26. Predicting the capability of each applicant in repaying a loan

t.sureshbabu@gmail.com

SURESH BABU THANICKACHALAM

Table of Contents

Abstract

This project aims to use machine learning methodologies on historical loan application data to predict whether an applicant will be able to repay a loan or default. Home Credit Company has provided various datasets and merged all the information at the application level. Impute with appropriate data, and remove outliers using statistical analysis and Exploratory Analysis. The dataset contains categorical and continuous features only. As part of Future Engineering, both domain-based and statistical-based approaches are implemented. Future Transformation is done using a Standard scaler on continuous features and response encoding done on categorical features.

As an extension to the Visual EDA has driven feature sampling and baseline model development, the focus for this phase included data modeling to combine all datasets, feature engineering considering categorical and numerical features, and implementing experiments with various Models like Logistic Regression, Random Forest, XGBoost, LightGBM, Neural Networks, and TabNet.

The best performing Model is LightGBM with AUC 79% and Logistic Regression 77% followed by TabNet and Neural Networks of AUC of 76%.

1. Literature survey & Data Acquisition

1. Problem definition

- ✓ Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders.
- ✓ Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data including telco and transactional information to predict their clients' repayment abilities.
- ✓ Using Home Credit data, one needs to predict whether a given application will be able to repay the loan or not. Hence, the right applicant is not rejected.

2. Need for data science

- ✓ Decision of whether a person can repay a loan or not can be decided scientifically.
- ✓ The statistical and Machine Learning results would enable the deciding authority to decide the decision quickly and accurately.
- ✓ The lending company losses can be reduced by rejecting the right applications and revenue will be increased by sanctioning the right applications.
- ✓ This approach is more data-driven and avoids manual errors.
- ✓ Data-driven approach enables to fill the missing information with mean/median details.
- ✓ It helps to observe the common trends like whether the applicant can repay the loan or default.

3. Business Constraints

- ✓ No strict latency constraints - Given the application, the system can take a few seconds to a max of a minute to predict borrower capabilities.
- ✓ Interpretability is important – The system should be explainable “why” the applicant can repay or default. To do this, Feature Importance is very important and will help the decision-maker give meaningful reason to their stakeholders.
- ✓ Cost of Miss-Classification is high –The system should be more focused on reducing the misclassification, especially on False Negative.

2. Dataset: Description & Source

Home Credit has provided the dataset for this problem approximately 2.68GB in size. The dataset contains various files as follows,

```
In [1]: import os
        #files list
        print(os.listdir("./data/"))

['application_test.csv', 'application_train.csv', 'bureau.csv', 'bureau_balance.csv', 'credit_card_balance.csv', 'HomeCredit_columns_description.csv', 'installments_payments.csv', 'POS_CASH_balance.csv', 'previous_application.csv', 'sample_submission.csv']
```

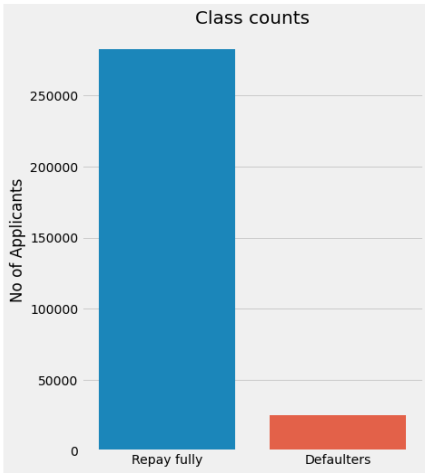
Sl.No	File Name	Description
1.	HomeCredit_columns_description.csv	This file contains file names, column names, details about the column
2.	application_train.csv	This file contains application details. Home Credit has segregated data Train and Test.
3.	application_test.csv	Test application details. However, this dataset can't be used for supervised Machine Learning Models as class labels are not given
4.	bureau.csv	Contains about applicant previous credits provided by other financial institutions that were reported to the Credit Bureau.
5.	bureau_balance.csv	Contains applicant's Monthly balances of previous credits in the Credit Bureau. It is a month-wise record. -1 denotes a fresh record.
6.	POS_CASH_balance.csv	Contains applicant Monthly balance snapshots of previous POS (point of sales) and cash loans with Home Credit.
7.	credit_card_balance.csv	Contains applicant's previous monthly credit card balance with Home credit.
8.	previous_application.csv	Contains applicant previous application details like whether the loan is approved or rejected and approved loan details etc.

9.	installments_payments.csv	Contains applicant's previous loan repayment history details.
10.	Sample_submission.csv	Contains rejected applications. [0.5 means positive, defaulters]

Application train dataset is main and other datasets like bureau and bureau_balance, etc. are one-to-many relationships. Hence, the child datasets should be grouped and merged to the main Train dataset.

3. Metric to Use to Validate Model

```
Defaulters examples = 24825
Repay fully examples = 282686
Proportion of Defaulters to Repay examples = 8.78%
```



It is clear looking at the graph that the data set is highly imbalanced because there is a high number of applicants who repay the loan compared to applicants who default.

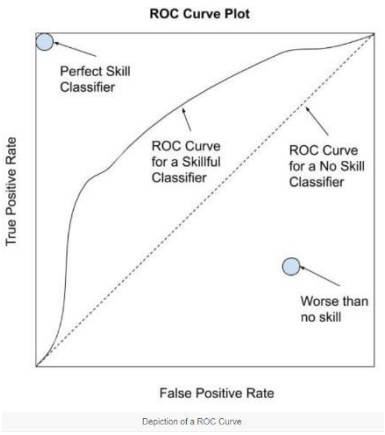
- ✓ **Scenario - 1**: False Negative (FN): The system detects that the applicant "repays" and hence, the lender approved the loan. In reality, the applicant "defaulted" would be a big loss to the lender.

- ✓ **Scenario - 2:** False Positive (FP): The system detects that the applicant would "default" and hence the loan rejected. In reality, the applicant is capable of "repaying" would be revenue loss to the lender.

- ✓ **Scenario 1 is riskier than scenario 2. Hence, the FN case is to be minimized. In order to minimise FN, metric recall/F1 Scores is more important than precision.**

Due to Imbalanced Data **Accuracy** is not a helpful metric.

- **ROC-AUC Curve**



ROC-AUC metric is TPR vs FPR. TPR covers **recall.** Hence it is a useful metric for imbalanced data. ROC-AUC should be greater than 0.5 means the model is sensible. Also, this metric Train vs Test curve would explain whether the Model is overfitted or not.

- ✓ F1-Score = (2 * Precision * Recall) / (Precision + Recall), focus on precision and recall. Hence the Model that generates a good F1-Score would be a better one.

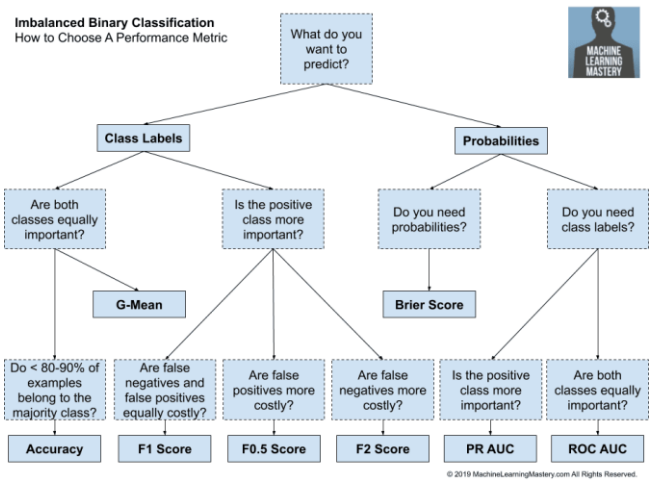
- ✓ Log loss = -((1 – y) * log(1 – yhat) + y * log(yhat))

  y = true value and yhat = predicted value.

- ✓ Is Probabilistic Metrics for Imbalanced Classification? A perfect classifier has a log loss of 0.0 with worse values being positive up to infinity.

- ✓ K-fold cross-validation is the gold standard for evaluating the performance of a machine-learning algorithm on unseen data with k set to 3, 5, or 10.

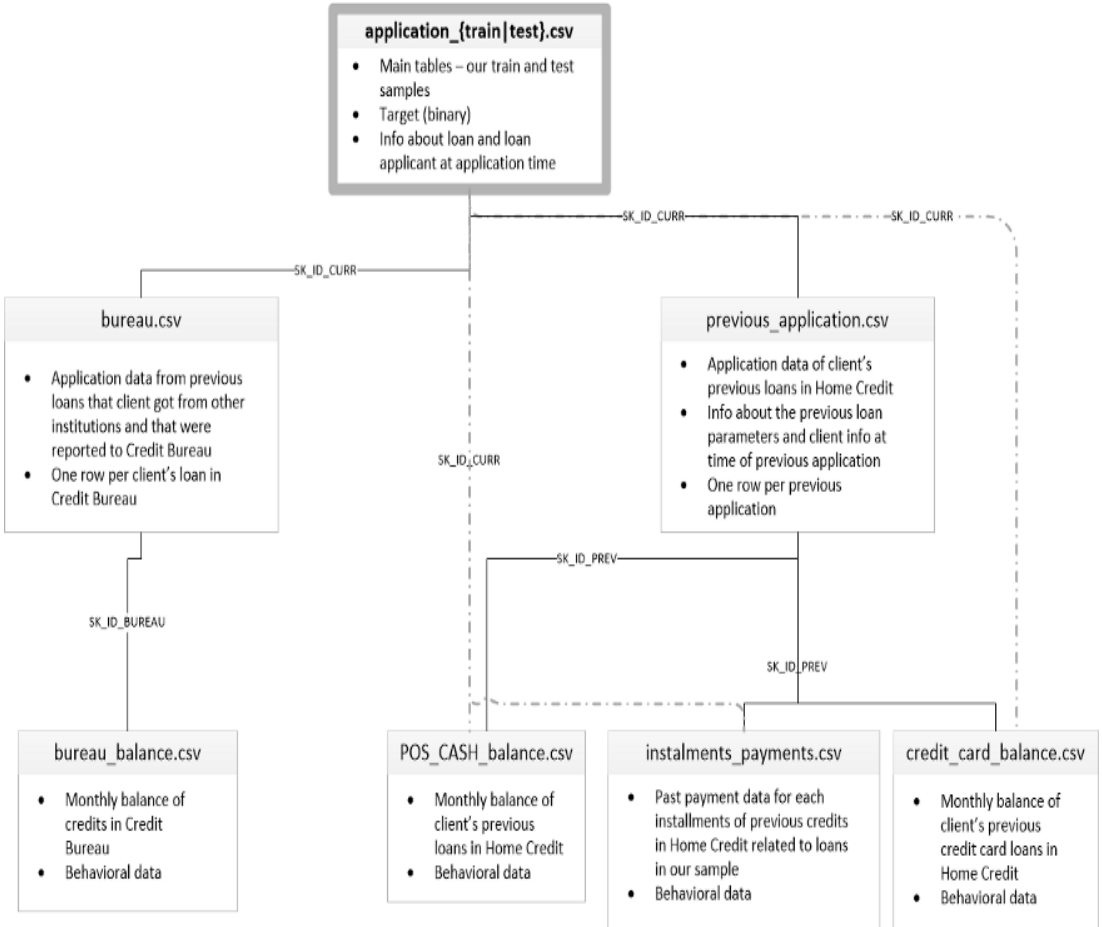
- ✓ Confusion Matrix: The confusion matrix is a handy presentation of the accuracy of a model with two or more classes.



credit:https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/

- ✓ When the model is trained use class weights to balance the prediction.

4. Entity Relationship between data



Credit: https://medium.com/analytics-vidhya/home-credit-loan-default-risk-7d660ce22942

5. Reduce Memory Usage

- ✓ Since the dataset size is very big, running the Model for the evaluation would be a big challenge in a moderate laptop configuration. Hence, remove unnecessary spaces and shrink the dataset based on true data. The following code helps to shrink the dataset significantly.

```
#Credit :- https://www.kaggle.com/rinnqd/reduce-memory-usage
def reduce_memory_usage(df):
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)

    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

    return df
```

```
df_train = reduce_memory_usage( pd.read_csv('./data/application_train.csv', encoding= 'unicode_escape'))
print('Number of data points : ', df_train.shape[0])
print('Number of features : ', df_train.shape[1])
df_train.head()
```

```
Memory usage of dataframe is 286.23 MB
Memory usage after optimization is: 92.38 MB
Decreased by 67.7%
Number of data points :  307511
Number of features :  122
```

6. Exploratory Data Analysis (EDA)

1. Find the Missing values

```
Sum of null values in each feature:
-----------------------------------
AMT_ANNUITY                    12
AMT_GOODS_PRICE               278
NAME_TYPE_SUITE              1292
OWN_CAR_AGE                202929
OCCUPATION_TYPE             96391
                              ...
AMT_REQ_CREDIT_BUREAU_DAY    41519
AMT_REQ_CREDIT_BUREAU_WEEK   41519
AMT_REQ_CREDIT_BUREAU_MON    41519
AMT_REQ_CREDIT_BUREAU_QRT    41519
AMT_REQ_CREDIT_BUREAU_YEAR   41519
Length: 67, dtype: int64
```

- ✓ It is clearly shown that a lot of features have missing values. Categorical features are imputed with the most frequent values and numerical features are imputed with median strategy. The median strategy would be less impact in the case of outliers. If more than 80% of data, even median would be impacted. In such a scenario, the feature should be removed.

```
# of columns before removing in train ~> 122
There are 0 columns with greater than 80% missing values.
# of columns after removing in train ~> 122
```

- ✓ All features, have at least 20% data. Hence, none of the columns were removed due to missing values.

2. Impute appropriate values for missing

```
#fix imputation
def fill_mostfrequent_value(df):
    cat_cols=df.select_dtypes(include=object).columns #categorical values
    for col in cat_cols:
        df[col].fillna(df[col].value_counts(dropna=True).index[0], inplace=True)
    return pd.DataFrame(df)
df_train=fill_mostfrequent_value(df_train)
```

```
def fill_median_value(df):
    num_cols=df.select_dtypes(exclude=object).columns
    for col in num_cols:
        df[col].fillna(df[col].median(axis=0,skipna = True),inplace=True)
    return pd.DataFrame(df)
#fill median value for Numerical columns
df_train = fill_median_value(df_train)
```

- ✓ Categorical features are imputed with most frequent values
- ✓ Numerical features are imputed with median values.

3. Feature-wise Analysis

- ✓ Since more columns are present in the dataset, randomly chosen a few columns to understand the feature from a statistical point of view.
  Ex: “DAYS_EMPLOYED”

```
df_train['DAYS_EMPLOYED'].describe()
```

```
count    307511.000000
mean      63815.045904
std      141275.766519
min      -17912.000000
25%       -2760.000000
50%       -1213.000000
75%        -289.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

- ✓ This numerical column has negative values. Days employed can’t be negative. Hence, find all negative numerical columns and if appropriate convert them into positive values.

- ✓ Columns DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAYS_ID_PUBLISH, and DAYS_LAST_PHONE_CHANGE have keyed in the system with negative values. Hence converted to positive values.
- ✓ Column DAYS_EMPLOYED having a max value of 365243 which is Error and hence fixed it as 0.
- ✓ Also DAYs columns are converted to Year by dividing it by 365.

4. Interquartile Range to Detect Outliers in Data

- ✓ The outliers may suggest experimental errors, variability in a measurement, or an anomaly.
- ✓ Outliers badly affect the mean and standard deviation of the dataset. These may statistically give erroneous results.
- ✓ Most machine learning algorithms do not work well in the presence of outliers. So it is desirable to detect and remove outliers.
- ✓ IQR is used to measure variability by dividing a data set into quartiles. The data is sorted in ascending order and split into 4 equal parts. Q1, Q2, Q3 called first, second and third quartiles are the values that separate the 4 equal parts.
- ✓ IQR is the range between the first and the third quartiles namely Q1 and Q3: IQR = Q3 – Q1. The data points which fall below Q1 – 1.5 IQR or above Q3 + 1.5 IQR are outliers.

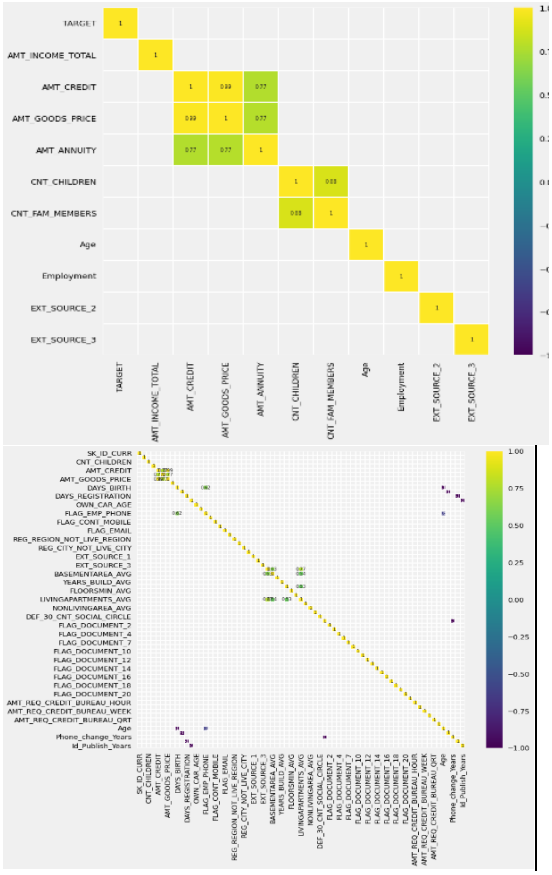
```
x = PrettyTable()
x.field_names = ["Sl.No","column", "IQR value","Q1-1.5IQR","Q3+1.5IQR"]

# IQR
for idx,nc in enumerate(numerical_columns):
    col=df_train[nc].values
    col = col[~np.isnan(col)]
    Q1 = np.percentile(col, 25,interpolation = 'midpoint')
    Q3 = np.percentile(col, 75,interpolation = 'midpoint')
    Q1_15IQR = Q1-np.round((Q3-Q1),2)
    Q3_15IQR = Q3+np.round((Q3-Q1),2)
    x.add_row([idx+1,nc,np.round((Q3-Q1),2),Q1_15IQR,Q3_15IQR])

x.sortby = "Q1-1.5IQR"
x.reversesort = True
x.sortby = "Q3+1.5IQR"
x.reversesort = True
x.align["Q1_15IQR"] = "r"
x.align["Q3_15IQR"] = "r"
print(x)
```

```
+-------+-----------------------------+-----------+----------------------+----------------------+
| Sl.No |            column           | IQR value |       Q1-1.5IQR      |       Q3+1.5IQR      |
+-------+-----------------------------+-----------+----------------------+----------------------+
|   4   |          AMT_CREDIT         |  538650.0 |       -268650.0      |       1347300.0      |
|   6   |       AMT_GOODS_PRICE       |  441000.0 |       -202500.0      |       1120500.0      |
|   1   |          SK_ID_CURR         |  177997.0 |        11148.5       |        545139.5      |
|   3   |       AMT_INCOME_TOTAL      |  90000.0  |        22500.0       |        292500.0      |
|   5   |         AMT_ANNUITY         |  18072.0  |        -1548.0       |        52668.0       |
|   10  |      DAYS_REGISTRATION      |   5470.0  |        -12950.0      |         3460.0       |
|   9   |        DAYS_EMPLOYED        |   2471.0  |        -5231.0       |         2182.0       |
|   79  |   DAYS_LAST_PHONE_CHANGE    |   1296.0  |        -2866.0       |         1022.0       |
|   11  |      DAYS_ID_PUBLISH        |   2579.0  |        -6878.0       |         859.0        |
|  106  |             Age             |   19.91   |         14.09        |        73.81625      |
|  109  |      Registration_Years     |    15.0   |         -9.5         |         35.5         |
|  110  |      Id_Publish_Years       |    7.1    |  -2.4007812499999996 |        18.896875     |
|   22  |   HOUR_APPR_PROCESS_START   |    4.0    |          6.0         |         18.0         |
|  107  |         Employment          |    6.8    |     -6.0001953125    |        14.4015625    |
|   12  |         OWN_CAR_AGE         |    0.0    |          9.0         |         9.0          |
|  108  |      Phone_change_Years     |    3.5    |     -2.7001953125    |        7.80078125    |
|  105  | AMT_REQ_CREDIT_BUREAU_YEAR  |    2.0    |         -1.0         |         5.0          |
|   77  |  OBS_60_CNT_SOCIAL_CIRCLE   |    2.0    |         -2.0         |         4.0          |
|   75  |  OBS_30_CNT_SOCIAL_CIRCLE   |    2.0    |         -2.0         |         4.0          |
|   19  |       CNT_FAM_MEMBERS       |    1.0    |          1.0         |         4.0          |
```

5. Find Correlation between Columns

- ✓ Model performance is dependent on features. The features should be independent and correlated with class labels. Highly correlated features should be removed.
- ✓ In order to identify the correlation between features, heatmap with dataset correlation technique was used.
- ✓ Since the dataset has more columns, multiple steps are followed with a small number of features and then consolidated in the last.
- ✓ If Col1 and highly correlated with Col2, one of the columns is removed.

```
def plot_heatmap_for_corr(df):
    corr=df.corr()
    plt.figure(figsize=(12, 10))

    sns.heatmap(corr[(corr >= 0.5) | (corr <= -0.4)],
            cmap='viridis', vmax=1.0, vmin=-1.0, linewidths=0.1,
            annot=True, annot_kws={"size": 8}, square=True);
```



- ✓ REGION_RATING_CLIENT, REGION_RATING_CLIENT_W_CITY are more than 50% correlated with REGION_POPULATION_RELATIVE. Hence, REGION_RATING_CLIENT, REGION_RATING_CLIENT_W_CITY fields are removed.
- ✓ These steps are repeated and removed multiple columns

- ✓ Using the correlation technique, around 40 features are reduced. That is from 122 to 84.

6. Check data duplication

- ✓ Data duplication also affects the performance of the model. Hence finding duplication is important.

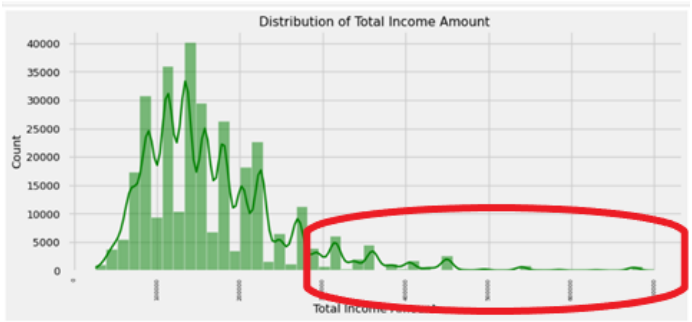
```
columns_without_id = [col for col in df_train.columns if col!='SK_ID_CURR']
#print any duplicate entries in train
print('Number of duplicates in the train data:',df_train[df_train.duplicated(subset = columns_without_id, keep=False)]
      .shape[0])
```

Number of duplicates in the train data: 0

No duplicated records were found in the training dataset.

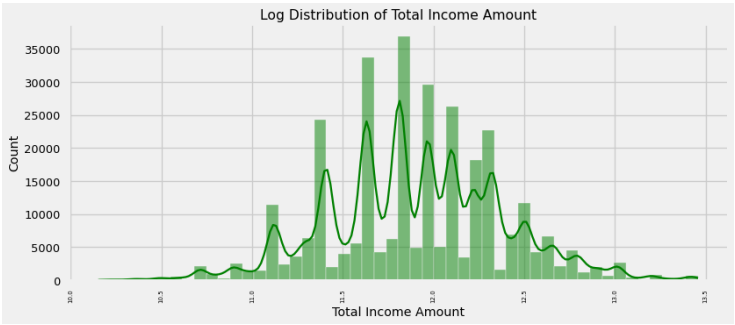
7. Univariate Analysis

- ✓ Univariate analysis is done using the histogram on certain columns and mostly numerical Ex: AMT_INCOME_TOTAL



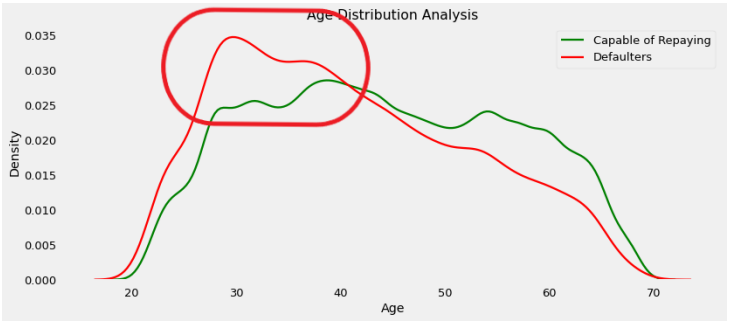
- ✓ Application Income total of less than 7million was plotted. Above 7million there are about 738 applications and the ratio of that application is less than 0.25%. Hence above 7m records are ignored for analysis.

- ✓ If we noticed it is highly skewed data. Even after dropping applications over 7 million, we see data is right-skewed. We can use log transformation to reduce the skewness



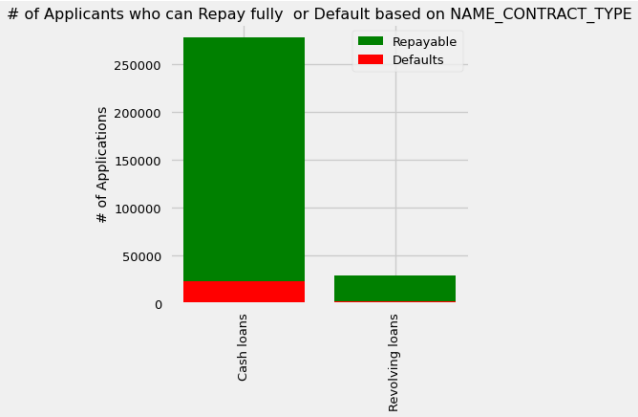
- ✓ After log transformation, the data was somewhat bell-curved.

- ✓ Let's try another feature Age distribution analysis based on classes.



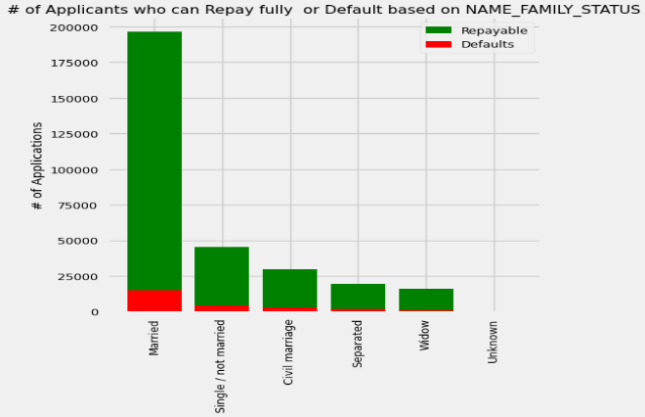
- ✓ Age between 25 and 35, the defaulters are high compared to the Age above 50.

✓ Name Contract Type is a categorical feature and has “Cash loans” and “Revolving Loans”.



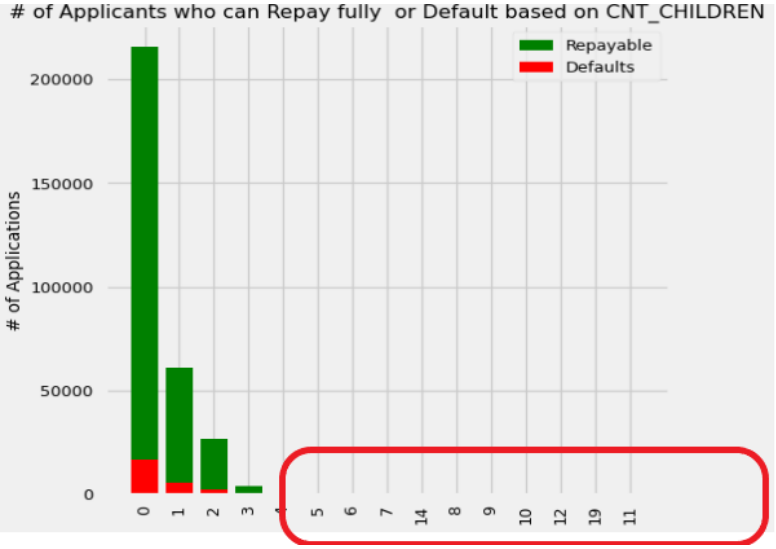
✓ From the bar chart, customers prefer Cash loans to Revolving Loans.

✓ NAME_FAMILY_STATUS has 5 categorical values.



✓ From the graph, Married people have taken loans more than other categories.
✓ Widow is very less and a couple of applications have an unknown category but very negligible.

✓ CNT_CHILDREN has multiple categorical values from 0 to 14

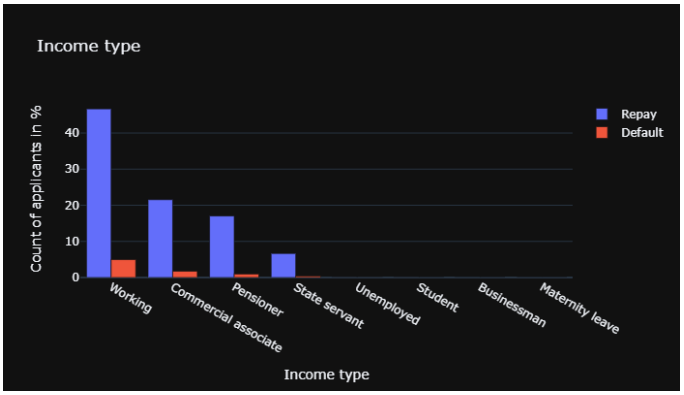


✓ From the graph, at the time loan was applied, customers have no children or less than 4.
✓ A High number of applications shows 0 children.
✓ More children could be an Error. [14, 12 and 11].
✓ Let us try to do percentile-based to understand the outliers.

```
99.0 percentile value is 3
99.1 percentile value is 3
99.2 percentile value is 3
99.3 percentile value is 3
99.4 percentile value is 3
99.5 percentile value is 3
99.6 percentile value is 3
99.7 percentile value is 3
99.8 percentile value is 3
99.9 percentile value is 4
100 percentile value is  19
```

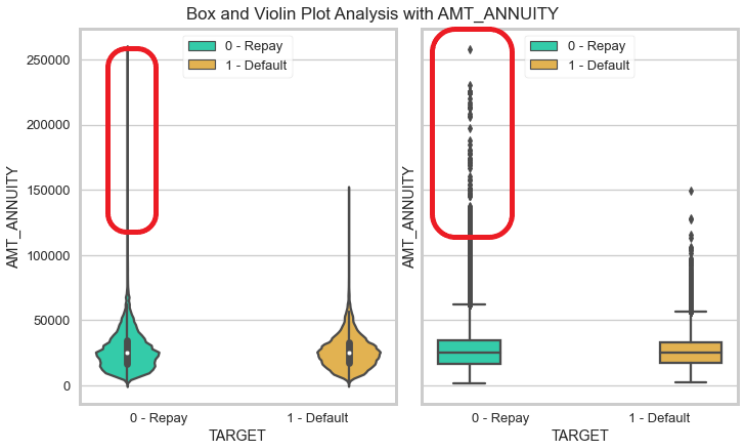
- ✓ We can safely remove or set to 6 if the number of children is more than 6.

- ✓ Let's try to understand the Income type with different bar charts.

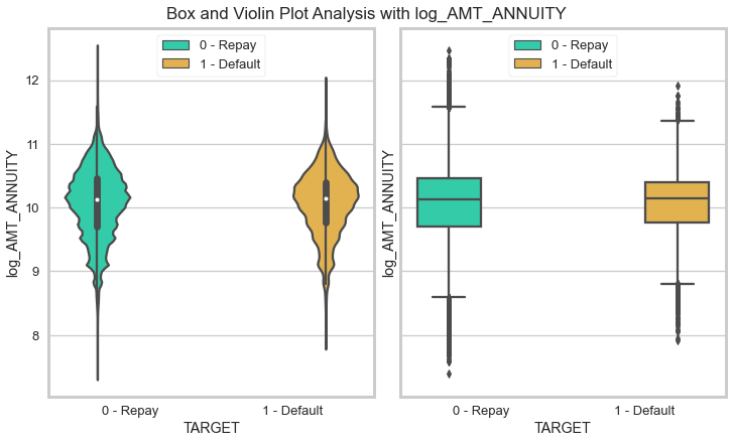


- ✓ Working people have taken more loans than businessmen or other types.
- ✓ Maternity Leave can't be an Income type. We can remove and impute with Unknown Income type.

- ✓ AMT_ANNUITY feature is a numerical one and will try to understand with Boxplot and violin plot. Violin plot combines density plot and IQR.



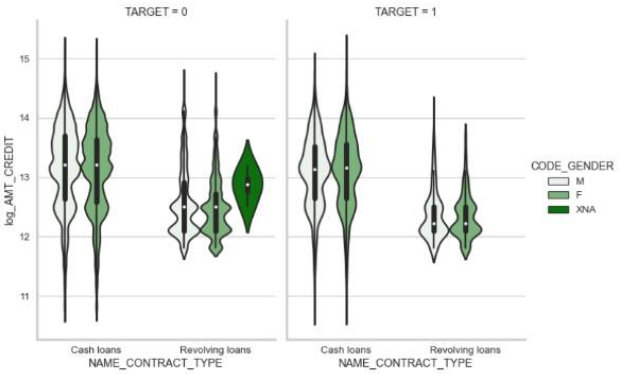
- ✓ From the plot, it is clearly shown that the data is right-skewed.
- ✓ Let's try to do log transformation and try it



- ✓ After log transformation, the distribution is better.

8. Bivariate Analysis:

- ✓ Let's try to understand two features at a time. Example:



- ✓ AMT_Credit is converted to a logarithmic value.
- ✓ XNA could be an error or doesn't want to disclose it.
- ✓ Based on their Credit Amount, Men & Women with Cash Loans have higher chances of repaying the loans.

7. Feature Engineering

- ✓ From the features given, most correlated and more missing data features are removed. We can add additional features through polynomial-based and domain-based features.

```
#Try to add some more features domain based
# Credit :- https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction#Feature-Engineering
epsilon=0.001 # to avoid div/0 error
def Feature_Engineering(df):
    df1 = pd.DataFrame()
    df1['SK_ID_CURR']=df['SK_ID_CURR']
    df1['pc_Credit_Income'] = np.round((df['AMT_CREDIT'] / (df['AMT_INCOME_TOTAL']+epsilon)),4)
    df1['pc_Annuity_Income'] = np.round((df['AMT_ANNUITY'] / (df['AMT_INCOME_TOTAL']+epsilon)),4)
    df1['pc_Credit_Annuity'] = np.round((df['AMT_CREDIT'] / (df['AMT_ANNUITY']+epsilon)),4)
    df1['Credit_Goods_Diff'] = df['AMT_CREDIT'] - df['AMT_GOODS_PRICE']
    df1['pc_Loan_Value']=np.round((df['AMT_CREDIT'] / (df['AMT_GOODS_PRICE']+epsilon)),4)
    df1['CREDIT_TERM'] = np.round((df['AMT_ANNUITY'] / (df['AMT_CREDIT']+epsilon)),4)
    df1['pc_Employment_Age'] = np.round((df['Employment'] / (df['Age']+epsilon)),4)
    return df1

df2=Feature_Engineering(df_train)
df_train = df_train.merge(df2, on = 'SK_ID_CURR', how = 'left')
print('Train data after Feature Engineering ~>{}'.format(df_train.shape))
```

- ✓ Domain-based features are created by dividing two features or the difference of two features.
- ✓ Ex: Percentage loan value is AMT_Credit / AMT_GOODS_PRICE
- ✓ Credit_Goods_Diff = AMT_CREDIT- AMT_GOODS_PRICE

```
# Try to add some polinomial features

# Make a new dataframe for polynomial features
#credit: https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction#Feature-Engineering
#https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

# Generate a new feature matrix consisting of all polynomial combinations of the features with degree less than
# or equal to the specified degree. For example, if an input sample is two dimensional and of the form [a, b],
# the degree-2 polynomial features are [1, a, b, a^2, ab, b^2].

# imputer for handling missing values
from sklearn.preprocessing import PolynomialFeatures
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='median')

poly_features = df_train[['SK_ID_CURR','EXT_SOURCE_1','EXT_SOURCE_2', 'EXT_SOURCE_3', 'Age', 'Employment','TARGET']]

poly_target = poly_features['TARGET']
poly_features = poly_features.drop(columns = ['TARGET'])

# Need to impute missing values
poly_features = imputer.fit_transform(poly_features[['EXT_SOURCE_1','EXT_SOURCE_2', 'EXT_SOURCE_3', 'Age','Employment']])

# Create the polynomial object with specified degree
poly_transformer = PolynomialFeatures(degree = 2)

# Train the polynomial features
poly_features= poly_transformer.fit_transform(poly_features)

print('Polynomial Features shape: ', poly_features.shape)
```

- ✓ Polynomial Features are extracted on a few columns like External sources, Age, Employment with the degree of 3.
- ✓ This will add additional columns.

- ✓ Train dataset cleaned as much as possible. Let's try to merge additional datasets are given.
- ✓ Since the relationship between the main dataset and other datasets are one-to-many relationships, let's try to group them at the application level and merge them into the main dataset.
- ✓ Feature engineering on additional datasets is incorporated based on statistical information such as count, mean, sum, min, and max.
- ✓ Categorical features are extracted at the application level and merged.

1. Bureau dataset

- ✓ This dataset contains 1.7million records and 17 features.
- ✓ Among these 'AMT_ANNUITY' feature name repeated. Since it is at the bureau level, didn't drop it but renamed it.

```
df_bureau.rename(columns={"AMT_ANNUITY": "BUREAU_AMT_ANNUITY"}, inplace = True)
```

- ✓ Sanity check is done on numerical columns and checked min and max values. Found few column values are in negative values. Hence, converted into a positive value.

```
for col in df_days_cols:
    print('column {} min value {} and max valaue {}'.format(col, df_days_cols[col].min(),df_days_cols[col].max()))
```

```
column DAYS_CREDIT min value -2922 and max valaue 0
column CREDIT_DAY_OVERDUE min value 0 and max valaue 2792
column DAYS_CREDIT_ENDDATE min value -42048.0 and max valaue 31200.0
column DAYS_ENDDATE_FACT min value -42016.0 and max valaue 0.0
column DAYS_CREDIT_UPDATE min value -41947 and max valaue 372
```

```
df_bureau['DAYS_CREDIT']=-df_bureau['DAYS_CREDIT']
df_bureau['DAYS_ENDDATE_FACT']=-df_bureau['DAYS_ENDDATE_FACT']
```

- ✓ Most frequent and median imputations are applied on categorical and numerical values respectively. After that, feature engineering is done on this dataset separately on categorical values and numerical values.

```
df_bureau_categorical = feature_engineering_categorical_variables(df=df_bureau, group_var = 'SK_ID_CURR', df_name = 'bureau')
df_bureau_categorical.head(1)
```

```
# Group by the applicant id, calculate aggregation statistics
df_bureau_agg = numeric_aggregate(df_bureau.drop(columns = ['SK_ID_BUREAU']),group_var = 'SK_ID_CURR', df_name = 'bureau')
df_bureau_agg.head(1)
```

- ✓ Then grouped the data, based on SK_ID_CURR column and merged to the main dataset.

```
#merge df_bureau categorical columns with train since it is multiple take the 1 record per I
df_bureau_categorical_group =df_bureau[cat_columns].groupby('SK_ID_CURR').head(1).reset_index(drop=True)
df_train=df_train.merge(df_bureau_categorical_group,on='SK_ID_CURR', how='left' )
```

```
# Merge bureau numerical aggregation with the training data
df_train = df_train.merge(df_bureau_agg, on = 'SK_ID_CURR', how = 'left')
# Merge bureau categorical numerical data with the training data
df_train = df_train.merge(df_bureau_categorical, on = 'SK_ID_CURR', how = 'left')
#df_train = reduce_memory_usage(df_train)
print('Number of data points : ', df_train.shape[0])
print('Number of features : ', df_train.shape[1])
```

```
Number of data points :  307511
Number of features :  199
```

2. Bureau balance dataset

- ✓ This is dataset is at the bureau level. Hence, this dataset features to be merged at the Bureau level and then at the application level. However, categorical columns are grouped at the application level and picked up the 1 column.
- ✓ This dataset contains 27.3 million records.
- ✓ Feature Month Balance can't be negative values. Hence converted into positive values

```
#since overdue doesn't have negative values, convert them into positive
df_bureau_balance['MONTHS_BALANCE'] = -df_bureau_balance['MONTHS_BALANCE']
```

- ✓ Standard imputation is done on all missing values.
- ✓ Feature Engineering is done these features and merged into the main dataset.

```
df_bureau_balance_agg = numeric_aggregate(df=df_bureau_balance,group_var = 'SK_ID_BUREAU', df_name = 'bureau_bal')
#Add Categorical field numerical counts
df_bureau_balance_categorical = feature_engineering_categorical_variables(df=df_bureau_balance,
                                                                           group_var = 'SK_ID_BUREAU', df_name = 'bureau_bal')
# Dataframe grouped by the loan
bureau_by_loan = df_bureau_balance_agg.merge(df_bureau_balance_categorical, right_index = True,
                                             left_on = 'SK_ID_BUREAU', how = 'left')
# Merge to include the SK_ID_CURR
bureau_by_loan = df_bureau[['SK_ID_BUREAU', 'SK_ID_CURR']]\
                          .merge(bureau_by_loan, on = 'SK_ID_BUREAU', how = 'left').reset_index(drop=True)
#drop SK_ID_BUREAU column and aggreate at client level
bureau_by_loan.drop(columns = ['SK_ID_BUREAU'],axis=1,inplace=True)
# Aggregate the stats for each client
bureau_balance_by_client = numeric_aggregate(df=bureau_by_loan,group_var = 'SK_ID_CURR', df_name = 'client')

# Merge with the training data
df_train = df_train.merge(bureau_balance_by_client, on = 'SK_ID_CURR', how = 'left').reset_index(drop=True)
```

3. Previous Loan Count

- ✓ Previous loan count is extracted from the bureau dataset and merged into the main dataset. This is domain-based feature extraction.

```
prev_loan_cnt = df_bureau.groupby('SK_ID_CURR', as_index=False)['SK_ID_BUREAU'].count().\
                                                rename(columns = {'SK_ID_BUREAU': 'prev_loan_cnt'})
#merge previous loan counts to train dataset
df_train = df_train.merge(prev_loan_cnt, on = 'SK_ID_CURR', how = 'left')
```

4. Previous Application Dataset:

- ✓ This dataset contains all previous loan details of the customer. This dataset contains 37 features and the many columns are similar to other datasets. Hence converted to unique names.

```
df_previous_application =reduce_memory_usage(pd.read_csv('./data/previous_application.csv', encoding= 'unicode_escape'))
print('Number of data points : ', df_previous_application.shape[0])
print('Number of features : ', df_previous_application.shape[1])
df_previous_application.head(1)
```

```
Memory usage of dataframe is 471.48 MB
Memory usage after optimization is: 309.01 MB
Decreased by 34.5%
Number of data points :  1670214
Number of features :  37
```

```
#Check previous application and current train dataset have similar columns. if so, rename pre_app columns, except ID columns
print('Similar columns~~>{}'.format(df_previous_application.columns.intersection(df_train.columns).tolist()))
```

```
Similar columns~~>['SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY', 'AMT_CREDIT', 'AMT_GOODS_PRICE', 'WEEKDAY_APPR_PROCESS_ST
ART', 'HOUR_APPR_PROCESS_START', 'NAME_TYPE_SUITE']
```

- ✓ Did some basic checking on these columns and found AMT_DOWN_PAYMENT has negative values. This could be a typo and hence, negative values reset to zero.

```
df_days_cols = df_previous_application.loc[:, df_previous_application.columns.str.contains('AMT')]
df_days_cols.columns
for col in df_days_cols:
    print('column {} min value {} and max valaue {}'.format(col, df_days_cols[col].min(),df_days_cols[col].max()))
```

```
column AMT_DOWN_PAYMENT min value -0.8999999761581421 and max valaue 3060045.0
```

```
#convert negative values to zero as there is not concept of negative down payment
df_previous_application.loc[df_previous_application['AMT_DOWN_PAYMENT'] <0,'AMT_DOWN_PAYMENT']=0
```

```
df_days_cols = df_previous_application.loc[:, df_previous_application.columns.str.contains('DAY')]
for col in df_days_cols:
    print('column {} min value {} and max valaue {}'.format(col, df_days_cols[col].min(),df_days_cols[col].max()))
```

```
column pre_app_WEEKDAY_APPR_PROCESS_START min value FRIDAY and max valaue WEDNESDAY
column NFLAG_LAST_APPL_IN_DAY min value 0 and max valaue 1
column DAYS_DECISION min value -2922 and max valaue -1
column DAYS_FIRST_DRAWING min value -2922.0 and max valaue 365243.0
column DAYS_FIRST_DUE min value -2892.0 and max valaue 365243.0
column DAYS_LAST_DUE_1ST_VERSION min value -2801.0 and max valaue 365243.0
column DAYS_LAST_DUE min value -2889.0 and max valaue 365243.0
column DAYS_TERMINATION min value -2874.0 and max valaue 365243.0
```

- ✓ The other columns have both negative and positive values. Didn't make any changes to that column as it requires domain expertise.

```
#DAYS DECISION will be +ve days
df_previous_application['DAYS_DECISION']=-df_previous_application['DAYS_DECISION']
```

- ✓ Standard imputation is done on all numerical and categorical columns.
- ✓ Feature Engineering is done all numerical and categorical columns and grouped at the application level.
- ✓ Categorical columns are extracted and merged with Train dataset.

```
#merge previous application categorical columns with train since it is multiple take the 1 record per ID
df_Pre_App_cat_group =df_previous_application[cat_columns].groupby('SK_ID_CURR').head(1).reset_index(drop=True)
df_train=df_train.merge(df_Pre_App_cat_group,on='SK_ID_CURR', how='left' )
```

```
# Group by the applicant id, calculate aggregation statistics
df_prev_app_numerical_agg = numeric_aggregate(df_previous_application.drop(columns = ['SK_ID_PREV']),
                                              group_var = 'SK_ID_CURR', df_name = 'pre_app').reset_index(drop=True)
#merge with df_train
df_train=df_train.merge(df_prev_app_numerical_agg,on='SK_ID_CURR', how='left')
```

```
df_prev_app_cat_agg = feature_engineering_categorical_variables(df=df_previous_application,
                                              group_var = 'SK_ID_CURR', df_name = 'pre_app').reset_index()
#merge with df_train
df_train=df_train.merge(df_prev_app_cat_agg,on='SK_ID_CURR', how='left' )
```

5. Monthly Cash Dataset

✓ This dataset contains 10 million records and 8 features.

```
df_cash_balance = reduce_memory_usage(pd.read_csv('./data/POS_CASH_balance.csv', encoding= 'unicode_escape'))
print('Number of data points : ', df_cash_balance.shape[0])
print('Number of features : ', df_cash_balance.shape[1])
df_cash_balance.head(1)
```

```
Memory usage of dataframe is 610.43 MB
Memory usage after optimization is: 238.45 MB
Decreased by 60.9%
Number of data points :  10001358
Number of features :  8
```

✓ One column is similar to the train dataset and has been renamed to be unique.

```
#Check bureau and train have similar columns. if so, rename bureau columns, except ID columns
print('Similar columns~~>{}'.format(df_cash_balance.columns.intersection(df_train.columns).tolist()))
```

```
Similar columns~~>['SK_ID_CURR', 'NAME_CONTRACT_STATUS']
```

✓ Feature Cash_Bal_MONTHS_BALANCE has negative values and hence converted to positive values.

```
df_cols = df_cash_balance.loc[:, df_cash_balance.columns.str.contains('BALANCE')]
for col in df_cols:
    print('column {} min value {} and max valaue {}'.format(col, df_cols[col].min(),df_cols[col].max()))
```

```
column Cash_Bal_MONTHS_BALANCE min value -96 and max valaue -1
```

```
df_cash_balance['Cash_Bal_MONTHS_BALANCE']=-df_cash_balance['Cash_Bal_MONTHS_BALANCE']
```

✓ Standard imputation, categorical features extraction, and feature engineering on both types of features are done and merged to Train dataset.

```
# Group by the applicant id, calculate aggregation statistics
df_cash_bal_numerical_agg = numeric_aggregate(df_cash_balance.drop(columns = ['SK_ID_PREV']),
                                              group_var = 'SK_ID_CURR', df_name = 'cash_bal').reset_index(drop=True)

df_train=df_train.merge(df_cash_bal_numerical_agg,on='SK_ID_CURR', how='left' )
df_cash_bal_cat_agg = feature_engineering_categorical_variables(df=df_cash_balance,
                                                                group_var = 'SK_ID_CURR', df_name = 'cash_bal').reset_index()
df_train=df_train.merge(df_cash_bal_cat_agg,on='SK_ID_CURR', how='left' )
```

6. Monthly Credit Data Dataset

✓ This dataset contains monthly credit card details about the customer and has 3.8 million records and 23 features.

```
df_CCard_balance = reduce_memory_usage(pd.read_csv('./data/credit_card_balance.csv', encoding= 'unicode_escape'))
print('Number of data points : ', df_CCard_balance.shape[0])
print('Number of features : ', df_CCard_balance.shape[1])
```

```
Memory usage of dataframe is 673.88 MB
Memory usage after optimization is: 289.33 MB
Decreased by 57.1%
Number of data points :  3840312
Number of features :  23
```

✓ Amount columns are having negative and positive values. For example: Amount_Balance can't be –ve. Similarly AMT_TOTAL_RECEIVABLE column can't be negative. Hence, these columns negative values are set to zero.

```
df_cols = df_CCard_balance.loc[:, df_CCard_balance.columns.str.contains('AMT')]
for col in df_cols:
    print('column {} min value {} and max valaue {}'.format(col, df_cols[col].min(),df_cols[col].max()))
```

```
column AMT_BALANCE min value 0.0 and max valaue 1505902.125
column AMT_CREDIT_LIMIT_ACTUAL min value 0 and max valaue 1350000
column AMT_DRAWINGS_ATM_CURRENT min value -6827.31005859375 and max valaue 2115000.0
column AMT_DRAWINGS_CURRENT min value -6211.6201171875 and max valaue 2287098.25
column AMT_DRAWINGS_OTHER_CURRENT min value 0.0 and max valaue 1529847.0
column AMT_DRAWINGS_POS_CURRENT min value 0.0 and max valaue 2239274.25
column AMT_INST_MIN_REGULARITY min value 0.0 and max valaue 202882.0
column AMT_PAYMENT_CURRENT min value 0.0 and max valaue 4289207.5
column AMT_PAYMENT_TOTAL_CURRENT min value 0.0 and max valaue 4278315.5
column AMT_RECEIVABLE_PRINCIPAL min value -423305.8125 and max valaue 1472316.75
column AMT_RECIVABLE min value -420250.1875 and max valaue 1493338.125
column AMT_TOTAL_RECEIVABLE min value -420250.1875 and max valaue 1493338.125
```

✓ Imputation technique, Feature Engineering on these columns are done and merged to Train dataset by grouping the records at the application level.

```
# Group by the applicant id, calculate aggregation statistics
df_CCard_bal_numerical_agg = numeric_aggregate(df_CCard_balance.drop(columns = ['SK_ID_PREV']),
                                               group_var = 'SK_ID_CURR', df_name = 'CC_bal').reset_index(drop=True)
df_train=df_train.merge(df_CCard_bal_numerical_agg,on='SK_ID_CURR', how='left' )

df_CCard_bal_cat_agg = feature_engineering_categorical_variables(df=df_CCard_balance,
                                                                 group_var = 'SK_ID_CURR', df_name = 'CC_bal').reset_index()
df_train=df_train.merge(df_CCard_bal_cat_agg,on='SK_ID_CURR', how='left' )
```

7. Installment Payments Dataset

- ✓ This dataset contains information about customer installment histories of previous application. It has 3 million records and 8 features.

```
df_Installment_payments = reduce_memory_usage(pd.read_csv('./data/installments_payments.csv', encoding= 'unicode_escape'))
print('Number of data points : ', df_Installment_payments.shape[0])
print('Number of features : ', df_Installment_payments.shape[1])
df_Installment_payments.head(1)
```

```
Memory usage of dataframe is 830.41 MB
Memory usage after optimization is: 311.40 MB
Decreased by 62.5%
Number of data points :  13605401
Number of features :  8
```

- ✓ Converted Days Feature to positive as it contains all values in –ve. Days can't be –ve.

```
df_cols = df_Installment_payments.loc[:, df_Installment_payments.columns.str.contains('INSTALMENT')]
for col in df_cols:
    print('column {} min value {} and max valaue {}'.format(col, df_cols[col].min(),df_cols[col].max()))
```

```
column NUM_INSTALMENT_VERSION min value 0.0 and max valaue 178.0
column NUM_INSTALMENT_NUMBER min value 1 and max valaue 277
column DAYS_INSTALMENT min value -2922.0 and max valaue -1.0
column AMT_INSTALMENT min value 0.0 and max valaue 3771487.75
```

```
df_cols = df_Installment_payments.loc[:, df_Installment_payments.columns.str.contains('DAY')]
for col in df_cols:
    print('column {} min value {} and max valaue {}'.format(col, df_cols[col].min(),df_cols[col].max()))
```

```
column DAYS_INSTALMENT min value 1.0 and max valaue 2922.0
column DAYS_ENTRY_PAYMENT min value -4920.0 and max valaue -1.0
```

- ✓ Standard Imputation techniques, converting features into statistical features, and merging to Train dataset is done.

```
# Group by the applicant id, calculate aggregation statistics
df_Install_payment_numerical_agg = numeric_aggregate(df_Installment_payments.drop(columns = ['SK_ID_PREV']),
                                                     group_var = 'SK_ID_CURR', df_name = 'Install_pay').reset_index(drop=True)

df_train=df_train.merge(df_Install_payment_numerical_agg,on='SK_ID_CURR', how='left' )
```

- ✓ Since all the datasets consume more memory, it is important to shrink the dataset at the time of loading, and explicit garbage collection is done as soon as it is merged to Train dataset.

```
gc.enable()
del df_Installment_payments,df_Install_payment_numerical_agg
gc.collect()
```

```
0
```

- ✓ Final Train dataset contains 307k records and 896 features. This includes ID column and a class label column as well.

8. Vectorization

- ✓ After preprocessing the raw data and merging all the features into one dataset, the features are to be vectorized for the machine learning process.
- ✓ The dataset contains categorical and numerical features only.
- ✓ Categorical vectorization: Categorical vectorization can be done in multiple ways like one-hot encoding, Label Encoding, and response encoding. One hot encoding, Label Encoding are sparse matrix, response encoding is dense matrix. So, let us try to understand the unique values of each categorical values.

```
# index of the categorical features
cat_indexes = [column_index for column_index, column_name in enumerate(df_train.columns) if column_name in categorical_columns]
# dimensions of each categorical feature
cat_dimensions = df_train[categorical_columns].nunique().tolist()
print(cat_dimensions)
```

```
[2, 3, 2, 2, 7, 8, 5, 6, 6, 18, 7, 58, 4, 3, 7, 2, 4, 4, 11, 8, 4, 7, 2, 25, 4, 4, 9, 7, 4, 27, 5, 3, 8, 11, 5, 17, 7, 5]
```

- ✓ Certain categorical features are having more categorical values. Hence, response encoding technique is applied. The class labels are binary, response encoding is created per class labels. Response encoding tries to create probability value per class on the train dataset. If the categorical type, doesn't have any values present in the training dataset, both classes [0, 1] will have equal weightage [0.5, 0.5].

```
#There are many categorical columns have more number of categories.
#Hence, tried to implement response coding for categorical values which will minimise the # of features
#compare to oneHotencoding.

#reference:appliedaicourse.com
def response_table(x_tr,feature,y_tr):
    r_table = dict()
    unique_cat_feature=x_tr[feature].unique()
    for i in range(len(unique_cat_feature)):
        total = x_tr.loc[:,feature][(x_tr[feature]==unique_cat_feature[i])].count()
        p_0   = x_tr.loc[:,feature][((x_tr[feature]==unique_cat_feature[i])&(y_tr==0))].count()
        p_1   = x_tr.loc[:,feature][((x_tr[feature]==unique_cat_feature[i])&(y_tr==1))].count()
        r_table[unique_cat_feature[i]]=[p_0/total,p_1/total]
    return r_table
```

```
def replace_with_prob(df,feature,r_table):
    final_list=list()
    features=df[feature]
    for i in features:
        if i in r_table.keys():
            final_list.append(r_table[i])
        else: #unseen data equal weightage.
            final_list.append([0.5,0.5])
    return np.asarray(final_list)
```

- ✓ All the categorical values are converted into response encoding and combined using the horizontal stack function.

```
#Below method will transform all categorical values to
def convert_cat_to_vector(cat_columns,x_tr,x_te,y_tr):
    cat_feature_names=list()
    #This method is to build all categorical features into numerical feature
    for idx,feature in enumerate(tqdm(cat_columns)):
        rtable = response_table(x_tr,feature,y_tr)
        tr_cat_prob = replace_with_prob(x_tr,feature,rtable)
        te_cat_prob = replace_with_prob(x_te,feature,rtable)
        cat_feature_names.append(feature+'_0')
        cat_feature_names.append(feature+'_1')
        if idx==0: # first categorical features
            tr_cat_vector  = coo_matrix(tr_cat_prob[:,0].reshape(-1,1))
            tr_cat_vector  = hstack((tr_cat_vector,coo_matrix(tr_cat_prob[:,1].reshape(-1,1))))

            te_cat_vector  = coo_matrix(te_cat_prob[:,0].reshape(-1,1))
            te_cat_vector  = hstack((te_cat_vector,coo_matrix(te_cat_prob[:,1].reshape(-1,1))))
        else:
            tr_cat_vector  = hstack((tr_cat_vector,coo_matrix(tr_cat_prob[:,0].reshape(-1,1))))
            tr_cat_vector  = hstack((tr_cat_vector,coo_matrix(tr_cat_prob[:,1].reshape(-1,1))))

            te_cat_vector  = hstack((te_cat_vector,coo_matrix(te_cat_prob[:,0].reshape(-1,1))))
            te_cat_vector  = hstack((te_cat_vector,coo_matrix(te_cat_prob[:,1].reshape(-1,1))))

    return tr_cat_vector,te_cat_vector,cat_feature_names
```

```
# y class lables.
y = df_train['TARGET'].values
#categorical featuers only
X =df_cat

#split data only train and test.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y,random_state=42)

#invoke categorical to vector

#invoke categorical response coding method to convert categorical values to numerical vector
tr_cat_vect,te_cat_vect,cat_feature_names= convert_cat_to_vector(cat_columns=categorical_columns,x_tr=X_train,x_te=X_test,y_tr=y_
```

```
100%|██████████████████████████████████████████████████████████████████████████| 38/38 [00:27<00:00,  1.38it/s]
```

✓ Numerical columns are vectorized using StandardScaler. StandardScaler is much more robust to new data than min-max scaler technique. Like categorical, all numerical categorical features are converted to standardscaler and combined using the horizontal stacking function.

- y = (x – mean) / standard_deviation

Where the *mean* is calculated as:

- mean = sum(x) / count(x)

And the *standard_deviation* is calculated as:

- standard_deviation = sqrt(sum((x – mean)^2) / count(x))

```
def convert_num_to_vector(num_columns,x_tr,x_te):
    # normalizer.fit(X_train['price'].values)
    # this will rise an error Expected 2D array, got 1D array instead:
    # array=[105.22 215.96  96.01 ... 368.98  80.53 709.67].
    # Reshape your data either using
    # array.reshape(-1, 1) if your data has a single feature
    # array.reshape(1, -1)  if it contains a single sample.
    #Imputation of numerical data
    stdscaler = StandardScaler(with_mean=True, with_std=True)
    feature_names=list()

    for idx,feature in enumerate(tqdm(num_columns)):
        #fit and transform only for training set
        X_tr_std_col = stdscaler.fit_transform(x_tr[feature].values.reshape(-1,1))
        #transform test datapoints
        X_te_std_col = stdscaler.transform(x_te[feature].values.reshape(-1,1))
        #feature names list
        feature_names.append(feature)
        if idx==0:
            tr_num_vector = X_tr_std_col
            te_num_vector = X_te_std_col
        else:
            tr_num_vector = np.hstack((tr_num_vector,X_tr_std_col))
            te_num_vector = np.hstack((te_num_vector,X_te_std_col))
    return  tr_num_vector,  te_num_vector,feature_names
#convert numerical fetures to vector form using normalizer
tr_num_vect,te_num_vect,num_feature_names= convert_num_to_vector(num_columns=numerical_columns,
                                                                 x_tr=X_train,x_te=X_test)
```

```
100%|██████████████████████████████████████████████████████████████████████████| 961/961 [05:46<00:00,  2.77it/s]
```

- ✓ After vectorizing, the number of features got increased to 1037 due to 2 class labels for response encoding.

```
print("Final Data matrix ...")
print("="*25)
print(X_train_vector.shape, y_train.shape)
print(X_test_vector.shape, y_test.shape)
print(len(num_feature_names))
```

```
Final Data matrix ...
=========================
(206032, 1037) (206032,)
(101479, 1037) (101479,)
1037
```

- ✓ After vectorization, features, class labels, and features are saved in separate files. These files will be used for all supervised model evaluation techniques.

Save vectors in pickle file.

```
import pickle
with open('./data/train_vector.pkl', 'wb') as f:
    pickle.dump(X, f)
```

```
with open('./data/yvalues.pkl', 'wb') as yvalues:
    pickle.dump(y, yvalues)
```

Save Feature names

```
with open('./data/feature_names.pkl', 'wb') as feature:
    pickle.dump(num_feature_names, feature)
```

9. Modeling

1. Logistic Regression

- ✓ The dataset is binary classification and class labels are given. Hence, first Logistic Regression is most appropriate to start with and treated as base model. Also the dataset has more number of features and records, LR performs very well on higher dimensions.
- ✓ Vectorized data is split into test and train datasets with percentages of 33% and 67%.
- ✓ Hyperparameter tuning is done with cross-validation and hence validation data didn't split.

```
#split data only train and test.
#Hypertuning with gridsearch and random
#hypertuning, will do automatic cv. Hence, split data into Train and Test only.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, stratify=y,random_state=42)
```

- ✓ SGDClassifier is an optimization technique with "log" loss function works as Logistic Regression.
- ✓ The dataset is imbalanced, hence class weight is applied.

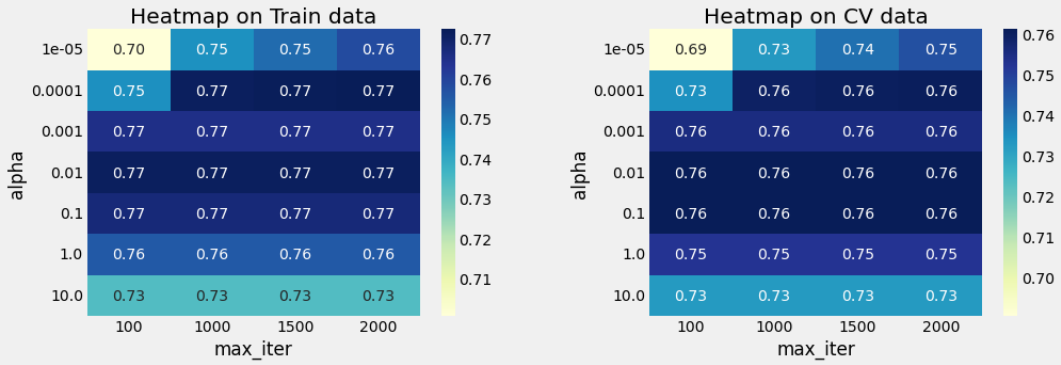
```
from sklearn.utils import class_weight
#class_weights = class_weight.compute_class_weight('balanced', np.unique(y_train),y_train)
#class_weights = dict(zip(np.unique(y_train), class_weight.compute_class_weight('balanced', np.unique(y_train),y_train)))
#y_integers = np.argmax(y_train, axis=1)
class_weights = class_weight.compute_class_weight(class_weight='balanced', classes=[0,1], y=y_train)
d_class_weights = dict(enumerate(class_weights))
d_class_weights
```

```
{0: 0.5439099467262235, 1: 6.193470811038297}
```

- ✓ Hyper parameter tuning is done with alpha, max_iter and penalty. Class weight fixed as balanced. GridSearchCV with 5 fold cross-validation is used.

```
#LogisticRegression and SGDClassifier linear models to train nonzero features
def optimize_lr(x,y):
    parameters_svm = {'alpha':[0.00001,0.0001,0.001,0.01,0.1,1,10],
                      'max_iter' : [100, 1000,1500,2000],
                      'penalty': ['l1', 'l2']
                     }
    clf=SGDClassifier(class_weight='balanced',learning_rate='optimal', eta0=0.0,
                      loss='log', random_state=42,n_jobs=-1)
    model = GridSearchCV(clf, parameters_svm,scoring='roc_auc',
                         verbose=0,cv=5, return_train_score=True)
    model.fit(X=x,y=y)
    results = pd.DataFrame.from_dict(model.cv_results_)
    return results, model
results_lr,model_lr = optimize_lr(x=X_train,y=y_train)
```

- ✓ Hyperparameter results are plotted in heatmap.



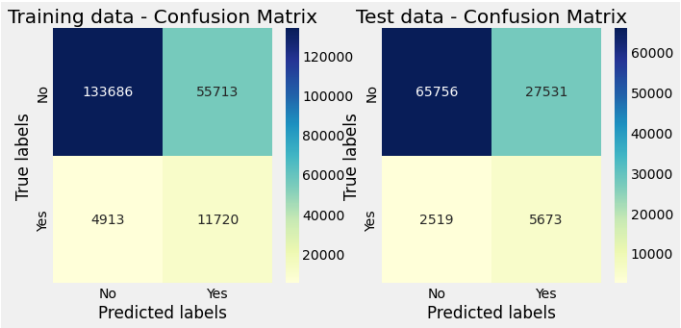
- ✓ GridSearchCV took more than 24 hours and the best estimator is

```
: model_lr.best_estimator_

: SGDClassifier(alpha=0.01, class_weight='balanced', loss='log', max_iter=100,
                n_jobs=-1, random_state=42)
```

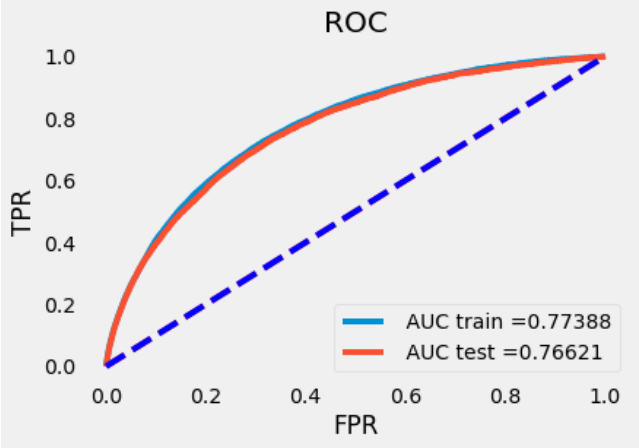
- ✓ Model is retrained with best estimators parameters and predicted both train and test data [un seen data].

```
#based on the best parameters, predict values and plot AUC  and return the model
def measure_accuracy(X_tr,X_te,y_tr,y_te):
    clf=SGDClassifier(class_weight=d_class_weights,learning_rate='optimal', eta0=0.0,
                      loss='log', random_state=42,n_jobs=-1,verbose=0,
                     alpha=0.01,max_iter=400)

    #clf.set_params(**best_model.best_params_)

    clf.fit(X_tr, y_train)

    y_tr_pred = batch_predict(clf, X_tr)
    y_te_pred = batch_predict(clf, X_te)

    plot_confusionmatrix(y_tr,y_tr_pred,y_te,y_te_pred)
    print('='*70)
    auc_train,auc_test=draw_roccurve(y_tr,y_tr_pred,y_te,y_te_pred)
    print('='*70)
    return clf, auc_train,auc_test
```

- ✓ and the results are plotted with confusion matrix and ROC Curve



- ✓ Prediction is very decent results yielded.
- ✓ FP is higher than FN. As per the metric, the model is tried to minimize the FN, which is good.

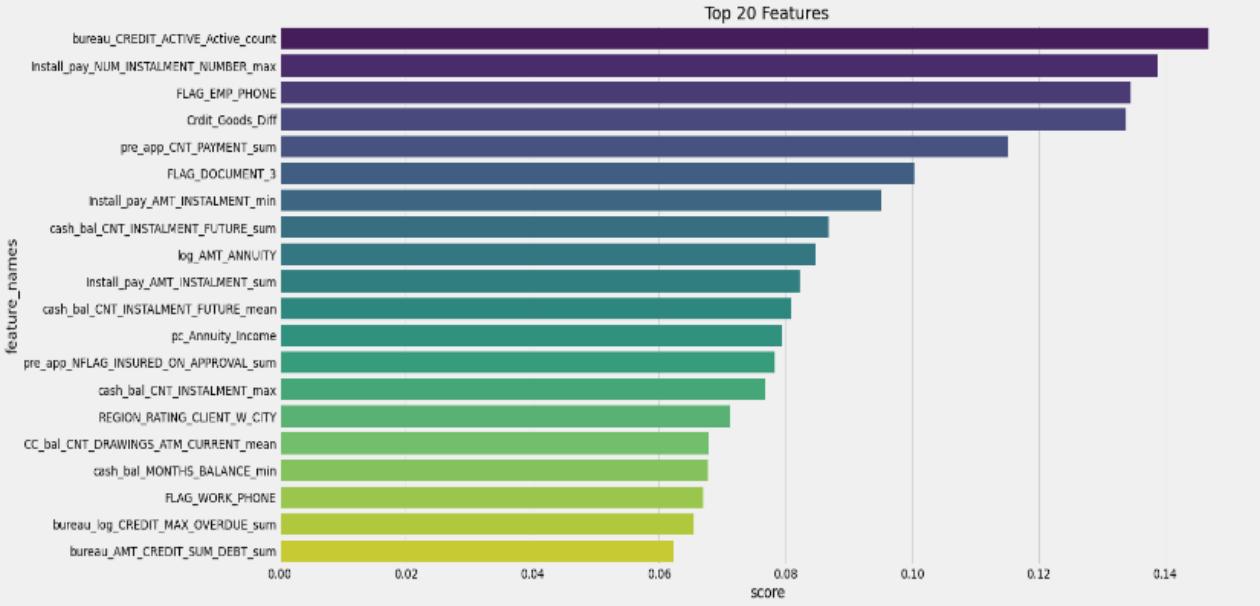


- ✓ From the ROC, Model is well balanced with train and test data. Both train and test results have yielded 77%

- ✓ From the classification report, F1-score of class 0 is 82% and class 1 is 28%. On the train dataset and test dataset is also very close to train 81% and 28% respective classes.

```
Train Results
******************************************************
              precision    recall  f1-score   support

           0       0.96      0.71      0.82    189399
           1       0.17      0.70      0.28     16633

    accuracy                           0.71    206032
   macro avg       0.57      0.71      0.55    206032
weighted avg       0.90      0.71      0.77    206032

******************************************************
Test Results
******************************************************
              precision    recall  f1-score   support

           0       0.96      0.70      0.81     93287
           1       0.17      0.69      0.27      8192

    accuracy                           0.70    101479
   macro avg       0.57      0.70      0.54    101479
weighted avg       0.90      0.70      0.77    101479
```

- ✓ Top 20 features are extracted from the model



- ✓ Model is interpretable and based on the coefficients, it gives the Feature importance.

Feature	Score
bureau_CREDIT_ACTIVE_Active_count	0.1469
Install_pay_NUM_INSTALMENT_NUMBER_max	0.1388
FLAG_EMP_PHONE	0.1346
Crdit_Goods_Diff	0.1338
pre_app_CNT_PAYMENT_sum	0.1152
FLAG_DOCUMENT_3	0.1004
Install_pay_AMT_INSTALMENT_min	0.0951
cash_bal_CNT_INSTALMENT_FUTURE_sum	0.0868
log_AMT_ANNUITY	0.0847
Install_pay_AMT_INSTALMENT_sum	0.0823
cash_bal_CNT_INSTALMENT_FUTURE_mean	0.0808
pc_Annuity_Income	0.0794
pre_app_NFLAG_INSURED_ON_APPROVAL_sum	0.0783
cash_bal_CNT_INSTALMENT_max	0.0769
REGION_RATING_CLIENT_W_CITY	0.0712
CC_bal_CNT_DRAWINGS_ATM_CURRENT_mean	0.0678
cash_bal_MONTHS_BALANCE_min	0.0677
FLAG_WORK_PHONE	0.067
bureau_log_CREDIT_MAX_OVERDUE_sum	0.0655
bureau_AMT_CREDIT_SUM_DEBT_sum	0.0623

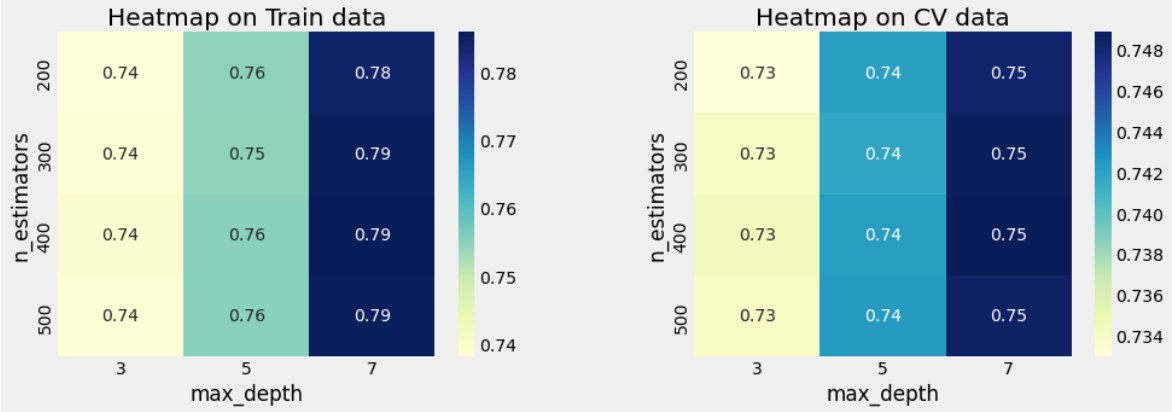
- ✓ From the table above, additional datasets like bureau install payments, previous application, Monthly cash, Credit Card details are provided are played an important role.
- ✓ Domain-based features like percentage Annuity Income, Credit_Good_difference, log transformation, and statistical transformation like sum, count, mean are also played important.

2. Random Forest

- ✓ Stacking ensemble model Random Forest is tried with the following parameters. N_estimators, max_depth and criterion.
- ✓ Hyperparameter tuning is done with gridsearchcv and 3 fold cross validation.

```
#credit:https://www.kaggle.com/saurabhshahane/lgbm-hyperparameter-tuning-with-optuna-beginners
#credit:https://towardsdatascience.com/kagglers-guide-to-lightgbm-hyperparameter-tuning-with-optuna-in-2021-ed048d9838b5

def optimize_rf(x,y):
    clf = ensemble.RandomForestClassifier(n_jobs=-1)

    #default method = Random
    param_grid = {
                  "n_estimators": [200,300,400,500],
                  "max_depth": [3,5,7],
                  "criterion":['gini','entropy'],
                 }
    model = model_selection.GridSearchCV(
            estimator=clf, param_grid=param_grid,scoring='roc_auc',
            verbose=0,cv=3, n_jobs=1, return_train_score=True)
    model.fit(X=x,y=y)
    results = pd.DataFrame.from_dict(model.cv_results_)
    return results, model
results_rf,model_rf = optimize_rf(x=X_train,y=y_train)
```

- ✓ Results are plotted using heatmap



- ✓ The best estimator is

```
model_rf.best_estimator_
```

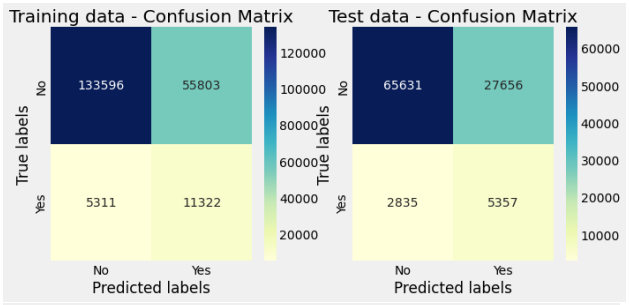
```
RandomForestClassifier(criterion='entropy', max_depth=7, n_estimators=400,
                       n_jobs=-1)
```

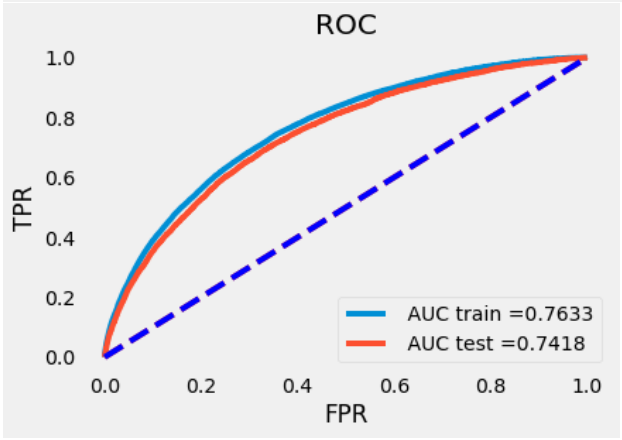
✓ Model is trained with best parameters and weight class is applied.

```
#based on the best parameters, predict values and plot AUC  and return the model
def measure_accuracy(X_tr,X_te,y_tr,y_te):
    clf=SGDClassifier(class_weight=d_class_weights,learning_rate='optimal', eta0=0.0,
                      loss='log', random_state=42,n_jobs=-1,verbose=0,
                     alpha=0.01,max_iter=400)

    #clf.set_params(**best_model.best_params_)

    clf.fit(X_tr, y_train)

    y_tr_pred = batch_predict(clf, X_tr)
    y_te_pred = batch_predict(clf, X_te)

    plot_confusionmatrix(y_tr,y_tr_pred,y_te,y_te_pred)
    print('='*70)
    auc_train,auc_test=draw_roccurve(y_tr,y_tr_pred,y_te,y_te_pred)
    print('='*70)
    return clf, auc_train,auc_test,y_tr_pred,
```

```
#measure accuracy hypertuned with RandomSearch
#rf_model,auc_tr_rf_model,auc_te_rf_model = measure_accuracy(model_rf,X_train,X_test,y_train,y_test)
rf_model,auc_tr_rf_model,auc_te_rf_model = measure_accuracy(X_train,X_test,y_train,y_test)
```

✓ Confusion Matrix and AUC of Train and Test datasets





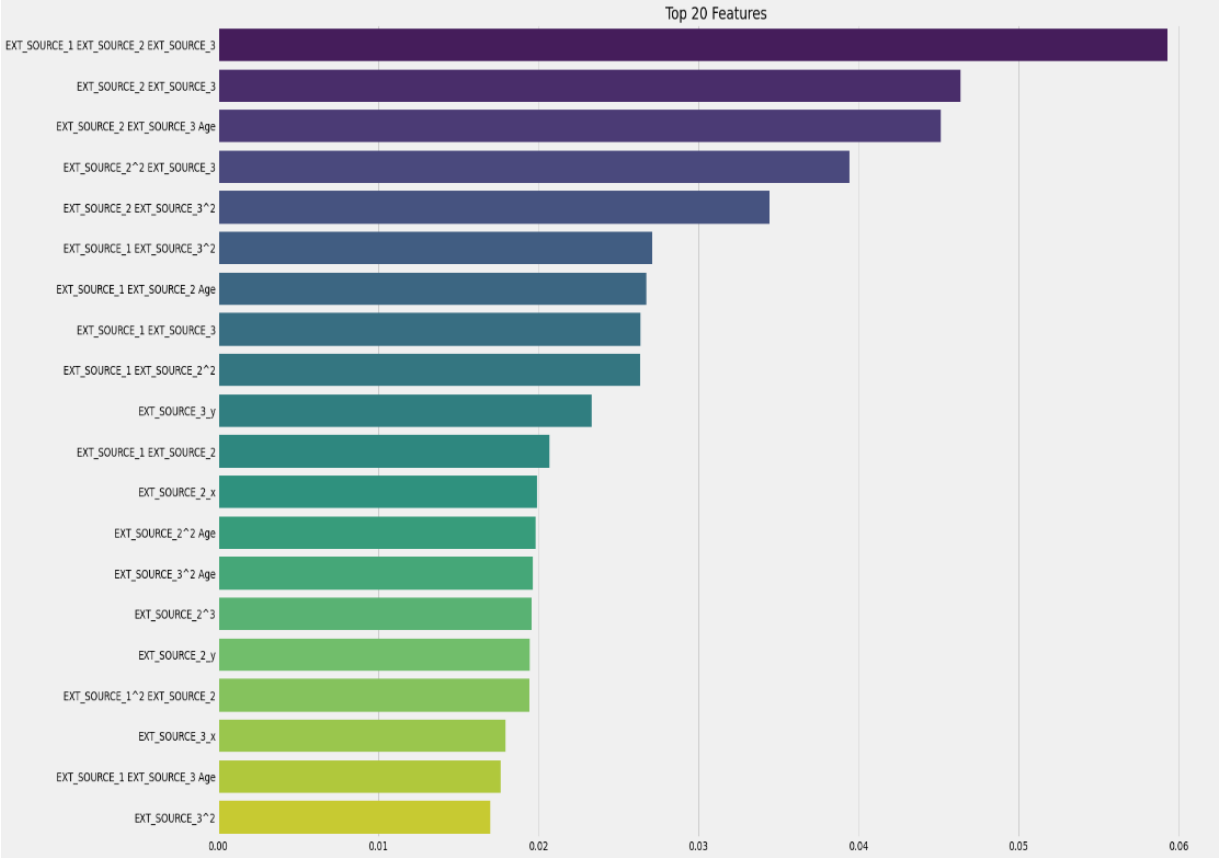
✓ RF with class labels are balanced, it decent job and FN is smaller compared to FP.

✓ Train AUC is 76% and Test AUC 74%.

✓ The Model is not overfitted. However, compared to LR, performance is less. Maybe further tuning, would improve.

✓ Model training hours took more than LR Model. Hence, this model left as is and moved to boosting ensemble models

✓ Feature importance



✓ Feature importance-wise, External source 1, 2, and 3 and related polynomial features are dominated. Hence, further tuning dropped.

3. XGBoost

✓ XGBoost is another ensemble model that uses the GBDT technique.

✓ scale_pos_weight is the ratio of a number of negative classes to the positive class is used to balance the dataset.

```
#Define weight ratio
weight_ratio = float(len(y_train[y_train == 0]))/float(len(y_train[y_train ==1]))
w_array = np.array([1]*y_train.shape[0])#positive class
w_array[y_train==1] = weight_ratio
w_array[y_train==0] = 1- weight_ratio #negative class
```

✓ XGBoost consumes more time than LR, Random Forest. Also GridSearch hyperparameter tuning takes more time. Hence, optuna is tried. It uses different algorithms, such as GridSearch, Random Search, Bayesian and Evolutionary algorithms to find the optimal hyperparameter values.

```
from warnings import simplefilter
simplefilter("ignore", category=RuntimeWarning)

optuna.logging.set_verbosity(optuna.logging.WARNING)
study = optuna.create_study(direction="minimize", study_name="XGBoost Classifier" )
func = lambda trial: objective(trial, X_train, y_train)
study.optimize(func, n_trials=100)
```

✓ Using StratifiedKFold 5 fold cross validation is done

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
cv_scores = np.empty(5)

for idx, (train_idx, valid_idx) in enumerate(cv.split(X_data, y_label)):
    x_train, x_valid = X_data[train_idx], X_data[valid_idx]
    y_train, y_valid = y_label[train_idx], y_label[valid_idx]

    #Define weight ratio
```

✓ XGBoost has multiple parameters.

```
def objective(trial, X_data, y_label):
    param_grid = {
        "verbosity": 0,
        "objective" : trial.suggest_categorical("objective", ['binary:logistic']),
        "n_estimators": trial.suggest_int('n_estimators', 50, 10000),
        "learning_rate": trial.suggest_categorical('learning_rate', [0.008,0.009,0.01,0.012,0.014,0.016,0.018, 0.02]),
        "alpha": trial.suggest_loguniform('alpha', 1e-3, 10.0),
        'gamma': trial.suggest_loguniform('gamma', 1e-3, 10.0),
        "lambda": trial.suggest_loguniform('lambda', 1e-3, 10.0),
        "min_child_weight": trial.suggest_int('min_child_weight', 1, 200),
        'subsample' : trial.suggest_loguniform('subsample', 0.5, 1),
        'colsample_bytree': trial.suggest_loguniform('colsample_bytree', 0.5, 1),
        'colsample_bylevel': trial.suggest_loguniform('colsample_bylevel', 0.5, 1),
        "max_depth": trial.suggest_categorical('max_depth', [5,7,9,11,13,15,17,20]),
        }
```

✓ Best results obtained using optuna is

```
print(f"\tBest value (rmse): {study.best_value:.5f}")
print(f"\tBest params:")

for key, value in study.best_params.items():
    print(f"\t\t{key}: {value}")
```

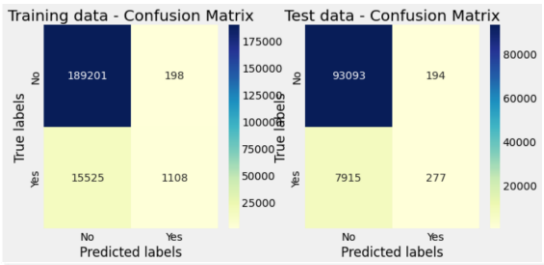
```
        Best value (rmse): 0.66379
        Best params:
                objective: binary:logistic
                n_estimators: 5509
                learning_rate: 0.008
                alpha: 0.0014764404145375422
                gamma: 0.006709171934295603
                lambda: 0.004841559984476445
                min_child_weight: 18
                subsample: 0.801485530162779
                colsample_bytree: 0.8775747304615185
                colsample_bylevel: 0.5456219632677068
                max_depth: 20
```

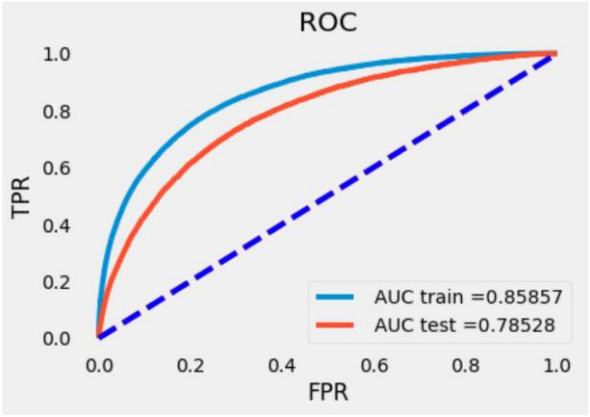
✓ With the best results, it is overfitting. Hence, using these parameters, values are further trimmed to not overfitting.

```
param_grid = {
    "verbosity": 0,
    "objective" :'binary:logistic',
    "n_estimators":10000,
    "learning_rate": 0.001,
    "alpha": 0.001,
    'gamma': 0.005,
    "lambda": 0.004,
    "min_child_weight": 20,
    'subsample' : 0.8,
    'colsample_bytree': 0.9,
    'colsample_bylevel': 0.5,
    "max_depth": 7,
    }
```

```
clf = XGBClassifier(tree_method = "exact",predictor = "cpu_predictor",
                    sample_weight=d_class_weights,
                    random_state=42)
```

✓ Confusion Matrix and ROC Curve of XGBoost:

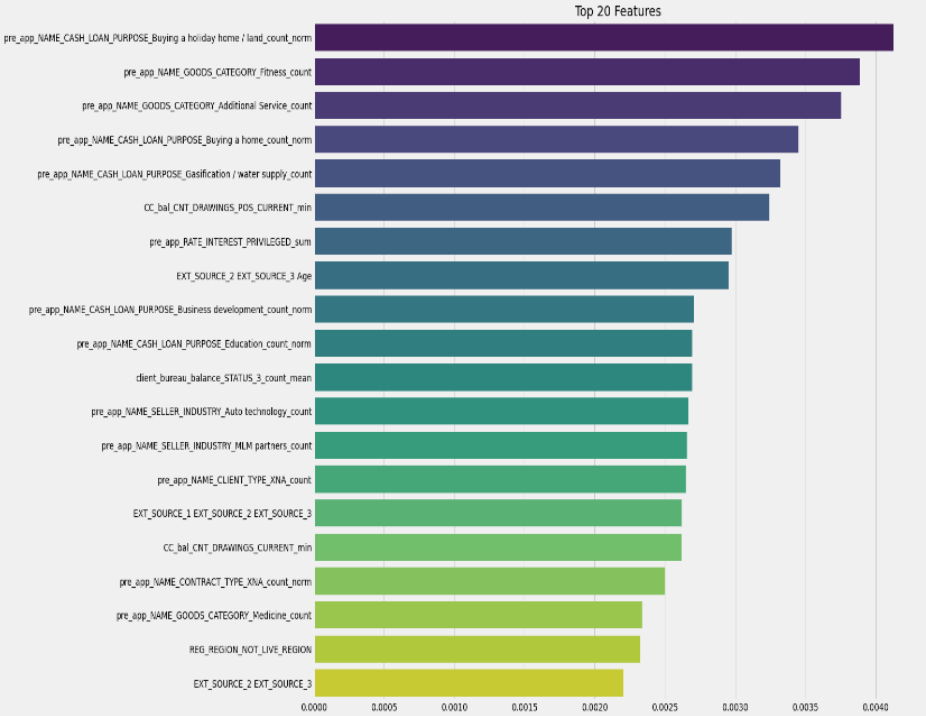




- ✓ From the confusion matrix, the model performance is worst.
- ✓ Class Balance would be wrong. Further tuning is required.

- ✓ Though, the train and test AUC is better than RF and LR, the predicted classes are worst. Sofar, XGBoost performance is very low. Mostly it could be error in code.

✓ Feature Importance:



✓ Compare to all the previous models, XGBoost took more time and hyper tuning runs more than 24 hours, decided to move on and try with LightGBM GBDT Model.

4. **LightGBM**
   - ✓ This model is again ensamble GBDT technique only. However, it runs faster than XGBoost.
   - ✓ Optuna hyperparameter tuning, class weights are implemented. Like XGBoost, this model also has lot of parameters,

```
param_grid = {
    "verbose":-1, "silent":True,
    "n_estimators": trial.suggest_categorical("n_estimators", [10000]),
    "learning_rate": trial.suggest_float("learning_rate", 0.001, 0.3),
    'min_child_samples': trial.suggest_int('min_child_samples', 5, 100),
    "num_leaves": trial.suggest_int("num_leaves", 20, 3000, step=20),
    "max_depth": trial.suggest_int("max_depth", 3, 12),
    "min_data_in_leaf": trial.suggest_int("min_data_in_leaf", 200, 10000, step=100),
    "max_bin": trial.suggest_int("max_bin", 200, 300),
    "lambda_l1": trial.suggest_int("lambda_l1", 0, 100, step=5),
    "lambda_l2": trial.suggest_int("lambda_l2", 0, 100, step=5),
    "min_gain_to_split": trial.suggest_float("min_gain_to_split", 0, 15),
    "feature_fraction": trial.suggest_float("feature_fraction", 0.6, 0.9, step=0.1),
    "bagging_fraction": trial.suggest_float("bagging_fraction", 0.5, 0.7, step=0.1),
    "bagging_freq": trial.suggest_int('bagging_freq', 1, 7),

    }
```

   - ✓ 5 fold Cross validation used with the help of StratifiedKFold

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

cv_scores = np.empty(5)
for idx, (train_idx, test_idx) in enumerate(cv.split(X_data, y_label)):
    x_train, x_test = X_data[train_idx], X_data[test_idx]
    y_train, y_test = y_label[train_idx], y_label[test_idx]
```

   - ✓ LightGBMPruningCallback method is used to prune the tuning.

```
clf_model.fit(x_train,y_train,eval_set=[(x_test,y_test)],eval_metric='binary_logloss',early_stopping_rounds=10,
            callbacks=[LightGBMPruningCallback(trial, "binary_logloss")])
```

   - ✓ Optuna study was run with 70 trials.

```
from warnings import simplefilter
simplefilter("ignore", category=RuntimeWarning)
warnings.filterwarnings('ignore')

optuna.logging.set_verbosity(optuna.logging.WARNING)

study = optuna.create_study(direction="minimize", study_name="LGBM Classifier")
optuna.logging.set_verbosity(optuna.logging.WARNING)
func = lambda trial: objective(trial, X_train, y_train)
study.optimize(func, n_trials=70,callbacks=[logging_callback])
```

- ✓ Optuna Study results:

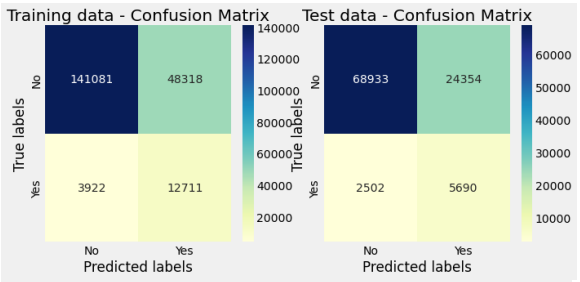
```
print(f"\tBest value (loss minimize): {study.best_value:.5f}")
print(f"\tBest params:")

for key, value in study.best_params.items():
    print(f"\t\t{key}: {value}")
```

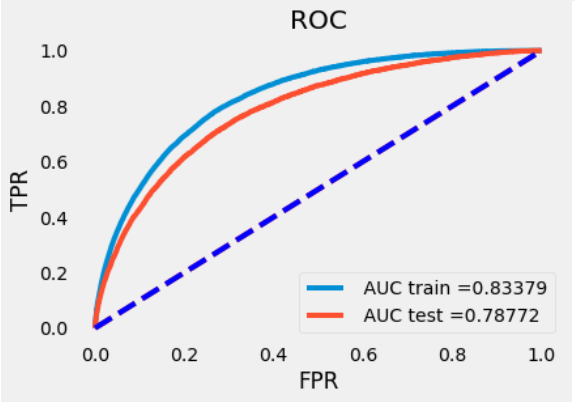
```
        Best value (rmse): 0.33096
        Best params:
                n_estimators: 10000
                learning_rate: 0.2726951856436256
                min_child_samples: 23
                num_leaves: 240
                max_depth: 11
                min_data_in_leaf: 500
                max_bin: 237
                lambda_l1: 0
                lambda_l2: 55
                min_gain_to_split: 1.7921437338679456
                feature_fraction: 0.7
                bagging_fraction: 0.6
                bagging_freq: 2
```

- ✓ The best parameters are overfitting, tuned manually further to reduce the overfitting.

```
clf = lgb.LGBMClassifier(
                        n_estimators=10000,learning_rate=0.001756951856436256,
                        min_child_samples=45,num_leaves=1500,max_depth=7,
                        min_data_in_leaf=2500,max_bin=230,lambda_l1= 5,lambda_l2=45,
                        min_gain_to_split=4.25,
                        feature_fraction=0.8,bagging_fraction=0.6,bagging_freq= 3,
                        class_weight=d_class_weights
                       )
params={'verbose':-1,'verbose_eval':False}
clf.set_params(**params)
```

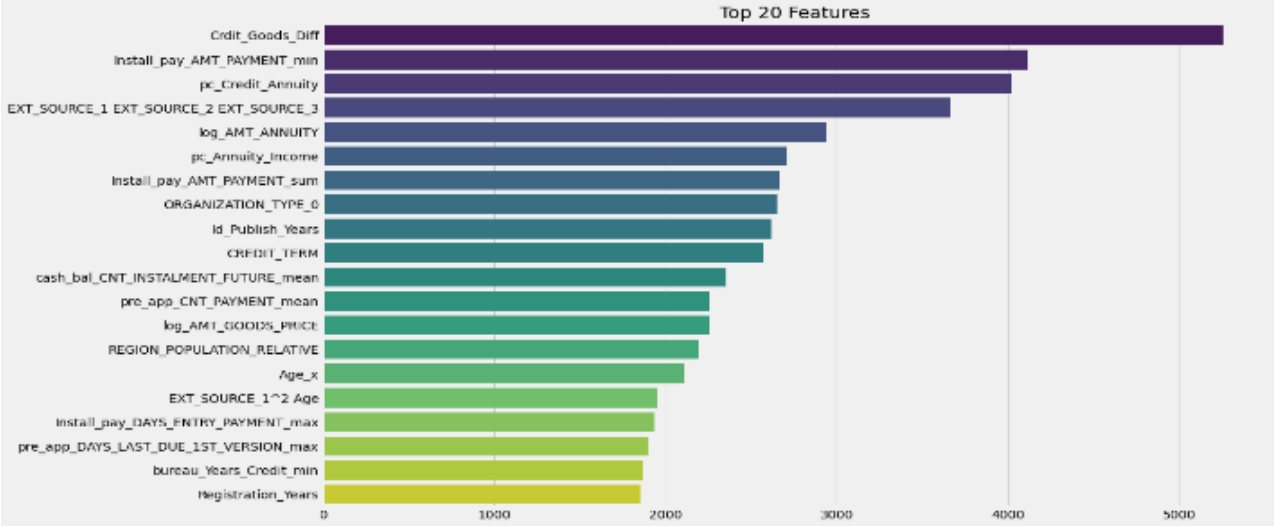
- ✓ Model is trained with the above parameters and predicted both train and test [unseen] data
- ✓ Plotted confusion matrix and ROC Curve.





- ✓ Sofar, is the best model and true predicted class label are high and FN is much lesser compared to previous models.
- ✓ Train AUC is 83% and Test 79% which is higher than the previous models.
- ✓ Slightly overfitted, further tuning, overfitting can be reduced.

✓ Future Importance



Feature	Score
Crdit_Goods_Diff	5251
Install_pay_AMT_PAYMENT_min	4113
pc_Credit_Annuity	4021
EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3	3662
log_AMT_ANNUITY	2938
pc_Annuity_Income	2704
Install_pay_AMT_PAYMENT_sum	2666
ORGANIZATION_TYPE_0	2651
Id_Publish_Years	2618
CREDIT_TERM	2569
cash_bal_CNT_INSTALMENT_FUTURE_mean	2352
pre_app_CNT_PAYMENT_mean	2255
log_AMT_GOODS_PRICE	2254
REGION_POPULATION_RELATIVE	2192
Age_x	2110
EXT_SOURCE_1^2 Age	1951
Install_pay_DAYS_ENTRY_PAYMENT_max	1938
pre_app_DAYS_LAST_DUE_1ST_VERSION_max	1897
bureau_Years_Credit_min	1870
Registration_Years	1852

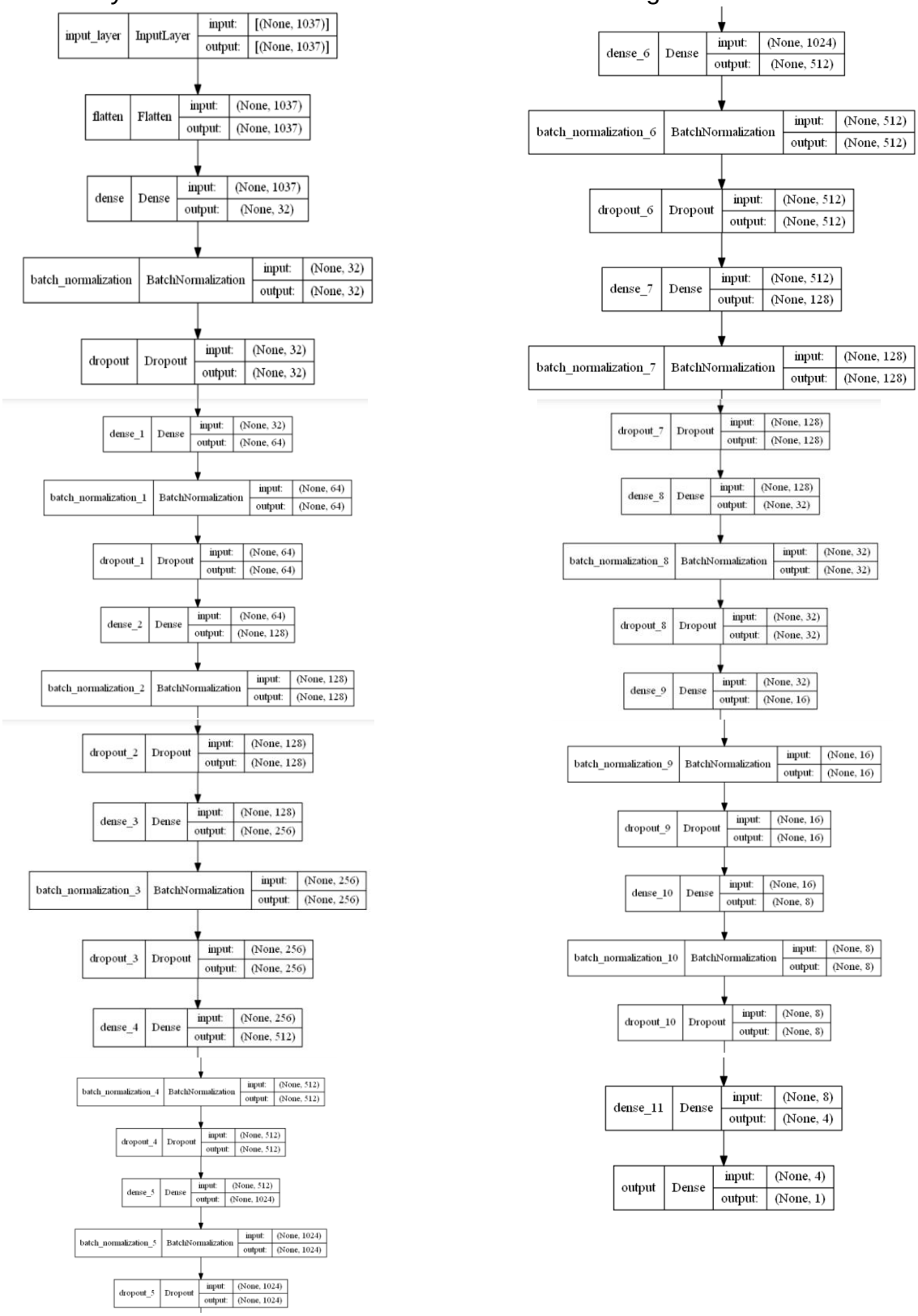
✓ Like Logistic Regression, Feature Importance also from various datasets, domain based, polynomial based and statistical based features are played good role.

✓ Among all supervised models, LightGBM performance is much better compare to others. Also, training time, compare to other ensemble models, this is better. Run time sofar LR linear model is better than any ensemble models.

✓ The state of art technique is deep learning, hence decided to try ANN model.

5. Deep Learning ANN

- ✓ Fully Connected Neural Network tried with following architecture.



✓ Input layer designed with glorot normal, LeakyRelu, and L2 Regularizer

```
#initializer
initializer = tf.keras.initializers.glorot_normal(seed=0)
#initializer =tf.keras.initializers.he_normal(seed=0)
act_layer = tf.keras.layers.LeakyReLU(alpha=0.0655)
l2_reg =l2(0.0012)
#Input Layer: Categorical features, Numerical features
input_layer = Input(shape=(X_train.shape[1],),name='input_layer')

#flat the input layer
flat_layer = Flatten()(input_layer)
```

✓ FCN designed with Dense, BatchNormalization and Dropout

```
# Dense layer 1 Layer
Dense1 =Dense(32, activation=act_layer,kernel_initializer=initializer,kernel_regularizer=l2_reg)(flat_layer)
#batch Normalization
bn1 = BatchNormalization()(Dense1)
# dropout after batch norm 1
drop1 =Dropout(0.25)(bn1)
```

✓ FCN layer increased to 64, 128, 256 hidden layers with BN and dropout of 25%.

```
# Dense layer after drop 3
Dense4 =Dense(256, activation=act_layer,kernel_initializer=initializer,kernel_regularizer=l2_reg)(drop3)
#batch Normalization
bn4 = BatchNormalization()(Dense4)
# dropout after batch norm 4
drop4 =Dropout(0.25)(bn4)
```

✓ Further FCN layer increased to 512 and 1024 with a dropout of 40% and 50%.

```
# Dense layer after drop 4
Dense5 =Dense(512, activation=act_layer,kernel_initializer=initializer,kernel_regularizer=l2_reg)(drop4)
#batch Normalization
bn5 = BatchNormalization()(Dense5)
# dropout after batch norm 5
drop5 =Dropout(0.4)(bn5)
```

```
# Dense layer after drop5
Dense6 =Dense(1024, activation=act_layer,kernel_initializer=initializer,kernel_regularizer=l2_reg)(drop5)
#batch Normalization
bn6 = BatchNormalization()(Dense6)
# dropout after batch norm 6
drop6 =Dropout(0.5)(bn6)
```

✓ Then FCN layer decreased 512 (40% dropout), 128, 32, 16 and 8 with 20% dropout

```
# Dense layer after drop6
Dense7 =Dense(512, activation=act_layer,kernel_initializer=initializer,kernel_regularizer=l2_reg)(drop6)
#batch Normalization
bn7 = BatchNormalization()(Dense7)
# dropout after batch norm 7
drop7 =Dropout(0.4)(bn7)
```

```
# Dense layer after dropout 10
Dense11 =Dense(8, activation=act_layer,kernel_initializer=initializer,kernel_regularizer=l2_reg)(drop10)
#batch Normalization
bn11 = BatchNormalization()(Dense11)
# dropout after batch norm 11
drop11 =Dropout(0.2)(bn11)
```

✓ Last layer is only dense with 4 hidden neurons

```
# Dense layer after dropout 11
Dense12 =Dense(4, activation=act_layer,kernel_initializer=initializer,kernel_regularizer=l2_reg)(drop11)
```

✓ Output layer is sigmoid(since it is binary classification)

```
#output layer
output =Dense(1, activation='sigmoid',name='output')(Dense12)
```

✓ Adam optimizer is implemented with learning rate 8e-4.

```
#adam optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=8e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-07,name='Adam')
```

✓ Custom Callback function to update accuracy, auc to be calculated on epoch_ends

```
import tensorflow.keras
import sklearn.metrics
from sklearn.metrics import roc_curve, auc,roc_auc_score

class CustomMetrics(tf.keras.callbacks.Callback):
    def __init__(self,validation_data):
        super(CustomMetrics, self).__init__()
        self.X_val = validation_data[0]
        self.y_val = validation_data[1]
        self.auc =[]
    def on_train_begin(self, logs={}):
        self.customMetrics={'val_accuracy':[], 'roc_auc':[],'val_loss':[],'loss':[]}
    def on_epoch_end(self, epoch, logs={}):
        #If you are getting any NaN values(either weigths or loss) while training, you have to terminate your training.
        if logs.get('loss') is None:
            if np.isnan(logs.get('loss')) or np.isinf(logs.get('loss')):
                print("Invalid loss and terminated at epoch {}".format(epoch))
                self.model.stop_training = True
        elif logs.get('loss')<=0.01:
            print('loss has reduced below 001. Hence stop training!!!')
            self.model.stop_training = True
        elif np.round(logs.get('val_accuracy'),2)>=0.98:
            print('validation accuracy reached over 98%. Hence stop training to avoid overfitting!!!')
            self.model.stop_training = True

        self.customMetrics['val_loss'].append(logs.get('val_loss'))
        self.customMetrics['loss'].append(logs.get('loss'))
        self.customMetrics['val_accuracy'].append(logs.get('val_accuracy'))
        try:
            y_pred = np.asarray(self.model.predict(self.X_val))
            y_true = self.y_val
            self.customMetrics['roc_auc'].append(sklearn.metrics.roc_auc_score(y_true, y_pred))
        except ValueError:
            pass

        self.auc.append(roc_auc_score(y_true,y_pred))
        print('-\tauc  : {:.4f}'.format(roc_auc_score(y_true,y_pred)))
        return
metrics=CustomMetrics(validation_data=(X_cv,np.array(y_cv)))
```

✓ Learning rate is changed on every epoch ends with 3% reduction and every 3rd epoch is reduced with 5%.

```
def changeLearningRate(epoch,lr):
    #val_acc = metrics.customMetrics['val_acc']
    changed = lr #default condition
    if epoch <= 1:
        changed = lr
    #Cond2. For every 3rd epoch, decay your learning rate by 4%.
    elif epoch > 1 and epoch%3==0:
        changed = lr*(1-0.050)
        print('reducing learing rate with 5% on every 3rd epoch {} '.format(changed)
    else:
        changed = lr*(1-0.03)
        print('reducing learing rate with 3% ~~> {}'.format(changed))
    return changed
```

✓ Tensor board is activated to log all the metrics and loss function on each epoch ends

```
%load_ext tensorboard
#%reload_ext tensorboard
logs_base_dir = "/logs/fit/NN/"
os.makedirs(logs_base_dir, exist_ok=True)
```

✓ Class weight is added

```
from sklearn.utils import class_weight
#class_weights = class_weight.compute_class_weight('balanced', np.unique(y_train),y_train)
#class_weights = dict(zip(np.unique(y_train), class_weight.compute_class_weight('balanced', np.unique(y_train),y_train)))
#y_integers = np.argmax(y_train, axis=1)
class_weights = class_weight.compute_class_weight(class_weight='balanced', classes=[0,1], y=y_train)
d_class_weights = dict(enumerate(class_weights))
d_class_weights
```

```
{0: 0.5439096274931637, 1: 6.193512203876526}
```

✓ Early stopping is activated based on validation accuracy with 5 patience.

```
#learning rate and EarlyStop
lrschedule = LearningRateScheduler(changeLearningRate, verbose=1)
earlystop = EarlyStopping(monitor='val_accuracy', min_delta=0.001, patience=5, verbose=1)
```

✓ Checkpoint is activated to store best model based on the histories.

```
#saving model
filepath=os.path.join(os.getcwd()+ "/model_save/weights-{epoch:02d}-{val_accuracy:.4f}.hdf5")
checkpoint = ModelCheckpoint(filepath=filepath, monitor='val_accuracy',  verbose=1, save_best_only=True, mode='auto')
```

✓ Model is trained with batch size 1024 and max of 30 epochs.

```
#model fit
model_history=model.fit(X_train,y_train, class_weight=d_class_weights,
                    validation_data=(X_cv,y_cv),epochs=30, batch_size=1024,
                    callbacks=[tensorboard_callback,metrics,lrschedule,earlystop,checkpoint])
```

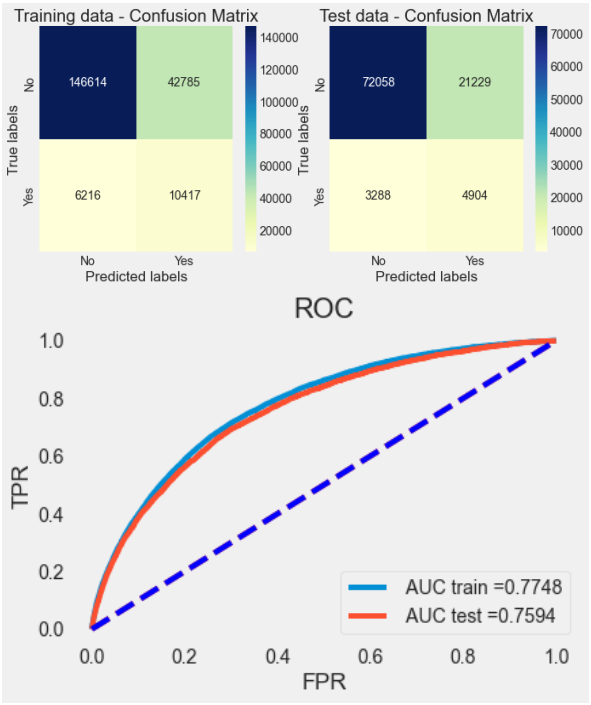
✓ Epoch 6 is yielded better validation accuracy

```
Epoch 00006: val_accuracy improved from 0.71470 to 0.80637, saving model to C:\DS\1. AAIC\001.0ML.Project\26.repayloaan/model_s
ave\weights-06-0.8064.hdf5
135/135 [==============================] - 17s 126ms/step - loss: 0.9509 - accuracy: 0.6596 - val_loss: 0.8309 - val_accuracy:
0.8064 - lr: 6.9363e-04
reducing learing rate with 5% on every 3rd epoch 0.0006589499011170119
```

- ✓ Model stopped early at epoch 11, due to no further improvement.

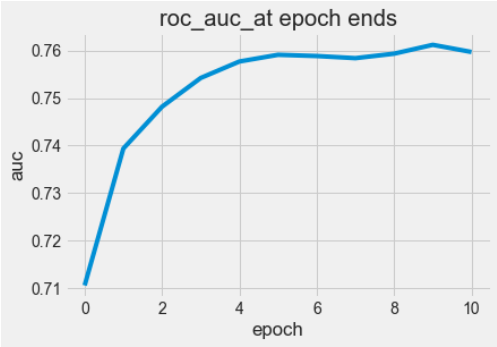
```
Epoch 00011: val_accuracy did not improve from 0.80637
135/135 [==============================] - 17s 123ms/step - loss: 0.6859 - accuracy: 0.6862 - val_loss: 0.6701 - val_accuracy:
0.7580 - lr: 5.7134e-04
Epoch 00011: early stopping
```

- ✓ Confusion matrix and ROC drawn based on model predicted train and test data



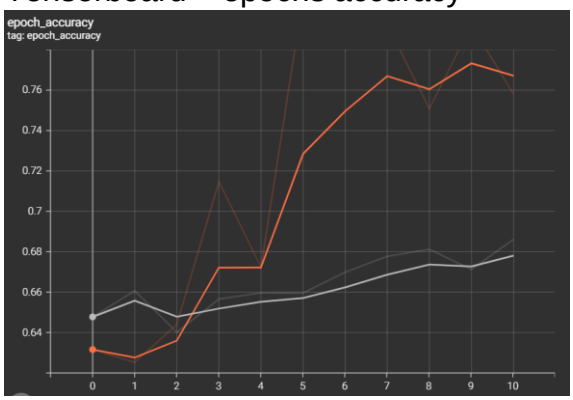
- ✓ Model is did a decent job. However, FN is much higher than the supervised models.
- ✓ Model Train AUC 77% and Test AUC 76% which is good and very similar to LR.
- ✓ Since DL, has more number of hyper parameters, it requires, hyper parameter tuning.
- ✓ Sofar, LightGBM followed with LR are better performance models.

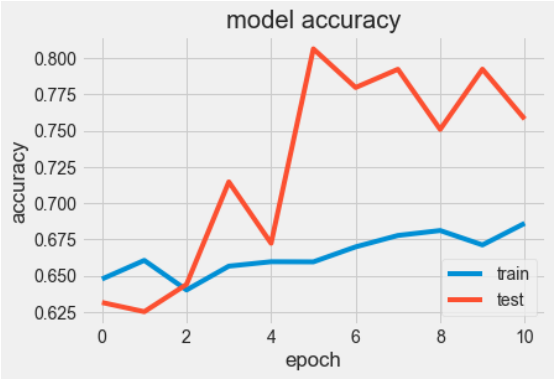
- ✓ ROC_ on each epochs



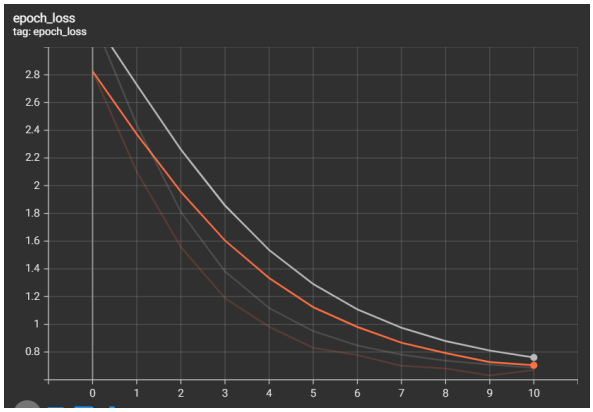
- ✓ ROC didn't improve much and struck at 76%.

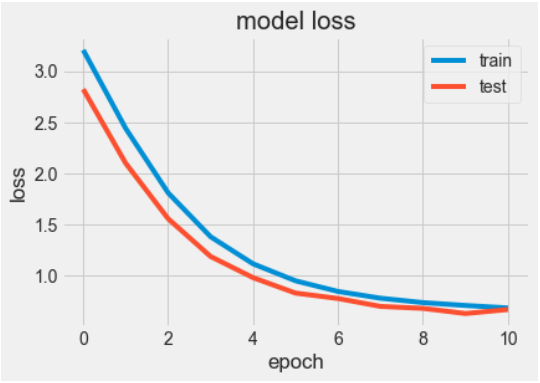
✓ Tensorboard – epochs accuracy





✓ Tensorboard – loss

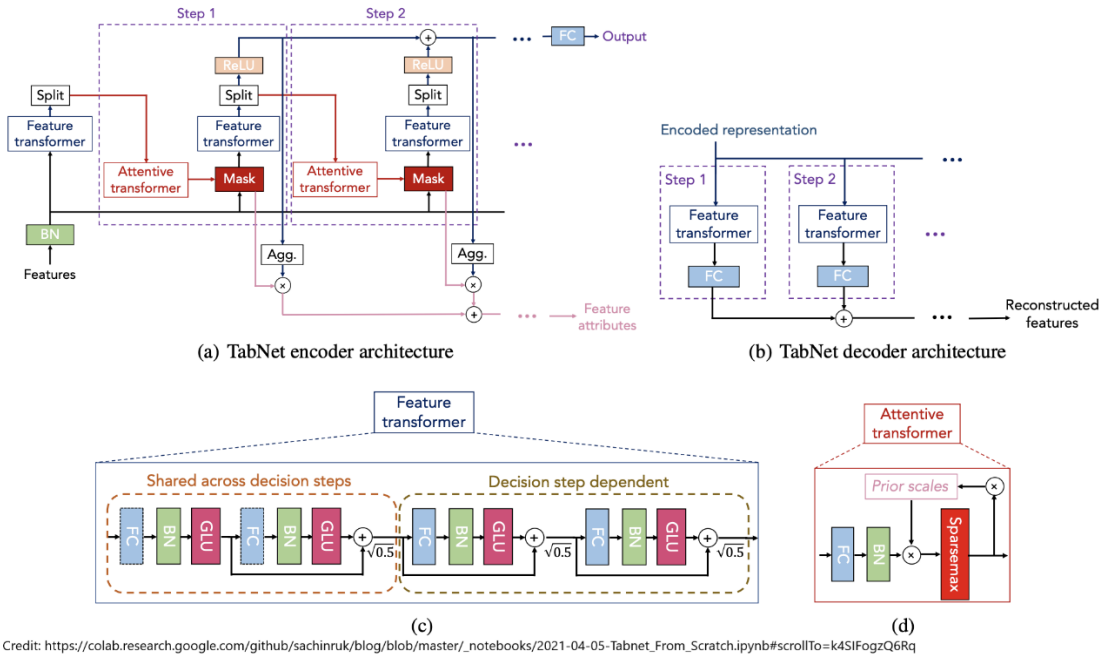




✓ Deep learning training is faster. However, a lot of parameters are very sensitive. Minor changes in parameters are impacts the class labels. FN is slightly higher than LR and LGBM. Feature Importance is not implicit. We have to use LIME or SHAP.

6. TabNET- Pytorch

- ✓ TabNET is an encoder-decoder, tree-based neural network model suited for tabular data. And is trained using gradient descent-based optimization. Since it is at very inception level, it is important to understand the architecture.



(a) TabNet encoder architecture (b) TabNet decoder architecture

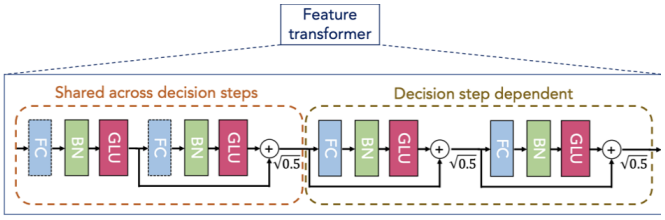
(c) (d)

Credit: https://colab.research.google.com/github/sachinruk/blog/blob/master/_notebooks/2021-04-05-Tabnet_From_Scratch.ipynb#scrollTo=k4SIFogzQ6Rq

- ✓ TabNet data doesn't require to be preprocessed. TabNET uses sequential attention to choose features at each decision step, enabling interpretability and better learning as the learning capacity is used for the most useful features.
- ✓ Feature selection is instance-wise, e.g. it can be different for each row of the training dataset.
- ✓ TabNet employs a single deep learning architecture for feature selection and reasoning.
- ✓ After Batch Normalization, the data feed into Future Transformer. It also has skip connection [like residual network] and passes directly to next layer of Future Transformer through mask.
- ✓ Future Transformer (FT): It contains Fully Connected Block (FCB) as basic unit. This block contains dense layer, Batch Normalization and GLU activation layer. Gated Linear Unit is matrix multiplication of features with sigmoidal value of features.

```
def GLU(x):
    return x * tf.sigmoid(x)

class FCBlock(layers.Layer):
    def __init__(self, units):
        super().__init__()
        self.layer = layers.Dense(units)
        self.bn = layers.BatchNormalization()

    def call(self, x):
        return GLU(self.bn(self.layer(x)))
```

✓ FT contains shared and decision blocks.



(c)

✓ Shared block is a stack of FCB in sequential order. Between FCB there is a skip connection like residual connection and multiplied with sqrt(0.5).

✓ Decision block also contains FCB and is similar to shared block except residual connection in each block and acts as individual.

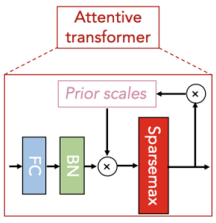
Shared Block

```
class SharedBlock(layers.Layer):
    def __init__(self, units, mult=tf.sqrt(0.5)):
        super().__init__()
        self.layer1 = FCBlock(units)
        self.layer2 = FCBlock(units)
        self.mult = mult

    def call(self, x):
        out1 = self.layer1(x)
        out2 = self.layer2(out1)
        return out2 + self.mult * out1
```

Decision Block

```
class DecisionBlock(SharedBlock):
    def __init__(self, units, mult=tf.sqrt(0.5)):
        super().__init__(units, mult)

    def call(self, x):
        out1 = x * self.mult + self.layer1(x)
        out2 = out1 * self.mult + self.layer2(out1)
        return out2
```

✓ Attentive Transformer (AT): The attentive transformer decides which bits of the input features (x) it needs to pay attention to (mask) at each step.



(d)

✓ AT contains splits and prior layers. The split module simply splits the output of the feature transformer into two portions. One portion feeds into the attentive transformer, and the other goes into the output of our overall network. The portions (n_d, n_a) are hyper-parameters that the user needs to specify and it would sum to the number of output nodes of the decision layer. Typically n_d and n_a are 8-64.

- ✓ The attentive layer takes the n_a output nodes from the decision block, runs it through a dense layer and batch norm layer before passing through a sparsemax layer.
- ✓ Sparsemax is similar to softmax in that the output sums to one. However, it drives some of the smaller outputs to exactly zero.
- ✓ The prior layer is used as a tool to suppress some of the inputs that were used before in a previous step. Similar to bootstrap sampling (column sampling/row sampling).
- ✓ In the first step none of the input had been used before, and therefore the output of the attentive transformer is unaffected by the prior. However, in the second step (and onwards), the prior is updated to be prior = previous_prior * (constant - previous_mask).

```python
class Prior(layers.Layer):
    def __init__(self, gamma=1.1):
        super().__init__()
        self.gamma = gamma

    def reset(self):
        self.P = 1.0

    def call(self, mask):
        self.P = self.P * (self.gamma - mask)
        return self.P
```

```python
class AttentiveTransformer(layers.Layer):
    def __init__(self, units):
        super().__init__()
        self.layer = layers.Dense(units)
        self.bn = layers.BatchNormalization()

    def call(self, x, prior):
        return sparsemax(prior * self.bn(self.layer(x)))
```

- ✓ Loss Function: In addition to output loss function CrossEntropy, additional loss function on the mask values drive the values to 0 or 1.

$$L_{mask} = -M \log(M + \epsilon)$$

- ✓ The second part of encoder architecture is the actual output of the model. The n_d number of inputs that do not go through the attentive layer gets passed through a ReLU activation at each step before being added up for the final output. This is compared against a target by using a task specific loss function

```python
# collapse
class TabNet(keras.Model):
    def __init__(self, input_dim, output_dim, steps, n_d, n_a, gamma=1.3):
        super().__init__()
        # hyper-parameters
        self.n_d, self.n_a, self.steps = n_d, n_a, steps
        # input-normalisation
        self.bn = layers.BatchNormalization()
        # Feature Transformer
        self.shared = SharedBlock(n_d+n_a)
        self.first_block = DecisionBlock(n_d+n_a)
        self.decision_blocks = [DecisionBlock(n_d+n_a)] * steps
        # Attentive Transformer
        self.attention = [AttentiveTransformer(input_dim)] * steps
        self.prior_scale = Prior(gamma)
        # final layer
        self.final = layers.Dense(output_dim)

        self.eps = 1e-8
        self.add_layer = layers.Add()
```

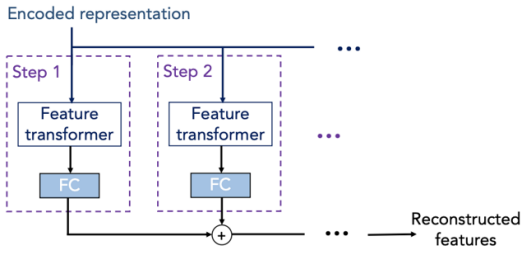
```python
@tf.function
def call(self, x):
    self.prior_scale.reset()
    final_outs = []
    mask_losses = []

    x = self.bn(x)
    attention = self.first_block(self.shared(x))[:,:self.n_a]
    for i in range(self.steps):
        mask = self.attention[i](attention, self.prior_scale.P)
        entropy = mask * tf.math.log(mask + self.eps)
        mask_losses.append(
            -tf.reduce_sum(entropy, axis=-1) / self.steps
        )

        prior = self.prior_scale(mask)
        out = self.decision_blocks[i](self.shared(x * prior))
        attention, output = out[:,:self.n_a], out[:,self.n_a:]
        final_outs.append(tf.nn.relu(output))

    final_out = self.add_layer(final_outs)
    mask_loss = self.add_layer(mask_losses)

    return self.final(final_out), mask_loss
```

```python
def mask_importance(self, x):
    self.prior_scale.reset()
    feature_importance = 0

    x = self.bn(x)
    attention = self.first_block(self.shared(x))[:,:self.n_a]
    for i in range(self.steps):
        mask = self.attention[i](attention, self.prior_scale.P)

        prior = self.prior_scale(mask)
        out = self.decision_blocks[i](self.shared(x * prior))
        attention, output = out[:,:self.n_a], out[:,self.n_a:]
        step_importance = tf.reduce_sum(tf.nn.relu(output), axis=1, keepdims=True)
        feature_importance += mask * step_importance

    return feature_importance
```

```python
# collapse
from keras.losses import SparseCategoricalCrossentropy

sce = SparseCategoricalCrossentropy(from_logits=True)
reg_sparse = 0.01
def full_loss(y_true, y_pred):
    logits, mask_loss = y_pred
    return sce(y_true, logits) + reg_sparse * mask_loss.mean()
```

```python
def mask_loss(y_true, mask_losses):
    return tf.reduce_mean(mask_losses)
```

- ✓ The TabNet decoder architecture consists of a feature transformer, followed by the fully connected layers at the decision step to reconstruct tabular features from the TabNet encoded representations. The output is then summed to the reconstructed features.



(b) TabNet decoder architecture

- ✓ TabNet has unsupervised pre-training part which significantly improves performance for fast adaptation on the supervised classification task.
- ✓ TabNet implementation – preprocessed data loaded

```
#df_train = pd.read_csv("./data/pre_processed_df_train.csv",sep='\t', encoding='utf-8', nrows=75000)
df_train = pd.read_csv("./data/pre_processed_df_train.csv",sep='\t', encoding='utf-8')
```

- ✓ Data is split into Train, valid and Test data

Train, Validation and Test Split

```
#split categorical data only train and test.
X_tr, X_test, y_tr, y_test = train_test_split(df_train, y, test_size=0.33, stratify=y,random_state=42)
X_train,X_val,y_train,y_val =train_test_split(X_tr,y_tr,test_size=0.33,stratify=y_tr,random_state=42)
```

- ✓ The only part needs to be transformed in the preprocessed data is categorical. In order to embed the categorical data, categorical dimensions of the feature to be extracted.

```
# index of the categorical features
cat_indexes = [column_index for column_index, column_name in enumerate(df_train.columns) if column_name in categorical_features]
# dimensions of each categorical feature
cat_dimensions = df_train[categorical_features].nunique().tolist()
```

- ✓ Categorical embedding dimensions to be extracted. This can be set to 1. However, if we set the max embedding dimensions, the model converges faster.

```
# index of the categorical features
cat_indexes = [column_index for column_index, column_name in enumerate(df_train.columns) if column_name in categorical_features]
# dimensions of each categorical feature
cat_dimensions = df_train[categorical_features].nunique().tolist()
```

✓ Vectorization of categorical data: Only categorical data is Label-encoded. Rest of the numerical data, not vectorized and used as is from the preprocessed data.

vectorization - Categorical Features

```
from sklearn.preprocessing import LabelEncoder
# Numeric columns will be scaled by StandardScaler
#scaler = StandardScaler()
# Categorical will be transformed using Weight of Evidence approach
l_enc = LabelEncoder()

types = df_train.dtypes
categorical_columns = {}
categorical_dims =  {}
for idx,col in enumerate(tqdm(df_train.columns)):
    if types[col] == 'object': #categorical values
        l_enc = LabelEncoder()
        X_train[col] = l_enc.fit_transform(X_train[col].values)
        X_val[col] = l_enc.fit_transform(X_val[col].values)
        X_test[col] = l_enc.fit_transform(X_test[col].values)
```

```
100%|██████████| 999/999 [00:02<00:00, 368.34it/s]
```

✓ TabNet pytorch code implementation is used. Hyper parameters of TabNet

```
tabnet_params = {
                    'n_d': 32, 'n_a': 32, 'n_steps': 5,'gamma':1.4,
                    'cat_idxs':cat_indexes,
                    'cat_dims':cat_dimensions,
                    'cat_emb_dim': cat_embedding_dim,
                    'n_independent': 2,
                    'n_shared': 2, 'epsilon': 1e-15,
                    'momentum': 0.05, 'verbose': 1,
                    'optimizer_fn': torch.optim.Adam,
                    'optimizer_params': dict(lr=1.9e-2),
                    'scheduler_fn': torch.optim.lr_scheduler.StepLR,
                    'scheduler_params': {"step_size":20,'gamma':0.95},
                    'mask_type': 'entmax', #sparsemax
                    'input_dim': X_train.shape[1],
                    'output_dim': 2,
                    'device_name': 'auto',
                    'lambda_sparse': 1.2e-4,
                    'clip_value' : 2.0
                }
```

✓ Pretrained unsupervised model is trained with train data without class labels.

```
unsupervised_model = TabNetPretrainer(**tabnet_params )
unsupervised_model.fit(X_train=np.array(X_tr),eval_set=[np.array(X_cv)]
                       ,pretraining_ratio=0.8 ,max_epochs=max_epochs , patience=3
                       ,batch_size=1024, virtual_batch_size=256,num_workers=0
                       ,drop_last=False)

#save unsupervised model
unsupervised_model.save_model('./results/test_pretrain')
```

```
Device used : cpu
epoch 0  | loss: 1179224260372410.5| val_0_unsup_loss: 8514.125|  0:01:22s
epoch 1  | loss: 1153366112171267.0| val_0_unsup_loss: 584.07562|  0:02:44s
epoch 2  | loss: 1175375222225109.8| val_0_unsup_loss: 387.67023|  0:04:07s
epoch 3  | loss: 2312384226626707.0| val_0_unsup_loss: 1833.19604|  0:05:32s
epoch 4  | loss: 3449872097100011.5| val_0_unsup_loss: 7389.73291|  0:06:56s
epoch 5  | loss: 24471.51838| val_0_unsup_loss: 2303.71973|  0:08:20s

Early stopping occurred at epoch 5 with best_epoch = 2 and best_val_0_unsup_loss = 387.67023
Best weights from best epoch are automatically used!
Successfully saved model at ./results/test_pretrain.zip

'./results/test_pretrain.zip'
```

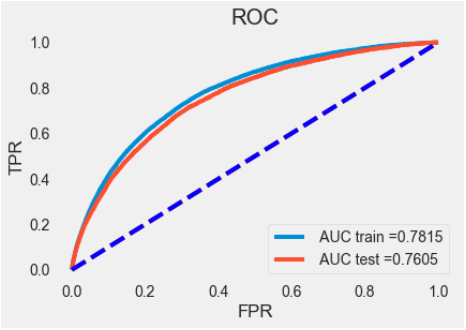
- ✓ The same parameters used in unsupervised, is used in TabNetClassification task. Weights are loaded from the pretrained unsupervised task. This converges faster

```
clf = TabNetClassifier(**tabnet_params)
model_history=clf.fit(X_train=np.array(X_train), y_train=np.array(y_train),
                      eval_set=[(np.array(X_train), np.array(y_train)), (np.array(X_val), np.array(y_val))],
                      eval_name=['train', 'valid'], eval_metric=['auc'],
                      from_unsupervised=loaded_pretrain,
                      max_epochs=max_epochs , patience=3,
                      batch_size=1024, virtual_batch_size=256,
                      num_workers=0,
                      drop_last=False,
                      #weights=d_class_weights
                      weights=1 # No sampling
                     )
```

- ✓ Epochs results. As we see the results below, the Model converges faster.

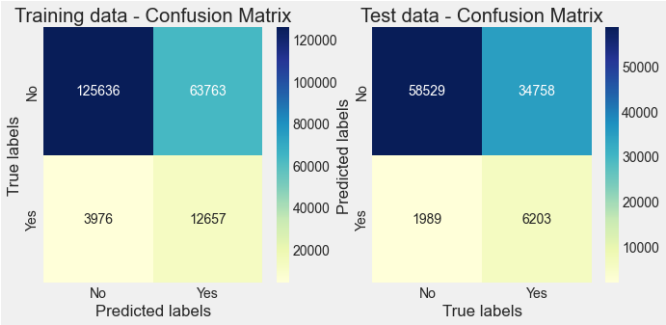
```
Device used : cpu
Loading weights from unsupervised pretraining
epoch 0  | loss: 0.64268 | train_auc: 0.70845 | valid_auc: 0.70346 |  0:03:59s
epoch 1  | loss: 0.61918 | train_auc: 0.72309 | valid_auc: 0.71882 |  0:07:58s
epoch 2  | loss: 0.61387 | train_auc: 0.72855 | valid_auc: 0.72288 |  0:11:56s
epoch 3  | loss: 0.60551 | train_auc: 0.74198 | valid_auc: 0.73355 |  0:15:54s
epoch 4  | loss: 0.59123 | train_auc: 0.75004 | valid_auc: 0.7345  |  0:19:52s
epoch 5  | loss: 0.59724 | train_auc: 0.74146 | valid_auc: 0.73019 |  0:23:50s
epoch 6  | loss: 0.59386 | train_auc: 0.75883 | valid_auc: 0.74616 |  0:27:48s
epoch 7  | loss: 0.58146 | train_auc: 0.75913 | valid_auc: 0.74696 |  0:31:47s
epoch 8  | loss: 0.588   | train_auc: 0.75374 | valid_auc: 0.74386 |  0:35:44s
epoch 9  | loss: 0.58414 | train_auc: 0.76881 | valid_auc: 0.75412 |  0:39:41s
epoch 10 | loss: 0.57946 | train_auc: 0.76982 | valid_auc: 0.75728 |  0:43:38s
epoch 11 | loss: 0.57601 | train_auc: 0.76962 | valid_auc: 0.75868 |  0:47:36s
epoch 12 | loss: 0.57188 | train_auc: 0.78107 | valid_auc: 0.76289 |  0:51:33s
epoch 13 | loss: 0.55921 | train_auc: 0.79038 | valid_auc: 0.76372 |  0:55:30s
epoch 14 | loss: 0.54603 | train_auc: 0.8014  | valid_auc: 0.76014 |  0:59:26s
epoch 15 | loss: 0.5326  | train_auc: 0.81706 | valid_auc: 0.75016 |  1:03:24s
epoch 16 | loss: 0.5138  | train_auc: 0.82974 | valid_auc: 0.74338 |  1:07:26s

Early stopping occurred at epoch 16 with best_epoch = 13 and best_valid_auc = 0.76372
Best weights from best epoch are automatically used!
```

- ✓ ROC Curve of Train [Train +validation data] and Test data



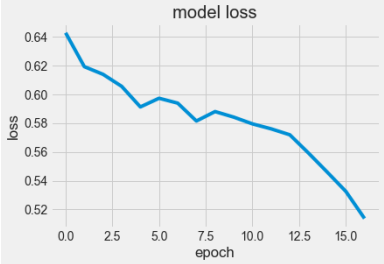
- ✓ Model is not overfitted and AUC of Train 78% and AUC of Test is 76%.
- ✓ Compare to LightGBM, the score is less only.
- ✓ May require additional parameter tuning.

- ✓ Confusion matrix of Train and Test data



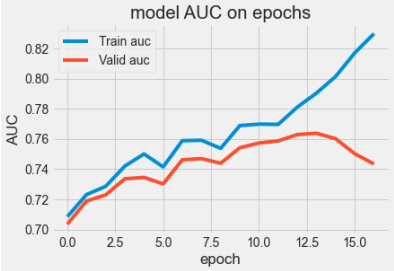
- ✓ The Model is not biased to any particular class. And also FN part is less compared to FP. However, FP part is higher compare to LGBM Model.

✓ Model Loss at each epoch:



✓ As we can observe, the first epoch loss itself is less than 1.

✓ Model AUC at each epoch:



✓ AUC has improved till 13th epoch. After which it starts deteriorating and train AUC increases. Hence the training stopped to avoid overfitting.

✓ Classification Report:

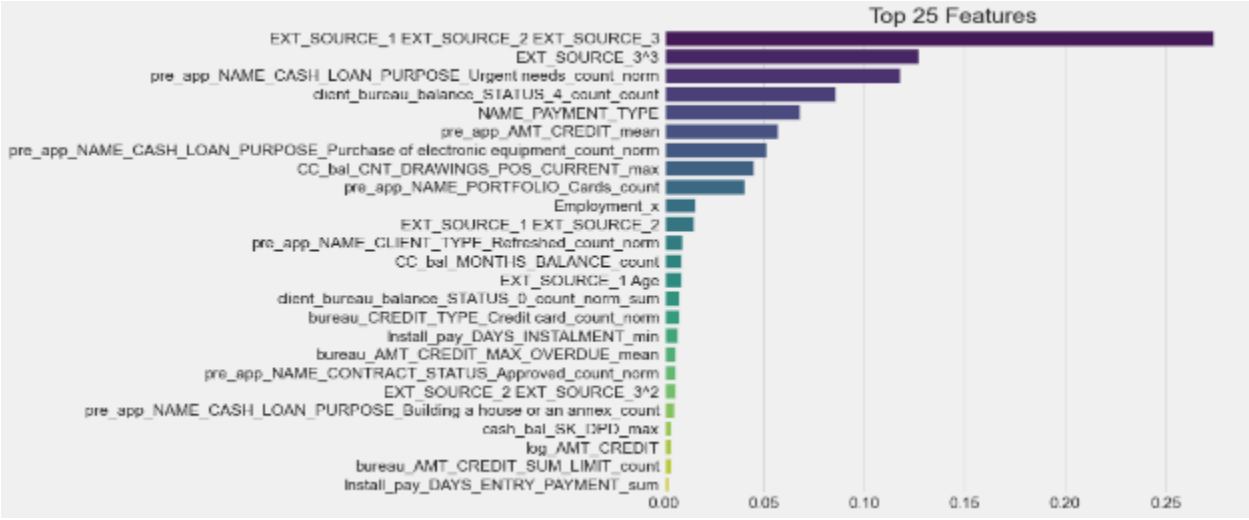
Train Results

	precision	recall	f1-score	support
0	0.97	0.66	0.79	189399
1	0.17	0.76	0.27	16633
accuracy			0.67	206032
macro avg	0.57	0.71	0.53	206032
weighted avg	0.90	0.67	0.75	206032

Test Results

	precision	recall	f1-score	support
0	0.97	0.63	0.76	93287
1	0.15	0.76	0.25	8192
accuracy			0.64	101479
macro avg	0.56	0.69	0.51	101479
weighted avg	0.90	0.64	0.72	101479

✓ Feature Importance extracted from the Model:



Feature	Score
EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3	0.2739
EXT_SOURCE_3^3	0.127
pre_app_NAME_CASH_LOAN_PURPOSE_Urgent needs_count_norm	0.1175
client_bureau_balance_STATUS_4_count_count	0.0853
NAME_PAYMENT_TYPE	0.0676
pre_app_AMT_CREDIT_mean	0.0569
pre_app_NAME_CASH_LOAN_PURPOSE_Purchase of electronic equipment_count_norm	0.0511
CC_bal_CNT_DRAWINGS_POS_CURRENT_max	0.0447
pre_app_NAME_PORTFOLIO_Cards_count	0.0401
Employment_x	0.0156
EXT_SOURCE_1 EXT_SOURCE_2	0.0149
pre_app_NAME_CLIENT_TYPE_Refreshed_count_norm	0.009
CC_bal_MONTHS_BALANCE_count	0.0085
EXT_SOURCE_1 Age	0.0084
client_bureau_balance_STATUS_0_count_norm_sum	0.0074
bureau_CREDIT_TYPE_Credit card_count_norm	0.0072
Install_pay_DAYS_INSTALMENT_min	0.0067
bureau_AMT_CREDIT_MAX_OVERDUE_mean	0.006
pre_app_NAME_CONTRACT_STATUS_Approved_count_norm	0.0059
EXT_SOURCE_2 EXT_SOURCE_3^2	0.0057
pre_app_NAME_CASH_LOAN_PURPOSE_Building a house or an annex_count	0.0054
log_AMT_CREDIT	0.0034
cash_bal_SK_DPD_max	0.0034
bureau_AMT_CREDIT_SUM_LIMIT_count	0.0034
Install_pay_DAYS_ENTRY_PAYMENT_sum	0.0026

- ✓ All three External Sources features are the topmost priority.
- ✓ Polynomial feature transformation, statistical-based feature transformation all played an important role.
- ✓ Additional datasets are also played an important role.

10. LIME and SHAP Explainability

- ✓ Explaining predictions to non-technical folks, LIME and SHAP can help.
- ✓ LIME: Model agnosticism refers to the property of LIME using which it can give explanations for any given supervised learning model by treating it as a ‘black-box’ separately.
- ✓ Local explanations mean that LIME gives explanations that are locally faithful within the surroundings or vicinity of the observation/sample being explained.
- ✓ Sampling and obtaining a surrogate dataset: LIME provides locally faithful explanations around the vicinity of the instance being explained. By default, it produces 5000 samples(see the num_samples variable) of the feature vector following the normal distribution. Then it obtains the target variable for these 5000 samples using the prediction model, whose decisions it’s trying to explain.
- ✓ Feature Selection from the surrogate dataset: After obtaining the surrogate dataset, it weighs each row according to how close they are to the original sample/observation. Then it uses a feature selection technique like Lasso to obtain the top important features.
- ✓ LIME also employs a Ridge Regression model on the samples using only the obtained features. The outputted prediction should theoretically be similar in magnitude to the one outputted by the original prediction model. This is done to stress the relevance and importance of these obtained features.
- ✓ Since this dataset is a tabular dataset, LIME’s API offers the LimeTabularExplainer to extract features. And, LightGBM model is outperformed in all supervised models. Hence, will consider LightGBM model.

✓ Load the trained LightGBM model.

```
: #load LGBM trained model
filename='./results/model_lgb.pkl'
model = joblib.load(open(filename, 'rb'))
```

✓ Load the preprocessed vectorized dataset and split the train and test data.

```
with open('./data/train_vector.pkl', 'rb') as f:
    X = pickle.load(f)

with open('./data/feature_names.pkl', 'rb') as f:
    feature_names = pickle.load(f)

with open('./data/yvalues.pkl', 'rb') as f:
    y = pickle.load(f)
```

```
#split data only train and test.
X_train, X_test, y_train, y_test = train_test_split(df_train, y, test_size=0.33, stratify=y,random_state=42)
```

✓ Load the LimeTabularExplainer

```
#Credit:https://www.analyticsvidhya.com/blog/2020/10/unveiling-the-black-box-model-using-explainable-ai-lime-shap-industry-use-cc
#credit:https://www.kdnuggets.com/2019/12/interpretability-part-3-lime-shap.html
class_names = [0, 1]
#instantiate the explanations for the data set
limeexplainer = LimeTabularExplainer(X_test.values,
                                     class_names=class_names,
                                     feature_names = X_test.columns,
                                     feature_selection="lasso_path",
                                     discretize_continuous = True,
                                     #discretizer="entropy",
                                    )
idx=0 # the rows of the dataset
explainable_exp = limeexplainer.explain_instance(X_test.values[idx], model.predict_proba, num_features=8, labels=class_names)
explainable_exp.show_in_notebook(show_table=True, show_all=False)
```

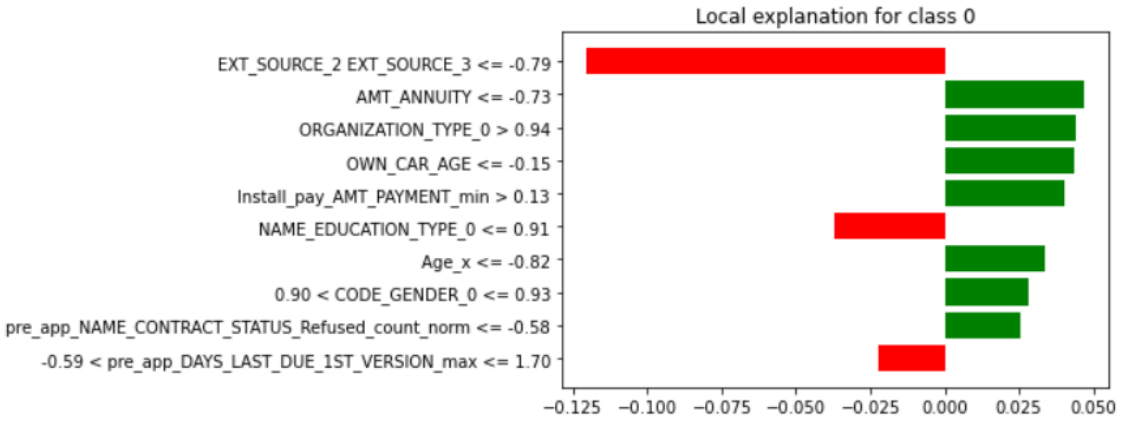
✓ In this, we are asking LIME to explain the decisions behind the 0th test data prediction by explaining the top 8 features which contributed to the said model's prediction.

✓ LimeTabularExplainer takes as input train data and generated explainer object which can then be used to explain the individual prediction.

- ✓ From the above, prediction probabilities that LightGBM model is 83% confident on the class label 0 [customers would repay], and only 17% confident on the class label 1 [customers default] for given test data sample.
- ✓ Also, for that given test data, Name_Education_Type and Organization_Type are contributes to class 0 and have higher weights for class 0. And install_pay, External Source 1 and 3, Code Gender, and Family status are contributing towards Class 1 and having more weights for class 1. AMT_Annuity and OWN_CAR have less weight to contribute for class 1.
- ✓ Submodular Pick explains model globally by combining local explanations

```
from lime import submodular_pick
sp_obj = submodular_pick.SubmodularPick(limeexplainer, X_test.values, model.predict_proba,
                                        sample_size=500, num_features=10, num_exps_desired=5)
```

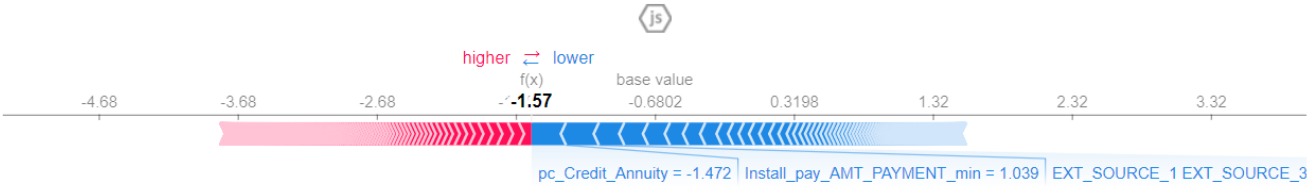
- ✓ Submodular pick is trained with 500 samples and top 10 features to be picked.



- ✓ External Features 2 and 3 weight is greater than 0.73 contributes to class 0[repay]. Lesser weight <= -0.79, contributes class 1 [Default].
- ✓ Similarly, AMT_Annuity weight > 0.51, contributes to class 1 and less weight contributes to class 0.
- ✓ Name_Education_type_0 is <= 0.91, contributes to class 1.

- ✓ Likewise, previous application past due weight more than -0.59 and less than or equal to 1.7 contributes to class 1[default]. Meaning, if the customer defaulted, having more past due balance/past due days would be higher.
- ✓ Hence, LIME explainability is easier even for non-technical people.
- ✓ SHAP: It is a game-theoretic approach to explain the output of any machine learning model. A prediction can be explained by assuming that each feature value of the instance is a “player” in a game where the prediction is the payout. Shapley values – a method from coalitional game theory – tells us how to fairly distribute the “payout” among the features.
- ✓ With SHAP, we can generate explanations for a single prediction. The SHAP plot shows features that contribute to pushing the output from the base value (average model output) to the actual predicted value.

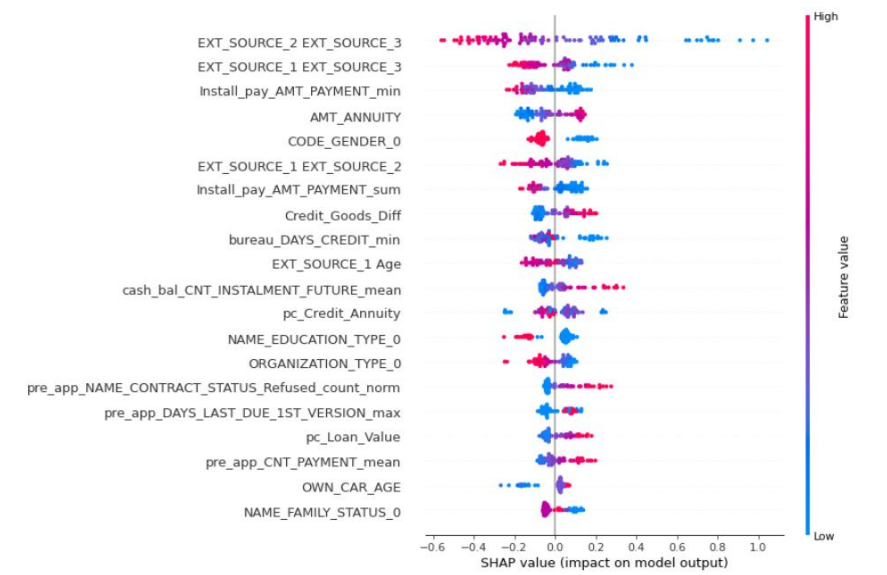
```
shap.initjs() # run to show the plot
shap.force_plot(explainer.expected_value, shap_values=shap_values[0,:], features=feature_display.iloc[0,:])
```



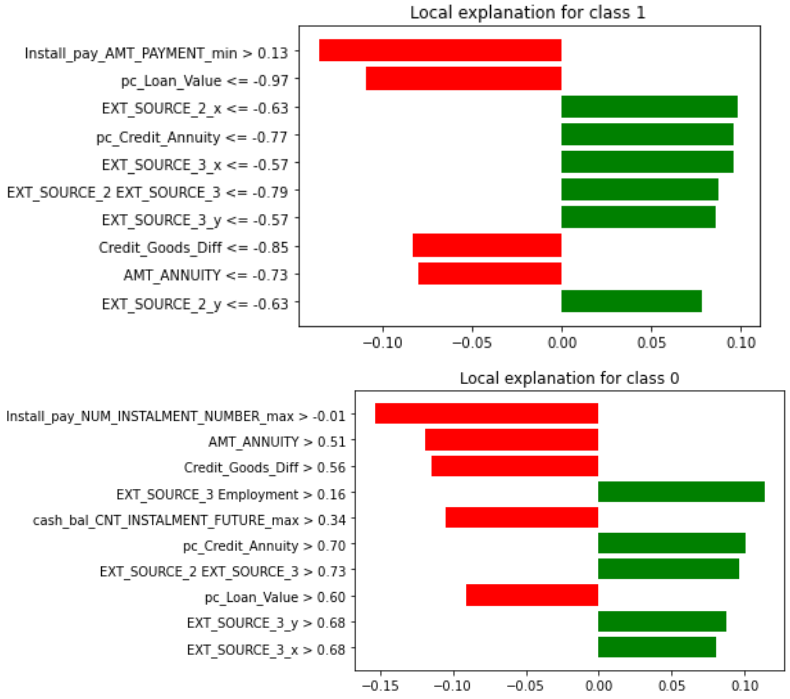
- ✓ Red color indicates features that are pushing the prediction higher, and blue color indicates just the opposite.
- ✓ From the above plot, FLAG_WORK_PHONE, INSTALL_PAY_AMT_PAYMENT (less weight), BUREAU_DAYS_CREDIT_ENDDATE, AGE, ORGANIZATION_TYPE are pushes the loan to repay. On the other hand pc_Credit_Annuity weight <-1.472, Install_pay_AMT_Payment_min = 1.039 pushes the default.

- ✓ Explaining the entire dataset: By plotting summary charts we can visualize the importance of the features and their impact on the prediction. The one below sorts features by the sum of SHAP value magnitudes over all samples. It also uses SHAP values to show the distribution of the impacts each feature has. The color represents the feature value — red indicating high and blue indicating low.

```
shap.summary_plot(shap_values, features)
```



- ✓ Even for Neural Networks can be interpreted with LIME and Shap.
- ✓ From LIME SubmodularPick



11. Summary

Model AUC Scores

Model	Train AUC Score %	Test AUC Score %
Logistic Regression	77.39	76.62
Random Forest	76.33	74.18
XG Boost	85.86	78.53
LightGBM	83.38	78.77
Deep Learning – Neural Network	77.48	75.94
Deep Learning – TabNet (pytorch implementation)	78.15	76.05

- ✓ Considering AUC and confusion matrix results, LightGBM performed well. The worst performance is XGBoost. Though XGBoost AUC values are good, class label prediction is the worst. This could be a model configuration error.
- ✓ Logistic Regression, is robust in high dimensional data and latency is very good.
- ✓ Next to LR, Neural Network models are faster compared to tree-based models.
- ✓ AUC is a good metric and provides model performance and overfitting or not. However, a confusion matrix is also important to analyze the model performance in terms of class prediction.
- ✓ TabNET is model-agnostic, inbuilt feature importance deep neural network. Has encoder-decoder, mask attention, skip connection, pretrained models. Still, it requires more understanding and tuning.
- ✓ Above all models, the performance doesn't improve much beyond 79%. This could be,
    - ✓ Numerical features skewed data, needs to be transformed to a logarithmic value
    - ✓ Outliers are removed manually, but outliers still exist and need to employ robust models would identify outliers smartly.
    - ✓ Furthermore domain based features to be extracted.
    - ✓ Hyper parameters tuning to be applied on deep learning models to improve the performance.