HOME CREDIT DEFAULT RISK - PHASE 3

TEAM MEMBERS:

Hanish Chidipothu - hachid@iu.edu
Nidhi Vraj Sadhuvala - nsadhuva@iu.edu
Chandra Sagar Bhogadi - cbhogadi@iu.edu
Srinivas Vaddi - svaddi@iu.edu

Hanish, Sagar and Nidhi are Master's Students in Data Science while Srinivas is a Computer Science student at Luddy School of Informatics. We met in this class and have been working on this project together. We learned a lot and had a great time on our journey. These strategies, we believe, will be beneficial in our job market. We met on a regular basis at Zoom meetings to plan our project and produce Python code and reports.

ABSTRACT

HomeCredit uses Machine Learning Modeling to provide unsecured loans based on a user's credit history, repayment behaviors, and other data. Credit history is a metric that explains a user's reliability based on factors such as the user's average/minimum/maximum balance, Bureau scores recorded, salary, and repayment practices. We used kaggle dataset to undertake exploratory data analysis, develop machine learning pipelines, and evaluate models across many evaluation metrics for a model to be deployed as part of this project. We used a deep learning model in Phase-3. Using Pytorch, we created a binary classification Machine Learning model in Python. We created a model, trained it, and evaluated it. We accomplished this by constructing a neural network (NN) with one linear layer, a RELU layer, and a sigmoid function. The linear NN had a high AUC of 0.609 and a 91% test accuracy.

DATA AND TASK DESCRIPTION

- application_{train|test}.csv
  This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET). Static data for all applications. One row represents one loan in our data sample.
- POS_CASH_balance.csv
  This dataset gives information about previous credits information such as contract status, number of installments left to pay, DPD(days past due), etc. of the current application. Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
- bureau.csv
  All client's previous credits provided by other financial institutions were reported to the Credit Bureau (for clients who have a loan in our sample). For every loan in our sample, there are as many rows as the number of credits the client had in the Credit Bureau before the application date.
- bureau_balance.csv
  Monthly balances of previous credits in the Credit Bureau. This table has one row for each month of history of every previous credit reported to the Credit Bureau.
- credit_card_balance.csv
  Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.
- previous_application.csv
  All previous applications for Home Credit loans of clients who have loans in our sample. There is one row for each previous application related to loans in our data sample.
- installments_payments.csv
  Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample. There is a) one row for every payment that was made plus b) one row each for missed payment. One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.

Train dataset in application.csv:

- Shape: (307511, 122)
- First five rows and seven columns look like:

SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
100002	1	Cash loans	M	N	Y	0
100003	0	Cash loans	F	N	N	0
100004	0	Revolving loans	M	Y	Y	0
100006	0	Cash loans	F	N	Y	0
100007	0	Cash loans	M	N	Y	0

- We discarded features with null values more than 30%. We present null percentage of ten features in the below table as an example.

  Nullity/missing values via column graph:

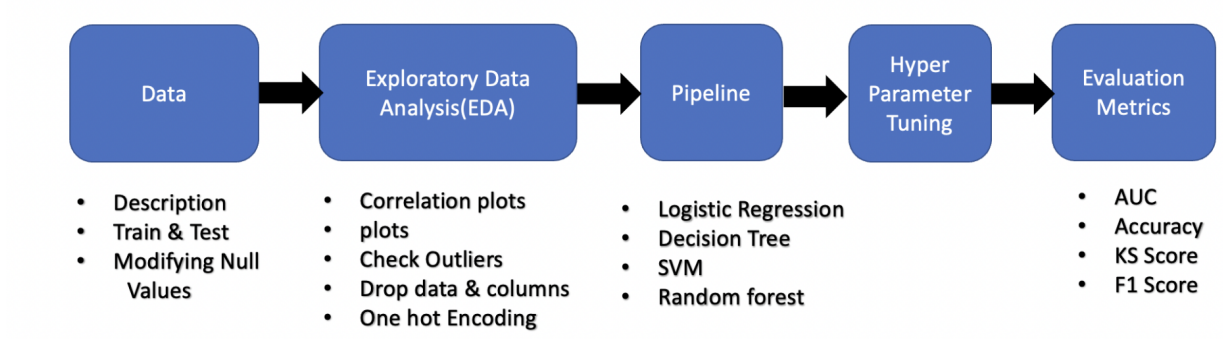
	col_name	null_count	count_%
0	NAME_CONTRACT_TYPE	0	0.000000
1	CODE_GENDER	0	0.000000
2	FLAG_OWN_CAR	0	0.000000
3	FLAG_OWN_REALTY	0	0.000000
4	CNT_CHILDREN	0	0.000000
...	...	...	...
115	AMT_REQ_CREDIT_BUREAU_DAY	41519	13.501631
116	AMT_REQ_CREDIT_BUREAU_WEEK	41519	13.501631
117	AMT_REQ_CREDIT_BUREAU_MON	41519	13.501631
118	AMT_REQ_CREDIT_BUREAU_QRT	41519	13.501631
119	AMT_REQ_CREDIT_BUREAU_YEAR	41519	13.501631

TASK TO BE TACKLED

HomeCredit uses Machine Learning Modeling to provide unsecured loans based on the consumers' historical credit history, repayment trends, and other data. Credit history is a metric that explains a user's trustworthiness based on variables such as the user's average/minimum/maximum balance, Bureau scores reported, salary, and repayment patterns. We use kaggle datasets to undertake exploratory data analysis, develop machine learning pipelines, and evaluate models across many evaluation metrics for a model to be deployed as part of this project.

The goal of the HCDR initiative is to forecast the ability of the financially underserved population to repay loans. This project is significant because both the lender and the borrower require well-established predictions. Real-time Homecredit can show its consumers loan offers with the highest amount and APR thanks to its ML pipelines, which acquire data from data suppliers via APIs, run EDA, and fit it to the model to generate scores in microseconds. As a result, risk analysis becomes extremely important in this situation, because NPA (Non-Performing Asset) is expected to be less than 5% in order to run a profitable firm.

WORKFLOW



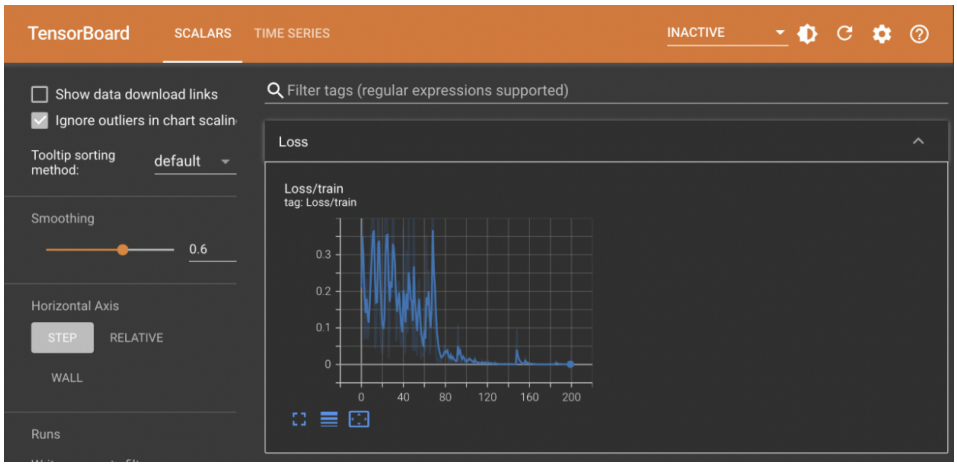
IMPLEMENTING NEURAL NETWORK

According to,

https://machinelearningmastery.com/pytorch-tutorial-develop-deep-learning-models/, the life-cycle of a PyTorch model has five steps. Prepare the data first. The second step is to define the model. Finally, the model must be trained. Finally, assess the model before making predictions.

- In section 3, we completed the first step. We'll define and train the model in section 4.
- To begin, in the Figure 2 below, we show a loss prediction model on a TensorFlow plot. In chart scaling, we ignored outliers. Smoothing was set to 0.6.
- The optimal loss function plot is shown below in Figure 2 for various numbers of epochs.

Figure 2. Plotting the best loss function over a range of epochs



DEFINE THE MODEL

We define the layers of a model when we define it. To override the layers and propagate input, we use the forward() function. We employ Linear, Conv2d, MaxPool2d, RELU, Softmax, and Sigmoid extensively among the many available layers.

- A linear layer, for example, connects layers. They were convoluted by Conv2d, then they were pooled by Maxpool2d.
- Activation functions include RELU, Softmax, and Sigmoid layers. We have codes for these in our ipynb.

```
# model definition
class MLP(Module):
    # define model elements
    def __init__(self, n_inputs):
        super(MLP, self).__init__()
        # input to first hidden layer
        self.hidden1 = Linear(n_inputs, 140)
        kaiming_uniform_(self.hidden1.weight, nonlinearity='relu')
        self.act1 = ReLU()
        # second hidden layer
        self.hidden2 = Linear(140, 20)
        kaiming_uniform_(self.hidden2.weight, nonlinearity='relu')
        self.act2 = ReLU()
        # third hidden layer and output
        self.hidden3 = Linear(20, 1)
        xavier_uniform_(self.hidden3.weight)
        self.act3 = Sigmoid()

    # forward propagate input
    def forward(self, X):
        # input to first hidden layer
        X = self.hidden1(X)
        X = self.act1(X)
         # second hidden layer
        X = self.hidden2(X)
        X = self.act2(X)
        # third hidden layer and output
        X = self.hidden3(X)
        X = self.act3(X)
        return X
```

TRAIN THE MODEL

We define loss functions and optimization strategies in this section:

- Loss functions: The loss functions that we learned in class are used. That is, the loss of binary cross-entropy. For binary classification, this loss function is utilized.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1 - y_i) \cdot log(1 - p(y_i))$$

- CrossEntropyLoss - This loss function is used for categorical classification.

$$J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(\hat{p}_k^{(i)}\right)$$

- Mean Squared Error loss

$$L(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N}(y - \hat{y}_i)^2$$

- As an optimization algorithm, we employ stochastic gradient descent.
- For training epochs, we create a loop. Then, for mini-batches and stochastic gradient descent, we write an inner loop.
- The ipynb file contains the codes for training the model.
- The next section will go over how to evaluate the model.

```
# train the model
def train_model(train_dl, model):
    # define the optimization
    criterion = MSELoss()
    optimizer = SGD(model.parameters(), lr=0.01, momentum=0.9)
    # enumerate epochs
    for epoch in range(100):
        # enumerate mini batches
        for i, (inputs, targets) in enumerate(train_dl):
            # clear the gradients
            optimizer.zero_grad()
            # compute the model output
            yhat = model(inputs)
            # calculate loss
            loss = criterion(yhat, targets)
            # credit assignment
            loss.backward()
            # update model weights
            optimizer.step()
```

EVALUATE THE MODEL

We may now evaluate the model on the test dataset because it has been fitted.

- With the predictions and actuals, we defined the evaluate model.
- We got a numpy array and rounded the numbers to class values. Then we stored it.
- The final step is to calculate accuracy.
- So we gathered the test dataset predictions, compared them to the expected values of the test set predictions, and finally produced the performance metric. As previously said, we have taken the reference from:

  https://machinelearningmastery.com/pytorch-tutorial-develop-deep-learning-models/

- The ipynb file contains the codes.

```
# evaluate the model
def evaluate_model(test_dl, model):
    predictions, actuals = [], []
    for i, (inputs, targets) in enumerate(test_dl):
        # evaluate the model on the test set
        yhat = model(inputs)
        # retrieve numpy array
        yhat = yhat.detach().numpy()
        actual = targets.numpy()
        actual = actual.reshape((len(actual), 1))
        # round to class values
        yhat = yhat.round()
        # store
        predictions.append(yhat)
        actuals.append(actual)
    predictions, actuals = vstack(predictions), vstack(actuals)
    # calculate accuracy
    acc = accuracy_score(actuals, predictions)
    return acc
```

LEAKAGE

When data from outside the training dataset is used to build the model, this is known as data leakage. Because the model is being run on data that it has already seen, data leakage frequently causes unnaturally high levels of performance on the test set.

We run the model several times to make sure there are no data leaks. Across numerous runs, the results were nearly the same, and there was no sudden increase in accuracy. We additionally tested and validated the model on numerous test and validation sets to ensure that the findings were consistent. In those several runs, we likewise didn't notice a jump in accuracy. We also checked for duplicates. This ensures that no duplicate examples are used in both training and test data.

MODELING PIPELINES/ RESULTS AND DISCUSSIONS:

(1) Input feature families and counts per family / Total number of input features

The following are the first five rows and twelve columns:

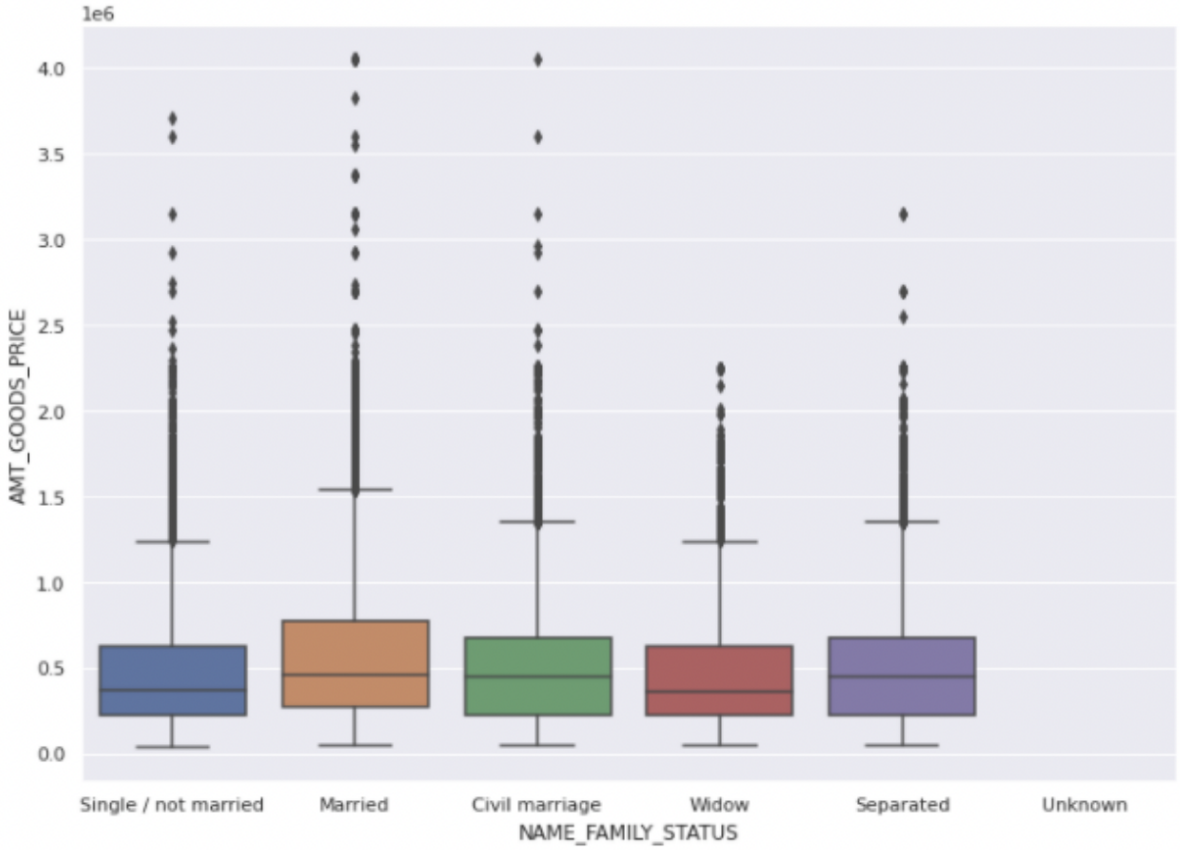
SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	NAME_TYPE_SUITE
100002	1	Cash loans	M	N	Y	0	202500.0	406597.5	24700.5	351000.0	Unaccompanied
100003	0	Cash loans	F	N	N	0	270000.0	1293502.5	35698.5	1129500.0	Family
100004	0	Revolving loans	M	Y	Y	0	67500.0	135000.0	6750.0	135000.0	Unaccompanied
100006	0	Cash loans	F	N	Y	0	135000.0	312682.5	29686.5	297000.0	Unaccompanied
100007	0	Cash loans	M	N	Y	0	121500.0	513000.0	21865.5	513000.0	Unaccompanied

Table - Head of Input features

We used imputation to fill null values in the NAME TYPE SUITE column with "Other C" because our null values were not trivial.

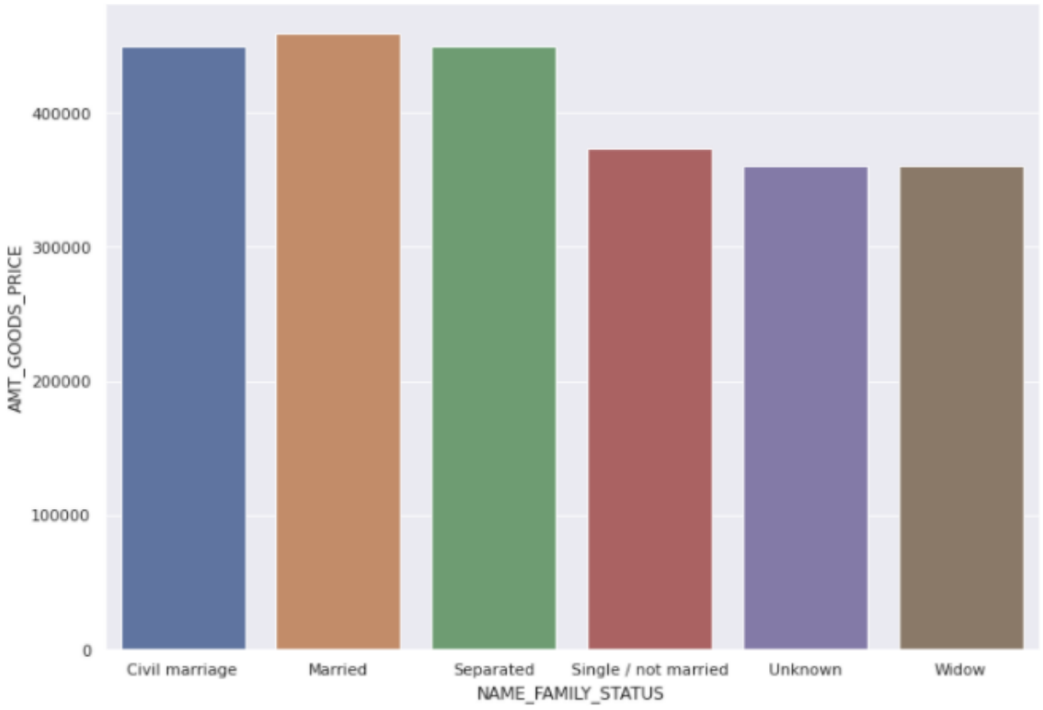
- In the columns containing the phrase AMT REQ CREDIT, we filled null values with 0.
- In the column containing the phrase CNT SOCIAL CIRCLE, we filled null values with 0.
- We used median to fill in the null values in the CNT FAM MEMBERS column.
- We used the median for the corresponding category to fill null values in the column AMT GOODS PRICE.
- We removed one row that had the column DAYS LAST PHONE CHANGE set to null.
- We removed 12 records that had the column AMT ANNUITY set to null.

We evaluate the families of input attributes and count per family with bar and box plots for more exploratory data analysis. Using these plots, we can easily count the amount of input features.



Box plots of input feature families

- Each family status has a similar median.
- The married median is the greatest, while the single median is the lowest.
- The married group is the characteristic with the most outliers.



Bar plots showing input feature families

The AMT GOODS Price of each group is shown in the above Figure based on the family status:

- The married group, for example, has the highest AMT GOODS PRICE of about 450000.
- With an AMT GOODS PRICE of around 360000, the widow group has the lowest AMT GOODS PRICE.
- Separated and civil marriages have roughly the same AMT GOODS PRICE of 430000.
- After Widow, the second lowest group (excluding unknown) is Single and Not Married, with around 365000 AMT GOODS PRICE.

We also check the correlation with EXT_3 as shown in the below table:

	col_name	correlation_with_EXT_3
15	DAYS_BIRTH	0.205474
70	TARGET	0.178929
18	DAYS_ID_PUBLISH	0.131598
20	FLAG_EMP_PHONE	0.115284
16	DAYS_EMPLOYED	0.113426
37	EXT_SOURCE_2	0.109728
17	DAYS_REGISTRATION	0.107570
5	AMT_INCOME_TOTAL	0.088906
36	ORGANIZATION_TYPE	0.087994

- The top ten correlations of input features with EXT 3 are shown in the above Table. DAYS BIRTH has the highest correlation with EXT 3 (20.5%).
- ORGANIZATION TYPE has a correlation of 8.8 percent with EXT 3, which is the top tenth correlation.

Hyperparameters and settings taken into account

For decision Making Tree, Lasso Regression, Ridge Regression, and Logistic Regression, hyperparameter tweaking for grid search was performed.

- For a Decision Tree model, we experimented with different maximum depths and sample splits.
- For Lasso Regression, we varied the alpha parameters and adjusted the penalty weighting to the loss function.
- For Ridge Regression, we employed different alpha settings and varied the penalty to loss function weighting.
- We utilized several C parameters for Logistic Regression and varied the penalty strength.
- We modified the amount of epochs, layers, and neurons per layer in PyTorch to improve deep learning performance.

LOSS FUNCTIONS:

As previously stated, we employ loss functions in the following manner.

- Loss functions include: The loss functions that we learned in class are used. That is, the loss of binary cross-entropy. For binary classification, this loss function is utilized.

RESULTS AND DISCUSSIONS:

We tried altering numerous parameters such as the number of epochs, the number of hidden layers, and the number of neurons per layer as part of hyperparameter tuning for MLP classification deep neural networks using Pytorch. Then we tallied the AUC and accuracy for each iteration and discovered that there isn't much of a change, except for one combination with three hidden layers, where AUC rose (74,35,8).

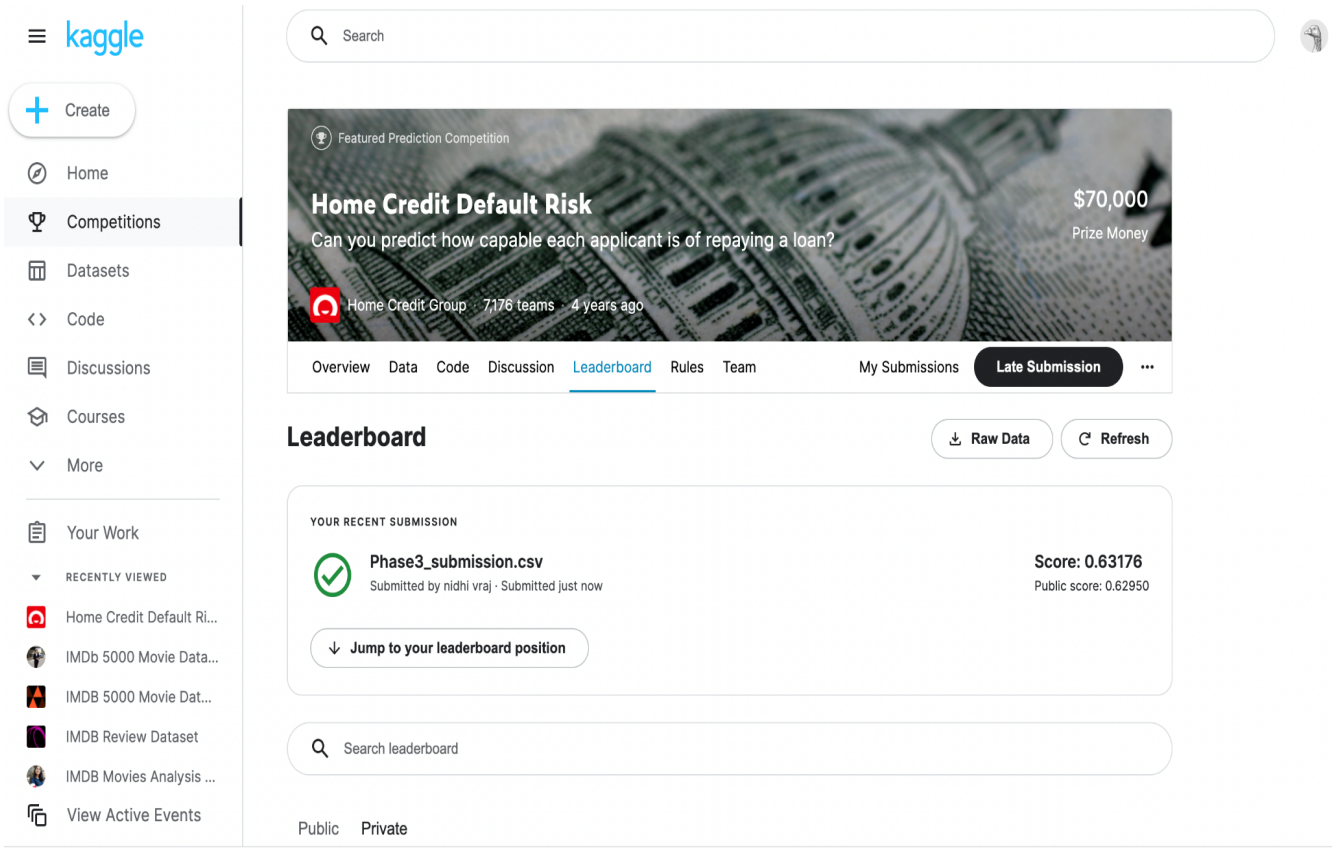
For binary classification, we used a Multilayer Perceptron Model.

The sigmoid activation function is used to forecast the probability of class 1 in the model. We used stochastic gradient descent to optimize the model. We used the TensorBoard to show the loss function. We also looked for Data Leakage by looking at Pre-Processing stages, Duplicates, and testing and validating the model many times.

For the past month, we have had the results of the investigations. The deep learning model has the best accuracy and AUC among the categorization models. The ridge regression model has the lowest MSE and the highest AUC among the regression models. In phase three, we used Pytorch to create the MLP Classification model, and we were able to achieve a better result. In the future, we can use the hybrid Pytorch-FastAI technique to try a multi-task deep learning model.

In order to summarize the project, We collected data in the first phase, performed EDA, and developed a baseline model using logistic regression. We executed feature engineering and tweaked the hyper parameters in the second step. Using Tensor board and binary classification, we constructed a pytorch deep learning model in this step. We intend to provide a new multi-task loss function in Pytorch as a stretch goal. The issue is that a multi-layer perceptron regression model did not perform adequately.

KAGGLE SUBMISSION



CONCLUSION

The goal of the HCDR initiative is to forecast the ability of the financially underserved population to repay loans. This project is significant because both the lender and the borrower require well-established predictions. Real-time Homecredit can show its consumers loan offers with the highest amount and APR thanks to its ML pipelines, which acquire data from data suppliers via APIs, run EDA, and fit it to the model to generate scores in microseconds. As a result, risk analysis becomes extremely important in this situation, because NPA (Non-Performing Asset) is expected to be less than 5% in order to run a profitable firm.

Credit history is a measure of a user's credibility that is calculated using characteristics such as the user's average/minimum/maximum balance, Bureau scores reported, salary, and repayment patterns that may be analyzed using the user's past timely defaults/repayments. Other criteria such as location data, social media data, calling/SMS data, and so on are included in alternate data. We would use the datasets given by kaggle for exploratory data analysis, machine learning pipelines, and model evaluation across many evaluation metrics for a model to be deployed as part of this project.

We estimated various models in phase 2, including classification and regression models. Feature selection, data imputation, and hyperparameter tweaking. We began by performing feature selection and imputation. The missing values of specified features were filled in. Then, based on our past understanding, we opted to add relevant functionality. The hyperparameters were then fine-tuned using GridSearchCV. We trained and assessed numerous models, including Logistic Regression, Decision Tree Model, Lasso Regression, and Ridge Regression, to discover the best one. In phase 2, classification models will not be able to outperform the baseline model. The ridge regression model has the best performance among regression models.

In phase 3, we used PyTorch to design a deep learning model and created additional models. The issues we had were that we couldn't increase the baseline model's test accuracy or AUC using the feature selection and imputation stage. Unlike regression models, we were unable to improve on the baseline classification model. To overcome these challenges, we propose to create a multilayer model for loan default classification in PyTorch in phase 3. We'll create and implement a new multitask loss function in Pytorch as a stretch goal. These were submitted to Kaggle, and our results were published.