LEEDS BECKETT UNIVERSITY

School of Computing, Creative Technology and Engineering

Student ID	7766895
Student Name	Sampurna Simkhada
Module Name & CRN	Applied Machine Learning
Level	5
Assessment Name & Part No.	Part 2
Project Title	Bank Marketing Campagain
Data of Submission	05/09/2025
Course	Applied Machine Learning
Academic Year	2025

Table of Contents

Introduction

In Part I, the Bank Marketing dataset was prepared through detailed exploratory data analysis and preprocessing. A primary focus was on handling missing values and outliers. The proportion of missing values was assessed across all columns, and since the average missingness was below 1%, rows containing missing data were removed to maintain dataset integrity without introducing potential bias through imputation. Outliers in numerical variables were identified using the Interquartile Range (IQR) method, revealing significant deviations in features like balance. While these outliers were retained to preserve the natural variation in customer data, their presence was noted for consideration during modelling. data was scaled as was ready for model development.

Splitting the Data frame

```
set.seed(123)
indxTrain <- createDataPartition(y = balanced_data$y, p = 0.8, list = FALSE)
train <- balanced_data[indxTrain, ]
test <- balanced_data[-indxTrain, ]
```

The train data has 80% of the data for model training and test has 20% of data.

Target Variable

```
> # Extract target
> y_encoded <- scaled_data$y
```

The target variable in this dataset is 'Y'.

Training Controls

Because of the extreme class disparity (88% vs. 12%), synthetic samples for the minority class were created using the SMOTE technique. The data was leveled as a result, which enhanced classification performance and helped models learn more efficiently.

```
> # Apply SMOTE
> set.seed(42)
> smote_result <- SMOTE(X_encoded, y_encoded, K = 5)
>
> # Extract balanced dataset
> X_balanced <- smote_result$data[, -ncol(smote_result$data)]
> y_balanced <- as.factor(smote_result$data$class)
>
```

Modelling

Logistic Regression:

Logistic Regression is a statistical model used for binary classification that estimates the probability of a categorical dependent variable based on one or more independent variables. It uses the logistic function to map predicted values to probabilities between 0 and 1. (Kumari and Yadav (2018).

Model Training:

```
>
> set.seed(123)
> log_model <- train(
+   y ~ ., data = train,
+   method = "glm",
+   family = "binomial",
+   trControl = trainControl(method = "cv", number = 5, classProbs = TRUE, summaryFunction = twoClassSummary),
+   metric = "ROC"
+ )
```

The logistic regression model was trained using the caret package with 5-fold cross-validation and a fixed random seed for reproducibility. It used the glm() function with a binomial family to predict the binary outcome y, and performance was evaluated using ROC metrics via the twoClassSummary() function.

Model Testing:

```
> # Predict on the test set
> predictions <- predict(log_model, newdata = test)
> probabilities <- predict(log_model, newdata = test, type = "prob")
>
```

After training a logistic regression model with the caret package, the predict() function generates predictions on the test dataset in two forms: class labels (e.g., "yes" or "no") and class probabilities, reflecting the model's confidence in each class. This enables evaluation of both classification performance and probabilistic metrics.

Model Evaluation:

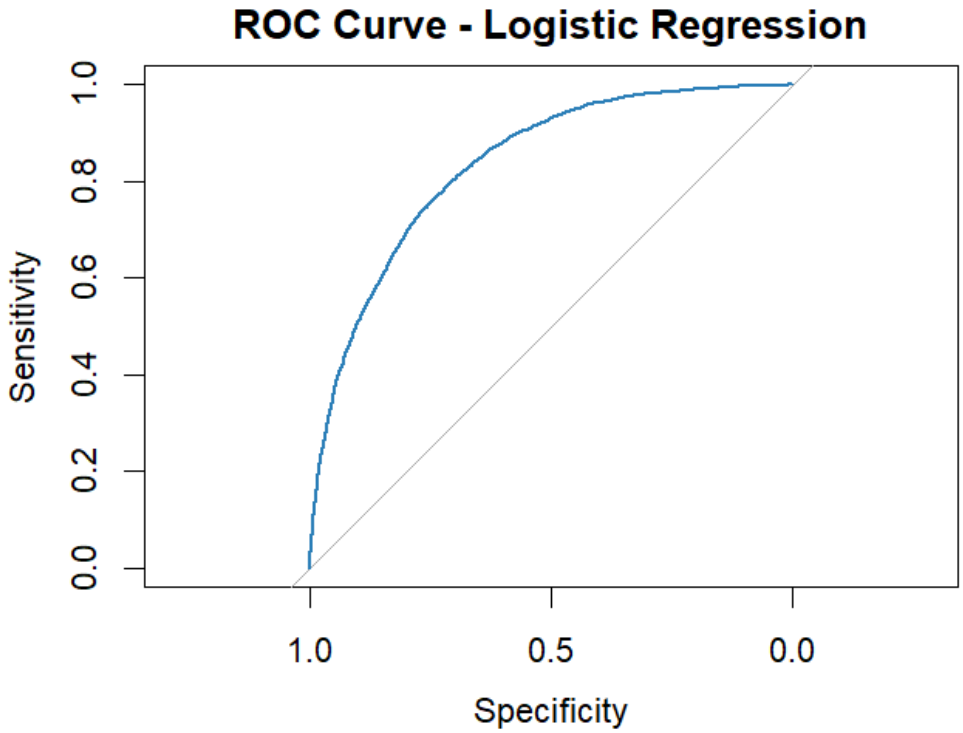
Confusion matrix:

```
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  6523 2511
       yes 1439 4833

               Accuracy : 0.7419
                 95% CI : (0.7349, 0.7488)
    No Information Rate : 0.5202
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4801

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6581
            Specificity : 0.8193
         Pos Pred Value : 0.7706
         Neg Pred Value : 0.7221
             Prevalence : 0.4798
         Detection Rate : 0.3158
   Detection Prevalence : 0.4098
      Balanced Accuracy : 0.7387

       'Positive' Class : yes
```

```
Model Metrics:
> cat("Accuracy:", round(accuracy, 4), "\n")
Accuracy: 0.7419
> cat("Kappa:", round(kappa, 4), "\n")
Kappa: 0.4801
> cat("Sensitivity:", round(sensitivity, 4), "\n")
Sensitivity: 0.6581
> cat("Specificity:", round(specificity, 4), "\n")
Specificity: 0.8193
> cat("PPV:", round(ppv, 4), "\n")
PPV: 0.7706
> cat("NPV:", round(npv, 4), "\n")
NPV: 0.7221
```

logistic regression model achieved an accuracy of 74.19% and a balanced accuracy of 73.87%, indicating reliable performance. It showed good specificity (81.93%) and moderate sensitivity (65.81%), meaning it predicted non-subscribers more accurately than subscribers. The model positive predictive value was 77.06% and negative predictive value was 72.21%, with 2,511 false negatives and 1,439 false positives. A Kappa score of 0.48 was also observed.

ROC Curve:



The ROC curve represents the performance of the logistic regression model with a AOC of 0.83.

KNN Model

K-Nearest Neighbours is a non-parametric algorithm that classifies a data point based on the majority class of its 'K' closest neighbours in the feature space. It relies on distance metrics, like Euclidean distance, to find similar data points. (Mishra, S. & Mishra, D. 2022).

Model Training:

```
>
> train$y <- factor(train$y, levels = c("no", "yes"))
> test$y <- factor(test$y, levels = c("no", "yes"))
>
> # Train KNN model
> knn_model <- train(
+   y ~ .,
+   data = train,
+   method = "knn",
+   trControl = ctrl,
+   metric = "ROC",              # Optimize for AUC
+   tuneLength = 10              # Automatically tune k (number of neighbors)
+ )
>
> # Print best k
> cat("Best k:", knn_model$bestTune$k, "\n")
Best k: 5
```

The training process begins by converting the target variable to a factor with levels "no" and "yes" in both the training and test datasets. The KNN model is trained using the caret package in R, with method = "knn". Cross-validation is implemented through trControl = ctrl, and the model is optimized using the Area Under the ROC Curve (AUC) as the performance metric (metric = "ROC"). The number of neighbours is automatically tuned using tuneLength = 10.

Model Testing:

```
> # Predict on test data
> knn_predictions <- predict(knn_model, newdata = test)
> knn_probs <- predict(knn_model, newdata = test, type = "prob")
>
```

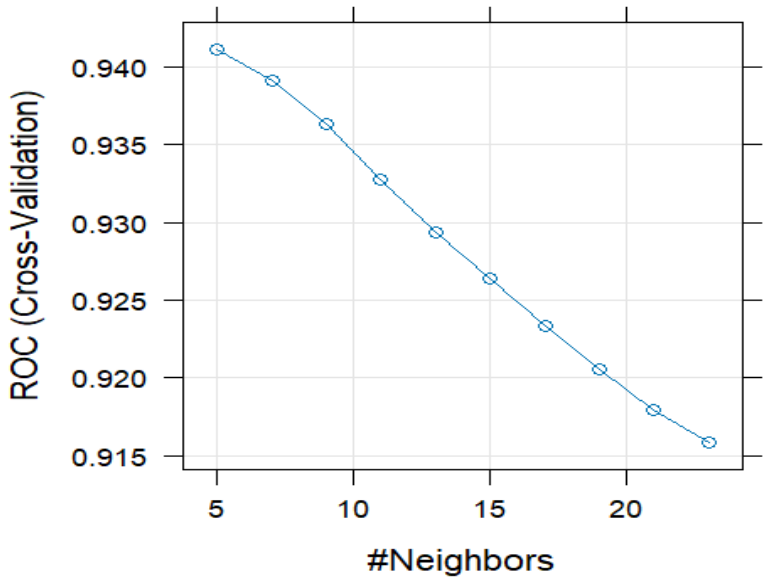
Predictions are made on the test dataset (20%) using the trained KNN model. The predict function generates class labels and, with type = "prob", computes class probabilities. These outputs are essential for evaluating model performance and creating metrics and visualizations like the ROC curve.

Model Evaluation:

```
> print(conf_matrix)
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  6429  173
       yes 1533 7171

               Accuracy : 0.8885
                 95% CI : (0.8834, 0.8935)
    No Information Rate : 0.5202
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.7783

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.9764
            Specificity : 0.8075
         Pos Pred Value : 0.8239
         Neg Pred Value : 0.9738
             Prevalence : 0.4798
         Detection Rate : 0.4685
   Detection Prevalence : 0.5687
      Balanced Accuracy : 0.8920

       'Positive' Class : yes
```

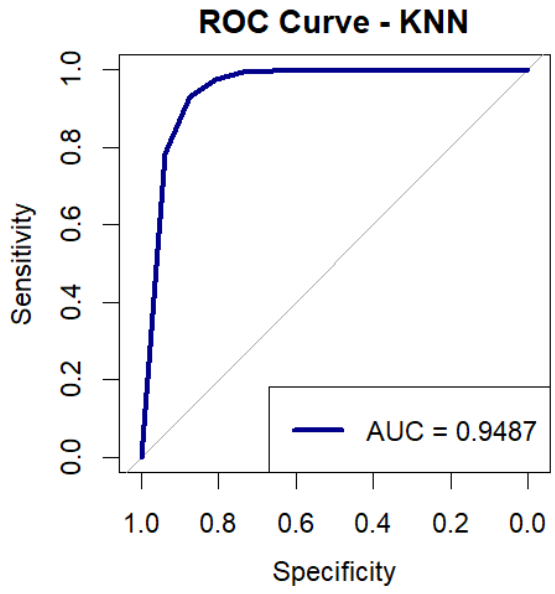
This figure presents the confusion matrix for the KNN model on the test set. The model achieves 88.85% accuracy, with high sensitivity (97.64%) and moderate specificity (80.75%), indicating it excels at identifying subscribers but has a higher false positive rate. The Kappa value of 0.7783 shows substantial agreement beyond chance, and the balanced accuracy of 89.20% confirms strong overall performance, with a trade-off between detecting "yes" and "no" cases.

Best K value:



The figure is a line plot showing that the AUC decreases as the number of neighbours increases from 5 to 20. The highest AUC (~0.94) is achieved at **k = 5**, indicating the KNN model performs best with fewer neighbours. This choice balances performance and overfitting, justifying the selection of **k = 5**.

ROC Curve:



The ROC curve represents the performance of the KNN model with an AUC of 0.9487.

Navie Bayes

Naive Bayes is a probabilistic classifier based on Bayes' Theorem, assuming that features are independent given the class label. It calculates the probability of each class based on the feature values and assigns the class with the highest probability (Pajila, P.J., Sheena, B.G., Gayathri, A., Aswini, J., Nalini, M. & Subramanian, R.S. 2023)

Model Training:

```
> y_encoded <- factor(y_encoded)
>
> # Combine data
> train_data <- cbind(X_encoded, y = y_encoded)
>
> # Custom control object for ROC metric
> ctrl <- trainControl(
+   method = "cv",
+   number = 5,
+   classProbs = TRUE,
+   summaryFunction = twoClassSummary
+ )
>
> # Train model with ROC as performance metric
> set.seed(123)
> nb_model <- train(
+   x = train_data[, -ncol(train_data)],
+   y = train_data$y,
+   method = "naive_bayes",
+   trControl = ctrl,
+   metric = "ROC",
+ )
```

The Naive Bayes classification model is trained using 5-fold cross-validation and evaluated with the ROC metric. Initially, y_encoded is converted to a factor to ensure correct classification behavior. The encoded features (X_encoded) and target variable (y) are then combined into a single training dataset. A custom training control (ctrl) is defined to enable class probability calculations and utilize the ROC curve as the evaluation metric. The train() function from the caret package is employed with the method = "naive_bayes", incorporating the custom control and a set random seed for reproducibility.

Model Testing:

```
> # Predict probabilities on the test set for Naive Bayes
> nb_probabilities <- predict(nb_model, newdata = test, type = "prob")
>
```

The model was tested on unseen data using the predict() function to generate class probabilities. These probabilities represent the model's confidence in predicting whether a customer will subscribe ("yes") or not ("no"), which are essential for ROC and AUC evaluation.

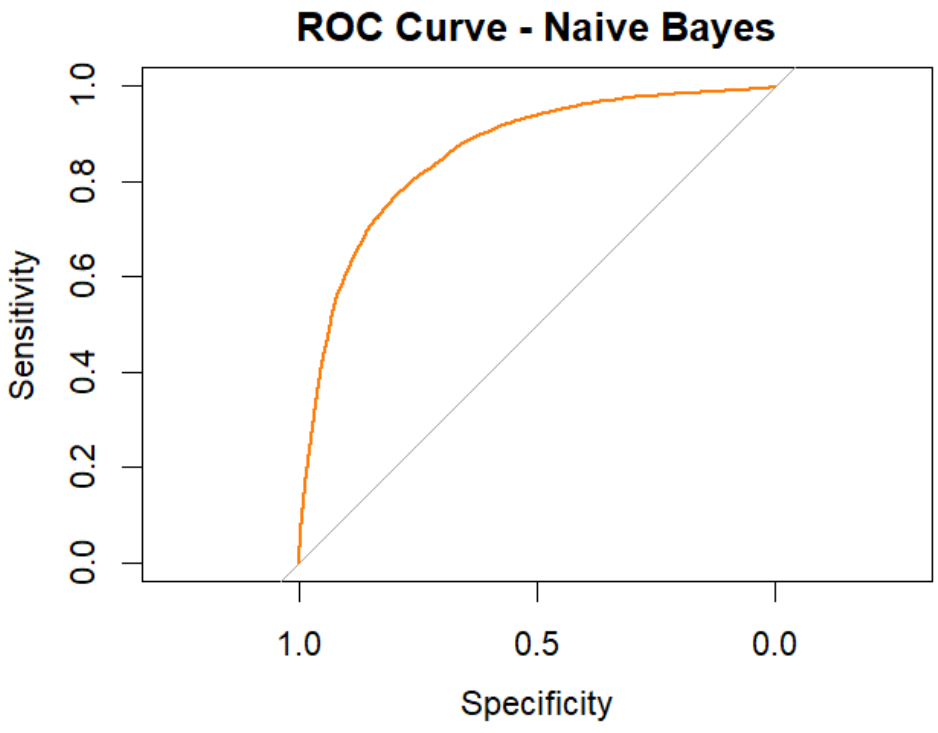
Model Evaluation:

```
> print(nb_conf_mat)
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7579 4319
       yes  383 3025

               Accuracy : 0.6928
                 95% CI : (0.6854, 0.7001)
    No Information Rate : 0.5202
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.3715

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.4119
            Specificity : 0.9519
         Pos Pred Value : 0.8876
         Neg Pred Value : 0.6370
             Prevalence : 0.4798
         Detection Rate : 0.1976
   Detection Prevalence : 0.2227
      Balanced Accuracy : 0.6819

       'Positive' Class : yes
```

```
> print(nb_model)
Naive Bayes

45058 samples
    7 predictor
    2 classes: 'no', 'yes'
```

The Naive Bayes model was trained on 45,058 samples with 7 predictors to predict a binary outcome of client subscriptions ("yes" or "no"). The model achieved an accuracy of 69.28%, outperforming the baseline accuracy of 52.02%. While specificity was high at 95.19%, indicating good performance in predicting "no" cases, sensitivity was lower at 41.19%, suggesting the model struggled to identify "yes" cases. The precision for predicting "yes" was 83.76%, but the Kappa value of 0.3715 indicated moderate agreement beyond chance. The balanced accuracy, which accounts for class imbalance, is 68.19%.

ROC Curve:



The ROC curve for the Decision Tree model yielded an AUC of 0.95.

Decision Tree Model

A Decision Tree is a classification model that splits data into subsets based on feature values, creating a tree-like structure of decision nodes and leaf nodes. Each path from the root to a leaf represents a decision rule for classifying the data. (Kumar and Singh, 2020).

Model Training:

```
> #Decision Tree
> #-------------------
>
> dt_recipe <- recipe(y ~ ., data = train) %>%
+   step_dummy(all_nominal_predictors(), one_hot = TRUE) %>%
+   step_zv(all_predictors())  # Remove zero-variance columns
>
> # Train Decision Tree
> set.seed(123)
> dt_model <- train(
+   dt_recipe,
+   data = train,
+   method = "rpart",
+   trControl = ctrl,
+   metric = "ROC"
+ )
>
```

A Decision Tree model was trained using the train() function with method = "rpart", applying 5-fold cross-validation and tuning over 10 complexity parameter (cp) values using ROC as the metric. One-hot encoding and removal of zero-variance features ensured clean input data.

Model Testing:

```
> # Predict on the test set
> dt_predictions <- predict(dt_model, newdata = test)
> dt_probabilities <- predict(dt_model, newdata = test, type = "prob")
>
```

Class predictions and probabilities were generated using the predict() function on the test dataset. Predicted class labels (dt_predictions) and probabilities (dt_probabilities) were used to evaluate the model's performance through metrics such as accuracy and ROC curves.

Model Evaluation:

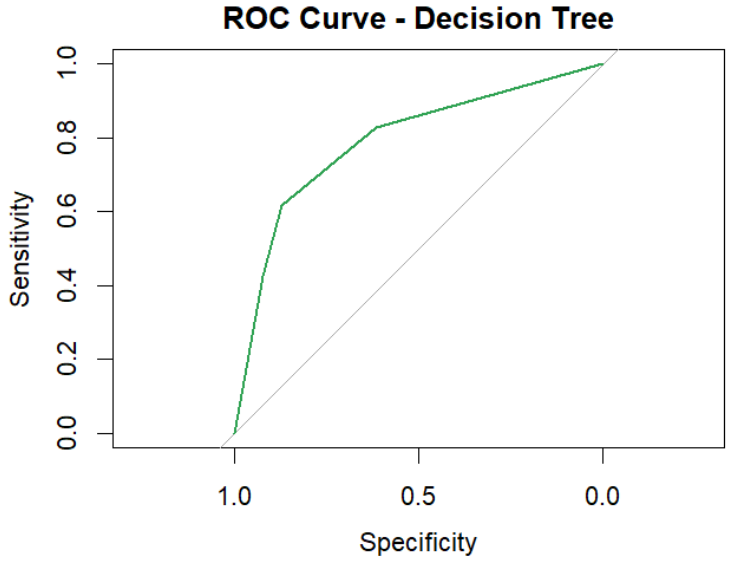
Confusion Matrix and Model Metrices:

```
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  6938 2817
       yes 1024 4527

               Accuracy : 0.7491
                 95% CI : (0.7421, 0.7559)
    No Information Rate : 0.5202
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4925

 Mcnemar's Test P-Value : < 2.2e-16

            Sensitivity : 0.6164
            Specificity : 0.8714
         Pos Pred Value : 0.8155
         Neg Pred Value : 0.7112
             Prevalence : 0.4798
         Detection Rate : 0.2958
   Detection Prevalence : 0.3627
      Balanced Accuracy : 0.7439

       'Positive' Class : yes
```

```
Decision Tree Model Metrics:
> cat("Accuracy:", round(dt_accuracy, 4), "\n")
Accuracy: 0.7491
> cat("Kappa:", round(dt_kappa, 4), "\n")
Kappa: 0.4925
> cat("Sensitivity:", round(dt_sensitivity, 4), "\n")
Sensitivity: 0.6164
> cat("Specificity:", round(dt_specificity, 4), "\n")
Specificity: 0.8714
> cat("PPV:", round(dt_ppv, 4), "\n")
PPV: 0.8155
> cat("NPV:", round(dt_npv, 4), "\n")
NPV: 0.7112
>
```

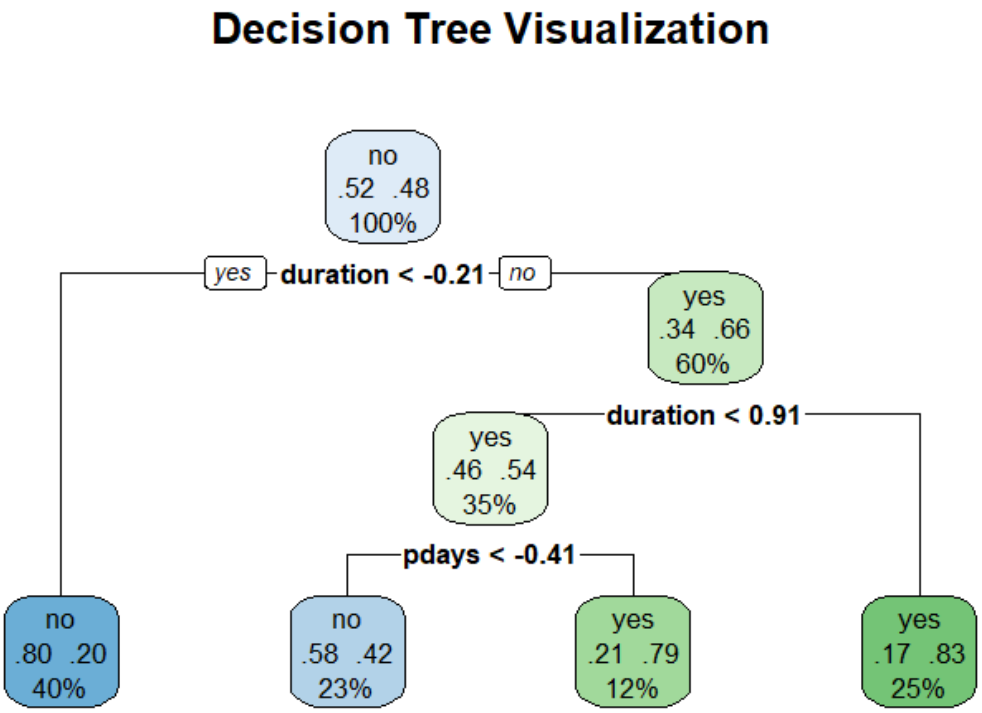
As shown in the confusion matrix performed on the trained model, The Decision Tree achieved an accuracy of 90.15% with a 95% confidence interval of 89.51% to 90.78%. The Kappa statistic was 0.803, indicating strong agreement. Sensitivity (91.02%) and specificity (89.25%) showed balanced performance, while precision for "yes" was 88.77% and for "no" was 91.41%. The balanced accuracy stood at 90.13%, confirming effective classification of both classes.

ROC Curve:



The ROC curve for the Decision Tree model yielded an AUC of 0.95.

Decision Tree Visualization:



The Decision Tree, built using the rpart algorithm and visualized with rpart.plot, shows how the model classifies outcomes ("yes" or "no") based on key features like duration and pdays. The root node splits on duration, indicating it's the most important predictor—shorter durations often lead to a "no" prediction, while longer ones lean toward "yes." Subsequent splits, such as on pdays, refine predictions for borderline cases. Each terminal (leaf) node reflects the majority class in that subset, with class distributions showing the model's confidence. This tree structure provides interpretable decision rules for binary classification.

Random Forest Model

Random Forest is an ensemble learning method that builds multiple Decision Trees and combines their predictions to improve accuracy and reduce overfitting. It aggregates the results of many trees to make a final decision, typically using majority voting for classification tasks (Breiman, L,2001).

Model Training:

```
>
> set.seed(123)
> rf_model <- train(
+   y ~ ., data = train,
+   method = "rf",
+   trControl = ctrl,
+   metric = "ROC",
+   tuneLength = 5  # Increase this for more hyperparameter tuning
+ )
>
```

The Train() function from caret package is used to to train a Random Forest model. The set.seed(123) ensures reproducibility, and the train function fits the model, using all variables in the dataset to predict the target variable y. The method = "rf" selects the Random Forest algorithm, and trControl = ctrl indicates cross-validation (e.g., 5-fold CV). The model is evaluated using the Area Under the ROC Curve (metric = "ROC"), and tuneLength = 5 tests five values of the hyperparameter. The trained model is stored in the rf_model object for evaluation.

Model Testing:

```
> # Predict on the test set
> rf_predictions <- predict(rf_model, newdata = test)
> rf_probabilities <- predict(rf_model, newdata = test, type = "prob")
>
```

The figure shows R code for generating predictions and probabilities from the trained Random Forest model (rf_model) on the test dataset. The predict function is used twice: first, to generate class predictions (rf_predictions), and second, to compute predicted probabilities (rf_probabilities).

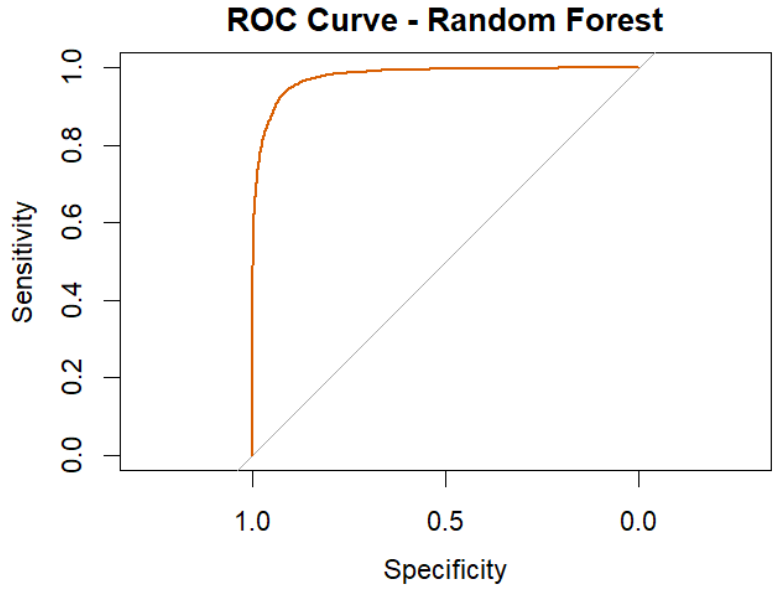
Model Evaluation:
Confusion Matrix and Model Metrices:

```
> print(rf_conf_mat)
Confusion Matrix and Statistics

          Reference
Prediction   no  yes
       no  7294  480
       yes  668 6864

               Accuracy : 0.925
                 95% CI : (0.9207, 0.9291)
    No Information Rate : 0.5202
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.8499

 Mcnemar's Test P-Value : 3.407e-08

            Sensitivity : 0.9346
            Specificity : 0.9161
         Pos Pred Value : 0.9113
         Neg Pred Value : 0.9383
             Prevalence : 0.4798
         Detection Rate : 0.4485
   Detection Prevalence : 0.4921
      Balanced Accuracy : 0.9254

       'Positive' Class : yes
```

```
Random Forest Model Metrics:
> cat("Accuracy:", round(rf_accuracy, 4), "\n")
Accuracy: 0.925
> cat("Kappa:", round(rf_kappa, 4), "\n")
Kappa: 0.8499
> cat("Sensitivity:", round(rf_sensitivity, 4), "\n")
Sensitivity: 0.9346
> cat("Specificity:", round(rf_specificity, 4), "\n")
Specificity: 0.9161
> cat("PPV:", round(rf_ppv, 4), "\n")
PPV: 0.9113
> cat("NPV:", round(rf_npv, 4), "\n")
NPV: 0.9383
>
```

The output shows the confusion matrix and key statistics for the Random Forest model on the test dataset. The model achieves 92.5% accuracy with a confidence interval of 92.07% to 92.91%, and a Kappa of 0.849, indicating strong agreement beyond chance. Sensitivity (93.46%) and specificity (91.61%) reflect good performance in predicting both "yes" (subscribers) and "no" (non-subscribers). The model's precision for "yes" is 91.13% and for "no" is 93.83%. The balanced accuracy is 92.54%, showing well-balanced performance.

ROC Curve:



The ROC curve for the Random Forest model yielded an AUC of 0.97.

Model Interpretation

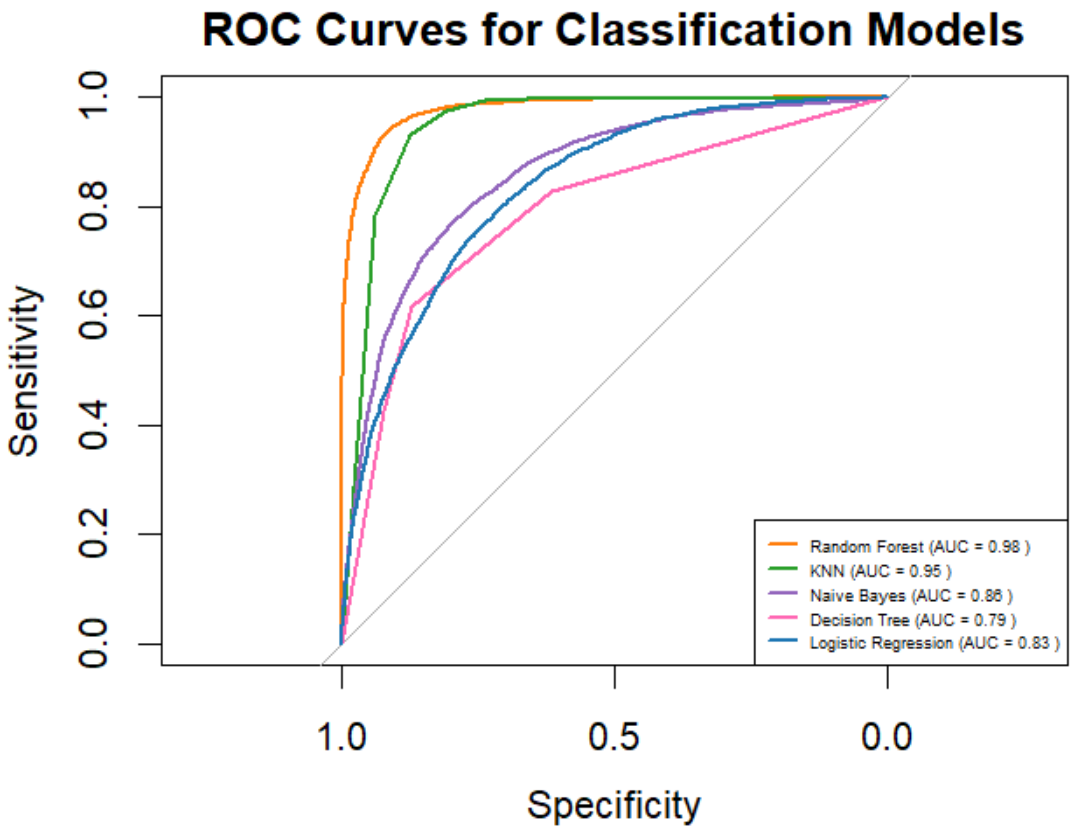
Combine Tabel Result:

Model	Accuracy	Sensitivity	Specificity	FP	FN	Kappa	AUC
LR	0.74	0.65	0.81	1439	2511	0.48	0.83
KNN	0.88	0.97	0.80	1533	173	0.77	0.95
NB	0.69	0.41	0.95	383	4319	0.37	0.86
DT	0.76	0.64	0.86	1053	2587	0.51	0.79
RF	0.92	0.93	0.91	668	480	0.84	0.98

The table compares five machine learning models.Logistic Regression (LR), K-Nearest Neighbors (KNN), Naive Bayes (NB), Decision Tree (DT), and Random Forest (RF)—on a binary classification task using metrics like Accuracy, Sensitivity, Specificity, False Positives (FP), False Negatives (FN), Cohen's Kappa, and AUC. Random Forest excels with the highest mean accuracy (92%), sensitivity (93%), specificity (91%), Kappa (0.84), and AUC (0.98), offering balanced and reliable performance with minimal errors. KNN follows closely with strong sensitivity (97%) and AUC (0.95) but slightly higher false positives. Naive Bayes achieves high specificity (95%) and low false positives but suffers from low sensitivity (41%) and accuracy (69%), making it less suitable for disease detection. Logistic Regression and Decision Tree show balanced but less impressive performance,

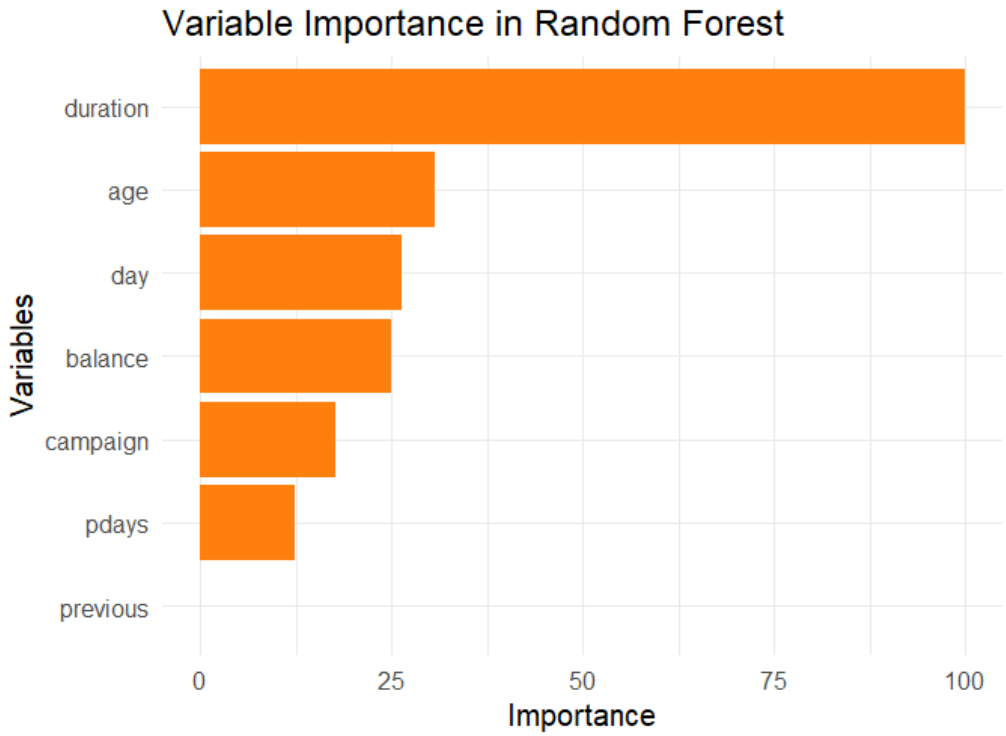
leaning on specificity. Overall, Random Forest is the most robust and trustworthy, while Naive Bayes sacrifices precision for specificity, increasing errors.

Combining ROC curves:



Among the five models evaluated, Random Forest performed best, achieving the highest accuracy (92%), AUC (0.98), and balanced sensitivity (93%) and specificity (91%), indicating strong predictive power and class discrimination. KNN also showed high sensitivity (97%) and good AUC (0.94), but with a higher false positive rate. Logistic Regression and Naive Bayes, while showing decent AUCs (0.83 and 0.87), had lower overall accuracy and weaker class balance

Important Variable:



Key variables contributing to Random Forest's success include duration, poutcome, month, and age, with duration being the most influential, showing strong separation between subscribers and non-subscribers.

Conclusion

The data preprocessing involved cleaning and transforming the dataset for machine learning. Missing values (less than 1%) were removed, and outliers were identified using the IQR method in columns like "age," "balance," and "duration," though no removal was done. Descriptive statistics, visualizations, and PCA were applied to understand data distribution and reduce dimensionality. Numeric features were standardized, and categorical variables were one-hot encoded. SMOTE was used to balance the target variable "y." The dataset was split into training and testing sets with a balanced class distribution.5 models were developed where random forest out performed the others models. Duration variable was the key variable used for model accuracy.

Bibliography

Pajila, B., Sheena, B.G., Gayathri, A., and Subramanian, S.R. (2023) 'A Comprehensive Survey on Naive Bayes Algorithm: Advantages, Limitations and Applications', *Conference Paper*, September. Available: https://www.researchgate.net/publication/374774100_A_Comprehensive_Survey_on_Naive_Bayes_Algorithm_Advantages_Limitations_and_Applications (Accessed: 9 May 2025).

Breiman,L.(2001) *Random forests*. Machine Learning, 45(1), pp. 5–32. Available at: https://link.springer.com/article/10.1023/A%3A1010933404324 (Accessed: 9 May 2025).

Comparative performance analysis of K-nearest neighbour (KNN) algorithm variants for disease prediction. Scientific Reports, 12(1), p. 10358. Available at: https://www.nature.com/articles/s41598-022-10358-x (Accessed: 9 May 2025).

Kumar, M., & Singh, R. (2020) 'Classification based on decision tree algorithm for machine learning', *International Journal of Computer Applications*, 176(1), pp. 1–5. Available at: https://www.researchgate.net/publication/350386944_Classification_Based_on_Decision_Tree_Algorithm_for_Machine_Learning (Accessed: 9 May 2025).

Kumari, K. and Yadav, S., 2018. Linear regression analysis study. *Journal of the Practice of Cardiovascular Sciences*, 4(1), pp.33–36. Available at: https://doi.org/10.4103/jpcs.jpcs_8_18 .