Bank Marketing Prediction
MA 5790

Goutham Thota, Jaya Surya Thota

December 2023

Introduction:

In the current financial setting, a bank's performance is largely determined by how well its marketing campaigns operate, especially when it comes to advertising term deposit services. As new marketing strategies such as phone campaigns take off, financial institutions are under pressure to find new customers quickly and improve the services they provide. In light of this, the application of data mining techniques has attracted a lot of interest, particularly when it comes to accurately and perceptively managing enormous volumes of data.

The goal of this study is to investigate the effectiveness of predictive models in relation to direct marketing campaigns for term deposit subscriptions run by a Portuguese banking company. Using a dataset with several samples and predictive variables, the main objective is to predict whether or not clients would sign up for term deposits. The objective is to identify the best predictive model for customer subscription prediction by analyzing several models, such as Support Vector Machines, Neural Networks, etc.

The methodology involves numerous stages of carefully understanding the data, creating dummy variables for all the categorical columns, resolving imbalance in the dataset, and carefully spending our data into training and testing. The ultimate goal is to identify a best predictive model that can forecast consumer subscriptions with enough accuracy to offer crucial insights to improve customer engagement and future marketing tactics. This study aims to improve marketing precision in the banking industry by utilizing cutting-edge data-driven techniques. This would enable institutions to better customize their tactics and build deeper relationships with their clients.

Data:

The dataset involves information about marketing campaigns conducted by a Portuguese bank through phone calls. In these campaigns, multiple contacts with the same client were made to determine whether the client would subscribe to a bank term deposit ('yes') or not ('no'). Here's a simple breakdown:

- **Number of Samples:** The dataset includes records from 45,210 marketing interactions.
- **Number of Variables:** There are 17 different aspects of information considered in each interaction.
- **Categorical Predictors:** 9 of these variables are categorical, indicating characteristics with distinct categories (e.g., job type, marital status).

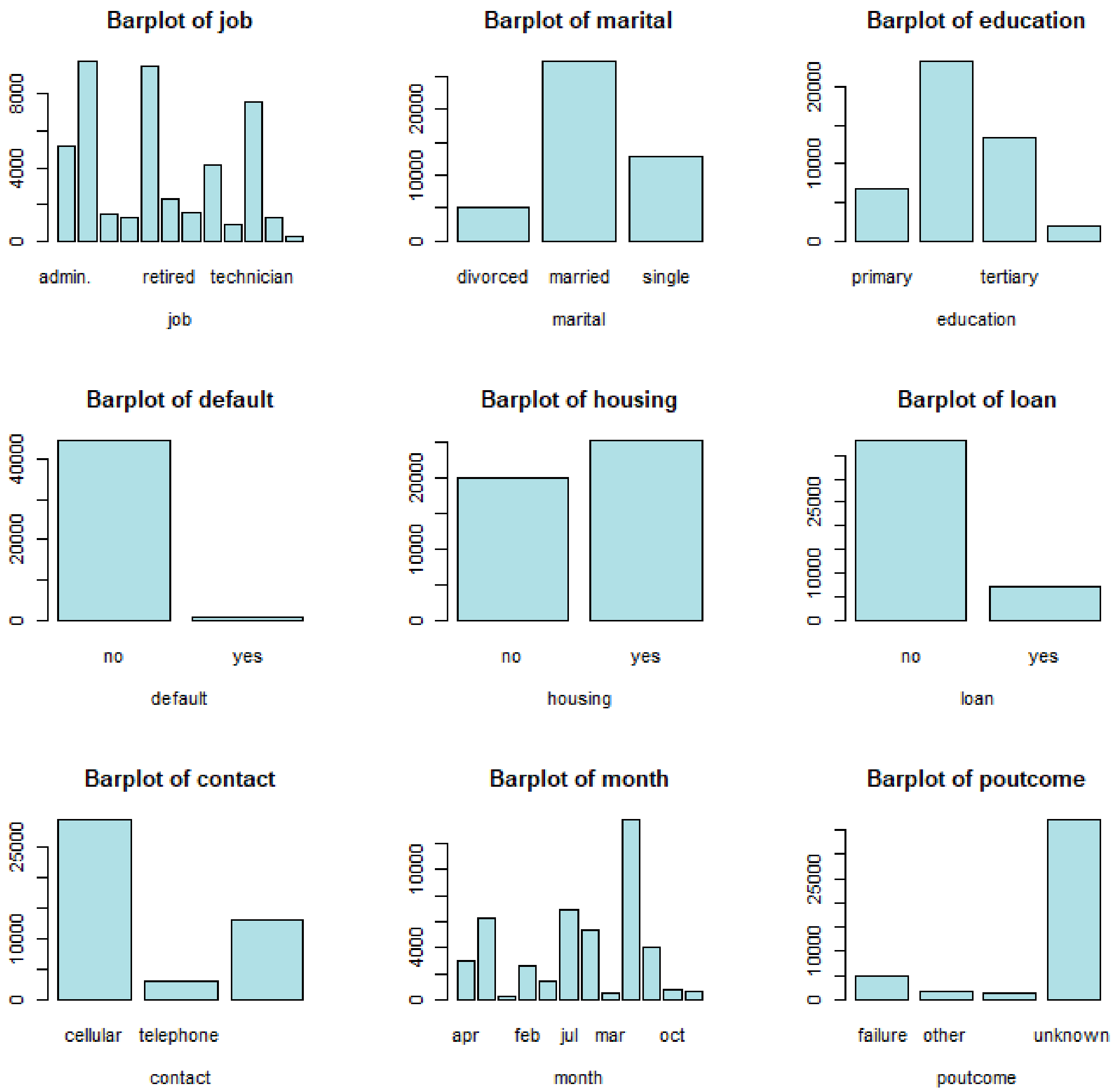
- **Continuous Predictors:** 7 variables are continuous, representing numerical data (e.g., age, balance).
- **Target Variable:** There is 1 main variable of interest, which is likely whether the client subscribed to the bank term deposit ('yes') or not ('no').

Below table explains the description of predictors:

Categorical Predictors (Binary)	
Variable Name	**Description**
Default	Has credit in default?
Housing	Has housing loan?
Loan	Has personal loan?
Deposit (Target)	has the client subscribed a term deposit?
Categorical Predictors	
Job	Type of job : admin, blue-collar, entrepreneur, etc.
Marital	Marital Status
Education	Education level
Contact	Contact communication type: cellular, telephone
Month	Last contact month of year
Poutcome	Outcome of the previous marketing campaignt month of year: failure, nonexistent , success
Continuous Predictors	
Age	Age
Balance	average yearly balance
Day	Last contact day of the week

Duration	last contact duration, in seconds
Campaign	number of contacts performed during this campaign and for this client
Pdays	number of days that passed by after the client was last contacted from a previous campaign
Previous	number of contacts performed before this campaign and for this client

Table 1. Description of Predictors



Distribution of Categorical Predictors

Preprocessing:

Missing Data

Since our data doesn't have any missing values, we didn't perform any imputation.

Dummy Variables

Converted categorical predictors into dummy variables using the "dummyVars" function. We got a total of 48 predictors after adding dummy variables.

Near Zero Variance

After creating dummy variables, we checked for degenerates and removed them. These are the near zero predictors that we found using "nearZeroVar" function, "default", "pdays", "jobentrepreneur", "jobhousemaid", "jobself-employed", "jobstudent", "jobunemployed", "jobunknown", "educationunknown", "monthdec", "monthjan", "monthmar", "monthoct", "monthsep", "poutcomeother", "poutcomesuccess". We left with 32 predictors, 26 being categorical and 6 are continuous.

Correlations

A plot showing the correlation among the remaining 32 predictors was generated to examine how they relate to each other. In the plot, blue indicates positive correlations, while red indicates negative ones. Predictors with correlations greater than 0.65 were removed and those predictors are "contactunknown", "educationtertiary", "poutcomeunknown" "maritalmarried".

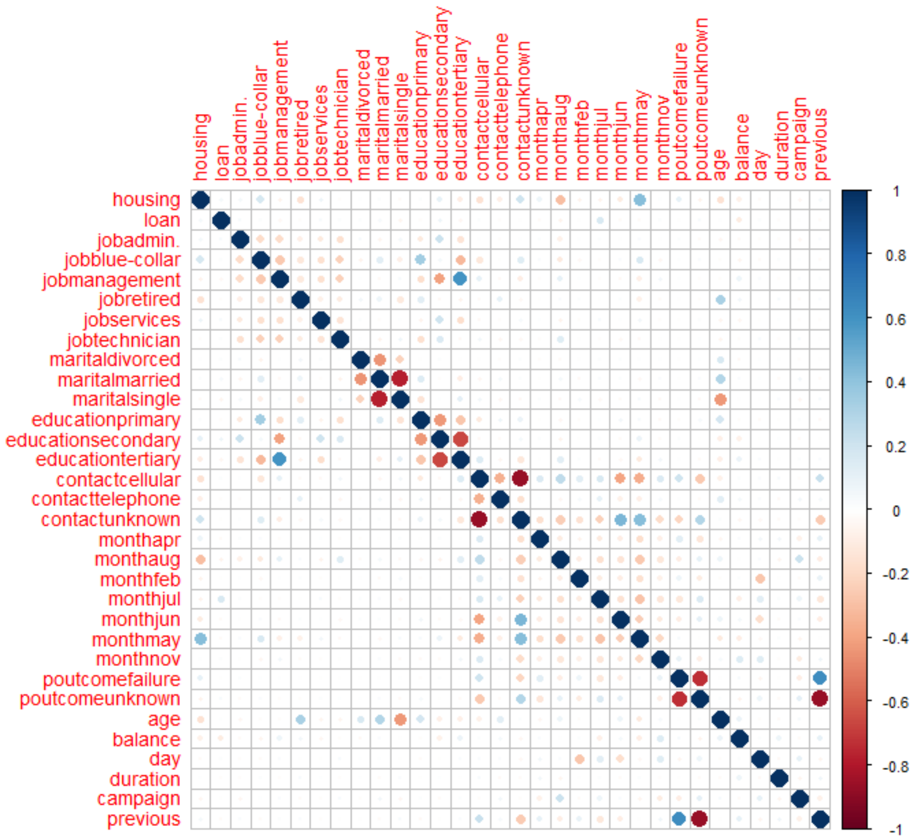


Figure 1. Correlation Plot of 32 predictors

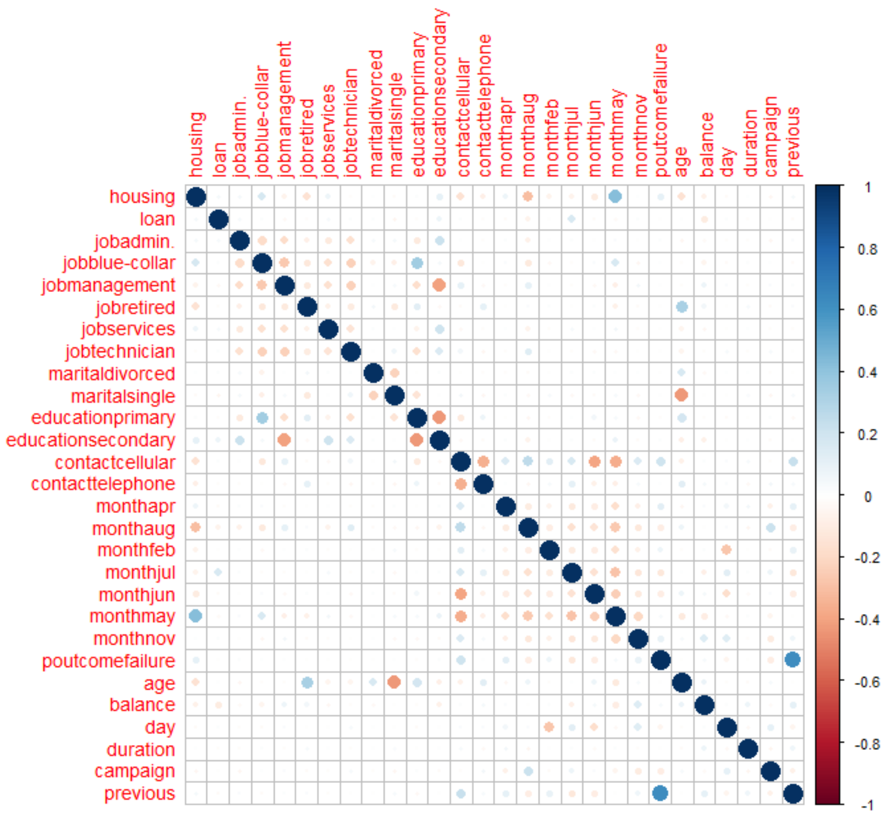


Figure 2. Correlation plot after removing highly correlated predictors.

Transformations

Before going to the model building, we need to check the distributions of continuous predictors to make sure the distributions have no outliers and are normal.

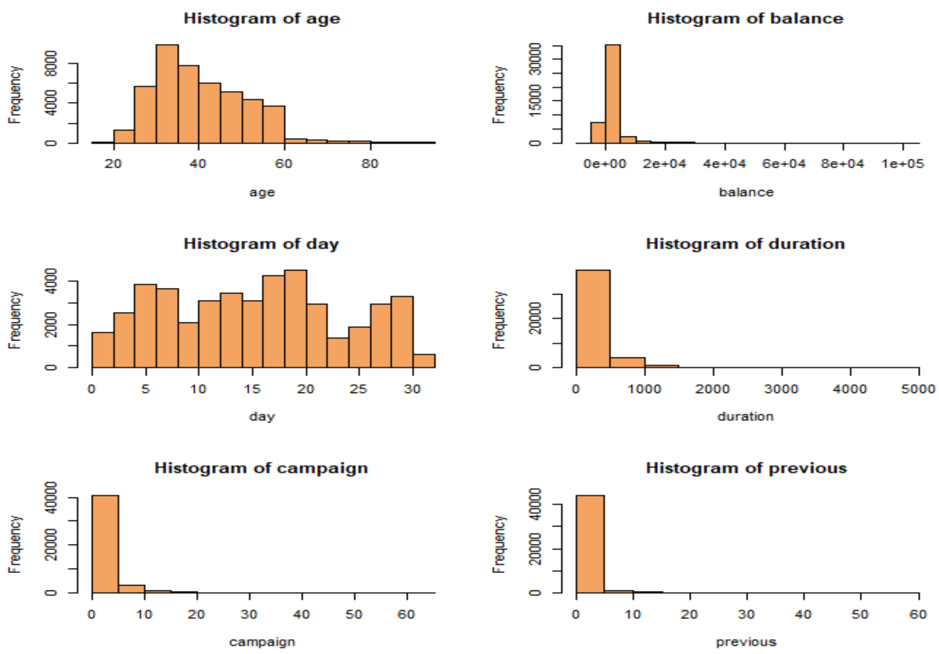


Figure 3. Distribution (Histograms) of continuous predictors.

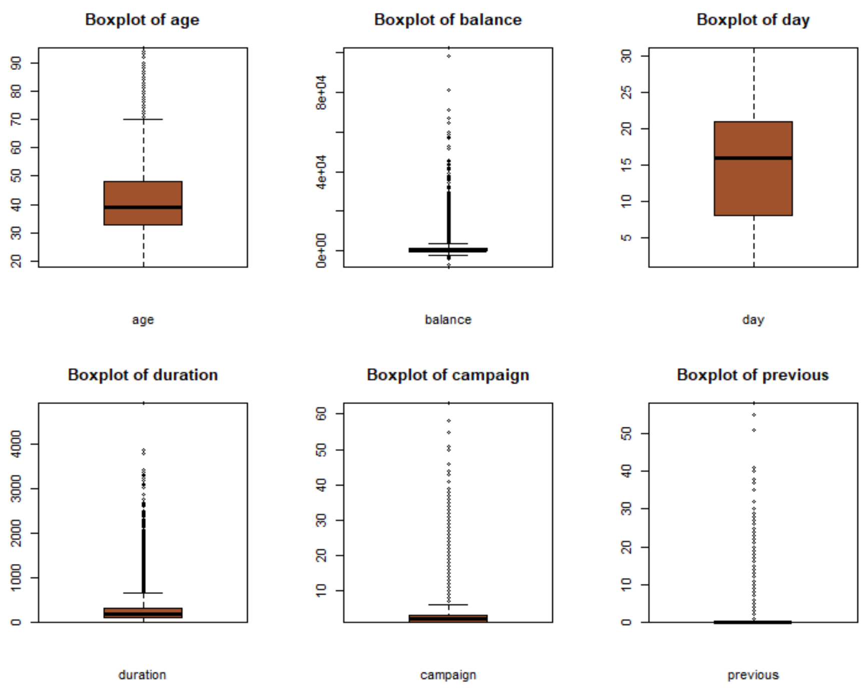


Figure 4. Distribution (Boxplots) of continuous predictors.

Predictor	Skewness	Interpretation
Age	0.68475925	Moderately Right Skewed
Balance	8.35966378	Highly Right Skewed
Day	0.09307122	Approximately Symmetric
Duration	3.14411010	Highly Right Skewed
Campaign	4.89826479	Highly Right Skewed
Previous	7.82957702	Highly Right Skewed

Table 2. Skewness of Predictors.

From the above distributions, we can see that almost all continuous predictors are not normal, and they have outliers in them. We can also confirm this from the skewness values. To correct the skewness of the continuous predictors and make them normalize we performed Box-Cox transformation. We used Spatial Sign transformation to address the outliers in the data. We can the distributions and skewness below after the transformation.

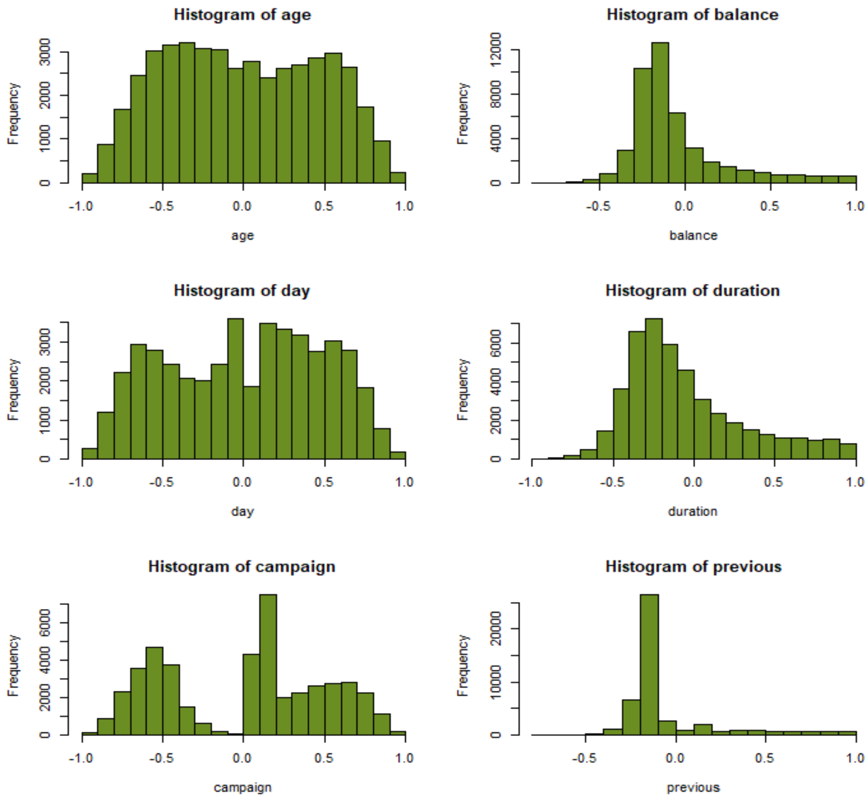


Figure 5. Distribution (Histograms) after the transformation.

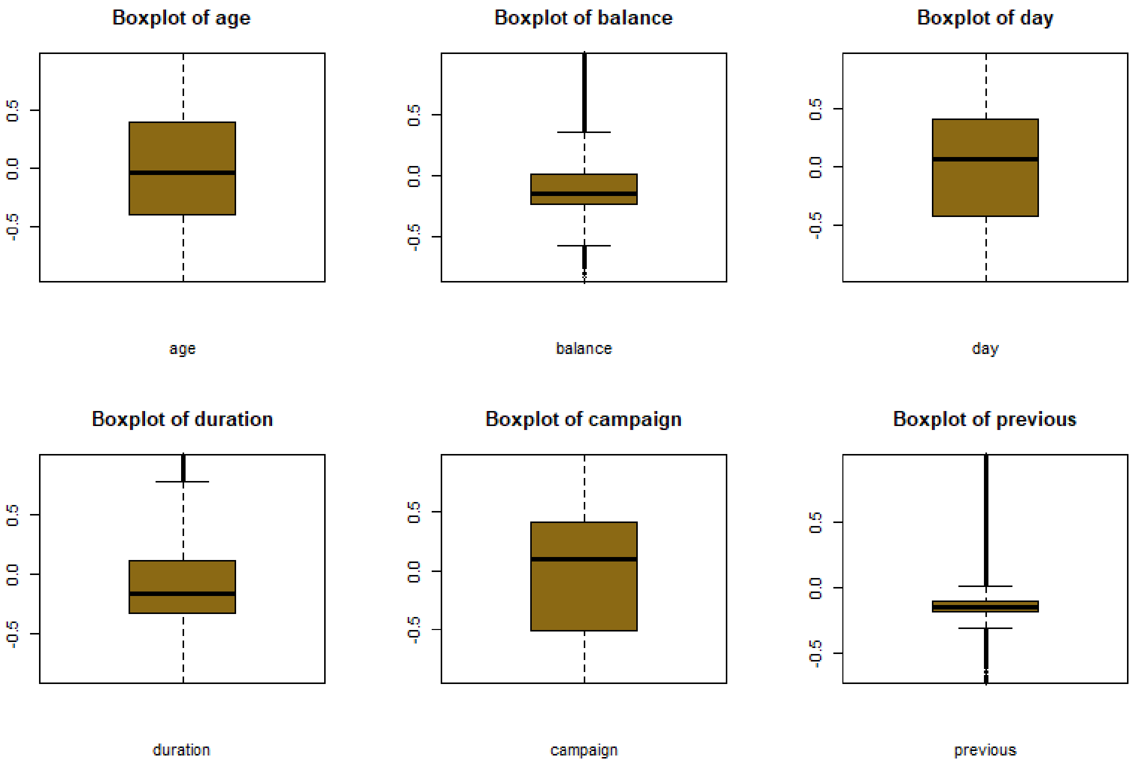


Figure 6. Distribution (Boxplots) after the transformation.

Predictor	Skewness after Box Cox	Skewness after Spatial Sign
Age	0.09889626	0.06354351
Balance	8.35966378	1.62330154
Day	-0.17839430	-0.12294714
Duration	3.14411010	0.96570825
Campaign	0.14909605	-0.07707232
Previous	7.82957702	2.20938085

Table 3. Skewness after Transformations.

Data Splitting:

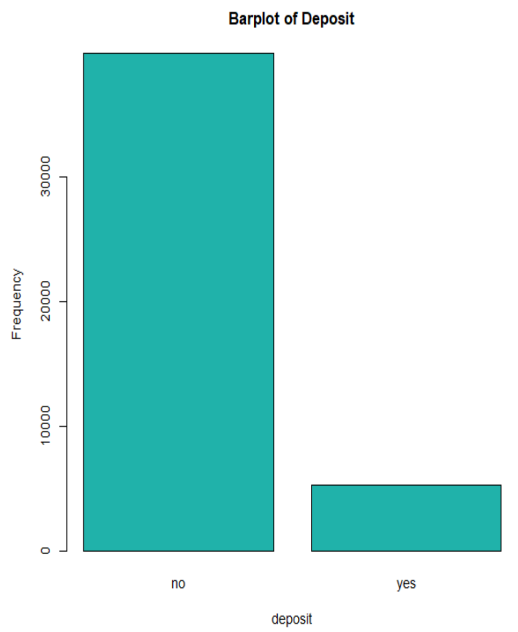


Figure 7. Distribution of Target Variable.

As our target variable is imbalanced, we are using stratified random sampling and going to split the data into 70% train and 30% test data. To check the models effectively, we are using 10-fold cross-validation. After preprocessing we are using 28 predictors for model building and a total number of samples is 45210. Since our data is imbalanced, we are using "Kappa" as a classification statistic.

Model Building:

Models

We're using a variety of models to create our predictive models, covering both linear and non-linear models. In the linear models, we're using logistic regression, linear discriminant analysis (LDA), and partial least squares discriminant analysis (PLSDA), along with penalized models. From non-linear, we have models like nonlinear discriminant analysis, neural networks, flexible discriminant analysis, support vector machines, K-nearest neighbors, and Naive Bayes. These different models give us a range of ways to identify and understand the patterns in our data, contributing to a thorough and strong approach to building our models.

Model Summary

We used training set to tune all the models and below table summarizes the models' performance. All the summary statistics and tuning parameters of these models are in the appendix.

Linear Models	Best Tuning Parameter	Accuracy	Kappa
Logistic regression	NA	0.8923469	0.3370045
LDA	NA	0.8902299	0.3745043
PLSDA	ncomp = 4	0.8862172	0.08386809
Penalized Models	alpha = 0.1 and lambda = 0.01.	0.8912409	0.272778651
Non-linear Models	**Best Tuning Parameter**	**Accuracy**	**Kappa**
NDLA	subclasses = 1	0.8902299	0.3745043
Neural Networks	size = 9 and decay = 1	0.9012575	0.43556644
FDA	degree = 3 and nprune = 14	0.8921263	0.4427430
SVM	sigma = 0.01266439 and C = 0.25	0.8969916	0.4148633
KNN	K = 1	0.8644146	0.3234554
Naïve Bayes	NA	0.8734198	0.357957

Table 4. Model's Training Summary

From the table, we can see that the Kappa value is high for the Linear Discriminant Analysis model from linear models and the Flexible Discriminant Analysis from non-linear models. So, the top two models are Linear Discriminant Analysis and the Flexible Discriminant Analysis.

Top two Models

The two best-performing models were then applied to make predictions on the test dataset, and the resulting accuracy and kappa values are provided below.

Top two Models	Test Set	
	Accuracy	Kappa
Linear Discriminant Analysis	0.889	0.3562
Flexible Discriminant Analysis	0.8949	0.4444

Table 5. Summary of Top two models.

Conclusion

Best Model

Among the top two models, when we compare the test set Kappa value of top models, we can see that the Kappa value is high for the **Flexible Discriminant Analysis** model. So, we recommend **Flexible Discriminant Analysis** as the best model for this data.

Model Performance

Confusion Matrix

```
Confusion Matrix and Statistics

          Reference
Prediction    no   yes
       no  11416   866
       yes   560   720

               Accuracy : 0.8949
                 95% CI : (0.8896, 0.9)
    No Information Rate : 0.8831
    P-Value [Acc > NIR] : 7.682e-06

                  Kappa : 0.4444

 Mcnemar's Test P-Value : 6.648e-16

            Sensitivity : 0.9532
            Specificity : 0.4540
         Pos Pred Value : 0.9295
         Neg Pred Value : 0.5625
             Prevalence : 0.8831
         Detection Rate : 0.8418
   Detection Prevalence : 0.9056
      Balanced Accuracy : 0.7036

       'Positive' Class : no
```

Important Predictors

Important Predictors	Importance
Duration	100.000
Previous	70.334
Poutcomefailure	52.660
housing	43.794
day	22.297
monthapr	22.297
monthfeb	22.297

monthmay	11.680
contactcellular	6.576
campaign	0.000

Table 6. FDA variable importance

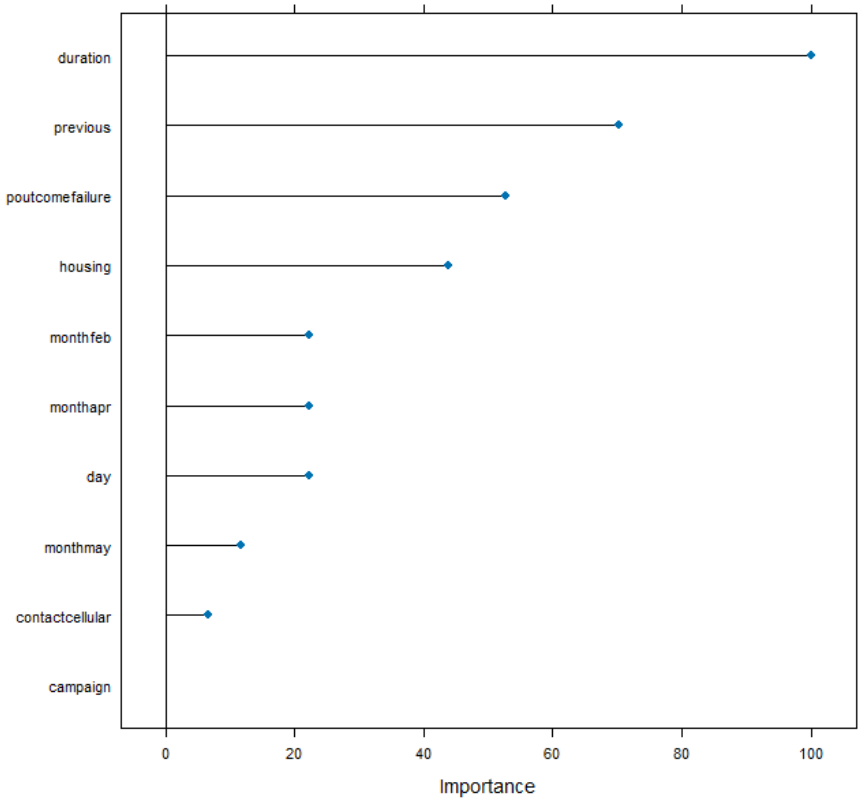


Figure 8. Important Predictors

References:

Moro,S., Rita,P., and Cortez,P.. (2012). Bank Marketing. UCI Machine Learning Repository. https://doi.org/10.24432/C5K306.

Appendix: Summary Statistics & Tuning Plot of Models on Train Data

Linear Models

Logistic Regression

```
Generalized Linear Model

31648 samples
   28 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (28), scaled (28)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 28484, 28482, 28483, 28484, 28484, 28482, ...
Resampling results:

  Accuracy   Kappa
     0.8923469  0.3370045
```

Linear discriminant analysis (LDA)

```
Linear Discriminant Analysis

31648 samples
   28 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (28), scaled (28)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 28484, 28482, 28483, 28484, 28484, 28482, ...
Resampling results:

  Accuracy   Kappa
  0.8902299  0.3745043
```

Partial least squares discriminant analysis (PLSDA)

```
Partial Least Squares

31648 samples
   28 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (28), scaled (28)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 28484, 28482, 28483, 28484, 28484, 28482, ...
Resampling results across tuning parameters:

  ncomp  Accuracy   Kappa
  1      0.8834998  0.01044861
  2      0.8852376  0.05410159
  3      0.8861855  0.07513384
  4      0.8862172  0.08386809

Kappa was used to select the optimal model using the largest value.
The final value used for the model was ncomp = 4.
```

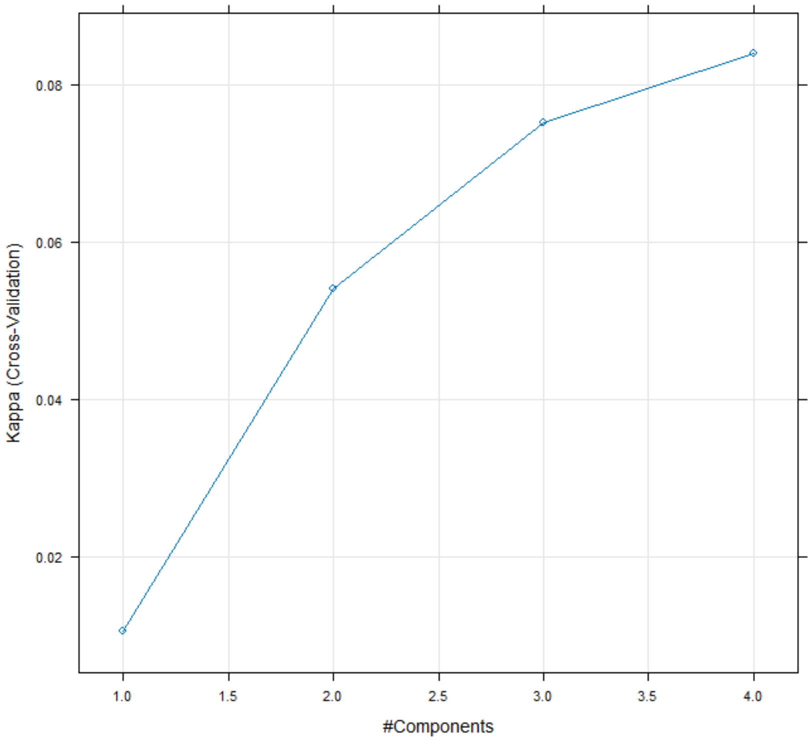


Figure 9. Tuning Plot of PLSDA

Penalized models

```
glmnet

31648 samples
   28 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (28), scaled (28)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 28484, 28482, 28483, 28484, 28484, 28482, ...
Resampling results across tuning parameters:

  alpha  lambda      Accuracy   Kappa
  0.0    0.01000000  0.8913674  0.267737513
  0.0    0.03111111  0.8894400  0.175524428
  0.0    0.05222222  0.8872915  0.102626521
  0.0    0.07333333  0.8848585  0.042977716
  0.0    0.09444444  0.8839737  0.017879482
  0.0    0.11555556  0.8832470  0.004624647
  0.0    0.13666667  0.8829942  0.000000000
  0.0    0.15777778  0.8829942  0.000000000
  0.0    0.17888889  0.8829942  0.000000000
  0.0    0.20000000  0.8829942  0.000000000
  0.1    0.01000000  0.8912409  0.272778651
  0.1    0.03111111  0.8888712  0.154447621
  0.1    0.05222222  0.8855536  0.061178569
  0.1    0.07333333  0.8834998  0.010862954
  0.1    0.09444444  0.8829942  0.000000000
  0.1    0.11555556  0.8829942  0.000000000
  0.1    0.13666667  0.8829942  0.000000000
  0.1    0.15777778  0.8829942  0.000000000
  0.1    0.17888889  0.8829942  0.000000000
  0.1    0.20000000  0.8829942  0.000000000
  0.2    0.01000000  0.8911146  0.267154314
  0.2    0.03111111  0.8881130  0.135281927
  0.2    0.05222222  0.8841949  0.033011398
```

0.2	0.07333333	0.8829942	0.000000000
0.2	0.09444444	0.8829942	0.000000000
0.2	0.11555556	0.8829942	0.000000000
0.2	0.13666667	0.8829942	0.000000000
0.2	0.15777778	0.8829942	0.000000000
0.2	0.17888889	0.8829942	0.000000000
0.2	0.20000000	0.8829942	0.000000000
0.4	0.01000000	0.8905143	0.251828682
0.4	0.03111111	0.8865964	0.098816901
0.4	0.05222222	0.8831522	0.003201828
0.4	0.07333333	0.8829942	0.000000000
0.4	0.09444444	0.8829942	0.000000000
0.4	0.11555556	0.8829942	0.000000000
0.4	0.13666667	0.8829942	0.000000000
0.4	0.15777778	0.8829942	0.000000000
0.4	0.17888889	0.8829942	0.000000000
0.4	0.20000000	0.8829942	0.000000000
0.6	0.01000000	0.8904511	0.240200794
0.6	0.03111111	0.8853008	0.069825558
0.6	0.05222222	0.8829942	0.000000000
0.6	0.07333333	0.8829942	0.000000000
0.6	0.09444444	0.8829942	0.000000000
0.6	0.11555556	0.8829942	0.000000000
0.6	0.13666667	0.8829942	0.000000000
0.6	0.15777778	0.8829942	0.000000000
0.6	0.17888889	0.8829942	0.000000000
0.6	0.20000000	0.8829942	0.000000000
0.8	0.01000000	0.8900720	0.226578319
0.8	0.03111111	0.8841000	0.040129809
0.8	0.05222222	0.8829942	0.000000000
0.8	0.07333333	0.8829942	0.000000000
0.8	0.09444444	0.8829942	0.000000000
0.8	0.11555556	0.8829942	0.000000000
0.8	0.13666667	0.8829942	0.000000000
0.8	0.15777778	0.8829942	0.000000000
0.8	0.17888889	0.8829942	0.000000000
0.8	0.20000000	0.8829942	0.000000000
1.0	0.01000000	0.8895350	0.214993642
1.0	0.03111111	0.8832470	0.011925210
1.0	0.05222222	0.8829942	0.000000000
1.0	0.07333333	0.8829942	0.000000000
1.0	0.09444444	0.8829942	0.000000000
1.0	0.11555556	0.8829942	0.000000000
1.0	0.13666667	0.8829942	0.000000000
1.0	0.15777778	0.8829942	0.000000000
1.0	0.17888889	0.8829942	0.000000000
1.0	0.20000000	0.8829942	0.000000000

Kappa was used to select the optimal model using the largest value.
The final values used for the model were alpha = 0.1 and lambda = 0.01.

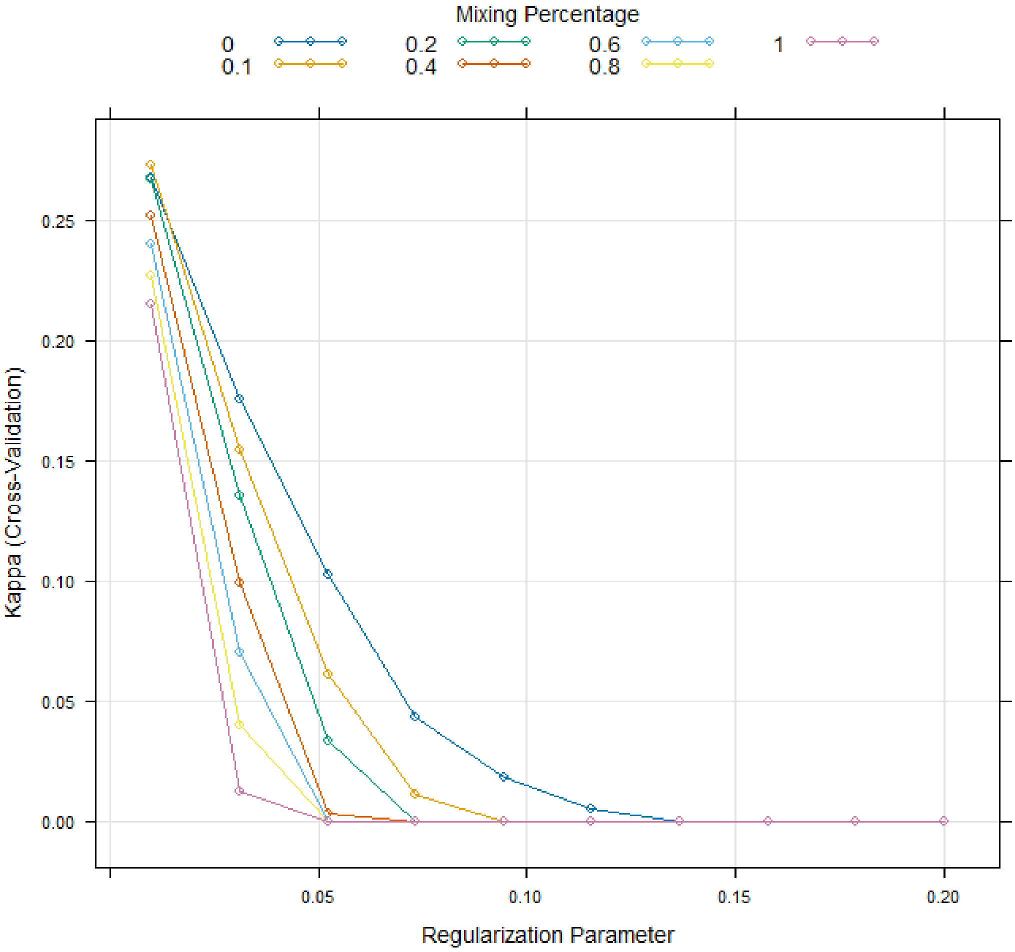


Figure 10. Tuning Plot of Penalized Model

Non-linear Models

Nonlinear Discriminant Analysis

```
Mixture Discriminant Analysis

31648 samples
   28 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (28), scaled (28)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 28484, 28482, 28483, 28484, 28484, 28482, ...
Resampling results across tuning parameters:

  subclasses  Accuracy   Kappa
   1          0.8902299  0.3745043
   2          0.8682687  0.2083588
   3          0.8527572  0.2021066
   4          0.8504491  0.2643869
   5          0.8512074  0.2649004
   6          0.8553456  0.2646369
   7          0.8593908  0.2912579
   8          0.8527531  0.2957629
   9          0.8520949  0.3280412
  10          0.8546818  0.3224095

Kappa was used to select the optimal model using the largest value.
The final value used for the model was subclasses = 1.
```

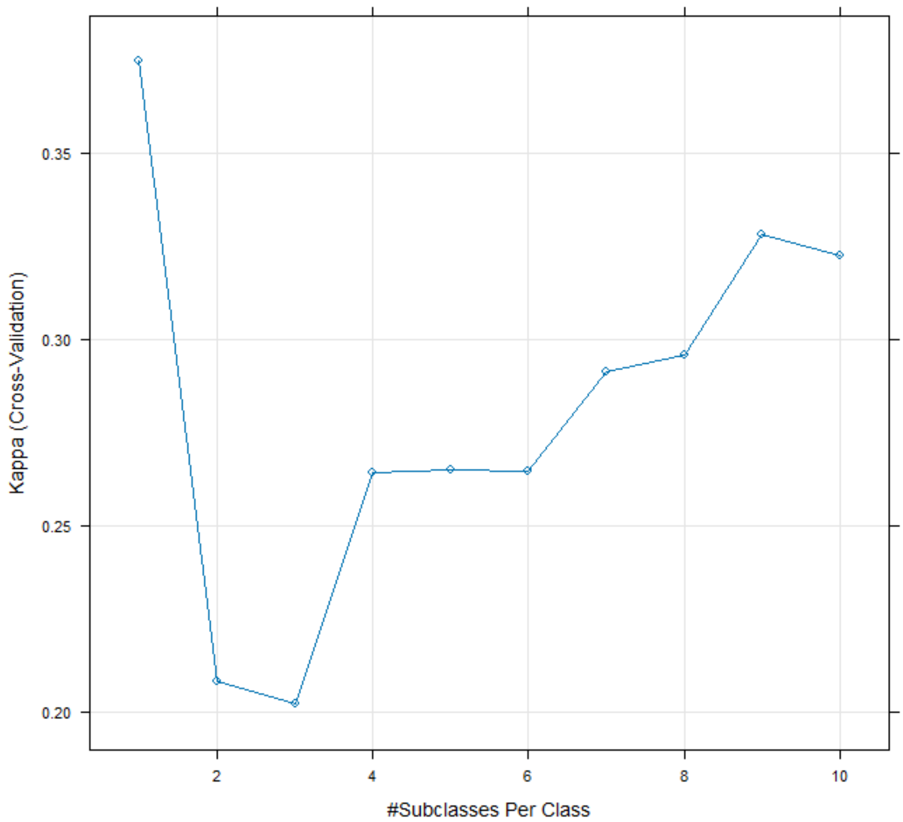


Figure 11. Tuning Plot of NLDA

Neural Networks

```
Neural Network

31648 samples
   28 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (28), scaled (28)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 28484, 28482, 28483, 28484, 28484, 28482, ...
Resampling results across tuning parameters:

  size  decay  Accuracy   Kappa
   1    0.0    0.8834681  0.03021165
   1    0.1    0.8900086  0.25494000
   1    1.0    0.8913358  0.32367146
   1    2.0    0.8908302  0.31261132
   2    0.0    0.8880814  0.19369716
   2    0.1    0.8921566  0.29951003
   2    1.0    0.8952222  0.38204436
   2    2.0    0.8961069  0.38553089
   3    0.0    0.8897871  0.23659448
   3    0.1    0.8938638  0.35639238
   3    1.0    0.8949693  0.38250364
   3    2.0    0.8953171  0.37121766
   4    0.0    0.8943688  0.35288053
   4    0.1    0.8964865  0.36858676
   4    1.0    0.8982241  0.40316114
   4    2.0    0.8987609  0.40430453
   5    0.0    0.8960754  0.40604025
```

```
   5    0.1    0.8981292  0.40692228
   5    1.0    0.8989508  0.41041031
   5    2.0    0.8992350  0.40320778
   6    0.0    0.8968018  0.41762122
   6    0.1    0.8983187  0.42499213
   6    1.0    0.8992980  0.41021093
   6    2.0    0.8984769  0.40404233
   7    0.0    0.8981287  0.43183291
   7    0.1    0.9004039  0.43388006
   7    1.0    0.8999302  0.41915417
   7    2.0    0.9005937  0.41645850
   8    0.0    0.8962964  0.42120819
   8    0.1    0.8979713  0.41946017
   8    1.0    0.9004676  0.42742913
   8    2.0    0.9004674  0.41603447
   9    0.0    0.8973710  0.42753499
   9    0.1    0.8992978  0.43457197
   9    1.0    0.9012575  0.43556644
   9    2.0    0.8998986  0.41911086
  10    0.0    0.8957910  0.41794018
  10    0.1    0.8987610  0.43199499
  10    1.0    0.9000883  0.42833853
  10    2.0    0.9010679  0.42903684

Kappa was used to select the optimal model using the largest value.
The final values used for the model were size = 9 and decay = 1.
```

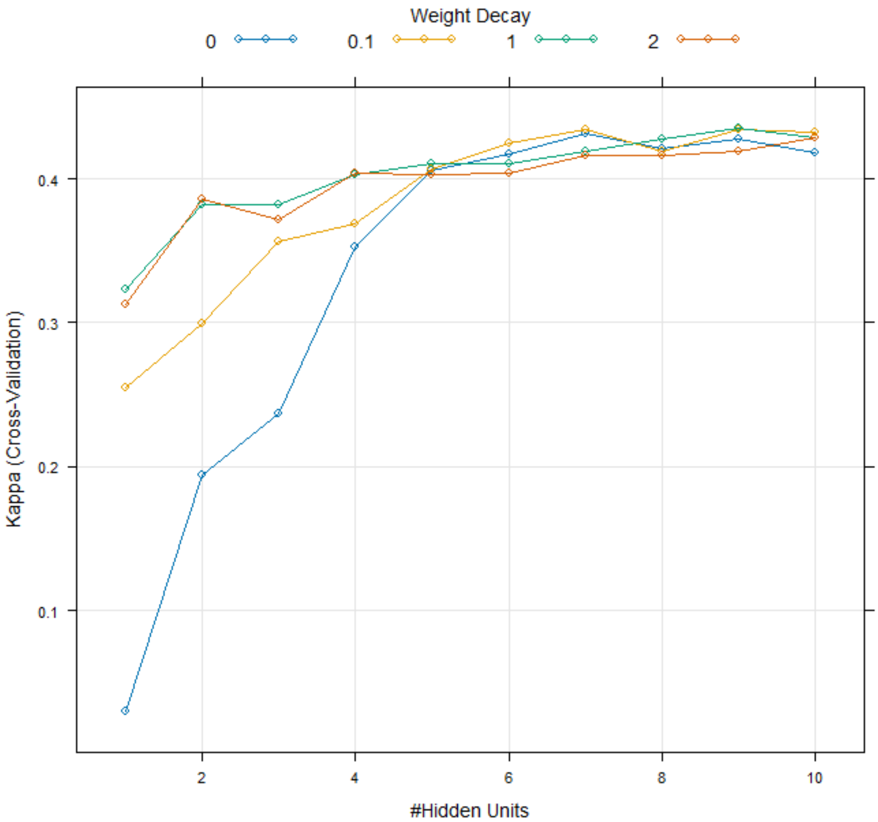


Figure 12. Tuning Plot of Neural Networks

Flexible Discriminant Analysis

```
Flexible Discriminant Analysis

31648 samples
   28 predictor
```

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

Pre-processing: centered (28), scaled (28)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 28483, 28484, 28484, 28484, 28484, 28483, ...
Resampling results across tuning parameters:

  degree  nprune  Accuracy   Kappa
  1        2      0.8751906  0.3006356
  1        3      0.8790456  0.3192757
  1        4      0.8807834  0.3227694
  1        5      0.8886508  0.3988148
  1        6      0.8911154  0.4177867
  1        7      0.8913365  0.4200995
  1        8      0.8916209  0.4211233
  1        9      0.8909255  0.4202135
  1       10      0.8919364  0.4255687
  1       11      0.8914310  0.4262615
  1       12      0.8906095  0.4237267
  1       13      0.8920313  0.4305563
  1       14      0.8913360  0.4268729
  1       15      0.8909886  0.4241814
  2        2      0.8829942  0.0000000
  2        3      0.8821414  0.1502207
  2        4      0.8831847  0.3102156
  2        5      0.8877346  0.3756584
  2        6      0.8886822  0.3809964
  2        7      0.8895672  0.4014635
  2        8      0.8896620  0.4076981
  2        9      0.8908626  0.4186063
  2       10      0.8904204  0.4179337
  2       11      0.8917788  0.4261875
  2       12      0.8922528  0.4274857
  2       13      0.8914629  0.4255542
  2       14      0.8926637  0.4358865
  2       15      0.8924109  0.4355146
  3        2      0.8829942  0.0000000
  3        3      0.8821414  0.1502207
  3        4      0.8831847  0.3102156
  3        5      0.8877346  0.3756584
  3        6      0.8892826  0.3863278
  3        7      0.8895672  0.4014635
  3        8      0.8896620  0.4076981
  3        9      0.8906414  0.4194266
  3       10      0.8903570  0.4217880
  3       11      0.8917157  0.4294689
  3       12      0.8922212  0.4363939
  3       13      0.8922212  0.4413404
  3       14      0.8921263  0.4427430
  3       15      0.8915262  0.4410463

Kappa was used to select the optimal model using the largest value.
The final values used for the model were degree = 3 and nprune = 14.
```

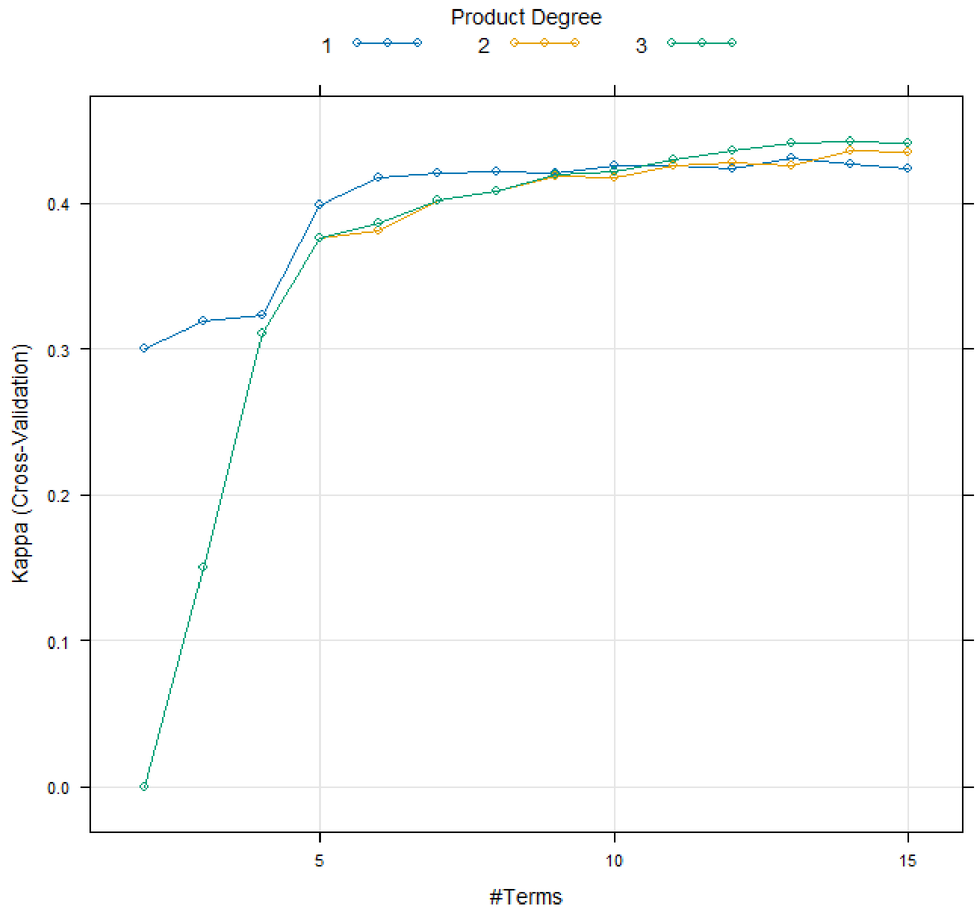


Figure 13. Tuning Plot of FDA

Support Vector Machines

```
Support Vector Machines with Radial Basis Function Kernel

31648 samples
   28 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (28), scaled (28)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 28484, 28482, 28483, 28484, 28484, 28482, ...
Resampling results across tuning parameters:

  C      Accuracy   Kappa
   0.25  0.8969916  0.4148633
   1.00  0.8971496  0.3828091
   4.00  0.8982551  0.3784174
  16.00  0.8959169  0.3581372

Tuning parameter 'sigma' was held constant at a value of 0.01266439
Kappa was used to select the optimal model using the largest value.
The final values used for the model were sigma = 0.01266439 and C = 0.25.
```

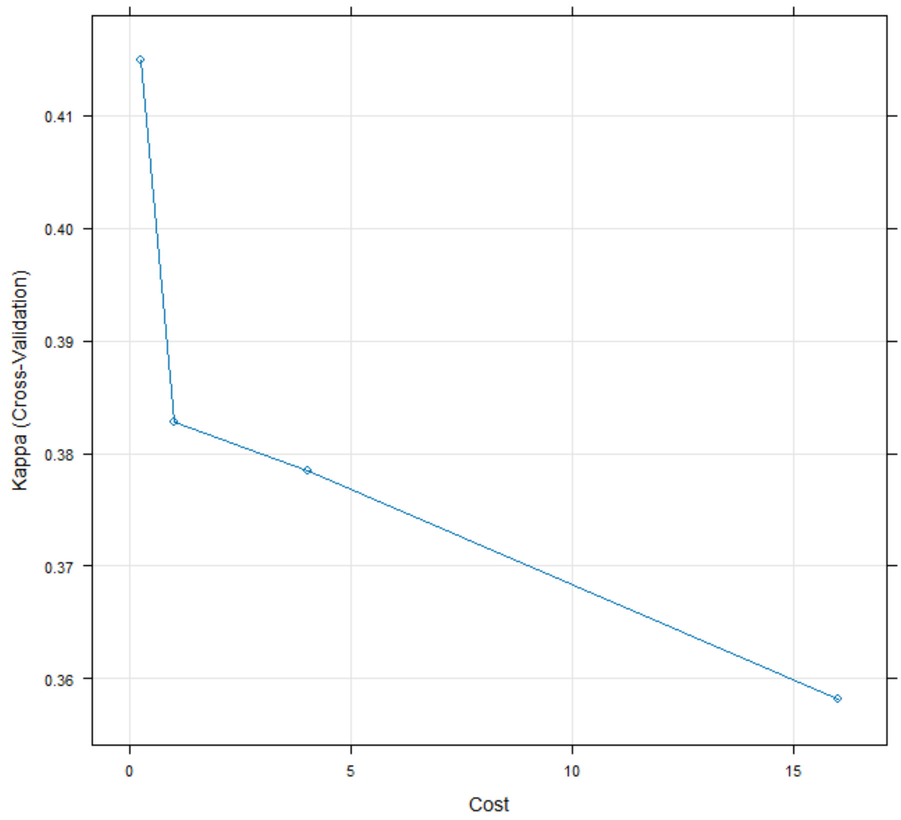


Figure 14. Tuning Plot of SVM

K-Nearest Neighbors

```
k-Nearest Neighbors

31648 samples
   28 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (28), scaled (28)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 28484, 28482, 28483, 28484, 28484, 28482, ...
Resampling results across tuning parameters:

  k   Accuracy   Kappa
   1  0.8644146  0.3234554
   2  0.8624237  0.3106057
   3  0.8784436  0.3117898
   4  0.8801187  0.3185401
   5  0.8846367  0.3009827
   6  0.8831832  0.2883998
   7  0.8860587  0.2793152
   8  0.8852689  0.2694741
   9  0.8871647  0.2628467
  10  0.8869750  0.2578917
  11  0.8881441  0.2530882
  12  0.8878913  0.2487129
  13  0.8886183  0.2389654
  14  0.8878599  0.2296640
  15  0.8880812  0.2151571
  16  0.8885235  0.2177760
  17  0.8888711  0.2112513
```

```
18   0.8883971   0.2083038
19   0.8887763   0.2016805
20   0.8895346   0.2045268
21   0.8888079   0.1908663
22   0.8885237   0.1866699
23   0.8890292   0.1854783
24   0.8884288   0.1790696
25   0.8886500   0.1768216
26   0.8884288   0.1741531
27   0.8887447   0.1712817
28   0.8881443   0.1657661
29   0.8883656   0.1616130
30   0.8883338   0.1611973
31   0.8883971   0.1569719
32   0.8886183   0.1604321
33   0.8885234   0.1561300
34   0.8883339   0.1519071
35   0.8882392   0.1472698
36   0.8877021   0.1408475
37   0.8875125   0.1378084
38   0.8872597   0.1352483
39   0.8871965   0.1285971
40   0.8870384   0.1255979
41   0.8868804   0.1225337
42   0.8870384   0.1242086
43   0.8873544   0.1237608
44   0.8874808   0.1227237
45   0.8871648   0.1182114
46   0.8869753   0.1168087
47   0.8867858   0.1116095
48   0.8865329   0.1092905
49   0.8867225   0.1091314
50   0.8865329   0.1052568

Kappa was used to select the optimal model using the largest value.
The final value used for the model was k = 1.
```

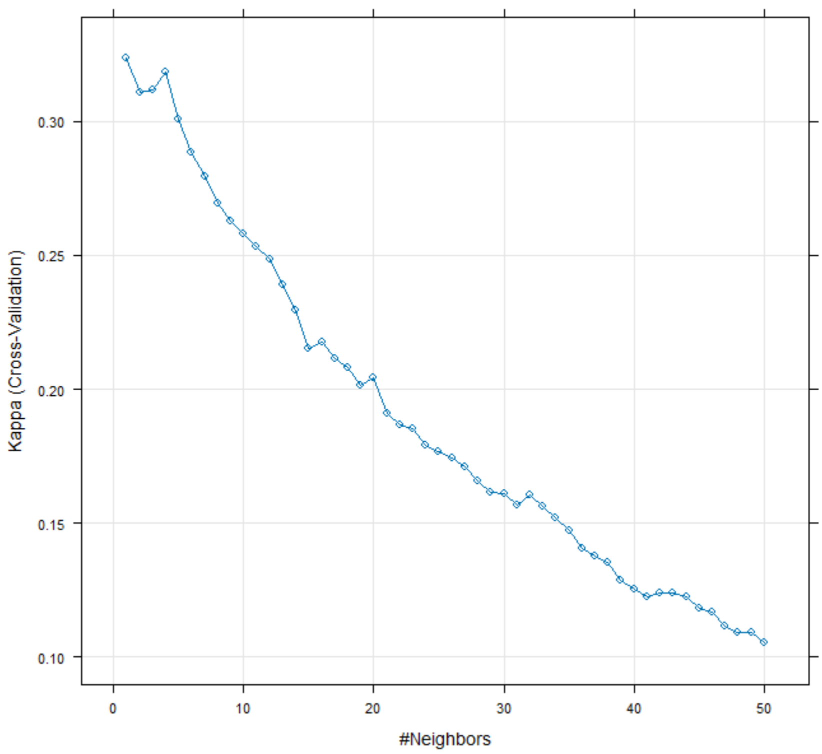


Figure 15. Tuning Plot of KNN

Naive Bayes

```
Naive Bayes

31648 samples
    28 predictor
    2 classes: 'no', 'yes'

Pre-processing: centered (28), scaled (28)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 28484, 28482, 28483, 28484, 28484, 28482, ...
Resampling results:

  Accuracy   Kappa
  0.8734198  0.357957

Tuning parameter 'fL' was held constant at a value of 2
Tuning parameter 'usekernel' was held constant at a value of TRUE
Tuning parameter 'adjust' was
 held constant at a value of TRUE
```

R Code:

```
library(mlbench)

library(e1071)

library(caret)

library(dplyr)

library(corrplot)

library(MASS)


dF = read.csv("C:/Users/91938/Desktop/Fall 2023/PM/Project/bank-full.csv", sep = ';')

dim(dF)

head(dF,5)

colnames(dF)[colnames(dF) == "y"] <- "deposit"

table(dF$deposit)

colSums(is.na(dF))

summary(dF)

######removing oulier###############

max_previous <- max(dF$previous)

dF <- dF[dF$previous != max_previous, ]

##########missing values############

image(is.na(dF), main = "Missing Values", xlab = "Observation", ylab = "Variable",

   xaxt = "n", yaxt = "n", bty = "n")

axis(1, seq(0, 1, length.out = nrow(dF)), 1:nrow(dF), col = "white")

################################


lapply(dF, unique)

continuous_Vars <- names(dF)[sapply(dF, is.numeric)]

categorical_Vars <- colnames(dF[, !names(dF)%in%continuous_Vars])

#barplots

par(mfrow=c(3,3))

for (i in categorical_Vars) {

  data <- table(dF[[i]])
```

```
  barplot(data, main = paste('Barplot of', i), xlab = i, col = 'powderblue')

}

par(mfrow=c(1,1))

barplot(table(dF[, 'deposit']), main = 'Barplot of Deposit', xlab = 'deposit', col = 'lightseagreen', ylab =
'Frequency')

#########binary#########

binary_columns <- c("default", "housing", "loan")

head(df)

# Use lapply to apply ifelse to each binary column

dF[,binary_columns] <- lapply(dF[,binary_columns], function(x) ifelse(x == "yes", 1, 0))

#dummy variables

dummy_vars <- dummyVars("~job + marital + education + contact + month + poutcome", data = dF)

df_with_dummies <- predict(dummy_vars, newdata = dF)

dummy <- c("job", "marital", "education", "contact", "month", "poutcome")

df <- cbind(dF[, !names(dF)%in%dummy], df_with_dummies)

df <- df[,-which(names(df) == 'deposit')]

dim(df)

nearZeroVar(df, names = T)

columns_to_remove <- c("default","pdays", "jobentrepreneur", "jobhousemaid", "jobself-
employed", "jobstudent",

  "jobunemployed", "jobunknown", "educationunknown",

  "monthdec", "monthjan", "monthmar", "monthoct", "monthsep", "poutcomeother",

  "poutcomesuccess"

)

# Remove the specified columns from the data frame

df <- df[, !names(df) %in% columns_to_remove]

dim(df)

continuous_vars <- c("age", "balance","day","duration", "campaign", "previous")

categorical_vars <- colnames(df[, !names(df)%in%continuous_vars])



#barplots
```

```
par(mfrow=c(4,4))
for (i in categorical_vars) {
  data <- table(df[[i]])
  barplot(data, main = paste('Barplot of', i), xlab = i, col = 'powderblue')
}
par(mfrow=c(1,1))

nearZeroVar(df, names=T)
############################
#histograms
par(mfrow=c(3,2))
for (i in continuous_vars) {
  data <- df[[i]]
  hist(data, main = paste('Histogram of', i), xlab = i, col = 'sandybrown')
}
par(mfrow=c(1,1))
skewness_values <- apply(df[continuous_vars], 2, skewness)
skewness_values
#boxplots
par(mfrow=c(2,3))
for (i in continuous_vars) {
  data <- df[[i]]
  boxplot(data, main = paste('Boxplot of', i), xlab = i, col = 'sienna')
}
par(mfrow=c(1,1))

###########BOXCOX#############
par(mfrow = c(4, 2))
for (i in continuous_vars){
  hist(df[[i]], main = paste("Before Transformation ",i) , xlab = i, horiz = TRUE)
  bct_ri = BoxCoxTrans(df[[i]])
```

```
  trans <- predict(bct_ri, df[[i]])
  hist(trans, main = paste("After Transformation ",i) , xlab = i, horiz = TRUE)
}
par(mfrow = c(1,1))


columns_to_transform <- df[,continuous_vars]
transDf <- preProcess(columns_to_transform, method = c("BoxCox","center", "scale"))
transDf
#apply transformation
transformed <- predict(transDf, columns_to_transform)
transformed
sapply(transformed,skewness)


########histograms after trans##########
par(mfrow=c(3,2))
for (i in colnames(transformed)) {
  hist(transformed[[i]], main = paste('Histogram of', i), xlab = i, col = 'olivedrab')
}
par(mfrow=c(1,1))
#########SS##########
transSS <- spatialSign(transformed[, continuous_vars])
Transform_spa <- as.data.frame(transSS)
sapply(Transform_spa, skewness)
########boxplots after SS#########
par(mfrow=c(2,3))
for (i in continuous_vars) {
  data <- Transform_spa[[i]]
  boxplot(data, main = paste('Boxplot of', i), xlab = i, col = 'goldenrod4')
}
```

```
par(mfrow=c(1,1))

########################

par(mfrow=c(3,2))

for (i in colnames(Transform_spa)) {

  hist(Transform_spa[[i]], main = paste('Histogram of', i), xlab = i, col = 'olivedrab')

}

par(mfrow=c(1,1))

##########correlation############

conPred <- df[, continuous_vars]

hist(conPred$previous)

correlations <- cor(df[, continuous_vars])

corrplot(correlations, method = 'color')

correlations1 <- cor(Transform_spa[, continuous_vars])

corrplot(correlations1, method = 'color')

##########

correlations1 <- cor(dF[, continuous_Vars])

corrplot(correlations1, method = 'color')


##### Final DF #####

final_df <- cbind(df[, categorical_vars],Transform_spa)

dim(final_df)

correlations_1 <- cor(final_df)

corrplot(correlations_1)

highCorr <- findCorrelation(abs(correlations_1), cutoff = .65, names = T)

length(highCorr)

highCorr

final_df[,c(17, 14)]

final <- final_df[, -highCorr]

corrplot(cor(final))

dim(final)

######## Model Building ########
```

```
dF$deposit <- as.factor(dF$deposit)
set.seed(5790)
index <- createDataPartition(dF$deposit, p = 0.7, list = FALSE)
x_train <- final[index,]
x_test <- final[-index,]
y_train <- dF$deposit[index]
y_test <- dF$deposit[-index]
dim(x_test)
length(y_test)
ctrl <- trainControl(method = "cv",
            number= 10,
            summaryFunction = defaultSummary,
            classProbs = TRUE,
            savePredictions = TRUE)

set.seed(5790)
logistic <- train(x_train,
          y = y_train,
          preProcess = c('center', 'scale'),
          method = "glm",
          metric = "Kappa",
          trControl = ctrl)
logistic
plot(logistic)
pred_logistic <- predict(logistic, x_test)
confusionMatrix(data = pred_logistic,
          reference = y_test)

## LDA
set.seed(5790)
lda <- train(x_train,
```

```
            y = y_train,
            method = "lda",
            preProcess = c('center', 'scale'),
            metric = "Kappa",
            trControl = ctrl)
lda
plot(lda)
pred_lda <- predict(lda, x_test)
confusionMatrix(data = pred_lda, y_test)

## PLSDA
set.seed(5790)
plsda <- train(x = x_train,
             y = y_train,
             method = "pls",
             tuneGrid = expand.grid(.ncomp = 1:4),
             preProcess = c("center","scale"),
             metric = "Kappa",
             trControl = ctrl)

plsda

plot(plsda)
pred_plsda <- predict(plsda, x_test)
confusionMatrix(data = pred_plsda, y_test)

## PM
set.seed(5790)
glmnGrid <- expand.grid(.alpha = c(0, .1, .2, .4, .6, .8, 1),
                  .lambda = seq(.01, .2, length = 10))
glmn <- train(x=x_train,
```

```
            y = y_train,
            method = "glmnet",
            tuneGrid = glmnGrid,
            preProc = c("center", "scale"),
            metric = "Kappa",
            trControl = ctrl)
glmn
plot(glmn)
pred_glmn <- predict(glmn, x_test)
confusionMatrix(pred_glmn, y_test)
```

```
####### Non linear discriminant analysis ######
library(mda)
set.seed(5790)
mdaFit <- train(x = x_train,
              y = y_train,
              method = "mda",
              metric = "Kappa",
              preProcess = c('center', 'scale'),
              tuneGrid = expand.grid(.subclasses = 1:10),
              trControl = ctrl)
mdaFit
plot(mdaFit)
pred_MDA <- predict(mdaFit, x_test)
confusionMatrix(pred_MDA, y_test)
```

```
############### Neural Networks #############
nnetGrid <- expand.grid(.size = 1:10, .decay = c(0, .1, 1, 2))
maxSize <- max(nnetGrid$.size)
numWts <- (maxSize * (28 + 1) + (maxSize+1)*2) ## 4 is the number of predictors

set.seed(5790)
library(caret)

nnetFit <- train(x=x_train,
                 y = y_train,
                 method = "nnet",
                 metric = "Kappa",
                 preProc = c("center", "scale"),
                 tuneGrid = nnetGrid,
                 trace = FALSE,
                 maxit = 100,
                 MaxNWts = numWts,
                 trControl = ctrl)
nnetFit
plot(nnetFit)
pred_nnet<-predict(nnetFit,x_test)

confusionMatrix(data = pred_nnet,
                reference=y_test)

########## Flexible Discriminant Analysis ############
marsGrid <- expand.grid(degree = 1:3, nprune = 2:15)

# Train the model
fdaTuned <- train(
```

```
  x = x_train,
  y = y_train,
  method = "fda",
  metric = "Kappa",
  preProcess = c('center', 'scale'),
  tuneGrid = marsGrid,
  trControl = trainControl(method = "cv"))

fdaTuned
plot(fdaTuned)
pred_fda <- predict(fdaTuned, x_test)
confusionMatrix(data = pred_fda,
                reference=y_test)

############## Support Vector Machines ##########
set.seed(5790)
library(kernlab)
library(caret)
sigmaRangeReduced <- sigest(as.matrix(x_train))

svmRGridReduced <- expand.grid(.sigma = sigmaRangeReduced[1],
                               .C = 2^(seq(-2, 4, by = 2)))
set.seed(5790)
svmRModel <- train(x=x_train,
                   y = y_train,
                   method = "svmRadial",
                   metric = "Kappa",
                   preProc = c("center", "scale"),
                   tuneGrid = svmRGridReduced,
                   fit = FALSE,
                   trControl = ctrl)
```

```
svmRModel
plot(svmRModel)
pred_svm <- predict(svmRModel, x_test)
confusionMatrix(data = pred_svm,
               reference=y_test)

############ K-Nearest Neighbors #############
set.seed(5790)
knnFit <- train(x=x_train,
               y = y_train,
               method = "knn",
               metric = "Kappa",
               preProc = c("center", "scale"),
               ##tuneGrid = data.frame(.k = c(4*(0:5)+1, 20*(1:5)+1, 50*(2:9)+1)), ## 21 is the best
               tuneGrid = data.frame(.k = 1:50),
               trControl = ctrl)

knnFit
plot(knnFit)
pred_knn<-predict(knnFit, x_test)
confusionMatrix(data = pred_knn,
               reference=y_test)

########## Naive Bayes ##########
install.packages("klaR")
library(klaR)
set.seed(5790)
nbFit <- train(x_train,
              y_train,
              method = "nb",
              metric = "ROC",
```

```
            preProc = c("center", "scale"),
            ##tuneGrid = data.frame(.k = c(4*(0:5)+1, 20*(1:5)+1, 50*(2:9)+1)), ## 21 is the best
            tuneGrid = data.frame(.fL = 2,.usekernel = TRUE,.adjust = TRUE),
            trControl = ctrl)

nbFit
plot(nbFit) #NO TUNING PARAMETER
pred_nb <- predict(nbFit, x_test)
confusionMatrix(data = pred_nb,
             reference=y_test)


important_pred <- varImp(fdaTuned)
plot(important_pred)
head(x_test)
```