Phase 4 Report

Group Number 7

Michael Casey

Andrew Cruez

Peter Stewart

Hemraj Yadav

Home Credit Default Risk (HCDR)

Phase Leader Plan

Phase	Phase Leader	Phase Name	Contributor Name	Contribution
Phase 1	Stewart, Peter Vincent	Project Proposal	Stewart, Peter Vincent	Create Credit Assignment Plan, Machine Learning Pipeline flow, Coordinate Phase Tasks, Phase Planning, phase submission and Discussion
			Yadav, Hemraj	Abstract , Data Description, Data Dictionary, Phase Planning
			Cruez, Andrew Thomas	Phase Leader Plan, Phase Planning
			Casey, Michael	Machine Learning Algorithms and Metrics, Phase Planning, Schedule meetings
Phase 2	Yadav, Hemraj	Explortory Data Analysis and Baseline Pipeline modeling	Yadav, Hemraj	Data Dictionary, Coorelation, Text based Analysis, Visual EDA, Missing Data Analysis, Coordinate Phase Tasks, Phase Planning, phase submission and Discussion
			Stewart, Peter Vincent	Update Credit Assignment Plan, Data Description, Build Pipeline: Multi-Class Perceptron, Loss Function Esed, Experiment Table, Feature Engineering, 2 minutes video
			Cruez, Andrew Thomas	Update Abstract, Workflow Diagrams, Build Pipeline: Naive Bayes, Written Result and Conclusion, 2 minutes video
			Casey, Michael	Create Slide Deck, Prepare talking points/ script, Write Abstract, Build Pipeline: Logistic Regression,Schedule meeting, 2 minutes video
Phase 3	Cruez, Andrew Thomas	Hyperparameters and Testing Metrics	Cruez, Andrew Thomas	hyperparameters tunning, Feature Engineering, Data Description, Credit Assignment plan, 2 minutes video, Presentation, Phase Planning, Submit the assignment and discussion
			Stewart, Peter Vincent	hyperparameters tunning, Feature Engineering, feature selection, Presentation, 2 minutes video
			Casey, Michael	Feature Engineering,Update Abstract, Visual EDA, Model Evaluation, Presentation, 2 minutes video, Schedule meeting
			Yadav, Hemraj	Feature Engineering, Data Description, Visual EDA, Project Leader Plan, Presentation, 2 minutes video
Phase 4	Casey, Michael	Final Report	Casey, Michael	Update Abstract, Neural Network, Phase Planning, Schedule meeting, Submit the assignment and discussion, kaggle Submission, Finalize Presentation
			Stewart, Peter Vincent	Advance model architecture, Neural Network, hperparameters, experiments, Finalize Presentation
			Yadav, Hemraj	Neural Network, Visual EDA, Data Description, Project Leader Plan, Finalize Presentation
			Cruez, Andrew Thomas	Conclusion, Neural Network, Discussion, Credit Assignment Plan, Finalize Presentation

Abstract

We modified our feature engineering to include PCA(50, 150 and 250 components) and SelectKBest(chi2 and K = 100, 200 and 300)). We compared these feature engineering tools against our baseline dataset(full 731 features as defined in the Data Lineage Diagram) and found the full dataset produced higher AUC scores vs Kbest or PCA. We evaluated 7 MLP, 5 Logistic Regression, 4 Neural Net(NN) architectures and 3 Multinomial NB classifiers. The MLP experiments (using best classifier sgd solver) show the highest train AUC score of .7427 occurring with the full dataset (though PCA with 250 components at AUC .7416). Neural Nets had the lowest test AUC score of 0.513.

Logistic Regression(using best regularization of Ridge, c values of 100) produced the highest AUC score(.7576) of any experiment in our study. The held-out test set using this model produced an AUC score on Kaggle of .746.

Overall, the factors improving our Kaggle AUC score in this phase include more refined feature engineering(discussed in Data Lineage) and enhanced experimentation.

Data Lineage/ Data Description

Feature Engineering Automated- Featuretools is an automated tool for generating new features from an existing dataset. First, we will setup our datasets and define the dataframe dictionary and relationships list to help the program understand which datasets we want to create features for and how the datasets are connected to each other. Then we run a deep feature synthesis on the data during which featuretools will cycle through its primitives (different transformations on the data) and it will produce a feature matrix with new useful features based on the algorithm explained here (https://www.jmaxkanter.com/papers/DSAA_DSM_2015.pdf). Our implementation below was somewhat limited by the resource intensive nature of the deep feature synthesis process. It took several attempts to successfully generate features with the tool, the steps that we took to ensure results could be produced withing a few hours processing time were to reduce the number of features generated with the max_feature parameter. Max depth is the number of features that will be stacked together during the feature synthesis process and the default of 2 is what we used to also reduce the complexity and processing time. Another issue with this tool is the potential for data leakage. Based on some tutorials available on the web it seems to be common practice to combine the train and test datasets before doing deep feature synthesis to ensure the column transformations are the same for both train and test. We feel that this process could result in data leakage as the featuretools program may be using both train and test data to make determinations about which features to keep during deep feature synthesis. So, in order to prevent any leakage, we are taking the more challenging approach of separately running the feature transformations on the test set using a feature encoding from the train data which tells feature tools to create a matching set of features for test. After this was done we then took the further step of checking the columns in both data sets and only keeping matching column names for the final data.

Feature Engineering Manually Generated- In phase 3 we ran an experiment where we tested our models based on a training set generated in featuretools and one created manually based on the guidelines of the project to focus on frequency, monetary value and recency. We found that the featuretools data was a bit better in terms of accuracy, but both seemed to work fairly well so for this phase we decided to combine the datasets for our modeling experiments.

Feature Engineering Frequency- For our Manual feature engineering training set we will create the dataset based on the recommendation to focus on data relevant to frequency, recency, and monetary value. The following code will build individual features to explore these aspects of our dataset. The first segment will be features based on frequency. We generated features based on count of active credit, count of days past due from credit bureau reporting, total number of drawings from different sources, and frequency of having the insured on approval flag on previous applications. We also tested the correlation of the features against the target data to determine if there were good indications of effectiveness.

Feature Engineering Monetary Value- Within the Previous Application dataset(previous_application.csv) we created some new features. We summed the total credit amounts (AMT_ANNUITY,

AMT_APPLICATION, AMT CREDIT, AMT_DOWN_PAYMENT) by borrower and grouped by the contract status (Approved and Refused). We created a Loan to Value feature by dividing the amount of credit extended by price of good. We filled NAs with 0 to ensure we don't assign debt to someone without debt.

Within Bureau.csv we created two groups of columns by Active and Closed debts. We totaled the credit amounts (AMT_CREDIT_MAX_OVERDUE, AMT_CREDIT_SUM, AMT_CREDIT_SUM_DEBT, AMT_CREDIT_SUM_LIMIT, AMT_CREDIT_SUM_OVERDUE, AMT_ANNUITY) for each borrower and grouped by Active and Closed debts. We assigned 0 to fill NAs to ensure we don't assign debt to someone without debt.

Within credit card data(credit_card_balance.csv) we summed the monetary features(AMT_BALANCE, AMT_PAYMENT_CURRENT, AMT_PAYMENT_TOTAL_CURRENT, AMT_RECEIVABLE_PRINCIPAL, AMT_RECIVABLE, AMT_TOTAL_RECEIVABLE) by borrower as well as added a Net_Outstanding_Credit_CARD_Balance feature which is calculated as the AMT_TOTAL_RECEIVABLE less AMT_PAYMENT_TOTAL_CURRENT. This new feature shows the credit card balance that is unpaid.

Within installment payments we summed the features by borrower and added a new feature called Balance that is calculated using the AMT_INSTALMENT minus AMT_PAYMENT. This balance feature shows the unpaid amount of the installment.

Feature Engineering Recency- A number of time related features were present in the secondary datasets. Recency or how time relates to a customer or client can sometimes be a helpful indicator in someone's value as a customer. To help determine which time based features would be most helpful in determining a client's creditworthiness, we aggregated time features using transformations such as mean, median, min, max, and sum. Once we had these values, we generated correlation values compared to the target feature. For each time related feature we used the most correlated aggregation as a new feature for our model.

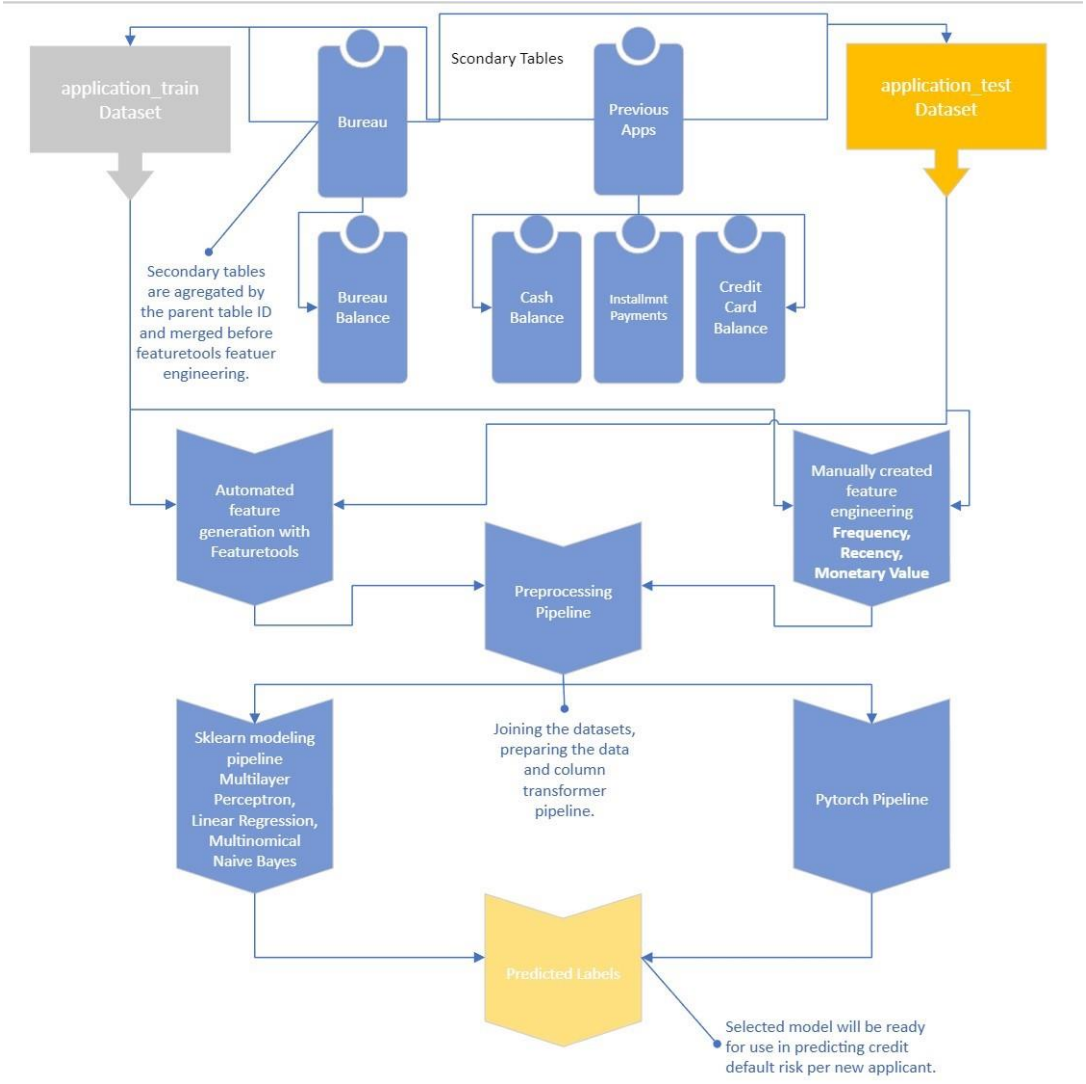
Top Features

The top 5 highest correlated features are DAYS_BIRTH, REGION_RATING_CLIENT_W_CITY, EXT_SOURCE_3, EXTSOURCE_2, EXT_SOURCE_1. Our best features in the select K best gridsearch were SUM(previous.FT.AMT_CREDIT), REGION_RATING_CLIENT_W_CITY and SUM(previous_FT.AMT_APPLICATION). We tweaked our features to replace NAs with zeros(previously)median for numerical features.

From the bureau dataset we created the features mean days credit, mean credit day overdue, total days credit enddate, minimum days enddate fact, and mean days credit update. From the POS Cash Balance

dataset we created minimum months balance, minimum days past due, and maximum days past due with tolerance (low loan amounts ignored). From the credit card balance dataset we created median months balance, maximum days past due, and maximum days past due with tolerance (low loan amounts ignored). From the previous application dataset we created minimum days decision, mean days first drawing, maximum days first due, median days last due first, median days last due last, and median days termination of previous application. From installments payments we created minimum days installment, and minimum days entry payment.

Data Lineage Diagram



Experiment Data Description – With feature engineering completed our next step is to bring our 2 train/test datasets together and set up our feature families and data pipeline. During this process we load

from the CSV files we created earlier (this step is necessary due to resource issues and to protect against kernel crashes that would wipe our data if we tried to do the entire notebook in one session.) Then we will ensure we are only merging unique columns, merge the dataframe, create our set aside test dataframe for generating our Kaggle score later, do a train/test split on our train data, and set up our column transformers and data pipelines.

The next section of the notebook is the model and experiment pipeline which is where we will run our data through a series of experiments with multilayer perceptron, binomial logistic regression, and multinomial naïve bayes algorithms. We will select our best model from these options to submit for scoring on Kaggle.

It looks like our gridsearch found the best parameters to be activation = logistic and solver = sgd. Now we'll use those values to run some experiments. We are going to play with the values of K first, this will determine how many features we will use in the model with the chi squared method as the selector. Our first experiment is k = 200.

With our first 3 experiments we find that as we increase the value of K we have longer processing times and increases in accuracy of less than 1% per 100 K increase. Next, we will do some experimentation with PCA which will reduce the dimensions of our data based on the value that we enter for the component count.

After running several PCA values for our model we see that the best overall AUC score is achieved with 250 components but this was only a small increase over 150 components which in turn was a larger increase over 50 components. For our final experiment with MLP let’s try running without trimming the features or dimension reduction and see how the score does with the full dataset.

Looking at the experiment log for the full dataset we see a slight increase in the score but also a large increase in the processing time. So, this indicates that select K best and PCA can be a useful tool for saving time and getting a reasonable estimate of the model effectiveness. Going forward with our other models we can try using PCA 150 with the grid search and exploring a larger set of parameters hopefully without long waits for processing.

Based on running 2 gridsearches it looks like our best parameters for logistic regression will be a C value of either 100 or 10000, regularization of l2, and solver of lbfgs. Since lbfgs is the default for solver we won't need to include that in the parameter listing. Next, we will test out the parameters and see what our best score is for logistic regression.

After running the logistic regression experiments, we see that our best test AUC score so far is using logistic regression, C 100, and the full training dataset. At this point that is the model to beat for our Kaggle submission. Next, we look at multinomial naïve bayes.

After trying different hyperparameters with multinomial naïve bays it looks clear that we are not getting scores anywhere close to our best models. Based on our experiments our best model is logistic regression with C of 100 and the full dataset. We also got close to that score with multilayer perceptron with the full dataset as well. It looks like we could go ahead and generate Kaggle submission files for both these models and see how we do.

Leakage Analysis

We don't believe our model suffers from leakage. During the generation of features using featuretools we ran into a couple potential leakage issues that we had to look out for. First when we set up our datasets to generate features, we wanted to make sure to remove the target column before doing any transformations of the data to ensure that the target column would stay intact and not influence the feature selection process. Second, we made the determination that the way feature generation was setup in the starter notebook example ran the risk of data leakage since it required the joining of test and train data for feature generation. Luckily there was a new process in featuretools that allowed for the generation of features from a feature encode file generated based on the feature engineering done on the train set. With these measures we felt confident that our automated feature engineering was not at risk of leakage.

Looking further down our pipeline we ensured that the “Application_test” dataset from Kaggle was always held aside from the training data sets. At each merge and feature transformation the test set was set aside and was never mixed in with the train set. We only used the set aside test set for our final scoring and Kaggle submission. Our best performing model accuracy of .746 is also reasonable and not suspiciously high. In this way we believe the test set was always kept safe from influencing the model building process.

Gap Analysis

Our AUC score in the Kaggle submission was .746(731 features). Currently(as of 11:11 PM ET on 12/13), the highest score on the board is .766(actually two scores of .766).

One group(25) used several models(338 features using Logistic Regression, Decision Tree Classifier, Random Forest Classifier, LGBM, XGBoost, Neural Network) and the other(5) used Pytorch MLP with 453 features. The groups with scores close to ours used MLP models. Group 15(MLP) had a score of .749(93 features) while group 1(MLP) had score of .744(536). Of the top scoring teams our team is using the most features.

Tasks to be Tackled

1. Generate test dataset from featuretools and refine the parameters.
2. Determine what other features we want to include from previous feature engineering.
3. Combine all feature engineering datasets into one train/test set.
4. Test aspects of column transformer pipeline (how to use imputer, other ways to define data besides num/cat features.)
5. Examine ways to manage large dataset such as PCA or Select K best.
6. Implement PYtorch with tensor board.

Introduction

As we come to the end of our journey to help identify credit-worthiness for loans by using machine learning, Phase 4 was our opportunity to tighten up our features and models, and experiment with implementation of a neural network. While earlier phases of the project involved EDA, building models, feature engineering, and hyperparameter tuning, our Phase 4 was primarily used examining existing code for errors, getting our existing models performing as well as possible, and using PyTorch to implement a neural network, before landing on our final best model for credit-worthiness prediction.

Feature Engineering and transformers

In Phase 4 we took the opportunity to revisit our feature engineering and transformations to make our model perform better. In Phase 2 we chose a handful of features from secondary tables to aggregate and join to the primary table, which led to Kaggle scores ~50%. In Phase 3 we were more deliberate with the features we engineered, and by using manual creation of recency, frequency, and monetary (RFM) features, and automated feature engineering using FeatureTools. The creation and addition of these features, in addition to hyperparameter tuning, increased our Kaggle scores around 2%.

While our efforts in Phase 3 improved our scores, there was still room for improvement. We felt our methods for feature engineering and hyperparameter tuning were solid, so we investigated the code itself to see if we could spot any mistakes that may have impacted our output. We found a small error in how we were generating our predictions, which we corrected, after which our experiment scores improved, and our Kaggle scores shot up to the 70% range. To further experiment with features, we also introduced feature selection and principal component analysis into our pipelines.

Pipelines

In each phase of the project we built on and adjusted our pipelines. One way we updated pipelines in Phase 4 was the adjustment of imputing methods for numerical pipelines. Previously we used the median as our imputer method for numerical values. As part of our desire to increase our AUC scores, we decided to try changing this to replace NaNs with 0s. Based on our previously low AUC scoring and accuracy scores, we thought using median may have been skewing results in a negative way. Using 0s as our imputer, among other changes, seems to have led to an increase in our model accuracy and AUC scores.

Another way we adjusted pipelines in Phase 4 was the addition of feature selection and principal component analysis to determine the impact of using different subsets of features on the model. Previously after data was sent through numerical and categorical pipelines for processing, the data would be combined into one dataset and fed through our models to get an output (including Gridsearch in some cases). In Phase 4, the data is passed through each model pipeline multiple times in some cases. Some passes used “SelectKBest” to see how scores would change only using the top K features. We tested models with 100, 200, and 300 features. We also tested models using principal component analysis (PCA) to reduce the number of components and see how this type of feature reduction impacted scores. We tested with numbers of components of 50, 150, and 250.

We also built a special pipeline in Phase 4 to allow our data to pass through PyTorch and PyTorch Lightning neural networks. This was primarily to make sure we didn’t try to pass any categorical data through the network and to make sure the data could be converted to tensors as easily as possible by

using regularization such as MinMaxScaler. This was a trial-and-error exercise to figure out what processing was needed to get the data tensor ready.

Experimental results

With the improvements to accuracy and AUC scores from fixing our prediction outputs, and feature selection and component reduction added to pipelines, we decided to test some of our existing models (multilayer perceptron, logistic regression, and multinomial Naive Bayes, all using sklearn) again, in addition to feeding our data through new neural networks using PyTorch and PyTorch Lightning. We conducted a total of 19 experiments.

Sklearn models experiment log:

	Pipeline	RunTime	TrainAcc	TestAcc	ValidationAcc	Train_Auc	Test_Auc	Valid_Auc	Description
0	Baseline MLPClassifier with 731 inputs	426.909412	92.00%	91.94%	91.65%	73.32%	73.80%	73.95%	MLPClassifier pipeline with K = 200
1	Baseline MLPClassifier with 731 inputs	314.393162	92.00%	91.94%	91.64%	72.35%	72.99%	73.23%	MLPClassifier pipeline with K = 100
2	Baseline MLPClassifier with 731 inputs	523.367481	92.00%	91.95%	91.65%	73.73%	74.09%	74.31%	MLPClassifier pipeline with K = 300
3	Baseline MLPClassifier with 731 inputs	213.663062	92.00%	91.94%	91.64%	70.43%	70.98%	71.41%	MLPClassifier pipeline with PCA 50 comp
4	Baseline MLPClassifier with 731 inputs	347.640794	92.00%	91.94%	91.64%	73.98%	74.40%	74.47%	MLPClassifier pipeline with PCA 150 comp
5	Baseline MLPClassifier with 731 inputs	389.107080	92.00%	91.94%	91.65%	74.16%	74.56%	74.62%	MLPClassifier pipeline with PCA of 250 comp
6	Baseline MLPClassifier with 731 inputs	511.415132	92.00%	91.94%	91.65%	74.27%	74.62%	74.74%	MLPClassifier pipeline with full data
7	Baseline LogReg with 731 inputs	44.282214	92.00%	91.93%	91.66%	74.72%	74.84%	74.88%	logistic regression with c:10000 and PCA 150
8	Baseline LogReg with 731 inputs	48.071999	92.00%	91.93%	91.67%	74.71%	74.81%	74.86%	logistic regression with c:100 and PCA 150
9	Baseline LogReg with 731 inputs	35.161997	92.03%	91.94%	91.68%	75.76%	75.46%	75.75%	logistic regression with c:10000 and full set
10	Baseline LogReg with 731 inputs	31.969999	92.03%	91.95%	91.69%	75.71%	75.51%	75.73%	logistic regression with c:10000 and full set
11	Baseline LogReg with 731 inputs	32.520990	92.03%	91.95%	91.69%	75.71%	75.51%	75.73%	logistic regression with c:100 and full set
12	Baseline MultinomialNB with 731 inputs	12.995312	88.54%	89.66%	89.34%	64.24%	64.66%	64.86%	multinomial NB pipeline with alpha 10 K=100 fe...
13	Baseline MultinomialNB with 731 inputs	12.514811	86.26%	87.55%	87.28%	64.77%	65.03%	65.19%	multinomial NB pipeline with alpha 10 and full...
14	Baseline MultinomialNB with 731 inputs	11.588998	85.85%	87.33%	87.03%	64.82%	65.12%	65.25%	multinomial NB pipeline with alpha 1 and full set

PyTorch log:

	Architecture string	Optimizer	Epochs	Test accuracy
0	731-500-2	<class 'torch.optim.adam.Adam'>	10	92.0%
1	731-400-400-2	<class 'torch.optim.sgd.SGD'>	5	91.9%

PyTorch Lightning logs:

```
       Test metric             DataLoader 0
────────────────────────────────────────────────────────
        test_acc           0.9185619950294495
        test_loss          0.24861015379428864
────────────────────────────────────────────────────────
[{'test_loss': 0.24861015379428864, 'test_acc': 0.9185619950294495}]
```

```
       Test metric             DataLoader 0
────────────────────────────────────────────────────────
        test_AUC           0.5131668448448181
        test_loss          0.6282966732978821
────────────────────────────────────────────────────────
[{'test_loss': 0.6282966732978821, 'test_AUC': 0.5131668448448181}]
```

Discussion

Looking at the results of our experiments, we can see some significant changes from previous phases. The accuracies and AUC scores of our models are much higher than the type of results of previous phases. Properly configuring the outputs of our predictions, feature engineering, and hyperparameter tuning all played a part in increasing our scores. While our experiments show promising results in the 70%+ range for AUC scores, looking at the Kaggle and other submissions shows there is still room to increase our scores even further. Feature engineering and further exploration of neural network parameters and optimizers are areas we would explore if we were to continue with the project. Additionally, in an environment where new data was coming in regularly, our models would need to be revisited to see how the ingestion of new data changed things.

Looking at our sklearn models we now have a variety of experiments to better understand how each model performs under different conditions. For our multi-layer perceptron from sklearn, we used the existing hyperparamters we tuned and tested performance with k best feature values of 100, 200, and 300, in addition to testing with principal component analysis (PCA) component values of 50, 150, and 250, and then again with all data. The general trend for this was that while train, test, and validation scores changed little if at all when experimenting, the AUC scores increased when features or component numbers increased, however the amount of time to process the data also increased. Overall, the best performer for multilayer perceptron in sklearn was the one that used all available data with AUC scores all above 74%. This was interesting to note, and good to know that should the case arise where a tradeoff of computing time and accuracy is needed, features/components are a good area to explore.

Logistic regression was our best performer in previous phases, and with the correction of our prediction outputs, this is still the case. In the case of logistic regression, we decided to experiment with different c values for hyperparameters, and with PCA of 150 components vs using the entire data set. When we used PCA with 150 components on the logistic regression model data, comparing results of using c values of 100 vs 10,000 showed little difference. Both models generated almost the exact same test, train, and validation accuracy scores, while AUC scores were only ~0.01-0.03% higher when using the c value of 10,000. For example, for test AUC score our results were 74.88% vs 74.86% for c=10,000 and c=100 respectively. Experiments using all data always had higher AUC scores than experiments using PCA. We saw our highest test and train AUC scores when c=100 and used all data (75.71% and 75.51%), however the validation AUC score was highest when all data was used and c=10,000 (75.75%).

We wanted to see how our changes in Phase 4 also impacted our Multinomial Naïve Bayes (MNB) model which had shown promising results previously. Despite the improvements shown in other models, MNB showed little improvement, with AUC scores still hovering around 65% for train, test, and validation. We experimented with alpha value hyperparameters of 10 and 1 and with k best features of 100 vs using the whole data set. Results showed little change between the experiments.

We ran two experiments using the standard version of PyTorch. The first experiment was implementing the neural network with an architecture of 731-500-relu-2, optim.adam as the optimizer, and ran 10 epochs. This gave us a test accuracy of 92% and test loss of .245. The second experiment was run with an architecture of 731-400-relu-400-2, optim.sgd as the optimizer, and 5 epochs. The result of this experiment was a test accuracy of 91.9% and loss of .279. The results of the neural network with standard PyTorch were promising and proved out that if the project were to continue, it would be worth exploring the numerous parameters we could test for neural networks, including numbers of layers and different optimizers.

In addition to getting a neural network working with PyTorch, we also had the opportunity to try to implement a neural network using PyTorch Lightning. While similar, Lightning implements some features which make it a bit easier to work than standard PyTorch, however, we struggled to implement this due to its relative new-ness and comparative lack of familiarity within the community in regards to troubleshooting and support. In our Lightning model we used the architecture of 723-32-relu-16-2, the optim.adam optimizer, and 10 epochs. We also ran a model with an architecture of 732-relu-40-reli-24-relu-16-2-sigmoid with an optim.adam optimizer. These both returned a model with a validation accuracy score of 91.8% and a loss of .248. The result was similar to the standard PyTorch implementation, however once we were able to get a model running on Lightning, some of the advantages of Lightning became more apparent.

PyTorch in general is great at utilizing GPUs and TPUs to efficiently train models compared to the computing times of models we ran in sklearn. Both the standard PyTorch and Lightning also have the ability to tap in to TensorBoard which is another advantage over sklearn. Tensorboard plugs into PyTorch easily to show us how models are learning and improving in each epoch in an interactive cell, compared to needing to manually generate visualizations using matplotlib and/or seaborn with sklearn.

In our Phase 4 experiments, which were our first attempt to implement neural networks, the AUC scores for our nerual networks were around 50.5%, which indicated that although we had successfully built models, there was still room to improve by trying out different architectures and trying different optimizers. Based on all of our experiment results, our best performing model to predict credit worthiness was our logistic regression model with a Kaggle score of .745.

Conclusion¶

Our challenge was to prove the hypothesis that it was possible to use machine learning models to predict credit-worthiness of people with little to no credit history. We conducted EDA, feature engineering, built pipelines, tuned hyperparameters, and explored machine learning models including neural networks. In Phase 4, we tied everything together by ensuring predictions were generated as efficiently as possible, experimented on models, and determined our best model to be logistic regression (regularization of Ridge, c values of 100) after comparing train, test, and validation scores, as well as AUC scores. After submitting this model's predictions to Kaggle, we ended with a final submission score of .745. We tested significance and found Logistic Regression was statistically significant relative to our second-best model(MLP). This proves our hypothesis that machine learning can be used to predict credit-worthiness. In the future, there is still room to explore additional feature engineering and PyTorch parameters.

Credit Assignment Plan

FP_GroupN_ 7- HCDR Project

Group Members	Color Code For Task Responsibility:
Michael Casey	
Andrew Cruez	
Peter Stewart	
Hemraj Yadav	

Phase	Description	PHASE ONE	PHASE TWO	PHASE THREE	PHASE FOUR	
		Group Member	Group Member	Group Member	Group Member	
3	Tasks					Hours Est
4	Tasks					Hours Est
4.1	Video Presentation					
4.1.1	Create Slide Deck					2
4.1.2.1	a title slide (with the project name, Group Number, the team member names, and photos)					
4.1.2.2	outline slide with good descriptive section headings					
4.1.2.3	Team names, photos, Project description					
4.1.2.4	Visual EDA					
4.1.2.5	Modeling Pipelines Explored					
4.1.2.6	Results and discussion of results					
4.1.2.5	Conclusion and next steps					
4.2	Team and project meta information					2
4.2.1	Phase leadership plan					1
4.2.2	Credit Assignement plan					1
4.3	Phase 3 Abstract (150 words)					2
4.3.1	Write Abstract					1.5
4.3.2	Proofread Abstract					0.5
4.4	Project Description					3
4.4.1	Data Description					
4.4.2	Tasks to be tackled					
4.4.3	Provide diagrams to aid understanding the workflow					
4.5	Neural Network					3
4.5.1	Implement Neural Network Model					
4.5.1.1	Research and attempt pytorch Lightning					
4.5.1.2	Research pytorch based on vanilla approach					
4.5.2	Experiment with 2 Network Archetectures					
4.5.3	Report NN archetecture in string form					
4.5.4	Finalize pytorch with tensor board					
4.6	Data Leakage					3
4.6.1	Write Data Leakage report					
4.7	Modeling Pipelines					3
4.7.1	Visualization of Modeling Pipeline					
4.7.2	Families of input features					
4.7.3	Number of input features					
4.7.4	Hyperparameters and settings considered					
4.7.5	Loss function used (data loss and regularization parts) in latex					
4.7.6	Number of experiments conducted					
4.7.7	Experiment table with the following details per experiment:					
4.7.7.1	Baseline experiment					
4.7.7.2	Any additional experiments					
4.7.7.3	Final model tuned					
4.7.7.4	best results (1 to three) for all experiments you conducted with the following details					
4.7.7.5	The families of input features used					
4.7.7.6	For train/valid/test record results.					
4.8	Results and discussion of results					3
4.8.1	Written Results/ Discussion					
4.8.2	Proofread Results/ Discussion					
4.9	Conclusion					2
4.9.1	Written Results/ Discussion					
4.9.2	Proofread Results/ Discussion					