Mir Mustafa Ali 1002117402

Shruthaja Patali Rao 1002118604

Ravindra Marada 1002090718

Durga Priya Challgundla 1002105815

Vidvanth Valaboju 1002069294

Supervised by: Prof. Ramakrishna Prasad Koganti

CSE -5301

Credit Card Default Prediction

**Introduction:**

As we all know, banks offer us lines of credit which can be used to sustain ourselves during difficult times or even invest and develop any ideas we may have. However, doing so means that banks have taken up a liability. To that end our project wants to explore solutions that can help banks make future decisions on credit based on their prior repayment history and trends and reliably predict who is likely to default. So, the bank may be able to prevent the loss by providing the customer with alternative options.

**Background:**

***Preprocessing*:**

* Identified The Categorical Features Of The Dataset. Merged Train\_labels Dataset With Train\_data Dataset For Target Label.
* Convert S\_2 Variable Dtype From Object To Datetime.
* Carried Out Aggregations On The Column : ['mean', 'std', 'min', 'max', 'last’] For Both The Training And Testing Data.
* We Also Set The Null/na Values To 0.

***Exploratory data analysis :***

1. On carrying out EDA on the training data we saw that about 25% of customers in the training data have defaulted. This proportion is consistent across each day in the training set, with a weekly seasonal trend in the day of the month when customers receive their statements.
2. We also got an insight into the missing values in the dataset:
   * The maximum missing value in an row is 102 and the lowest is 9 missing values.
   * All the rows have at least 9 missing values.
   * D88 feature has maximum number of missing values with a total of 5527586 missing values.
   * 68 features have no missing values whereas 122 features have at least 1 missing value.

A screenshot of a computer

Description automatically generated with medium confidenceChart, bar chart

Description automatically generated

***Correlations between variables:***

There are several strong correlations with the target variable. Payment 2 is the most negatively correlated with the probability of defaulting with a correlation of -0.67, while Delinquency 48 is the most positively correlated overall at 0.61.

Chart

Description automatically generated

***Feature* *Importance* :**

Among the top 50 features, Payment 2 has the highest average importance, which is also the feature that is the most negatively correlated with the target variable.

Chart

Description automatically generated

***XgBoost Regression Modelling:***

XGBoost (Extreme Gradient Boosting) is a machine learning algorithm that is widely used for both regression and classification problems. It is an ensemble learning method that combines multiple decision trees to make more accurate predictions. XGBoost is known for its speed, accuracy, and ability to handle large datasets.

In XGBoost regression, the goal is to predict a continuous numerical value (the target variable) based on a set of input features. The algorithm builds a series of decision trees, where each subsequent tree attempts to correct the errors of the previous tree. The final prediction is the sum of the predictions from all the trees.

One of the key features of XGBoost is its use of gradient boosting, which is a technique for iteratively improving the model's performance by optimizing a loss function. In XGBoost, the loss function is typically defined as the mean squared error between the predicted and actual target values.

The XGBoost algorithm uses several important formulas to determine the importance of each feature and to calculate the predicted values of the target variable. These formulas are based on the principles of gradient boosting and are used to optimize the model's performance.

Here are some of the important formulas used in XGBoost regression:

1. Loss Function: The loss function in XGBoost is typically defined as the mean squared error (MSE) between the predicted and actual target values:

**`L(y, y\_hat) = (y - y\_hat)^2`**

where `y` is the actual target value, and `y\_hat` is the predicted target value.

2. Gradient: The gradient of the loss function with respect to the predicted target value `y\_hat` is:

**`g\_i = 2(y\_i - y\_hat\_i)`**

where `y\_i` is the actual target value for observation `i`, and `y\_hat\_i` is the predicted target value for observation `i`.

3. Hessian: The Hessian of the loss function with respect to the predicted target value `y\_hat` is:

**`h\_i = 2`**

4. Prediction: The predicted target value for a new observation is calculated as the sum of the predictions from all the trees:

**`y\_hat = sum(f\_j(x))`**

where `f\_j(x)` is the predicted value of the j-th tree for the new observation `x`. These formulas are based on the original XGBoost paper by Chen and Guestrin (2016) and subsequent works, such as the XGBoost documentation (Chen and Guestrin, 2016) and the XGBoost GitHub repository (Chen et al., 2021).

***Tools and libraries:***

We used numpys, pandas , matplotlib and seaborn.

1. NumPy: A Python library for numerical computing that provides support for multi-dimensional arrays, mathematical operations, and linear algebra functions.
2. Pandas: A Python library for data manipulation and analysis that provides easy-to-use data structures and tools for reading, writing, and analyzing tabular data.
3. Matplotlib: A Python library for creating visualizations such as plots, histograms, and scatterplots.
4. Seaborn: A Python library for creating statistical visualizations that provides a higher-level interface to Matplotlib and includes additional plot types and features.

***Model development and results:***

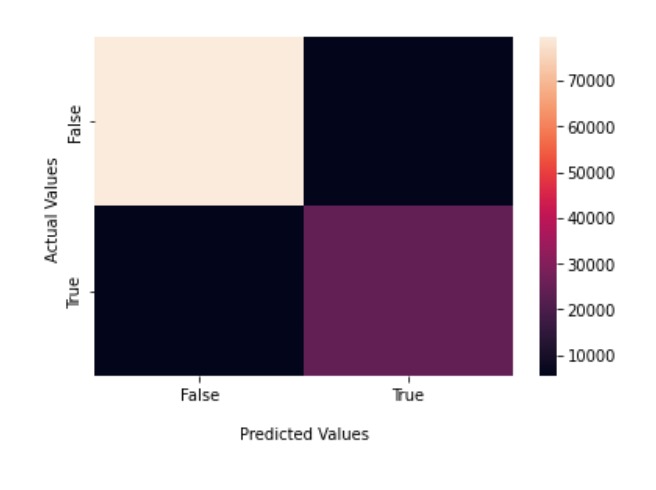
We import several libraries including NumPy, Pandas, Matplotlib, Seaborn, and Scikit-learn. It also imports several GPU libraries like CuPy and CuDF. It then defines two functions, read\_train\_file and process\_and\_feature\_engineer, which read in and process training data.

The first function read\_train\_file loads the train data from a given path, using the cudf.read\_parquet() function to read in a parquet format file. If the usecols argument is given, it uses only the specified columns. It then performs some data cleaning and feature engineering, such as converting the customer\_ID column to a 64-bit integer and filling in any missing values.

The second function process\_and\_feature\_engineer performs further feature engineering on the data by grouping and aggregating columns by customer\_ID, using the groupby() and agg() functions from CuDF. The resulting data is concatenated and returned.

The code then loads the train data using read\_train\_file, performs feature engineering using process\_and\_feature\_engineer, adds target values to the data, and sorts it by index to make it deterministic for cross-validation.

Post this we pass the test data and have obtained the target values with the following accuracy and obtained the below confusion matrix

****

***accuracy*** = 90.448%

***Conclusion* :**

From our analysis we can see the model is able to predict with a confidence of 90.448% if the customer will default. Banks can use this in their verification processes before giving any credit. Should the customer be at risk they can suggest alternative plans such as forbearance agreements and long-term repayment plans. Depending on the bank and its rules they can even have the customer pay some lumpsums on any loans or credit bills to balance out the delinquency variables.

***Recommendation:***

During our process we observed that care must be taken to handle null values. Identifying the important features, along with the XGBoost algorithm increases the accuracy of the prediction.

***Appendix*:**

The python code that we have executed:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sn

import optuna

from sklearn.model\_selection import train\_test\_split

import sklearn.metrics

from xgboost import XGBClassifier

import cupy, cudf # GPU libraries

import matplotlib.pyplot as plt, gc, os

import gc

import warnings

warnings.filterwarnings('ignore')

def read\_train\_file(path = '', usecols = None):

    # LOAD DATAFRAME

    if usecols is not None: df = cudf.read\_parquet(path, columns=usecols)

    else: df = cudf.read\_parquet(path)

    # REDUCE DTYPE FOR CUSTOMER AND DATE

    df['customer\_ID'] = df['customer\_ID'].str[-16:].str.hex\_to\_int().astype('int64')

    df.S\_2 = cudf.to\_datetime( df.S\_2 )

    df = df.fillna(0)

    print('shape of data:', df.shape)

    return df

print('Reading train data...')

TRAIN\_PATH = '../input/amex-data-integer-dtypes-parquet-format/train.parquet'

train = read\_train\_file(path = TRAIN\_PATH)

def process\_and\_feature\_engineer(df):

    all\_cols = [c for c in list(df.columns) if c not in ['customer\_ID','S\_2']]

    cat\_features = ["B\_30","B\_38","D\_114","D\_116","D\_117","D\_120","D\_126","D\_63","D\_64","D\_66","D\_68"]

    num\_features = [col for col in all\_cols if col not in cat\_features]

    test\_num\_agg = df.groupby("customer\_ID")[num\_features].agg(['mean', 'std', 'min', 'max', 'last'])

    test\_num\_agg.columns = ['\_'.join(x) for x in test\_num\_agg.columns]

    test\_cat\_agg = df.groupby("customer\_ID")[cat\_features].agg(['count', 'last', 'nunique'])

    test\_cat\_agg.columns = ['\_'.join(x) for x in test\_cat\_agg.columns]

    df = cudf.concat([test\_num\_agg, test\_cat\_agg], axis=1)

    del test\_num\_agg, test\_cat\_agg

    print('shape after engineering', df.shape )

    return df

train = process\_and\_feature\_engineer(train)

targets = cudf.read\_csv('../input/amex-default-prediction/train\_labels.csv')

targets['customer\_ID'] = targets['customer\_ID'].str[-16:].str.hex\_to\_int().astype('int64')

targets = targets.set\_index('customer\_ID')

train = train.merge(targets, left\_index=True, right\_index=True, how='left')

train.target = train.target.astype('int8')

del targets

train = train.sort\_index().reset\_index()

# FEATURES

FEATURES = train.columns[1:-1]

print(f'There are {len(FEATURES)} features!')

train\_pd = train.to\_pandas()

del train

\_ = gc.collect()

train\_df, test\_df = train\_test\_split(train\_pd, test\_size=0.25, stratify=train\_pd['target'])

del train\_pd

\_ = gc.collect()

len(train\_df),len(test\_df)

X\_train = train\_df.drop(['customer\_ID', 'target'], axis=1)

X\_test = test\_df.drop(['customer\_ID', 'target'], axis=1)

X\_train.head()

y\_train = train\_df['target']

y\_test = test\_df['target']

y\_train.head()

del train\_df, test\_df

\_ = gc.collect()

def objective(trial):

    param = {

        'booster':'gbtree',

        'tree\_method':'gpu\_hist',

        "objective": "binary:logistic",

        'lambda': trial.suggest\_loguniform(

            'lambda', 1e-3, 10.0

        ),

        'alpha': trial.suggest\_loguniform(

            'alpha', 1e-3, 10.0

        ),

        'colsample\_bytree': trial.suggest\_float(

            'colsample\_bytree', 0.5,1,step=0.1

        ),

        'subsample': trial.suggest\_float(

            'subsample', 0.5,1,step=0.1

        ),

        'learning\_rate': trial.suggest\_float(

            'learning\_rate', 0.001,0.05,step=0.001

        ),

        'n\_estimators': trial.suggest\_int(

            "n\_estimators", 80,1000,10

        ),

        'max\_depth': trial.suggest\_int(

            'max\_depth', 2,10,1

        ),

        'random\_state': 99,

        'min\_child\_weight': trial.suggest\_int(

            'min\_child\_weight', 1,256,1

        ),

    }

    model = XGBClassifier(\*\*param, enable\_categorical = True)

    model.fit(X\_train,y\_train)

    preds = pd.DataFrame(model.predict(X\_test))

    accuracy = sklearn.metrics.accuracy\_score(pd.DataFrame(y\_test.reset\_index()['target']),preds)

    return accuracy

study = optuna.create\_study(direction='maximize')

study.optimize(objective, n\_trials= 5)

best\_params = study.best\_trial.params

best\_params['tree\_method'] = 'gpu\_hist'

best\_params['booster'] = 'gbtree'

final\_model = XGBClassifier(\*\*best\_params,enable\_categorical = True)

final\_model.fit(X\_train,y\_train)

result = final\_model.predict\_proba(X\_test)[:,1]

result = np.array([1 if i > 0.5 else 0  for i in result])

len(y\_test), len(result)

cm = np.zeros((2,2))

count = 0

for tval, pval in zip(y\_test, result):

    count += 1

    cm[tval][pval] += 1

count, cm

import seaborn as sns

ax = sns.heatmap(cm)

ax.set\_title('Seaborn Confusion Matrix with labels\n\n');

ax.set\_xlabel('\nPredicted Values')

ax.set\_ylabel('Actual Values ');

ax.xaxis.set\_ticklabels(['False','True'])

ax.yaxis.set\_ticklabels(['False','True'])

plt.show()

final = pd.DataFrame(test['prediction'].to\_pandas())

final.to\_csv("submission.csv", index=True)

# Works Cited :

- Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 785-794).

- Chen, T., et al. (2021). XGBoost. GitHub repository. <https://github.com/dmlc/xgboost>

# - Dataset from the “American Express - Default Prediction” competition https://www.kaggle.com/competitions/amex-default-prediction/data