Indian Institute Of Technology Madras

MS4610 - Introduction to Data Analytics

Project : Customer Default Preiction

Group 3

AE17B109	Ananthu Nair S
BS18B001	Aditya Gupta
CE17B135	Volam Santhosh Kumar
BS18B010	Abhishek Kumar Raj
CE19B096	Tejal Meshram
BS18B022	Mohit Kushwah

13th December 2021

Contents

1 Problem Statement

This project aims to predict the whether a person applying for the credit card is expected to be a credit defaulter or not in the upcoming 12 months. The person is said to be a defaulter if they fail to pay the bills for 3 months consecutively.

1.1 Dataset

The dataset consists of 47 variables with the train dataset having an extra column containing the label for whether the customer is a defaulter or not. The variables include both categorical and continuous variables. There are around 83000 records in the complete training set.

2 Data Preprocessing

2.1 Identifying Categorical Data

Variable "mvar47" clearly seems to be a nominal categorical variable that takes the values 'C' or 'L' which could correspond to the two types of cards that are being provided by American Express (C for Charge Card and L for Lending Card).
Among the other 46 variables we used a cutoff of 100 unique values to determine categorical features. Variables `mvar16, mvar17, mvar18, mvar19, mvar20, mvar28, mvar34, mvar35, mvar36, mvar37, mvar38, mvar39, mvar43, mvar45, mvar46` are marked as nominal and ordinal categorical variables, and the variables except these are marked as numeric continuous variables.

2.2 Missing values

In the train and the test dataset, missing Values were marked as "missing" or "na". In order to handle these correctly and consistently, we first replaced all the "missing" and "na" with `numpy.nan`. Below plots represents the percentage of missing values for each variable in the train and test dataset:

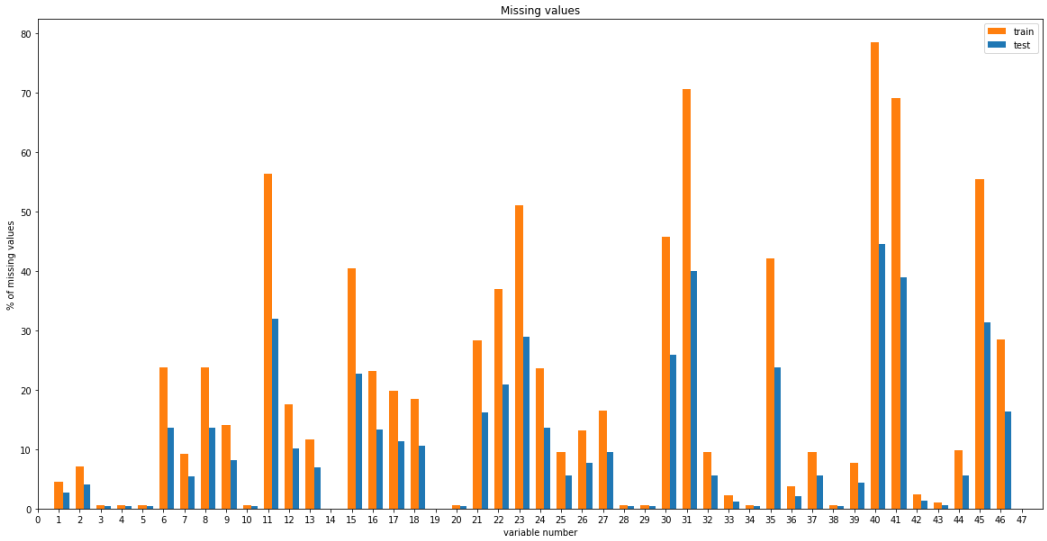

Figure 1: % of missing values in train and test data

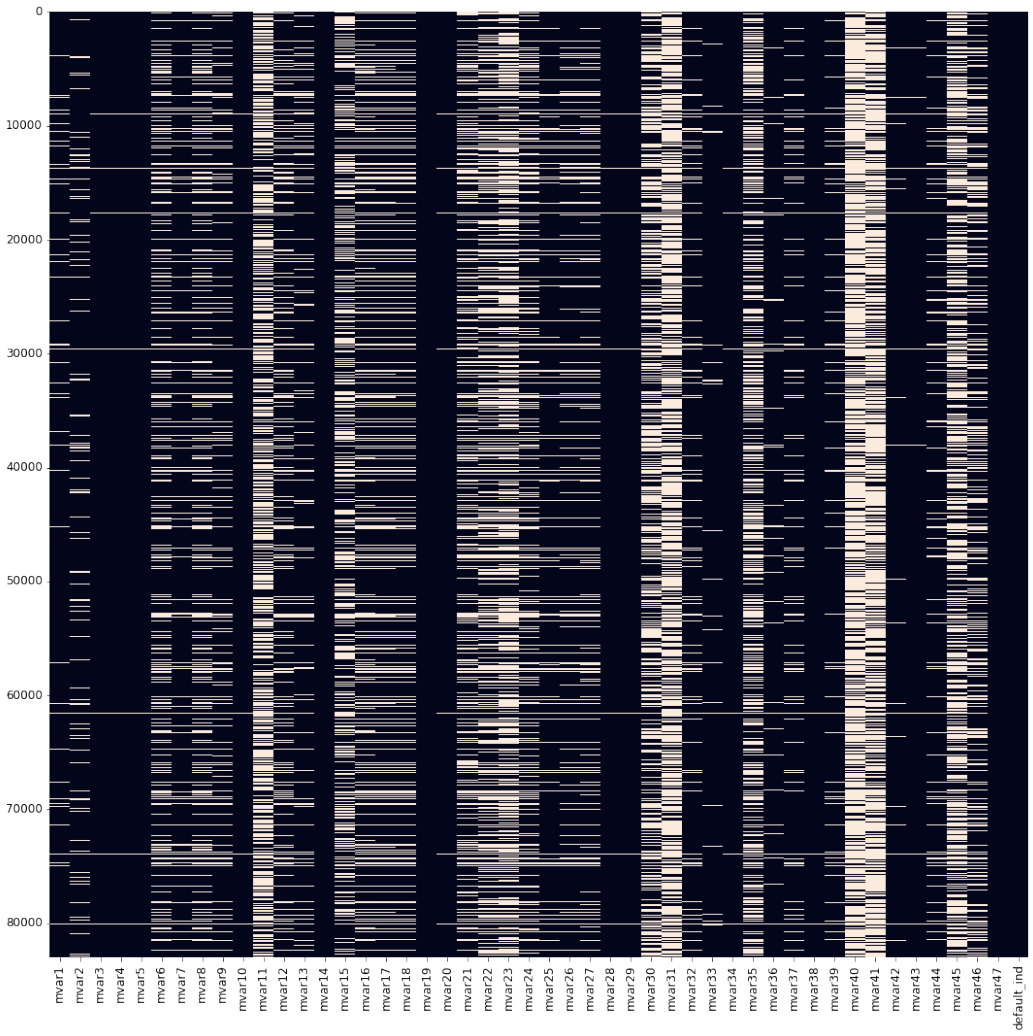

Figure 2: Heatmap showing the missing values for each feature, corresponding to each records. The white line corresponds to the missing data.

From both these plots, one can observe that variables like `mvar11, mvar15, mvar22, mvar23, mvar30, mvar31, mvar35, mvar40, mvar41, mvar45` have more than 35% of data missing. The variables that had majority missing were dropped from the columns list.
To fill in the NaN values in the dataset we tried different methods like mean fill, median fill, forward fill, backward fill, K-nearest neighbour imputation[1]. Mean imputer found to have greater f1 score when compared to other methods on a logistic regression model. Mean fill with the below mentioned transformations has better f1 scores and accuracies as compared to the kNN imputer with standard scaler. This can be seen from Table 2 and Table 3. Hence we move forward with a mean fill strategy.

2.3 Outliers

We looked at the boxplot for all the features and found that most of these features have outliers. To determine the outliers we used Inter-Quartile Range (IQR) proximity rule. The data points which fall below `Q1 - 1.5 IQR` (lower bound) or above `Q3 + 1.5 IQR` (upper bound) are deemed outliers.
Where Q1 and Q3 are the 25^{th} and 75^{th} percentile of the dataset respectively, and IQR represents the inter-quartile range and given by

$$IQR = Q3 - Q1$$

The box-plot for all the numeric features were constructed and it shows the outliers in each of them. The plot is shown in Figure 3 below.

From the figure it can be seen that there are some features like `mvar4, mvar5, mvar16, mvar17, mvar18, mvar34, mvar35, mvar39, mvar45, and mvar46`, where all the values marked as outliers. We will be skiping these variables when treating the outliers.
For the rest of the features we replaced the outliers which were lower than the lower bound by the 10th percentile value of the feature and, the outliers which were higher than the upper bound by the 90th percentile value of the feature.

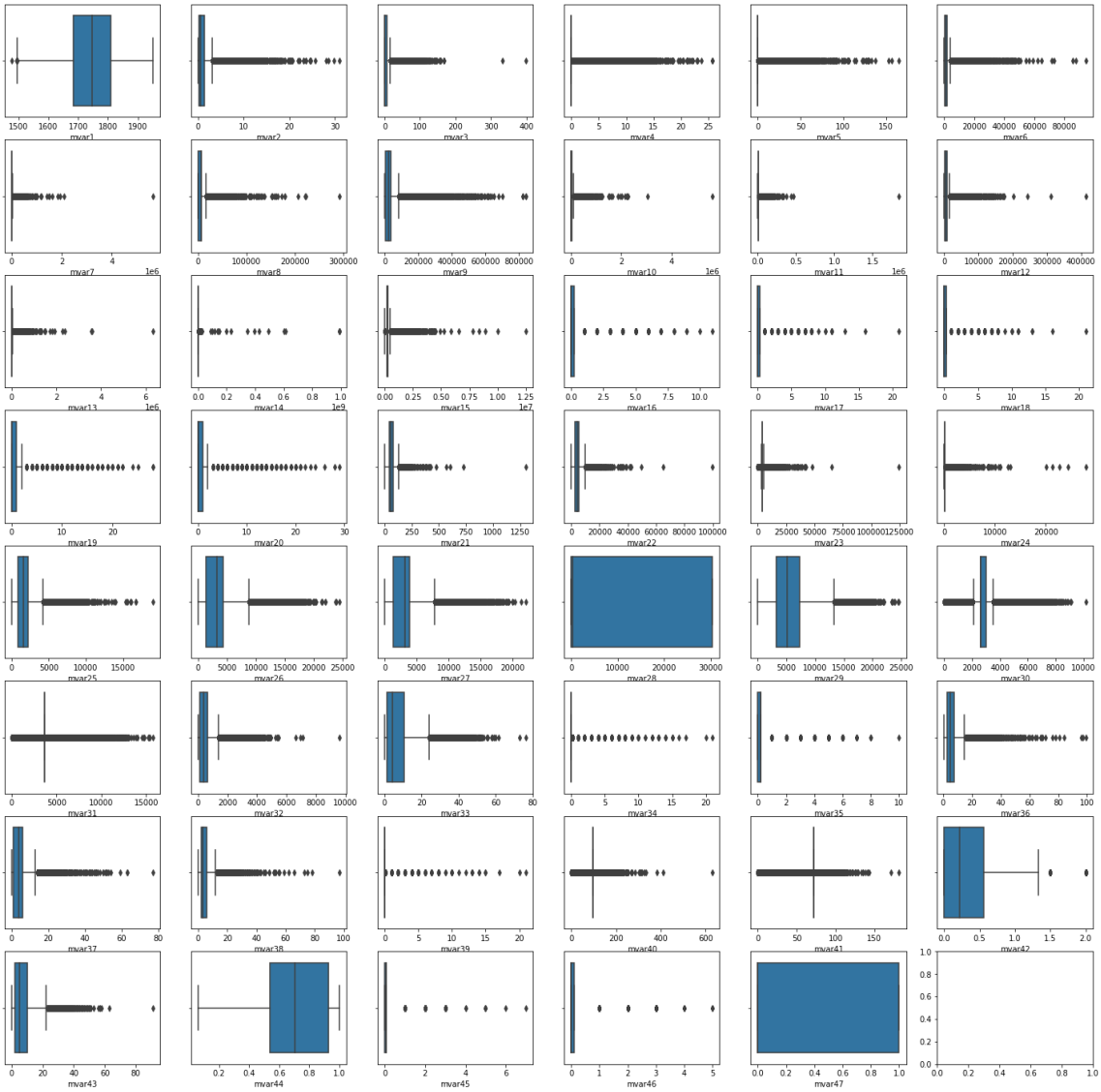

Figure 3: Figure showing the outliers in the train dataset. Each figure corresponds to a feature from `mvar1! through mvar46! in order`

2.4 Normalization and Transformation

Features like `mvar3, mvar4, mvar5, mvar16, mvar17, mvar18, mvar34, mvar35, mvar39, mvar40, mvar41, mvar42, mvar43, mvar45, mvar46` follow the power law distribution, i.e. a handful of the values have many points. We performed log tranformation on these features.

Whereas on features which had values spread uniformly over a range like `mvar1, mvar2, mvar6, mvar7, mvar8` etc. we performed min-max normalization.

3 Feature Selection

3.1 Correlation

In order to remove highly correlated features from the dataset, we calculated pair-wise correlation for numeric features. Plot below show a heatmap of the correlation matrix.

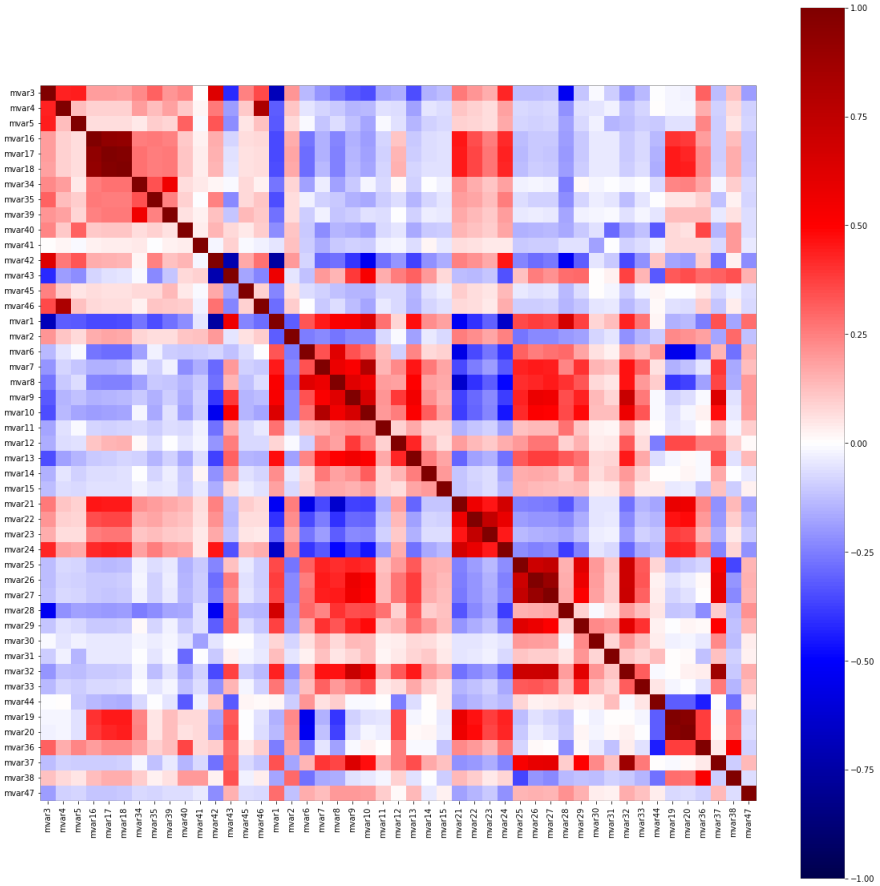

Figure 4: Figure showing the pair-wise correlation for train features

A cut-off of 0.7 was used to identify highly correlated features. The Pearson's correlation for the features having a higher than 0.7 correlation are tabulated below.

Feature1	Feature2	Pearson's correlation
mvar26	mvar25	0.701971
mvar32	mvar26	0.702131
mvar32	mvar27	0.707512
mvar9	mvar32	0.718141
mvar42	mvar43	0.723792
mvar25	mvar27	0.729958
mvar22	mvar23	0.730273
mvar42	mvar1	0.758282
mvar7	mvar10	0.797196
mvar46	mvar4	0.825589
mvar32	mvar37	0.878466
mvar27	mvar26	0.902904
mvar16	mvar18	0.919233
mvar17	mvar16	0.936170
mvar19	mvar20	0.968570
mvar18	mvar17	0.981352

Table 1: Table showing Highly Correlated features and their corresponding Pearson's Correslation

In order to deal with features that have high correlation, one of the two variables from the table above were removed.

4 Label Balancing

It was also seen that the dataset is highly imbalanced, with only 23855 positive (1) but 59145 negative (0) labels. So the dataset needs to be balanced for the model to perform well. There are numerous strategies in literature that are used to deal with unbalanced data. We use tried the following two.

4.1 SMOTE

We tried SMOTE (Synthetic Minority Over-Sampling Technique)[2] on the training data to balance the dataset by oversampling the minority class, in this case the positive class[3]. However, its performance was seen to be very suboptimal. It could be due to the over complication drawbacks as mentioned in this article [4]

4.2 Weighing Minority Class

Apart from Data generation we also tried weighting the label for the minority class classification. This can be done using `scale_pos_weight` parameter in XGBoost and LightGBM. The optimal value for `scale_pos_weight` was found to be 2.6.
This strategy performed better than smote and hence we proceed with this.

5 Model Selection

Once the data was pre-processed completely, different models were used on the training data and the validation scores were calculated. Different metrics were used to score each models, namely, the F1 Score, Accuracy Score, Precision and Recall. The following were the models that were evaluated -

- Logistic Regression
- Stochastic Gradient Descend Classifier
- Gaussian Naive Bayes Classifier
- k Nearest Neighbours Classifier
- Random Forest Classifier
- Extra Trees Classifier
- Gradient Boosting Classifier
- AdaBoost Classifier
- Extreme Gradient Boosting Classifier (XGBoost)
- LightGBMciteLGBM

The whole dataset was first split into training and validation data. Then the model was trained on the training data, with the scores calculated for both training and validation data. The scores for the validation data are tabulated below. The best 2 models were ensembled to make a soft Voting Classifier and it was also evaluated.

Models	F1 Score	Accuracy	Precision	Recall
LogisticRegression	49.1	76.54	65.07	39.42
SGDClassifier	49.35	76.44	64.43	39.99
GaussianNB	27.92	74.44	73.18	17.25
KNeighborsClassifier	45.07	72.24	52.15	39.69
RandomForestClassifier	47.33	76.75	67.62	36.41
ExtraTreesClassifier	49.24	76.84	66.37	39.14
GradientBoostingClassifier	51.59	77.25	66.25	42.25
AdaBoostClassifier	49.97	76.61	64.69	40.71
XGBClassifier	57.03	71.58	50.38	65.72
LGBMClassifier	58.58	71.25	49.94	70.84
VotingClassifier	58.2	71.88	50.74	68.22

Table 2: Table showing the Validation Data scores for various models preprocessed using mean imputer with Logarithmic and Min Max Scaler

Models	F1 Score	Accuracy	Precision	Recall
LogisticRegression	48.33	76.29	64.05	38.81
SGDClassifier	37.64	75.08	66.07	26.32
GaussianNB	41.99	75.14	63.02	31.49
KNeighborsClassifier	45.83	73.24	54.37	39.61
RandomForestClassifier	45.03	76.22	66.37	34.07
ExtraTreesClassifier	47.94	76.64	66.03	37.63
GradientBoostingClassifier	50.04	76.82	65.16	40.62
AdaBoostClassifier	48.95	76.24	63.43	39.85
XGBClassifier	57.05	71.88	50.63	65.35
LGBMClassifier	58.0	71.01	49.49	70.04
VotingClassifier	58.05	71.99	50.75	67.81

Table 3: Table showing the Validation Data scores for various models preprocessed using kNN imputer with Standard Scaler

As mentioned, the VotingClassifier was ensembled using the `XGBoostClassifer` and the `LGBMClassifier`

6 Hyperparameter Tuning

Once the model was decided, the tuning of its hyperparameters were done using optuna[5]. The objective to be maximized was chosen to be the mean score obtained by 3-fold cross-validation. The metric used to score was the Area under the curve of ROC.
For the Ensemble model, each of the classifier has its own set of hyperparameter tuning.

6.1 Tuning of XGBoost Classifier

XGBoost provides a lot of parameters that influences the model, however we tune only a sub-set of it that are relevant to the problem at hand. The parameters[6] that were tuned using optuna are given below.

`n_estimators`	The number of trees that the model uses
`max_depth`	The maximum depth of a tree
`reg_alpha`	The $L1$ regularization parameter, α
`reg_lambda`	The $L2$ regularization parameter, λ
`min_child_weight`	Minimum sum of hessian needed in a child
`gamma`	Minimum loss reduction to continue making partitions on leaf
`learning_rate`	Boosting learning rate
`colsample_bytree`	Subsample ratio for each tree

6.2 Tuning of LightGBM Classifier

Similar to the XGBoost Classifier, we tune a subset of the total available parameters for LightGBM. The parameters[7] that were chosen are provided below.

`n_estimators`	The number of Boosted trees in the model
`num_leaves`	The maximum number of leaves that each tree uses
`max_depth`	The maximum depth of a tree
`reg_alpha`	The $L1$ regularization parameter, α
`reg_lambda`	The $L2$ regularization parameter, λ
`min_child_samples`	Minimum number of samples needed in a child
`min_child_weight`	Minimum sum of hessian needed in a child
`learning_rate`	The Boosting learning rate

7 Final Submission

Once the hyperparameters were tuned, the model was then trained on the given Training Data after cleaning, and predictions were made on the cleaned Test Data.

References

[1] Jason Brownlee. knn imputation for missing values in machine learning. https://machinelearningmastery.com/knn-imputation-for-missing-values-in-machine-learning/, Aug 2020.

[2] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, Jun 2002.

[3] Guillaume Lemaître, Fernando Nogueira, and Christos K. Aridas. Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5, 2017.

[4] Smote — overcoming class imbalance problem using smote.

[5] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

[6] Xgboost python package. https://xgboost.readthedocs.io/en/stable/python/index.html.

[7] lightgbm.lightgbmclassifier. https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html.