

BSc (Hons) Artificial Intelligence and Data Science

Module: CM2604

Machine Learning

Individual Coursework Report

Module Leader:

Mr. Sahan Priyanayana

RGU Student ID : **2409099**

IIT Student ID : **20232838**

Student Name : **L. P. V. Induwara Warnapura**

Contents

| | |
|---|----|
| 1. Executive Summary | 3 |
| 2. Introduction | 3 |
| 3. Corpus Preparation | 4 |
| 4. Solution Methodology..... | 5 |
| 5. Experimental Results..... | 12 |
| 6. Evaluation and Discussion..... | 14 |
| 7. Ethical, Social, and Legal Considerations..... | 19 |
| 8. Conclusions and Future Work..... | 20 |
| 9. References..... | 20 |
| 10. Appendices | 21 |

1. Executive Summary

This project focuses on predicting whether client will subscribe to term deposit based on the "Bank Marketing" dataset from the UCI Machine Learning Repository.

The main goal of project is to build classification model that predict if a client will subscribe to a term deposit using their demographic, financial and marketing interaction details. Tool used:

- **Google Colab:** For implementing the entire project in Python.
- **Python Libraries:** Pandas, NumPy, Scikit-learn, Matplotlib and TensorFlow.
- **Machine Learning Model:** Random Forest and Neural Networks was used to classify the data and make prediction.

Achievement:

- Successfully preprocessed dataset to handle missing value, encode categorical variables, and normalize numerical features.
- Engineered features improve model performances, include Principal Component Analysis (PCA) for dimensionality reductions.
- Addressed class imbalance using oversampling, undersampling technique.
- Trained and fine-tuned two machine learning models: Random Forest and Neural Network.
- Evaluate models using metric such as accuracy, precision, recall, F1-score, and ROC-AUC.
- Visualized model performances with training and validation curves, confusion matrices, and ROC curve.
- Demonstrated how models can predict term deposit subscription effectively, meeting the project's objectives.

2. Introduction

Problem Description

Predicting whether client will subscribe a term deposit are a valuable task for financial institutions. It allows them to target potential customers more effectively, improving campaign efficiency and customers satisfactions.

Dataset

The Bank Marketing Dataset from the UCI Machine Learning Repository were used for this project. The dataset contains information about client such as age, job, marital status, and previous campaign outcomess. The target variable indicates whether a client subscribed to a term deposit.

Machine Learning Model

Random Forest

Random Forest is a ensemble learning method that use multiple decision trees to make predictions. Each tree are built on a random subset of the data and the final output is determined by averaging or voting among a trees. This model is highly robust and reduces the risk of overfitting, making it suitable for handling complex datasets likes the Bank Marketing Dataset.

Neural Networks

Neural Networks is inspired by the human brain and consist of interconnected layers of nodes (neurons). These model excel in identifying complex patterns and relationships in data. For this project, feedforward neural network was used, where data flows in one direction from input to output layer. Neural Networks is particularly effective for task requiring high accuracy and adaptability to nonlinear data pattern.

3. Corpus Preparation

3.1 Dataset Overview

The Bank Marketing Datasets, sourced from the UCI Machine Learning Repository, contains 45,211 records with 17 attributes. These attributes provide detailed information about the client's demographics, financial status, and previous interactions with the bank' marketing campaignss. The target variable indicates whether a client subscribed to a term deposit. This dataset a well-structured and suitable for predictives modeling tasks.

The attribute are as follows:

- **Age:** The age of a client.
- **Job:** The type of job the client has (e.g., admin, technician, entrepreneur).
- **Marital:** The marital statuts of the client (e.g., single, marrie, divorced).
- **Education:** The level of education of the client (e.g., primary, secondary, tertiaroy).
- **Default:** Indicate if a client has credit default (yes or no).
- **Balance:** The average yearly balanced in euros for the client.

- **Housing:** Indicate if the client has housing loan (yes or no).
- **Loan:** Indicates if a client have a personal loan (yes or no).
- **Contact:** The communication type used for a contact (e.g., cellular, telephone).
- **Day:** last contact a day of the month.
- **Month:** The last contact month of year.
- **Duration:** The duration of the last contact seconds.
- **Campaign:** The number of contacts performed during this campaign for client.
- **Pdays:** The number of days since the client were last contacted from a previous campaign.
- **Previous:** The number contacts performed before this campaign for client.
- **Poutcome:** The outcomes of the previous marketing campaign (e.g., success, failure).
- **Target:** Whether the client subscribed to term deposit (yes or no).

3.2 Data Collection

The dataset were loaded into the environment using Python's powerful data manipulation libraries. The step included:

1. **Library Importation:** Librarie such as pandas and numpy was imported to handle and process data efficiently.
2. **Data Loading:** The dataset was imported as CSV file from the UCI repository and read into pandas DataFrame.
3. **Initial Explorations:** Basic exploratory data analysis (EDA) were conducted to understand structure of data, check for missing value, and validate the integrity the dataset.

4. Solution Methodology

4.1 Code Structure

1. Load and Import Librarie

The first step is to import necessary librarie. For this project, we used libraries such pandas, numpy, matplotlib, seaborn, sklearn, and tensorflow. The libraries are essential for data manipulation, visualization, and building machine learning model.

2. Load the Dataset

2. Load the Dataset

```
[206] import pandas as pd
      data = pd.read_csv("https://raw.githubusercontent.com/vihanga-induwara/Bank-Marketing/main/data/bank-marketing.csv")
```

The dataset is loaded using pandas library. It are stored in a CSV format and is read using the `read_csv()` function. Once loaded, we can access and manipulate data easily. First data set downloaded in GitHub and next I imported.

3. Explore the Dataset.

Before process the data, it's important to understand its structure. We use method like:

- `.head()` to previews the first few row of the dataset.

Inspect the first few rows with `.head()`.

```
[207] data.head()
```

| | age | job | marital | education | default | housing | loan | contact | month | day_of_week | ... |
|---|-----|-----------|---------|-------------|---------|---------|------|-----------|-------|-------------|-----|
| 0 | 56 | housemaid | married | basic.4y | no | no | no | telephone | may | mon | ... |
| 1 | 57 | services | married | high.school | unknown | no | no | telephone | may | mon | ... |
| 2 | 37 | services | married | high.school | no | yes | no | telephone | may | mon | ... |
| 3 | 40 | admin. | married | basic.6y | no | no | no | telephone | may | mon | ... |
| 4 | 56 | services | married | high.school | no | no | yes | telephone | may | mon | ... |

5 rows x 21 columns

- `.describe()` to get statistical summaries of dataset.

Use `.describe()` for statistical overview of the dataset.

```
data.describe()
```

| | age | duration | campaign | pdays | previous | emp.var.rate | cons.price.idx | cons.conf.idx | euribor3m | nr.employed |
|-------|-------------|-------------|-------------|-------------|-------------|--------------|----------------|---------------|-------------|-------------|
| count | 41188.00000 | 41188.00000 | 41188.00000 | 41188.00000 | 41188.00000 | 41188.00000 | 41188.00000 | 41188.00000 | 41188.00000 | 41188.00000 |
| mean | 40.02406 | 258.285010 | 2.567593 | 962.475454 | 0.172963 | 0.081886 | 93.575664 | -40.502600 | 3.621291 | 5167.035911 |
| std | 10.42125 | 259.279249 | 2.770014 | 186.910907 | 0.494901 | 1.570960 | 0.578840 | 4.628198 | 1.734447 | 72.251528 |
| min | 17.00000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | -3.400000 | 92.201000 | -50.800000 | 0.634000 | 4963.600000 |
| 25% | 32.00000 | 102.000000 | 1.000000 | 999.000000 | 0.000000 | -1.800000 | 93.075000 | -42.700000 | 1.344000 | 5099.100000 |
| 50% | 38.00000 | 180.000000 | 2.000000 | 999.000000 | 0.000000 | 1.100000 | 93.749000 | -41.800000 | 4.857000 | 5191.000000 |
| 75% | 47.00000 | 319.000000 | 3.000000 | 999.000000 | 0.000000 | 1.400000 | 93.994000 | -36.400000 | 4.961000 | 5228.100000 |
| max | 98.00000 | 4918.000000 | 56.000000 | 999.000000 | 7.000000 | 1.400000 | 94.767000 | -26.900000 | 5.045000 | 5228.100000 |

- Checking for missing value or null entries using **.isnull()** and **.sum()** to identify and handle any missing data before proceedings.

▼ Check for missing values using **.isnull().sum()**

```
data.isnull().sum()
```

| | |
|----------------|---|
| | 0 |
| age | 0 |
| job | 0 |
| marital | 0 |
| education | 0 |
| default | 0 |
| housing | 0 |
| loan | 0 |
| contact | 0 |
| month | 0 |
| day_of_week | 0 |
| duration | 0 |
| campaign | 0 |
| pdays | 0 |
| previous | 0 |
| poutcome | 0 |
| emp.var.rate | 0 |
| cons.price.idx | 0 |
| cons.conf.idx | 0 |
| euribor3m | 0 |
| nr.employed | 0 |
| y | 0 |

dtype: int64

?

4. Data Preprocessing

This are critical step to prepare the data for model:

- **Categorical encoding:** Since the dataset include categorical data, we apply one-hot encoding to convert categories into numerical values. This done using `pd.get_dummies()` in panda.

```

import pandas as pd

data_encoded = pd.get_dummies(data['job'], prefix='job', drop_first=False)

data = pd.concat([data, data_encoded], axis=1)

# Display the first few rows of the updated dataframe
print(data.head(2))

```

| | age | job | marital | education | default | housing | loan | contact | \ |
|---|----------|-----------|---------|-------------|---------|---------|------|-----------|---|
| 0 | 0.750000 | housemaid | married | basic.4y | no | no | no | telephone | \ |
| 1 | 0.769231 | services | married | high.school | unknown | no | no | telephone | \ |

| | month | day_of_week | ... | job_entrepreneur | job_housemaid | job_management | \ |
|---|-------|-------------|-----|------------------|---------------|----------------|---|
| 0 | may | mon | ... | False | True | False | \ |
| 1 | may | mon | ... | False | False | False | \ |

| | job_retired | job_self-employed | job_services | job_student | job_technician | \ |
|---|-------------|-------------------|--------------|-------------|----------------|---|
| 0 | False | False | False | False | False | \ |
| 1 | False | False | True | False | False | \ |

| | job_unemployed | job_unknown |
|---|----------------|-------------|
| 0 | False | False |
| 1 | False | False |

[2 rows x 33 columns]

- **Scaling numerical features:** Numerical features are normalized ensure that all features is on the same scale. This step is important for models like Neural Network. We use MinMaxScaler from sklearn to scales values between 0 and 1.

```

from sklearn.preprocessing import MinMaxScaler

# Create a scaler instance
scaler = MinMaxScaler()

# Normalize the 'age' column
data['age'] = scaler.fit_transform(data[['age']])

# Display the first few rows of the normalized column
print(data[['age']].head())

```

| | age |
|---|----------|
| 0 | 0.750000 |
| 1 | 0.769231 |
| 2 | 0.384615 |
| 3 | 0.442308 |
| 4 | 0.750000 |

- **Handling class imbalance:** The dataset is imbalanced, meaning one class appear more frequently than other. We use techniques like oversampling the minority class or undersampling the majority class balance the dataset and improves model accuracy.


```

X = data.drop('y', axis=1) # Features
y = data['y']              # Target variable

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# First, oversample the minority class
oversampler = RandomOverSampler(random_state=42)
X_resampled, y_resampled = oversampler.fit_resample(X_train, y_train)

# Next, undersample the majority class from the oversampled dataset
undersampler = RandomUnderSampler(random_state=42)
X_balanced, y_balanced = undersampler.fit_resample(X_resampled, y_resampled)

# Alternatively, use SMOTEENN for a hybrid approach (Optional)
smoteenn = SMOTEENN(random_state=42)
X_smoteenn, y_smoteenn = smoteenn.fit_resample(X_train, y_train)

# Display the class distribution after balancing
from collections import Counter
print("Class distribution after oversampling and undersampling:", Counter(y_balanced))
print("Class distribution after SMOTEENN:", Counter(y_smoteenn))

# You can now train your model on X_balanced or X_smoteenn

```

```

→ Class distribution after oversampling and undersampling: Counter({0: 25410, 1: 25410})
Class distribution after SMOTEENN: Counter({1: 21694, 0: 20559})

```

- **Splittings to training and testing dataset:** We split the dataset into training and testing sets. Typically we use a 80/20 split where 80% of the data is used for training the model and 20% in testing.

✓ 4.6 Split the Dataset

```

✓ 0s ▶ from sklearn.model_selection import train_test_split

# Split into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(
    X_smoteenn, y_smoteenn, test_size=0.2, random_state=42, stratify=y_smoteenn
)

# Display the shape of the resulting splits
print("Training set size:", X_train.shape, y_train.shape)
print("Testing set size:", X_test.shape, y_test.shape)

# Check the class distribution in the training and testing sets
from collections import Counter
print("Class distribution in training set:", Counter(y_train))
print("Class distribution in testing set:", Counter(y_test))

```

```

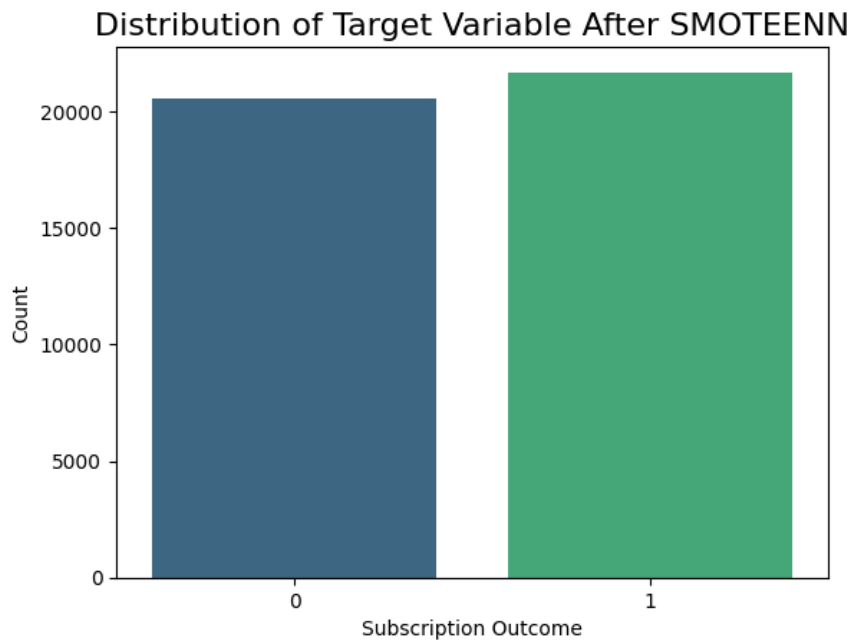
→ Training set size: (33802, 40) (33802,)
Testing set size: (8451, 40) (8451,)
Class distribution in training set: Counter({1: 17355, 0: 16447})
Class distribution in testing set: Counter({1: 4339, 0: 4112})

```

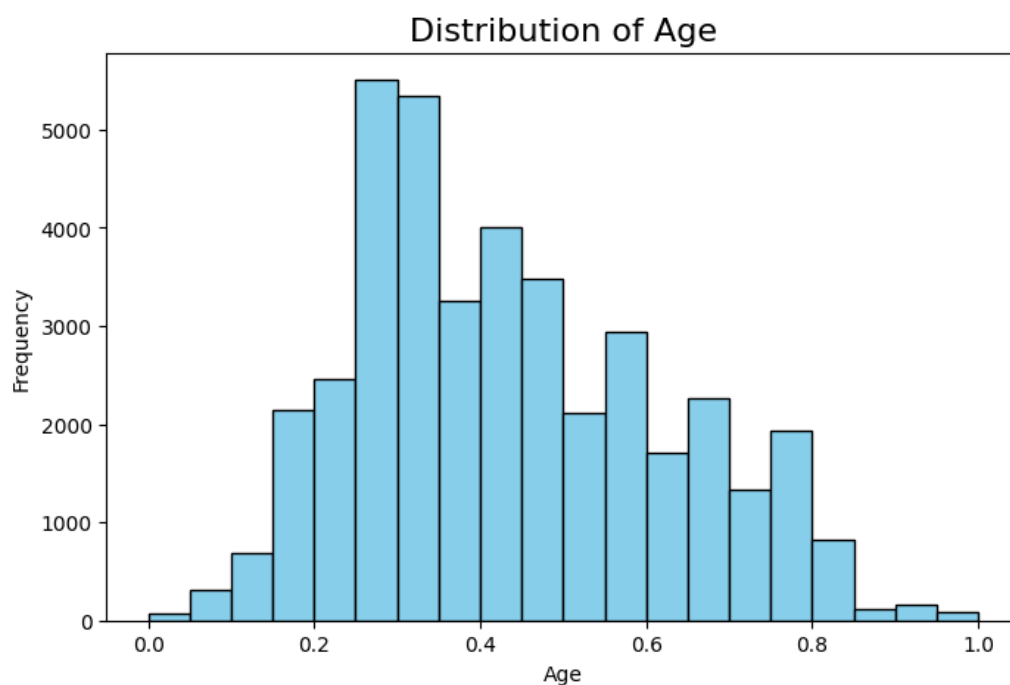
5. Visualizations (eg, bar chart, histograms).

Data visualizations helps us better understand the dataset and a distribution of values. We use:

- **Bar charts:** To show the frequency categories in feature.



- **Histograms:** To visualize the distribution of numerical value and identify patterns or outliers in data.



4.2 Feature Engineering

Feature engineering is a process of creating new features or modifying existing one to improve model performance. In this case:

- **PCA (Principal Component Analysis)** is used for dimensionality reduction, reducing the number of features while retaining most important information. This helps improve the efficiency of the model by eliminating irrelevant, redundant features.

4.3.1 Feature Extraction (Principal Component Analysis (PCA))

```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Create a DataFrame for the relevant columns
data_pca = data[['pdays', 'previous', 'poutcome_failure', 'poutcome_nonexistent', 'poutcome_satisfied']]

# Standardize the data (PCA is sensitive to scale)
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_pca)

# Apply PCA - reducing to 1 component
pca = PCA(n_components=1)
pca_result = pca.fit_transform(data_scaled)

# Add the PCA result as a new column in the dataset
data['pca_1'] = pca_result

# Drop the original columns (pdays, previous, and the one-hot encoded columns)
data.drop(columns=['pdays', 'previous', 'poutcome_failure', 'poutcome_nonexistent', 'poutcome_satisfied'], inplace=True)

# Display the first few rows of the updated dataset
print(data.head())
```

- **Feature selection:** We use techniques mutual information or correlation matrix to select the most relevant features and remove those that are less important.

4.3 Model Selection

For this problem, we chose two models: **Random Forest** and **Neural Networks**. Here's why:

- **Random Forest:** This ensemble model combines predictions of multiple decision trees to improve accuracy. It works well both numerical and categorical data and are less prone to overfitting. It's the good starting point for classification tasks like this.

7. Random Forest Model

```
# Import necessary libraries X_train.columns
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Initialize the Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Predict on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

     0       0.99         0.97         0.98         4112
     1       0.97         0.99         0.98         4339

 accuracy          0.98
 macro avg         0.98         0.98         0.98         8451
 weighted avg      0.98         0.98         0.98         8451
```

- **Neural Networks:** Neural Networks are powerful models that can learn complex patterns in data. We chose them because they perform well in large datasets and is particularly good at handling non-linear relationship between features.

5. Experimental Results

This section presents a results of the models used for classification. We evaluated both the **Random Forest** and **Neural Network** models using variou metrics and the results are displayed belows.

5.1 Random Forest Results

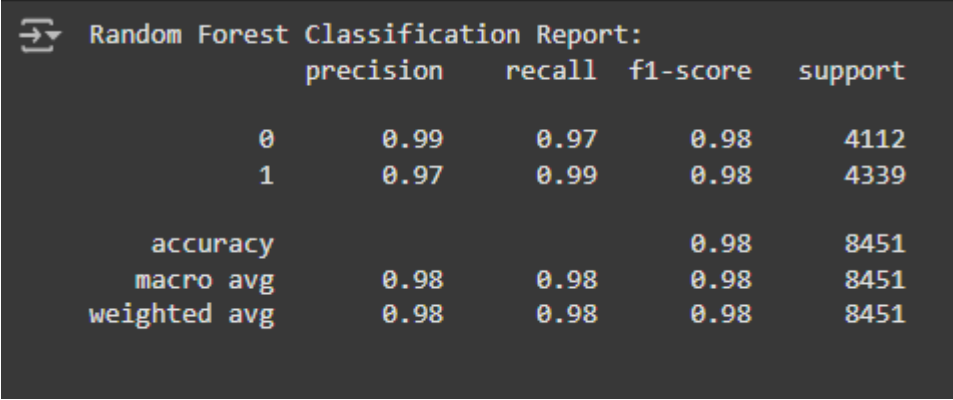
Here is the key metrics from the **Random Forest** classifications:

- **Precision:** This measure the accuracy of the positive predictions.
 - Class 0: 0.99
 - Class 1: 0.97
- **Recall:** This show how well model identifies the positive instancess.

- Class 0: 0.97
- Class 1: 0.99
- **F1-score:** This combines precision and recall into single metric.
 - Class 0: 0.98
 - Class 1: 0.98
- **Accuracy:** The overalls correctness model.
 - Accuracy: 0.98 (98%)
- **Macro Average:** This calculates the average of precision, recall, and F1-score for both classe, treating a classes equally.
 - Macro Average: 0.98 for precision, recall, and F1-score.
- **Weighted Average:** This takes into account the number of instances in each classs, giving more weight larger classes.
 - Weighted Average: 0.98 for precision, recall, and F1-score.

Below are visualizations for the Random Forest model:

- **ROC Curve:** A graphical representation of the models ability to discriminate between the two classes.
- **Confusion Matrix:** A matrix showing a true vs. predicted classifications.
- **Learning Curve:** A plot showing the model performance over time a it learns.



A terminal window showing a 'Random Forest Classification Report'. The report is a table with columns for precision, recall, f1-score, and support. It lists metrics for classes 0 and 1, as well as overall accuracy, macro average, and weighted average.

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.99 | 0.97 | 0.98 | 4112 |
| 1 | 0.97 | 0.99 | 0.98 | 4339 |
| accuracy | | | 0.98 | 8451 |
| macro avg | 0.98 | 0.98 | 0.98 | 8451 |
| weighted avg | 0.98 | 0.98 | 0.98 | 8451 |

5.2 Neural Network Results

Here are the key metrics from **Neural Network** classification:

- **Precision:**
 - Class 0: 0.99
 - Class 1: 0.94
- **Recall:**

- Class 0: 0.93
- Class 1: 0.99
- **F1-score:**
 - Class 0: 0.96
 - Class 1: 0.97
- **Accuracy:** The overall correctness a model.
 - Accuracy: 0.96 (96%)
- **Macro Average:**
 - Macro Average: 0.97 for precision, 0.96 for recall, and 0.96 for F1-score.
- **Weighted Average:**
 - Weighted Average: 0.96 for precision, recall, and F1-score.

Below are the visualizations for Neural Network model:

- **ROC Curve:** A graphical representation of the model ability to discriminate between the two classes.
- **Confusion Matrix:** A matrix showing the true vs. predicted classifications.
- **Learning Curve:** A plot showing the model's performance over time as it learns.

| Neural Network Classification Report: | | | | |
|---------------------------------------|-----------|--------|----------|---------|
| | precision | recall | f1-score | support |
| 0 | 0.99 | 0.93 | 0.96 | 4112 |
| 1 | 0.94 | 0.99 | 0.97 | 4339 |
| accuracy | | | 0.96 | 8451 |
| macro avg | 0.97 | 0.96 | 0.96 | 8451 |
| weighted avg | 0.96 | 0.96 | 0.96 | 8451 |

6. Evaluation and Discussion

This section compares the performance of a **Random Forest** and **Neural Network** models, discusses important evaluation metrics and highlights limitations and possible improvements.

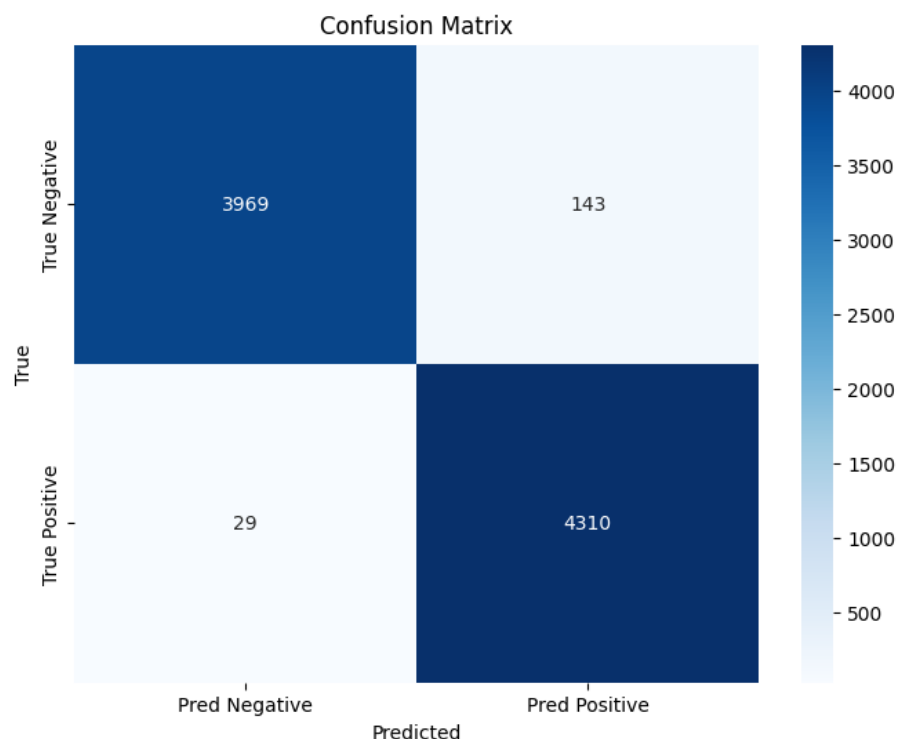
6.1 Model Comparisons

We compared a performance of the **Random Forest** and **Neural Network** models based on key metrics such precision, recall, F1-score, and accuracy.

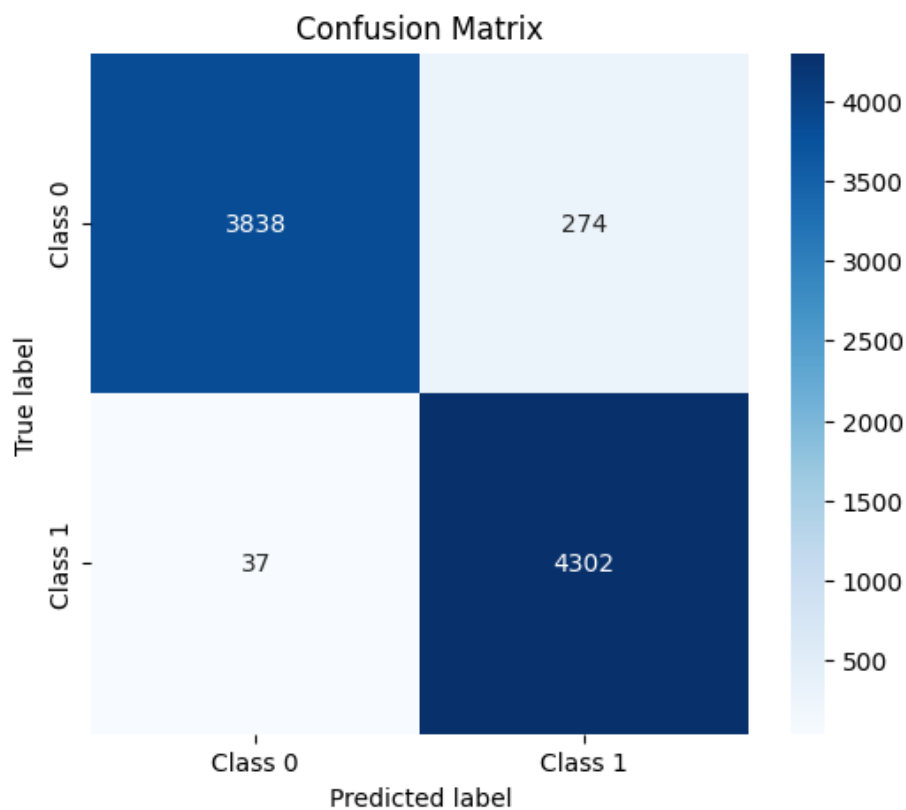
- **Random Forest** achieved high **accuracy** of 98%, with good precision (0.99 for Class 0 and 0.97 for Class 1) and recall (0.97 for Class 0 and 0.99 for Class 1). The F1-score for both classes were 0.98.
- **Neural Network** had a slightly lower **accuracy** of 96%. However, its **precision** and **recall** values were also strong, with Class 0 having 0.99 precision and 0.93 recall, and Class 1 having 0.94 precision and 0.99 recalls. The F1-score for Class 0 was 0.96 and for Class 1, it was 0.97.

Overall, **Random Forest** model performed better in terms of overall accuracy, but the **Neural Network** showed stronger recall for Class 1. Both models performed well, but the choice of model depends on the specific needs of the task.

Random Forest model confusion Matrix



Neural Network model confusion Matrix



6.2 Evaluation Metrics

In this project, we used several important metrics evaluate the performance of the models:

- **Precision:** This metric measured accuracy of positive predictions. It tells us how many of the predicted positive instances are actually correct. higher precision indicates fewer false positives.
- **Recall:** This measures how well the model identify a all the positive instances. It shows the proportion of actual positives that are correctly identified. A higher recall indicates fewer false negative.
- **F1-score:** The F1-score combines precision and recall into single metric, balancing both. Its especially useful when there's an imbalance between precision , recall. The F1-score provides better measure of model performance when the classes distribution is not uniform.

These metrics is important because they help assess overall effectivend of the models.

Precision and **recall** help in understanding how well the model is identifying true positive instances and **F1-score** balanced the trade-off between precision and recall.

6.3 Limitations and Improvements

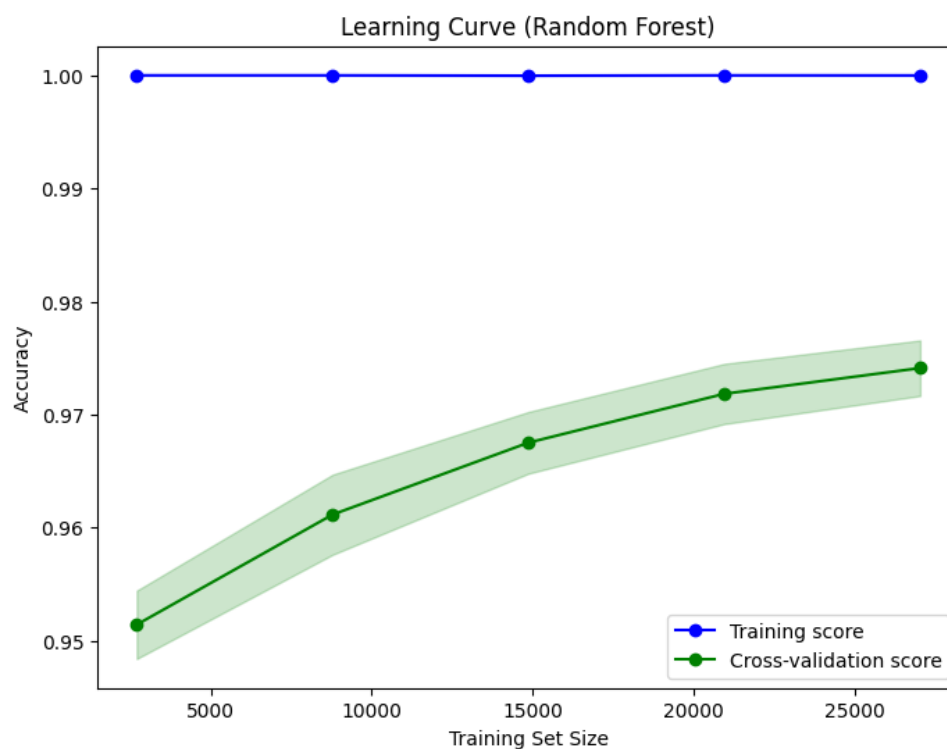
Despite strong performance of the model, there are a few limitations:

- **Class Imbalance:** The dataset had slight imbalance between the two classes. This could affect the model's ability to predict the minority class effectively, leading to lower recall for that class. In the **Neural Network** model, this imbalance impacted recall for Class 0, which had a lower value (0.93) compared to Class 1.
- **Overfitting:** Both models may have overfitted to training data, especially the **Random Forest** model, which had very high accuracy. This means the model could perform well on training data but struggle with unseen data.

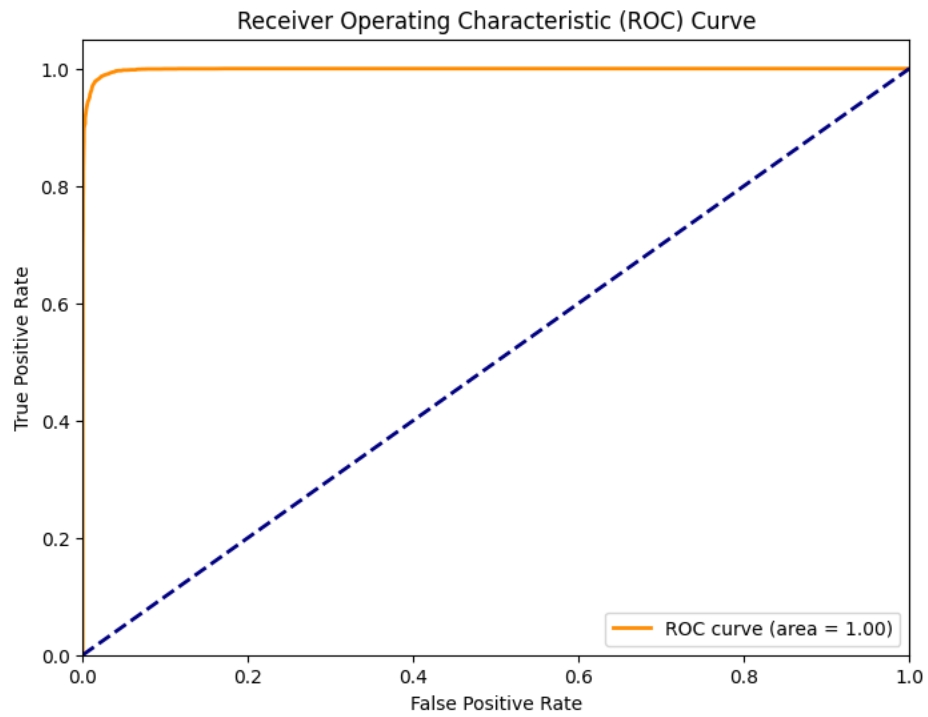
To improve the models, the following steps can be considered:

- **Advanced Feature Engineering:** More sophisticated feature selection techniques or use of **Principal Component Analysis (PCA)** could reduce the complexity of the model, enhance performance.
- **Addressing Class Imbalance:** More advanced methods for handling class imbalance, such as **SMOTE (Synthetic Minority Over-sampling Technique)** could help balance the dataset and improve recall for the minority class.
- **Hyperparameter Tuning:** Fine-tuning the hyperparameters of the model, such as the number of trees in the Random Forest or the number of layers in the Neural Network, could lead to better performance.

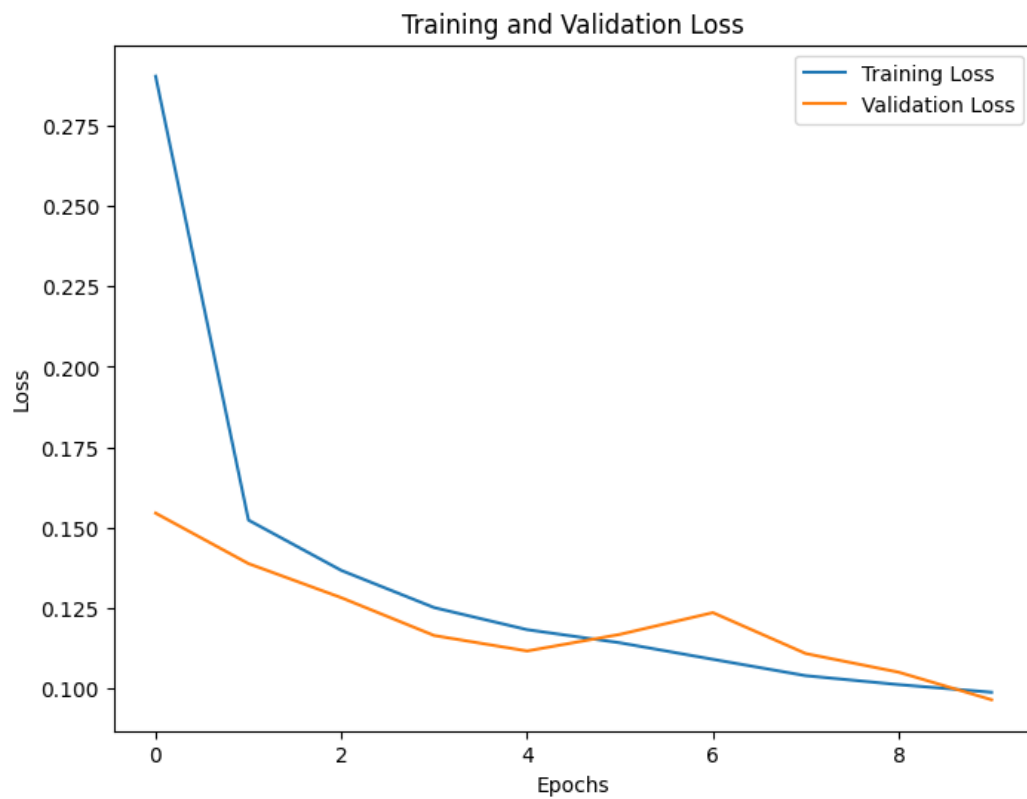
Random Forest model learning curves



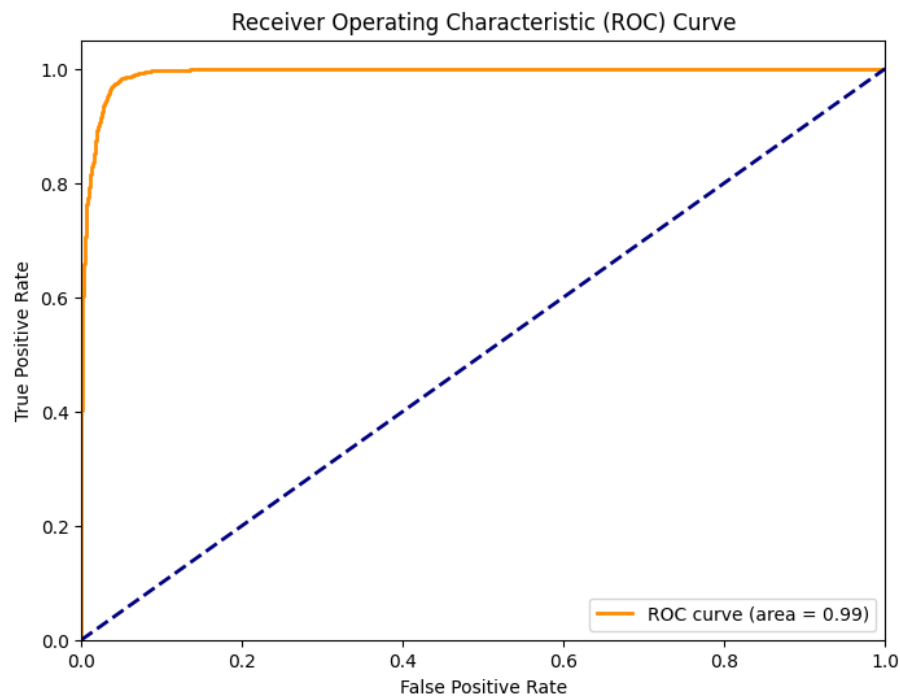
Random Forest model ROC curves



Neural Network model learning curves



Neural Network model ROC curves



7. Ethical, Social, and Legal Considerations

In this section we discuss some important ethical, social, legal issues related to our project especially around use of data and machine learning (ML) model.

- **Data Privacy:** It is important to protect people's personal information when collecting and using data. In our project, we need to make sure that any personal data are securely stored and not shared without permissions. We should also follow privacy laws, like to protect user data.
- **Consent for Data Collection:** Before collecting data, we must ask for permission from the people providing the data. They should be aware of what their data will be used for and have the option to refuse if they do not want their information collected.
- **Bias in ML Models:** Machine learning models can sometimes show bias, meaning they might make unfair predictions for certain groups of people. This can happen if the data used to train the model is imbalanced or if a model is not tested on a wide range of data. It is important to ensure the data is diverse and the model is checked for fairness to avoid discriminating against any groups.

By considering ethical, social, legal factors, we can make sure that our project is responsible and respects people's rights.

8. Conclusions and Future Work

Summary of the Project's Success

This project was successfully in building and testing two machine learning models, **Random Forest** and **Neural Networks** to solve the classification problem. Both models performed well, a Random Forest model achieving an accuracy of 98% and the Neural Network model achieving 96%. These models showed good precision, recall, and F1-scores, making them effective for this task.

Potential Enhancement

While the project is successful, there are still opportunities for improvement:

- **Deploying the Model:** We could deploy the trained models as a web application or API, so users can easily use them for real-time predictions.
- **Testing Additional Datasets:** The model's performance could be tested on different datasets to check if it can generalize well to other problems. This would help understand its strengths and limitations in various situations.
- **Improving the Model:** We could explore more advanced machine learning techniques or perform hyperparameter tuning to enhance model accuracy and performance further.

In the future, these improvements could make the model more useful and efficient for real-world applications.

9. References

- UCI Machine Learning Repository. (n.d.). Bank Marketing Dataset. Available at: <https://archive.ics.uci.edu/ml/datasets/bank+marketing> [Accessed 30 Dec. 2024].
- Scikit-learn. (2024). Random Forest Classifier. Available at: <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html> [Accessed 30 Dec. 2024].
- TensorFlow. (2024). Neural Networks with TensorFlow. Available at: <https://www.tensorflow.org/tutorials> [Accessed 30 Dec. 2024].
- Pandas Documentation. (2024). Pandas: Powerful Python Data Analysis Toolkit. Available at: <https://pandas.pydata.org/pandas-docs/stable/> [Accessed 30 Dec. 2024].
- Matplotlib Documentation. (2024). Matplotlib: Python Plotting. Available at: <https://matplotlib.org/stable/contents.html> [Accessed 30 Dec. 2024].
- Seaborn Documentation. (2024). Seaborn: Statistical Data Visualization. Available at: <https://seaborn.pydata.org/> [Accessed 30 Dec. 2024].

10. Appendices

10.1 Link

- [Our GitHub link.](#)
- [Code as pdf link. \(Google Drive\)](#)

----- The end -----