CS571 Artificial Intelligence

Professor Sreedevi Gutta

Final Project Report

Loan Prediction

Adriana Caetano and Torsha Mazumdar

Spring 2022

Project Overview

The loan prediction problem dataset, downloaded from Kaggle, was used for our project. After a thorough analysis and visualization to understand the data, the dependencies and the limitations, we use 3 machine learning models and compare the performances.

Motivation

Dream Housing Finance company deals in all home loans. They have a presence across all urban, semi-urban, and rural areas. Customer-first applies for a home loan after that company validates the customer eligibility for a loan.

With the automation of many different sectors, loan approval is also an area that could be automated. The client could fill up a form online and a program could pre-approve a client, to be reviewed by a person for checking documentation only after passing the first screening of the algorithm. This could accelerate the process for approved clients while minimizing the risk for the institutions. It leads to reduced wait times and a more efficient loan application and approval process. Dream Housing Finance company aims to achieve these goals and made this dataset accessible, enabling researchers to develop models for this purpose.

To develop an efficient model for this purpose, we need to first try out different models, run a number of experiments aiming to increase the prediction accuracy, with a high metric for recall because we don't want to grant a loan to someone that should not be eligible..

Dataset

Problem category

1. Supervised Learning - learning from the data based on sample input-output pairs. Our dataset has input fields and an output label, hence, it is a supervised learning problem.
    - Input - Customer Attributes, Loan Amount, Term
    - Output/Label - Loan Status
2. Binary Classification - The output label only has 2 values, i.e., loan status is either Y(Yes) or N(No). Therefore, it is a binary classification problem.

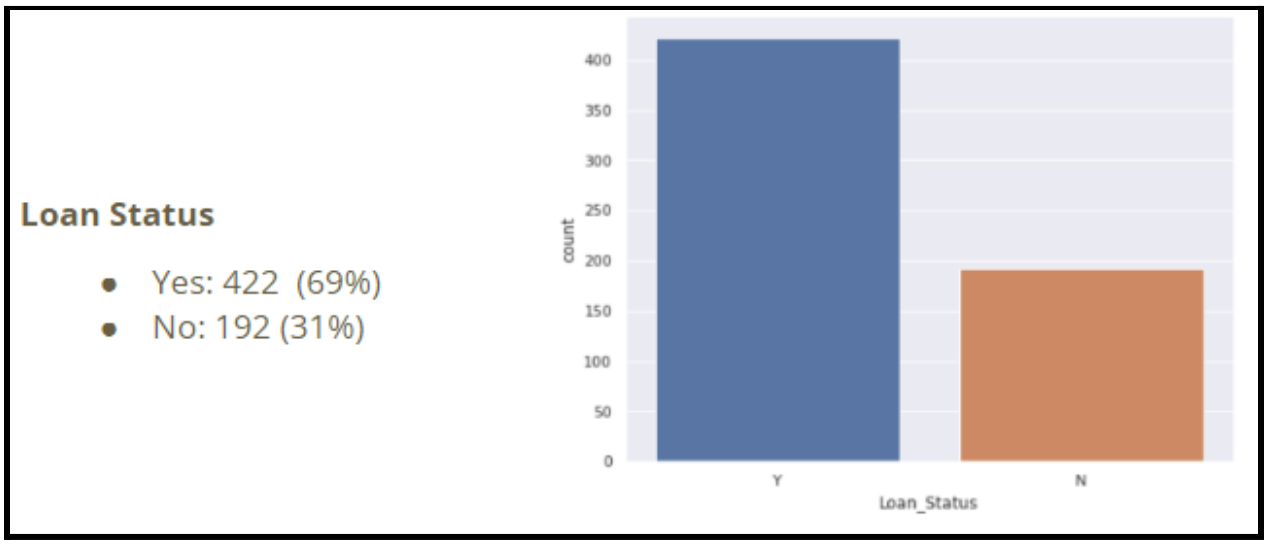
Dataset description

It is a short dataset with 13 columns and 614 rows. An overview of the dataset can be seen in the table below:

Features	Values				Datatype	Non-null Count	Missing Values
Loan_ID	Unique ID				categorical	614	0
Gender	Male	Female			categorical	601	13
Married	Yes	No			categorical	611	3
Dependents	0	1	2	3+	categorical	599	15
Education	Graduate	Not Graduate			categorical	614	0
Self_Employed	Yes	No			categorical	582	32
ApplicantIncome	$150 - $81000 per month				numerical	614	0
CoapplicantIncome	$0 - $41667 per month				numerical	614	0
LoanAmount	$9000 - $700000				numerical	592	22
Loan_Amount_Term	12 months - 480 months				numerical	600	14
Credit_History	1	0			numerical	564	50
Property_Area	Rural	Urban	Semiurban		categorical	614	0
Loan_Status	Y	N			categorical	614	0

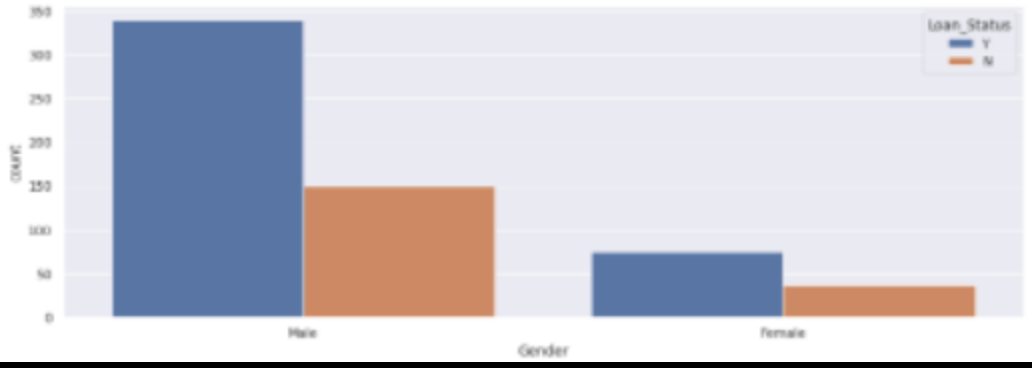
The handling of missing values is discussed later.

Visualization



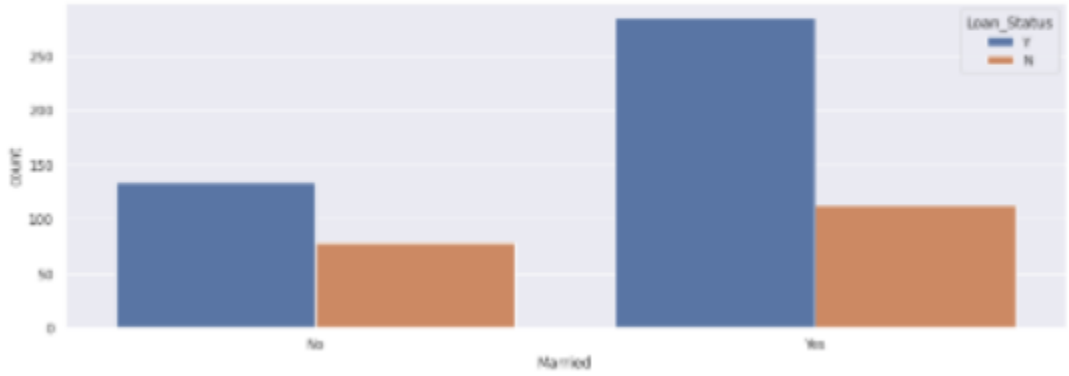
Gender

Male: 489 (Y 69%, N 31%) Female: 112 (Y 67%, N 33%)



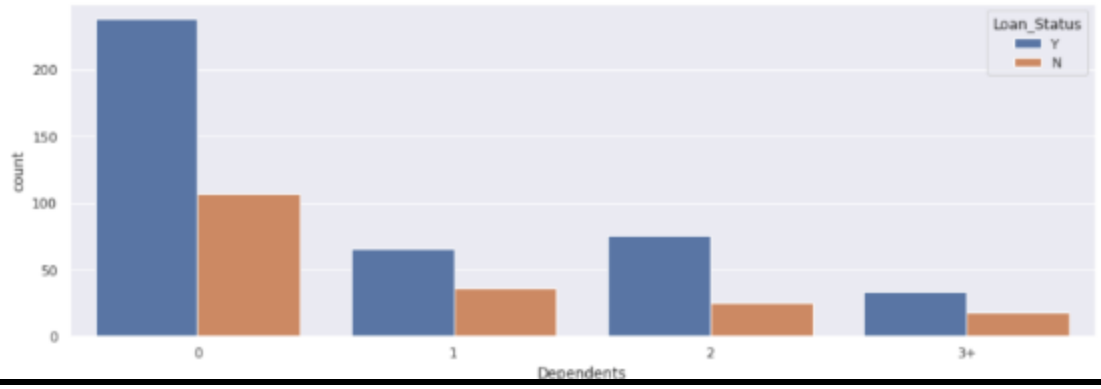
Married

No: 213 (Y 63%, N 37%) Yes: 398 (Y 72%, N 28%)



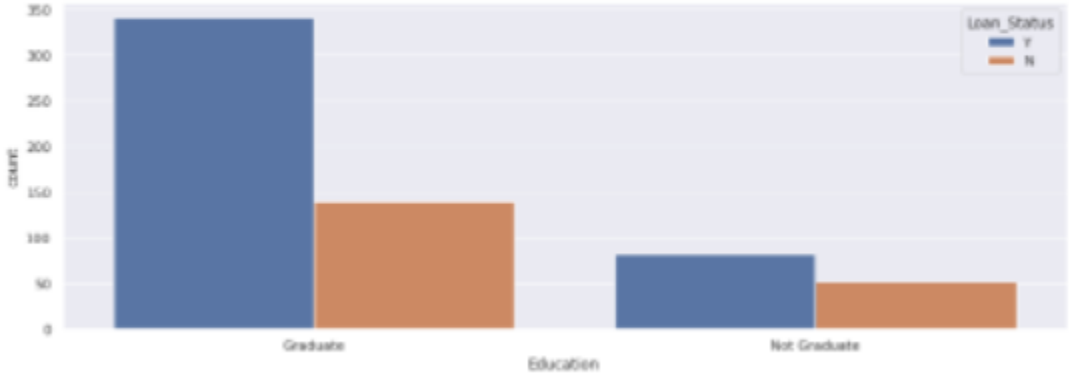
Dependents

0: 345 (Y 69%, N 31%) 1: 102 (Y 65%, N 35%)
2: 101 (Y 76%, N 24%) 3+: 51 (Y 64%, N 36%)



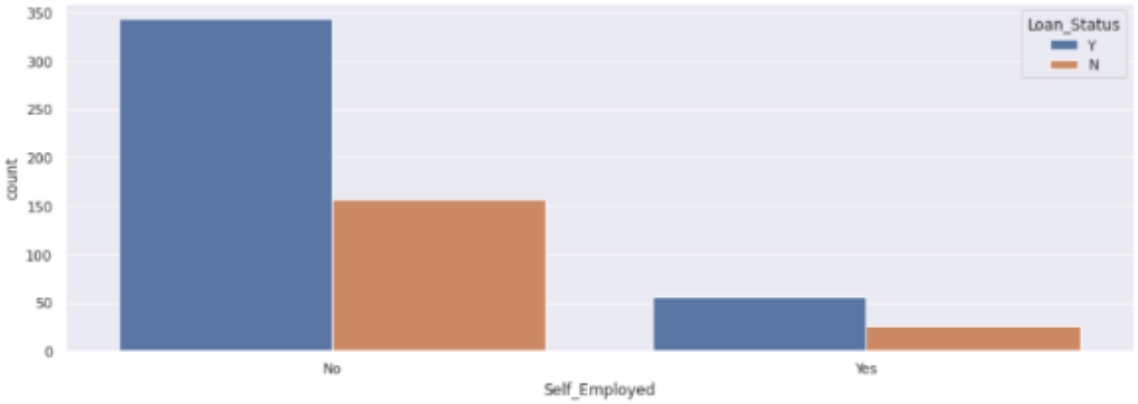
Education

Graduate: 480 (Y 71%, N 29%) Not Graduate: 134 (Y 61%, N 39%)



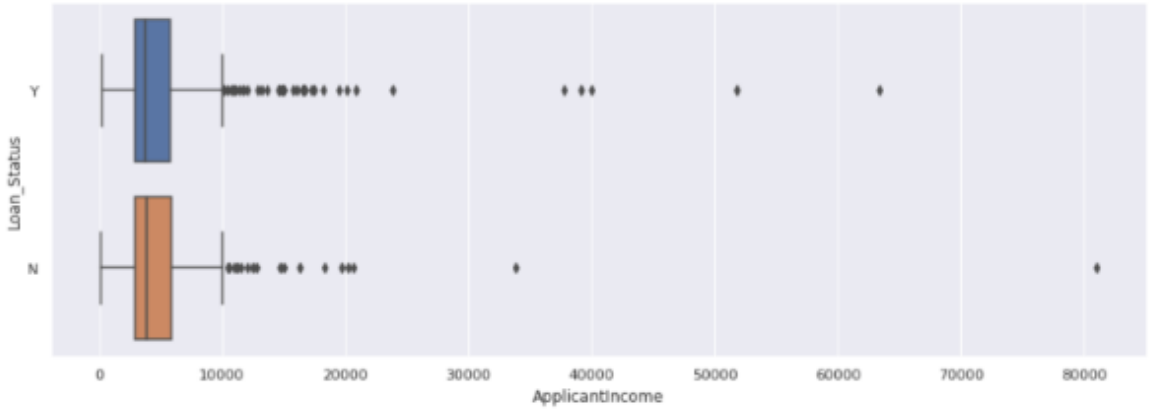
Self-Employed

No: 500 (Y 69%, N 21%) Yes: 82 (Y 68%, N 22%)



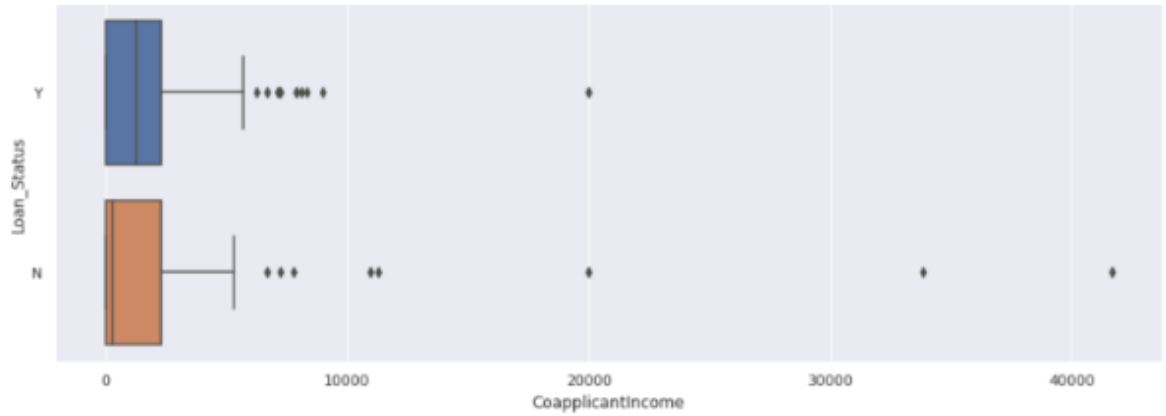
Applicant Income

Min: 150 Max: 81000 25%: 2877 50%: 3812 75%: 5795 Mean: 5403



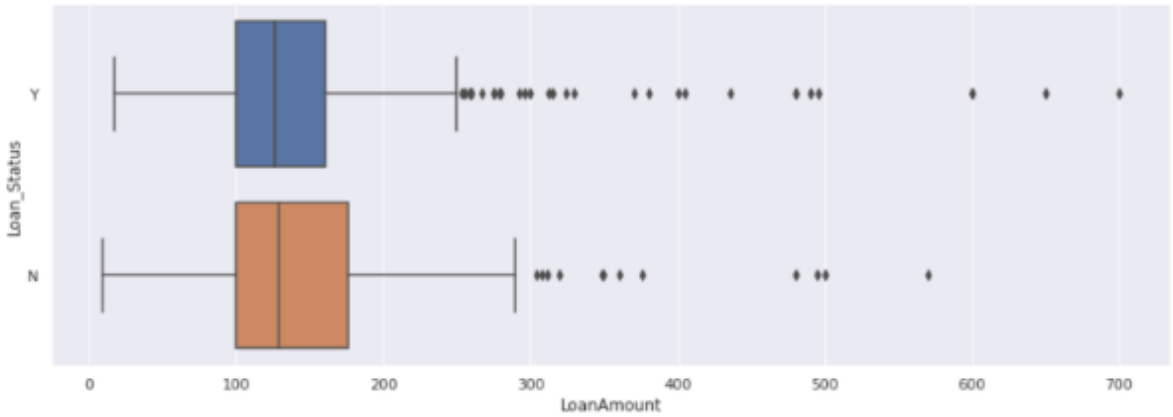
Co-Applicant Income

Min: 0 Max: 41667 25%: 0 50%: 1188 75%: 2297 Mean: 1621



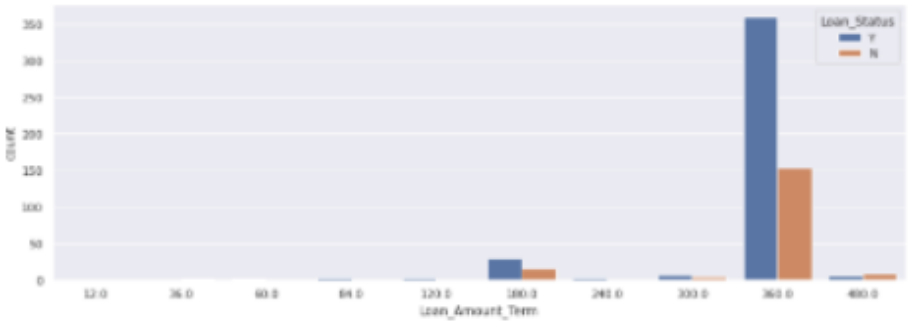
Loan Amount

Min: 9k Max: 700k 25%: 100k 50%: 128k 75%: 168k Mean: 146k



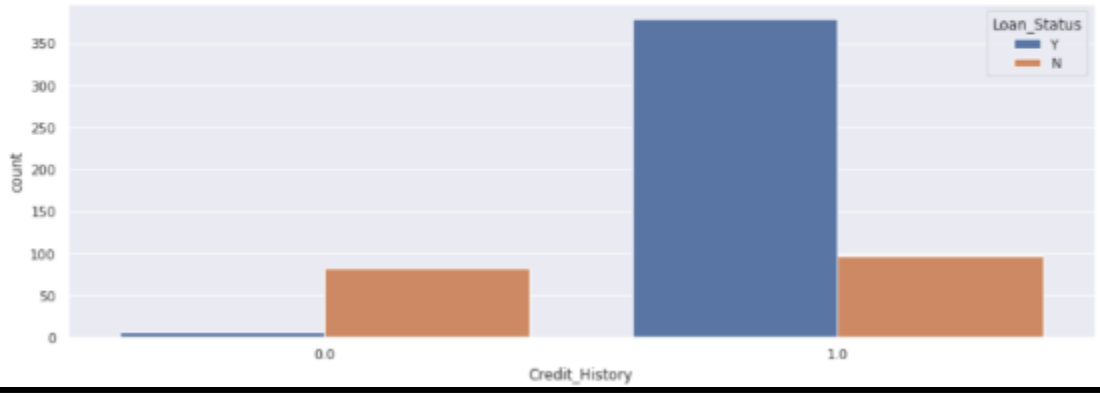
Loan Amount Term

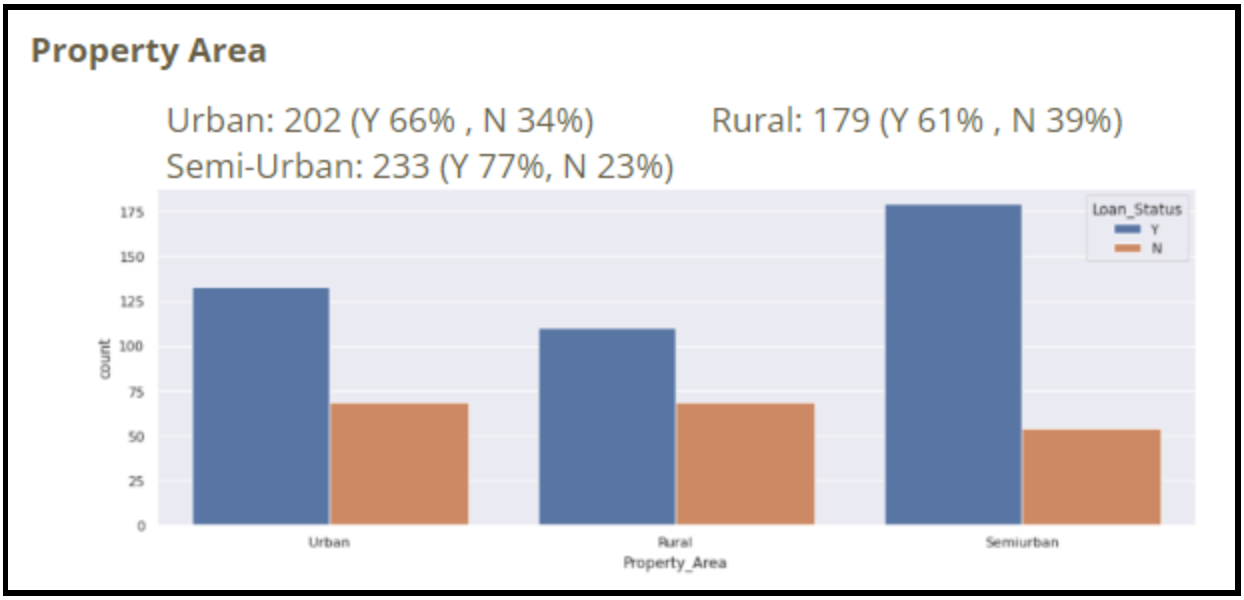
0-7 years: 9 (Y 66%, N 34%) 10 years: 3 (Y 100%) 15 years: 44 (Y 66%, N 34%)
20 years: 4 (Y 75%, N 25%) 25 years: 13 (Y 62%, N 38%)
30 years: 512 (Y 70%, N 30%) 40 years: 15 (Y 40%, N 60%)



Credit History

0: 89 (Y 8% , N 92%) 1: 475 (Y 80% , N 20%)





Preprocessing

Loan ID was dropped since this feature was not relevant for the classification problem.

The missing values were first categorized as missing completely at random (MCAR) or Missing not at random (MNAR). The fields with missing values which were unrelated to any other field were considered MCAR and 2 fields(dependents and credit history) were considered MNAR since we assumed that the missing values in this field are related to the reason it's missing, also known as nonignorable nonresponse.

The missing values were filled up using the below criteria:

- MCAR categorical features : Mode
- MCAR numerical features : Mean or Median
- Zero for dependents(MNAR): if a person "forgot" to fill up the number of dependents, most likely they don't have dependents
- Zero for credit history(MNAR): if a person "forgot" to fill up the credit history, most likely they don't have history

Field wise values for missing data :

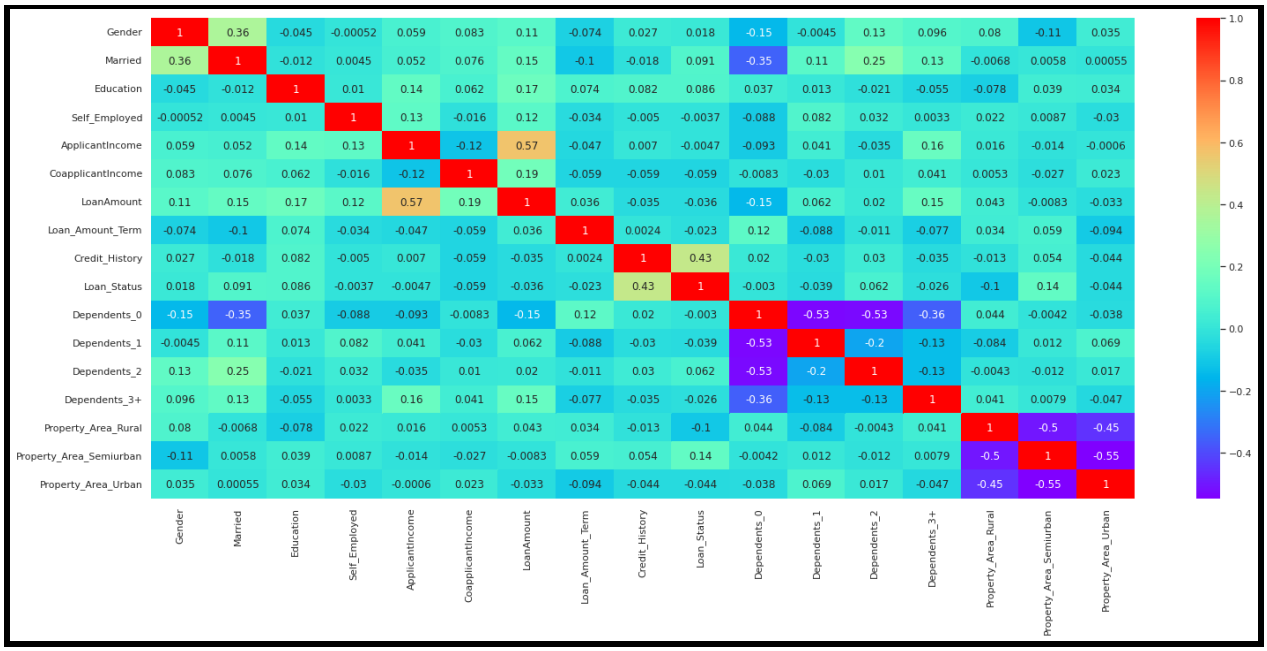
Gender	→	mode
Married	→	mode
Dependents	→	0
Self_Employed	→	mode
LoanAmount	→	mean
Loan_Amount_Term	→	median

Credit_History → 0

Encoding

The ML models (discussed later), used in our project are compatible with both categorical and numerical features. However, the scikit-learn libraries for the models in use do not support both categorical and numerical features. Therefore, we have to encode the categorical features into numerical ones.

1. Label Encoding - We performed label encoding for the fields - `'Gender'`, `'Married'`,`'Dependents'`,`'Education'`,`'Self_Employed'`,`'Property_Area'`. However, the training accuracy for the ML models was not satisfactory.
2. Pandas get_dummies - We then tried a simple pandas way of encoding the categorical fields. We achieved a higher training accuracy with this. We used get_dummies for categorical features that were not binary.
3. Pandas replace - We tried the simplest method of manually replacing the categorical values for numerical values. This technique produces similar results to the get_dummies, but it does not increase the number of features. Although it does not increase the complexity of the problem, it also does not help visualizing which features are more relevant to achieve the results. We used replace for binary categorical features.
4. We did not use One-hot encoding since it adds many columns to the dataset. Since our dataset is small and simple, we wanted to keep it simple.



The heatmap helps us visualize the features that are the best predictors.

Splitting the dataset

70% of the dataset was used for training and 30% of the dataset was kept aside for testing. Since the dataset was small, we did not divide the dataset further into a validation dataset and used k-fold cross validation instead.

- 614 unique clients

The dataset divided into train and test, in a 70:30 ratio -

- 429 for train
- 185 for test

Resampling

The train dataset was imbalanced -

- 300 labels for Y (70%)
- 129 labels for N (30%)

Therefore, we decided to upsample the dataset since the original size of the dataset is small.

For imbalance datasets Sklearn has some ready to use techniques. We chose SMOTE, Synthetic Minority Oversampling TEchnique, to oversample the minority class creating new data points. SMOTE selects an example of the minority class, finds some k-nearest

neighbors, selects one at random, draws a line between them, and creates new data along that line. This technique creates new data points that are very similar to the original ones without having the exact same values.

After oversampling we have -

- 300 labels for Y (50%)
- 300 labels for N (50%)

Methods

We selected the below 3 models to experiment with -

1. Decision Tree Classifier
    i. simple and easy to interpret
    ii. trees can be visualized
2. Random Forest Classifier
    i. have much higher accuracy than the single decision tree
    ii. doesn't overfit the model, thus gives a good prediction on unseen datasets
    iii. low bias and low variance
3. Logistic Regression
    i. efficient for linear dataset
    ii. it can handle both dense and sparse input

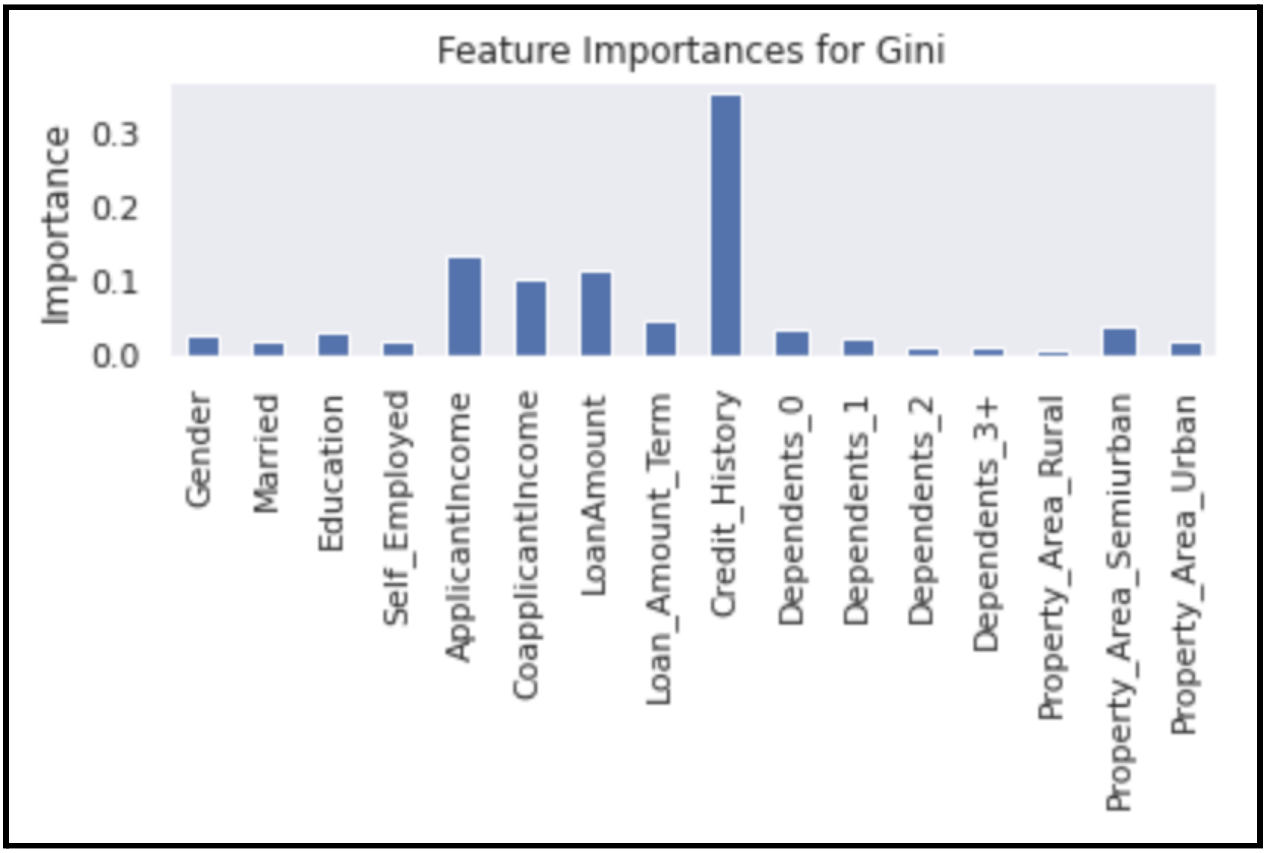
Decision Tree Classifier

- First we fit the upsampled(using SMOTE) training dataset on the decision tree model using DecisionTreeClassifier from sklearn.tree package using the default hyperparameters. For the oversampled dataset using SMOTE, we get a tree with 111 leaves and depth of 16.
- Then, we experiment with criterion gini and entropy to find the best results using cross-validation. We then decide to use gini, the default criterion, using 5-fold cross-validation.

CV for oversampled dataset using SMOTE

cv	Gini	Gini std	Entropy	Entropy std
2	0.7649999999999999	0.031666666666666676	0.7383333333333333	0.018333333333333368
3	0.75	0.008164965809277268	0.7516666666666666	0.008498365855987981
4	0.755	0.01787300882460601	0.7416666666666667	0.013642254619787412
5	0.7716666666666667	0.03100179206289712	0.7699999999999999	0.02147349787787521
6	0.7716666666666668	0.030776975521032313	0.7400000000000001	0.0608276253029822
7	0.7668946648426812	0.05090032890610718	0.7550908735587258	0.028481661144830764
8	0.7416666666666666	0.07053367989832943	0.7633333333333333	0.049328828623162485
9	0.7533795668124026	0.05836266581515716	0.7749635660083422	0.04427681858798265
10	0.7533333333333333	0.04068851871911235	0.7616666666666667	0.04716990566028301

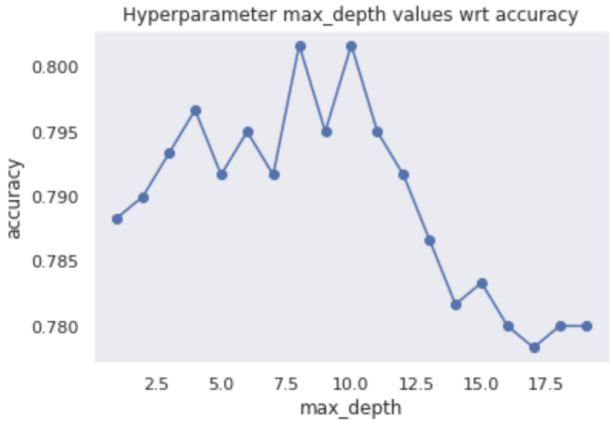
- To find out the features importances we used the attribute feature_importances_ from the tree model. It returns the relative importance of each feature on the classification prediction. With this information we experiment using the features that could give better results, and found that `['LoanAmount', 'Credit_History', 'ApplicantIncome', 'CoapplicantIncome', 'Loan_Amount_Term', 'Property_Area_Semiurban']` returns a score of 78.



- Next, we used GridSearchCV to find the best values for hyperparameters (max_depth, and max_leaf_nodes) for the selected features
    - k-Fold cross-validation : k=5

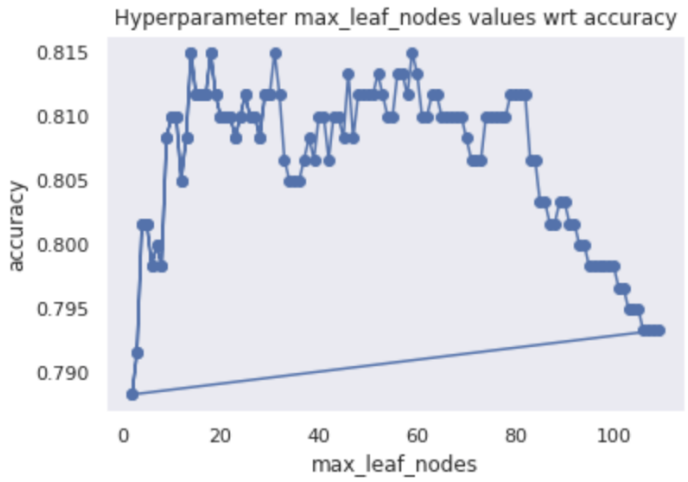
    - Max_depth =7
    - Max_leaf_nodes = 10
- Even though we got our values through GridSearchCV, we performed manual experiments for the hyperparameters and generated plots to see the performance to select the best results
    - Max_depth = 8

max_depth	Accuracy	std
1	0.7883333333333333	0.02449489742783177
2	0.79	0.024944382578492928
3	0.7933333333333333	0.019293061504650353
4	0.7966666666666666	0.029627314724385304
5	0.7916666666666666	0.01900292375165228
6	0.7949999999999999	0.020138409955990935
7	0.7916666666666667	0.025276251480171832
8	0.8016666666666667	0.016158932858054424
9	0.7950000000000002	0.021473497877875183
10	0.8016666666666667	0.009718253158075512
11	0.7949999999999999	0.016329931618554505
12	0.7916666666666667	0.02738612787525829
13	0.7866666666666667	0.01354006400772659
14	0.7816666666666666	0.022607766610417544
15	0.7833333333333334	0.022973414586817033
16	0.78	0.020138409955990946
17	0.7783333333333333	0.019436506316150997
18	0.78	0.020138409955990946
19	0.78	0.020138409955990946



    - Max_leaf_nodes = 14

max_leaf_nodes	Accuracy	std
2	0.7883333333333333	0.02449489742783177
3	0.7916666666666666	0.02357022603955158
4	0.8016666666666665	0.027588242262078067
5	0.8016666666666665	0.027588242262078067
6	0.7983333333333332	0.03045944480416178
7	0.8	0.026352313834736484
8	0.7983333333333332	0.028087165910587845
9	0.8083333333333333	0.019720265943665383
10	0.8100000000000002	0.019293061504650374
11	0.8100000000000002	0.01999999999999997
12	0.805	0.021473497877875208
13	0.8083333333333332	0.016666666666666653
14	0.8149999999999998	0.02380476142847615
15	0.8116666666666665	0.021473497877875215
16	0.8116666666666665	0.022730302828309745
17	0.8116666666666665	0.021473497877875215
18	0.8149999999999998	0.023213980461973545
19	0.8116666666666665	0.028185890875479597
20	0.8099999999999999	0.02953340857778224
21	0.8099999999999999	0.02953340857778224
22	0.8099999999999999	0.0260341655863555
23	0.8083333333333333	0.024152294576982387
24	0.8099999999999999	0.021984843263788197
25	0.8116666666666668	0.023921166824012203
26	0.8099999999999999	0.019293061504650388
27	0.8099999999999999	0.019293061504650388
28	0.8083333333333333	0.014907119849998582
29	0.8116666666666665	0.01795054935711501
30	0.8116666666666668	0.02013840995599095

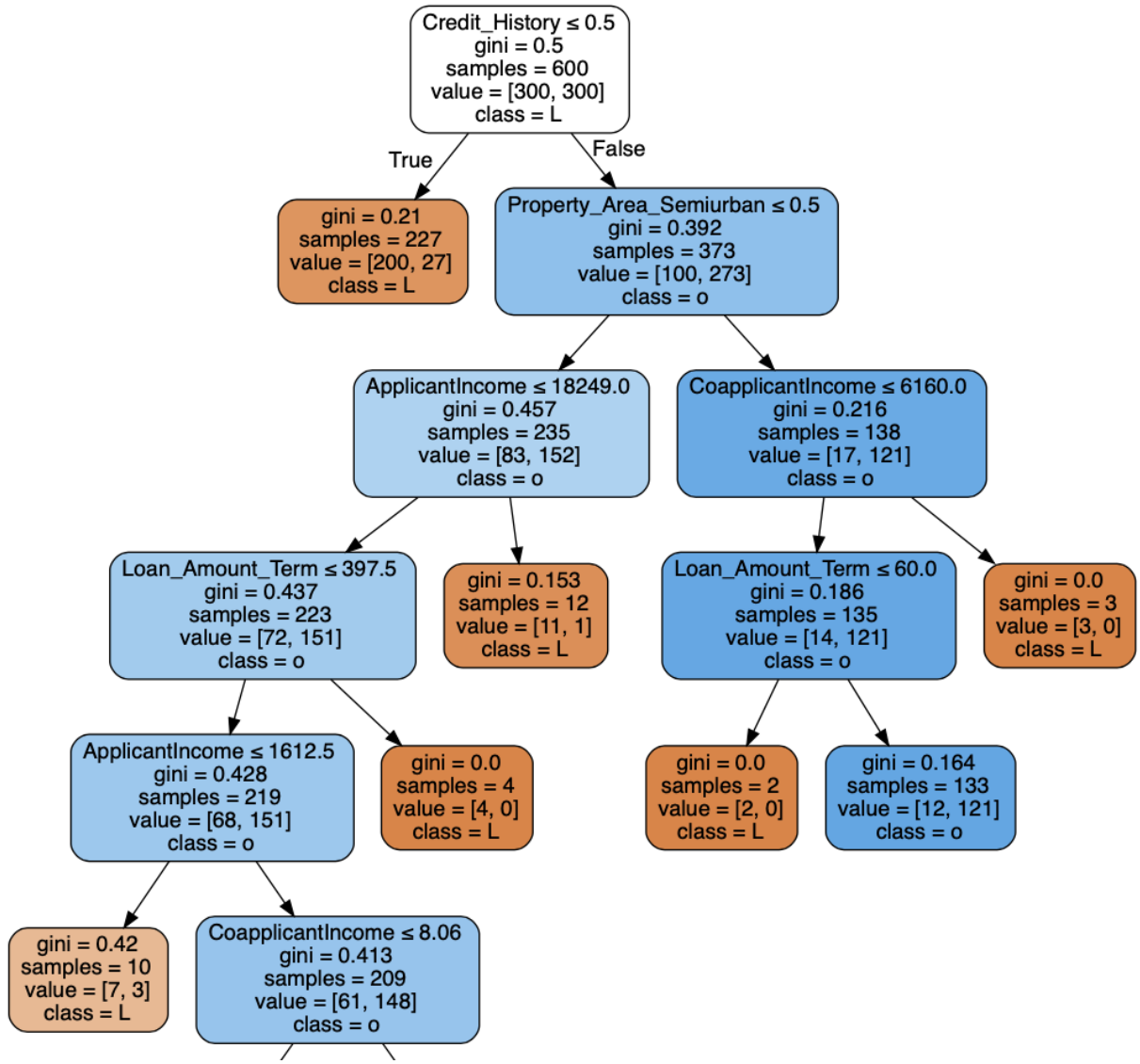


- We saw that using other hyperparameters did not make a significant difference

and so to avoid overfitting, we used the selected features with max_depth and max_leaf_nodes using the values found on the gridSearchCV

- Our final model was trained using the above found most important features, criterion=gini (default), max_depth=7, and max_leaf_nodes=10 . This gave us a training accuracy of 1 with a score of .812, partially represented below.

```
Final Decision Tree:

Depth: 7 Number of leaves: 10
```

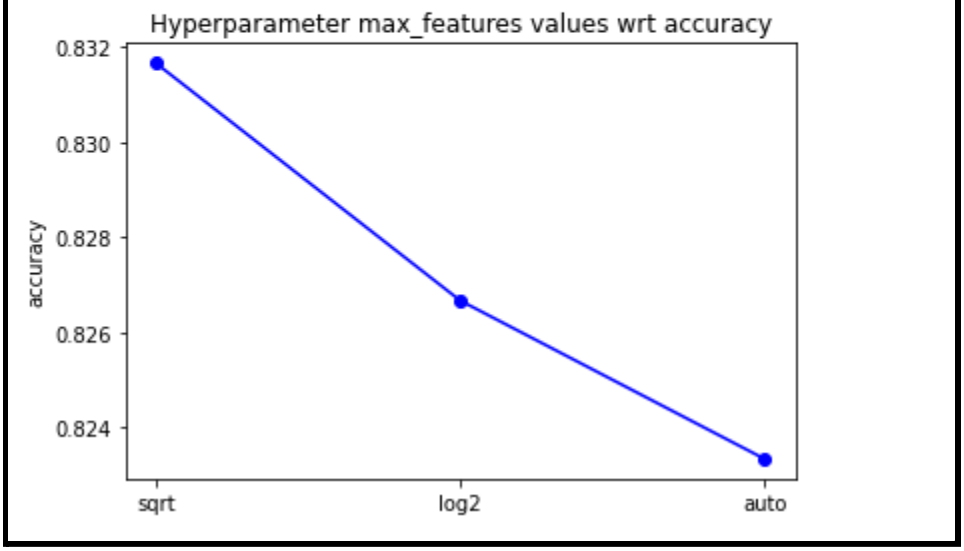


Random Forest Classifier

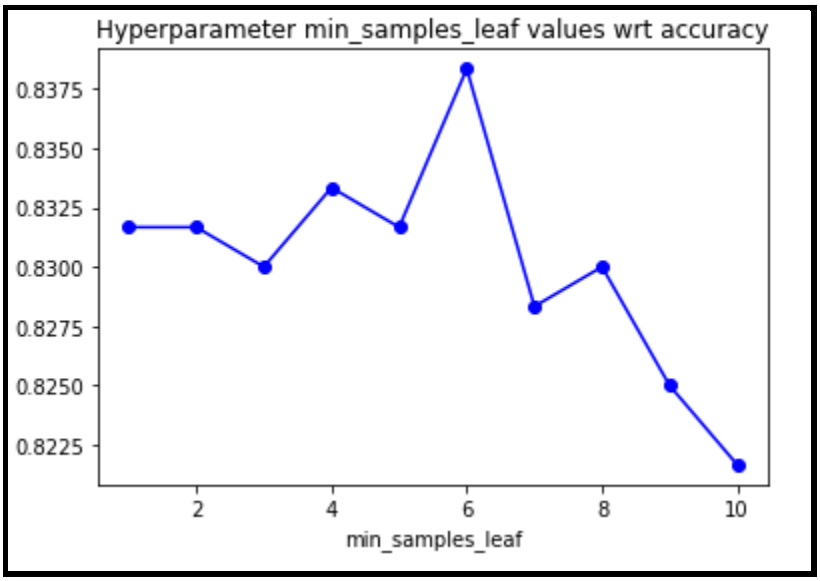
- We fit the upsampled(using SMOTE) training dataset on the random forest model using RandomForestClassifier from sklearn.ensemble package.
- Next, we used GridSearchCV and cross-validation to find the best values for hyperparameters (criterion and n_estimators).

    - k-Fold cross-validation : k=6
    - Number of iterations : 100
    - Used the mode or the most commonly returned hyperparameter value using GridSearchCV
    - Criterion vs count returned →'gini': 47, 'entropy': 53. We chose entropy.
    - The 2 most common values for n_estimators was → 50,100. 100 is the default value and in practice 100 gives better results, so we chose 100.
- Even though we got our values through GridSearchCV, we performed a few other experiments and generated plots to see if it improves the performance.

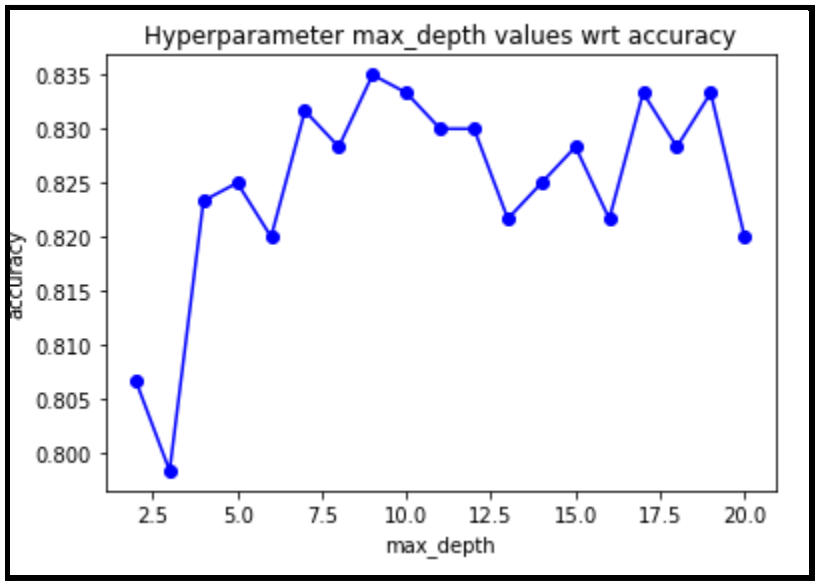
max_features	Accuracy	std
sqrt	0.8316666666666667	0.021147629234082508
log2	0.8266666666666667	0.023570226039551556
auto	0.8233333333333334	0.025603819159562



min_samples_leaf	Accuracy	std
1	0.8316666666666667	0.036247605284885895
2	0.8316666666666666	0.020344259359556142
3	0.8300000000000001	0.03214550253664317
4	0.8333333333333334	0.02357022603955157
5	0.8316666666666667	0.021147629234082515
6	0.8383333333333333	0.024776781245530836
7	0.8283333333333333	0.03435921354681382
8	0.8300000000000001	0.032659863237109024
9	0.8250000000000001	0.030956959368344503
10	0.8216666666666667	0.03387066905483594

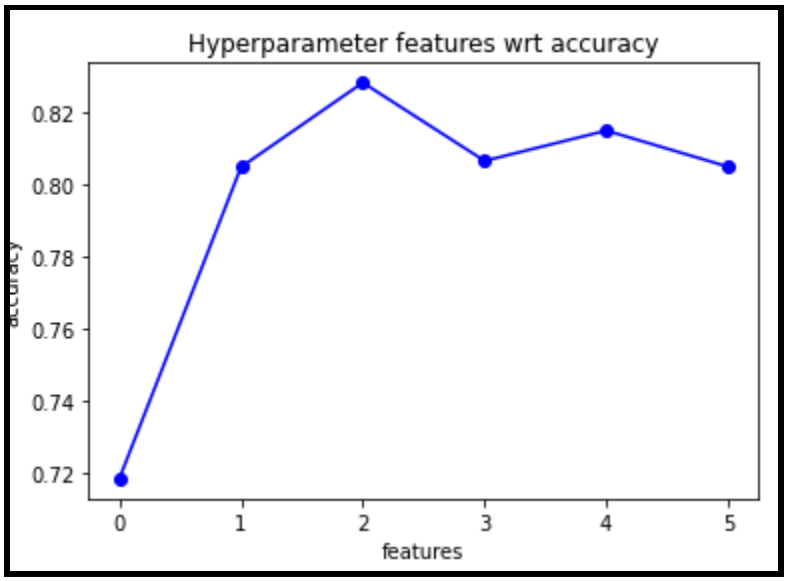


max_depth	Accuracy	std
2	0.8066666666666666	0.04459696053419883
3	0.7983333333333333	0.04297932319409208
4	0.8233333333333333	0.04714045207910316
5	0.8249999999999998	0.0419324854180304
6	0.82	0.03999999999999999
7	0.8316666666666667	0.027938424357066994
8	0.8283333333333333	0.03578485092263979
9	0.8350000000000001	0.028136571693556864
10	0.8333333333333334	0.037712361663282526
11	0.83	0.031091263510296032
12	0.83	0.02516611478423581
13	0.8216666666666668	0.033374973990834625
14	0.8250000000000001	0.03730504880933231
15	0.8283333333333333	0.03387066905483594
16	0.8216666666666668	0.030776975521032292
17	0.8333333333333334	0.03543381937578215
18	0.8283333333333333	0.030230595245361744
19	0.8333333333333334	0.038151743807531974
20	0.82	0.02380476142847614



- We saw that using other hyperparameters did not make a significant difference and so to avoid overturning, we used n_estimators = 100 and criterion=entropy
- Next, we tried feature importances and found `['LoanAmount','Credit_History','ApplicantIncome','CoapplicantIncome','Loan_Amount_Term']` to be the most important features as seen in the plots below.

Features	Accuracy
['LoanAmount', 'Credit_History']	0.7186046511627906
['LoanAmount', 'Credit_History', 'ApplicantIncome', 'CoapplicantIncome']	0.805061559507524
['LoanAmount', 'Credit_History', 'ApplicantIncome', 'CoapplicantIncome', 'Loan_Amount_Term']	0.8284737150674223
['LoanAmount', 'Credit_History', 'ApplicantIncome', 'CoapplicantIncome', 'Loan_Amount_Term', 'Property_Area_Rural']	0.8067617744772327
['LoanAmount', 'Credit_History', 'ApplicantIncome', 'CoapplicantIncome', 'Loan_Amount_Term', 'Property_Area_Rural', 'Gender']	0.8151651358217707
['LoanAmount', 'Credit_History', 'ApplicantIncome', 'CoapplicantIncome', 'Loan_Amount_Term', 'Property_Area_Rural', 'Gender', 'Married']	0.8051397303107289

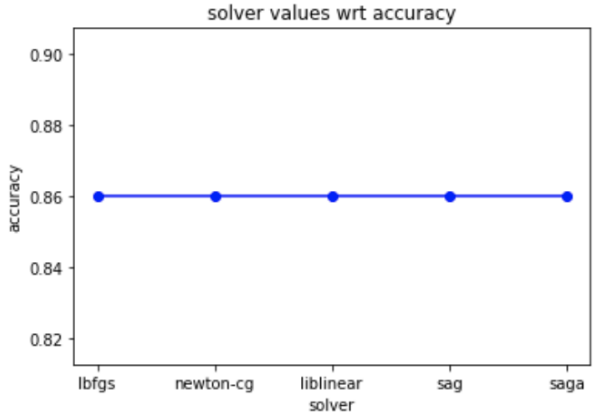


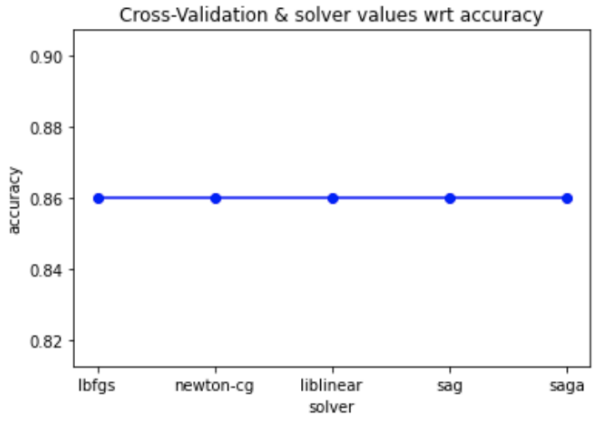
- Our final model was trained using the above found most important features, n_estimators = 100 and criterion=entropy. This gave us a training accuracy of 1.

Logistic Regression

- For the Logistic Regression Model we should scale the data before running the algorithm, since this is very sensitive to the data range and takes longer to converge for data with different scales. To make sure that we'll use the same scaling for training and testing datasets, we create a pipeline using the library `sklearn.pipeline` and apply StandardScaler from the `sklearn.preprocessing` library.
- Then we perform some experiments to decide what is the best k for the cross validation, and other hyperparameters.

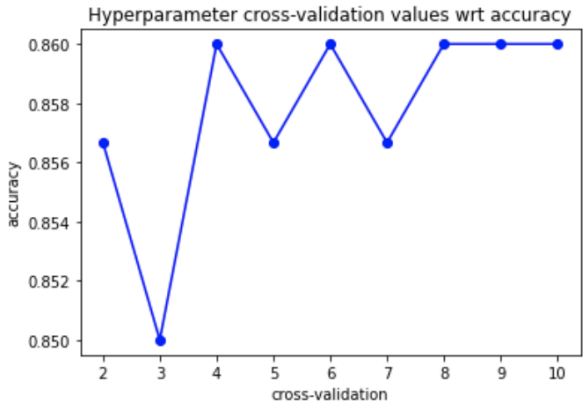
solver	LR acc	LRCV acc
lbfgs	0.86	0.86
newton-cg	0.86	0.86
liblinear	0.86	0.86
sag	0.86	0.86
saga	0.86	0.86





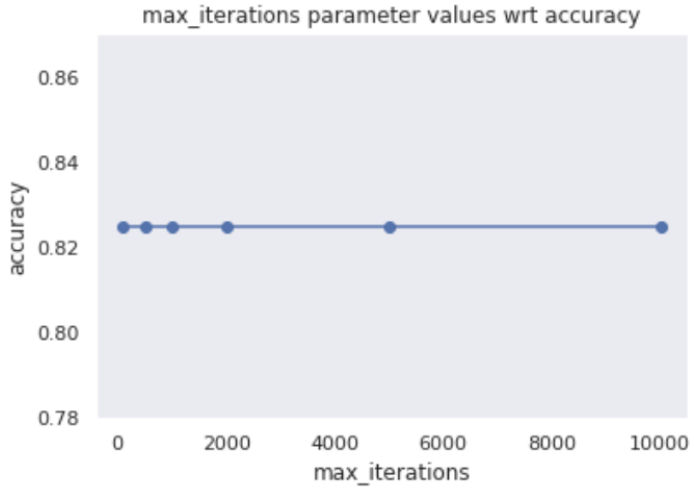
Model 0:

cv	accuracy
2	0.8566666666666667
3	0.85
4	0.86
5	0.8566666666666667
6	0.86
7	0.8566666666666667
8	0.86
9	0.86
10	0.86



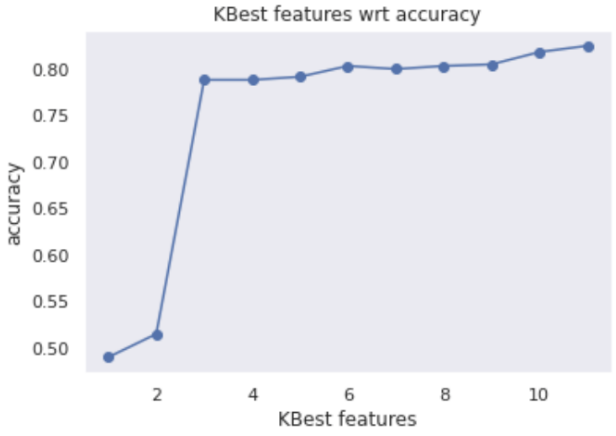
- We can see that apart from 'liblinear', all other solvers ‘newton-cg’, ‘sag’, ‘saga’ and ‘lbfgs’ have the same accuracy when using CV. Since our data set is small, ‘newton-cg’ and 'lbfgs' are appropriate to use. We decided to choose 'lbfgs', the default solver, because it only stores the last few updates and saves memory.
- Max_iteterations don’t improve the results.

max_iterations	Accuracy
100	0.825
500	0.825
1000	0.825
2000	0.825
5000	0.825
10000	0.825



- From the library we imported SelectKBest and chi2 to select the important features for our model, but it did not improve the model performance.

KBest features	Accuracy
1	0.49
2	0.515
3	0.7883333333333333
4	0.7883333333333333
5	0.7916666666666666
6	0.8033333333333333
7	0.8
8	0.8033333333333333
9	0.805
10	0.8183333333333334
11	0.825



- Based on the results of the tests above, we decided on the following hyperparameters values of cv = 4-fold and default solver= 'lbfgs'.

Model evaluation

For each ML algorithm we computed the results of True Positive - TP, False Positive - FP, True Negative - TN, and False Negative - FN. Then, we used these values to evaluate our models using confusion matrix, accuracy,precision, recall and F1-score.

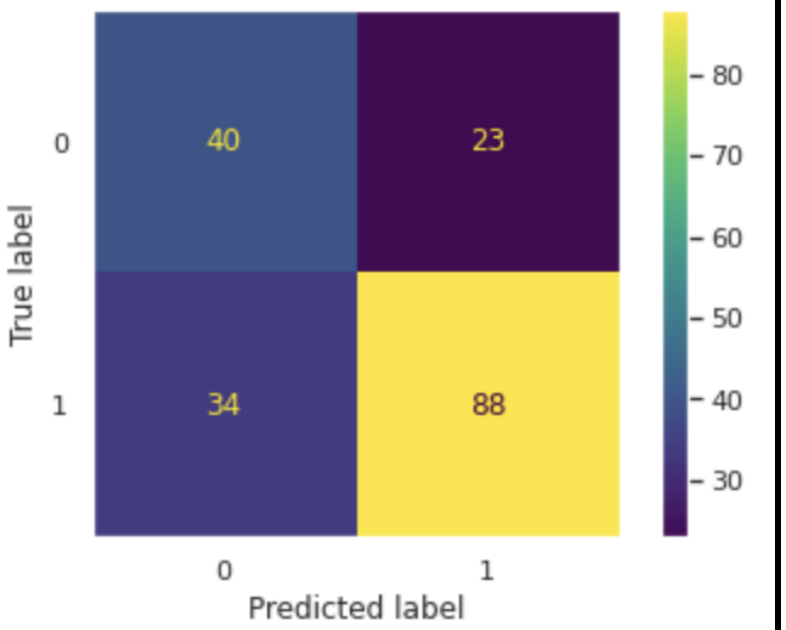
1. Confusion Matrix -> Cij is equal to the number of observations known to be in group 'i' and predicted to be in group 'j'. Confusion matrix whose i-th row and j-th column entry indicates the number of samples with true label being i-th class and predicted label being j-th class
2. Accuracy -> a fraction of correctly classified samples over the total samples on dataset (% total correct predictions)
3. F1 score -> The F1 score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0 (F1 = 2 * (precision * recall) / (precision + recall))

4. Precision -> the fraction of correctly classified positive over the total predicted positive (measures how precise the predictions are)
5. Recall -> the fraction of correctly classified positive over the total true positive (indicates what percentage of the classes we're interested in were actually captured by the model)

Decision Tree Classifier

Our test accuracy using decision tree classifier was 69%, with recall for granted loans of 79%

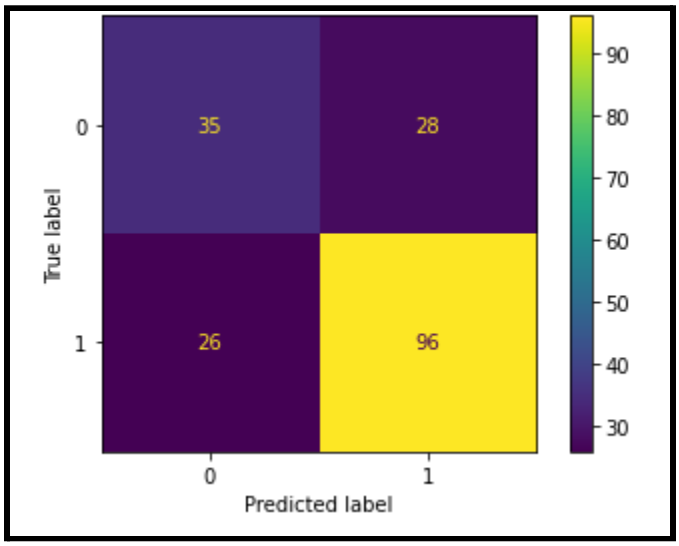
Confusion Matrix:



	precision	recall	f1-score	support
0	0.63	0.54	0.58	74
1	0.72	0.79	0.76	111
accuracy			0.69	185
macro avg	0.68	0.67	0.67	185
weighted avg	0.69	0.69	0.69	185

Random Forest Classifier

Our test accuracy using random forest classification was 71%, with recall for granted loans of 77%.

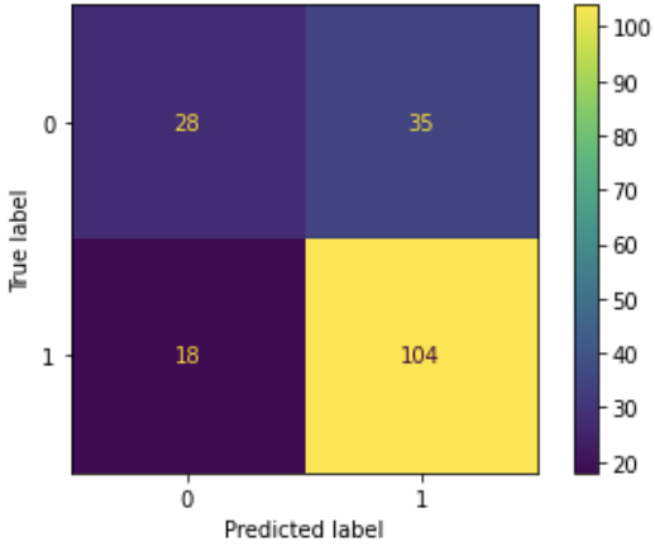



```
Accuracy: 0.708

F1 score: [0.56451613 0.7804878 ]
              precision    recall  f1-score   support

           0       0.56      0.57      0.56        61
           1       0.79      0.77      0.78       124

    accuracy                           0.71       185
   macro avg       0.67      0.67      0.67       185
weighted avg       0.71      0.71      0.71       185
```

Logistic Regression

Our test accuracy using logistic regression was 71%, with recall for granted loans of 75%.

Confusion Matrix:



	precision	recall	f1-score	support
0	0.44	0.61	0.51	46
1	0.85	0.75	0.80	139
accuracy			0.71	185
macro avg	0.65	0.68	0.66	185
weighted avg	0.75	0.71	0.73	185

Conclusion

After data preprocessing and applying different models to our dataset, none of our models produced high scores on the measured metrics for this problem. We modified our models to use less hyperparameters, since our previous models with more fine-tuning were overfitting the model. In the end, Logistic Regression produced slightly better results than Random Forest.

Thoughts on Artificial Intelligence before and after this class

Before this class our knowledge on Artificial Intelligence was very limited. Although one of us had a Machine Learning class before, it was only scratching the surface on names

and concepts, but far from understanding them. It was like magic before, but now that we learned how these algorithms work it is quite fascinating. We know that there is still a lot to learn, but we both feel that now we have a strong foundation that will help us to build on top. This class transformed us, and has also inspired us to do our thesis in this field.

Prior Researches

Loan data is predicted by using some machine learning algorithms like classification, logic regression, Decision Tree and gradient boosting [1].The analytical process started from data cleaning and processing, Missing value imputation with mice package, then exploratory analysis and finally model building and evaluation. The best accuracy on the public test set is 0.811. Decision tree algorithm is effective because it provides better results in classification problems. It is extremely intuitive, easy to implement and provides interpretable predictions. It produces out of bag estimated error which was proven to be unbiased in many tests. It is relatively easy to tune with. It gives the highest accuracy result for the problem.

The random forest algorithm is adopted to build a model for predicting loan default in the lending club.The SMOTE method is adopted to cope with the problem of imbalance class in the dataset, and then a series of operations such as data cleaning and dimensionality reduction are carried out.[2] Results are compared with other three algorithms of logistic regression, decision tree and support vector machine. The experiment shows that the random forest algorithm performs outstandingly better than the other three algorithms in the prediction of loan default and has strong ability of generalization.

[3] This paper used Naïve Bayes, Decision tree (unpruned and pruned) and Random Forest classifiers to build loan default prediction models.This paper also applied several data preprocessing techniques, and compared between three features selection algorithms: Information Gain, Genetic Algorithm and Particle Swarm Optimization. It can be concluded that the data preprocessing stage is an important stage when building a classification model, as it has a valuable impact on the quality of the model's outcomes and accuracy. Applying features selection algorithms is very significant as well when having a large dataset. Not only does it enhance accuracy, but also it improves the performance of the model.

References

[1] P. Usha Supriya, M. Pavani, Nagarapu Saisushma, Nam-buri Vimala Kumari, and Khullar Vikas. Loan prediction by using machine learning models. 2019.

[2] Lin Zhu, Dafeng Qiu, Daji Ergu, Cai Ying, and Kuiyi Liu. A study on predicting loan default based on the random forest algorithm. *Procedia Computer Science* 162:503–513, 2019. 7th International Conference on Information Technology and Quantitative Management (ITQM 2019): Information technology and quantitative management based on Artificial Intelligence.

[3] Ahmad al Qerem, Ghazi Al-Naymat, Mays Alhasan, and Mutaz Al-Debei. Default prediction model: The significant role of data engineering in the quality of out-comes.*The International Arab Journal of Information Technology* 17:635–644, 07 2020.