A Comparison of Machine Learning Models for Loan Approval Classification

Word Count: 4,567

Abstract

Loan approval automation is an important subject that can have significant financial, reputational and regulatory implications when done incorrectly. This study will prove that key predictor variables can be used to build a machine learning model that can improve the accuracy compared to a baseline model when classifying loan applications. A dataset comprising of applicants financial and demographic data was modelled using Logistic Regression, Decision Tree and Random Forest algorithms. Steps were taken to ensure no data leakage between training and testing data. Modelling was done on the data with no further preprocessing but also with preprocessing techniques such as feature selection and oversampling to balance the dataset applied. These models were then evaluated using accuracy, F1 score and AU-ROC metrics to compare the models. The Random Forest model with feature selection performed best with Accuracy: 0.81, F1: 0.87, and AUC-ROC: 0.73. The differences in the models were marginal but this model did slightly better on the minority negative class. The introduction of feature selection had a surprisingly minimal overall effect on the models. Using SMOTE to balance the data made the models marginally better at predicting the minority negative class but often meaning a loss in overall accuracy. The findings suggest using machine learning techniques can improve classification prediction over a baseline model and pave the way for future work using more sophisticated methods and larger datasets.

Introduction

Loan approval is a crucial step in the loan application process. It is the stage where a financial institution makes the decision to grant or deny a loan to an individual. It is important for both financial institutions and borrowers that loans are only given when they are highly likely to be paid back. For financial institutions this process helps to minimise the risk of defaults which will lead to financial losses and reduces profitability. Regulations are also in place that must be complied to by law to stop loans being given to people who won't be able to pay. For borrowers, loans give access to home purchases or funding for education. However, loans that can't be paid can lead to financial hardship and future difficulties in lending. A fair approval process ensures applicants are assessed on the same criteria across the board and will reduce any potential for bias and discrimination.

Literature review

Predictive modelling and machine learning (ML) techniques can be used to automate the loan approval process. With sufficient datapoints a model can be created that given a loan applicants data can predict whether they will be a good fit for the loan

and therefore whether their application will be approved. Algorithms can be created that use common characteristics that give insights into future repayment likelihood [1]. Previous studies use various methods and algorithms to produce models that perform to various degrees. The process of loan approval presents itself as a binary classification problem with the outcome being either a Yes or No in essence.

Goyal and Kaur [2] studied loan eligibility and found using ensemble methods such as bagging, or boosting to predict loan approval gave the best results. Bagging works by training multiple models on subsets of the data, then voting on each model to make a prediction. It helps to reduce bias and overfitting. Boosting builds models sequentially, each model corrects errors from previous models, this improves overall accuracy by reducing bias and variance [3]. Gonzalez et al. [4] analyse ensemble techniques and explain their benefits as they fuse several models and combine their prediction power and improve on the prediction performance of a single model. Additionally, Bhargav and Sashirekha [5] studied the same data set I've focused on and compared random forest and decision tree algorithms and found random forests outperformed the decision tree with accuracies of 79.4% and 67.3% respectively.

Abdullah et al. [6] used a series of machine learning models to analyse non-performing loans from financial institutions in emerging countries. Their results also showed the Random Forest algorithm with most performant with an accuracy of 76.10%. Shoumo et al. [7] compared various machine learning models and used metrics such as F1 score and AUC-ROC along with accuracy for their evaluation. They found that a support vector machine model used in conjunction with recursive feature elimination and cross-validation gave the best results.

The dataset used consists of 11 predictor variables and the target variable, the data set is imbalanced with 422 loans approved and 192 not approved. The data comes from Kaggle [8]. It was cleaned and analysed in coursework 1 which will be briefly described in the methods section. My study will be a statistical analysis comparing 3 different supervised learning models benchmarked against a baseline model. My baseline model will predict all loan status as approved as this is the majority class, this gives a baseline accuracy to beat of 68.85%. The models analysed will be Logistic Regression, Decision Trees and Random Forest.

Logistic regression uses the sigmoid function to return the probability as the output. This is compared to a predetermined threshold which assigns it to the according label [9]. Logistic regression provides a simple and interpretable model, however, it's results can be limited with problems that have complex relationships.

Decision Trees work by creating complex decision structures. The data is split into subsets based on the features that give the most information. Tree branches are built that act like if-else statements that build decisions. Each internal node represents the decision on a feature and each leaf node represents a label class. Decision trees, unlike many machine learning models are easy to interpret as they use human like decision making. They are however prone to overfitting and can become biased especially when the classes are imbalanced [10].

Random Forest is an ensemble model built by combining multiple decision trees. The mode of the outcome of each decision tree is used to assign a class label. This approach helps to reduce overfitting, producing more generalised results and a better accuracy as it uses the predicting power of multiple trees. Random Forests can also provide feature importance estimates which can help in feature selection. Random Forests models come with limitations. They lose the interpretability of decision trees due to the large number of trees involved. Training can be slow and like decision trees can be biased if the classes are imbalanced [11].

I will build three sets of models to compare against each other. Firstly, will be with the data as it comes. Secondly, I will balance the data using SMOTE and lastly, I will model using selected features and using hyperparameter tuning. Synthetic Minority Over-sampling Technique (SMOTE) introduced by Chawla et al. [12] is a technique for oversampling imbalanced data sets. Rather than duplicating existing instances of the minority class it creates synthetic examples. This is done using K nearest neighbours to select datapoints and then create a new one in the feature space between neighbours. SMOTE helps to improve the performance of machine learning models by providing a more balanced data set for training. It can, however, introduce noise to the data so needs to be used carefully.

To evaluate the models, I will use a variety or metrics. Accuracy will provide an easily interpretable evaluation of the model. It measures the proportion of correct predictions against the total number predictions. As the data is imbalanced it can be misleading. F1 score is the harmonic mean of precision and recall and is useful with imbalanced data as it provides a better measure of performance on the minority class as balancing precision and recall means it isn't overlooked, if either is low it will also be returned as low too. AUC-ROC is another evaluation metric suited for imbalanced data sets. It measures how well a model distinguishes between classes. The ROC plots true positives against false positives and the AUC represents the area under the curve of the plot. The higher the value the better the performance. Each model will also be represented as a confusion matrix. A confusion matrix shows actual vs predicted classifications and gives a visual overview of the errors made by the model.

I hypothesise the key variables for modelling will be:

Credit History – my EDA in CW1 showed most people with credit history had their loans approved and the majority of people with no credit history were rejected (see Figure 4). It had a p-value very close to zero when analysed with a chi square test

Loan to income ratio – This is the ratio of the applicant's income that is used on their loan

Married – my EDA in CW1 showed there was a statistically significant p-value (0.034) when analysed against Loan_Status

Education – My EDA also showed this was statistically significant with a p-value of 0.043

In my analysis I aim to compare the three chosen algorithms and evaluate to see which gives the best result. I hypothesise the best performing algorithm will be Random Forest. Random Forests are better at handling complex relationships than decision tress and logistic regression and they are less prone to overfitting and with an imbalanced data set like the one I'm working with this is a concern.

Figure 1

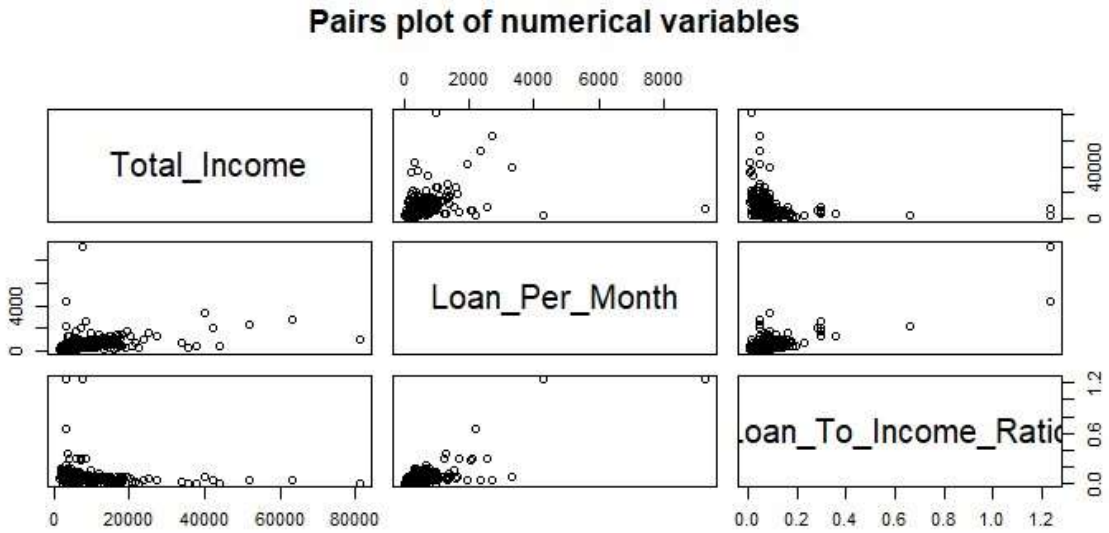


Figure 2

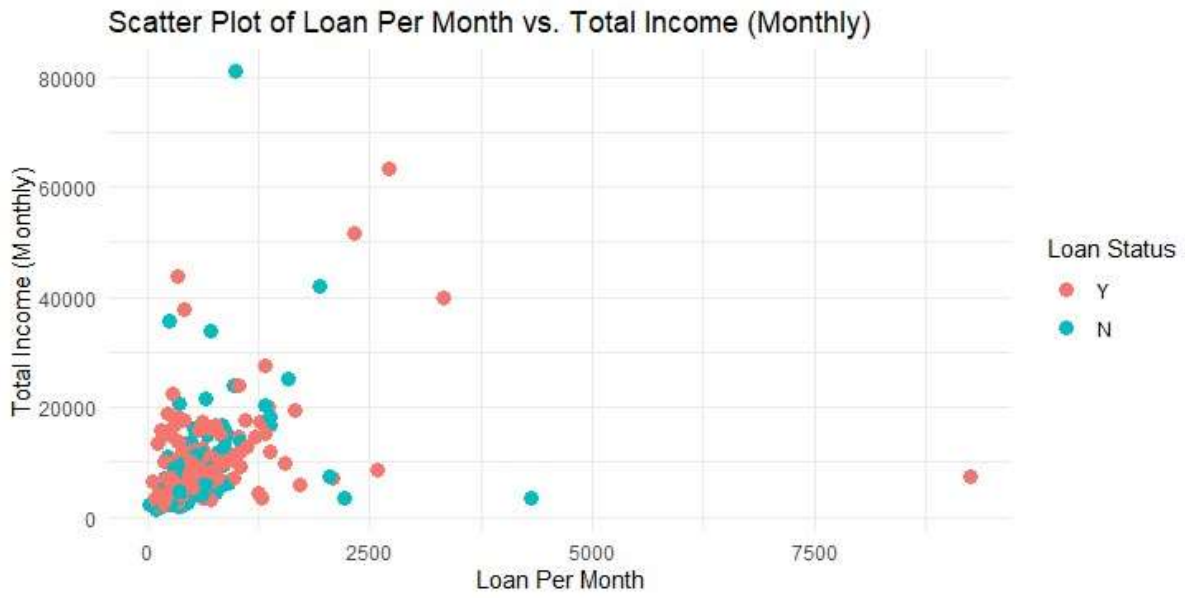


Figure 3

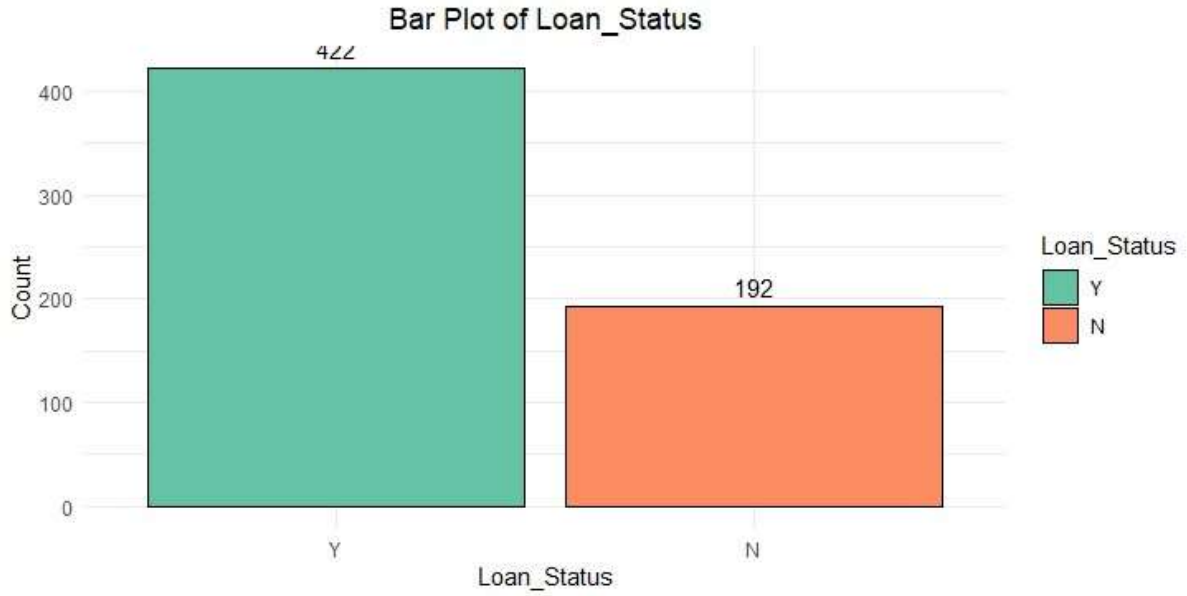
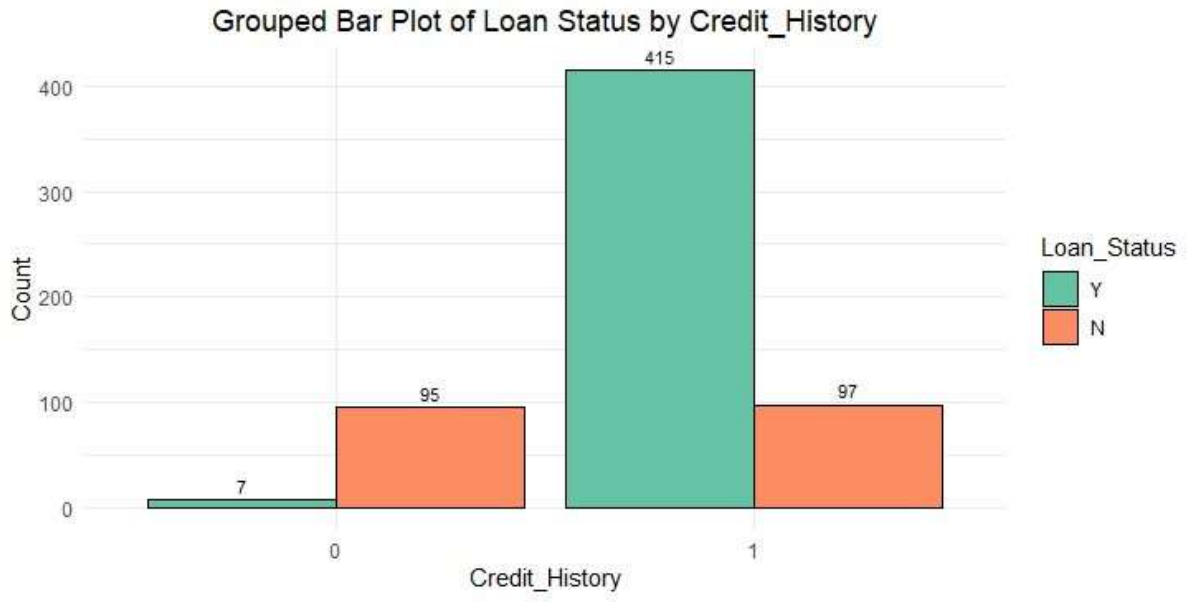


Figure 4



Methods

The dataset used for the study came from Kaggle and can be downloaded here [8]. In CW1 this dataset was cleaned and analysed ready for modelling in CW2. I went through several cleaning steps. I deleted Loan_ID as this was a reference for the record and meaningless for analysis. I changed the categorical variables to Factors to make analysis easier and for the modelling process. I changed some of the column headers to adhere to the principles of tidy data. I then moved onto filling missing data as there were several columns that were incomplete. For this I used a variety of statistical approaches. For Gender and Married I noted there was a correlation between the two variables, so I used this relationship to help impute the nulls. For Loan_Amount I used the median split by Property_Area. For Credit_History I interpolated nulls with the mode dependant on Loan_Status as they are highly dependent. I filled the nulls on the remaining variables with the mode as they did not seem to be dependent on anything else.

My code for this project starts by writing a function that checks whether packages are installed, if they aren't it installs them and then the relevant packages are loaded ready to use in the body of the code [13]. I then set the working directory to the source file location. When running from a new environment it will automatically run from the correct place without errors and load the data set that's saved as a csv file.

The next section provides a summary of the data and some of the relationships. This was from findings in my CW1 but carried forward to provide some context. It shows there are a total of 614 records with the target variable Loan_Status split with 422 records as approved (Y) and 192 not approved (N) (See figure 3). Here I provide a few basic statistical summaries by calling str() and summary(). I also transform the categorical variables back to factors as reloading from csv has caused them to be coming through as character data types. I also provide some summary plots by calling pairs() (see Figure 1); this plots all the numeric variables against each other to show trends. I then create two functions to create bar plots for summarising some key categorical variables and a scatter plot comparing loan per month vs monthly total income (see figures 2-4). I then run my statistical tests from CW1 using chi-squared tests for the categorical variables and \t-tests for the numeric. The summary of the data and relationships is in code lines 45-161.

I then moved on to the modelling and preprocessing sections of my code. I started by splitting my data into training and test datasets. I first set the seed so the split will be replicated in the same way each time the code is run. I then used the caTools package which splits a data set by a given percentage. The function retains the class ratios in the split which is critical especially with an imbalanced data set. I chose to use a split of 70% training data and 30% test. This should give enough data to train the models well but retain enough to be able to test adequately. After the split I did a quick check to make sure the target variable labels looked to be distributed correctly. See code lines 161-186.

In code lines 193-198 I created a baseline model predicting all majority class. This will be used as a baseline to evaluate future models. If a model can do any better than this, it is progress.

I then went on to preprocessing the data. For this I created a function that has steps to label encode the categorical variables and scale the features using standard scaling. It takes in the training and test sets as arguments to the function. The function ensures the same preprocessing steps are applied to the test data set as to the training set. This will avoid data leakage from train to test and should give a more robust model that should be able to generalise to new data. Label encoding is important as many machine learning models will not work when categorical variables are present. Tree based models do not necessarily need numeric data as they work with categorical features too, but logistic regression only accepts numeric values so is needed. Label encoding assigns an integer value to the categorical data for example 0,1,2,3. I chose label encoding over onehot encoding as it's more suitable for tree-based models as they can learn relationships between the features and the target variable without assuming the data is ordinal. However, the downside of this and potentially where onehot encoding is could have been better is for the logistic regression model which is more suited to onehot encoded data [14]. For scaling I chose standardisation so the data will have a mean of 0 and standard deviation of 1. This means each feature will be to the same scale meaning features with larger ranges will be treated the same as those with smaller ranges and won't dominate the models learning. To do this I used the preprocess() function from the caret package [15]. After applying the preprocessing steps, the function returns a list containing the train and test data. The function is then applied, and variables assigned for both the train and test sets. See code lines 203-236

I then went on to train the models. I continued to use the caret package for this process. I first set the trainControl function [16], this function helps to control the training process, I set this so it performs 5-fold cross validation. This means it will train the model on 5 subsets of the data on each run it will train on 4 subsets and test on the remaining subset and the result will be averaged to see how the model performs. Cross validation helps to get a better sense of how a model will generalise to new data.

The models were then trained in turn. I again used the caret package for this step and the train() function. Each model was passed the training data specifying which the target variable to use is and which are the predictor variables, the model to use (logistic regression, decision tree and random forest as discussed earlier), the trainControl variable set up as above and the metric used to evaluate during training, for this I used accuracy. After training all the models I proceeded to use the trained models to make predictions on the test data set using the caret predict() function. See code lines 241-284.

The next section of code creates a function for evaluating the models using the predictions attained. The function takes in as arguments the model predictions, the true labels, the positive class hardcoded as 'Y', and a chart title. I have again chosen caret as the package used for most of the evaluation metrics used. I have chosen caret as it's simple and effective at most of the tasks I've required for this study.

The first evaluation technique the function gives is a confusion matrix [17]. A confusion matrix compares and visualises the frequency of predicted classes against

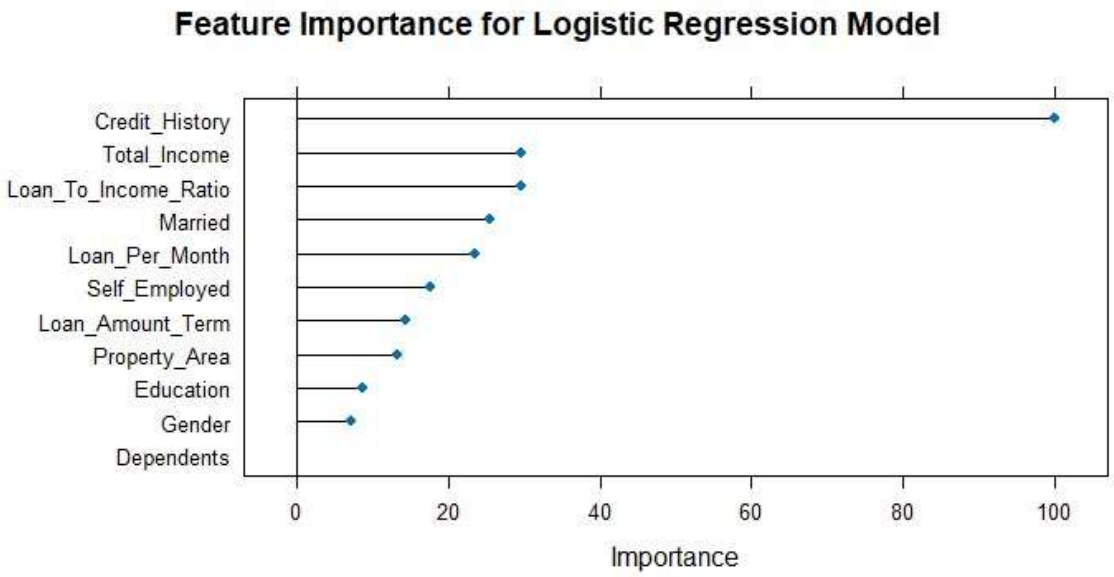
expected classes. It helps to show in an easily digestible form the number of accurate and inaccurate instances of predictions in each classification model. The function then extracts accuracy, sensitivity, precision and recall from the confusion matrix object [18]. The F1 score is then calculated using the precision and recall values. The last metric to be calculated is the ROC_AUC [19]. These evaluation metrics and techniques used together should give a good picture of each model's performance. The metrics are then brough together into a tibble so they can be easily viewed later in a table. The final task the function does is use ggplot to visualise the confusion matrix as a heatmap [20]. This makes it a bit easier to interpret when viewing alongside the other tables. The evaluation function is then used to evaluate my first set of models and print them to generate the data. Bind_rows is used to join all the tibbles containing the evaluation metrics together so they can be viewed and analysed easily. See code lines 288 - 363

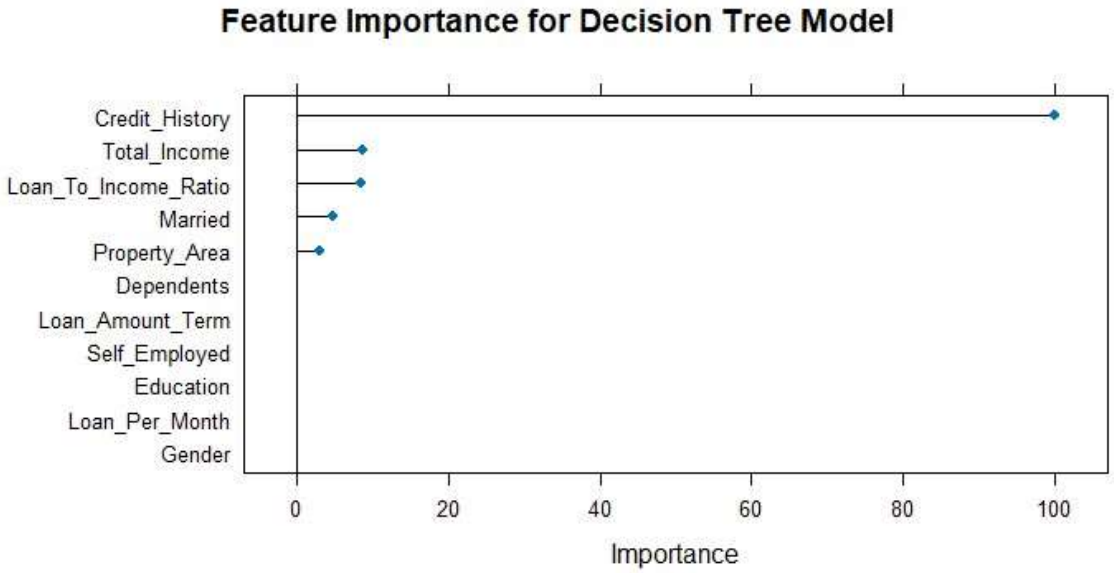
The next part of my code applies SMOTE to the training set to generate oversampling data. To generate this, I used the SMOTE function from the DMwR package [21]. Within the function I set it, so it differentiates between the predictor and target variables and assigns the SMOTE object that includes the synthetic results and the original data to a variable called smote_result. The combined data is then extracted to the variable train_data_smote. The code then pulls the target variable out of the SMOTE object and converts (it's called 'class' in the object) it back to a factor and renames it back to Loan_Status. The same steps were then taken to model, predict, evaluate and put results into a combined tibble for each algorithm but using the data with smote applied. See code lines 367-440.

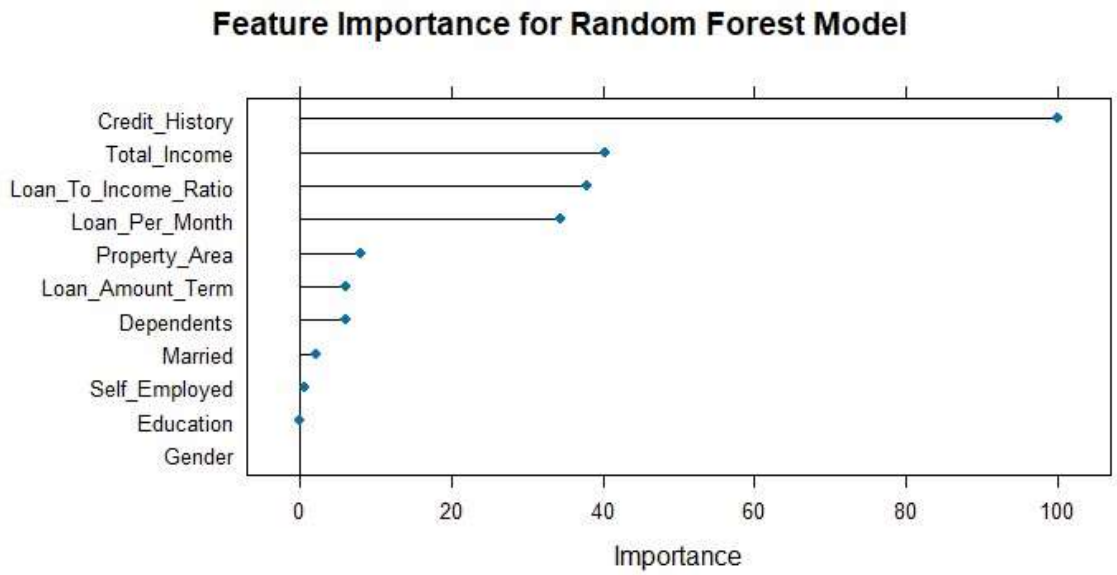
The next set of models I ran are tuned to include just the most important features for each model. This will help to reduce the noise and make the models less prone to overfitting. At this stage I chose to use the data without SMOTE as these had performed better so far. To reduce the features I used carets varImp() function [22] that calculates scores for each variable to show which had the most positive influence on the model training then plotted them – see figure 5. I did this separately for each model and then reduce the data down for each based on the outcome of this. At this stage I also performed some parameter tuning. For the decision tree model, I used tuneLength = 10, this controls the complexity parameter and controls the pruning process which can help to reduce overfitting. Setting this means the model will try 10 different values for the parameter and use the final model will be the one that achieves the best results. For Random Forest I used a tunegrid with the mtry parameter being searched over. This controls the number of variables being considered at each split. It trains a model with each combination and the model with the best result is selected to be used. I also added fixed parameter for number of trees and the min and max number of nodes. I then repeated the steps taken previously to predict and evaluate the data. See code lines 444 – 575. Finally, I joined all the results tibbles together and pivoted it to get it into a good format for interpreting the results. See figure 6

Figure 5

Feature Importance







Results

Results are shown in figures 6 and confusion matrices of each model shown ion figure 7. As this is an imbalanced dataset just analysing accuracy could be misleading so I'm using F1 score and AUC-ROC to help show models that do well at predicting both classes. A baseline model was created (Baseline Dumb Model) to benchmark all other models against. This predicted all instances as Y or approved, the model has a relatively low accuracy (0.69); mimicking the split of Y and N in the target class, and F1 score as there were no negative class predictions. AUC-ROC is 0.5 which reflects 0 negative class predictions.

The first set of models' proper use the test data with no SMOTE oversampling and no further feature selection. The logistic regression model (Accuracy: 0.81, F1: 0.87, AUC-ROC: 0.72) is accurate at predicting the positive class and shows some ability to distinguish between classes but struggles predicting the negative class with 29 incorrectly predicted as Y. The Decision Tree (Accuracy: 0.82, F1: 0.88, AUC-ROC: 0.27) model slightly improves on its positive class predictions but is worse at predicting the minority negative class. The low AUC-ROC suggests the model is overfitting. The Random Forest (Accuracy: 0.79, F1: 0.86, AUC-ROC: 0.71) model performs similarly to the logistic regression model but shows more tendency to predict the negative class when it is positive.

The second set of models use SMOTE to oversample the training data. The Logistic regression and Decision Tree models perform the same (Accuracy: 0.75, F1: 0.82, AUC-ROC: 0.69). The introduction of SMOTE causes the prediction of minority negative class to improve but the overall accuracy drops as they incorrectly predict some positive classes as negative. The Random Forest model does slightly better (Accuracy: 0.77, F1: 0.84, AUC-ROC: 0.70). It is better at predicting the minority negative class than the other models and better than the original Random Forest

model but is not a significant improvement and is worse at predicting the positive class.

The third set of models apply feature selection and hyperparameter tuning. I used the data without SMOTE as this did not have a big impact on the previous models. Feature selection didn't have a significant impact on any of the models. The Logistic Regression (Accuracy: 0.80, F1: 0.87, AUC-ROC: 0.72) and Decision Tree (Accuracy: 0.81, F1: 0.88, AUC-ROC: 0.27) models performed similarly. The Random Forest (Accuracy: 0.81, F1: 0.87, AUC-ROC: 0.73) model showed a slight improvement in AUC-ROC which shows a minor performance benefit over the previous models. It is slightly better at predicting the minority negative class but sacrifices some accuracy as the positive class prediction is slightly worse.

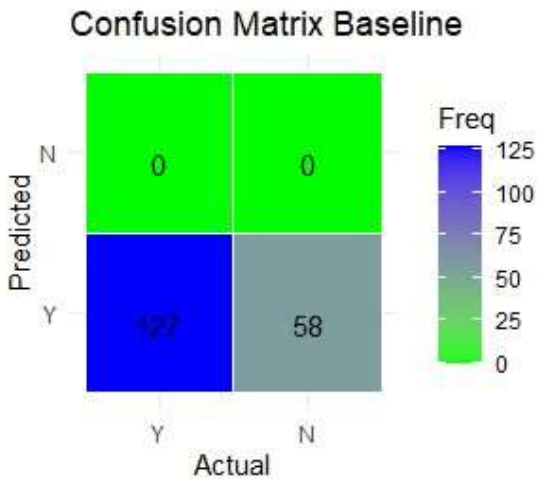
In summary, the introduction of SMOTE and feature selection didn't have a significant impact on the model's performance. Applying SMOTE slightly improved the model's ability to predict the negative class but at a cost of overall accuracy. No significant improvement with feature selection applied suggests the extra features were not adding noise that caused the models to underperform. The model that showed the best overall balance between Accuracy, F1 and AUC ROC was the Random Forest model with feature selection. Indicating this would be the best performing model for the problem. However, it was very close with the Logistic Regression model without feature selection with only a very minor difference. The Random Forest in my opinion is better as it's slightly better at predicting the negative minority class.

Deciding whether to prioritise a model that gives more false positives (Type I Error) or more false negatives (Type II Errors) is always dependent on the problem at hand. With loan approval classification, a Type I Error would approve a loan that should have been rejected, this could end up lending to someone that would not be able to repay the loan causing losses for the bank. A Type II Error would predict that a loan should not be approved when it should, this would mean a missed opportunity for the bank and missed revenue. Deciding on which is more critical is a balancing act. There are business impacts on prioritising Type II errors but there are also regulatory and risk factors to consider if you prioritise type II errors. I have chosen a model that tries to balance the two out.

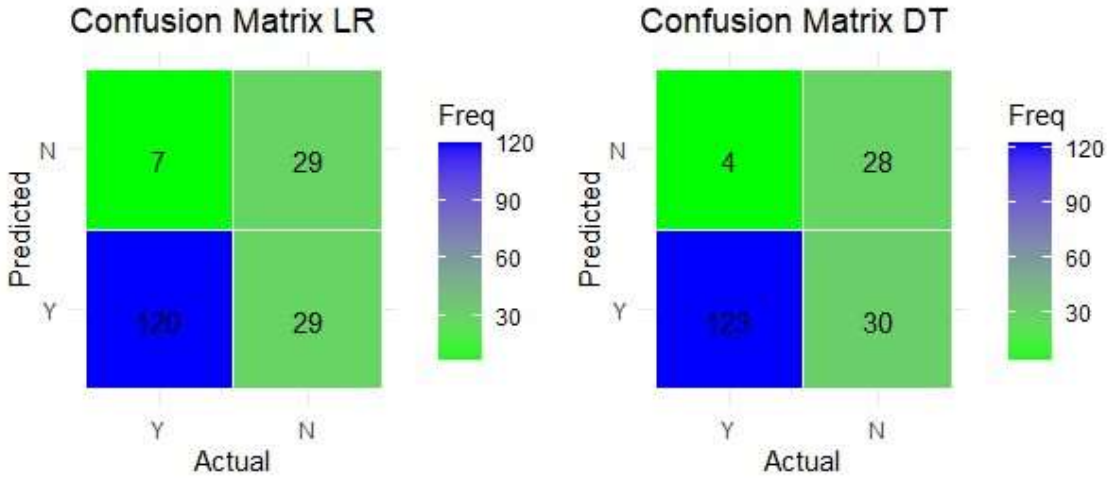
Figure 6

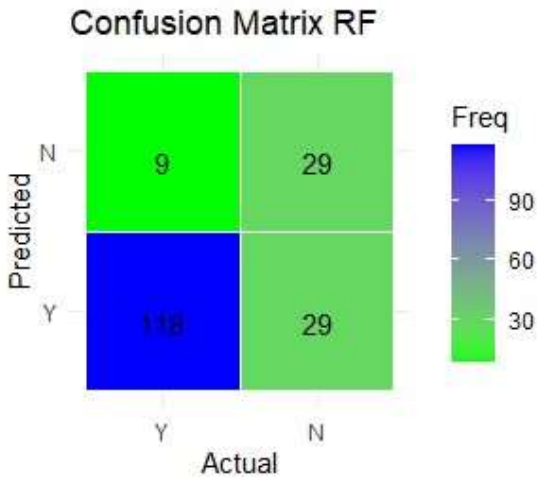
Model	Accuracy	F1 Score	AUC ROC
Baseline Dumb Model	0.6864865	0.8141026	0.5000000
Logistic Regression	0.8054054	0.8695652	0.7224409
Decision Tree	0.8162162	0.8785714	0.2743687
Random Forest	0.7945946	0.8613139	0.7145669
Logistic Regression (smote)	0.7459459	0.8199234	0.6885012
Decision Tree (smote)	0.7459459	0.8199234	0.6885012
Random Forest (smote)	0.7675676	0.8377358	0.7042493
Logistic Regression (fs)	0.8054054	0.8695652	0.7224409
Decision Tree (fs)	0.8162162	0.8785714	0.2743687
Random Forest (fs)	0.8054054	0.8686131	0.7271246

Figure 7

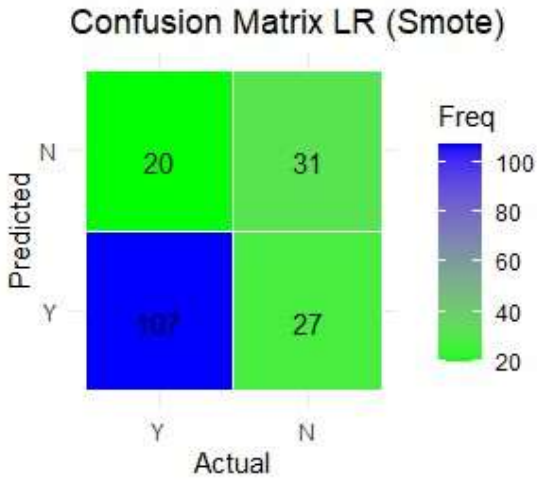


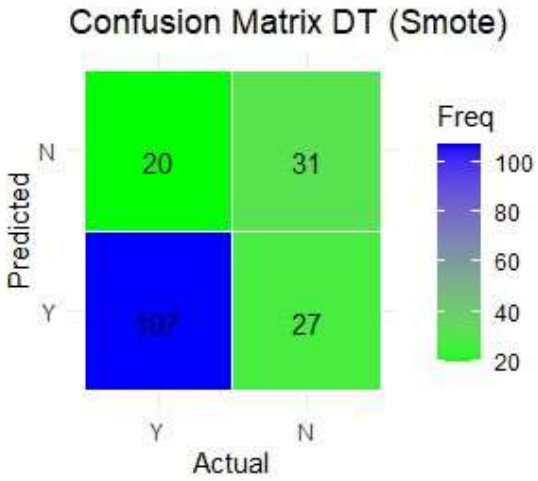
Initial Models

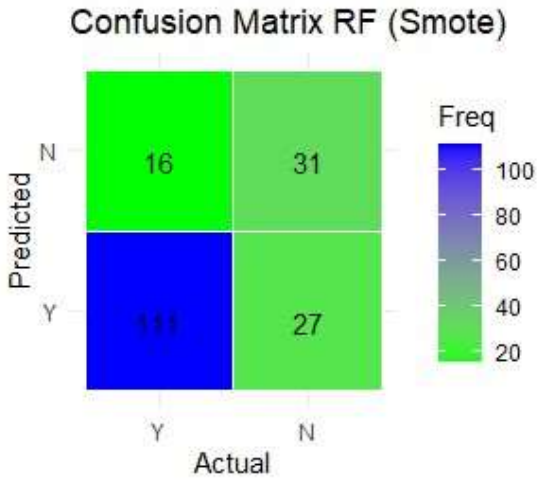




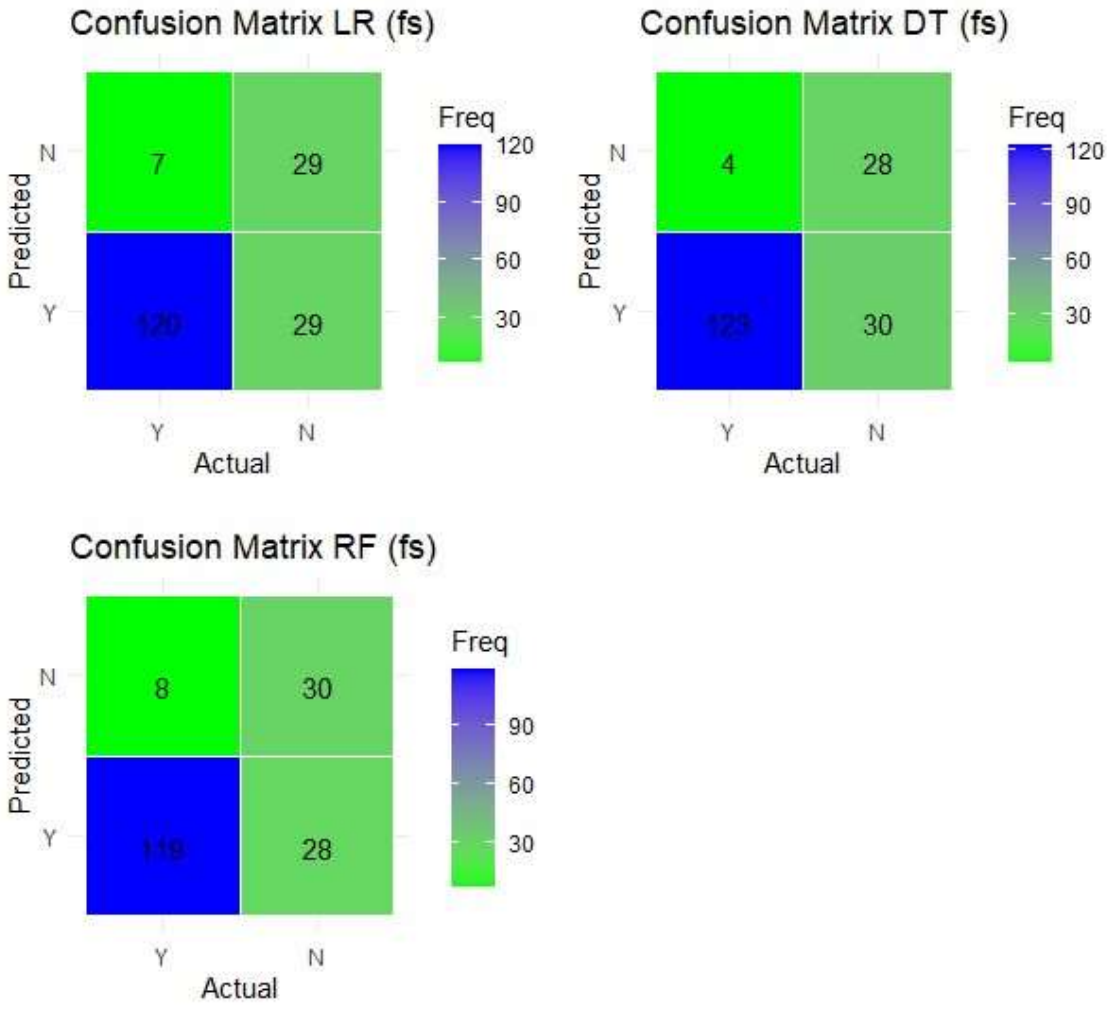
Models with SMOTE







Models with feature selection



Discussion & Conclusion

The results of the study show that all models using machine learning techniques were able to beat a baseline model that predicted all the majority positive class. This shows that machine classification is better than a random model. The models were all good at predicting the positive class but had some issues predicting the minority class. The models with SMOTE were slightly better at predicting the minority class but the best all round model was the Random Forest model with feature selection. As hypothesised, the most important variable as identified by running feature importance on each of the models was the Credit_History variable. Two of the other variables identified through my EDA were also up there with the most important – Loan_To_Income_Ratio and Married. Another that scored highly was the Total_Income variable

The results were not particularly impressive. There wasn't a substantial difference between models, and I was surprised that logistic regression that is known to be quite a simple algorithm did not significantly outperform the Random Forest algorithm which is known to be able to capture more complex relationships. I was

also surprised SMOTE and feature selection did not make much of a difference as often with imbalanced data the models struggle to predict the minority class but SMOTE should have helped to correct this.

There could be several reasons for the lack of change in the results and a number or areas I feel could be tested further or improved on in if the study was replicated. The EDA and data cleaning in CW1 could be improved and updated. The way I handled nulls could have impacted the data. These could have led to inconsistencies in the data, and inadvertently contimnating the data or intrdocuing imputation bias. The Credit_History variable especially could have been imputed differently; this was imputed using the mode dependaont Loan_Status. It might have been better to try dropping the nulls from here, so the data wasn't contaminated. This could have impacted the models' decisions on the minority class. The project could have benefited from further feature engineering. This, however, would have needed better domain knowledge and I am not an expert in the field of loan approvals. Further features could have given some new insights that could have helped with modelling. I could also have tried different sampling techniques. To balance the dataset, I could have used undersampling. However, as there were only 614 original records undersampling would have reduced the dataset significantly losing lots of important data. I chose to use the data without SMOTE applied to use for the feature selected models. A further study could benefit from trying this with the SMOTE data too. This could have had an impact on the results.

In further works in this area, it would be beneficial to address data issues such as the way nulls were handled. In an ideal world the dataset would be much larger so some further data collection would be a vital step to try and get better models. I tried 3 models out, but the study could have benefited from more complex ensemble methods such as XGBoost or even a Neural Network approach.

Bibliography

1. Koulouridi E, Kumar S, Nario L, Pepanides T, Vettori M. Managing and monitoring credit risk after the COVID-19 pandemic [Internet]. Mckinsey.com. McKinsey & Company; 2020 [cited 2024 Sep 5]. Available from: https://www.mckinsey.com/capabilities/risk-and-resilience/our-insights/managing-and-monitoring-credit-risk-after-the-covid-19-pandemic

2. Goyal A, Kaur R. Loan prediction using ensemble technique [Internet]. 2016. Available from: https://www.ijarcce.com/upload/2016/march-16/IJARCCE%20128.pdf

3. Dey R. Bagging v/s boosting - roshmita dey [Internet]. Medium. 2024 [cited 2024 Sep 5]. Available from: https://medium.com/@roshmitadey/bagging-v-s-boosting-be765c970fd1

4. González S, García S, Del Ser J, Rokach L, Herrera F. A practical tutorial on bagging and boosting based ensembles for machine learning: Algorithms, software tools, performance study, practical perspectives and opportunities. Inf Fusion [Internet]. 2020;64:205–37. Available from: http://dx.doi.org/10.1016/j.inffus.2020.07.007

5. Bhargav P, Sashirekha K. A machine learning method for predicting loan approval by comparing the Random Forest and decision tree algorithms. Surv Fish Sci [Internet]. 2023 [cited 2024 Sep 5];10(1S):1803–13. Available from: https://sifisheriessciences.com/journal/index.php/journal/article/view/414

6. Abdullah M, Chowdhury MAF, Uddin A, Moudud-Ul-Huq S. Forecasting nonperforming loans using machine learning. J Forecast [Internet]. 2023;42(7):1664–89. Available from: http://dx.doi.org/10.1002/for.2977

7. Shoumo SZH, Dhruba MIM, Hossain S, Ghani NH, Arif H, Islam S. Application of machine learning in credit risk assessment: A prelude to smart banking. In: TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON). IEEE; 2019. p. 2023–8.

8. Kaggle.com. [cited 2024 Sep 5]. Available from: https://www.kaggle.com/datasets/rishikeshkonapure/home-loan-approval/data.

9. Gong D. Top 6 machine learning algorithms for classification [Internet]. Towards Data Science. 2022 [cited 2024 Sep 5]. Available from: https://towardsdatascience.com/top-machine-learning-algorithms-for-classification-2197870ff501

10. Geeksforgeeks.org. [cited 2024 Sep 5]. Available from: https://www.geeksforgeeks.org/decision-tree-introduction-example/.

11. Sinap V. A comparative study of loan approval prediction using machine learning methods. Gazi Üniv Fen Bilim Derg C Tasar ve Teknol [Internet]. 2024;12(2):644–63. Available from: https://dergipark.org.tr/tr/download/article-file/3808895

12. Chawla NV. SMOTE: Synthetic minority over-sampling technique [Internet]. Arxiv.org. 2002 [cited 2024 Sep 5]. Available from: https://arxiv.org/pdf/1106.1813

13. Elegant way to check for missing packages and install them? [Internet]. Stack Overflow. [cited 2024 Sep 5]. Available from: https://stackoverflow.com/questions/4090169/elegant-way-to-check-for-missing-packages-and-install-them

14. Geeksforgeeks.org. [cited 2024 Sep 5]. Available from: https://www.geeksforgeeks.org/convert-factor-to-numeric-and-numeric-to-factor-in-r-programming/)(https://dplyr.tidyverse.org/reference/across.html

15. Kuhn M. 3 Pre-Processing [Internet]. Github.io. 2019 [cited 2024 Sep 5]. Available from: https://topepo.github.io/caret/pre-processing.html

16. TrainControl function - RDocumentation [Internet]. Rdocumentation.org. [cited 2024 Sep 5]. Available from: https://www.rdocumentation.org/packages/caret/versions/6.0-92/topics/trainControl

17. Kuhn M. 17 Measuring Performance [Internet]. Github.io. 2019 [cited 2024 Sep 5]. Available from: https://topepo.github.io/caret/measuring-performance.html

18. How to extract Accuracy from caret's confusionMatrix? [Internet]. Stack Overflow. [cited 2024 Sep 5]. Available from: https://stackoverflow.com/questions/23433641/how-to-extract-accuracy-from-carets-confusionmatrix

19. How to calculate AUC (area under curve) in R? [Internet]. GeeksforGeeks. 2022 [cited 2024 Sep 5]. Available from: https://www.geeksforgeeks.org/how-to-calculate-auc-area-under-curve-in-r/

20. Plot confusion matrix in R using ggplot [Internet]. Stack Overflow. [cited 2024 Sep 5]. Available from: https://stackoverflow.com/questions/37897252/plot-confusion-matrix-in-r-using-ggplot

21. SMOTE function - RDocumentation [Internet]. Rdocumentation.org. [cited 2024 Sep 5]. Available from: https://www.rdocumentation.org/packages/DMwR/versions/0.4.1/topics/SMOTE

22. VarImp function - RDocumentation [Internet]. Rdocumentation.org. [cited 2024 Sep 5]. Available from: https://www.rdocumentation.org/packages/caret/versions/6.0-92/topics/varImp