Bank Marketing Campaign Predictive Modelling

Isaac Dada

School of Computing, Engineering & Digital Tech, Teesside University, United Kingdom.
A0157300.live.tees.ac.uk

***Abstract*— With the growing need for banks and financial institutions to recognize and understand their customers' desires and behaviors, hence, the goal of implementing a proper marketing strategy that would guarantee maximum acceptance of the institution's services has become very important. The research in this report is focused on evaluating bank marketing campaigns targeted towards customers' purchase of bank's term deposit, specifically to determine if customers will agree to the scheme or not. The dataset used in this research is available on UCI Machine Learning Repository at the following page: archive.ics.uci.edu/ml/datasets/Bank+Marketing. Due to the large class imbalance in the target of the dataset used for this research; it contains a higher percentage of *"No" – majority class* as compared to *"Yes" – minority class,* tackling this issue becomes an imperative sub-task which was also covered in the experiment using KMeans for under-sampling of the majority class to select n-samples of clusters found within the majority class. In this experiment, three machine learning models were implemented, and their performance was evaluated against one another. Gradient boosting had the highest performance of all the algorithms used in this study, with an accuracy of 85.7 percent, followed by Random Forest with an accuracy of 83.6 percent, and Decision Tree with 75.8 percent.**

Keywords— Bank Marketing, Undersampling, Decision Tree, Gradient-boosting models, Predicting Term Deposit Subscriptions, Random forest.

INTRODUCTION

Bank deposits are their primary source of income. Many banks have a variety of accounts to persuade consumers to invest their money. Hence, efficient marketing can help a bank expand the number of term deposit subscribers. To reach their clients, banks should have a successful marketing campaign strategy. With the improvements in technology, banks can use machine learning with Big Data to collect and analyze customer data. These data can be used to identify the likelihood of customers subscribing to term deposits. Thus, the interaction between the bank officials and customers becomes more efficient in increasing the number of customers who are willing to subscribe to term deposits (Oni, 2020).

Machine learning predictive modelling techniques such as classification algorithms is important as it would not only assist banks in reaching a larger number of customers but also in selling their goods to other interested individuals in the general population. As a result, banks will raise the number of term deposit subscribers by targeting a certain category of customers. Banks need to start implementing the newest technologies in big data, data mining and machine learning in all public services to be more productive and generate more revenues. In addition, several frameworks that support machine learning with big data play an essential role in allowing the deploying of these methods easily within the infrastructure of the banks and financial institutions (Phan et al. 2019).

Contributions of This Paper and Related Work

Recent research has shed light on various issues on implementing predictive modeling for bank marketing campaigns. Hung et al. (2019) discussed in ***Term Deposit Subscription Prediction Using Spark MLlib*** methods for predicting customer's response to term deposit subscriptions, however, "oversampling" was used to handle the class imbalance of the banking marketing dataset. Secondly, Jamiu O., (2020) discussed in ***Exploratory Analysis of Bank Marketing Campaign Using Machine Learning: Logistic Regression, Support Vector Machine, and K-Nearest Neighbor***; factors that increase the customer's decision to subscribe to a term deposit, however, the methodology focused on understanding the weight of each feature on the target outcome. Lastly, Asare-Frempong and Jayabalan (2017) research discussed how banks would benefit from direct marketing when concentrating on consumers that have shown interest in their services. In this scenario, the researchers used different classifiers such as Logic Regression, Decision Tree, Random Forest, and Multiplayer Perception Neural Network to determine the determinants of consumer reaction to a direct marketing campaign (MLPNN) with random forest had the highest accuracy of 87%. However, the class imbalance was tackled using the 'Spread-subsample' method.

The main contribution of this paper is to use a combination of machine learning classification algorithms - Decision Tree, Random Forest, and Gradient-Boosted Tree to create models that can predict term deposit subscriptions. Also, this paper utilizes an under-sampling methodology to handle the class imbalance as compared to oversampling and spread subsample method from previous.

Research question

1. What is the most important bank marketing variable that influences a customer's decision to open a term deposit?
2. What classifier gives the highest accuracy in predicting customer's likelihood of subscription using the under-sampled data set.

METHODOLOGY

1. Technical discussion overview on classification algorithm choice – Decision Tree, Random Forest, Gradient Boosting,
2. Exploratory Data Analysis and Data Pre-processing – Class imbalance handling,
3. Feature selection – Label encoding and Principal component analysis (PCA),
4. Experiment – Modelling and hyperparameter tuning.

Technical Overview on the Classification Algorithm

Firstly, the decision tree is constructed by dividing training data into smaller and smaller samples repetitively. At each node, the algorithm tests the conditions and splits the data according to the conditional statement's result. For this experiment, the decision tree aimed at minimizing the measure of impurity of the dataset. This is also referred to as entropy which is the measurement of uncertainty or impurity. From the experiment, targeting minimizing entropy produced a more accurate tree as compared to maximizing the information gain of features.

Secondly, the random forest algorithm implemented combines the predictions of several decision trees of various depths. The final prediction is calculated by combining the predictions of each decision tree in the forest. Random forests work well on datasets with high dimensions and faster to train than decision trees. Also, outliers are handled by Random Forest by binning them. Each decision tree has a high variance, but it has a low bias. For a random forest, when the variance of all the trees is averaged, we get a low bias and modest variance.

Lastly, Gradient boosting is a technique for building trees one at a time, with each new tree helping in the correction of errors made by the previous tree. These trees are grown sequentially, meaning that each tree is grown using knowledge from previous trees. This model built in the experiment aimed to improve the performance of the random forest model by iterating various learning rates over a fixed n-estimator.

DATA PRE-PROCESSING

The imported modules and packages which aided the experiment can be accessed in the link https://github.com/idada29/Bank-Marketing-Predictive-Modelling/blob/main/CodeCommit.ipynb. Also, the data set used can be accessed here: archive.ics.uci.edu/ml/datasets/Bank+Marketing. The dataset has 49732 observations and 17 features. The overview of the dataset is shown below:

```
from pandas_profiling import ProfileReport
banking_data.profile_report()
```

Overview | Warnings 4 | Reproduction

Dataset statistics

Number of variables	17
Number of observations	49732
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	4521
Duplicate rows (%)	9.1%
Total size in memory	6.5 MiB
Average record size in memory	136.0 B

Variable types

Numeric	7
Categorical	6
Boolean	4

The overview shows, there are no missing cells however pointed out 4521 duplicated rows which is about 9.1% of the data set. However, further exploration of the data showed that these were not actual entire duplication of rows but rather rows with similar content in about 5-7 columns.

Another overview of the features generated by the .profile_report() function is shown in the jupyter notebook.
The features and datatype of the dataset are:

```
banking_data.info()
```

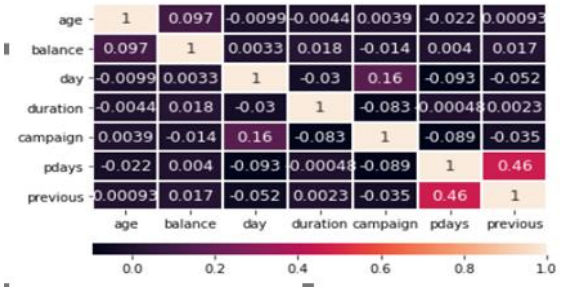
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 49732 entries, 0 to 49731
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   age         49732 non-null  int64
 1   job         49732 non-null  object
 2   marital     49732 non-null  object
 3   education   49732 non-null  object
 4   default     49732 non-null  object
 5   balance     49732 non-null  int64
 6   housing     49732 non-null  object
 7   loan        49732 non-null  object
 8   contact     49732 non-null  object
 9   day         49732 non-null  int64
 10  month       49732 non-null  object
 11  duration    49732 non-null  int64
 12  campaign    49732 non-null  int64
 13  pdays       49732 non-null  int64
 14  previous    49732 non-null  int64
 15  poutcome    49732 non-null  object
 16  y           49732 non-null  object
```

The dataset contained no empty cells:

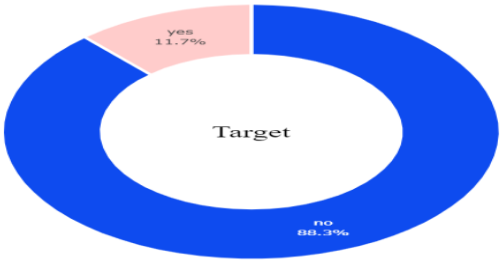
```
# Check for missing values in the training dataset,
# output indicates that our dataset has no null values

print(banking_data.isnull().values.any())
print(" ")
print(banking_data.isnull().sum())
```

```
False
```

Using Pearson's method, a pairwise correlation of the features shows a high positive correlation between the previous contact with customers and Pdays – indicates how long it has been since the customers were last called.

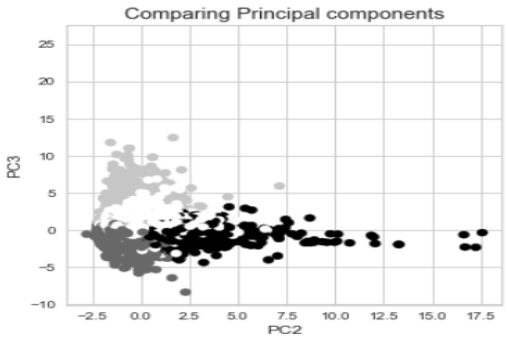


Lastly, under pre-processing the class imbalance need to be handled before the modeling begins. The data set has a skewed distribution in the outcome ("y"), hence a need for balancing. The methodology used was under-sampling of the majority class. Using KNN to determine the number of clusters within the data set, then selecting randomized n-samples within each cluster. The output of this procedure would equate to the number of observations for the binary class.



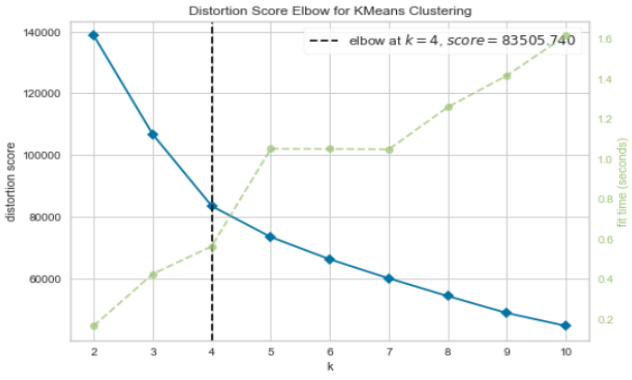
Procedures to carry out:

1. Separate the numeric and categorical features, next create a Pandas dummy data for the categorical features.
2. Scale the values of the numerical features using standardScaler (), convert the np.array of scaled values into a data frame, and concatenate with the categorical feature dummy above.
3. Due to the increased number of columns (51 columns) in the new concatenation, PCA is needed to reduce the dimensionality which would enhance better interpretational and decrease any information loss. It accomplished this by generating new uncorrelated variables that sequentially maximize variance.



Refer to appendices for codes, section 3 in the attached link:

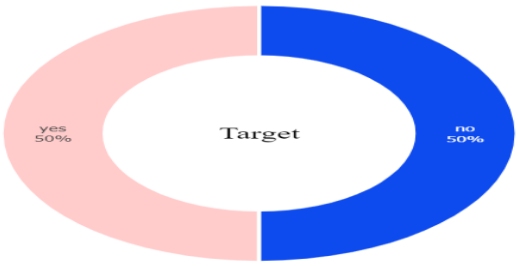
Using KElbowVisualizer to visualize the cluster found:



4. Kmeans2 was used to calculate the centroids of the PCA transformed data, these centroids were used to fit a model for predicting clusters.

	PC1	PC2	PC3	PC4	cluster
0	-0.439873	-0.349162	0.075978	-0.067971	3
1	3.617713	-0.085725	0.171214	0.280383	2
2	2.677662	-0.642673	0.382832	0.218129	2
3	-0.360358	-0.787237	-0.734811	-0.675056	3
4	-0.288691	0.867277	-1.360119	-1.534676	1

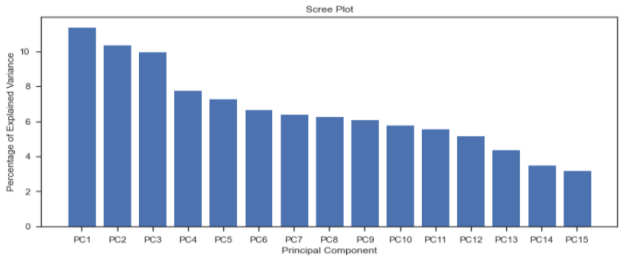
5. At this point, an array of an index that selects the majority target label that coincides with the iteration of values within the different 4 clusters was created. Also, after iterating through the previous steps, 1452 samples of 4 clusters of the majority class have been selected which equals 5807 observations. The resampled data merge both minority class observations and the newly sampled majority class to give a balanced data set.

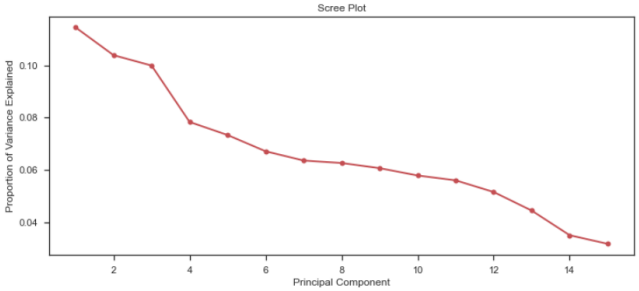


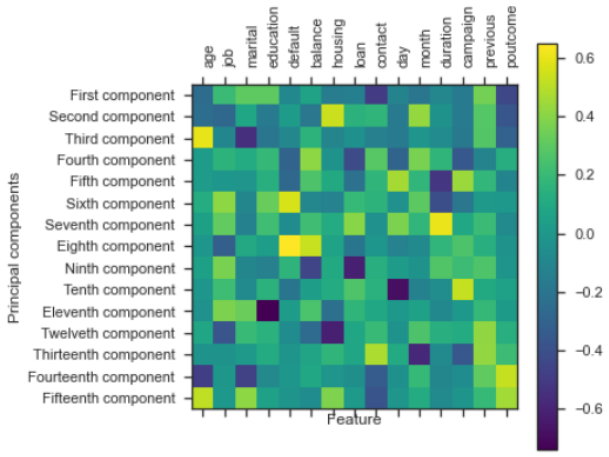
For other codes for visualizations see the link in the appendices

Feature selection – Label encoding and Principal component analysis (PCA)

Before building models, the categorical features were transformed using LabelEncoder() to numeric labels. The reason for selecting label encoder as compared to one-hot encoding is because the features in most columns are either ordinal or just binary which prevents the possibility of the model wrongly capturing their relationship in order. Also, scaling was required to normalize the values of all features. Finally, to understand the percentage of variance explained by each component, PCA was applied.







EXPERIMENT–MODELLING AND HYPERPARAMETER TUNING

Decision Tree:

Due to the need for iterative hyperparameter tuning to select the best possible hyperparameters which would yield the highest accuracy of prediction, GridSearch Cross-validation method was used to find the most appropriate hyperparameters which would yield the highest mean score in the accuracy of prediction. However, the main challenge with this was the time it takes for this process to compute. It took approximately 2 minutes 55 seconds for the complete computing of the grid. However, it aids in fitting your model to your training set by looping around predefined hyperparameters, you can choose the best parameters from the hyperparameters mentioned.

This gives a result showing the best parameters for the decision tree model using cv = 5 folds (Note: cv = 10, was too computationally expensive and takes a lot more time to compute completely):

Mean score: 0.7313677562473002 {'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 18, 'random_state': 91, 'splitter': 'random'}

However, to have a broader overview of how this conclusion was derived, sklearn cv_results_ can make available different iteration made available by the classifier.

Consequent to sorting by the mean_test_score, the top 10 best iterations are shown below: (Note: it indicated entropy as the best criterion, splitter as random, max_depth as 4, and random _state as 91)

	param_criterion	param_splitter	param_max_depth	param_min_samples_leaf	param_random_state	mean_test_score
222	entropy	random	4	19	91	0.731368
202	entropy	random	4	18	91	0.731368
162	entropy	random	4	16	91	0.730593
182	entropy	random	4	17	91	0.730593
142	entropy	random	4	15	91	0.728614
122	entropy	random	4	14	91	0.728614
3434	gini	random	14	17	97	0.725344
2742	gini	random	8	19	91	0.724658
3660	gini	random	16	17	90	0.723280
3780	gini	random	17	17	90	0.722419

At this point, from sklearn.model_selection import the train_ test_split would be used to split the dataset into test and train.

Training the decision tree model based on the best hyperparameter selection yielded a 75.84% accuracy on the test set which is about 2.71% better than the mean score derived from the GridSearch cross-validation.

```
Predicted values:
['yes' 'no' 'no' ... 'no' 'no' 'no']
Confusion Matrix:  [[1131  599]
 [ 243 1513]]
Accuracy :  75.84624211130236
Report :               precision    recall  f1-score   support

          no       0.82      0.65      0.73      1730
         yes       0.72      0.86      0.78      1756

    accuracy                           0.76      3486
   macro avg       0.77      0.76      0.76      3486
weighted avg       0.77      0.76      0.76      3486
```

Accuracy on training set: 0.767
Accuracy on test set: 0.758

The tree can be visualized using sklearn.tree export_graphviz module. The ROC AUC score was used to rate predictions rather than their values; it calculates prediction accuracy regardless of the classification threshold used.

```
ROC_AUC is 0.8279186472145047 and accuracy rate is 0.7584624211130235
```

ROC score is usually a range of 0-1, if closer to 1 then the decision tree model has a rate of 82% correct prediction.

The final step was to calculate the importance of the features on the decision tree models:

```
Feature importances:
      Features  Feature Importance
11    duration            0.767617
6      housing            0.152463
8      contact            0.066809
14    poutcome            0.009426
1          job            0.001884
0          age            0.000972
10       month            0.000643
9          day            0.000187
2      marital            0.000000
3    education            0.000000
4      default            0.000000
5      balance            0.000000
7         loan            0.000000
12    campaign            0.000000
13    previous            0.000000
```

Random Forest:

For cross-validation and hyper tuning of parameters for the random forest, RandomizedSearchCV was used to reduced the large computing time taken by gridsearchcv to run iterations of parameters (processing time of over 1 hour).
The following is the result of the 5 fold random search cross-validation.

```
RandomForestClassifier(n_estimators=272,max_features=7,bootstrap=False,max_depth=14)
RandomForestClassifier(n_estimators=319,max_features=9,bootstrap=False,max_depth=10)
RandomForestClassifier(n_estimators=339,max_features=8,bootstrap=False,max_depth=3)
```

However, for each time the random search cv was run a new selection is produced as hyperparameters are randomly selected instead of complete iterating as grid search does.

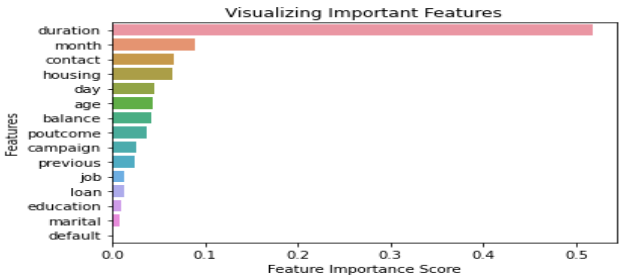
Out of the three iterations, the second combination of hyperparameter avoided overfitting while maintaining an accuracy of 83.7%

```
Accuracy on training set: 0.922
Accuracy on test set: 0.837
```

The results from classifier 1 and classifier 3 with other parameters are (1) fitting training set more and (3) has low accuracy.

Accuracy on training set: 0.987 Accuracy on training set: 0.775
Accuracy on test set: 0.853 Accuracy on test set: 0.765

The graph below shows the feature importances for the chosen RandomForest classifier:



Gradient Boosting Classifier:

Random search cross-validation was used to iterate over various parameters for the best mean score. A fixed n-estimator was set at 250 and criterion set as "friedman_mse" because it produces a better estimation than mean square error while iterating over different learning rates and max_depth varying between 0.3-0.5 and a range of 2-5 respectively.

	param_criterion	param_learning_rate	param_max_depth	param_n_estimators	mean_test_score
1	friedman_mse	0.3	3	250	0.783190
2	friedman_mse	0.3	4	250	0.781383
5	friedman_mse	0.4	4	250	0.779489
4	friedman_mse	0.4	3	250	0.778887
0	friedman_mse	0.3	2	250	0.772692
3	friedman_mse	0.4	2	250	0.770453

The first set of hyperparameter when used for prediction produced a more generalized model compared to the other.

```
Accuracy on training set: 0.937
Accuracy on test set: 0.857
```

As the hyperparameters are lowered, the training set's accuracy decreases while the test set's accuracy increases exponentially.

RESULTS AND DISCUSSION

As a classification problem, the metrics used for evaluating and record the results of these models were

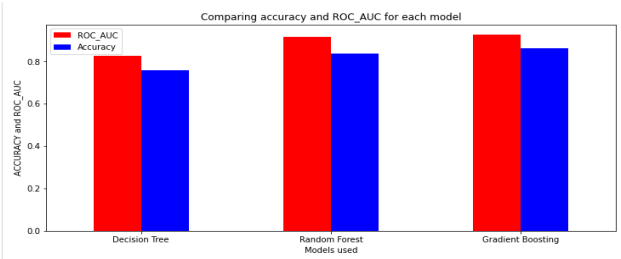
1. ROC_AUC score
2. Confusion matrix
3. Accuracy

The final results and comparison of the ROC_AUC score and accuracy of the three models are shown below:

Models	ROC_AUC	Accuracy
Gradient Boosting	0.925743	0.856569
Random Forest	0.915177	0.836202
Decision Tree	0.827919	0.758462

The computational speed varied depending on the number of hyperparameters and the type of cross-validation used. The results are shown as:

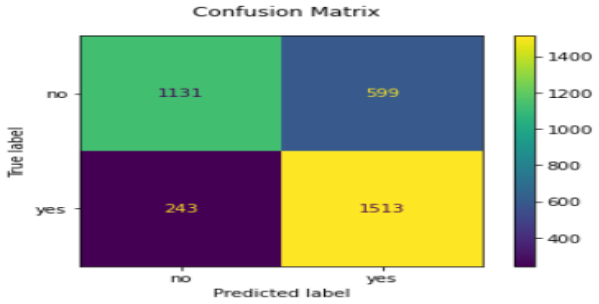
Models	Cross-Validation Used	Computing Time
Gradient Boosting	GridSearch CV	3 minutes
Decision tree	GridSearch CV	2 minutes 35 seconds
Random Forest	RandomizedSearch CV	1 minute 50 seconds



Confusion Matrix

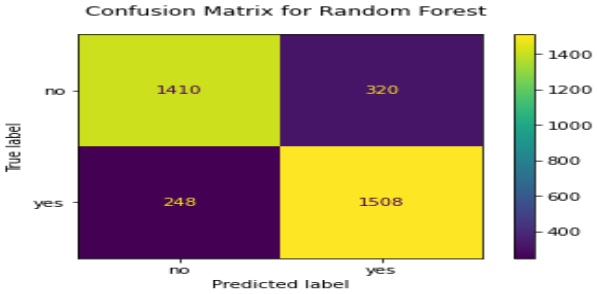
Decision Tree:

Although the performance of the decision tree ranked lower than the other two models, however, this model had the advantage of faster processing and training time. Also, this model had the second-highest False 'Yes' lower than the random forest.



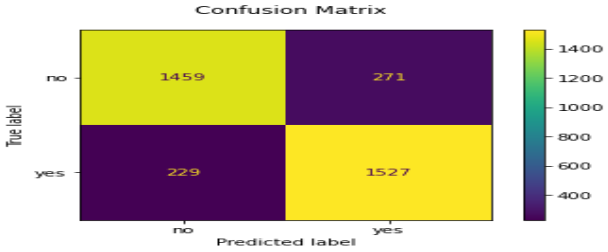
Random Forest:

As I iterated over an increasing amount of max_depth values, the nodes gradually become purer until the maximized value was achieved with overfitting.



Gradient Boosting

This model allowed me to identify the previous error, and the error is minimized dramatically over time by replacing one bad learner with the next learner. Thus, reducing the variance (the degree of change the predicted estimate would have experienced if the training dataset was changed) but increase the bias.



Also, a Neural network classification model based on the unsampled data was implemented focused on prediction disparities based on standardization. However, this would be deeply studied in future work. Results are shown below for models built without standardization and with standardization from left to right respectively.

```
# Building my neural network with the following steps:
neural_clf = MLPClassifier(random_state=42,max_iter=1000)
neural_clf.fit(independent_train_scaled, target_train)
```

```
Accuracy on training set: 0.74  Accuracy on training set: 0.907
Accuracy on test set: 0.74      Accuracy on test set: 0.808
```

LIMITATIONS OF THE STUDY

The study's biggest limitation is the lack of a good processor. External cloud-based applications, which may have addressed this, were not used for the sake of reproducibility of results and experiments.

CONCLUSION

In conclusion, the under-sampling approach was used to deal with the issue of imbalanced data, and three classifiers were used to estimate subscriptions for a bank's term deposit with an accuracy of about 86 percent. The study's findings could help banks in predicting a customer's inclination for term deposit subscriptions. For a future review of this work, I would attempt to compute broader cross-validation of hyperparameters using a cloud-based application to derive performance and to improve computing speed for these processes.

REFERENCES

APPENDICES

https://a0157300.scedt.tees.ac.uk/ML%20ICA/trial.html