MS4610: Introduction to Data Analytics | AMEX PROJECT

Developing a credit risk model that predicts the defaulting likelihood of a customer

Team Name: MARSSS Group-12

A Project Report by:-

CH18B067	Shania Mitra
CH19B045	Anshika Gulati
CH19B062	Richi Kothari
CH19B067	Mayur Mohan Mate
CH19B088	Shrey K. Patel
CH19B094	Swathi G

TABLE OF CONTENTS:

PROBLEM STATEMENT

Financial institutions use credit risk analysis models to determine the probability of default of a potential borrower. The models provide information on the level of a borrower's credit risk at any particular time. If the lender fails to detect the credit risk in advance, it exposes them to the risk of default and loss of funds. Lenders rely on the validation provided by credit risk analysis models to make key lending decisions on whether or not to extend credit to the borrower and the credit to be charged.

Aim: To develop a credit risk model that predicts the likelihood of a customer to default a payment. The Data present for a customer is at any given point of time. To develop a model to predict the likelihood of the customer defaulting after 12 months.

DATA VISUALIZATION AND CLEANING

	application_key	mvar1	mvar2	mvar3	mvar4	mvar5	mvar6	mvar7	mvar8	mvar9	...	mvar39	mvar40	mvar41	mvar42	mvar43	mvar44	mvar45
0	230032	1696	1.6541	0.000	0.0	0.0	0	6015	322	40369	...	1	73.78	82.547	0.08696	10	0.63899	na
1	230033	1846	0.8095	0.000	0.0	0.0	102	7532	3171	18234	...	0	99.129	missing	0	13	0.63836	na
2	230034	1745	0.4001	0.000	0.0	0.0	missing	2536	missing	missing	...	0	missing	29.29	0	1	1.00000	na
3	230035	1739	0.2193	0.000	0.0	0.0	1982	26440	4955	20316	...	0	96.272	missing	0.15385	3	0.53241	0
4	230036	1787	0.0118	0.225	0.0	0.0	5451	5494	5494	7987	...	0	115.019	missing	0	1	0.92665	na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
82995	578064	1748	0.3044	0.000	0.0	0.0	0	20114	5574	77386	...	0	missing	8.445	0.04348	13	0.48002	0
82996	578065	1846	NaN	0.000	0.0	0.0	793	18608	18608	63820	...	0	missing	missing	0	5	0.46925	na
82997	578066	1907	0.0381	0.000	0.0	0.0	2478	24775	24775	100294	...	0	137.164	42.857	0	19	0.67960	na
82998	578067	1744	1.8301	0.000	0.0	0.0	496	917	496	116164	...	0	missing	missing	0.09375	3	0.45317	na
82999	578068	1832	1.5561	0.000	0.0	0.0	1016	7928	7928	132208	...	0	missing	67.1	0	18	0.83278	0

83000 rows × 49 columns

Fig.1: Raw Data

The raw dataset has 47 independent features and 83,000 data points. The target is in the column 'default_ind' and it is binary categorical. 0 (zero) indicates no default, whereas 1 (one) indicates that the user has defaulted.

Visual observation shows that there are lots of missing values as seen in Fig.1. The first step is to replace these missing values with a null value 'numpy.nan'.

Taking a look at the Class Distribution:

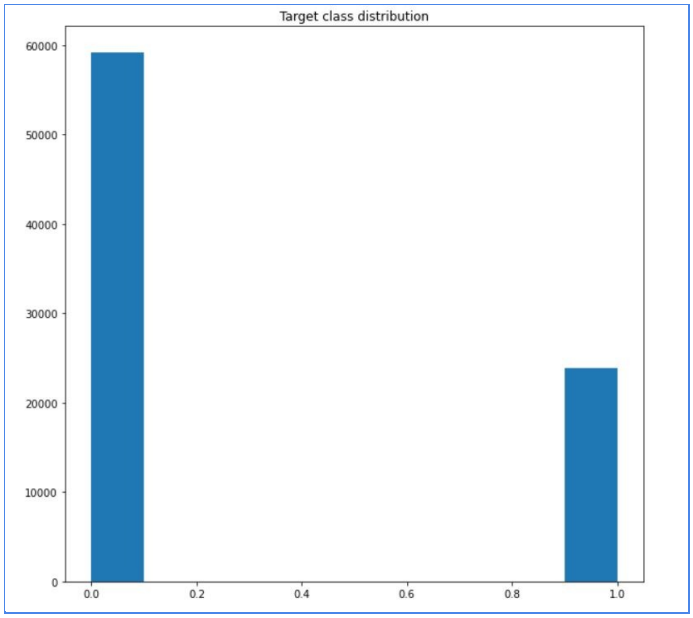


Fig.2: Class Distribution

As we can observe in Fig.2, we have more than twice the class0 data points as compared to class1.

This implies that the dataset is imbalanced. This can negatively affect the performance of Machine Learning Algorithms, as they more frequently get exposed to class0 data points as opposed to class1.

We can resolve this issue by providing appropriate weights to the respective classes.

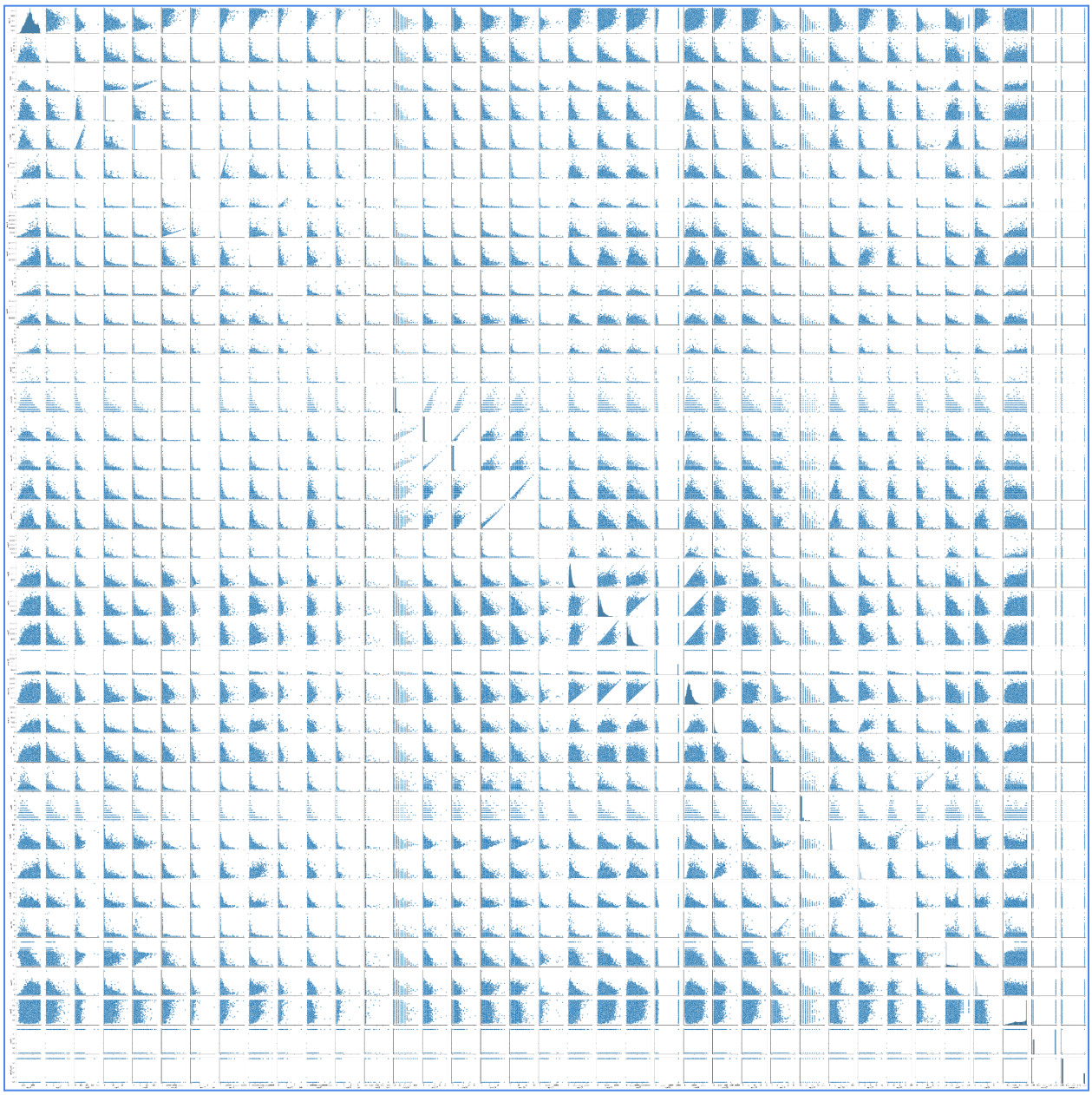


Fig.3: Scatter plots between each feature-pair (Please zoom-in the above image)

FEATURE CORRELATION

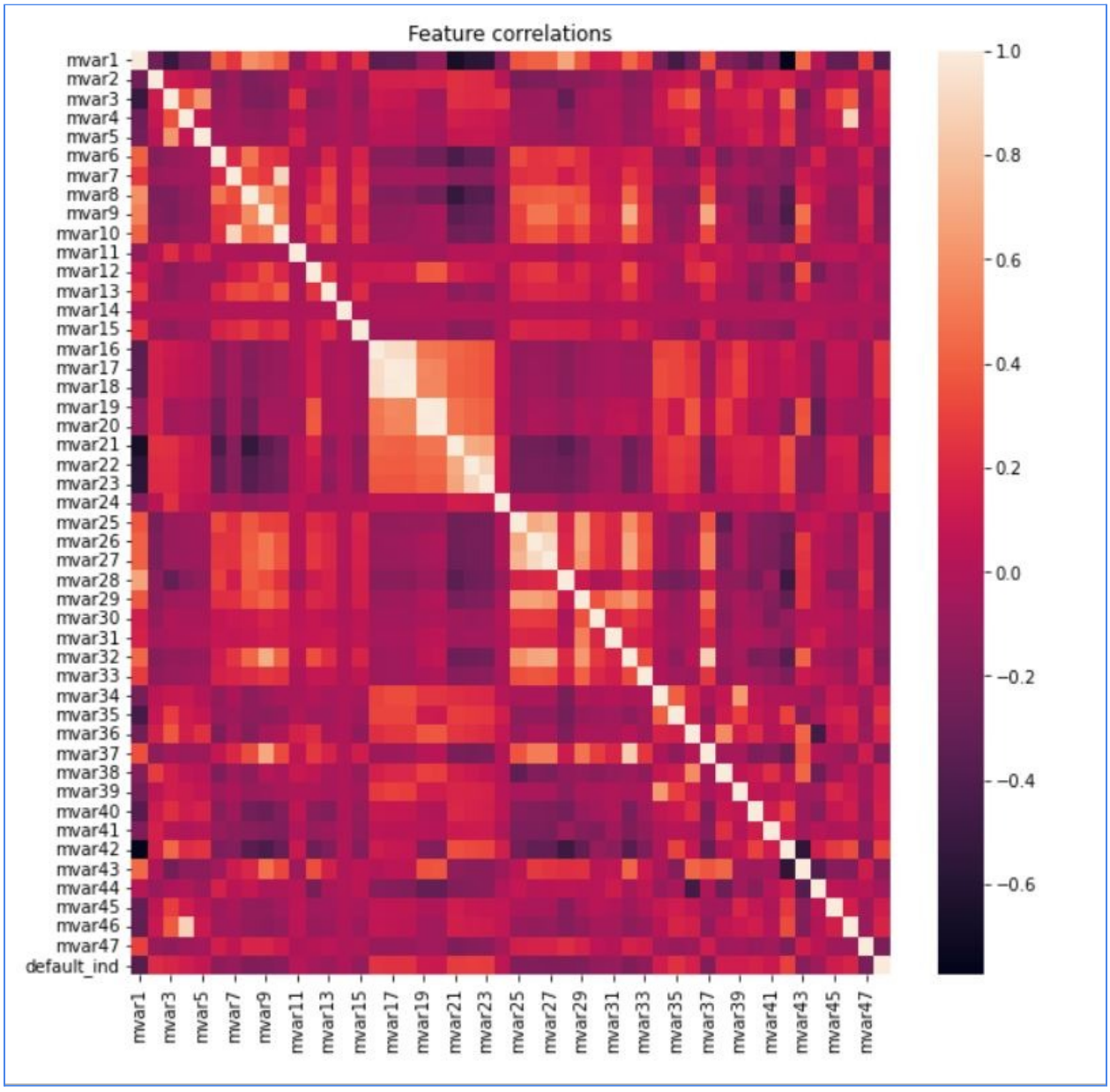


Fig.4: Heat-map for correlations amongst features

This heat-map provides a visual way to look at the strength of linear dependence amongst various features. It is noticed that some features have a very high correlation. Hence, there is multicollinearity in the dataset. Although multicollinearity doesn't affect the predictive

Feature1	Feature2	Correlation
mvar1	mvar42	-0.774
mvar4	mvar46	0.881
mvar7	mvar10	0.892
mvar9	mvar32	0.718
mvar16	mvar17	0.937
mvar16	mvar18	0.928
mvar17	mvar18	0.989
mvar19	mvar20	0.985
mvar21	mvar22	0.703
mvar22	mvar23	0.894
mvar25	mvar26	0.708
mvar25	mvar27	0.755
mvar26	mvar27	0.913
mvar32	mvar37	0.866

Fig.5: Strongly Correlated Features

accuracy, it can undermine the importance of certain features when statistical tests for feature importance are carried out.

We kept a correlation threshold of 0.7. If features had an absolute correlation greater than this threshold, they were deemed strongly correlated. Fig.5 shows the features which are strongly correlated. Some of these features were removed while feature importances were derived.

FEATURE SELECTION

In this study, all features with >30% null values are dropped so that imputation does not lead to additional noise.

Feature	Non-null values
mvar11	36283
mvar15	49481
mvar22	52332
mvar23	40689
mvar30	45012
mvar31	24461
mvar35	48132
mvar40	17930
mvar41	25736
mvar45	37080

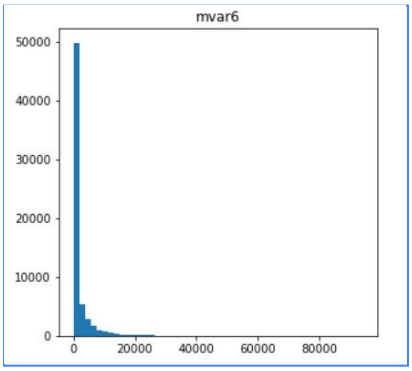
These features have very few non-null values. Imputing them can introduce significant bias in the data. Hence, we decide to drop them.

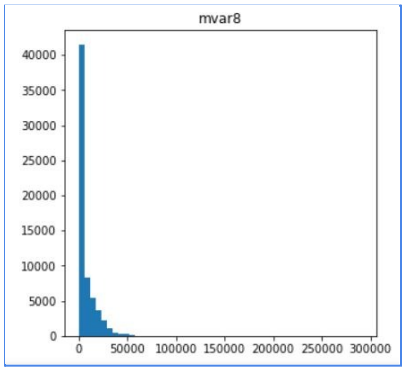
We have considered the columns with less than 30 unique values as categorical, and the rest, numerical.

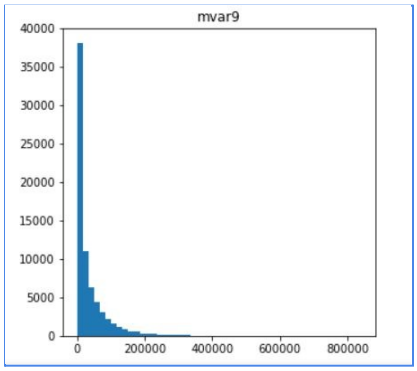
Feature	Unique values
mvar16	13
mvar17	16
mvar18	16
mvar19	27
mvar20	29
mvar34	21
mvar35	11
mvar39	20
mvar45	9
mvar46	7
mvar47	2

Out of 83,000 values, these features had less than 30 unique values and hence, these are considered categorical.

DISTRIBUTION OF NUMERICAL FEATURES



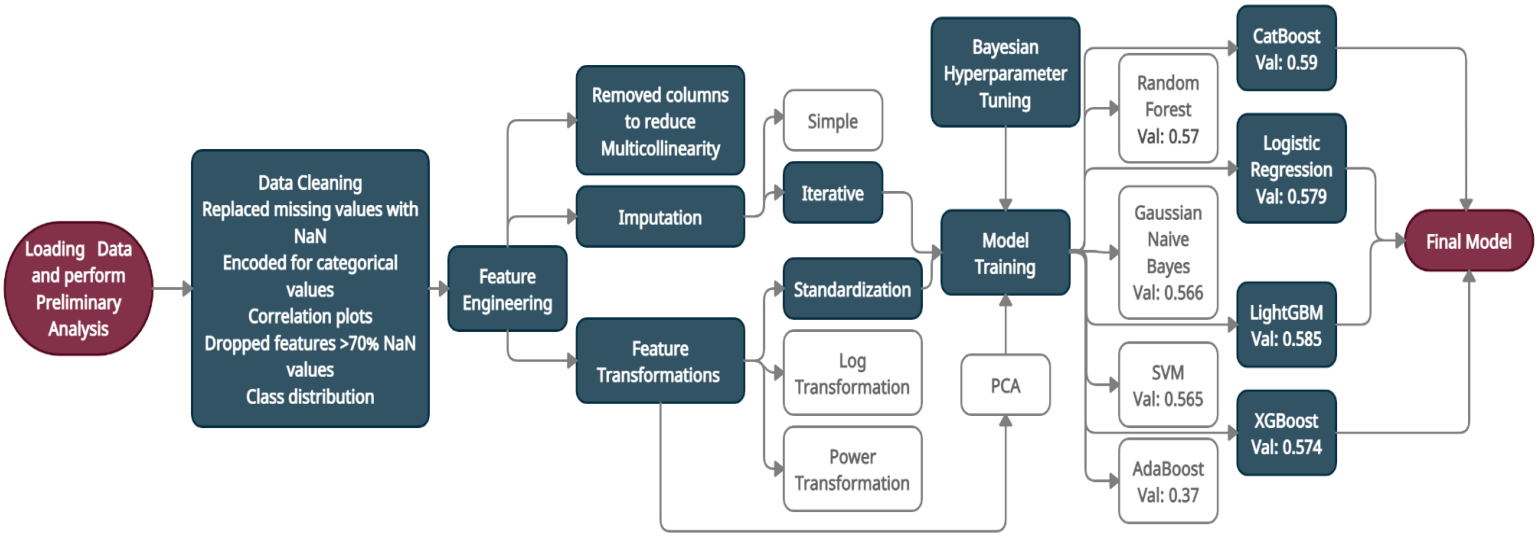




We observe that a lot of features have highly skewed distributions. We speculate that this can cause a problem for the learning algorithms as they get exposed to lower values more frequently than the higher ones.

Hence, we decide to try out some feature transformations to get the distributions closer to Gaussian and see whether we get any improvements.

OVERVIEW OF APPROACH



INPUT PIPELINE

A pipeline is a sequence of transformations through which the data flows. These transformations are carried out by special objects called transformers. Raw data cannot be directly used for training the model as it contains missing values and features of different scales. This can cause numerical issues. We set up a pipeline in the following fashion:

- **Imputation**
    - Since we have missing values, we decide to use imputation to fill them.
    - Two possible strategies:
        - Simple Imputation: Fill the missing values with a statistic such as Mean, Median or Mode.
        - Iterative Imputation: Predict the missing values by regressing with the rest of the features.
- **Feature Scaling**
    - Strategies:

- Standardization: Subtracting Mean and dividing by the Variance to get all the features to 0 Mean and 1 Variance. This is important for the numerical stability of the learning algorithms.
- Log Transformation: Since we observed a lot of features had skewed distribution, Log Transformation can help reduce the skewness and bring the distribution closer to Gaussian. This can sometimes improve the model performance.
- Power Transformation: Advanced feature transformations like the Yeo-Johnson transformation can reduce skewness.

HYPERPARAMETER TUNING

Present-day Machine Learning models tend to have multiple tunable hyperparameters. These are parameters that are not learnt by the algorithm and are set by the user, but can have profound impact on the performance of the model. The optimal set of hyper-parameters has to be searched through. There are 3 common techniques:

- **Grid Search**: Grid-searching is the process of scanning the data to configure optimal parameters for a given model. Depending on the type of model utilized, certain parameters are necessary. Grid-searching can be applied across machine learning to calculate the best parameters to use for any given model. It is important to note that Grid-searching can be extremely computationally expensive and may take your machine quite a long time to run. Grid-Search will build a model on each parameter combination possible. It iterates through every parameter combination and stores a model for each combination.
- **Randomized Search**: Random Search replaces the exhaustive enumeration of all combinations by selecting them randomly. This can be simply applied to the discrete setting described above, but also generalizes to continuous and mixed spaces. It can outperform Grid search, especially when only a small number of hyperparameters affects the final performance of the machine learning algorithm. In this case, the optimization problem is said to have a low intrinsic dimensionality.
- **Bayesian Search**: Bayesian optimization is a global optimization method for noisy black-box functions. Applied to hyperparameter optimization, Bayesian optimization

builds a probabilistic model of the function mapping from hyperparameter values to the objective evaluated on a validation set. By iteratively evaluating a promising hyperparameter configuration based on the current model, and then updating it, Bayesian optimization aims to gather observations revealing as much information as possible about this function and, in particular, the location of the optimum. It tries to balance exploration (hyperparameters for which the outcome is most uncertain) and exploitation (hyperparameters expected close to the optimum). In practice, Bayesian optimization has been shown to obtain better results in fewer evaluations compared to grid search and random search, due to the ability to reason about the quality of experiments before they are run.

Upon experimentation with each of these methods, we found Bayesian Optimization to work most efficiently and converge the fastest. Thus, we proceed with Bayesian Optimization for all further experiments.

METRICS USED FOR MODEL SELECTION

We observe that the data is imbalanced, as can be seen in Fig. 2. Thus, this makes accuracy an unsuitable metric since even if the model predicts all samples to be of class 0, the baseline accuracy would be around 71%, which gives us the false impression that the model is performing well.

Precision tells us the number of correctly predicted classes out of all the samples the model predicts while Recall tells us the number of positive classes correctly predicted by the model out of all the positive classes in the data set. For a model to perform well, it must balance well between Precision and Recall. Thus, we use F1-Score which is the Harmonic Mean of Precision and Recall. A high F1-score ensures that both Precision and Recall are high.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

$$F_1\text{-score}$$

$$\frac{1}{\frac{1}{2}\left(\frac{1}{\text{recall}} + \frac{1}{\text{precision}}\right)}$$

Cross Validation

Cross-validation is a technique used to protect against overfitting in a predictive model, particularly in a case where the amount of data may be limited. In cross-validation, you make a fixed number of folds (or partitions) of the data, run the analysis on each fold, and then average the overall error estimate. In this study, we use **5-fold cross validation** to prevent our models from overfitting.

MODEL TRAINING

```
['mvar11',
 'mvar15',
 'mvar22',
 'mvar23',
 'mvar30',
 'mvar31',
 'mvar35',
 'mvar40',
 'mvar41',
 'mvar45',
 'mvar42',
 'mvar46',
 'mvar7',
 'mvar16',
 'mvar17',
 'mvar20',
 'mvar26',
 'mvar27',
 'mvar32']
```

In this section, we highlight the steps taken to train various models along with the theory behind the models used. Initially, feature selection is performed, where features with <30% non-null values and strongly correlated features are removed. This is done since, imputation of features with a large number of null-values would lead to addition of noise, while retaining highly correlated features may lead to unstable models.

Models and Approaches:

1. Logistic Regression

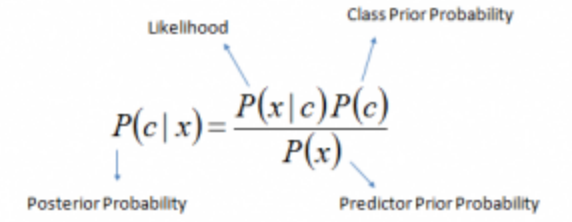
Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, logistic regression is a predictive analysis. Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables. The output from the logistic function is the probability of the point belonging to class 1 (P)

$$P = \frac{e^{a+bX}}{1+e^{a+bX}}$$

2. Gaussian Naive Bayes

Naive Bayes is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

The Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.



$$P(c \mid \mathbf{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Above,

- $P(c|x)$ is the posterior probability of *class* (c, *target*) given *predictor* (x, *attributes*).
- $P(c)$ is the prior probability of *class*.
- $P(x|c)$ is the likelihood which is the probability of the predictor given *class*.

- $P(x)$ is the prior probability of the predictor.

3. Support Vector Machine

Support Vector Machine is a non-probabilistic classifier. It tries to find a decision boundary in the feature vector space to maximize the width of the gap between two different classes. It is one of the most robust sets of algorithms and can be easily extended to perform non-linear classification tasks via the kernel trick by implicitly mapping the inputs into a higher dimensional space.

4. Random Forest

Random Forest is one of the ensemble learning methods used for classification problems like this. It is based on decision trees and the outcome is predicted by taking the mean of the output from many decision trees. Random forests help in cases of overfitting and increase precision of prediction. It is particularly suited for this dataset as it provides an effective way to handle missing data and work with large datasets that have many features.

5. Adaboost

Adaptive Boosting is also one of the ensemble learning methods used for classification problems like this. It is an adaptive method as it's weights are re-assigned to each instance and incorrectly classified instances are penalised by assigning higher weights. It is also used when there is a need to reduce bias and variance both for a dataset. The learners in AdaBoost grow sequentially. This means that every subsequent learner grows from the previous one and becomes stronger. It is an effective algorithm based boosting technique.

6. LightGBM

LightGBM is also a gradient boosting technique with tree based learning algorithms. It is considered to be a fast processing algorithm compared to others. While other algorithms trees grow horizontally, the LightGBM algorithm grows vertically meaning it grows leaf-wise and other algorithms grow level-wise. LightGBM chooses the leaf with large loss to grow. It can lower down more loss than a level wise algorithm when growing the same leaf.
The model consists of the following hyperparameters that are obtained using Bayesian Optimization:

- Max depth: It gives the depth of the tree and also controls the overfitting of the model. If you feel your model is getting overfitted lower down the max depth.
- Min_data_in_leaf: Leaf minimum number of records also used for controlling overfitting of the model.
- Feature_fraction: It decides the randomly chosen parameter in every iteration for building trees. If it is 0.7 then it means 70% of the parameter would be used.
- Bagging_fraction: It checks for the data fraction that will be used in every iteration. Often, used to increase the training speed and avoid overfitting.
- Early_stopping_round: If the metric of the validation data does show any improvement in last early_stopping_round rounds. It will lower the imprudent iterations.
- Lambda: It states regularization. Its values range from 0 to 1.
- Min_gain_to_split: Used to control the number of splits in the tree.

A significant advantage with this model is that it does not require missing values to be imputed. Missing values are handled by associating them with values that reduce the overall loss.

7. XGBoost

XGBoost is a decision tree based algorithm that falls under the ensemble class of methods. It is based on gradient boosting techniques and is considered the most powerful algorithm for prediction and classification based problems in recent times. Apart from its similarity to other decision tree based algorithms, it is popular for its compatibility with hardware and other combinations on the software end that make it computationally efficient. Like LGBM, this too handles missing values without the user having to impute them.

8. CatBoost

CatBoost, short for Category Boosting, is an algorithm that is based on decision trees and gradient boosting like XGBoost. CatBoost does especially well with data containing “categorical variables.”. In other models, categorical variables are handled through “OneHotEncoding” which creates additional columns to capture the information. Alternatively, CatBoost starts by shuffling the data, creating “permutations”.For each, it assigns a “default” value for each class to the first few examples. Next, it calculates the value in each new row by looking at previous examples with the same class, and counting

the number of positive labels, then performing a calculation.Then the model proceeds by building “symmetric binary trees” for each permutation of the data.To avoid overfitting, CatBoost builds new models at each step (n), by shuffling the rows and looking at n^2 previous examples. CatBoost also automatically handles hyperparameter tuning and can even handle missing values like LightGBM.

RESULTS

Model Performances:

Model	Train	Validation
Logistic Regression	0.58	0.579
XGBoost	0.57	0.574
Light Gradient Boosting Machine	0.59	0.585
CatBoost	0.62	0.59
Support Vector Machine	0.57	0.565
Random Forest	0.576	0.57
Adaboost	0.48	0.37
Gaussian Naïve Bayes	0.567	0.566

Scaling used: Standard scaling

Imputation used: Iterative Imputation

Combinations of other scaling methods and imputations were tested but they yielded poorer performance. Hence, we settled with Standard Scaling and Iterative Imputation.

Observing Feature Importances of Best Performing Models:

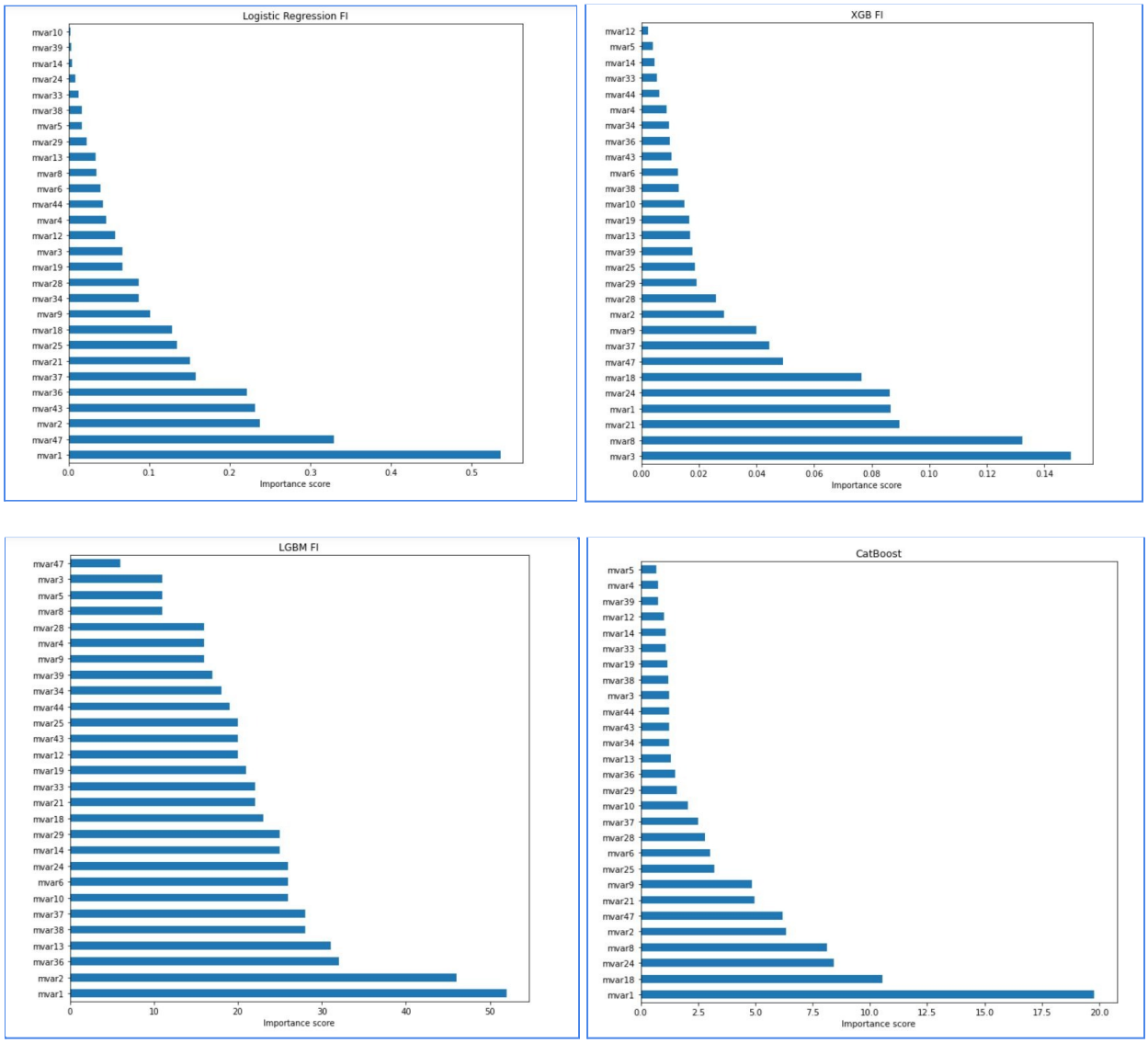


Fig.5. Feature Importance Plots for (a) **Logistic Regression** (b) **XGBoost** (c) **LightGBM** (d) **CatBoost**

Upon observing the feature importances of (a) Logistic Regression (b) XGBoost (c) LightGBM (d) CatBoost, we observe that 3 of the 4 best performing models report 'mvar1' as the most important feature in predicting the target. On retraining the models with a subset of the most important features as reported by the plots, we observed that they lead to a loss in predictive power.

Re-training with Addition of Strongly Correlated Features:

On retraining all models after adding the strongly correlated features we observe that LGBM, LR , SVM and GNB show an improvement in performance, while the remaining models do not show any significant improvement. Hence, in the final model, strongly correlated features are discarded.

Model	Validation (Without Correlated Features)	Validation(With Correlated Features)
Logistic Regression	0.579	0.58
XGBoost	0.574	0.572
Light Gradient Boosting Machine	0.585	0.586
CatBoost	0.59	0.585
Support Vector Machine	0.565	0.57
Random Forest	0.57	0.57
Adaboost	0.37	0.39
Gaussian Naïve Bayes	0.566	0.571

Fig.6. Comparison table of validation F1 scores for with and without correlated features.

PRINCIPAL COMPONENT ANALYSIS (PCA)

Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance.

We tried training the most promising models on 10, 15, 20, and 25 principal components. Here are the results:

Model Performances on 10 Principal Components

Models	Traning	Validation
Logistic Regression	0.525	0.524
XGBoost	0.529	0.526
Light GBM	0.535	0.531
CatBoost	0.583	0.542

Model Performances on 15 Principal Components

Models	Traning	Validation
Logistic Regression	0.553	0.553
XGBoost	0.538	0.536
Light GBM	0.557	0.5488
CatBoost	0.6753	0.5468

Model Performances on 20 Principal Components

Model Performances on 25 Principal Components

Models	Traning	Validation
Logistic Regression	0.5589	0.5587
XGBoost	0.5314	0.5285
Light GBM	0.6315	0.5624
CatBoost	0.6102	0.5639

Models	Traning	Validation
Logistic Regression	0.57	0.57
XGBoost	0.56	0.56
Light GBM	0.63	0.57
CatBoost	0.65	0.58

Reducing the dimension of the data using PCA didn't provide any improvement on performance. In fact, all the principal components have to be used to maintain the performance.

Finally we tried combining all of the best models using the voting classifier. This model is an ensemble of several models. The prediction is made by taking a vote amongst all of the models combined. There are two ways of doing this: Hard Voting and Soft Voting.

- Hard Voting: The most frequent prediction across the models is chosen.
- Soft Voting: The prediction probabilities of each model are averaged and then the decision is made.

In this study, we combine the voting classifier with a soft voting strategy.

PERFORMANCE OF THE COMBINED VOTING CLASSIFIER

Confusion matrix:

	0	1
0	43515	15630
1	6574	17281

Mean Precision : 0.508 (Out of all the samples marked 1, the fraction of samples that actually defaulted)

Mean Recall : 0.701 (Fraction of actually defaulted samples that were detected)

Mean Accuracy : 0.718 (Fraction of samples correctly classified)

Mean F1 : 0.589 (Harmonic mean of Precision and Recall)

CONCLUSION

The best model we found is a combination of 4 models : - **Logistic Regression, XGBoost, CatBoost and Light GBM**. The models indicated "**mvar1**" as the most important feature in detecting 'default'. We achieved an average F1 score of **0.589**.

Standardization with the **Iterative Imputation** scheme worked the best.

Bayesian Hyperparameter tuning efficiently found a good set of hyperparameters.

Dimensionality reduction with **PCA** only led to a drop in performance.

BIBLIOGRAPHY

1. Hyperparameter Optimization
2. Cross-validation queries
3. Support Vector Machines
4. CatBoost Algorithm
5. Gradient Boosting with CatBoost
6. XGBoost Algorithm
7. Explanation of XGBoost
8. AdaBoost Algorithm
9. Creately for Overview pipeline
10. Official documentation for all the used Python Libraries