    duration <= 533
    pdays > 7
    cons.price.idx > 92.379
    ->  class no  [0.743]

Rule 4/7: (105.8/32.2, lift 1.9)
    duration > 228
    pdays <= 7
    ->  class yes  [0.692]

Rule 4/8: (36.7/11.3, lift 1.9)
    monthmar > 0
    duration > 228
    duration <= 616
    ->  class yes  [0.682]

Rule 4/9: (144.3/48.5, lift 1.8)
    contacttelephone <= 0
    day_of_weekwed > 0
    duration > 228
    duration <= 533
    cons.price.idx > 92.379
    ->  class yes  [0.662]

Rule 4/10: (596.2/243.1, lift 1.7)
    jobself-employed <= 0
    duration > 616
    cons.price.idx > 92.379
    ->  class yes  [0.592]

Default class: no

-----  Trial 5:  -----

Rules:

Rule 5/1: (251.1/10, lift 1.5)
    duration <= 88
    ->  class no  [0.957]

Rule 5/2: (2838.9/1166.3, lift 1.0)
    duration > 88
    ->  class no  [0.589]

Rule 5/3: (42.4/7.5, lift 2.1)
    age > 74
    duration > 88
    ->  class yes  [0.809]

Rule 5/4: (28.5/6.9, lift 1.9)
```

```
    educationbasic.6y > 0
    monthmay <= 0
    duration > 88
    nr.employed <= 5099.1
    ->  class yes  [0.741]

Rule 5/5: (30.1/8, lift 1.9)
    monthdec > 0
    duration > 88
    ->  class yes  [0.719]

Rule 5/6: (122.9/46.1, lift 1.6)
    jobretired <= 0
    defaultunknown > 0
    monthmay <= 0
    duration > 366
    nr.employed > 5099.1
    ->  class yes  [0.623]

Rule 5/7: (490.6/185.5, lift 1.6)
    age <= 74
    educationbasic.6y <= 0
    monthmay <= 0
    day_of_weekwed <= 0
    duration > 88
    poutcomenonexistent > 0
    nr.employed <= 5099.1
    ->  class yes  [0.621]

Default class: no

-----  Trial 6:  -----

Rules:

Rule 6/1: (443.5/53.1, lift 1.5)
    duration <= 127
    ->  class no  [0.879]

Rule 6/2: (123.7/21.9, lift 1.4)
    age > 32
    jobmanagement > 0
    previous <= 0
    ->  class no  [0.818]

Rule 6/3: (2768.6/1011, lift 1.1)
    pdays > 15
    ->  class no  [0.635]

Rule 6/4: (182.4/50.1, lift 1.8)
    age > 28
    duration > 361
    campaign <= 5
    previous > 0
```

```
    ->  class yes  [0.723]

Rule 6/5: (280.5/87.2, lift 1.7)
    duration > 127
    pdays <= 15
    ->  class yes  [0.688]

Rule 6/6: (237.4/77.2, lift 1.7)
    age > 28
    age <= 32
    jobself-employed <= 0
    duration > 361
    campaign <= 5
    ->  class yes  [0.673]

Rule 6/7: (424.4/164.9, lift 1.5)
    age > 28
    jobself-employed <= 0
    maritalmarried <= 0
    duration > 361
    campaign <= 5
    ->  class yes  [0.611]

Rule 6/8: (1334.8/634.6, lift 1.3)
    duration > 361
    campaign <= 5
    ->  class yes  [0.525]

Default class: no

-----  Trial 7:  -----

Rules:

Rule 7/1: (479.9/32.6, lift 1.5)
    duration <= 152
    ->  class no  [0.930]

Rule 7/2: (1181.4/149.9, lift 1.4)
    monthmar <= 0
    duration <= 637
    euribor3m > 1.281
    ->  class no  [0.872]

Rule 7/3: (555.4/115.6, lift 1.3)
    contacttelephone > 0
    monthmar <= 0
    duration <= 797
    euribor3m > 0.778
    ->  class no  [0.791]

Rule 7/4: (78.1/17.5, lift 1.2)
    duration > 152
    euribor3m > 0.731
```

```
    euribor3m <= 0.778
    ->  class no  [0.769]

Rule 7/5: (183.2/41.8, lift 1.2)
    jobservices > 0
    monthmar <= 0
    euribor3m > 1.281
    ->  class no  [0.769]

Rule 7/6: (193.2/48, lift 2.0)
    monthmar <= 0
    duration > 152
    euribor3m <= 0.731
    ->  class yes  [0.749]

Rule 7/7: (89.9/28.5, lift 1.8)
    monthmar > 0
    duration > 152
    ->  class yes  [0.679]

Rule 7/8: (589.7/247.1, lift 1.6)
    jobservices <= 0
    monthmar <= 0
    duration > 637
    euribor3m > 1.281
    ->  class yes  [0.581]

Rule 7/9: (938.6/408.4, lift 1.5)
    contacttelephone <= 0
    duration > 152
    euribor3m <= 1.281
    ->  class yes  [0.565]

Rule 7/10: (726.3/317.3, lift 1.5)
    monthmar <= 0
    duration > 637
    ->  class yes  [0.563]

Default class: no

-----  Trial 8:  -----

Rules:

Rule 8/1: (413/4.5, lift 1.4)
    duration <= 152
    ->  class no  [0.987]

Rule 8/2: (1343.8/92, lift 1.3)
    age <= 74
    duration <= 677
    nr.employed > 5076.2
    ->  class no  [0.931]
```

```
Rule 8/3: (220.9/17.3, lift 1.3)
    educationbasic.9y > 0
    duration <= 677
    ->  class no  [0.918]

Rule 8/4: (1592.7/285.4, lift 1.2)
    age <= 74
    monthdec <= 0
    duration <= 401
    ->  class no  [0.820]

Rule 8/5: (165.9/36.2, lift 1.1)
    age <= 74
    jobretired > 0
    ->  class no  [0.778]

Rule 8/6: (1979.4/460.7, lift 1.1)
    age <= 48
    cons.price.idx > 92.843
    ->  class no  [0.767]

Rule 8/7: (91.8/17.6, lift 3.0)
    educationbasic.9y <= 0
    monthdec <= 0
    duration > 401
    duration <= 677
    nr.employed <= 5076.2
    ->  class yes  [0.802]

Rule 8/8: (41.6/9.8, lift 2.8)
    age > 74
    duration > 152
    ->  class yes  [0.752]

Rule 8/9: (107.5/26.9, lift 2.8)
    age > 48
    jobretired <= 0
    duration > 677
    cons.price.idx > 92.843
    ->  class yes  [0.745]

Rule 8/10: (29.5/7.8, lift 2.7)
    monthdec > 0
    duration > 152
    ->  class yes  [0.723]

Default class: no

-----  Trial 9:  -----

Rules:

Rule 9/1: (490.3/15.2, lift 1.5)
    duration <= 172
```

```
    ->  class no  [0.967]

Rule 9/2: (107.8/4, lift 1.5)
    campaign > 5
    pdays > 21
    ->  class no  [0.955]

Rule 9/3: (102.4/7.1, lift 1.5)
    educationunknown > 0
    pdays > 21
    euribor3m > 0.715
    ->  class no  [0.922]

Rule 9/4: (1594.1/141.3, lift 1.4)
    duration <= 679
    pdays > 21
    ->  class no  [0.911]

Rule 9/5: (1674.9/177.2, lift 1.4)
    educationbasic.6y <= 0
    duration <= 837
    pdays > 21
    euribor3m > 0.715
    ->  class no  [0.894]

Rule 9/6: (47.4, lift 3.2)
    educationbasic.6y > 0
    duration > 679
    ->  class yes  [0.980]

Rule 9/7: (29.6, lift 3.1)
    duration > 679
    euribor3m <= 0.715
    ->  class yes  [0.968]

Rule 9/8: (468.8/84.5, lift 2.7)
    duration > 172
    pdays <= 21
    ->  class yes  [0.818]

Rule 9/9: (504.5/152, lift 2.3)
    educationunknown <= 0
    duration > 837
    campaign <= 5
    ->  class yes  [0.698]

Default class: no


Evaluation on training data (3090 cases):

Trial           Rules
-----     ----------------
        No      Errors
```

```
   0         5  272( 8.8%)
   1         6  299( 9.7%)
   2         5  411(13.3%)
   3        10  311(10.1%)
   4        10  330(10.7%)
   5         7  429(13.9%)
   6         8  341(11.0%)
   7        10  330(10.7%)
   8        10  273( 8.8%)
   9         9  260( 8.4%)
boost           218( 7.1%)   <<


       (a)   (b)    <-classified as
      ----  ----
      2725    26    (a): class no
       192   147    (b): class yes


    Attribute usage:

    100.00% duration
     99.94% euribor3m
     99.55% age
     99.55% pdays
     97.80% cons.price.idx
     97.35% monthmar
     97.31% nr.employed
     95.83% monthoct
     93.27% poutcomesuccess
     90.10% educationbasic.6y
     83.82% previous
     83.72% monthdec
     83.56% educationunknown
     72.23% day_of_weekwed
     49.22% contacttelephone
     36.70% jobblue-collar
     29.22% educationbasic.9y
     27.15% campaign
     16.15% monthjul
     15.63% jobself-employed
     13.37% jobservices
     12.01% jobretired
      9.42% monthmay
      7.06% poutcomenonexistent
      5.57% maritalmarried
      5.02% jobmanagement
      1.88% defaultunknown


Time: 0.4 secs
```

Testing the Model

Based on confusion matrix for test data, using the decision tree model we have correctly classified 901 + 40 = 941 observations and misclassified 16 + 40 = 56 representing a 91% accuracy.

Hide

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

```
Confusion Matrix and Statistics


DTPredictions  no yes
          no  901  72
          yes  16  40

               Accuracy : 0.9145
                 95% CI : (0.8957, 0.9308)
    No Information Rate : 0.8912
    P-Value [Acc > NIR] : 0.007798

                  Kappa : 0.4352
 Mcnemar's Test P-Value : 4.545e-09

            Sensitivity : 0.9826
            Specificity : 0.3571
         Pos Pred Value : 0.9260
         Neg Pred Value : 0.7143
             Prevalence : 0.8912
         Detection Rate : 0.8756
   Detection Prevalence : 0.9456
      Balanced Accuracy : 0.6698

       'Positive' Class : no
```

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 all the 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

```
NBModel <- train(train[,-20], train$y, method = "nb",trControl= trainControl(method = "cv", numbe
r = 10, repeats = 5))
NBModel
```

```
Naive Bayes

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

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

  usekernel  Accuracy   Kappa
  FALSE      0.8844626  0.4239128
   TRUE      0.9019415  0.3593001

Tuning parameter 'fL' was held constant at a value of 0
Tuning
 parameter 'adjust' was held constant at a value of 1
Accuracy was used to select the optimal model using the largest value.
The final values used for the model were fL = 0, usekernel = TRUE
 and adjust = 1.
```

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

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

```
Confusion Matrix and Statistics


NBPredictions  no yes
          no  899  87
          yes  18  25

               Accuracy : 0.898
                 95% CI : (0.8778, 0.9158)
    No Information Rate : 0.8912
    P-Value [Acc > NIR] : 0.2601

                  Kappa : 0.279
 Mcnemar's Test P-Value : 3.22e-11

            Sensitivity : 0.9804
            Specificity : 0.2232
         Pos Pred Value : 0.9118
         Neg Pred Value : 0.5814
             Prevalence : 0.8912
         Detection Rate : 0.8737
   Detection Prevalence : 0.9582
      Balanced Accuracy : 0.6018

       'Positive' Class : no
```

Support Vector Machines

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

```
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
  20 predictor
   2 classes: 'no', 'yes'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 2781, 2781, 2781, 2781, 2780, 2781, ...
Resampling results:

  Accuracy   Kappa
  0.9084099  0.4400775

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

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

```
Confusion Matrix and Statistics


SVMPredictions  no yes
           no  897  75
           yes  20  37

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

                  Kappa : 0.3933
 Mcnemar's Test P-Value : 3.02e-08

            Sensitivity : 0.9782
            Specificity : 0.3304
         Pos Pred Value : 0.9228
         Neg Pred Value : 0.6491
             Prevalence : 0.8912
         Detection Rate : 0.8717
   Detection Prevalence : 0.9446
      Balanced Accuracy : 0.6543

       'Positive' Class : no
```

Model Evaluation

We created three 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:

precision: success rate in identifying whether a customer did not subscibe to the deposit account 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

```
model = c("dec","nb","svm")
recall = c(0.9826, 0.9706, 0.9760)
precision = c(0.9260, 0.9242, 0.9284)
fmeasure <- 2 * precision * recall / (precision + recall)
eval_table = data.frame(model,recall,precision,fmeasure)
eval_table
```

model <fctr>	recall <dbl>	precision <dbl>	fmeasure <dbl>
dec	0.9826	0.9260	0.9534608
nb	0.9706	0.9242	0.9468319
svm	0.9760	0.9284	0.9516051

3 rows

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.

Tuning with Principal Component Analysis

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

```
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

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

```
Confusion Matrix and Statistics


DTPredictions2  no yes
           no  903  66
           yes  14  46

               Accuracy : 0.9223
                 95% CI : (0.9042, 0.9379)
    No Information Rate : 0.8912
    P-Value [Acc > NIR] : 0.0005051

                  Kappa : 0.4967
 Mcnemar's Test P-Value : 1.184e-08

            Sensitivity : 0.9847
            Specificity : 0.4107
         Pos Pred Value : 0.9319
         Neg Pred Value : 0.7667
             Prevalence : 0.8912
         Detection Rate : 0.8776
   Detection Prevalence : 0.9417
      Balanced Accuracy : 0.6977

       'Positive' Class : no
```

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

```
NBModel2 <- train(trainingData[,-20], trainingData$y, method = "nb",trControl= trainControl(metho
d = "cv", number = 10, repeats = 5))
NBPredictions2 <-predict(NBModel2, testData, na.action = na.pass)
confusionMatrix(table(NBPredictions2, testData$y))
```

```
Confusion Matrix and Statistics


NBPredictions2  no yes
           no  887  76
           yes  30  36

               Accuracy : 0.897
                 95% CI : (0.8768, 0.9149)
    No Information Rate : 0.8912
    P-Value [Acc > NIR] : 0.2941

                  Kappa : 0.3522
 Mcnemar's Test P-Value : 1.238e-05

            Sensitivity : 0.9673
            Specificity : 0.3214
         Pos Pred Value : 0.9211
         Neg Pred Value : 0.5455
             Prevalence : 0.8912
         Detection Rate : 0.8620
   Detection Prevalence : 0.9359
      Balanced Accuracy : 0.6444

       'Positive' Class : no
```

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

```
set.seed(40)
TrainingDataIndex <- createDataPartition(bank$y, p=0.75, list = FALSE)
trainingData <- bank[TrainingDataIndex,]
testData <- bank[-TrainingDataIndex,]
SVModel2 <- train(y ~ ., data = trainingData,
                 method = "svmPoly",
                 preProcess = c("pca"),
                 trControl= trainControl(method = "cv", number = 10),
                 tuneGrid = data.frame(degree = 1,
                                       scale = 1,
                                       C = 1))
SVpredictions2 <-predict(SVModel2, testData, na.action = na.pass)
confusionMatrix(table(SVpredictions2, testData$y))
```

```
Confusion Matrix and Statistics


SVpredictions2  no yes
           no  896  67
           yes  21  45

               Accuracy : 0.9145
                 95% CI : (0.8957, 0.9308)
    No Information Rate : 0.8912
    P-Value [Acc > NIR] : 0.007798

                  Kappa : 0.4622
 Mcnemar's Test P-Value : 1.61e-06

            Sensitivity : 0.9771
            Specificity : 0.4018
         Pos Pred Value : 0.9304
         Neg Pred Value : 0.6818
             Prevalence : 0.8912
         Detection Rate : 0.8707
   Detection Prevalence : 0.9359
      Balanced Accuracy : 0.6894

       'Positive' Class : no
```

Discussion

The previous section showed that all classifiers did not perform accurately in predicting the term deposit subcribers despite the stratified sampling. This implies the imbalance class problem was prevalent.The NB model assumes the descriptive features to follow normality that are not necessarily true. The solution would be a transformation on numeric features.

Conclusion

Among three classifiers, the Naive Bayes produces the best performance in predicting if an individual will subscribe to a term deposit or not. We split the data into training and test sets. After using the PCA, we observed that even though there is no improvement in the recall value, the method produces an higher accuracy. Also, the accuracy of all the models are almost close to each other. We can try to create an ensemble model to see if there is a significant improvement in the performance of the model.