HCDR Team 1 Bears

Zack Seliger
zseliger@gmail.com

Keegan Moore
keegmoor@iu.edu

Rajasimha
ragallam@iu.edu

Jagan Lakku
slakku@iu.edu

Abstract

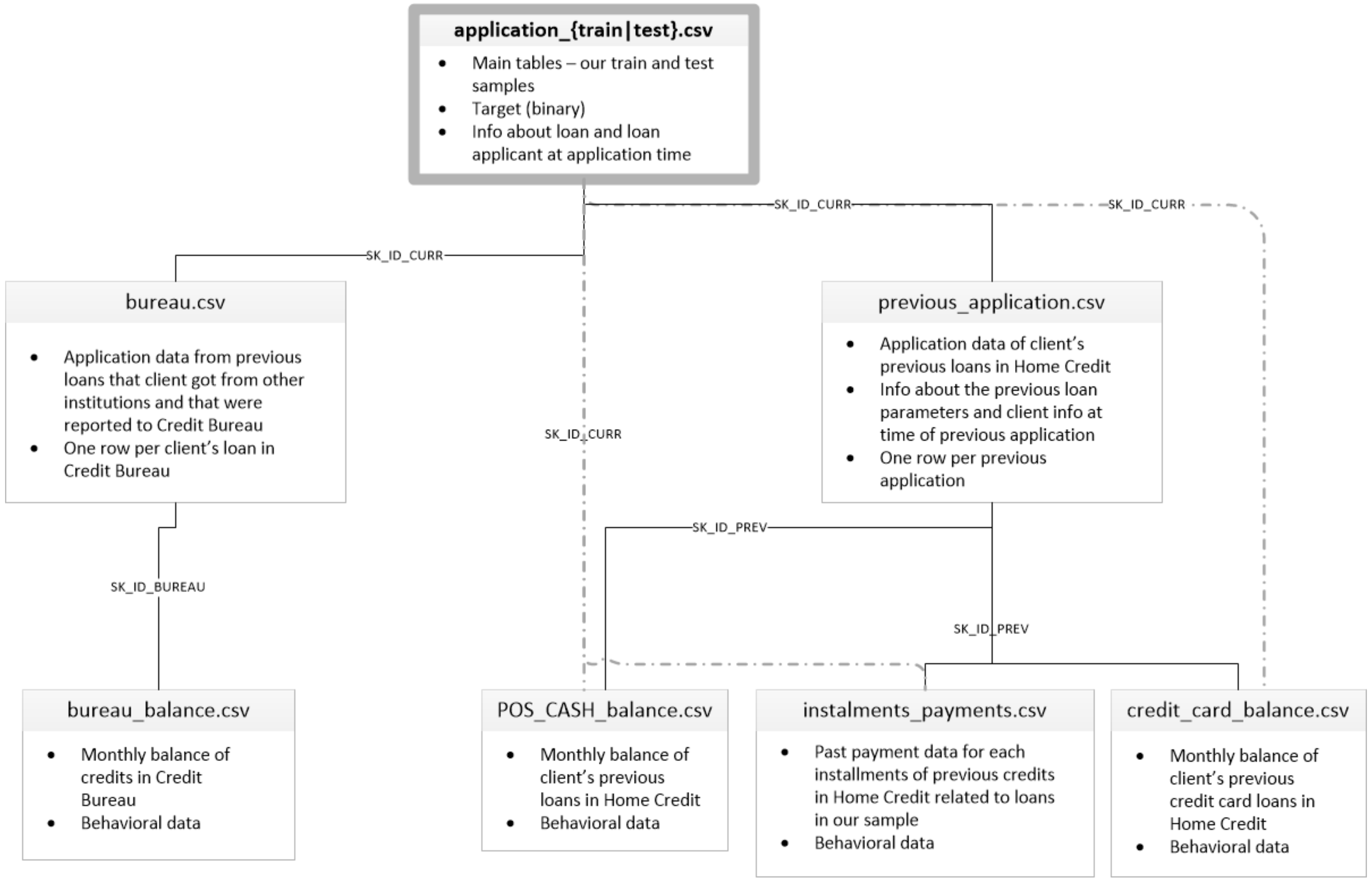
The goal of this project is to use a machine learning model that can predict whether or not someone would be able to repay a loan.

In Phase 2, we optimized our models and performed feature engineering to make our model better. We experimented with LogisticRegression, XGBoost, and LGBM models. We introduced more data that made our models stronger, and engineered and selected new features.

We trained and tuned an LGBM model, which was very strong, before moving on to XGBoost. Untrained XGBoost gave us decent results, but took much much longer to train. We also ran into RAM problems with Colab when running Grid Search, which ultimately made it impractical for us to do. In the end, we decided to submit with our tuned LGBM model, where we got a ROC_AUC score of 0.752. This is a relatively sizable increase from Phase 1, where our model got 0.729.

Description

Our data is split up among several different CSV files and looks like this:

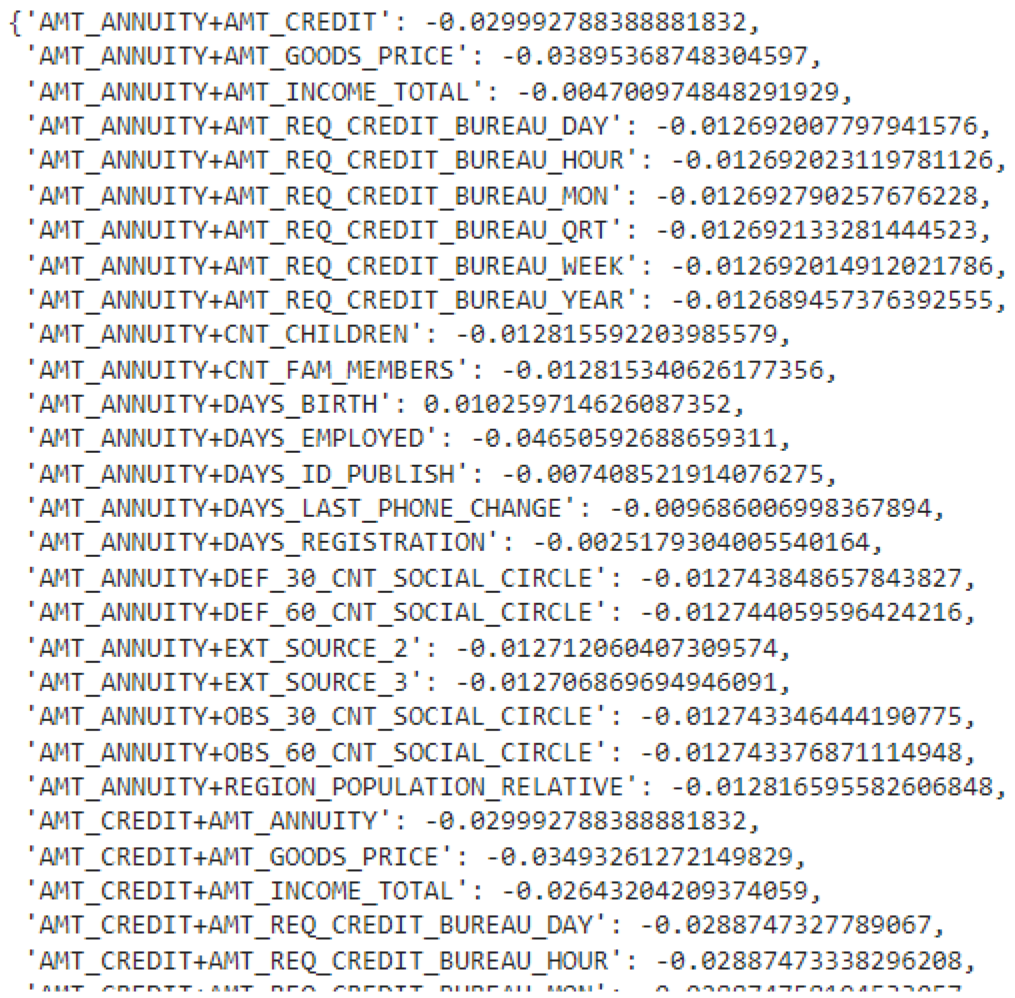

application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_BUREAU
SK_ID_PREV
SK_ID_PREV
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans in Home Credit
Behavioral data
instalments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

Application{train|test}.csv contains static data for all applications and one row represents one loan. SK_ID_CURR represents the ID for that row, which can be tied in with the other datasets for this problem. There are also more IDs, like SK_ID_BUREAU and SK_ID_PREV, that correspond to bureau and previous application data, respectively.
In Phase 1, we looked at the application dataset and datasets that we could access with just SK_ID_CURR. In Phase 2, we experimented with aggregating datasets using

SK_ID_BUREAU and SK_ID_PREV to collect more information about our loan. We also created new features by manipulating the features we had.

Feature Engineering

To start, we took a brute force approach. For each dataset, we performed every operation (addition, subtraction, multiplication, and division) with every numerical feature on every other numerical feature. To determine whether a new feature was valuable or not, we looked at the correlation with the target, and if it was greater than

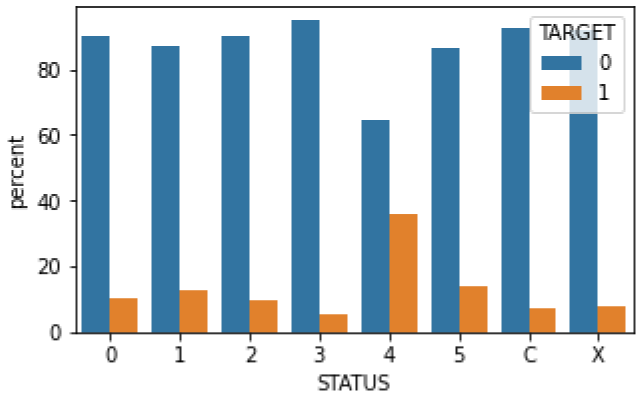


```
{'AMT_ANNUITY+AMT_CREDIT': -0.029992788388881832,
 'AMT_ANNUITY+AMT_GOODS_PRICE': -0.03895368748304597,
 'AMT_ANNUITY+AMT_INCOME_TOTAL': -0.004700974848291929,
 'AMT_ANNUITY+AMT_REQ_CREDIT_BUREAU_DAY': -0.012692007797941576,
 'AMT_ANNUITY+AMT_REQ_CREDIT_BUREAU_HOUR': -0.012692023119781126,
 'AMT_ANNUITY+AMT_REQ_CREDIT_BUREAU_MON': -0.012692790257676228,
 'AMT_ANNUITY+AMT_REQ_CREDIT_BUREAU_QRT': -0.012692133281444523,
 'AMT_ANNUITY+AMT_REQ_CREDIT_BUREAU_WEEK': -0.012692014912021786,
 'AMT_ANNUITY+AMT_REQ_CREDIT_BUREAU_YEAR': -0.012689457376392555,
 'AMT_ANNUITY+CNT_CHILDREN': -0.012815592203985579,
 'AMT_ANNUITY+CNT_FAM_MEMBERS': -0.012815340626177356,
 'AMT_ANNUITY+DAYS_BIRTH': 0.010259714626087352,
 'AMT_ANNUITY+DAYS_EMPLOYED': -0.04650592688659311,
 'AMT_ANNUITY+DAYS_ID_PUBLISH': -0.007408521914076275,
 'AMT_ANNUITY+DAYS_LAST_PHONE_CHANGE': -0.009686006998367894,
 'AMT_ANNUITY+DAYS_REGISTRATION': -0.0025179304005540164,
 'AMT_ANNUITY+DEF_30_CNT_SOCIAL_CIRCLE': -0.012743848657843827,
 'AMT_ANNUITY+DEF_60_CNT_SOCIAL_CIRCLE': -0.012744059596424216,
 'AMT_ANNUITY+EXT_SOURCE_2': -0.012712060407309574,
 'AMT_ANNUITY+EXT_SOURCE_3': -0.012706869694946091,
 'AMT_ANNUITY+OBS_30_CNT_SOCIAL_CIRCLE': -0.012743346444190775,
 'AMT_ANNUITY+OBS_60_CNT_SOCIAL_CIRCLE': -0.012743376871114948,
 'AMT_ANNUITY+REGION_POPULATION_RELATIVE': -0.012816595582606848,
 'AMT_CREDIT+AMT_ANNUITY': -0.029992788388881832,
 'AMT_CREDIT+AMT_GOODS_PRICE': -0.03493261272149829,
 'AMT_CREDIT+AMT_INCOME_TOTAL': -0.02643204209374059,
 'AMT_CREDIT+AMT_REQ_CREDIT_BUREAU_DAY': -0.0288747327789067,
 'AMT_CREDIT+AMT_REQ_CREDIT_BUREAU_HOUR': -0.02887473338296208,
 'AMT_CREDIT+AMT_REQ_CREDIT_BUREAU_MON': -0.028874750194522057
```

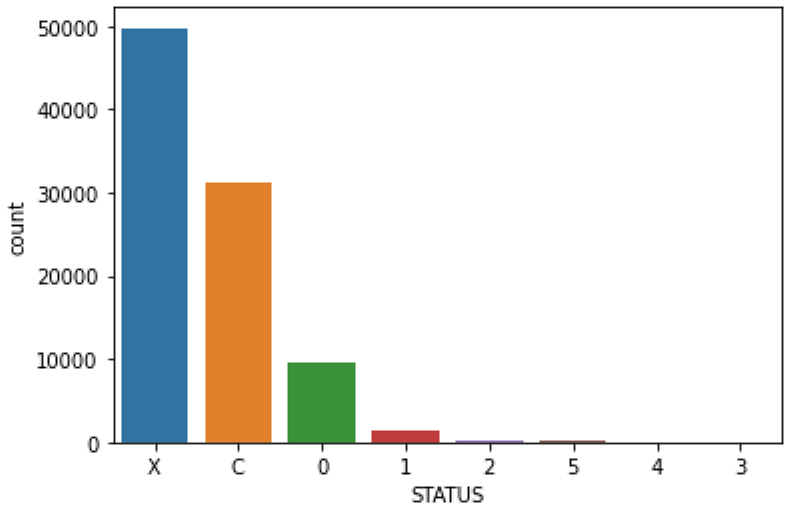
either individual feature, it was considered. Here is an example of application data and addition.

In this way, we added 11 features and were able to increase our ROC_AUC score from 0.74 to 0.745 using LogisticRegression.

We also came up with ways to use SK_ID_PREV and SK_ID_BUREAU. First, we aggregated other datasets based on SK_ID_PREV and combined them into the previous_applications dataset. Then, we aggregated previous_applications based on SK_ID_CURR. We selected the features with the highest correlation values and did EDA to determine which features to ultimately keep. We did a similar process with SK_ID_BUREAU on the bureau dataset. Here is an example of some EDA for the STATUS feature:



```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa501a8ee50>
```

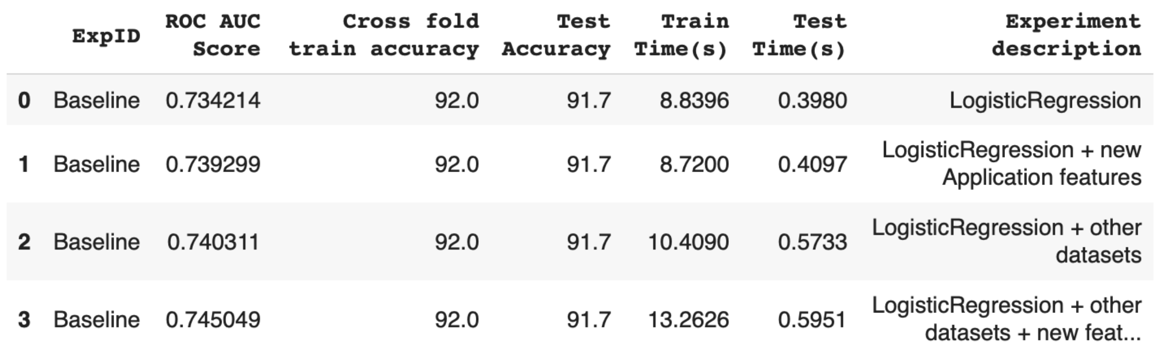


With this new data, we took our untuned LGBM model from an ROC_AUC score of 0.755 to 0.763.

Models

Baseline Model:

->We started working with a LogisticRegression model as our baseline. Firstly, we used the baseline model on application data and then on all data sets . We then added the new features to the baseline model and compared the results.

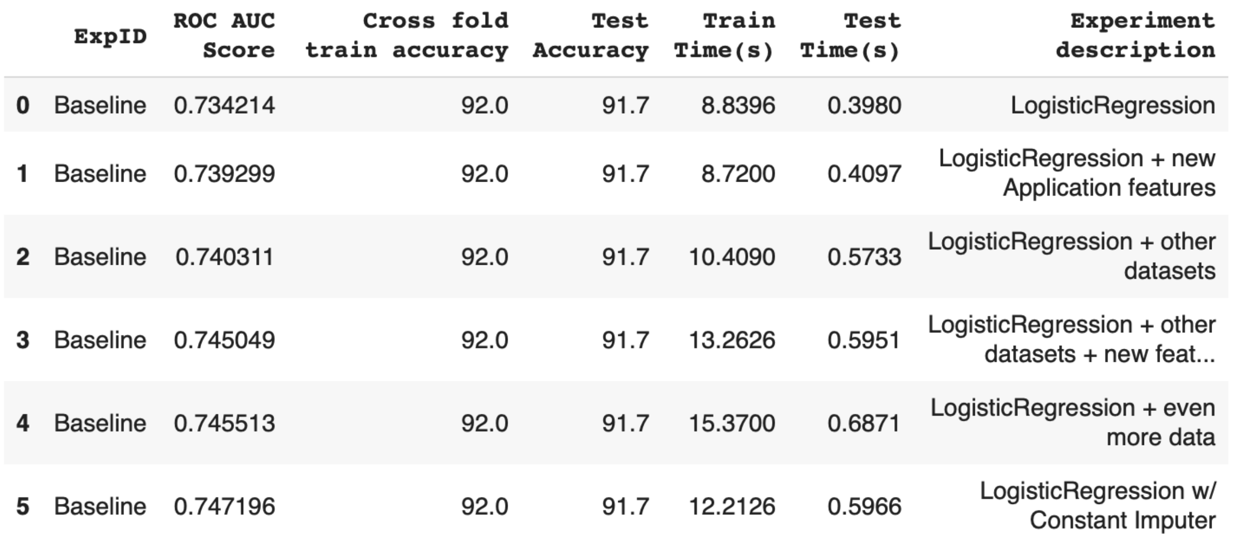


	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.3980	LogisticRegression
1	Baseline	0.739299	92.0	91.7	8.7200	0.4097	LogisticRegression + new Application features
2	Baseline	0.740311	92.0	91.7	10.4090	0.5733	LogisticRegression + other datasets
3	Baseline	0.745049	92.0	91.7	13.2626	0.5951	LogisticRegression + other datasets + new feat...

->As you can see, the results of our baseline model improve as we add more data. Between IDs 2 and 3, we added our newly engineered features.

Tweaking Imputers:

We had an idea to use a constant strategy with our categorial Imputer instead of the most frequent strategy. This is because, for some data types, the NaN values actually corresponded to another class instead of a lack of data. For example, for employment data, it seemed that unemployed clients were assigned NaN. Therefore, it doesn't make sense to group them in with another class and pollute our data. Below are the experimental results for the baseline model after making this change.

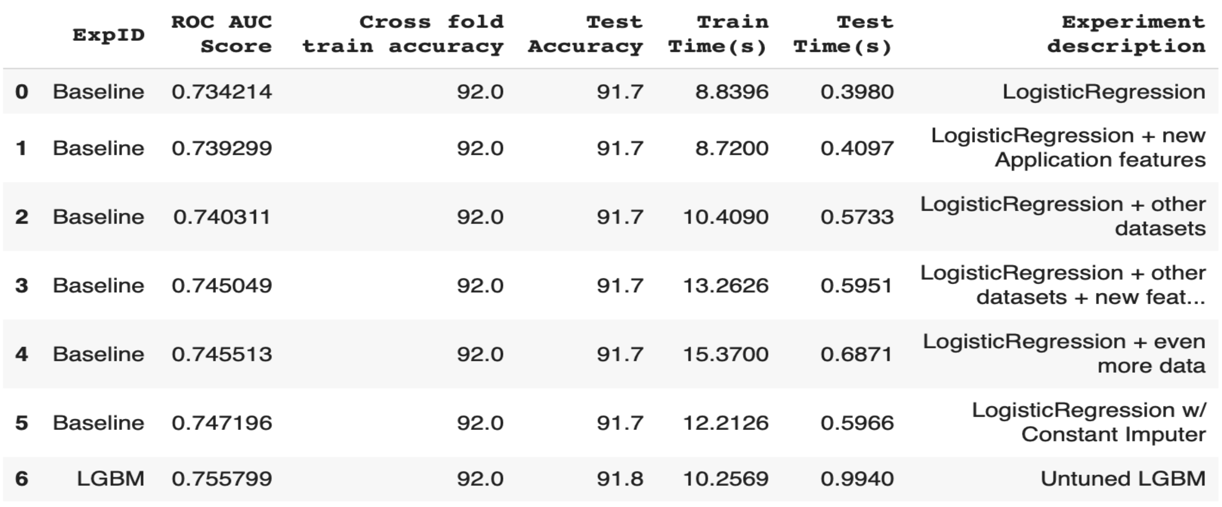


	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.3980	LogisticRegression
1	Baseline	0.739299	92.0	91.7	8.7200	0.4097	LogisticRegression + new Application features
2	Baseline	0.740311	92.0	91.7	10.4090	0.5733	LogisticRegression + other datasets
3	Baseline	0.745049	92.0	91.7	13.2626	0.5951	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92.0	91.7	15.3700	0.6871	LogisticRegression + even more data
5	Baseline	0.747196	92.0	91.7	12.2126	0.5966	LogisticRegression w/ Constant Imputer

->We can see from the above results that there is an improvement in ROC_AUC. So we should continue using this change to the imputer in the future.

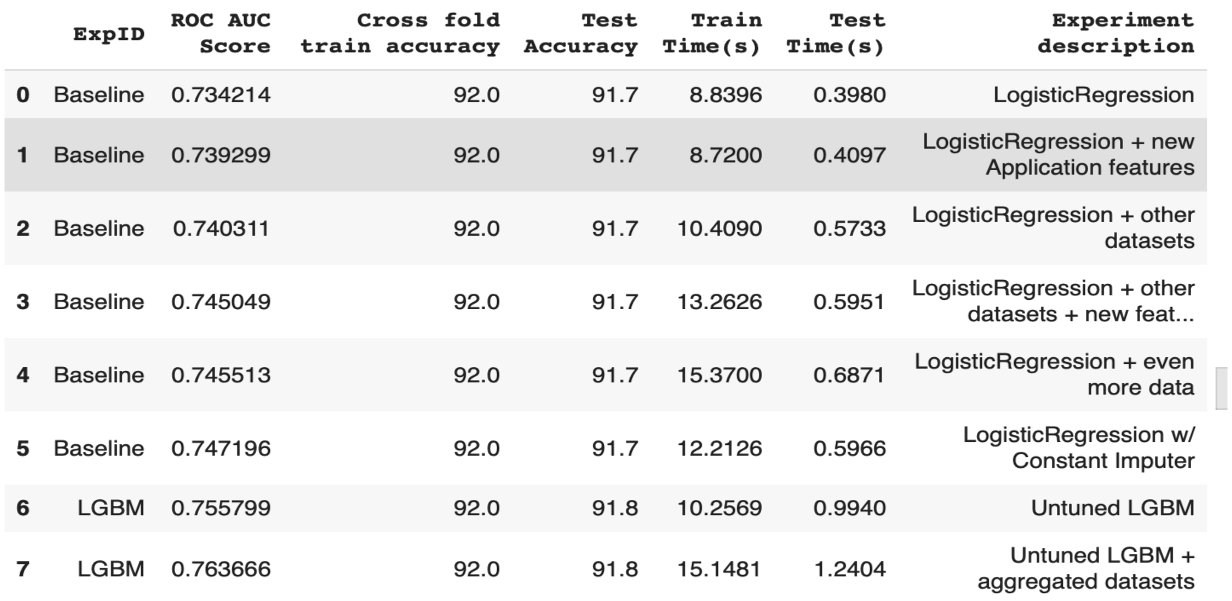
LGBM Model:

-> As a part of exploring more models in order to improve the ROC_AUC , conducted the experiment analysis on Untuned LGBM. Firstly, we trained an Untuned LGBM model to see if there is any improvement. As we can see that there is an improvement in ROC, we can still tune this and improve even more.



	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.3980	LogisticRegression
1	Baseline	0.739299	92.0	91.7	8.7200	0.4097	LogisticRegression + new Application features
2	Baseline	0.740311	92.0	91.7	10.4090	0.5733	LogisticRegression + other datasets
3	Baseline	0.745049	92.0	91.7	13.2626	0.5951	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92.0	91.7	15.3700	0.6871	LogisticRegression + even more data
5	Baseline	0.747196	92.0	91.7	12.2126	0.5966	LogisticRegression w/ Constant Imputer
6	LGBM	0.755799	92.0	91.8	10.2569	0.9940	Untuned LGBM

->Now that we have added the new dataset feature to the LGBM model and observed that there is a better improvement in the ROC score.These new features clearly make our model better. Now, we need to do optimize it using Grid Search



	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
0	Baseline	0.734214	92.0	91.7	8.8396	0.3980	LogisticRegression
1	Baseline	0.739299	92.0	91.7	8.7200	0.4097	LogisticRegression + new Application features
2	Baseline	0.740311	92.0	91.7	10.4090	0.5733	LogisticRegression + other datasets
3	Baseline	0.745049	92.0	91.7	13.2626	0.5951	LogisticRegression + other datasets + new feat...
4	Baseline	0.745513	92.0	91.7	15.3700	0.6871	LogisticRegression + even more data
5	Baseline	0.747196	92.0	91.7	12.2126	0.5966	LogisticRegression w/ Constant Imputer
6	LGBM	0.755799	92.0	91.8	10.2569	0.9940	Untuned LGBM
7	LGBM	0.763666	92.0	91.8	15.1481	1.2404	Untuned LGBM + aggregated datasets

Grid Search for LGBM:

Below shows us the best parameters which fit in for the LGBM model for the given data

```
Fitting 3 folds for each of 192 candidates, totalling 576 fits
best train score: 76.3
Best Parameters:
        predictor__colsample_bytree: 0.5
        predictor__max_depth: 10
        predictor__min_split_gain: 1
        predictor__num_leaves: 31
```

Tuned LGBM:

-> Now we have added the extra data that we got from the "More Data" experiment to an LGBM model with tuned hyperparameters

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
8	LGBM	0.763879	92	91.7	42.14050	2.552300	LGBM tuned

Untuned XGBoost Model:

Now that we've taken a deep dive into our LGBM model, we looked into XGBoost. We started by training an untuned XGBoostClassifier with all of our data.

	ExpID	ROC AUC Score	Cross fold train accuracy	Test Accuracy	Train Time(s)	Test Time(s)	Experiment description
9	XGBoost	0.756850	91.9	91.7	230.20260	1.238900	Untuned XGBoost

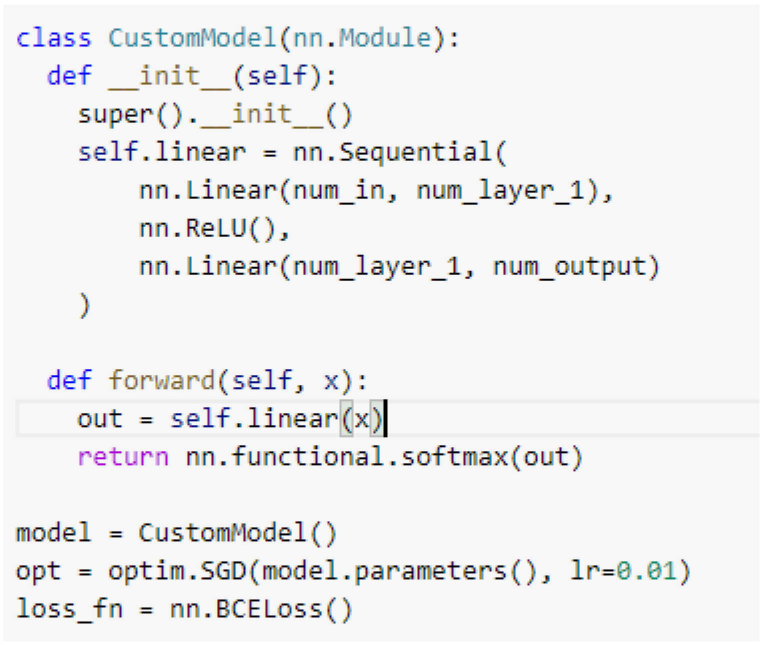
We noticed a few things here. One was that it was a bit worse than LGBM, but that doesn't matter too much. What does matter is the training time. For untuned LGBM vs untuned XGBoost, XGBoost took over 10 times as long to train.

Nevertheless, we tried to perform Grid Search, but we ran into another problem. For our search, XGBoost used up too much RAM and we weren't able to get results from it, even after it taking much longer than LGBM to train.

In the end, we decided to not use XGBoost and only submit our tuned LGBM model.

Neural Network:

We created a Neural Network model for our data. To make it simpler and easier to train, we only used data from application_train and application_test. We used our sklearn pipeline to preprocess our data before feeding it into our model. Our architecture was pretty simple as well.



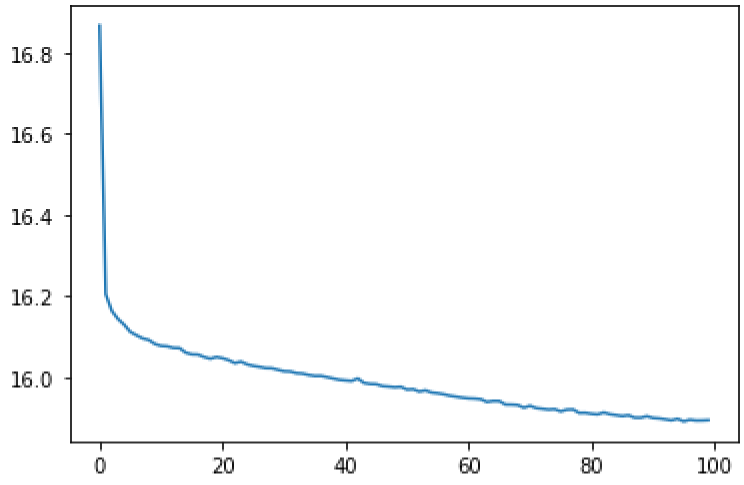
```
class CustomModel(nn.Module):
  def __init__(self):
    super().__init__()
    self.linear = nn.Sequential(
        nn.Linear(num_in, num_layer_1),
        nn.ReLU(),
        nn.Linear(num_layer_1, num_output)
    )

  def forward(self, x):
    out = self.linear(x)
    return nn.functional.softmax(out)

model = CustomModel()
opt = optim.SGD(model.parameters(), lr=0.01)
loss_fn = nn.BCELoss()
```

We had an input of the number of features we had, which was around 78. We used a ReLU activation function before going to our hidden layer, which we arbitrarily decided to be 20 nodes large. We used softmax for our output.

We chose to use Stochastic Gradient Descent as our optimizer and a loss function of Binary Cross-Entropy, which can be defined as:

$$l_n = -w_n \left[y_n \cdot \log x_n + (1 - y_n) \cdot \log(1 - x_n) \right]$$

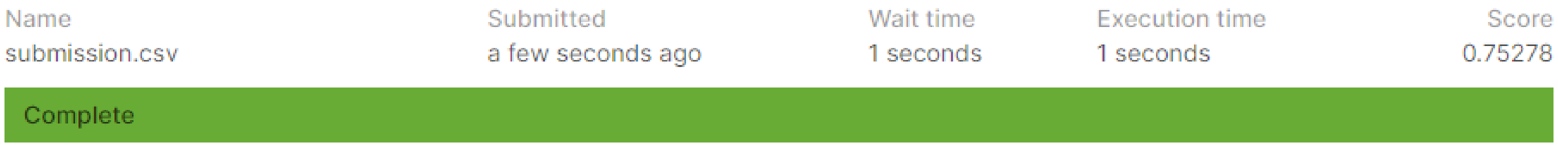
We trained this model for 100 epochs and a batch size of 64. Here is the loss over time:



We achieved a final ROC_AUC of 0.739, which is actually better than our baseline model for the same data, which was 0.734.

Conclusion

The accuracy of these kinds of machine learning models is important, since it can directly affect the lives of those looking for loans. We wish to make our models as optimal as possible, and we believe well-executed machine learning models can quite accurately predict HCDR. In this phase, we used rigorous methods to gather the most from our data, and have tested many different kinds of models. After feature engineering and hyperparameter tuning, we submitted to Kaggle and got a ROC_AUC of 0.752. This is an improvement from our basic model, and is a stepping stone to finishing on a good note in phase 3 with more improvements.



Name	Submitted	Wait time	Execution time	Score
submission.csv	a few seconds ago	1 seconds	1 seconds	0.75278

Complete

We also trained a basic Neural Network, but just with the basic application data. For Phase 3, we plan on adding all of our other features to the neural network model, and tweaking the network architecture, optimizers, and hyperparameters.