AC & ML Semester Project

1st Jason Wille
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
1352200@students.wits.ac.za

2nd Bongani Shube
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
2112465@students.wits.ac.za

3rd Preshen Goobiah
School of Computer Science and Applied Mathematics
University of the Witwatersrand
Johannesburg, South Africa
1355880@students.wits.ac.za

I. Introduction

Direct marketing campaigns incur substantial costs, particularly when outreach is conducted via SMS, email, or telephone. Effective targeting is therefore essential to maximize return on investment. This study uses the Bank Marketing dataset from the UCI Machine Learning Repository, which includes client demographic attributes and historical interaction data with a financial institution. The primary objective is to build a predictive model capable of identifying clients most likely to subscribe to a term deposit offer. Rather than optimizing for overall classification accuracy, the focus is on maximizing the Area Under the Receiver Operating Characteristic Curve (AUC), a metric that better captures the model's ability to distinguish between positive and negative classes, especially in the presence of class imbalance. To this end, multiple classification algorithms — both linear and non-linear — are evaluated to identify approaches that most effectively support improved targeting performance.

II. Method

A. Dataset Description

The dataset under study originates from a direct marketing campaign conducted by a Portuguese bank [1]. The campaign primarily involved phone calls, and the core objective is to predict whether a client will subscribe to a term deposit offer, making this a binary classification problem. The dataset comprises both demographic and behavioral features, as detailed in Table I.

TABLE I
Features that made up the Bank Marketing Dataset [1]

Feature	Type	Description
Age	Integer	Age of the client
Job	Categorical	Type of job held by the client
Marital	Categorical	Marital status
Education	Categorical	Level of education
Default	Binary	Has credit in default?
Balance	Integer	Average yearly balance in Euros
Housing	Binary	Has a housing loan?
Loan	Binary	Has a personal loan?
Contact	Categorical	Contact communication type
Day of Week	Date	Last contact day of the week
Month	Date	Last contact month of the year
Duration	Integer	Last contact duration measured in seconds
Campaign	Integer	Number of contacts performed during this campaign for this client
Previous Days	Integer	Number of days that passed by after the client was last contacted from a previous campaign
Previous	Integer	Number of contacts performed before this campaign for this client
Previous Outcome	Categorical	Outcome of the previous marketing campaign

B. Data Preprocessing

The `default`, `housing`, and `loan` features originally contained string values "yes" or "no". These were converted to binary format, mapping "yes" to 1 and "no" to 0.

The `age`, `balance`, and `campaign` features were normalized using min-max normalization to scale values to the range $[0, 1]$. This normalization can improve model performance by reducing training time, as it prevents the model from needing to learn scale differences between features.

Categorical features such as `job`, `marital`, `contact`, and `month` were one-hot encoded. Although decision trees can theoretically handle categorical variables, the scikit-learn implementation does not currently support them natively. One-hot encoding was preferred over ordinal encoding for these features, as they do not exhibit any natural ordering; imposing an artificial order could mislead the model.

The `education` and `previous outcome` features were ordinally encoded, based on the natural progression within each feature. The encoding order for education was defined as {unknown, primary, secondary, tertiary}, while for previous outcome it was {unknown, failure, other, success}.

The `duration` feature was excluded from training because its value strongly correlates with the target: when duration is 0, the target is always negative. However, since call duration is unknown prior to making contact, this feature is not usable

for real-time prediction and was therefore dropped to avoid data leakage.

C. Data Analysis

Exploratory analysis was conducted to inspect feature distributions and identify potential outliers. Kernel density estimates (KDEs) were generated for key numeric features. For example, the KDE for the `age` feature is shown in Figure 1. No data points were identified as outliers that warranted removal or replacement, as their distributions did not indicate anomalies likely caused by data entry errors or inconsistencies.

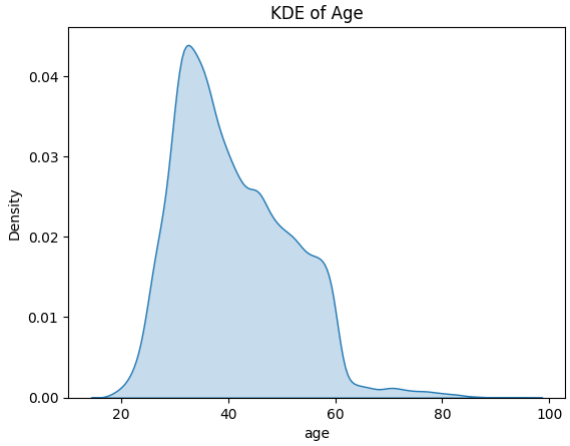

Fig. 1. KDE showing the distribution of the age feature.

To assess class separability, Linear Discriminant Analysis (LDA) was applied. The resulting projection, shown in Figure 2, indicates that the classes are not linearly separable. This finding suggests that models capable of capturing non-linear relationships are likely to perform better on this dataset.

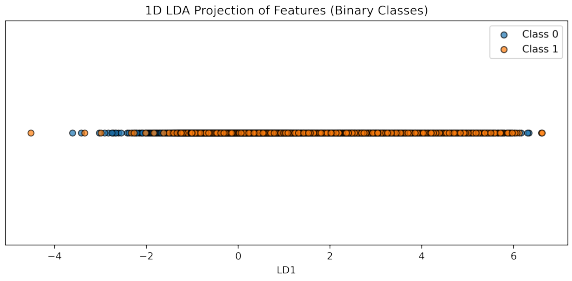

Fig. 2. Linear Discriminant Analysis Plot

Additionally, the dataset was found to exhibit a high degree of class imbalance: of the total 45,211 instances, 39,922 (88.3%) belong to the negative class and only 5,289 (11.7%) to the positive class. This imbalance poses a challenge for standard classification models, which may default to majority class predictions and achieve deceptively high accuracy. For this reason, accuracy was not used as the primary optimization metric. Instead, metrics such as F1-score, area under the ROC curve (AUC), precision, and recall were prioritized to provide a more balanced evaluation of model performance — especially with respect to identifying the minority class.

D. Experimental Design

To address the classification task of predicting whether a client will subscribe to a term deposit, four distinct models were implemented and evaluated. A logistic regression model was first trained and served as a baseline due to its simplicity and interpretability. This baseline provided a reference point for comparing more complex models.

Subsequently, three additional models were explored: a decision tree, a random forest, and an artificial neural network. These models were selected to represent a range of learning paradigms—from interpretable tree-based methods to more expressive, non-linear models. For each model, hyper-parameter tuning was conducted using grid search with cross-validation. The results reported in Section III reflect the best-performing configuration for each model type based on validation metrics.

1) Logistic Regression: We use `LogisticRegression` from scikit-learn. It was trained with grid search over the hyper-parameters (Table II) on 5 fold cross validation. AUC was optimized.

TABLE II
HYPER-PARAMETER GRID FOR LOGISTIC REGRESSION

Hyper-parameter	Values
`C (Reg Strength)`	`[0.01, 0.1, 1, 10, 20]`
`penalty`	`["l1", "l2"]`
`solver`	`["liblinear"]`
`class_weight`	`[None, "balanced"]`

The logistic regression model with the best results that are reported in Section III had the following values for the hyper-parameters:

- `C = 1`
- `class_weight = "balanced"`
- `penalty = "l1"`
- `solver = "liblinear"`

2) Decision Tree: We use `DecisionTreeClassifier` from scikit-learn. When training the decision tree, a grid search was used in order to train the hyper-parameters. The hyper-parameter grid that was searched over included the values in Table III. The hyper-parameter tuning was optimised to choose the model with the highest AUC.

TABLE III
HYPER-PARAMETER GRID FOR THE DECISION TREE

Hyper-parameter	Values
`max_depth`	`[3, 5, 10, 20, None]`
`min_samples_split`	`[2, 5, 10]`
`min_samples_leaf`	`[1, 2, 5, 10]`
`criterion`	`["gini", "entropy"]`

Figure 3 shows the structure of a decision tree with the maximum depth set to 3. The decision tree with the best

results that are reported in Section III had the following values for the hyper-parameters:

- `max_depth = 10`
- `min_samples_split = 2`
- `min_samples_leaf = 10`
- `criterion = "entropy"`

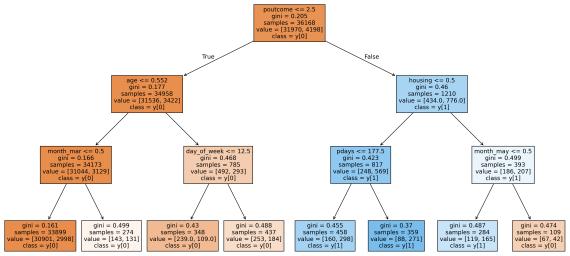

Fig. 3. Structure of a decision tree with a maximum depth of 3.

3) Random Forest: We use `RandomForestClassifier` from scikit-learn. A grid search was also used to train the hyper-parameters of the random forest. The hyper-parameter grid that was searched over is shown in Table IV. The hyper-parameter tuning was optimised to choose the model with the highest AUC. The random forest with the best results that are reported in Section III had the following values for the hyper-parameters:

- `n_estimators = 200`
- `max_depth = 20`
- `min_samples_split = 10`
- `min_samples_leaf = 2`
- `criterion = "entropy"`

TABLE IV
HYPER-PARAMETER GRID FOR THE RANDOM FOREST

Hyper-parameter	Values
n_estimators	[100, 200]
max_depth	[3, 5, 10, 20, None]
min_samples_split	[2, 5, 10]
min_samples_leaf	[1, 2, 5, 10]
criterion	["gini", "entropy"]

4) Artificial Neural Network: PyTorch served as the framework for constructing the ANN, with hyper-parameter optimization conducted using scikit-learn's grid search. Table V outlines the range of hyper-parameters evaluated. Unlike the other models, the hyper-parameter tuning using the two libraries was unable to optimize for AUC and instead focused on optimizing for F1-score. However, the optimized model still achieved a high AUC, as reflected in Table VI.

Figure 4 depicts the structure of the ANN optimised. It included three hidden layers, with 36, 32, and 16 nodes respectively. The hyper-parameter values for the ANN that yielded the best results, presented in Section III, are as follows:

- `hidden_sizes = [36, 32, 16]`
- `learning_rate = 0.001`
- `epochs = 20`

TABLE V
HYPER-PARAMETER GRID FOR THE ANN

Hyper-parameter	Values
hidden_sizes	[64, 32], [36, 32, 16]
learning_rate	[0.01, 0.001]
epochs	[10, 20]
batch_size	[32, 64]
optimizer_name	["adam", "sgd"]
weight_decay	[1e-4, 1e-3]

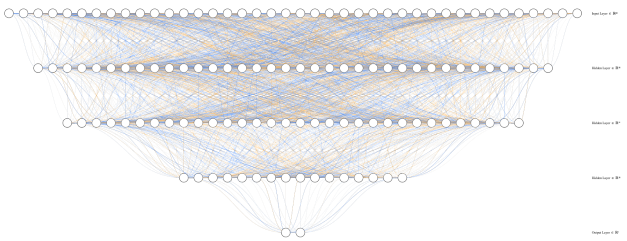

Fig. 4. Structure of the ANN with 3 hidden layers.

- `batch_size = 32`
- `optimizer_name = "adam"`
- `weight_decay = 1e-3`

E. Training Setup

Before model training, the dataset was partitioned into training, validation, and test subsets. The test set comprised 20% of the total data, the validation set 10%, and the remaining 70% formed the training set. Data preprocessing steps were then applied separately to each subset.

This ordering — splitting prior to preprocessing — was chosen to avoid data leakage and ensure that information from the validation and test sets did not influence the training process. This is also mentioned in Section IV.

III. RESULTS

The code used to run the experiments is available on GitHub[1]. The logistic regression model achieved an overall accuracy of 78% and an AUC of 0.77, indicating reasonably good discriminatory power. While the model's precision was relatively low at 0.29, it achieved a higher recall of 0.61, suggesting that it was more effective at identifying positive responders than avoiding false positives. The F1-score of 0.40 reflects a modest balance between precision and recall. These results suggest that while logistic regression offers a useful baseline, it may benefit from further tuning or more complex models to improve positive class identification.

Both the decision tree and random forest models achieved an overall accuracy similar to that of the artificial neural network. Their precision scores were relatively good; however, they struggled in terms of recall. This shortfall is important, especially in the context of targeting bank clients who are likely to subscribe. The random forest model achieved an AUC of 0.80, indicating solid overall discriminatory performance.

[1]GitHub Repository: ACML Project

However, the limited recall suggests that these models were less effective at identifying the positive class, and their performance requires further investigation before drawing strong conclusions.

The ANN model achieved an accuracy of 89%, indicating that it correctly predicted the outcome for the majority of clients. However, this high accuracy is likely influenced by class imbalance, as most clients did not subscribe to the term deposit. The model's precision of 0.63 implies that, when it predicts a client will subscribe, it is correct 63% of the time. In contrast, the recall is relatively low at 0.28, meaning the model successfully identified only 28% of the actual subscribers. This highlights a significant limitation in detecting true positives. The F1-score of 0.39 reflects the imbalance between precision and recall, suggesting the model's overall effectiveness in identifying likely subscribers is limited. Despite this, the Area Under the ROC Curve (AUC) value of 0.79 indicates that the model has a good ability to distinguish between clients who will and will not subscribe. This suggests that performance could be improved through threshold tuning or by addressing the class imbalance present in the data.

TABLE VI
METRICS FOR THE DIFFERENT MODELS

Model	Precision	Recall	Accuracy	F1	AUC
Logistic Regression	0.29	0.61	0.78	0.40	0.77
Decision Tree	0.59	0.16	0.89	0.26	0.75
Random Forest	0.72	0.16	0.89	0.26	0.80
ANN	0.63	0.28	0.89	0.39	0.79

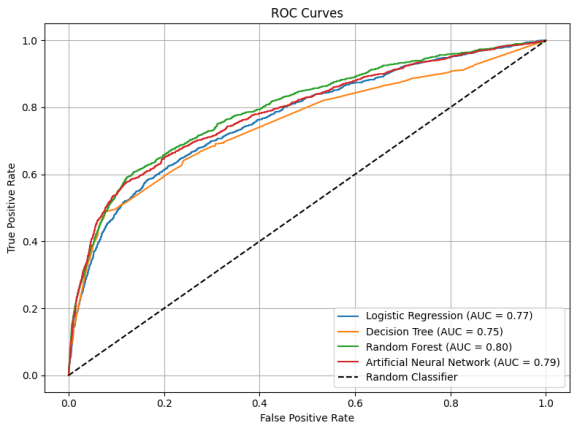

Fig. 5. ROC curves of the best performing models of each model used.

IV. DISCUSSION

One of the challenges faced when trying to model the Bank Marketing dataset was that of high class imbalance. The distribution of positive to negative labels for the dataset is shown in Figure 6. This imbalance meant that it was easy to achieve a model with an accuracy of at least 88.3% but that achieving a high recall or precision would be more challenging. This is why AUC was used as a metric to optimise for when performing hyper-parameter tuning. By optimising the AUC we hoped to learn models that were well rounded.

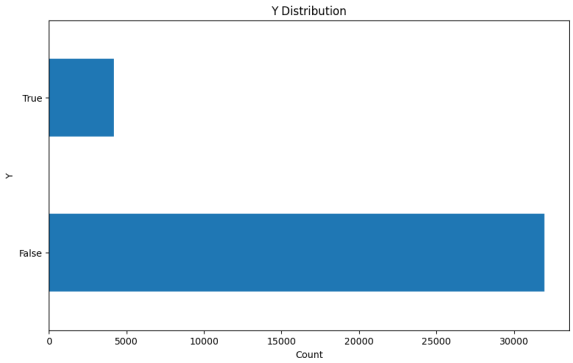

Fig. 6. Highly imbalanced distribution of class labels.

Another method used to address the significant class imbalance was the Synthetic Minority Oversampling Technique (SMOTE). SMOTE works by generating synthetic examples of the minority class to balance the class distribution. The resulting distribution of the target variable after applying SMOTE is shown in Figure 7.

Surprisingly, applying SMOTE and retraining the random forest resulted in decreased performance on the test set. This outcome underscores the complexity of handling imbalanced data, as oversampling can sometimes lead to overfitting or reduced generalisation, particularly when the synthetic samples do not reflect the true distribution of the minority class in unseen data.

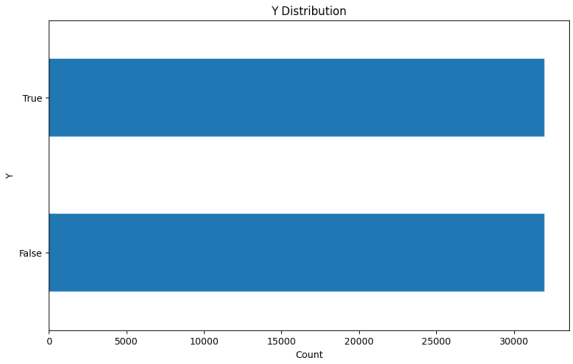

Fig. 7. Balanced distribution of class labels after SMOTE.

In the initial experiments, particularly with the ANN model—the model produced perfect scores across all performance metrics, which was later identified as a result of data leakage. Specifically, the issue arose because min-max scaling was applied to the entire dataset before the train-test split. This allowed the model to indirectly access information from the test set during training, resulting in overly optimistic performance. To correct this and obtain realistic evaluation results, the data preprocessing pipeline was revised. Min-max scaling was applied *after* splitting the data into training and testing sets, ensuring that the test data remained completely

unseen during the training process. This adjustment led to more reasonable and reliable model performance. Further hyper-parameter tuning was conducted on the revised pipeline to enhance the model's effectiveness. Among the continuous variables, features such as `age`, `balance`, and `campaign` were found to be particularly significant in predicting client subscription behaviour.

REFERENCES

[1] S. Moro, P. Rita, and P. Cortez, "Bank marketing," UCI Machine Learning Repository, 2014, DOI: https://doi.org/10.24432/C5K306.