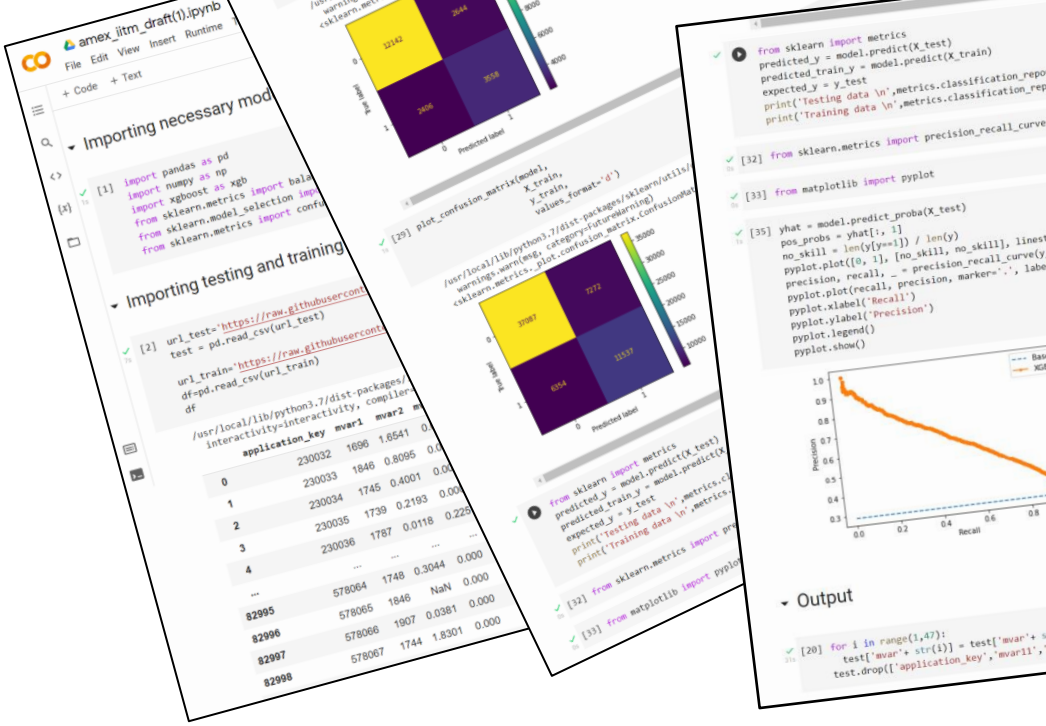


AMEX PROJECT REPORT

Customer Default Prediction

Group13	
Name	**Roll No.**
Akhil Krishna M M	CE19B036
Benny S L	ME19B087
Vishnu S Menon	CE19B103
Amritha P K	CE19B001
V Sai Dheeraj Chandra	CE19B025
T Pujitha	CE19B030

Contents

Introduction

In this report, we ‘Group13’, write about various approaches that we tried and one final model we obtained as a part of Amex Project on Customer Default Prediction. It was a fun journey of ups and downs in ratings, using different methods for analysing data, countering missing values, outliers and correlated variables, finally model fitting.

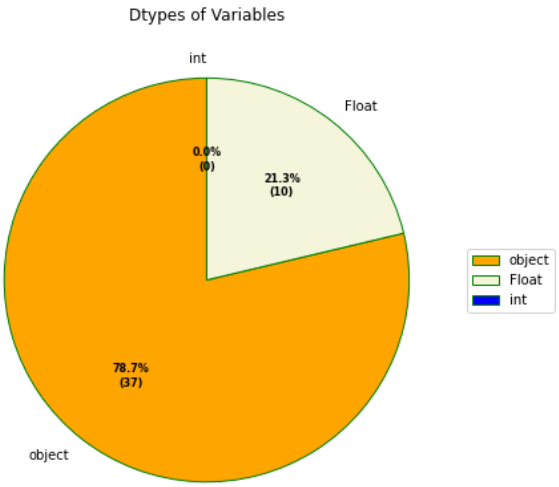
We express our sincere gratitude to Dr. Nandan Sudarsanam, Associate Professor, Department of Management Studies, who made us capable of doing this project. We sincerely thank Vamsi Praveen K, Nisarg Shah and Mohammed Sadiq for the guest lecture from AmEx. We are grateful to AmEx and IITM Doms for providing us with the opportunity to do this project. We also thank our teaching assistants Nishant, Pratyush Yadav and Anusha Kumar.

We were given a data base about customers with 47 variables with outcome that, whether they are default or not. There were no information about what each variable indicate. So feature selection became the most important part because removing one important variable will lead to loss of accuracy.

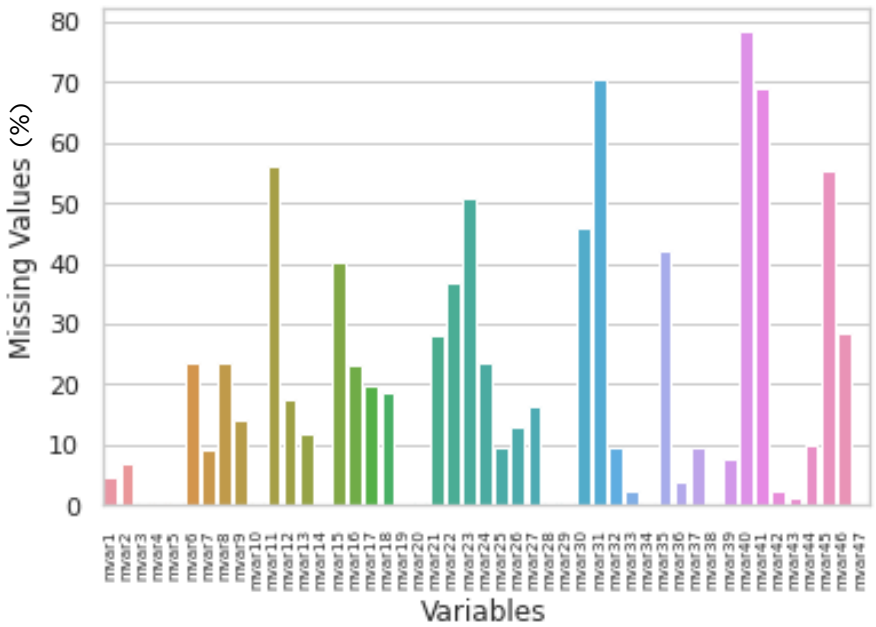
In the following pages, we walk you through different data pre-processing, analysis and fitting methods and one final model obtained mixing the right ingredients.

Data Analysis

The dataset contains 47 variables. Continuous variables: mvar1 to mvar46 and categorical variable: mvar47



Data type of variables were either ‘object’ or ‘Float’. Further analysis of data showed that data type was shown ‘object’ because of missing data in the form of ‘na’ and ‘missing’. Missing data in the form of ‘NaN’ has numerical type ‘Float’.



Data Analysis

The next analysis was done for outliers. This can be done only after imputing missing values. Removing outliers using IQR failed due to huge amount of outliers in the dataset. Therefore it is important to use a model that can handle outliers.

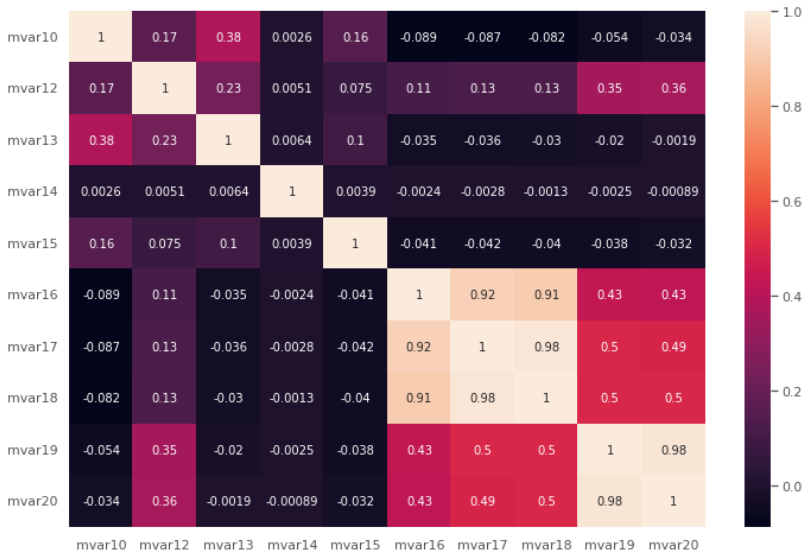
Rows were completely removed while attempting IQR method on outliers:

```
for i in variables:
  Q1 = np.percentile(df[i], 25, interpolation = 'midpoint')
  Q3 = np.percentile(df[i], 75, interpolation = 'midpoint')
  IQR = Q3 - Q1
  # Upper bound
  upper = np.where(df[i] >= (Q3+1.5*IQR))
  # Lower bound
  lower = np.where(df[i] <= (Q1-1.5*IQR))
  ''' Removing the Outliers '''
  df.drop(df[df[i] <= (Q1-1.5*IQR)].index, inplace = True)
  df.drop(df[df[i] >= (Q3+1.5*IQR)].index, inplace = True)
```

```
[81] print("New Shape: ", df.shape)

     New Shape:  (0, 43)
```

Correct method should be used for filling missing values, Otherwise we may end up deleting correlated variables that are not actually correlated or some important variables during correlation removing process.



Correlation heat map obtained for variables from 10 to 20 after doing simple linear imputation for missing values

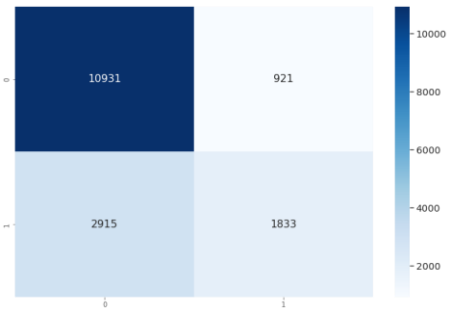
Missing Value Imputation

Methods for imputing missing values:

- Deterministic Regression imputation: Random values are predicted in a different column for missing values. Based on this random column, the missing values are predicted. The data provided contained lot of missing values which led to failure of this method - resulted in removing important features due to added correlation from this method.

- Linear imputation: In this method if there is a missing value between two known values a linear interpolation is done between the two values above and below it. This was a better method compared to deterministic regression.

	precision	recall	f1-score	support
0	0.92	0.79	0.85	13846
1	0.39	0.67	0.49	2754
accuracy			0.77	16600
macro avg	0.65	0.73	0.67	16600
weighted avg	0.83	0.77	0.79	16600



Model Summary on test data of Logistic regression with linear imputation on missing values

13	December 10, 2021	12:18 pm	5.59%
14	December 10, 2021	11:54 am	21.48%
15	December 09, 2021	04:47 pm	30.5%

Low rating for models were obtained while using deterministic regression imputation. At that point we realized that this method is adding correlations to variables. This resulted in removal of important variables.

Removing Correlated Variables

Correlated Variables with high vif or high correlation were to be removed. The approach used was to simply remove variables with vif > 2.5 or correlation > 40%.

- First data is checked for vif
- correlation data frame of a group of high 'vif' data were constructed
- correlation between them is checked
- Finally removing variables which have correlation between them keeping one variable.

Example of 'mvar17', 'mvar18', 'mvar19' and 'mvar20':

VIF data frame was created.
4 variables marked
got high vif values

	feature	VIF
0	const	1.659869
1	mvar1	5.112345
2	mvar2	1.165035
3	mvar3	2.838655
4	mvar4	2.838215
5	mvar5	1.773228
6	mvar6	1.529439
7	mvar7	5.177618
8	mvar8	2.220755
9	mvar9	3.208740
10	mvar10	6.734024
11	mvar12	1.536945
12	mvar13	1.269011
13	mvar14	1.000364
14	mvar15	1.057367
15	mvar16	7.205907
16	mvar17	37.141686
17	mvar18	31.598527
18	mvar19	31.944258
19	mvar20	32.391257
20	mvar21	2.213830
21	mvar22	1.580233
22	mvar24	1.048594
23	mvar25	3.164618

```
16   mvar17   37.141686
17   mvar18   31.598527
18   mvar19   31.944258
19   mvar20   32.391257
```

Correlation matrix between them was created. Found that 4 were Correlated between themselves.

	mvar17	mvar18	mvar19	mvar20
mvar17	1.000000	0.983247	0.500098	0.490790
mvar18	0.983247	1.000000	0.501699	0.499078
mvar19	0.500098	0.501699	1.000000	0.982952
mvar20	0.490790	0.499078	0.982952	1.000000

Removing 3 variables with high vif.

```
df.drop(['mvar17','mvar19','mvar20'], axis='colu
drps=drps+['mvar17','mvar19','mvar20']
```

Test Train Split

The train-test split is a technique for evaluating the performance of a machine learning algorithm. The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset.

- Train Dataset: Used to fit the machine learning model.
- Test Dataset: Used to evaluate the fit machine learning model

The objective is to estimate the performance of the machine learning model on new data that is not used to train the model.

```
(sum(y)/len(y))*100
```

```
28.740963855421686
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y)
```

```
(sum(y_train)/len(y_train))*100
```

```
28.74056224899598
```

```
(sum(y_test)/len(y_test))*100
```

```
28.742168674698792
```

Like in our problem, many classification problems do not have a balanced number of examples for each class label. As such, it is desirable to split the dataset into train and test sets in a way that preserves the same proportions of examples in each class as observed in the original dataset.

About XGBoost

XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. The default base learners of XGBoost are tree ensembles.

The tree ensemble model is a set of classification and regression trees (CART). Trees are grown one after another and attempts to reduce the misclassification rate are made in subsequent iterations.

XGBoost improves upon the base GBM framework through systems optimization and algorithmic enhancements:

System Optimization**:**

1. Parallelization: process of sequential tree building using parallelized implementation.
2. Tree Pruning**:** XGBoost uses 'max_depth' parameter, and starts pruning trees backward.
3. Hardware Optimization

Algorithmic Enhancements:

1. Regularization: uses both LASSO & Ridge regularization
2. Sparsity Awareness: learns the sparsity pattern in the data and visits only the default direction in each node
3. Weighted Quantile Sketch**:** employs distributed weighted quantile sketch algorithm to find the optimal split points
4. Cross-validation: built-in cross-validation no need to specify exact number of boosting iterations required in a single run.

XGBoost in this project

We have already seen the data analysis of the dataset provided. And we faced lot of trouble dealing with missing values and correlated features. XGBoost will deal with every mentioned problem single handedly!

- Missing values: XGBoost supports missing values by default. In tree algorithms, branch directions for missing values are learned during training. During the training time XGB decides whether the missing values should fall into the right node or left node. This decision is taken to minimize the loss. If there are no missing values during the training time, the tree made a default decision to send any new missing value to the right node.

- Correlated Features: xgboost removes the extra column before calculating the model, so the importance is not affected. However, when you add a column that is partially correlated to another, thus with a lower coefficient, the importance of the original variable x is lowered.

December 19, 2021	10:34 pm	53.14%
December 18, 2021	07:44 pm	59.76%

Significant decrease in amex rating occurred when we removed highly correlated variables: mvar17,mvar19 and mvar20 from xgboost model (removal of features having significant important on model building)

XGBoost Parameters

- booster
  - Data type : string, Default : ‘gbtree’
  - Specify which booster to use: gbtree, gblinear or dart.

- n_jobs
  - Data type : int, Default : 1
  - Number of parallel threads used to run xgboost
  - To run on all cores, value should not be entered and algorithm will detect automatically

- random_state
  - Data type : float, Default : 0
  - Random number seed
  - used for generating reproducible results and also for parameter tuning

- eval_metric
  - Data type : string
  - metric to be used for validation data
  - Typical values are: rmse (root mean square error), mae (mean absolute error), logloss (negative log-likelihood), error (Binary classification error rate) (0.5 threshold), merror (Multiclass classification error rate), mlogloss (Multiclass logloss), auc (area under the curve)

XGBoost Parameters

- objective
  - Data type : string, Default: 'binary:logistic‘
  - This defines the loss function to be minimized

- learning_rate
  - Data type : float, Default : 0.1
  - Boosting learning rate
  - Makes the model more robust by shrinking the weights on each step

- max_depth
  - Data type : int, Default : 0.3
  - Maximum tree depth for base learner
  - Used to control over-fitting as higher depth will allow model to learn relations very specific to a particular sample

- n_estimators
  - Data type : int, Default : 100
  - Number of trees to fit

XGBoost Parameters

- verbosity
  - Data type : int, Default : 1
  - The degree of verbosity. Valid values are 0 (silent) - 3 (debug)

- gamma
  - Data type : float, Default : 0
  - Minimum loss reduction required to make a further partition on a leaf node of the tree.
  - Gamma specifies the minimum loss reduction required to make a split
  - A node is split only when the resulting split gives a positive reduction in the loss function

- min_child_weight
  - Data type : int, Default : 1
  - Minimum sum of instance weight(hessian) needed in a child
  - Used to control over-fitting. Higher values prevent a model from learning relations which might be highly specific to the particular sample selected for a tree

XGBoost Parameters

- max_delta_step
  - Data type : int, Default :0
  - Maximum delta step we allow each tree's weight estimation to be.
  - If the value is set to 0, it means there is no constraint. If it is set to a positive value, it can help making the update step more conservative

- subsample
  - Data type : float, Default : 1
  - Subsample ratio of the training instance
  - Lower values make the algorithm more conservative and prevents overfitting but too small values might lead to under-fitting.

- colsample_bytree
  - Data type : float, Default : 1
  - Subsample ratio of columns when constructing each tree

- colsample_bylevel
  - Data type : float, Default : 1
  - Subsample ratio of columns for each level
  - Would be wise to keep it as default if we use both subsample and colsample_bytree

XGBoost Parameters

- colsample_bynode
  - Data type : float, Default : 1
  - Subsample ratio of columns for each split.
- reg_alpha
  - Data type : float, Default : 1
  - L1 regularization term on weights
  - used in case of very high dimensionality – make algorithm faster
- reg_lambda
  - Data type : float, Default : 1
  - L2 regularization term on weights
  - Used to handle the regularization part of XGBoost
- scale_pos_weight
  - Data type : float, Default : 1
  - Balancing of positive and negative weights
  - value greater than 0 should be used in case of high class imbalance as it helps in faster convergence
- base_score
  - Data type : float, Default : 0.5
  - The initial prediction score of all instances, global bias

Parameter Optimization

We have to deal with such big number of parameters in a xgboost model. Optimization methods play a key role in finding ideal parameters for our model.

Optimization Algorithms :

- Grid Search: exhaustive search over every combination of specified parameter values. If we specify 2 possible values for max_depth and 3 for n_estimators, Grid Search will iterate over 6 possible combinations.
- Bayesian Optimization: builds a model for the optimization function and explores the parameter space systematically, which is a smart and much faster way to find the parameters.

```
xgb_bo = BayesianOptimization(bo_tune_xgb, {'max_depth':(5,8),'subsample':(0.7,1),'colsample_bytree':(0.7,1),'colsample_bylevel':(0.6,0.9),'
xgb_bo.maximize(n_iter=5, init_points=8, acq='ei')
```

iter	target	colsam...	colsam...	learni...	max_depth	min_ch...	n_esti...	reg_alpha	reg_la...	subsample
1	-0.4014	0.7818	0.9	0.1651	7.662	2.034	130.2	0.6629	0.1474	0.77
2	-0.399	0.6993	0.9018	0.1835	5.415	2.647	135.2	0.6691	0.3275	0.873
3	-0.4017	0.7249	0.7585	0.2828	5.958	4.574	117.7	0.5093	0.1921	0.7101
4	-0.3984	0.7197	0.7469	0.08622	7.236	1.567	103.3	0.5161	0.3729	0.7429
5	-0.3996	0.8672	0.7678	0.1743	6.767	2.065	112.0	0.6114	0.3874	0.9133
6	-0.4059	0.6927	0.9729	0.2375	7.69	5.283	135.6	0.5052	0.1438	0.8481
7	-0.4038	0.695	0.8331	0.2686	6.214	5.503	144.9	0.7496	0.2186	0.8457
8	-0.4048	0.639	0.9963	0.2395	7.895	5.139	101.0	0.6073	0.2687	0.8839
9	-0.3985	0.8089	0.9567	0.05815	5.324	4.57	100.2	0.7577	0.3061	0.9301
10	-0.398	0.7484	0.8893	0.1059	6.107	2.748	143.6	0.7828	0.2039	0.7503
11	-0.4009	0.761	0.7126	0.2555	5.789	5.32	108.4	0.6186	0.2654	0.7525
12	-0.3985	0.7805	0.8551	0.06049	5.301	4.465	100.2	0.7967	0.1258	0.9996
13	-0.3979	0.7233	0.931	0.07203	6.13	2.12	143.1	0.7447	0.2609	0.7259

Bayesian optimization with 13 iterations tried in one of our models

Although, it was giving parameters, model failed to give better results in the rating. This was also one of the most time consuming model we used.

Parameter Optimization

Optimization Algorithms :

- Randomized Search: uses a large (possibly infinite) range of hyper parameter values, and randomly iterates a specified number of times over combinations of those values. We can specify the number of iterations.

Random checking using reasonable values :

Lot of resources are available suggesting reasonable range for each of the parameters. One way to do parameter optimization is to change values of paramaters and do several tests on the split dataset : test and train. We can then go via accuracy, F-scores, confusion matrix find the best set of parameters.

Reasonable range of some parameters:
max_depth: 3-10, learning_rate: 0.01-0.3, n_estimators: 100-1000, subsample: 0.1-1 etc.

XGBoost Model Fitting

XGBClassifier & XGBRegressor:

- XGBClassifier is used for classification problems with objective '*binary:logistic' for* binary classification and *'multi:softprob' for* multi-class classification.
- XGBRegressor is used while predicting numerical value outputs. '*reg:squarederror'* is the default loss function.

Early stopping rounds:

An approach to train complex machine learning models to avoid overfitting. It works by monitoring the performance of the model that is being trained on a separate test dataset and stopping the training procedure once the performance on the test dataset has not improved after a fixed number of training iterations.

Evaluation Metrics and Evaluation Set:

The XGBoost model can evaluate and report on the performance on a test set for the the model during training.
It supports this capability by specifying both an test dataset and an evaluation metric on the call to model.fit() when training the model and specifying verbose output.

XGBoost Model Fitting

Evaluation Metric

AUCPR:

The area under the precision-recall curve (AUCPR) is a single number summary of the information in the precision-recall (PR) curve. Similar to the receiver operating characteristic curve, the PR curve has its own unique properties that make estimating its enclosed area challenging.

Precision-recall curves (PR curves) are recommended for highly skewed domains where ROC curves may provide an excessively optimistic view of the performance.

```
[153]   validation_0-aucpr:0.622509
[154]   validation_0-aucpr:0.622575
[155]   validation_0-aucpr:0.622654
[156]   validation_0-aucpr:0.622727
[157]   validation_0-aucpr:0.622789
[158]   validation_0-aucpr:0.622834
[159]   validation_0-aucpr:0.622838
[160]   validation_0-aucpr:0.622903
[161]   validation_0-aucpr:0.622877
[162]   validation_0-aucpr:0.622858
[163]   validation_0-aucpr:0.622861
[164]   validation_0-aucpr:0.622804
[165]   validation_0-aucpr:0.62283
[166]   validation_0-aucpr:0.622818
[167]   validation_0-aucpr:0.622779
[168]   validation_0-aucpr:0.622718
[169]   validation_0-aucpr:0.622669
[170]   validation_0-aucpr:0.622731
Stopping. Best iteration:
[160]   validation_0-aucpr:0.622903

XGBClassifier(colsample_bylevel=0.5, learning_rate=0.04, max_depth=6,
              min_child_weight=1.5, n_estimators=300, random_state=2,
              scale_pos_weight=1.8)
```

we used aucpr metric in the final xgboost model

Final Model

We have tried mainly 2 models: Logistic Regression and XGBoost. And as anyone would expect XGBoost came out on top. Here are the steps we used in the final model.

i. Data preprocessing:

The first step is changing every type of missing data in continuous variables into unified numeric 'NaN' format. Categorical Variable 'mvar47' is then changed into numeric; mapping 'L' to '1' and 'C' to '0'. Finally every variable is in numeric format (Float and int) ready for a xgboost model.

Even though xgboost has good mechanisms to deal with missing data, we find it reasonable to remove variables with missing data (> 50%).
5 continuous variables are dropped.

```
mvar11 -> 56%
mvar23 -> 51%
mvar31 -> 71%
mvar40 -> 78%
mvar41 -> 69%
mvar45 -> 55%
```

Data is ready for boosting Algorithms.

Final Model

ii. Train-Test-Split:

Training dataset is split into test and train data frames with 0.75 test-train split. Stratifying using 'default_ind' make sure that train and test data frames contain similar percent of default customers. This kind of splitting gives an insight on the ability of model to predict.

iii. Parameter Tuning:

Belter results were obtained on random checking using reasonable values compared to bayesian optimization and hyperopt.

iv. Model Fitting:

Early stopping method is used while fitting the model. This will prevent overfitting.
Evaluation metric used: aucpr (Area Under the Precision-Recall Curve)
eval_set: x_test, y_test
Training data is then fitted using these parameters to build the model.

This model was finally used to predict the data.

Model Summary

Variables Used :

```
df.iloc[:,1:42].columns
```

```
Index(['mvar1', 'mvar2', 'mvar3', 'mvar4', 'mvar5', 'mvar6', 'mvar7', 'mvar8',
       'mvar9', 'mvar10', 'mvar12', 'mvar13', 'mvar14', 'mvar15', 'mvar16',
       'mvar17', 'mvar18', 'mvar19', 'mvar20', 'mvar21', 'mvar22', 'mvar24',
       'mvar25', 'mvar26', 'mvar27', 'mvar28', 'mvar29', 'mvar30', 'mvar32',
       'mvar33', 'mvar34', 'mvar35', 'mvar36', 'mvar37', 'mvar38', 'mvar39',
       'mvar42', 'mvar43', 'mvar44', 'mvar46', 'mvar47'],
      dtype='object')
```

Parameters Used for XGBoost :

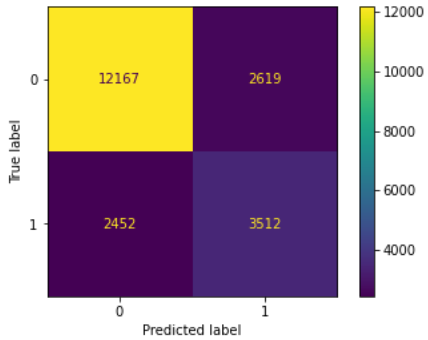
```
model.get_params()
```

```
{'base_score': 0.5,
 'booster': 'gbtree',
 'colsample_bylevel': 0.4,
 'colsample_bynode': 1,
 'colsample_bytree': 0.6,
 'gamma': 0,
 'learning_rate': 0.05,
 'max_delta_step': 0,
 'max_depth': 6,
 'min_child_weight': 1.5,
 'missing': None,
 'n_estimators': 200,
 'n_jobs': 1,
 'nthread': None,
 'objective': 'binary:logistic',
 'random_state': 6,
 'reg_alpha': 0,
 'reg_lambda': 1,
 'scale_pos_weight': 1.78,
 'seed': None,
 'silent': None,
 'subsample': 1,
 'verbosity': 1}
```

Model Summary

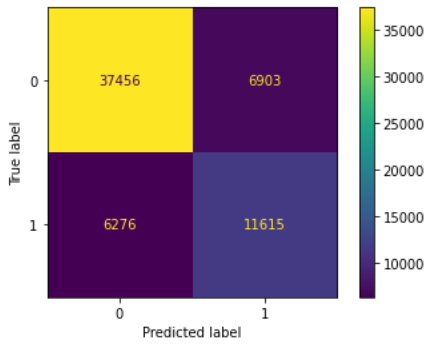
Testing data

	precision	recall	f1-score	support
0	0.84	0.82	0.83	14786
1	0.58	0.60	0.59	5964
accuracy			0.76	20750
macro avg	0.71	0.71	0.71	20750
weighted avg	0.76	0.76	0.76	20750

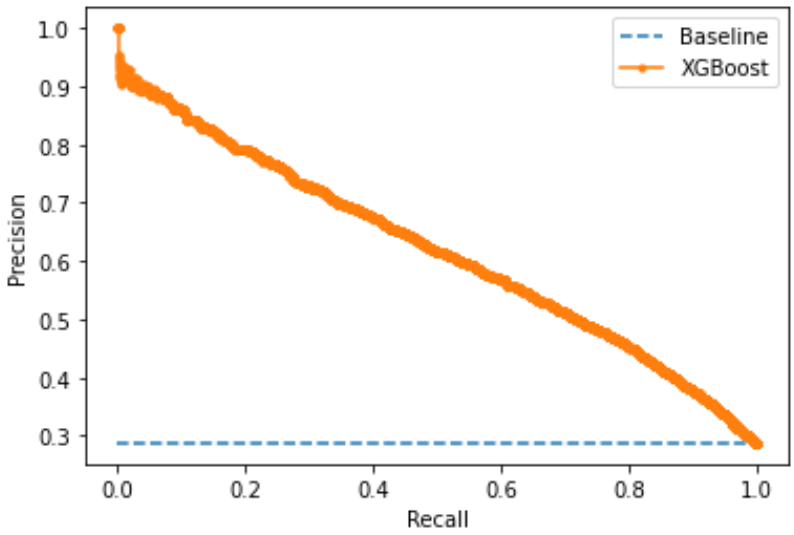


Training data

	precision	recall	f1-score	support
0	0.85	0.85	0.85	44359
1	0.63	0.64	0.64	17891
accuracy			0.79	62250
macro avg	0.74	0.74	0.74	62250
weighted avg	0.79	0.79	0.79	62250



Precision Recall Curve:



aucpr : 0.618847

Conclusion

On working for this project, we tried Logistic Regression as well as XGBoost model. XGBoost came out on top as expected.

Model	Minimum Amex Score	Maximum Amex Score
Logistic Regression	5.59%	51.96%
XGBoost	52.02%	59.86%

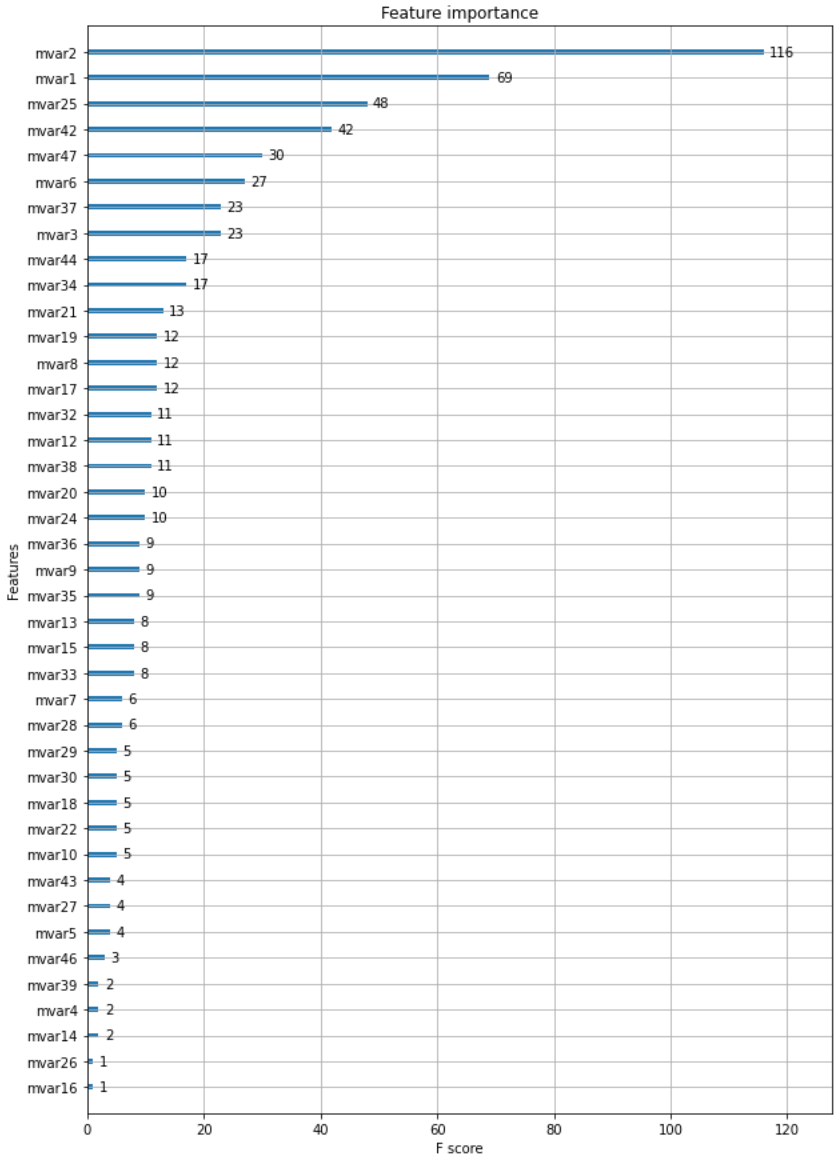
During Logistic Regression phase, main issue faced was with missing values due to improper missing value imputation methods. Very poor amex score was obtained while using deterministic regression for filling missing value followed by removal of correlated variables. Important features were removed resulting in a poor model.

XGBoost was rather smooth. We had a good time working with parameter optimization algorithms in XGBoost.

XGBoost model results:

December 20, 2021	05:53 pm	59.86%
December 20, 2021	02:54 pm	59.81%
December 19, 2021	11:05 pm	59.57%
December 19, 2021	10:34 pm	53.14%
December 18, 2021	07:44 pm	59.76%
December 18, 2021	10:03 am	56.63%
December 17, 2021	09:28 pm	53.18%
December 17, 2021	02:01 pm	52.02%
December 16, 2021	11:26 pm	53.27%
December 16, 2021	06:48 pm	52.73%

Conclusion



The feature importance bar plot was obtained from XGBoost model. This indicated that ‘mvar1’ was one of the important feature which often got removed along with correlated variables in logistic regression models!

References Used

- https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d
- https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/
- https://machinelearningmastery.com/avoid-overfitting-by-early-stopping-with-xgboost-in-python/
- https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/
- https://medium.com/hypatai/how-xgboost-handles-sparsities-arising-from-of-missing-data-with-an-example-90ce8e4ba9ca
- https://www.geeksforgeeks.org/python-programming-language/