**SAN DIEGO STATE UNIVERSITY**



**FINAL PROJECT**

**Bank Telemarketing Analysis**

**By,**

**Snehal Ilawe**

**824655189**

|  |  |  |
| --- | --- | --- |
| S.NO | Content | Page |
|  | Executive Summary | 3 |
|  | Discovery and Data Preparation | 6 |
|  | Model Planning and Building | 10 |
|  | Results and Performance | 13 |
|  | Discussion and recommendation | 20 |

1. Appendix (R Code) 23

**Executive Summary**

Marketing campaigns in banking sectors are strategized efforts to develop new customers and better engage existing customers. By leveraging Data Analytics, banks can evaluate their current marketing techniques and can identify patterns to maximize their effectiveness and build greater customer satisfaction. The purpose of the project is to identify main factors that contribute to the success of a campaign and successfully predict if a client would subscribe to a long-term deposit, using various classifications models. This project will enable the bank to understand its customer base and devise better marketing plans in the future to increase client subscription to its long-term deposit service.

The data analyzed in this project, consists of 41188 observations and 21 attributes that were collected via phone calls by the Portuguese Bank Institution during one of their direct marketing campaigns. The response is whether the client subscribed to a long-term deposit. For computational purposes, only 10000 rows of the above data were selected randomly by checking for their significance. Significance was calculated using Kolmogorov-Smirnov test for numerical data and the Pearson chi-square test for categorical data. While, there were some missing (unknown) values in attributes such as loan, housing, default, and marital status, the number of such observations were minimal (2.4%), and hence these observations were dropped. Additionally, attributes such as default and pdays (number of days that passed by after the client was last contacted from a previous campaign), that had near zero variance were dropped. Next, the distribution of the target data was highly imbalanced; 87% of the data was “No” and only 12% of data was “Yes”. To handle such high imbalance SMOTE technique was used to balance the data. Finally, the duration attribute (last contact duration), was ignored since the call duration is not known before a call is performed and hence would not logically contribute towards the models.

To predict whether a client will subscribe to long term deposit, different classification techniques were applied, such as Logistic Regression, Linear Discriminant Analysis (LDA), Decision Trees, Random Forest, SVM using radical kernel, Gradient Boosting Machine (GBM), and k-NN. Parameter tuning was performed using 10 fold cross-validation and tuning parameter grid. A random sample of 30% of data was used as test dataset and rest for training the models. Performance metrics such as ROC and test accuracy were used to evaluate model performance.

Table 1: Test Accuracy

|  |  |
| --- | --- |
| **Models** | **Test Accuracy** |
| Logistic Regression | 0.8450 |
| LDA | 0.8583 |
| GBM | 0.8828 |
| KNN | 0.8947 |
| RF | 0.8763 |
| SVM | 0.8457 |
| Decision Trees | 0.8681 |

A close up of a device

Description automatically generatedA screenshot of a social media post

Description automatically generated

Figure a: ROC for all Models Figure 2b: Attribute Importance

From Table 1 and Fig. (a) it can be noted that, the best model considering overall test accuracy is KNN, followed by GBM and Random Forest. By ROC metric Random Forest is better at distinguishing between the two classes, followed by GBM and Decision Trees. As per Fig. 1(b), the most important factors that influence customers decision are nr.employed (number of employees), previous (number of previous communication), emp.var.rate, euribor3m , cons.conf.idx (consumer confidence index), and consumer price index (cons.price.idx).

Therefore, to increase subscription to their long-term deposit service and to ensure the success of their marketing campaign, the Portuguese Bank Institute should focus on improving the quality and frequency of customer interaction. Other important factors for the bank to consider would be to time their campaigns when interest rates are high and the macroeconomic environment is stable.

**Discovery and Data Preparation**

Marketing campaigns in banking sectors are strategized efforts to develop better and engaging marketing campaigns. But over time it has reduced its effect on customers. Of the people contacted less than 1% of them actually subscribe to the long-term deposit. By leveraging Data Analytics, banks can evaluate their current marketing techniques and can identify patterns and anomalies that can help develop a better marketing plan to maximize its effectiveness and build greater customer satisfaction. It can also help identify the customer base, such as its demographics and customer history to analyze what factors influence clients to subscribe to long-term deposit.

The data consists of direct marketing campaigns via phone calls of a Portuguese Bank Institution for 41188 clients and 20 features. The response variable y is whether the client has subscribed to long-term deposit. The dataset is from public library at UCI Machine Learning repository <http://archive.ics.uci.edu/ml/datasets/bank+marketing>

Following is the Attribute Information:

|  |  |  |
| --- | --- | --- |
| Age | Numeric | Age of Customer |
| Job | Categorical | Type of Job |
| Marital | Categorical | Marital State(Married,Single,Divorced) |
| Educational | Categorical | Different Education Levels |
| Default | Categorical | If customer has credit in default |
| Housing | Categorical | If customer has housing loan |
| Loan | Categorical | If customer has any personal loan |
| Contact | Categorical | Contact communication type |
| Month | Categorical | Last contact Month |
| DayofWeek | Categorical | Last contact day of week |
| Duration | Numeric | Last contact duration |
| Campaign | Numeric | number of contacts performed during this campaign |
| Pdays | Numeric | number of days that passed by after the client was last contacted from a previous campaign |
| Previous | Numeric | number of contacts performed before this campaign and for this client |
| Poutcome | Categorical | outcome of the previous marketing campaign |
| Emp.var.rate | Numeric | employment variation rate - quarterly indicator |
| Cons.price.idx | Numeric | consumer price index - monthly indicator |
| Cons.conf.idx | Numeric | consumer confidence index - monthly indicator |
| Euribor3m | Numeric | euribor 3 month rate - daily indicator |
| Nr.employed | Numeric | number of employees - quarterly indicator |

The size of the data is large so for computational purpose random sample of 10000 rows

were selected by checking for their significance. Numerical data significance was calculated using Kolmogorov-Smirnov test and for the categorical data Pearson chi-square test was used. Sample with p-value greater than 0.5 was selected which is representative of the larger sample.

There were some missing(unknown) values in attributes such as loan, housing, default, marital but since the number of observations with such missing values were minimal (2.4%) these observations were dropped. The attribute default (Fig 3) has Non-Zero variance it has 7930 entries as No and 0 as Yes and 2070 as unknown. Since there is no variance, we cannot impute the missing values and so the attribute was removed.

A screenshot of a social media post

Description automatically generated

Figure 3: default Distribution

The attribute pdays (number of days that passed by after the client was last  
contacted from a previous campaign) has nonzero variance it has 96% data as customer was not contacted for previous campaign and also there are other attributes such as DayofWeek and Month which indicate the last contact day and month.

A screenshot of a cell phone

Description automatically generated

Figure 4:Pdays Distribution

There is a strong correlation between attributes emp.var.rate (employment variation),euribor3m(Euro Interbank Offered Rate), nr.employed (number of employees).

|  |  |  |  |
| --- | --- | --- | --- |
|  | emp.var.rate | euribor3m | nr.employed |
| emp.var.rate | 1.0000000 | 0.9718532 | 0.9072454 |
| euribor3m | 0.9718532 | 1.0000000 | 0.9451391 |
| nr.employed | 0.9072454 | 0.9451391 | 1.0000000 |

But all the three variables are significant in determining the target variable. The parameters and their effect will be further studied by fitting models.

From Fig 5, we can see that the data is highly imbalanced 87% of the data has response variable as “No” and only 12% of data is “Yes”.

A screenshot of a social media post

Description automatically generated

Figure 5:Target Attribute Distribution

Classification using such imbalanced data is always biased towards the majority class. To handle such high imbalance SMOTE (Synthetic Minority Oversampling TEchnique ) technique was used on the train set of data to balance the data. SMOTE is a hybrid technique that can be used to up or down the samples. Here SMOTE was used to up the minority samples. After Smote now we have 2912 with Yes and 4368 rows with No as the response.

**Model Planning and Building**

To predict if a client would subscribe to a long-term deposit, Logistic Regression, Logistic Regression with Principal Component Analysis, Linear Discriminant Analysis, Decision Tree, Random Forest, Random Forest with PCA, Support Vector Machine, Gradient Boosting and KNN were used. All the models are trained using the train() function of Caret package. Before training the models we set the computational parameters using trainControl(). We set 3 parameters, the method parameter is set to “cv” which indicates we are validating the models using Cross-Validation to choose the best metrics. The number parameter holds the number of folds here its is set to have 10 fold Cross validation.

To evaluate the models, popular classification techniques such as confusion metrics, ROC curve, Accuracy were calculated. Train set consist of 70% of the data and rest 30% consist of test data. Attribute Duration (last contact duration, in seconds (numeric) highly affects the output target (e.g., if duration=0 then y='no'). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, excluding duration to have a realistic predictive model.

1. **Logistic Regression with 10 fold cross validation:**

Bank marketing dataset has binary predictor either a client subscribes to long-term deposit or does not. Logistics model doesn’t suffer a lot from severe class imbalance. Logistic Regression with 10 fold cross validation with metric as “ROC” was implemented first by using all the predictors, then using Principal component analysis as there is multicollinearity and then implemented by removing those correlated attributes. Confusion matrix for all the misclassifications in test prediction are calculated along with Test Accuracy.

1. **Linear Discriminant Analysis:**

LDA provides discrimination between two or more conditions/classes. LDA was implemented using all the attributes with 10 fold cross validation and ROC as metric. Confusion matrix for all the misclassifications in test prediction are calculated along with Test Accuracy.

1. **Decision Tree:**

Trees provide visual tool that are easy to interpret and understand, understanding attributes that are important for making campaign a success is important. Decision Tree is implemented using Caret package rpart using ROC metric and with 10 fold cross validation and tuneGrid to select the optimal cp. Best cp is the one that maximize the cross-validation accuracy which can be estimated by testing different cp values and using cross-validation.

1. **Random forest:**

Random Forest does not allow overfitting and provides with higher accuracy. For bank telemarketing data predicting higher accuracy is important. Random Forest also provides with list of most important predictors that are useful to determine if a customer will subscribe to long-term deposit. Choosing good tuning parameter helps getting better results. Random Forest is implemented using Caret package with ROC metric and 10 fold cross validation and tuneGrid to estimate what is the best mtry value to get optimal results.

1. **Support Vector Machine:**

SVM is implemented using Caret package svmRadial with ROC metric and 10 fold cross validation and tuneGrid to try different C and sigma values. We also preprocess that data to center and scale before training.

1. **Gradient Boosting Machine:**

In boosting, the individual models are not built on completely random subsets of data and features but sequentially by putting more weight on instances with wrong predictions and high errors. GBM is implemented using Caret package gbm with 10 fold cross validation and tuneGrid parameter to get optimal model. We also preprocess that data to center and scale before training.

1. **K- Nearest Neighbor:**

Regular linear regression makes assumptions about the structure of the data (high bias), but its predictions are stable (low variance). We need a more flexible model that makes fewer assumptions. In contrast to linear regression methods, the k-nearest neighbor methods implement non-linear boundaries to our training and test data. KNN is implemented using Caret package kNN with 10 fold cross validation and tuneGrid parameter to get optimal model.. We also preprocess that data to center and scale before training.

**Results and Performance**

1. **Logistic Regression:**

Logistic Regression was run using different methods and 10-fold cross validation.

From Table 2, we can see that Logistic Regression performs better (Test Accuracy of 0.845 and AUC of 0.7492) using all the features. For both training and test data Logistic Regression with all features can provide better accuracy in predicting whether or not subscriber will subscribe to long-term deposits.

Table 2 Test and Train ROC and Accuracy

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Train ROC** | **Test ROC** | **Test Accuracy** |
| Logistic Regression | 0.8224775 | 0.7927 | 0.845 |
| Logistic Regression PCA | 0.8149573 | 0.7462 | 0.8429 |
| Logistic Regression with selected features | 0.821238 | 0.7461 | 0.8425 |

**For Test Set:**

Logistic Regression Confusion Matrix: Logistic Regression using PCA

|  |  |  |
| --- | --- | --- |
|  | Reference | |
| Prediction | yes | no |
| yes | 162 | 281 |
| no | 149 | 2183 |

|  |  |  |
| --- | --- | --- |
|  | Reference | |
| Prediction | yes | no |
| yes | 154 | 279 |
| no | 157 | 2185 |

Logistic Regression using Selective Features:

|  |  |  |
| --- | --- | --- |
|  | Reference | |
| Prediction | yes | no |
| yes | 157 | 283 |
| no | 154 | 2181 |

1. **Linear Discriminant Analysis:**

LDA provides better Test Accuracy (0.8584) and ROC (0.7559) than Logistic Regression. It can 90% correctly classify the client will not subscribe to long-term deposit.

Table 3: LDA Test and Train ROC

|  |  |  |  |
| --- | --- | --- | --- |
|  | ROC | Sens | Spec |
| Training | 0.8214482 | 0.6067916 | 0.9029297 |
| Test | 0.7559 | 0.5176 | 0.9013 |

**Test Confusion Matrix**

|  |  |  |
| --- | --- | --- |
|  | Reference | |
| Prediction | yes | no |
| yes | 161 | 243 |
| no | 150 | 2221 |

**Test Accuracy: 0.8584**

1. **Decision Tree**

Decision Tree with ROC metric and 10-fold cross validation with tuneGrid for 25 different

value of cp ranging from 0.01 to 0.1 was modeled. The final value used for the model was cp = 0.01. From Table 4: Decision Tree provides an Test accuracy of 0.8681 which is higher than Logistic Regression and LDA. Decision Trees can 91% correctly classify the client will not subscribe to long-term deposit. From Fig 6 we can visualize and identify the important attributes to determine y.

Table 4:Decision Tree Test and Train metrics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | ROC | Sens | Spec | Accuracy |
| Train | 0.8993 | 0.8303 | 0.9171 |  |
| Test | 0.7739 | 0.5113 | 0.9131 | 0.8681 |

**Test Confusion Matrix**

|  |  |  |
| --- | --- | --- |
|  | Reference | |
| Prediction | yes | no |
| yes | 159 | 214 |
| no | 152 | 2250 |

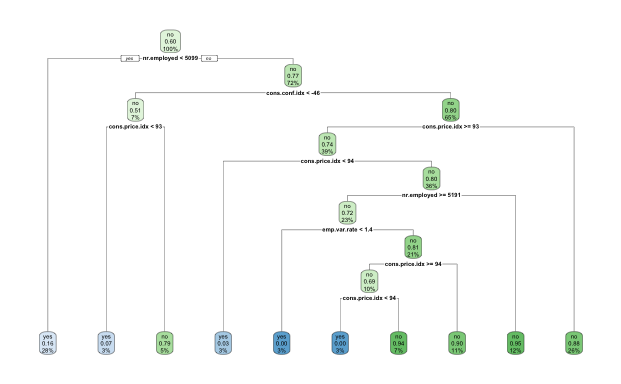


Figure 6: Decision Tree

1. **Random Forest:**

Random Forest with ROC metric and 10 fold cross validation with tuneGrid for 20 different values of mtry was modeled. The final value used for the model was mtry = 12 (Fig 7).

A close up of a map

Description automatically generated

Figure7: Number of Trees Figure : Attribute Importance

A screenshot of a social media post

Description automatically generated

From Fig 8, we can determine 5 of most important attributes as nr.employed, euribor3m, cons.conf.idx, cons.price.idx, previous. It provides a Test Accuracy of 0.876 which is best among all the previous implemented models. It can 93% times predict if the client will not subscribe to deposit.

**Test Confusion Matrix**

|  |  |  |
| --- | --- | --- |
|  | Reference | |
| Prediction | yes | no |
| yes | 131 | 163 |
| no | 180 | 2301 |

**Test result:**

|  |  |  |  |
| --- | --- | --- | --- |
| ROC | Sens | Spec | Accuracy |
| 0.7473 | 0.421 | 0.933 | 0.876 |

1. **SVM:**

SVM using 10-fold cross validation and center and scaling of parameter along with ROC metric with tuneGrid parameter to select best model from different tuning parameter of C and sigma.

From fig 9: The final values used for the model were sigma = 0.015 and C = 10

A close up of a map

Description automatically generated

Figure 9: SVM with different sigma

SVM provided Test Accuracy of 0.8458 and could 89% times predict if the client will not subscribe to deposit.

**Test** Confusion Matrix:

|  |  |  |
| --- | --- | --- |
|  | Reference | |
| Prediction | yes | no |
| yes | 149 | 266 |
| no | 162 | 2198 |

Test Results:

|  |  |  |  |
| --- | --- | --- | --- |
| ROC | Sens | Spec | Accuracy |
| 0.7292 | 0.4791 | 0.8920 | 0.8458 |

1. **GBM:**

GBM was run with ROC metric, 10 fold cross validation and using tunegrid for tuning different parameters to find the optimal model. From Fig 10, we can see that 5 is the optimal Tree depth. GBM provided with Test accuracy of 0.8829. It can 94% times predict if the client will not subscribe to deposit. Important Variables extracted from GBM are as follows:

|  |  |
| --- | --- |
| nr.employed | 42.52 |
| previous | 15.66 |
| cons.conf.idx | 15.21 |
| pdays | 10.47 |
| cons.price.idx | 5.12 |

**A close up of a map

Description automatically generated**

Figure 10: Max depth

**Test Results** Confusion Matrix

|  |  |  |
| --- | --- | --- |
|  | Reference | |
| Prediction | yes | no |
| yes | 123 | 137 |
| no | 188 | 2327 |

**Test result:**

|  |  |  |  |
| --- | --- | --- | --- |
| ROC | Sens | Spec | Accuracy |
| 0. 7689337 | 0.3955 | 0.9444 | 0.8829 |

1. **KNN:**

KNN was run with ROC metric, 10-fold cross validation and using tunegrid for tuning different parameters to find the optimal model. The final value used for the model was k = 25, with Training ROC Curve as 0.74 and Test ROC as 0.75. It provides with an Accuracy of 0.8948

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Train | k | ROC | Sens | Spec | Accuracy |
| 25 | 0.7413561 | 0.1950723 | 0.9836579 |  |
| Test |  | 0.7511 | 0.20900 | 0.98133 | 0.8948 |
|  |  |  |  |  |

**Test** Confusion Matrix:

|  |  |  |
| --- | --- | --- |
|  | Reference | |
| Prediction | yes | no |
| yes | 65 | 46 |
| no | 246 | 2418 |

**Discussion and Recommendations**

Performance measurement for all the models is presented in Area Under Curve and Accuracy matrix. AUC and ROC measure the ability to correctly classify clients who would subscribe to long-term deposit. Test measurement is the accuracy of the model for new data. From Table 5, we can observe that the best model considering overall test accuracy is KNN, followed by GBM and Random Forest. KNN can 98% correctly predict that the client will not subscribe to the long-term deposit and Banks can eliminate these candidates from targeting for the campaign.

Table 5: Test Accuracy for all Model

|  |  |  |  |
| --- | --- | --- | --- |
| **Models** | **Test Accuracy** | **Sensitivity** | **Specificity** |
| Logistic Regression | 0.8450 | 0.5209 | 0.8859 |
| LDA | 0.8583 | 0.5176 | 0.9013 |
| GBM | 0.8828 | 0.3955 | 0.9444 |
| KNN | 0.8947 | 0.2090 | 0.9813 |
| RF | 0.8763 | 0.4212 | 0.9338 |
| SVM | 0.8457 | 0.4791 | 0.8920 |
| Decision Trees | 0.8681 | 0.5113 | 0.9131 |

By ROC metric and Figure 11, we can observe that Random Forest is better at distinguishing between the two classes, followed by GBM and Decision Trees. For a successful bank marketing campaign, it is very important to classify clients by whether they will subscribe to long-term deposit or not. By selecting only most likely clients the campaigns can ensure efficiency and see improvements. Therefore, models like Random Forest and GBM are good classifying client response, and banks can allocate more marketing efforts to the clients who are classified as highly likely to accept term deposits, and call less to those who are unlikely to make term deposits.

A close up of a device

Description automatically generated

Figure 11: BoxPlot for ROC for all models

From Fig 12, We can determine that most important features for determining if client will subscriber to long-term deposit is nr.employed, euribor3m, cons.conf.idx, cons.price.idx, previous are top 5 most important attributes.

A screenshot of a social media post

Description automatically generated

Figure 12: Variable Importance

Therefore, to increase profitability and ensure success of their marketing campaign, they should focus on improving the quality and frequency of customer interaction and time their campaigns when interest rates are high and macroeconomic environment is stable.

**Recommendation:**

1. Banks need to reinvent their design and content of marketing campaigns, making it attractive to the target customers.
2. Some months have more success than other months so marketing campaigns must be targeted for those months.
3. More data is required for positive responses and by successful marketing strategies, this unbalance can be rectified.
4. A cluster analysis can be performed on the dataset to identify key features clients who subscribed and ones that did not subscribe to the long-term deposit.

**Appendix**

Bank\_Marketing

The data is related with direct marketing campaigns (phone calls) of a Portuguese banking institution. The classification goal is to predict if the client will subscribe a term deposit (variable y).

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(caret)

## Loading required package: lattice

## Loading required package: ggplot2

library(tidyverse)

## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.0 ──

## ✓ tibble 3.0.1 ✓ purrr 0.3.3  
## ✓ tidyr 1.0.2 ✓ stringr 1.4.0  
## ✓ readr 1.3.1 ✓ forcats 0.5.0

## Warning: package 'tibble' was built under R version 3.6.2

## ── Conflicts ────────────────────────────────────────── tidyverse\_conflicts() ──  
## x dplyr::filter() masks stats::filter()  
## x dplyr::lag() masks stats::lag()  
## x purrr::lift() masks caret::lift()

library(psych)

##   
## Attaching package: 'psych'

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

library(ggplot2)  
library(corrplot)

## corrplot 0.84 loaded

library(rpart)  
library(rpart.plot)  
library(caretEnsemble)

##   
## Attaching package: 'caretEnsemble'

## The following object is masked from 'package:ggplot2':  
##   
## autoplot

library(psych)  
library(pROC)

## Type 'citation("pROC")' for a citation.

##   
## Attaching package: 'pROC'

## The following objects are masked from 'package:stats':  
##   
## cov, smooth, var

library(glue)

##   
## Attaching package: 'glue'

## The following object is masked from 'package:dplyr':  
##   
## collapse

Reading the Data Data Set Information: The data is related with direct marketing campaigns of a Portuguese banking institution. The marketing campaigns were based on phone calls. Often, more than one contact to the same client was required, in order to access if the product (bank term deposit) would be (‘yes’) or not (‘no’) subscribed.

bank\_full\_data <- read.csv("bank-additional-full.csv",sep = ";",header=TRUE)  
dim(bank\_full\_data)

## [1] 41188 21

**DATA EXPLORATION**

head(bank\_full\_data)

## age job marital education default housing loan contact month  
## 1 56 housemaid married basic.4y no no no telephone may  
## 2 57 services married high.school unknown no no telephone may  
## 3 37 services married high.school no yes no telephone may  
## 4 40 admin. married basic.6y no no no telephone may  
## 5 56 services married high.school no no yes telephone may  
## 6 45 services married basic.9y unknown no no telephone may  
## day\_of\_week duration campaign pdays previous poutcome emp.var.rate  
## 1 mon 261 1 999 0 nonexistent 1.1  
## 2 mon 149 1 999 0 nonexistent 1.1  
## 3 mon 226 1 999 0 nonexistent 1.1  
## 4 mon 151 1 999 0 nonexistent 1.1  
## 5 mon 307 1 999 0 nonexistent 1.1  
## 6 mon 198 1 999 0 nonexistent 1.1  
## cons.price.idx cons.conf.idx euribor3m nr.employed y  
## 1 93.994 -36.4 4.857 5191 no  
## 2 93.994 -36.4 4.857 5191 no  
## 3 93.994 -36.4 4.857 5191 no  
## 4 93.994 -36.4 4.857 5191 no  
## 5 93.994 -36.4 4.857 5191 no  
## 6 93.994 -36.4 4.857 5191 no

**What are the Datatypes for the bank marketing data columns**

str(bank\_full\_data)

## 'data.frame': 41188 obs. of 21 variables:  
## $ age : int 56 57 37 40 56 45 59 41 24 25 ...  
## $ job : Factor w/ 12 levels "admin.","blue-collar",..: 4 8 8 1 8 8 1 2 10 8 ...  
## $ marital : Factor w/ 4 levels "divorced","married",..: 2 2 2 2 2 2 2 2 3 3 ...  
## $ education : Factor w/ 8 levels "basic.4y","basic.6y",..: 1 4 4 2 4 3 6 8 6 4 ...  
## $ default : Factor w/ 3 levels "no","unknown",..: 1 2 1 1 1 2 1 2 1 1 ...  
## $ housing : Factor w/ 3 levels "no","unknown",..: 1 1 3 1 1 1 1 1 3 3 ...  
## $ loan : Factor w/ 3 levels "no","unknown",..: 1 1 1 1 3 1 1 1 1 1 ...  
## $ contact : Factor w/ 2 levels "cellular","telephone": 2 2 2 2 2 2 2 2 2 2 ...  
## $ month : Factor w/ 10 levels "apr","aug","dec",..: 7 7 7 7 7 7 7 7 7 7 ...  
## $ day\_of\_week : Factor w/ 5 levels "fri","mon","thu",..: 2 2 2 2 2 2 2 2 2 2 ...  
## $ duration : int 261 149 226 151 307 198 139 217 380 50 ...  
## $ campaign : int 1 1 1 1 1 1 1 1 1 1 ...  
## $ pdays : int 999 999 999 999 999 999 999 999 999 999 ...  
## $ previous : int 0 0 0 0 0 0 0 0 0 0 ...  
## $ poutcome : Factor w/ 3 levels "failure","nonexistent",..: 2 2 2 2 2 2 2 2 2 2 ...  
## $ emp.var.rate : num 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 1.1 ...  
## $ cons.price.idx: num 94 94 94 94 94 ...  
## $ cons.conf.idx : num -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 -36.4 ...  
## $ euribor3m : num 4.86 4.86 4.86 4.86 4.86 ...  
## $ nr.employed : num 5191 5191 5191 5191 5191 ...  
## $ y : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...

**Taking subset of the data**

sample\_size = 10000  
set.seed(100)  
idxs = sample(1:nrow(bank\_full\_data),sample\_size,replace=F)  
subsample = bank\_full\_data[idxs,]  
pvalues = list()  
for (col in names(bank\_full\_data)) {  
 if (class(bank\_full\_data[,col]) %in% c("numeric","integer")) {  
 # Numeric variable. Using Kolmogorov-Smirnov test  
   
 pvalues[[col]] = ks.test(subsample[[col]],bank\_full\_data[[col]])$p.value  
   
 } else {  
 # Categorical variable. Using Pearson's Chi-square test  
   
 probs = table(bank\_full\_data[[col]])/nrow(bank\_full\_data)  
 pvalues[[col]] = chisq.test(table(subsample[[col]]),p=probs)$p.value  
   
 }  
}

## Warning in ks.test(subsample[[col]], bank\_full\_data[[col]]): p-value will be  
## approximate in the presence of ties

print(pvalues)

## $age  
## [1] 0.9517764  
##   
## $job  
## [1] 0.7156145  
##   
## $marital  
## [1] 0.1030109  
##   
## $education  
## [1] 0.8656887  
##   
## $default  
## [1] 0.633827  
##   
## $housing  
## [1] 0.9418751  
##   
## $loan  
## [1] 0.9517895  
##   
## $contact  
## [1] 0.3885186  
##   
## $month  
## [1] 0.5467619  
##   
## $day\_of\_week  
## [1] 0.3718226  
##   
## $duration  
## [1] 0.6559342  
##   
## $campaign  
## [1] 0.4659303  
##   
## $pdays  
## [1] 1  
##   
## $previous  
## [1] 1  
##   
## $poutcome  
## [1] 0.3392778  
##   
## $emp.var.rate  
## [1] 0.9999985  
##   
## $cons.price.idx  
## [1] 0.9932788  
##   
## $cons.conf.idx  
## [1] 0.9999982  
##   
## $euribor3m  
## [1] 0.9882724  
##   
## $nr.employed  
## [1] 0.9994323  
##   
## $y  
## [1] 0.8878625

**The records have p-value greater than 0.5. Seems like a good subset**

**Summary of the data**

bank\_mkt\_data <- subsample  
summary(bank\_mkt\_data)

## age job marital education   
## Min. :17.00 admin. :2583 divorced:1045 university.degree :2992   
## 1st Qu.:32.00 blue-collar:2241 married :6136 high.school :2269   
## Median :38.00 technician :1629 single :2800 basic.9y :1472   
## Mean :40.02 services : 930 unknown : 19 professional.course:1270   
## 3rd Qu.:47.00 management : 733 basic.4y :1020   
## Max. :98.00 retired : 390 basic.6y : 536   
## (Other) :1494 (Other) : 441   
## default housing loan contact month   
## no :7930 no :4504 no :8231 cellular :6389 may :3283   
## unknown:2070 unknown: 241 unknown: 241 telephone:3611 jul :1775   
## yes : 0 yes :5255 yes :1528 aug :1524   
## jun :1263   
## nov :1027   
## apr : 614   
## (Other): 514   
## day\_of\_week duration campaign pdays previous   
## fri:1880 Min. : 4.0 Min. : 1.000 Min. : 0.0 Min. :0.0000   
## mon:2100 1st Qu.: 101.0 1st Qu.: 1.000 1st Qu.:999.0 1st Qu.:0.0000   
## thu:2038 Median : 178.0 Median : 2.000 Median :999.0 Median :0.0000   
## tue:2025 Mean : 254.6 Mean : 2.581 Mean :961.1 Mean :0.1729   
## wed:1957 3rd Qu.: 315.2 3rd Qu.: 3.000 3rd Qu.:999.0 3rd Qu.:0.0000   
## Max. :4199.0 Max. :56.000 Max. :999.0 Max. :6.0000   
##   
## poutcome emp.var.rate cons.price.idx cons.conf.idx   
## failure :1007 Min. :-3.40000 Min. :92.20 Min. :-50.80   
## nonexistent:8637 1st Qu.:-1.80000 1st Qu.:93.08 1st Qu.:-42.70   
## success : 356 Median : 1.10000 Median :93.44 Median :-41.80   
## Mean : 0.08255 Mean :93.57 Mean :-40.49   
## 3rd Qu.: 1.40000 3rd Qu.:93.99 3rd Qu.:-36.40   
## Max. : 1.40000 Max. :94.77 Max. :-26.90   
##   
## euribor3m nr.employed y   
## Min. :0.634 Min. :4964 no :8869   
## 1st Qu.:1.344 1st Qu.:5099 yes:1131   
## Median :4.857 Median :5191   
## Mean :3.626 Mean :5167   
## 3rd Qu.:4.961 3rd Qu.:5228   
## Max. :5.045 Max. :5228   
##

**DATA CLEANING**

Default has 7930 and 0 yes ans 2070 values as Unknown.

bank\_mkt\_data$default <- NULL

Checking for Non zero variance attributes

nzv <- nearZeroVar(bank\_mkt\_data)  
print(names(bank\_mkt\_data[nzv]))

## [1] "pdays"

pdays has 96% values that are 999 which means client was not previously contacted

histogram(bank\_mkt\_data$pdays)

A screenshot of a cell phone

Description automatically generated

Removing attribute pdays

bank\_mkt\_data$pdays <- NULL  
#bank\_data <- bank\_mkt\_data %>% mutate(pdays=ifelse(pdays==999,0,pdays))

Check for NA columns and missing values for each column

sum(is.na(bank\_mkt\_data))

## [1] 0

Changing factor columns to Character as there are many columns with Unknown values

bank\_data <- bank\_mkt\_data %>% mutate\_if(is.factor, as.character)

There are no NA values in any columns There are unknown values in some columns. Changing unkown to NA and then omiting those rows

bank\_data[bank\_data == "unknown"] <- NA\_character\_  
bank\_data <-na.omit(bank\_data)

Target variable: y - has the client subscribed a term deposit? (binary: ‘yes’, ‘no’)

bank\_data$y <- as.factor(bank\_data$y)  
  
bank\_data$y <- relevel(bank\_data$y, ref="yes")

**Pre-Processing**

descrCor <- cor(bank\_data[,c(1,10:12,14:18)], use = "everything")  
print(descrCor)

## age duration campaign previous emp.var.rate  
## age 1.000000000 0.010970933 0.01622431 0.03335947 -0.001819825  
## duration 0.010970933 1.000000000 -0.07904046 0.01996800 -0.026477744  
## campaign 0.016224315 -0.079040458 1.00000000 -0.07608647 0.157790196  
## previous 0.033359469 0.019968002 -0.07608647 1.00000000 -0.418306575  
## emp.var.rate -0.001819825 -0.026477744 0.15779020 -0.41830658 1.000000000  
## cons.price.idx -0.008061296 -0.001119678 0.13021116 -0.19724033 0.771834957  
## cons.conf.idx 0.137967255 -0.007188049 -0.01428436 -0.06304805 0.199614663  
## euribor3m 0.014721533 -0.032068756 0.14214203 -0.45401724 0.971853156  
## nr.employed -0.011123812 -0.040050587 0.15239578 -0.50028124 0.907245401  
## cons.price.idx cons.conf.idx euribor3m nr.employed  
## age -0.008061296 0.137967255 0.01472153 -0.01112381  
## duration -0.001119678 -0.007188049 -0.03206876 -0.04005059  
## campaign 0.130211156 -0.014284357 0.14214203 0.15239578  
## previous -0.197240333 -0.063048052 -0.45401724 -0.50028124  
## emp.var.rate 0.771834957 0.199614663 0.97185316 0.90724540  
## cons.price.idx 1.000000000 0.056194236 0.68491662 0.51783498  
## cons.conf.idx 0.056194236 1.000000000 0.28379417 0.10932203  
## euribor3m 0.684916616 0.283794168 1.00000000 0.94513910  
## nr.employed 0.517834983 0.109322032 0.94513910 1.00000000

highlyCorDescr <- findCorrelation(descrCor, cutoff = .75)

print(highlyCorDescr)

## [1] 8 5

Social and economic context attributes There is a high correlation between emp.var.rate,euribor3m ,nr.employed But they significant to the response variable

Using VIC to determine multicollinearity

model <- lm(as.integer(y)~.,bank\_data)  
car::vif(model)

## GVIF Df GVIF^(1/(2\*Df))  
## age 1.622780 1 1.273884  
## job 4.679566 10 1.080215  
## marital 1.310093 2 1.069856  
## education 3.376491 6 1.106723  
## housing 1.022406 1 1.011141  
## loan 1.006884 1 1.003436  
## contact 3.468039 1 1.862267  
## month 276.054835 9 1.366504  
## day\_of\_week 1.050443 4 1.006170  
## duration 1.021279 1 1.010584  
## campaign 1.055693 1 1.027469  
## previous 5.661735 1 2.379440  
## poutcome 6.103740 2 1.571806  
## emp.var.rate 244.424076 1 15.634068  
## cons.price.idx 91.302466 1 9.555232  
## cons.conf.idx 6.874500 1 2.621927  
## euribor3m 200.169997 1 14.148145  
## nr.employed 207.103962 1 14.391107

Using VIc we can see that there is a very high multicollinearity Removing nr.employed

model <- lm(as.integer(y)~.-nr.employed,bank\_data)  
car::vif(model)

## GVIF Df GVIF^(1/(2\*Df))  
## age 1.622693 1 1.273850  
## job 4.669353 10 1.080097  
## marital 1.310037 2 1.069845  
## education 3.372168 6 1.106604  
## housing 1.021862 1 1.010872  
## loan 1.006850 1 1.003419  
## contact 3.317498 1 1.821400  
## month 43.661846 9 1.233436  
## day\_of\_week 1.043741 4 1.005366  
## duration 1.021030 1 1.010460  
## campaign 1.054893 1 1.027080  
## previous 5.660779 1 2.379239  
## poutcome 6.099338 2 1.571523  
## emp.var.rate 150.397233 1 12.263655  
## cons.price.idx 16.662045 1 4.081917  
## cons.conf.idx 4.021254 1 2.005307  
## euribor3m 103.058231 1 10.151760

Removing nr.employed drastically reduced the VIF score Taking out emp.var.rate

model <- lm(as.integer(y)~.-nr.employed-emp.var.rate-month,bank\_data)  
car::vif(model)

## GVIF Df GVIF^(1/(2\*Df))  
## age 1.610006 1 1.268860  
## job 4.445047 10 1.077442  
## marital 1.303809 2 1.068571  
## education 3.302217 6 1.104673  
## housing 1.019589 1 1.009747  
## loan 1.005844 1 1.002918  
## contact 1.809035 1 1.345004  
## day\_of\_week 1.028232 4 1.003486  
## duration 1.014682 1 1.007314  
## campaign 1.036671 1 1.018170  
## previous 5.425493 1 2.329269  
## poutcome 5.884190 2 1.557477  
## cons.price.idx 3.212811 1 1.792432  
## cons.conf.idx 1.437386 1 1.198910  
## euribor3m 3.015705 1 1.736579

Visualizations Lets see the distribution of Age

bank\_data %>%   
 ggplot(., aes(x=age)) +  
 geom\_histogram(fill="steelblue")+ggtitle('Age Distributition')

## `stat\_bin()` using `bins = 30`. Pick better value with `binwidth`.

A screenshot of a social media post

Description automatically generated

Which Age group subscribed a term deposit

bank\_data %>%   
 ggplot(., aes(x=age,fill=as.character(y))) +  
 geom\_bar(position='dodge')

A picture containing pencil

Description automatically generated

What education level people subscribed a term deposit

bank\_data %>%   
 ggplot(., aes(y=education,fill=as.character(y))) +  
 geom\_bar(position="dodge")+  
 theme(axis.text.x = element\_text(angle = 90, hjust = 1))

A screenshot of a cell phone

Description automatically generated based on job how many subscribed a term deposit

bank\_data %>%   
 ggplot(., aes(x=job,fill=as.character(y))) +  
 geom\_bar(position="dodge")+  
 theme(axis.text.x = element\_text(angle = 90, hjust = 1))

A picture containing game

Description automatically generated

bank\_data %>%   
 ggplot(., aes(x=loan,fill=as.character(y))) +  
 geom\_bar(position="dodge")+  
 theme(axis.text.x = element\_text(angle = 90, hjust = 1))

A screenshot of a cell phone

Description automatically generated

bank\_data %>%   
 ggplot(., aes(x=marital,fill=as.character(y))) +  
 geom\_bar(position="dodge")+  
 theme(axis.text.x = element\_text(angle = 90, hjust = 1))

A screenshot of a cell phone

Description automatically generated

bank\_data %>%   
 ggplot(., aes(x=housing,fill=as.character(y))) +  
 geom\_bar(position="dodge")+  
 theme(axis.text.x = element\_text(angle = 90, hjust = 1))

A screenshot of a cell phone

Description automatically generated

bank\_data %>%   
 ggplot(., aes(x=duration,fill=as.character(y))) +  
 geom\_bar(position="dodge")+  
 theme(axis.text.x = element\_text(angle = 90, hjust = 1))

A close up of text on a white background

Description automatically generated

Distribution of target variable

table(bank\_data$y)

##   
## yes no   
## 1039 8216

bank\_data %>%   
 ggplot(., aes(x=factor(y))) +  
 geom\_bar(fill="steelblue")+  
 xlab("y")

A screenshot of a cell phone

Description automatically generated

ggtitle('Target Data Distribution')

## $title  
## [1] "Target Data Distribution"  
##   
## attr(,"class")  
## [1] "labels"

Data is highly imbalanced There are rows for 6452 No and only 939 rows for yes

bank\_data.pca <- prcomp(bank\_data[,c(1,10:12,14:18)], center = TRUE,scale = TRUE)  
summary(bank\_data.pca)

## Importance of components:  
## PC1 PC2 PC3 PC4 PC5 PC6 PC7  
## Standard deviation 1.9339 1.0685 1.0272 0.9689 0.91691 0.90842 0.65018  
## Proportion of Variance 0.4155 0.1269 0.1172 0.1043 0.09341 0.09169 0.04697  
## Cumulative Proportion 0.4155 0.5424 0.6596 0.7639 0.85735 0.94904 0.99601  
## PC8 PC9  
## Standard deviation 0.15845 0.1038  
## Proportion of Variance 0.00279 0.0012  
## Cumulative Proportion 0.99880 1.0000

PC1 can explain 38% of variation, PC2 can explain 13%

biplot(bank\_data.pca, scale=0)

A picture containing clock

Description automatically generated

cum\_p <- cumsum(bank\_data.pca$sdev^2 / sum(bank\_data.pca$sdev^2))  
plot(cum\_p, xlab="PCA", ylab="Cumulative Prop of Variance Explained", ylim=c(0,1), type='b')

A screenshot of a cell phone

Description automatically generated

Duration: last contact duration, in seconds (numeric). Important note: this attribute highly affects the output target (e.g., if duration=0 then y=‘no’). Yet, the duration is not known before a call is performed. Also, after the end of the call y is obviously known. Thus, this input should only be included for benchmark purposes and should be discarded if the intention is to have a realistic predictive model.

set.seed(1000)  
bank\_data$duration <- NULL  
#bank\_data$y <- as.factor(bank\_data$y)  
bank\_data <- bank\_data %>% mutate\_if(is.character, as.factor)  
ctrl <- trainControl(method = "cv", number=10, summaryFunction=twoClassSummary,  
 savePredictions=T,classProbs=T)  
#ctrl <- trainControl(method="cv", number=10)  
trainIndex <- createDataPartition(bank\_data$y, p=.7, list=F)  
train <- bank\_data[trainIndex,]  
valid <- bank\_data[-trainIndex,]

table(train$y)

##   
## yes no   
## 728 5752

library(DMwR)

## Loading required package: grid

## Registered S3 method overwritten by 'quantmod':  
## method from  
## as.zoo.data.frame zoo

#hybrid both up and down  
set.seed(9560)  
smote\_train <- SMOTE(y ~ ., perc.over = 300, k = 5, perc.under = 200,  
 data = train)   
table(smote\_train$y)

##   
## yes no   
## 2912 4368

smote\_train <- na.omit(smote\_train)

Fitting Models using all predictors 1. Logistic Regression with 10 fold Cross Validation

set.seed(7000)  
  
smote\_train$y <- as.factor(smote\_train$y)  
#Logistic regression  
glm.train.fit <- train(y ~ ., data=smote\_train,   
 method = "glm",  
 metric="ROC",  
 family=binomial,  
 trControl=ctrl)  
summary(glm.train.fit)

##   
## Call:  
## NULL  
##   
## Deviance Residuals:   
## Min 1Q Median 3Q Max   
## -2.4006 -0.6830 0.5144 0.7181 3.2028   
##   
## Coefficients:  
## Estimate Std. Error z value Pr(>|z|)   
## (Intercept) 27.987584 39.265698 0.713 0.475985   
## age 0.007555 0.003555 2.125 0.033569 \*   
## `jobblue-collar` -0.203083 0.099617 -2.039 0.041486 \*   
## jobentrepreneur 0.188604 0.189344 0.996 0.319205   
## jobhousemaid -0.015614 0.219383 -0.071 0.943260   
## jobmanagement -0.036261 0.127575 -0.284 0.776235   
## jobretired -0.626208 0.162314 -3.858 0.000114 \*\*\*  
## `jobself-employed` -0.294801 0.158100 -1.865 0.062230 .   
## jobservices -0.181472 0.119481 -1.519 0.128804   
## jobstudent -0.452574 0.188498 -2.401 0.016353 \*   
## jobtechnician -0.268183 0.100889 -2.658 0.007856 \*\*   
## jobunemployed -0.872136 0.173778 -5.019 5.20e-07 \*\*\*  
## maritalmarried 0.459561 0.095462 4.814 1.48e-06 \*\*\*  
## maritalsingle 0.028747 0.104322 0.276 0.782886   
## educationbasic.6y 0.197952 0.163080 1.214 0.224812   
## educationbasic.9y 0.262796 0.125572 2.093 0.036367 \*   
## educationhigh.school -0.012655 0.120158 -0.105 0.916122   
## educationilliterate -13.714318 170.179724 -0.081 0.935770   
## educationprofessional.course 0.280968 0.138024 2.036 0.041787 \*   
## educationuniversity.degree 0.026003 0.119996 0.217 0.828443   
## housingyes 0.138300 0.059900 2.309 0.020952 \*   
## loanyes -1.099115 0.069199 -15.883 < 2e-16 \*\*\*  
## contacttelephone -0.029779 0.088884 -0.335 0.737599   
## monthaug 0.081926 0.178258 0.460 0.645809   
## monthdec 0.336745 0.348393 0.967 0.333762   
## monthjul -0.358946 0.138982 -2.583 0.009804 \*\*   
## monthjun -0.001407 0.161763 -0.009 0.993059   
## monthmar -1.039994 0.245604 -4.234 2.29e-05 \*\*\*  
## monthmay 0.648925 0.119147 5.446 5.14e-08 \*\*\*  
## monthnov 0.540618 0.164997 3.277 0.001051 \*\*   
## monthoct 0.166985 0.221868 0.753 0.451672   
## monthsep 0.135589 0.260288 0.521 0.602424   
## day\_of\_weekmon 0.057010 0.095412 0.598 0.550164   
## day\_of\_weekthu -0.100913 0.095334 -1.059 0.289819   
## day\_of\_weektue 0.076138 0.098129 0.776 0.437808   
## day\_of\_weekwed -0.193965 0.094872 -2.045 0.040904 \*   
## campaign 0.076928 0.015329 5.018 5.21e-07 \*\*\*  
## previous -0.322528 0.097869 -3.295 0.000982 \*\*\*  
## poutcomenonexistent -0.451765 0.123460 -3.659 0.000253 \*\*\*  
## poutcomesuccess -2.273153 0.173991 -13.065 < 2e-16 \*\*\*  
## emp.var.rate 0.606307 0.169943 3.568 0.000360 \*\*\*  
## cons.price.idx -0.730842 0.256681 -2.847 0.004410 \*\*   
## cons.conf.idx -0.037801 0.012884 -2.934 0.003346 \*\*   
## euribor3m -0.286786 0.185719 -1.544 0.122541   
## nr.employed 0.007851 0.003460 2.269 0.023256 \*   
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## (Dispersion parameter for binomial family taken to be 1)  
##   
## Null deviance: 9799.0 on 7279 degrees of freedom  
## Residual deviance: 7010.9 on 7235 degrees of freedom  
## AIC: 7100.9  
##   
## Number of Fisher Scoring iterations: 13

print(glm.train.fit)

## Generalized Linear Model   
##   
## 7280 samples  
## 17 predictor  
## 2 classes: 'yes', 'no'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 6552, 6551, 6552, 6553, 6551, 6552, ...   
## Resampling results:  
##   
## ROC Sens Spec   
## 0.8224775 0.6208786 0.8962862

#valid$y <- as.factor(valid$y)  
#newdata <- valid[,!colnames(valid) %in% c("y")]  
#pred <- predict(glm.train.fit,newdata)  
#confusionMatrix(valid$y,pred)  
test.pred.prob <- predict(glm.train.fit, valid, type="prob")  
  
test.pred.class <- predict(glm.train.fit, valid)   
confusionMatrix(test.pred.class, valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 162 281  
## no 149 2183  
##   
## Accuracy : 0.845   
## 95% CI : (0.831, 0.8583)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3432   
##   
## Mcnemar's Test P-Value : 2.66e-10   
##   
## Sensitivity : 0.52090   
## Specificity : 0.88596   
## Pos Pred Value : 0.36569   
## Neg Pred Value : 0.93611   
## Prevalence : 0.11207   
## Detection Rate : 0.05838   
## Detection Prevalence : 0.15964   
## Balanced Accuracy : 0.70343   
##   
## 'Positive' Class : yes   
##

log\_accuracy <- mean(valid$y == test.pred.class )

d.log.roc<- roc(response= glm.train.fit$pred$obs, predictor=glm.train.fit$pred$yes)

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

test.log.roc<- roc(response= valid$y, predictor=test.pred.prob[[1]]) #assumes postive class Yes is reference level

## Setting levels: control = yes, case = no  
## Setting direction: controls > cases

plot(test.log.roc, legacy.axes=T)  
plot(d.log.roc, add=T, col="blue")  
legend(x=.2, y=.7, legend=c("Test Logit", "Train Logit"), col=c("black", "blue"),lty=1)

A close up of a map

Description automatically generated

print(glue("Test Area under the curve :{auc(test.log.roc)}"))

## Test Area under the curve :0.749270524491586

print(glue("Train Area under the curve :{auc(d.log.roc)}"))

## Train Area under the curve :0.822208508495854

1. Logistic Regression using 10 fold Cross Validation and PCA

set.seed(7000)  
  
#Logistic regression  
glm.pca.fit <- train(y ~ ., data=smote\_train,   
 method = "glm",  
 preProcess="pca",  
 metric="ROC",  
 family=binomial,  
 trControl=ctrl)  
print(glm.pca.fit)

## Generalized Linear Model   
##   
## 7280 samples  
## 17 predictor  
## 2 classes: 'yes', 'no'   
##   
## Pre-processing: principal component signal extraction (44), centered  
## (44), scaled (44)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 6552, 6551, 6552, 6553, 6551, 6552, ...   
## Resampling results:  
##   
## ROC Sens Spec   
## 0.8149573 0.6119392 0.8880451

Testing the Logistic Regression for validation set

#valid$y <- as.factor(valid$y)  
#newdata <- valid[,!colnames(valid) %in% c("y")]  
#confusionMatrix(valid$y,predict(glm.pca.fit,newdata))  
test.pca.prob <- predict(glm.pca.fit, valid, type="prob")  
test.pca.class <- predict(glm.pca.fit, valid)   
confusionMatrix(test.pca.class, valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 154 279  
## no 157 2185  
##   
## Accuracy : 0.8429   
## 95% CI : (0.8288, 0.8562)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3261   
##   
## Mcnemar's Test P-Value : 6.838e-09   
##   
## Sensitivity : 0.4952   
## Specificity : 0.8868   
## Pos Pred Value : 0.3557   
## Neg Pred Value : 0.9330   
## Prevalence : 0.1121   
## Detection Rate : 0.0555   
## Detection Prevalence : 0.1560   
## Balanced Accuracy : 0.6910   
##   
## 'Positive' Class : yes   
##

log\_pca\_accuracy <- mean(valid$y == test.pca.class )

test.pca.roc<- roc(response= valid$y, predictor=test.pca.prob[[1]]) #assumes postive class Yes is reference level

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

plot(test.pca.roc, legacy.axes=T,col="red")  
plot(test.log.roc, add=T, col="blue")  
legend(x=.2, y=.7, legend=c("Test PCA", "Test No PCA"), col=c("red", "blue"),lty=1)

A close up of a map

Description automatically generated

print(glue("PCA Area under the curve :{auc(test.pca.roc)}"))

## PCA Area under the curve :0.746267799724391

print(glue("Logistic Regression Area under the curve :{auc(test.log.roc)}"))

## Logistic Regression Area under the curve :0.749270524491586

-euribor3m-nr.employed

set.seed(7000)  
#Logistic regression  
glm.train.fit\_1 <- train(y ~ .-emp.var.rate-nr.employed ,data=smote\_train,   
 method = "glm",  
 family=binomial,  
 metric="ROC",  
 trControl=ctrl)  
print(glm.train.fit\_1)

## Generalized Linear Model   
##   
## 7280 samples  
## 17 predictor  
## 2 classes: 'yes', 'no'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 6552, 6551, 6552, 6553, 6551, 6552, ...   
## Resampling results:  
##   
## ROC Sens Spec   
## 0.821238 0.6136621 0.8997224

Testing the selected feature Logistic Regression

test.log.prob.1 <- predict(glm.train.fit\_1, valid, type="prob")  
test.log.class.1 <- predict(glm.train.fit\_1, valid)   
confusionMatrix(test.log.class.1 , valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 157 283  
## no 154 2181  
##   
## Accuracy : 0.8425   
## 95% CI : (0.8284, 0.8559)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3301   
##   
## Mcnemar's Test P-Value : 9.179e-10   
##   
## Sensitivity : 0.50482   
## Specificity : 0.88515   
## Pos Pred Value : 0.35682   
## Neg Pred Value : 0.93405   
## Prevalence : 0.11207   
## Detection Rate : 0.05658   
## Detection Prevalence : 0.15856   
## Balanced Accuracy : 0.69498   
##   
## 'Positive' Class : yes   
##

test.log.roc.1<- roc(response= valid$y, predictor=test.log.prob.1[[1]]) #assumes postive class Yes is reference level

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

plot(test.log.roc.1, legacy.axes=T,col="red")  
plot(test.log.roc, add=T, col="blue")  
legend(x=.2, y=.7, legend=c("Test Selected Logistic", "Test Logistic"), col=c("red", "blue"),lty=1)

A close up of a map

Description automatically generated

plot(test.log.roc.1,legacy.axes=T,col="red")  
plot(test.log.roc,add=T, col="blue")  
plot(test.pca.roc, add=T,col="green")  
legend(x=.2, y=.7, legend=c("Test Selected Logit", "Test Logit","Test LR PCA"), col=c("red", "blue","green"),lty=1)

A close up of a map

Description automatically generated

print(glue("Selective LoR Area under the curve :{auc(test.log.roc.1)}"))

## Selective LoR Area under the curve :0.746159487618491

print(glue("Logistic Regression Area under the curve :{auc(test.log.roc)}"))

## Logistic Regression Area under the curve :0.749270524491586

There is no change in accuracy by removing correlated features

ridge <- train(  
 y ~., data = smote\_train, method = "glmnet",  
 trControl = trainControl("cv", number = 10),  
 tuneGrid = expand.grid(alpha = 0, lambda = 0)  
 )  
predictions <- ridge %>% predict(valid)  
confusionMatrix(predictions,valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 157 258  
## no 154 2206  
##   
## Accuracy : 0.8515   
## 95% CI : (0.8378, 0.8646)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3491   
##   
## Mcnemar's Test P-Value : 3.886e-07   
##   
## Sensitivity : 0.50482   
## Specificity : 0.89529   
## Pos Pred Value : 0.37831   
## Neg Pred Value : 0.93475   
## Prevalence : 0.11207   
## Detection Rate : 0.05658   
## Detection Prevalence : 0.14955   
## Balanced Accuracy : 0.70006   
##   
## 'Positive' Class : yes   
##

lasso <- train(  
 y ~., data = smote\_train, method = "glmnet",  
 trControl = trainControl("cv", number = 10),  
 tuneGrid = expand.grid(alpha = 1, lambda = 0)  
 )  
predictions1 <- lasso %>% predict(valid)  
confusionMatrix(predictions1,valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 162 279  
## no 149 2185  
##   
## Accuracy : 0.8458   
## 95% CI : (0.8318, 0.859)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3447   
##   
## Mcnemar's Test P-Value : 4.505e-10   
##   
## Sensitivity : 0.52090   
## Specificity : 0.88677   
## Pos Pred Value : 0.36735   
## Neg Pred Value : 0.93616   
## Prevalence : 0.11207   
## Detection Rate : 0.05838   
## Detection Prevalence : 0.15892   
## Balanced Accuracy : 0.70383   
##   
## 'Positive' Class : yes   
##

set.seed(7000)  
#Logistic regression  
glm.train.fit\_1 <- train(y ~ .-euribor3m,data=smote\_train,   
 method = "glm",  
 metric="ROC",  
 family=binomial,  
 trControl=ctrl)  
print(glm.train.fit\_1)

## Generalized Linear Model   
##   
## 7280 samples  
## 17 predictor  
## 2 classes: 'yes', 'no'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 6552, 6551, 6552, 6553, 6551, 6552, ...   
## Resampling results:  
##   
## ROC Sens Spec   
## 0.8223675 0.6201925 0.897433

Testing the selected feature Logistic Regression

test.log.prob.1 <- predict(glm.train.fit\_1, valid, type="prob")  
test.log.class.1 <- predict(glm.train.fit\_1, valid)   
confusionMatrix(test.log.class.1 , valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 162 280  
## no 149 2184  
##   
## Accuracy : 0.8454   
## 95% CI : (0.8314, 0.8587)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.344   
##   
## Mcnemar's Test P-Value : 3.464e-10   
##   
## Sensitivity : 0.52090   
## Specificity : 0.88636   
## Pos Pred Value : 0.36652   
## Neg Pred Value : 0.93613   
## Prevalence : 0.11207   
## Detection Rate : 0.05838   
## Detection Prevalence : 0.15928   
## Balanced Accuracy : 0.70363   
##   
## 'Positive' Class : yes   
##

1. LDA

set.seed(7000)  
lda.train.fit <- train(y ~ ., data=smote\_train,   
 method = "lda",  
 metric="ROC",  
 family=binomial,  
 trControl=ctrl)  
print(lda.train.fit)

## Linear Discriminant Analysis   
##   
## 7280 samples  
## 17 predictor  
## 2 classes: 'yes', 'no'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 6552, 6551, 6552, 6553, 6551, 6552, ...   
## Resampling results:  
##   
## ROC Sens Spec   
## 0.8214482 0.6067916 0.9029297

d.lda.roc<- roc(response= lda.train.fit$pred$obs, predictor=lda.train.fit$pred$yes)

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

test.lda.prob <- predict(lda.train.fit, valid, type="prob")  
test.lda.class <- predict(lda.train.fit, valid)   
confusionMatrix(test.lda.class , valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 161 243  
## no 150 2221  
##   
## Accuracy : 0.8584   
## 95% CI : (0.8448, 0.8711)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3706   
##   
## Mcnemar's Test P-Value : 3.471e-06   
##   
## Sensitivity : 0.51768   
## Specificity : 0.90138   
## Pos Pred Value : 0.39851   
## Neg Pred Value : 0.93674   
## Prevalence : 0.11207   
## Detection Rate : 0.05802   
## Detection Prevalence : 0.14559   
## Balanced Accuracy : 0.70953   
##   
## 'Positive' Class : yes   
##

lda\_accuracy <- mean(valid$y == test.lda.class )  
test.lda.roc<- roc(response= valid$y, predictor=test.lda.prob[[1]])

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

Caret Rpart already prunes the Tree

set.seed(9000)  
  
rpart.train <- train(y ~. ,data=smote\_train,   
 method="rpart",  
 tuneGrid=data.frame(cp = seq(0.01, 0.1, len = 25)),  
 metric="ROC",  
 trControl=ctrl)  
print(rpart.train)

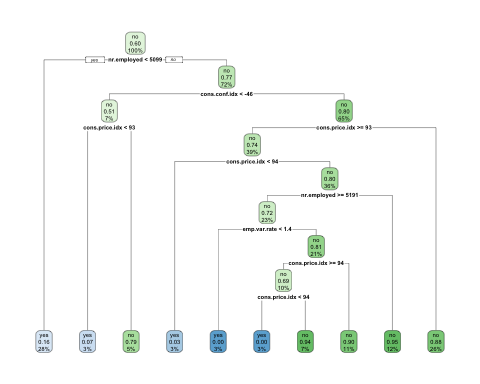
## CART   
##   
## 7280 samples  
## 17 predictor  
## 2 classes: 'yes', 'no'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 6551, 6552, 6553, 6552, 6553, 6552, ...   
## Resampling results across tuning parameters:  
##   
## cp ROC Sens Spec   
## 0.01000 0.8993383 0.8303629 0.9171247  
## 0.01375 0.8957152 0.8200784 0.9159779  
## 0.01750 0.8916824 0.8145989 0.9159779  
## 0.02125 0.8586246 0.7623888 0.9203315  
## 0.02500 0.8586246 0.7623888 0.9203315  
## 0.02875 0.8491163 0.7473203 0.9203315  
## 0.03250 0.8112221 0.6724592 0.9207892  
## 0.03625 0.7509753 0.5772702 0.9246804  
## 0.04000 0.7509753 0.5772702 0.9246804  
## 0.04375 0.7509753 0.5772702 0.9246804  
## 0.04750 0.7509753 0.5772702 0.9246804  
## 0.05125 0.7509753 0.5772702 0.9246804  
## 0.05500 0.7509753 0.5772702 0.9246804  
## 0.05875 0.7509753 0.5772702 0.9246804  
## 0.06250 0.7509753 0.5772702 0.9246804  
## 0.06625 0.7509753 0.5772702 0.9246804  
## 0.07000 0.7509753 0.5772702 0.9246804  
## 0.07375 0.7509753 0.5772702 0.9246804  
## 0.07750 0.7509753 0.5772702 0.9246804  
## 0.08125 0.7509753 0.5772702 0.9246804  
## 0.08500 0.7509753 0.5772702 0.9246804  
## 0.08875 0.7509753 0.5772702 0.9246804  
## 0.09250 0.7509753 0.5772702 0.9246804  
## 0.09625 0.7509753 0.5772702 0.9246804  
## 0.10000 0.7509753 0.5772702 0.9246804  
##   
## ROC was used to select the optimal model using the largest value.  
## The final value used for the model was cp = 0.01.

d.rpart.roc<- roc(response= rpart.train$pred$obs, predictor=rpart.train$pred$yes)

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

library(rpart.plot)  
rpart.plot(rpart.train$finalModel)



test.rpart.prob <- predict(rpart.train, valid, type="prob")  
test.rpart.class <- predict(rpart.train, valid)   
confusionMatrix(test.rpart.class , valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 159 214  
## no 152 2250  
##   
## Accuracy : 0.8681   
## 95% CI : (0.8549, 0.8805)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 0.99946   
##   
## Kappa : 0.3904   
##   
## Mcnemar's Test P-Value : 0.00143   
##   
## Sensitivity : 0.5113   
## Specificity : 0.9131   
## Pos Pred Value : 0.4263   
## Neg Pred Value : 0.9367   
## Prevalence : 0.1121   
## Detection Rate : 0.0573   
## Detection Prevalence : 0.1344   
## Balanced Accuracy : 0.7122   
##   
## 'Positive' Class : yes   
##

test.rpart.roc<- roc(response= valid$y, predictor=test.rpart.prob[[1]])

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

rpart\_accuracy <- mean(valid$y == test.rpart.class )

Random Forest

set.seed(1000)  
mtryGrid <- expand.grid(mtry = 2:20)  
rf\_random <- train(y~., data=smote\_train, method="rf",metric="ROC", tuneGrid=mtryGrid, trControl=ctrl)  
d.rf.roc<- roc(response= rf\_random$pred$obs, predictor=rf\_random$pred$yes)

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

print(rf\_random)

## Random Forest   
##   
## 7280 samples  
## 17 predictor  
## 2 classes: 'yes', 'no'   
##   
## No pre-processing  
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 6552, 6552, 6552, 6552, 6553, 6551, ...   
## Resampling results across tuning parameters:  
##   
## mtry ROC Sens Spec   
## 2 0.9296236 0.6700148 0.9510030  
## 3 0.9475372 0.8104611 0.9647424  
## 4 0.9537340 0.8423928 0.9677188  
## 5 0.9563766 0.8447983 0.9711534  
## 6 0.9576187 0.8461705 0.9693217  
## 7 0.9580622 0.8468531 0.9688640  
## 8 0.9579838 0.8482300 0.9688619  
## 9 0.9582966 0.8478887 0.9693212  
## 10 0.9583453 0.8461705 0.9700077  
## 11 0.9582386 0.8475439 0.9695489  
## 12 0.9583730 0.8465118 0.9693206  
## 13 0.9579753 0.8458269 0.9686352  
## 14 0.9576784 0.8451384 0.9686347  
## 15 0.9576922 0.8458257 0.9686341  
## 16 0.9578567 0.8468531 0.9677183  
## 17 0.9575538 0.8465118 0.9677183  
## 18 0.9572160 0.8427364 0.9693212  
## 19 0.9575282 0.8427341 0.9670307  
## 20 0.9574703 0.8437638 0.9679466  
##   
## ROC was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 12.

plot(rf\_random)

A close up of a map

Description automatically generated

Variable Importance

varImp(rf\_random)

## rf variable importance  
##   
## only 20 most important variables shown (out of 44)  
##   
## Overall  
## nr.employed 100.000  
## euribor3m 83.818  
## cons.conf.idx 61.753  
## cons.price.idx 57.856  
## previous 55.064  
## emp.var.rate 38.714  
## age 34.225  
## campaign 26.218  
## poutcomesuccess 12.887  
## loanyes 9.886  
## housingyes 6.344  
## poutcomenonexistent 6.057  
## contacttelephone 5.538  
## educationuniversity.degree 4.850  
## day\_of\_weekthu 4.768  
## maritalmarried 4.753  
## educationhigh.school 4.464  
## day\_of\_weekmon 4.429  
## monthmay 4.411  
## maritalsingle 4.275

plot(varImp(rf\_random))

A close up of a logo

Description automatically generated euribor3m, nr.employed, emp.var.rate, age are the most important attributes

test.rf.prob <- predict(rf\_random, valid, type="prob")  
test.rf.class <- predict(rf\_random, valid)   
confusionMatrix(test.rf.class , valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 131 163  
## no 180 2301  
##   
## Accuracy : 0.8764   
## 95% CI : (0.8636, 0.8884)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 0.9735   
##   
## Kappa : 0.3638   
##   
## Mcnemar's Test P-Value : 0.3876   
##   
## Sensitivity : 0.42122   
## Specificity : 0.93385   
## Pos Pred Value : 0.44558   
## Neg Pred Value : 0.92745   
## Prevalence : 0.11207   
## Detection Rate : 0.04721   
## Detection Prevalence : 0.10595   
## Balanced Accuracy : 0.67753   
##   
## 'Positive' Class : yes   
##

test.rf.roc<- roc(response= valid$y, predictor=test.rf.prob[[1]])

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

rf\_accuracy <- mean(valid$y == test.rf.class )

set.seed(1000)  
mtryGrid <- expand.grid(mtry = 2:20)  
rf\_random\_pca <- train(y~., data=smote\_train, method="rf", metric="ROC",preProcess="pca", tuneGrid=mtryGrid, trControl=ctrl)  
print(rf\_random\_pca)

## Random Forest   
##   
## 7280 samples  
## 17 predictor  
## 2 classes: 'yes', 'no'   
##   
## Pre-processing: principal component signal extraction (44), centered  
## (44), scaled (44)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 6552, 6552, 6552, 6552, 6553, 6551, ...   
## Resampling results across tuning parameters:  
##   
## mtry ROC Sens Spec   
## 2 0.9191201 0.7270207 0.9233032  
## 3 0.9191159 0.7294238 0.9242190  
## 4 0.9192873 0.7318258 0.9246746  
## 5 0.9191894 0.7338888 0.9226188  
## 6 0.9192076 0.7356023 0.9262791  
## 7 0.9192590 0.7325119 0.9249082  
## 8 0.9183651 0.7321682 0.9237619  
## 9 0.9181401 0.7349174 0.9244494  
## 10 0.9179313 0.7287389 0.9246798  
## 11 0.9175349 0.7359471 0.9230749  
## 12 0.9170113 0.7321706 0.9235352  
## 13 0.9167694 0.7338829 0.9228476  
## 14 0.9159317 0.7307949 0.9242211  
## 15 0.9164075 0.7287365 0.9239923  
## 16 0.9159459 0.7304477 0.9235352  
## 17 0.9155272 0.7338829 0.9210164  
## 18 0.9148772 0.7283905 0.9212432  
## 19 0.9155752 0.7294238 0.9228487  
## 20 0.9155075 0.7328579 0.9223899  
##   
## ROC was used to select the optimal model using the largest value.  
## The final value used for the model was mtry = 4.

plot(rf\_random\_pca)

A close up of a map

Description automatically generated

test.rf.pca.prob <- predict(rf\_random\_pca, valid, type="prob")  
test.rf.pca.class <- predict(rf\_random\_pca, valid)   
confusionMatrix(test.rf.pca.class , valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 174 380  
## no 137 2084  
##   
## Accuracy : 0.8137   
## 95% CI : (0.7987, 0.828)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3021   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.5595   
## Specificity : 0.8458   
## Pos Pred Value : 0.3141   
## Neg Pred Value : 0.9383   
## Prevalence : 0.1121   
## Detection Rate : 0.0627   
## Detection Prevalence : 0.1996   
## Balanced Accuracy : 0.7026   
##   
## 'Positive' Class : yes   
##

rf\_pca\_accuracy <- mean(valid$y == test.rf.pca.class )  
test.rf.pca.roc<- roc(response= valid$y, predictor=test.rf.pca.prob[[1]])

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

SVM se train()’s tuneGrid parameter to do some sensitivity analysis around the values C = 1 and sigma = 0.015 that produced the model with the best ROC value. Note that R’s expand.grid() function is used to build a dataframe contain all the combinations of C and sigma we want to look at.

#ctrl <- trainControl(method = "cv", savePred=T)  
svmGrid <- expand.grid(sigma = c(.01, .015, 0.2),  
 C = c(1:10)  
)  
#svmGrid <- expand.grid(sigma= c(-25, -20, -15,-10, -5, 0), C= c(0:5))  
svm.train <- train(y ~., data=smote\_train,   
 method="svmRadial",  
 metric="ROC",  
 preProcess = c("center","scale"),  
 trControl=ctrl,  
 tuneGrid=svmGrid)

## line search fails -1.952727 0.1866218 1.603683e-05 -7.940128e-06 -4.120835e-08 2.730947e-08 -8.77692e-13

## Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class prediction calculations failed; returning NAs

## Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class probability calculations failed; returning NAs

## Warning in data.frame(..., check.names = FALSE): row names were found from a  
## short variable and have been discarded

## line search fails -1.865078 0.1996283 1.33039e-05 -7.489975e-06 -3.466265e-08 2.538234e-08 -6.512618e-13

## Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class prediction calculations failed; returning NAs

## Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class probability calculations failed; returning NAs

## Warning in data.frame(..., check.names = FALSE): row names were found from a  
## short variable and have been discarded

## line search fails -1.997873 0.2661491 1.310516e-05 -8.079409e-06 -3.845018e-08 2.820464e-08 -7.317727e-13

## Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class prediction calculations failed; returning NAs

## Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class probability calculations failed; returning NAs

## Warning in data.frame(..., check.names = FALSE): row names were found from a  
## short variable and have been discarded

## line search fails -1.895174 0.2170459 1.405272e-05 -8.046715e-06 -3.758173e-08 2.741642e-08 -7.487376e-13

## Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class prediction calculations failed; returning NAs

## Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class probability calculations failed; returning NAs

## Warning in data.frame(..., check.names = FALSE): row names were found from a  
## short variable and have been discarded

## line search fails -1.899546 0.2213272 1.425819e-05 -8.560835e-06 -3.881633e-08 2.888296e-08 -8.007128e-13

## Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class prediction calculations failed; returning NAs

## Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class probability calculations failed; returning NAs

## Warning in data.frame(..., check.names = FALSE): row names were found from a  
## short variable and have been discarded

## line search fails -2.046338 0.2952498 1.104447e-05 -7.185981e-06 -3.501144e-08 2.566699e-08 -5.711254e-13

## Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class prediction calculations failed; returning NAs

## Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class probability calculations failed; returning NAs

## Warning in data.frame(..., check.names = FALSE): row names were found from a  
## short variable and have been discarded

## line search fails -2.014078 0.251121 1.422056e-05 -8.179751e-06 -4.120178e-08 2.887861e-08 -8.221323e-13

## Warning in method$predict(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class prediction calculations failed; returning NAs

## Warning in method$prob(modelFit = modelFit, newdata = newdata, submodels =  
## param): kernlab class probability calculations failed; returning NAs

## Warning in data.frame(..., check.names = FALSE): row names were found from a  
## short variable and have been discarded

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo, :  
## There were missing values in resampled performance measures.

svm.train

## Support Vector Machines with Radial Basis Function Kernel   
##   
## 7280 samples  
## 17 predictor  
## 2 classes: 'yes', 'no'   
##   
## Pre-processing: centered (44), scaled (44)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 6553, 6552, 6552, 6551, 6551, 6553, ...   
## Resampling results across tuning parameters:  
##   
## sigma C ROC Sens Spec   
## 0.010 1 0.8692894 0.6370240 0.9132340  
## 0.010 2 0.8810828 0.6837158 0.9150662  
## 0.010 3 0.8868601 0.7001942 0.9180411  
## 0.010 4 0.8910269 0.7084393 0.9212468  
## 0.010 5 0.8949956 0.7180457 0.9241953  
## 0.010 6 0.8933099 0.7148884 0.9253288  
## 0.010 7 0.8987566 0.7245846 0.9272044  
## 0.010 8 0.9007725 0.7283646 0.9288062  
## 0.010 9 0.9025482 0.7314551 0.9294922  
## 0.010 10 0.9038702 0.7345455 0.9297215  
## 0.015 1 0.8804394 0.6823424 0.9127758  
## 0.015 2 0.8920116 0.7156499 0.9180426  
## 0.015 3 0.8963298 0.7279653 0.9206368  
## 0.015 4 0.9038529 0.7347641 0.9281669  
## 0.015 5 0.9049467 0.7396990 0.9253726  
## 0.015 6 0.9072971 0.7462246 0.9267462  
## 0.015 7 0.9073826 0.7458993 0.9262486  
## 0.015 8 0.9099123 0.7469166 0.9278909  
## 0.015 9 0.9105966 0.7506920 0.9274332  
## 0.015 10 0.9110459 0.7524114 0.9285774  
## 0.200 1 0.8953360 0.8492397 0.8141005  
## 0.200 2 0.8934862 0.8512993 0.8099794  
## 0.200 3 0.8916977 0.8509591 0.8092940  
## 0.200 4 0.8902837 0.8512993 0.8081493  
## 0.200 5 0.8903146 0.8495846 0.8095239  
## 0.200 6 0.8900588 0.8516452 0.8070051  
## 0.200 7 0.8893439 0.8530128 0.8067773  
## 0.200 8 0.8890671 0.8499247 0.8065496  
## 0.200 9 0.8885065 0.8509568 0.8056311  
## 0.200 10 0.8883360 0.8495799 0.8072366  
##   
## ROC was used to select the optimal model using the largest value.  
## The final values used for the model were sigma = 0.015 and C = 10.

d.svm.roc<- roc(response= svm.train$pred$obs, predictor=svm.train$pred$yes)

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

This was quite a bit of calculation for an improvement of 0.0003247 in the ROC score, but it shows off some of what caret can do.

plot(svm.train)

A close up of a map

Description automatically generated

test.svm.prob <- predict(svm.train, valid, type="prob")  
test.svm.class <- predict(svm.train, valid)   
confusionMatrix(test.svm.class , valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 149 267  
## no 162 2197  
##   
## Accuracy : 0.8454   
## 95% CI : (0.8314, 0.8587)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 1   
##   
## Kappa : 0.3231   
##   
## Mcnemar's Test P-Value : 5.136e-07   
##   
## Sensitivity : 0.47910   
## Specificity : 0.89164   
## Pos Pred Value : 0.35817   
## Neg Pred Value : 0.93133   
## Prevalence : 0.11207   
## Detection Rate : 0.05369   
## Detection Prevalence : 0.14991   
## Balanced Accuracy : 0.68537   
##   
## 'Positive' Class : yes   
##

test.svm.roc<- roc(response= valid$y, predictor=test.svm.prob[[1]])

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

svm\_accuracy <- mean(valid$y == test.svm.class )

GBM fter reading in the data and dividing it into training and test data sets, caret’s trainControl() and expand.grid() functions are used to set up to train the gbm model on all of the combinations of represented in the data frame built by expand.grid(). Then train() function does the actual training and fitting of the model..

set.seed(100)  
gbmGrid <-expand.grid(interaction.depth = c(1, 5, 9),   
 n.trees = (1:30),   
 shrinkage = 0.1,  
 n.minobsinnode = 20)  
boost.train <- train(y ~ ., data=smote\_train,   
 method="gbm",  
 metric="ROC",  
 trControl=ctrl,  
 tuneGrid=gbmGrid)

## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2881 nan 0.1000 0.0288  
## 2 1.2422 nan 0.1000 0.0230  
## 3 1.2053 nan 0.1000 0.0185  
## 4 1.1753 nan 0.1000 0.0145  
## 5 1.1481 nan 0.1000 0.0131  
## 6 1.1267 nan 0.1000 0.0105  
## 7 1.1075 nan 0.1000 0.0094  
## 8 1.0923 nan 0.1000 0.0075  
## 9 1.0782 nan 0.1000 0.0071  
## 10 1.0664 nan 0.1000 0.0059  
## 20 1.0039 nan 0.1000 0.0022  
## 30 0.9703 nan 0.1000 0.0011  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2742 nan 0.1000 0.0348  
## 2 1.2103 nan 0.1000 0.0313  
## 3 1.1632 nan 0.1000 0.0231  
## 4 1.1083 nan 0.1000 0.0277  
## 5 1.0713 nan 0.1000 0.0182  
## 6 1.0301 nan 0.1000 0.0200  
## 7 0.9934 nan 0.1000 0.0181  
## 8 0.9585 nan 0.1000 0.0161  
## 9 0.9291 nan 0.1000 0.0143  
## 10 0.9021 nan 0.1000 0.0125  
## 20 0.7321 nan 0.1000 0.0064  
## 30 0.6453 nan 0.1000 0.0044  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2377 nan 0.1000 0.0521  
## 2 1.1451 nan 0.1000 0.0460  
## 3 1.0689 nan 0.1000 0.0382  
## 4 1.0052 nan 0.1000 0.0312  
## 5 0.9511 nan 0.1000 0.0274  
## 6 0.9089 nan 0.1000 0.0204  
## 7 0.8817 nan 0.1000 0.0134  
## 8 0.8446 nan 0.1000 0.0183  
## 9 0.8239 nan 0.1000 0.0097  
## 10 0.7946 nan 0.1000 0.0141  
## 20 0.6291 nan 0.1000 0.0054  
## 30 0.5670 nan 0.1000 0.0017  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2906 nan 0.1000 0.0285  
## 2 1.2455 nan 0.1000 0.0234  
## 3 1.2068 nan 0.1000 0.0184  
## 4 1.1751 nan 0.1000 0.0148  
## 5 1.1497 nan 0.1000 0.0119  
## 6 1.1273 nan 0.1000 0.0115  
## 7 1.1086 nan 0.1000 0.0091  
## 8 1.0931 nan 0.1000 0.0078  
## 9 1.0800 nan 0.1000 0.0068  
## 10 1.0676 nan 0.1000 0.0055  
## 20 1.0041 nan 0.1000 0.0020  
## 30 0.9711 nan 0.1000 0.0010  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2684 nan 0.1000 0.0385  
## 2 1.2105 nan 0.1000 0.0294  
## 3 1.1603 nan 0.1000 0.0247  
## 4 1.1211 nan 0.1000 0.0194  
## 5 1.0688 nan 0.1000 0.0263  
## 6 1.0345 nan 0.1000 0.0169  
## 7 0.9999 nan 0.1000 0.0175  
## 8 0.9665 nan 0.1000 0.0156  
## 9 0.9352 nan 0.1000 0.0155  
## 10 0.8993 nan 0.1000 0.0186  
## 20 0.7211 nan 0.1000 0.0062  
## 30 0.6383 nan 0.1000 0.0017  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2369 nan 0.1000 0.0534  
## 2 1.1733 nan 0.1000 0.0324  
## 3 1.0969 nan 0.1000 0.0380  
## 4 1.0469 nan 0.1000 0.0241  
## 5 1.0078 nan 0.1000 0.0196  
## 6 0.9483 nan 0.1000 0.0294  
## 7 0.9053 nan 0.1000 0.0211  
## 8 0.8631 nan 0.1000 0.0205  
## 9 0.8269 nan 0.1000 0.0186  
## 10 0.8087 nan 0.1000 0.0086  
## 20 0.6330 nan 0.1000 0.0051  
## 30 0.5543 nan 0.1000 0.0017  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2914 nan 0.1000 0.0288  
## 2 1.2446 nan 0.1000 0.0228  
## 3 1.2073 nan 0.1000 0.0187  
## 4 1.1761 nan 0.1000 0.0151  
## 5 1.1523 nan 0.1000 0.0123  
## 6 1.1299 nan 0.1000 0.0114  
## 7 1.1109 nan 0.1000 0.0088  
## 8 1.0948 nan 0.1000 0.0083  
## 9 1.0817 nan 0.1000 0.0068  
## 10 1.0697 nan 0.1000 0.0055  
## 20 1.0036 nan 0.1000 0.0019  
## 30 0.9693 nan 0.1000 0.0015  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2741 nan 0.1000 0.0352  
## 2 1.2166 nan 0.1000 0.0282  
## 3 1.1621 nan 0.1000 0.0264  
## 4 1.1038 nan 0.1000 0.0289  
## 5 1.0581 nan 0.1000 0.0233  
## 6 1.0266 nan 0.1000 0.0150  
## 7 0.9834 nan 0.1000 0.0206  
## 8 0.9543 nan 0.1000 0.0145  
## 9 0.9353 nan 0.1000 0.0092  
## 10 0.9090 nan 0.1000 0.0125  
## 20 0.7276 nan 0.1000 0.0024  
## 30 0.6366 nan 0.1000 0.0033  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2603 nan 0.1000 0.0431  
## 2 1.1872 nan 0.1000 0.0366  
## 3 1.1052 nan 0.1000 0.0414  
## 4 1.0395 nan 0.1000 0.0327  
## 5 1.0024 nan 0.1000 0.0179  
## 6 0.9618 nan 0.1000 0.0203  
## 7 0.9139 nan 0.1000 0.0234  
## 8 0.8897 nan 0.1000 0.0115  
## 9 0.8530 nan 0.1000 0.0183  
## 10 0.8230 nan 0.1000 0.0144  
## 20 0.6375 nan 0.1000 0.0028  
## 30 0.5591 nan 0.1000 0.0015  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2871 nan 0.1000 0.0291  
## 2 1.2410 nan 0.1000 0.0232  
## 3 1.2025 nan 0.1000 0.0187  
## 4 1.1708 nan 0.1000 0.0152  
## 5 1.1456 nan 0.1000 0.0129  
## 6 1.1231 nan 0.1000 0.0105  
## 7 1.1045 nan 0.1000 0.0098  
## 8 1.0883 nan 0.1000 0.0081  
## 9 1.0741 nan 0.1000 0.0067  
## 10 1.0623 nan 0.1000 0.0059  
## 20 0.9967 nan 0.1000 0.0020  
## 30 0.9642 nan 0.1000 0.0013  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2687 nan 0.1000 0.0379  
## 2 1.1937 nan 0.1000 0.0367  
## 3 1.1333 nan 0.1000 0.0294  
## 4 1.0942 nan 0.1000 0.0200  
## 5 1.0547 nan 0.1000 0.0193  
## 6 1.0164 nan 0.1000 0.0188  
## 7 0.9821 nan 0.1000 0.0165  
## 8 0.9426 nan 0.1000 0.0197  
## 9 0.9188 nan 0.1000 0.0115  
## 10 0.8896 nan 0.1000 0.0145  
## 20 0.7333 nan 0.1000 0.0021  
## 30 0.6425 nan 0.1000 0.0031  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2367 nan 0.1000 0.0537  
## 2 1.1480 nan 0.1000 0.0435  
## 3 1.0937 nan 0.1000 0.0257  
## 4 1.0410 nan 0.1000 0.0269  
## 5 0.9828 nan 0.1000 0.0291  
## 6 0.9324 nan 0.1000 0.0248  
## 7 0.9045 nan 0.1000 0.0135  
## 8 0.8796 nan 0.1000 0.0124  
## 9 0.8468 nan 0.1000 0.0158  
## 10 0.8142 nan 0.1000 0.0155  
## 20 0.6317 nan 0.1000 0.0050  
## 30 0.5536 nan 0.1000 0.0020  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2886 nan 0.1000 0.0279  
## 2 1.2422 nan 0.1000 0.0228  
## 3 1.2037 nan 0.1000 0.0179  
## 4 1.1746 nan 0.1000 0.0144  
## 5 1.1487 nan 0.1000 0.0133  
## 6 1.1279 nan 0.1000 0.0108  
## 7 1.1094 nan 0.1000 0.0094  
## 8 1.0928 nan 0.1000 0.0081  
## 9 1.0789 nan 0.1000 0.0069  
## 10 1.0665 nan 0.1000 0.0059  
## 20 1.0011 nan 0.1000 0.0021  
## 30 0.9660 nan 0.1000 0.0015  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2743 nan 0.1000 0.0358  
## 2 1.2163 nan 0.1000 0.0285  
## 3 1.1475 nan 0.1000 0.0340  
## 4 1.0952 nan 0.1000 0.0262  
## 5 1.0557 nan 0.1000 0.0191  
## 6 1.0162 nan 0.1000 0.0193  
## 7 0.9883 nan 0.1000 0.0135  
## 8 0.9654 nan 0.1000 0.0109  
## 9 0.9263 nan 0.1000 0.0188  
## 10 0.9027 nan 0.1000 0.0116  
## 20 0.7294 nan 0.1000 0.0061  
## 30 0.6338 nan 0.1000 0.0031  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2669 nan 0.1000 0.0386  
## 2 1.1751 nan 0.1000 0.0446  
## 3 1.1192 nan 0.1000 0.0269  
## 4 1.0454 nan 0.1000 0.0367  
## 5 0.9822 nan 0.1000 0.0309  
## 6 0.9301 nan 0.1000 0.0256  
## 7 0.8935 nan 0.1000 0.0180  
## 8 0.8554 nan 0.1000 0.0185  
## 9 0.8217 nan 0.1000 0.0165  
## 10 0.7969 nan 0.1000 0.0122  
## 20 0.6277 nan 0.1000 0.0048  
## 30 0.5583 nan 0.1000 0.0019  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2885 nan 0.1000 0.0285  
## 2 1.2422 nan 0.1000 0.0232  
## 3 1.2045 nan 0.1000 0.0185  
## 4 1.1758 nan 0.1000 0.0141  
## 5 1.1491 nan 0.1000 0.0135  
## 6 1.1269 nan 0.1000 0.0110  
## 7 1.1087 nan 0.1000 0.0092  
## 8 1.0919 nan 0.1000 0.0079  
## 9 1.0793 nan 0.1000 0.0067  
## 10 1.0675 nan 0.1000 0.0060  
## 20 1.0021 nan 0.1000 0.0020  
## 30 0.9703 nan 0.1000 0.0010  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2682 nan 0.1000 0.0387  
## 2 1.2046 nan 0.1000 0.0306  
## 3 1.1569 nan 0.1000 0.0238  
## 4 1.1013 nan 0.1000 0.0274  
## 5 1.0563 nan 0.1000 0.0223  
## 6 1.0267 nan 0.1000 0.0144  
## 7 0.9886 nan 0.1000 0.0182  
## 8 0.9517 nan 0.1000 0.0182  
## 9 0.9184 nan 0.1000 0.0167  
## 10 0.8925 nan 0.1000 0.0128  
## 20 0.7213 nan 0.1000 0.0057  
## 30 0.6331 nan 0.1000 0.0029  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2572 nan 0.1000 0.0432  
## 2 1.1884 nan 0.1000 0.0340  
## 3 1.1059 nan 0.1000 0.0411  
## 4 1.0378 nan 0.1000 0.0333  
## 5 0.9830 nan 0.1000 0.0277  
## 6 0.9396 nan 0.1000 0.0216  
## 7 0.8966 nan 0.1000 0.0211  
## 8 0.8725 nan 0.1000 0.0117  
## 9 0.8363 nan 0.1000 0.0181  
## 10 0.8075 nan 0.1000 0.0141  
## 20 0.6396 nan 0.1000 0.0037  
## 30 0.5606 nan 0.1000 0.0022  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2899 nan 0.1000 0.0291  
## 2 1.2431 nan 0.1000 0.0233  
## 3 1.2061 nan 0.1000 0.0186  
## 4 1.1770 nan 0.1000 0.0149  
## 5 1.1512 nan 0.1000 0.0123  
## 6 1.1277 nan 0.1000 0.0113  
## 7 1.1084 nan 0.1000 0.0092  
## 8 1.0930 nan 0.1000 0.0081  
## 9 1.0796 nan 0.1000 0.0063  
## 10 1.0672 nan 0.1000 0.0061  
## 20 1.0032 nan 0.1000 0.0021  
## 30 0.9704 nan 0.1000 0.0010  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2683 nan 0.1000 0.0386  
## 2 1.2082 nan 0.1000 0.0296  
## 3 1.1473 nan 0.1000 0.0298  
## 4 1.0931 nan 0.1000 0.0270  
## 5 1.0495 nan 0.1000 0.0218  
## 6 1.0193 nan 0.1000 0.0146  
## 7 0.9948 nan 0.1000 0.0121  
## 8 0.9586 nan 0.1000 0.0173  
## 9 0.9327 nan 0.1000 0.0122  
## 10 0.9084 nan 0.1000 0.0115  
## 20 0.7295 nan 0.1000 0.0073  
## 30 0.6420 nan 0.1000 0.0020  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2591 nan 0.1000 0.0435  
## 2 1.1946 nan 0.1000 0.0318  
## 3 1.1206 nan 0.1000 0.0377  
## 4 1.0512 nan 0.1000 0.0342  
## 5 1.0094 nan 0.1000 0.0197  
## 6 0.9616 nan 0.1000 0.0241  
## 7 0.9133 nan 0.1000 0.0230  
## 8 0.8698 nan 0.1000 0.0210  
## 9 0.8481 nan 0.1000 0.0107  
## 10 0.8202 nan 0.1000 0.0132  
## 20 0.6401 nan 0.1000 0.0050  
## 30 0.5565 nan 0.1000 0.0014  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2873 nan 0.1000 0.0288  
## 2 1.2422 nan 0.1000 0.0231  
## 3 1.2057 nan 0.1000 0.0184  
## 4 1.1760 nan 0.1000 0.0150  
## 5 1.1493 nan 0.1000 0.0128  
## 6 1.1274 nan 0.1000 0.0108  
## 7 1.1089 nan 0.1000 0.0091  
## 8 1.0933 nan 0.1000 0.0079  
## 9 1.0798 nan 0.1000 0.0065  
## 10 1.0679 nan 0.1000 0.0061  
## 20 1.0046 nan 0.1000 0.0020  
## 30 0.9718 nan 0.1000 0.0011  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2690 nan 0.1000 0.0374  
## 2 1.1951 nan 0.1000 0.0369  
## 3 1.1369 nan 0.1000 0.0280  
## 4 1.0981 nan 0.1000 0.0191  
## 5 1.0651 nan 0.1000 0.0162  
## 6 1.0230 nan 0.1000 0.0215  
## 7 0.9887 nan 0.1000 0.0169  
## 8 0.9578 nan 0.1000 0.0154  
## 9 0.9336 nan 0.1000 0.0118  
## 10 0.9013 nan 0.1000 0.0160  
## 20 0.7299 nan 0.1000 0.0063  
## 30 0.6397 nan 0.1000 0.0033  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2545 nan 0.1000 0.0448  
## 2 1.1572 nan 0.1000 0.0457  
## 3 1.0776 nan 0.1000 0.0384  
## 4 1.0163 nan 0.1000 0.0299  
## 5 0.9611 nan 0.1000 0.0272  
## 6 0.9231 nan 0.1000 0.0191  
## 7 0.8883 nan 0.1000 0.0170  
## 8 0.8570 nan 0.1000 0.0150  
## 9 0.8222 nan 0.1000 0.0173  
## 10 0.8037 nan 0.1000 0.0084  
## 20 0.6269 nan 0.1000 0.0065  
## 30 0.5595 nan 0.1000 0.0028  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2884 nan 0.1000 0.0285  
## 2 1.2442 nan 0.1000 0.0227  
## 3 1.2084 nan 0.1000 0.0184  
## 4 1.1781 nan 0.1000 0.0149  
## 5 1.1541 nan 0.1000 0.0124  
## 6 1.1317 nan 0.1000 0.0111  
## 7 1.1140 nan 0.1000 0.0090  
## 8 1.0973 nan 0.1000 0.0082  
## 9 1.0840 nan 0.1000 0.0066  
## 10 1.0720 nan 0.1000 0.0058  
## 20 1.0074 nan 0.1000 0.0022  
## 30 0.9745 nan 0.1000 0.0013  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2758 nan 0.1000 0.0340  
## 2 1.2024 nan 0.1000 0.0371  
## 3 1.1436 nan 0.1000 0.0292  
## 4 1.0894 nan 0.1000 0.0264  
## 5 1.0557 nan 0.1000 0.0161  
## 6 1.0180 nan 0.1000 0.0190  
## 7 0.9833 nan 0.1000 0.0171  
## 8 0.9619 nan 0.1000 0.0104  
## 9 0.9274 nan 0.1000 0.0164  
## 10 0.8969 nan 0.1000 0.0152  
## 20 0.7404 nan 0.1000 0.0022  
## 30 0.6469 nan 0.1000 0.0028  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2378 nan 0.1000 0.0540  
## 2 1.1504 nan 0.1000 0.0432  
## 3 1.0717 nan 0.1000 0.0392  
## 4 1.0047 nan 0.1000 0.0335  
## 5 0.9485 nan 0.1000 0.0277  
## 6 0.9172 nan 0.1000 0.0151  
## 7 0.8806 nan 0.1000 0.0179  
## 8 0.8450 nan 0.1000 0.0172  
## 9 0.8209 nan 0.1000 0.0116  
## 10 0.7912 nan 0.1000 0.0142  
## 20 0.6294 nan 0.1000 0.0066  
## 30 0.5545 nan 0.1000 0.0031  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2869 nan 0.1000 0.0285  
## 2 1.2417 nan 0.1000 0.0226  
## 3 1.2051 nan 0.1000 0.0184  
## 4 1.1752 nan 0.1000 0.0148  
## 5 1.1505 nan 0.1000 0.0120  
## 6 1.1279 nan 0.1000 0.0112  
## 7 1.1097 nan 0.1000 0.0091  
## 8 1.0939 nan 0.1000 0.0078  
## 9 1.0809 nan 0.1000 0.0066  
## 10 1.0691 nan 0.1000 0.0058  
## 20 1.0055 nan 0.1000 0.0022  
## 30 0.9735 nan 0.1000 0.0013  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2713 nan 0.1000 0.0371  
## 2 1.2089 nan 0.1000 0.0305  
## 3 1.1470 nan 0.1000 0.0311  
## 4 1.0959 nan 0.1000 0.0254  
## 5 1.0626 nan 0.1000 0.0161  
## 6 1.0112 nan 0.1000 0.0247  
## 7 0.9737 nan 0.1000 0.0183  
## 8 0.9415 nan 0.1000 0.0168  
## 9 0.9087 nan 0.1000 0.0162  
## 10 0.8808 nan 0.1000 0.0138  
## 20 0.7202 nan 0.1000 0.0053  
## 30 0.6421 nan 0.1000 0.0022  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2350 nan 0.1000 0.0565  
## 2 1.1521 nan 0.1000 0.0410  
## 3 1.0749 nan 0.1000 0.0383  
## 4 1.0299 nan 0.1000 0.0221  
## 5 0.9737 nan 0.1000 0.0277  
## 6 0.9287 nan 0.1000 0.0218  
## 7 0.8906 nan 0.1000 0.0186  
## 8 0.8521 nan 0.1000 0.0185  
## 9 0.8279 nan 0.1000 0.0120  
## 10 0.8000 nan 0.1000 0.0136  
## 20 0.6404 nan 0.1000 0.0041  
## 30 0.5620 nan 0.1000 0.0026  
##   
## Iter TrainDeviance ValidDeviance StepSize Improve  
## 1 1.2372 nan 0.1000 0.0538  
## 2 1.1435 nan 0.1000 0.0460  
## 3 1.0907 nan 0.1000 0.0257  
## 4 1.0216 nan 0.1000 0.0337  
## 5 0.9647 nan 0.1000 0.0279  
## 6 0.9204 nan 0.1000 0.0212  
## 7 0.8928 nan 0.1000 0.0137  
## 8 0.8549 nan 0.1000 0.0190  
## 9 0.8292 nan 0.1000 0.0119  
## 10 0.8005 nan 0.1000 0.0144  
## 20 0.6275 nan 0.1000 0.0039  
## 30 0.5544 nan 0.1000 0.0014

print(summary(boost.train))

A screenshot of a cell phone

Description automatically generated

## var rel.inf  
## nr.employed nr.employed 42.52439250  
## cons.price.idx cons.price.idx 15.66155548  
## cons.conf.idx cons.conf.idx 15.21297801  
## previous previous 10.47802178  
## emp.var.rate emp.var.rate 5.12677014  
## campaign campaign 3.73689113  
## monthmay monthmay 1.74547046  
## poutcomesuccess poutcomesuccess 1.65328032  
## euribor3m euribor3m 1.46562637  
## loanyes loanyes 1.19647498  
## poutcomenonexistent poutcomenonexistent 0.53139006  
## age age 0.27083555  
## maritalmarried maritalmarried 0.06941238  
## maritalsingle maritalsingle 0.06173131  
## jobretired jobretired 0.05919146  
## day\_of\_weekmon day\_of\_weekmon 0.05376674  
## day\_of\_weekthu day\_of\_weekthu 0.05205509  
## jobunemployed jobunemployed 0.03508424  
## monthmar monthmar 0.03391322  
## monthoct monthoct 0.03115878  
## jobblue-collar jobblue-collar 0.00000000  
## jobentrepreneur jobentrepreneur 0.00000000  
## jobhousemaid jobhousemaid 0.00000000  
## jobmanagement jobmanagement 0.00000000  
## jobself-employed jobself-employed 0.00000000  
## jobservices jobservices 0.00000000  
## jobstudent jobstudent 0.00000000  
## jobtechnician jobtechnician 0.00000000  
## educationbasic.6y educationbasic.6y 0.00000000  
## educationbasic.9y educationbasic.9y 0.00000000  
## educationhigh.school educationhigh.school 0.00000000  
## educationilliterate educationilliterate 0.00000000  
## educationprofessional.course educationprofessional.course 0.00000000  
## educationuniversity.degree educationuniversity.degree 0.00000000  
## housingyes housingyes 0.00000000  
## contacttelephone contacttelephone 0.00000000  
## monthaug monthaug 0.00000000  
## monthdec monthdec 0.00000000  
## monthjul monthjul 0.00000000  
## monthjun monthjun 0.00000000  
## monthnov monthnov 0.00000000  
## monthsep monthsep 0.00000000  
## day\_of\_weektue day\_of\_weektue 0.00000000  
## day\_of\_weekwed day\_of\_weekwed 0.00000000

d.gbm.roc<- roc(response= boost.train$pred$obs, predictor=boost.train$pred$yes)

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

plot(boost.train)

A close up of a map

Description automatically generated

test.gbm.prob <- predict(boost.train, valid, type="prob")  
test.gbm.class <- predict(boost.train, valid)   
confusionMatrix(test.gbm.class , valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 123 137  
## no 188 2327  
##   
## Accuracy : 0.8829   
## 95% CI : (0.8703, 0.8946)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 0.809068   
##   
## Kappa : 0.3661   
##   
## Mcnemar's Test P-Value : 0.005546   
##   
## Sensitivity : 0.39550   
## Specificity : 0.94440   
## Pos Pred Value : 0.47308   
## Neg Pred Value : 0.92525   
## Prevalence : 0.11207   
## Detection Rate : 0.04432   
## Detection Prevalence : 0.09369   
## Balanced Accuracy : 0.66995   
##   
## 'Positive' Class : yes   
##

test.gbm.roc<- roc(response= valid$y, predictor=test.gbm.prob[[1]])

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

gbm\_accuracy <- mean(valid$y == test.gbm.class )

histogram(~test.gbm.prob[[1]]|valid$y,xlab="Probability of Poor Segmentation")

A picture containing screenshot

Description automatically generated

KNN

set.seed(400)  
knnFit <- train(y ~ ., data = train, method = "knn",metric="ROC", trControl = ctrl, preProcess = c("center","scale"), tuneGrid = expand.grid(k = c(5:25)))  
print(knnFit)

## k-Nearest Neighbors   
##   
## 6480 samples  
## 17 predictor  
## 2 classes: 'yes', 'no'   
##   
## Pre-processing: centered (44), scaled (44)   
## Resampling: Cross-Validated (10 fold)   
## Summary of sample sizes: 5832, 5833, 5832, 5831, 5832, 5832, ...   
## Resampling results across tuning parameters:  
##   
## k ROC Sens Spec   
## 5 0.7010827 0.2361682 0.9695782  
## 6 0.7086062 0.2349315 0.9725317  
## 7 0.7154007 0.2128425 0.9746178  
## 8 0.7170618 0.2019216 0.9772261  
## 9 0.7176316 0.2005327 0.9775728  
## 10 0.7160108 0.2183790 0.9786165  
## 11 0.7195783 0.2032725 0.9803554  
## 12 0.7204937 0.2019406 0.9801812  
## 13 0.7238981 0.2019216 0.9808768  
## 14 0.7279212 0.1977930 0.9798333  
## 15 0.7303902 0.1978120 0.9822672  
## 16 0.7280765 0.1937024 0.9820930  
## 17 0.7279445 0.1937024 0.9827889  
## 18 0.7303276 0.1909627 0.9831362  
## 19 0.7328327 0.1895928 0.9833098  
## 20 0.7334533 0.1923326 0.9838321  
## 21 0.7346724 0.1923516 0.9838315  
## 22 0.7354081 0.1909817 0.9836579  
## 23 0.7373682 0.1978120 0.9841793  
## 24 0.7405721 0.1909627 0.9843530  
## 25 0.7413561 0.1950723 0.9836579  
##   
## ROC was used to select the optimal model using the largest value.  
## The final value used for the model was k = 25.

d.knn.roc<- roc(response= knnFit$pred$obs, predictor=knnFit$pred$yes)

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

test.knn.prob <- predict(knnFit, valid, type="prob")  
test.knn.class <- predict(knnFit, valid)   
confusionMatrix(test.knn.class , valid$y)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction yes no  
## yes 65 46  
## no 246 2418  
##   
## Accuracy : 0.8948   
## 95% CI : (0.8828, 0.906)  
## No Information Rate : 0.8879   
## P-Value [Acc > NIR] : 0.1324   
##   
## Kappa : 0.2647   
##   
## Mcnemar's Test P-Value : <2e-16   
##   
## Sensitivity : 0.20900   
## Specificity : 0.98133   
## Pos Pred Value : 0.58559   
## Neg Pred Value : 0.90766   
## Prevalence : 0.11207   
## Detection Rate : 0.02342   
## Detection Prevalence : 0.04000   
## Balanced Accuracy : 0.59517   
##   
## 'Positive' Class : yes   
##

test.knn.roc<- roc(response= valid$y, predictor=test.knn.prob[[1]])

## Setting levels: control = yes, case = no

## Setting direction: controls > cases

knn\_accuracy <- mean(valid$y == test.knn.class )

histogram(~test.knn.prob[[1]]|valid$y,xlab="Probability of Poor Segmentation")

A picture containing screenshot

Description automatically generated

Training ROCS

plot(d.log.roc, legacy.axes=T,main= "Train ROC curves for Log,SVM,GBM,KNN and RF models")  
plot(d.svm.roc, add=T, col="Yellow")  
plot(d.gbm.roc, add=T, col="Blue")  
plot(d.knn.roc, add=T, col="Red")  
plot(d.rf.roc, add=T, col="Orange")  
plot(d.rpart.roc, add=T, col="Pink")  
plot(d.lda.roc, add=T, col="Green")  
legend(x=.2, y=.7, legend=c("Logistic Regression", "SVM", "GBM","KNN","RF","DT","LDA"), col=c("black","Yellow","Blue","Red","Orange","Pink","Green"),lty=1)

A close up of a map

Description automatically generated

Test ROC

plot(test.log.roc, legacy.axes=T,main= "Test ROC curves for Log,SVM,GBM,KNN and RF models")  
plot(test.svm.roc, add=T, col="Yellow")  
plot(test.gbm.roc, add=T, col="Blue")  
plot(test.knn.roc, add=T, col="Red")  
plot(test.rf.roc, add=T, col="Orange")  
plot(test.rpart.roc, add=T, col="Pink")  
plot(test.lda.roc, add=T, col="Green")  
legend(x=.2, y=.7, legend=c("Logistic Regression", "SVM", "GBM","KNN","RF","DT","LDA"), col=c("black","Yellow","Blue","Red","Orange","Pink","Green"),lty=1)

A close up of a map

Description automatically generated

Test AUC Cmparison

result\_auc <- data\_frame("Models"=c("Logistic Regression","SVM","GBM","KNN","RF","Decision Tree"),  
 "AUC"=c(auc(test.log.roc),auc(test.svm.roc),auc(test.gbm.roc),auc(test.knn.roc),auc(test.rf.roc),auc(test.rpart.roc)))

## Warning: `data\_frame()` is deprecated as of tibble 1.1.0.  
## Please use `tibble()` instead.  
## This warning is displayed once every 8 hours.  
## Call `lifecycle::last\_warnings()` to see where this warning was generated.

print(result\_auc)

## # A tibble: 6 x 2  
## Models AUC  
## <chr> <dbl>  
## 1 Logistic Regression 0.749  
## 2 SVM 0.729  
## 3 GBM 0.769  
## 4 KNN 0.750  
## 5 RF 0.747  
## 6 Decision Tree 0.774

Test Accuracy Comparison

accuracy <- data\_frame("Models"=c("Logistic Regression","Log\_PCA","LDA","GBM","KNN","RF","RF\_PCA","SVM","Decision Tree"),  
 "AUC"=c(auc(test.log.roc),auc(test.pca.roc),auc(test.lda.roc),auc(test.gbm.roc),auc(test.knn.roc),auc(test.rf.roc),auc(test.rf.pca.roc),auc(test.svm.roc),auc(test.rpart.roc)),  
 "Test Accuracy"=c(log\_accuracy,log\_pca\_accuracy,lda\_accuracy,gbm\_accuracy,knn\_accuracy,rf\_accuracy,rf\_pca\_accuracy,svm\_accuracy,rpart\_accuracy))  
print(accuracy)

## # A tibble: 9 x 3  
## Models AUC `Test Accuracy`  
## <chr> <dbl> <dbl>  
## 1 Logistic Regression 0.749 0.845  
## 2 Log\_PCA 0.746 0.843  
## 3 LDA 0.756 0.858  
## 4 GBM 0.769 0.883  
## 5 KNN 0.750 0.895  
## 6 RF 0.747 0.876  
## 7 RF\_PCA 0.740 0.814  
## 8 SVM 0.729 0.845  
## 9 Decision Tree 0.774 0.868

rValues <- resamples(list(glm = glm.train.fit,lda=lda.train.fit, rf=rf\_random,svm=svm.train,gbm=boost.train,knn=knnFit,DT=rpart.train))  
#rValues$values  
print(summary(rValues))

##   
## Call:  
## summary.resamples(object = rValues)  
##   
## Models: glm, lda, rf, svm, gbm, knn, DT   
## Number of resamples: 10   
##   
## ROC   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## glm 0.7927450 0.8162387 0.8273872 0.8224775 0.8306672 0.8431275 0  
## lda 0.7927922 0.8133154 0.8234156 0.8214482 0.8314992 0.8417907 0  
## rf 0.9457682 0.9524410 0.9566981 0.9583730 0.9662607 0.9728821 0  
## svm 0.8993214 0.9021425 0.9087082 0.9110459 0.9217075 0.9252139 0  
## gbm 0.9263888 0.9378320 0.9421941 0.9411170 0.9454546 0.9508088 0  
## knn 0.7128410 0.7240446 0.7402566 0.7413561 0.7603425 0.7655774 0  
## DT 0.8769113 0.8905124 0.9022958 0.8993383 0.9061561 0.9186253 0  
##   
## Sens   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## glm 0.5773196 0.6073883 0.6192157 0.6208786 0.6263916 0.6838488 0  
## lda 0.5704467 0.5893471 0.5996564 0.6067916 0.6145936 0.6769759 0  
## rf 0.8150685 0.8363343 0.8402062 0.8465118 0.8608247 0.8865979 0  
## svm 0.7226027 0.7336770 0.7529892 0.7524114 0.7689003 0.7800687 0  
## gbm 0.8041237 0.8176370 0.8316151 0.8296768 0.8367698 0.8591065 0  
## knn 0.1506849 0.1643836 0.1917808 0.1950723 0.2191781 0.2500000 0  
## DT 0.7835052 0.8213058 0.8301794 0.8303629 0.8470790 0.8694158 0  
##   
## Spec   
## Min. 1st Qu. Median Mean 3rd Qu. Max. NA's  
## glm 0.8784404 0.8878719 0.8923252 0.8962862 0.9038902 0.9176201 0  
## lda 0.8832952 0.8935336 0.9016018 0.9029297 0.9078409 0.9313501 0  
## rf 0.9542334 0.9639431 0.9725400 0.9693206 0.9742563 0.9794050 0  
## svm 0.9153318 0.9221968 0.9302059 0.9285774 0.9346724 0.9472477 0  
## gbm 0.9405034 0.9473684 0.9496568 0.9514680 0.9535826 0.9656751 0  
## knn 0.9756522 0.9808696 0.9826238 0.9836579 0.9869633 0.9930435 0  
## DT 0.9061785 0.9107551 0.9152347 0.9171247 0.9250113 0.9290618 0

bwplot(rValues, layout = c(7, 1))

A screenshot of a cell phone

Description automatically generated

bwplot(rValues,metric="ROC",main=" GLM vs LDA vs RF vs SVM vs GBM vs KNN vs DT") # boxplot

A screenshot of a cell phone

Description automatically generated