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

Effective targeting is crucial for banks conducting direct marketing campaigns due to the inherent costs associated with outreach methods such as SMS, email, or phone calls. This project leverages the Bank Marketing dataset from the UCI Machine Learning Repository, which includes client demographic details and historical interactions with a banking institution. Our primary objective is to build a predictive classification model that accurately identifies clients who are most likely to accept a term deposit offer when directly marketed to. To achieve this, we explore and evaluate multiple classification techniques, spanning linear and non-linear models, aiming to enhance targeting efficiency, maximize client response rates, and ultimately reduce overall marketing expenditures.

II. Method

A. Dataset Description

The dataset used is related to a direct marketing campaign of a Portuguese bank [1]. The marketing campaign is based on phone calls and the classification goal is to predict whether a client will subscribe to a term deposit. This makes the task a binary classification task. The features that the dataset is made up of are shown in Table I.

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

Feature	Type	Description
Age	Integer	Age
Job	Categorical	Type of job
Marital	Categorical	Marital status
Education	Categorical	Education level
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 were originally either a string value of ‘yes’ or ‘no’. These features were converted into binary features where ‘yes’ values were mapped to 1 and ‘no’ values were mapped to 0.

The age, balance, and campaign features were all normalised using max-min normalisation. This allows all features to have a range of $[0, 1]$. This normalisation should help to decrease training time as the model will not need to learn how to scale the features based on their relative ranges.

The job, marital, contact, and month categorical features were all one-hot encoded to allow for these variables to be used. Even though decision trees do allow for categorical variables, the scikit-learn implementation is currently unable to handle categorical variables. These features were not ordinally encoded because they do not have an inherent order and encoding them ordinally might cause the model to believe that this type of order was present.

The education, and previous outcome categorical features were ordinally encoded. The order of education was specified as [‘unknown’, ‘primary’, ‘secondary’, ‘tertiary’] and the order of the previous outcome was specified as [‘unknown’, ‘failure’, ‘other’, ‘success’]. These orders represent a natural progression in the features.

The duration feature was dropped because when the duration is 0, then the target value is always false. The problem with this is that the call duration is not known before a call is made and therefore this feature cannot be used to help make predictions before calls were made.

C. Data Analysis

When data analysis was performed, the distributions of the features were inspected to look for outliers that could be replaced or removed depending on what might be causing the outlier. An example of a KDE obtained for the age feature is shown in Figure 1. Based on this analysis there were no data points that needed to be replaced or removed.

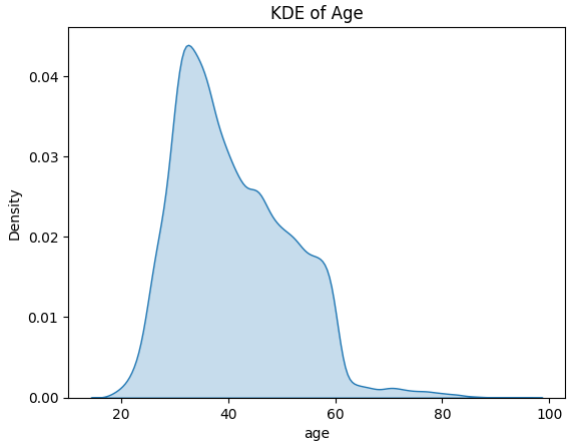

Fig. 1. KDE showing the distribution of the age feature.

It was also noted when performing linear discriminant analysis as shown in Figure 2 that the data is not linearly separable. This leads to the need for a model that can handle non-linearities.

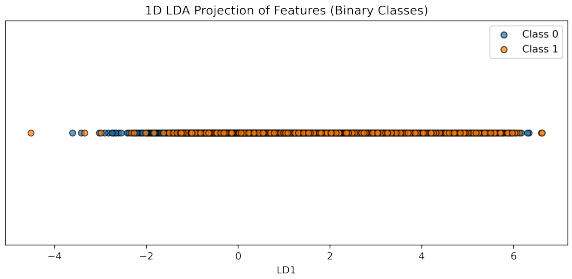

Fig. 2. Linear Discriminant Analysis Plot

It was also noted that the data had a high class imbalance. There were 39922 data points with a negative label and 5289 data points with a positive label. This means that only 11.7% of the data had positive labels. This class imbalance was handled differently in each of the models and the approach used will be mentioned in Section II-D. For this reason, a model with an accuracy of less than 88.3% could have been beaten by a model that simply predicts negative labels for any input. This is why accuracy was not used as the metric to optimise for and instead metrics such as F1, AUC, recall, and precision were used to give a better understanding of the performance of the models.

D. Experimental Design

In order to achieve the objective of predicting whether a client of the bank will subscribe to a term deposit, four different models were used. The first model that the data was trained on was a basic logistic regression model. This model served as a baseline and was used to compare the performance of the other models.

The other models that were used include a decision tree, a random forest and an artificial neural network. For each different model, hyper-parameter tuning was performed and the results of the best performing model for each class are reported in Section III.

1) Logistic Regression: We use `LogisticRegression` from `sklearn`. It was trained with grid search over the hyperparameters (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"]`

2) Decision Tree: When training the decision tree, a grid search was used in order to train the hyper-parameters. The scikit-learn implementation of a decision tree was used and 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"`

3) Random Forest: 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:

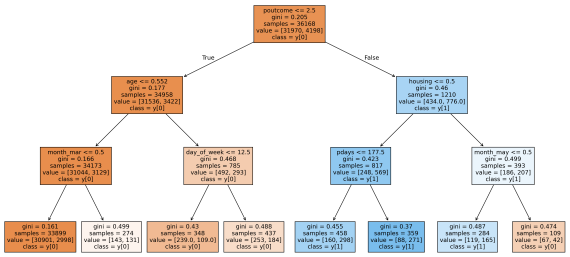

Fig. 3. Structure of a decision tree with a maximum depth of 3.

- `n_estimators = 200`
- `max_depth = 20`
- `min_samples_split = 2`
- `min_samples_leaf = 5`
- `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 sklearn'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 the table.

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]

Figure 4 depicts the structure of an ANN optimised to 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`
- `batch_size = 32`
- `optimizer_name = "adam"`
- `weight_decay = 1e-3`

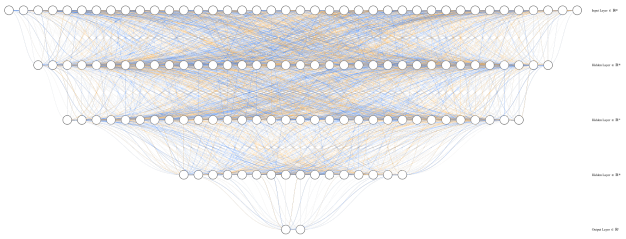

Fig. 4. Structure of the ANN with 3 hidden layers.

E. Training Setup

When preparing the data for training after performing data preprocessing, the data was split into training and testing sets. The test set was set to be 20% of the dataset.

III. RESULTS

The code used to run the experiments is available on GitHub[1].

TABLE VI
METRICS FOR THE DIFFERENT MODELS

Model	Precision	Recall	Accuracy	F1	AUC
Logistic Regression	0.29	0.61	0.78	0.40	0.76
Decision Tree	0.58	0.25	0.89	0.35	0.75
Random Forest	0.74	0.19	0.89	0.31	0.80
ANN	0.63	0.28	0.89	0.39	0.79

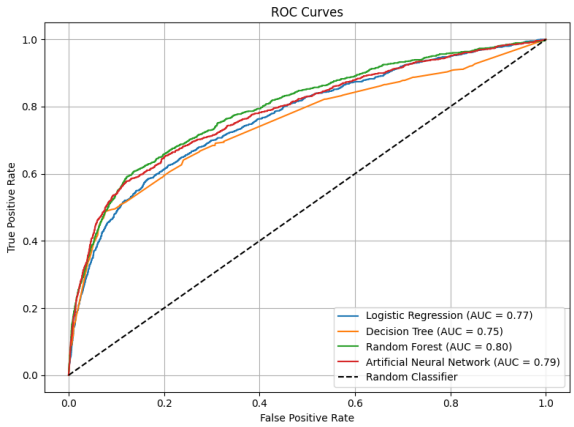

Fig. 5. ROC curves of the best performing models of each model used.

The logistic regression model achieved an overall accuracy of 78% and an AUC of 0.76, 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,

[1]GitHub Repository: ACML Project

it may benefit from further tuning or more complex models to improve positive class identification.

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.

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.

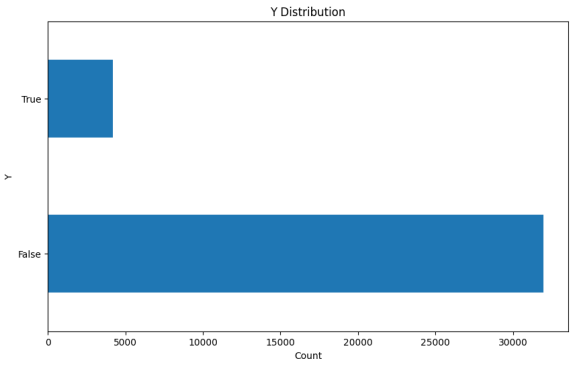

Fig. 6. Highly imbalanced distribution of class labels.

Another method that was used to address the high class imbalance was Synthetic Minority Oversampling Technique, also known as SMOTE. This method effectively creates synthetic data points that balance the classes. The distribution of the target variable after SMOTE was applied can be seen in Figure 7.

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

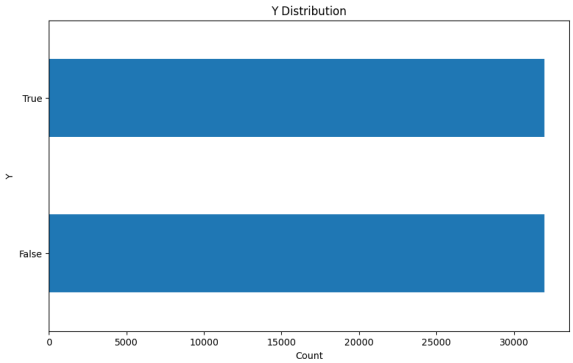

Fig. 7. Balanced distribution of class labels after SMOTE.

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.