Results and Analysis from Credit Card Loans Default Project:

Abstract:
This project aims to predict how capable each client is of paying off their credit card balances. The dataset represents default and non-default accounts of credit card clients in Taiwan from 2005. Using this historical data, I will try to build a predictive model that classifies whether an account will pay off its next month's balance (0) or default (1).

Results:

Decision Tree:

We begin by tuning the model with GridSearch, and collecting the best parameters that result in the highest score on the validation dataset. Because GridSearch is computationally expensive, we record the best parameters below:

Best parameters: {'max_depth': 25, 'min_samples_leaf': 1, 'min_samples_split': 2}
Best score on training dataset: 0.7605353575061273

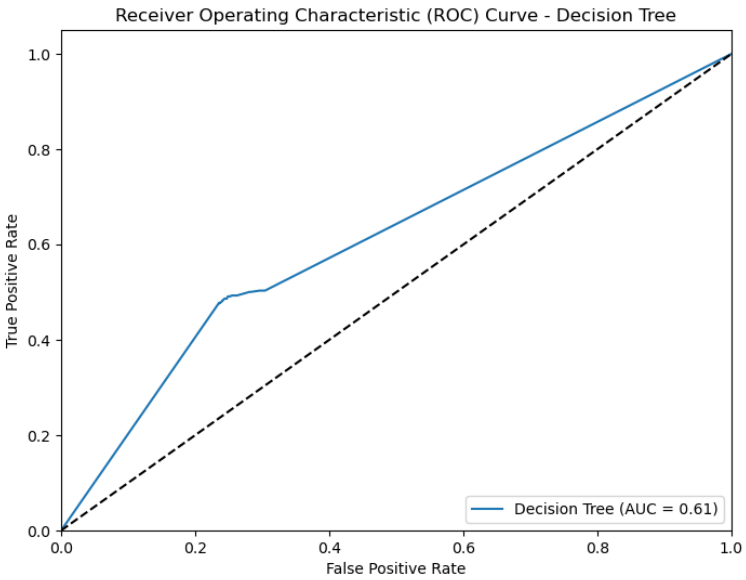
We then print out a classification report and confusion matrix with the tuned model on the validation dataset:

Tuned Model Results on Validation Dataset:

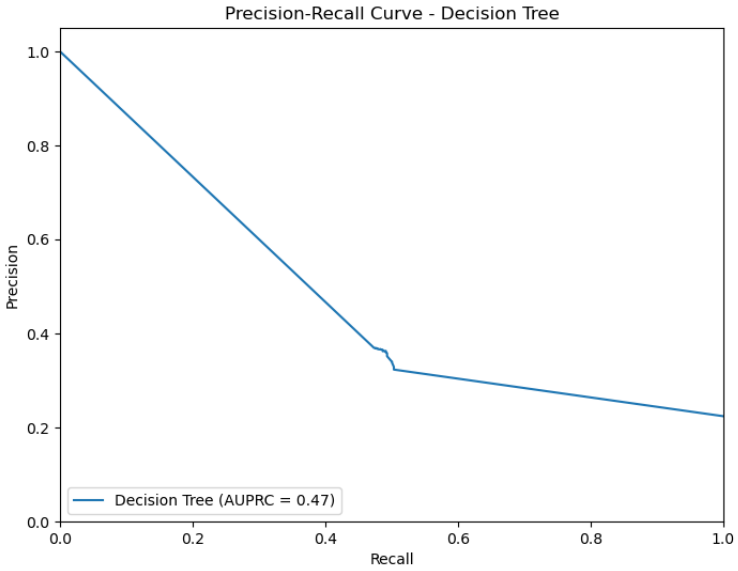
	precision	recall	f1-score	support
0	0.84	0.76	0.80	3725
1	0.37	0.48	0.42	1075
accuracy			0.70	4800
macro avg	0.60	0.62	0.61	4800
weighted avg	0.73	0.70	0.71	4800

AUC: 0.6203308880911502
[[2830 895]
[558 517]]

We continue evaluating this tuned model with the ROC curve and Precision-Recall curve. We also calculate the best threshold to find out the ideal "cut-off" point between each class by predicting probabilities of the instances and analyzing the results on the ROC curve and precision-recall curve. We want a threshold that maximizes precision and recall (which have an inverse relationship).



Note that the AUC is different here versus above, despite being the same model on the same (validation) dataset, because the above AUC is for binary classification at one threshold. In contrast, because we use the predicted probabilities to create this ROC, we are able to see the performance across different thresholds, and hence, see a greater ROC.



Best threshold: 0.47368421052631576

The best threshold represents the best probability for the "cut-off" between a 1 (default class) and 0 (non-default class). The default threshold is 0.5, and we see that our best threshold (calculated based on the ROC curve using a J Statistic in our program), is very close to 0.5.

We observe good performance (relative to default model performance on the dataset – not recorded here), but, we observe that the ideal hyperparameters (recorded above as "best parameters") are reminiscent of a model that is overfitting. For this reason, when we evaluate against the test dataset, we will also include the default model for comparison.

Results on Test Dataset with Tuned Decision Tree Model:

	precision	recall	f1-score	support
0	0.83	0.76	0.79	4687
1	0.34	0.45	0.39	1313
accuracy			0.69	6000
macro avg	0.59	0.60	0.59	6000
weighted avg	0.72	0.69	0.71	6000

AUC: 0.6037952197510866

Results on Test Dataset with Default Decision Tree Model:

	precision	recall	f1-score	support
0	0.83	0.76	0.79	4687
1	0.35	0.46	0.40	1313
accuracy			0.69	6000
macro avg	0.59	0.61	0.60	6000
weighted avg	0.73	0.69	0.71	6000

AUC: 0.6098571001673538

We see that our tuned model was indeed overfitting on the validation dataset, but only slightly. The difference in performance between the two models on the test dataset is basically negligible: we can reasonably conclude that both models will perform about the same in production.

Gradient Boosting Machine

Instead of using the computationally expensive process of GridSearch for the GBM, we understand that setting a smaller learning rate (learning_rate) in combination with a high number of trees (n_estimators) is a common strategy to achieve better generalization results.

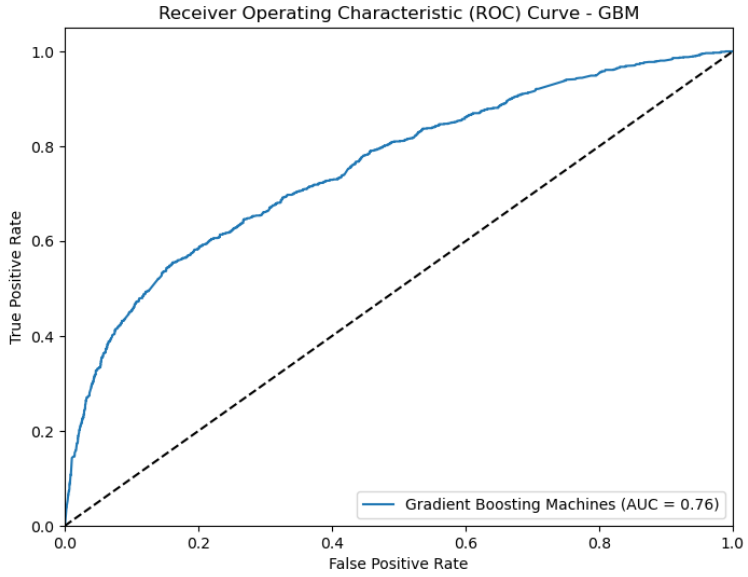
We choose the following parameters:
(max_depth = 3, n_estimators = 200, learning_rate = 0.01, random_state=42)

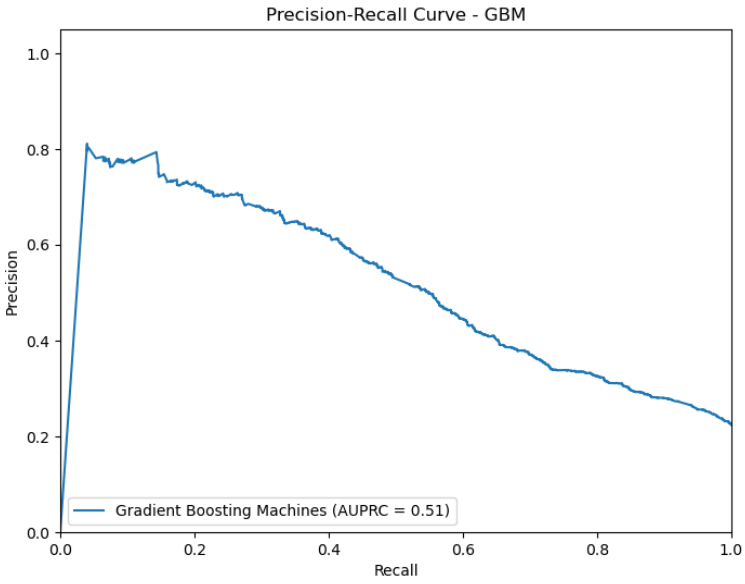
Results of Tuned Model on Validation Dataset:

	precision	recall	f1-score	support
0	0.87	0.84	0.85	3725
1	0.50	0.55	0.52	1075
accuracy			0.78	4800
macro avg	0.68	0.70	0.69	4800
weighted avg	0.78	0.78	0.78	4800

AUC = 0.6955454971125332

When we plot our AUC to account for predicted probabilities, we get an AUC = 0.76. As shown below by the ROC Curve:





The strange behavior at the start of the curve is a result of imbalanced classes, where because there are sparse positive examples, the model is not predicting positive classes. The curve normalizes later on. We will see this problem of not predicting positive classes later on with neural networks.

I will use Youden's J statistic to seek a balance between sensitivity and specificity (just as with Decision Trees):
Best threshold: 0.4983265685882021
We can conclude that 0.5 is the ideal threshold.

Results on Test Dataset with Tuned GBM Model:

	precision	recall	f1-score	support
0	0.87	0.83	0.85	4687
1	0.48	0.56	0.52	1313
accuracy			0.77	6000
macro avg	0.68	0.70	0.69	6000
weighted avg	0.79	0.77	0.78	6000

AUC: 0.6959836568909061

We see that our results generalized relatively well with the GBM.

Logistic Regression

In Logistic Regression, instead of using GridSearch or RandomSearch for hyperparameter tuning, I create various models and store their performance and hyperparameters in a dictionary.

Report 1 for parameters {'random_state': 42}:

	precision	recall	f1-score	support
0	0.83	0.73	0.78	3725
1	0.34	0.48	0.40	1075
accuracy			0.68	4800
macro avg	0.59	0.61	0.59	4800
weighted avg	0.72	0.68	0.69	4800

Report 2 for parameters {'penalty': 'l1', 'C': 1.0, 'solver': 'liblinear', 'random_state': 42}:

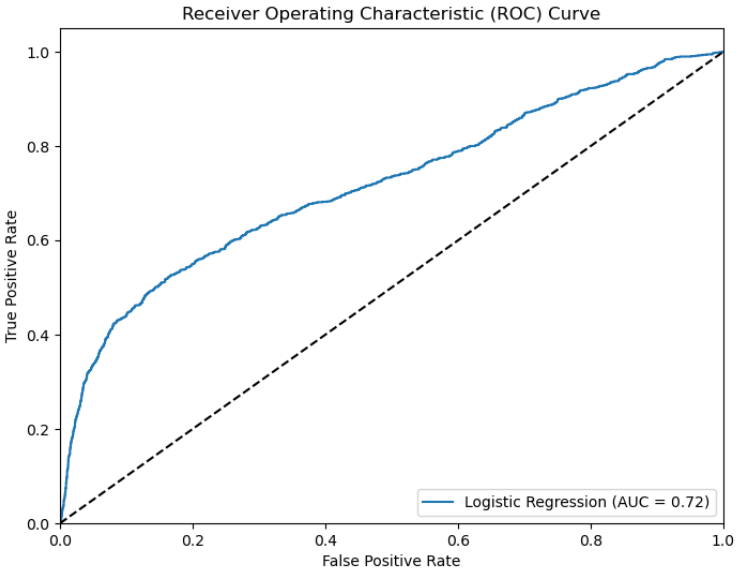
	precision	recall	f1-score	support
0	0.87	0.69	0.77	3725
1	0.37	0.63	0.47	1075
accuracy			0.68	4800
macro avg	0.62	0.66	0.62	4800
weighted avg	0.76	0.68	0.70	4800

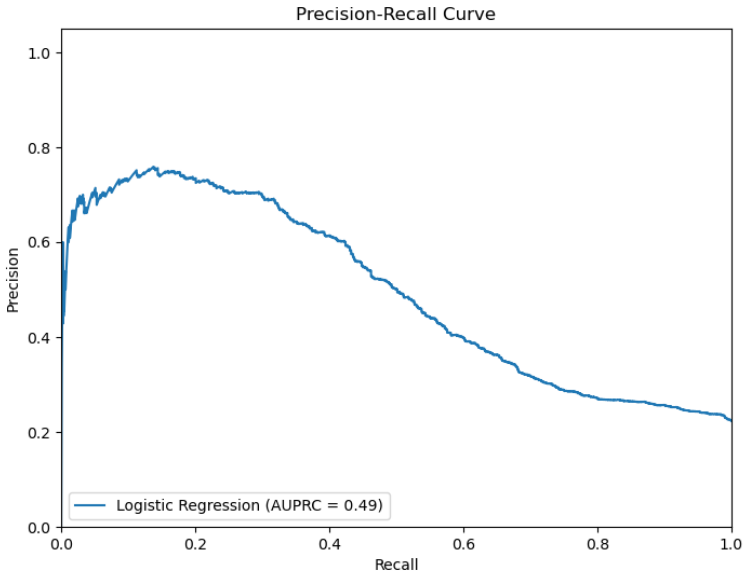
Report 3 for parameters {'penalty': 'l2', 'C': 10, 'solver': 'lbfgs', 'random_state': 42}:

	precision	recall	f1-score	support
0	0.83	0.72	0.77	3725
1	0.34	0.50	0.41	1075
accuracy			0.67	4800
macro avg	0.59	0.61	0.59	4800
weighted avg	0.72	0.67	0.69	4800

Based on these reports, we will use model 2 for testing

We then do the same as GBM and Decision Tress: get probabilities to build a proper ROC curve across varying thresholds. I also include a Precision / Recall curve for this model.





Best threshold: 0.5455269353696268

Results on Test Dataset with Tuned Logistic Regression Model::

	precision	recall	f1-score	support
0	0.87	0.69	0.77	4687

1	0.37	0.65	0.47	1313
accuracy			0.68	6000
macro avg	0.62	0.67	0.62	6000
weighted avg	0.76	0.68	0.70	6000

AUC: 0.6656203876776051

We see that model 2 did, in fact, generalize pretty well on the test dataset – performing identically in F-1 score on "1" (default accounts) as the validation set, and out-performing models 1 and 3 on their validation sets.

Random Forest:

When running GridSearch for our random forest model, it seemed that our model may have overfit the hyperparameters.

The hyperparamters were:
'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 150

Because these hyperparameters are characteristic of an overfitted model, we will compare these results versus the default parameters. Below, Accuracy Score 1 and Val Report 1 represent tuned parameters, and Accuracy Score 2 and Val Report 2 represent default parameters.

Accuracy Score 1: 0.793125
Accuracy Score 2: 0.7925
Val Report 1 classification:

	precision	recall	f1-score	support
0	0.85	0.89	0.87	3725
1	0.55	0.45	0.50	1075
accuracy			0.79	4800
macro avg	0.70	0.67	0.68	4800
weighted avg	0.78	0.79	0.79	4800

AUC: 0.6721492117995942
Val Report 2 classification:

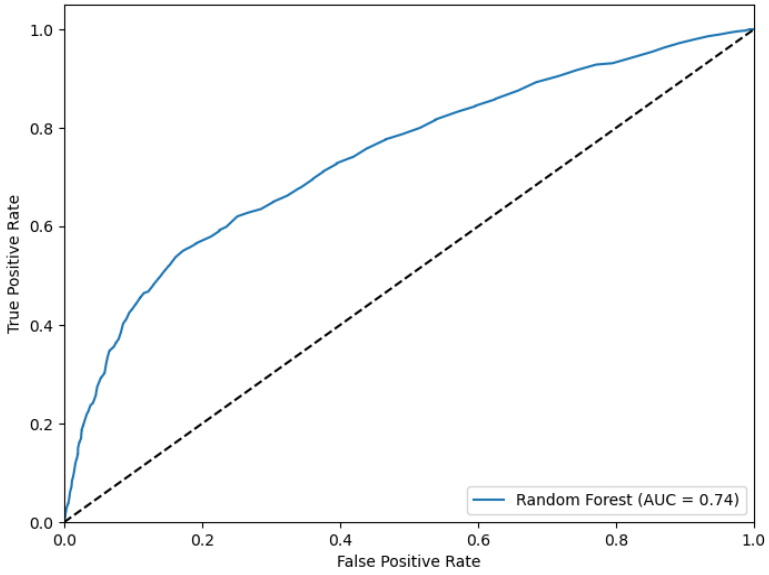
	precision	recall	f1-score	support
0	0.85	0.89	0.87	3725

1	0.54	0.46	0.50	1075
accuracy			0.79	4800
macro avg	0.70	0.67	0.68	4800
weighted avg	0.78	0.79	0.79	4800

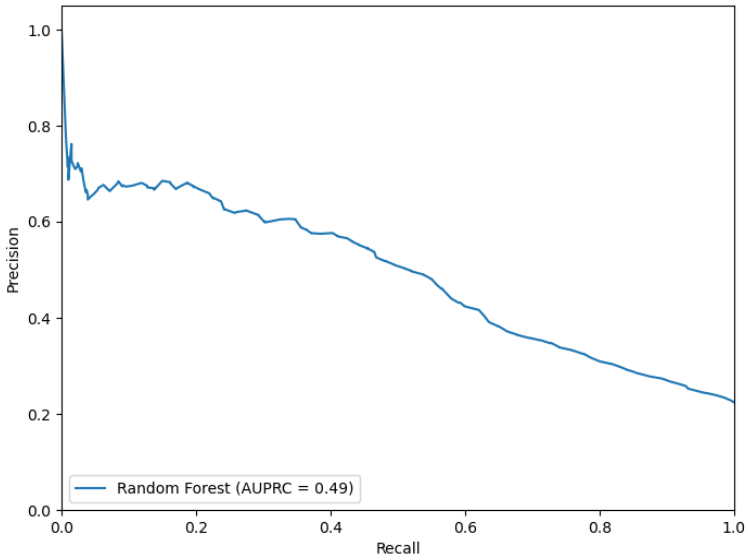
AUC: 0.6727391915092866

The results are nearly identical. So, our model did not necessarily overfit during tuning, but it did not result in better performance, either. Thus, I will use the default parameters.

ROC CURVE:



PRECISION / RECALL CURVE:



Best threshold: 0.43

Results on Test Dataset with Default Random Forest Model:

	precision	recall	f1-score	support
0	0.85	0.88	0.87	4687
1	0.51	0.45	0.48	1313
accuracy			0.79	6000
macro avg	0.68	0.66	0.67	6000
weighted avg	0.78	0.79	0.78	6000

AUC: 0.6649825130877631

SVM:

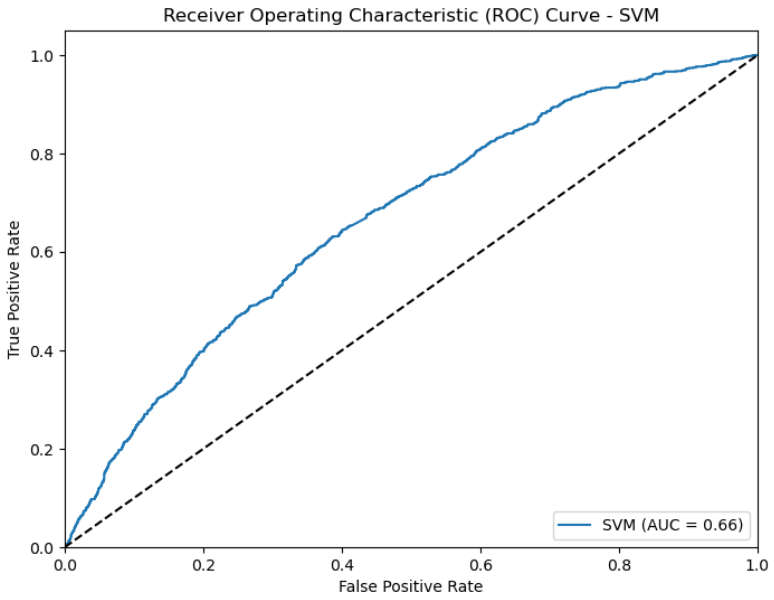
For SVM, I will once again manually play around with a couple combinations of parameters and consider “best practices” – such as not over-tuning - to avoid overfitting. I ended up with the following:

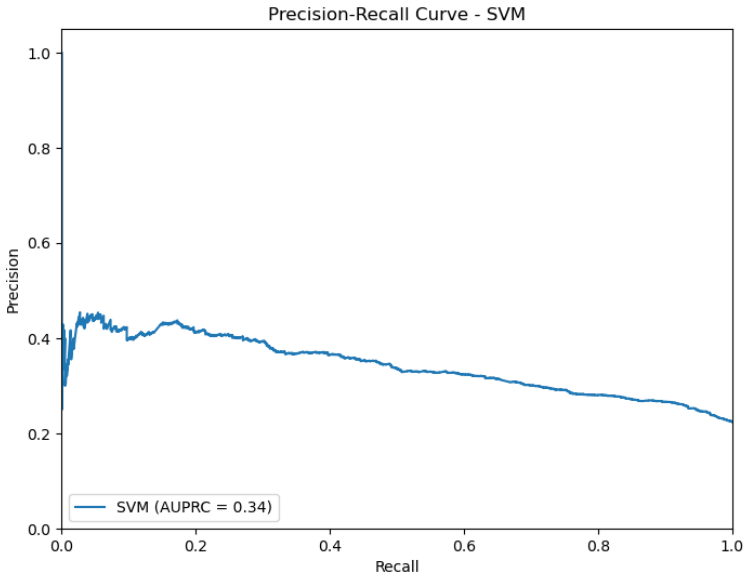
C=0.1,
kernel='rbf',
gamma='scale',
class_weight=None,
probability=True

Results on Validation Set:

	precision	recall	f1-score	support
0	0.86	0.56	0.68	3725
1	0.31	0.68	0.42	1075
accuracy			0.59	4800
macro avg	0.58	0.62	0.55	4800
weighted avg	0.73	0.59	0.62	4800

AUC: 0.6187295145934134





Best threshold: 0.5289013289127779

Results on Test Dataset with Tuned SVM Model:

	precision	recall	f1-score	support
0	0.86	0.55	0.67	4687
1	0.30	0.67	0.41	1313
accuracy			0.58	6000
macro avg	0.58	0.61	0.54	6000
weighted avg	0.73	0.58	0.61	6000

AUC: 0.6116807016409245

Neural Network:

I began with using Keras Tuner for the Neural Network. While this led to very high accuracy, the reality was that the model was simply "hacking" its way to a high accuracy score by simply predicting every instance as a negative class. This is not useful to us.

Trial 5 Complete
val_accuracy: 0.7759722272555033
Best val_accuracy So Far: 0.7760416666666666

	precision	recall	f1-score	support
0	0.78	1.00	0.87	3725
1	1.00	0.00	0.00	1075
accuracy			0.78	4800
macro avg	0.89	0.50	0.44	4800
weighted avg	0.83	0.78	0.68	4800

The above implies our Neural Network is predicting every instance as a negative class. This is not what we want. I will go back and change the hyperparameters of the neural network.

Performance on Validation Dataset:

	precision	recall	f1-score	support
0	0.77	0.00	0.01	3725
1	0.22	1.00	0.37	1075
accuracy			0.23	4800
macro avg	0.50	0.50	0.19	4800
weighted avg	0.65	0.23	0.09	4800

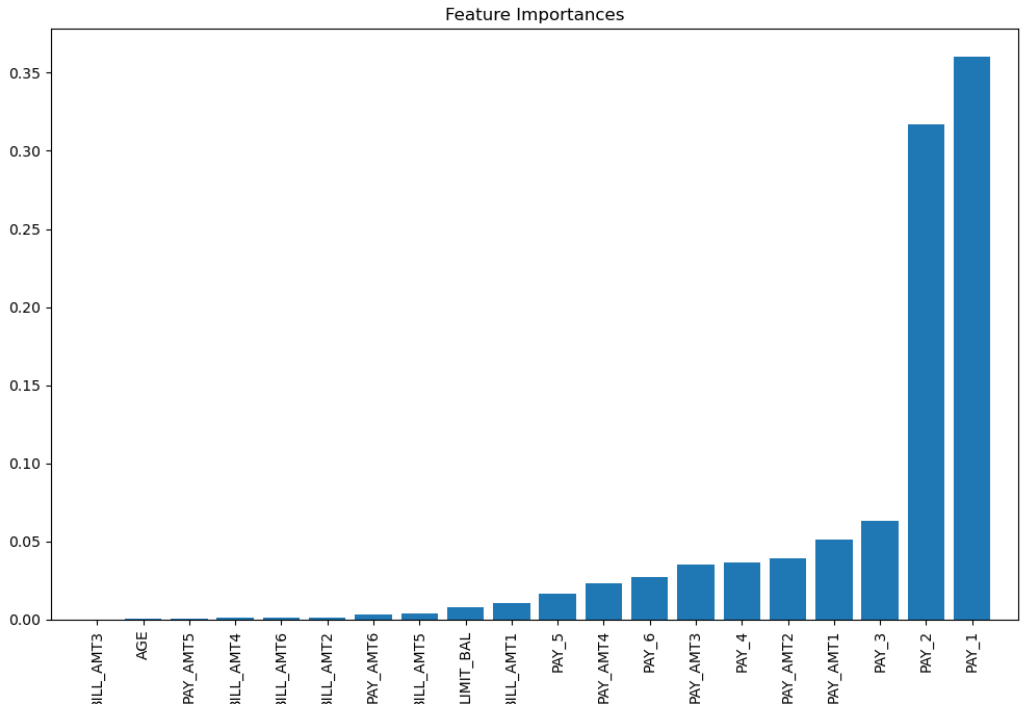
AUC: 0.499956297799282
[[17 3708]
[5 1070]]
(4778, 22)
Here, we see the model predict (almost) every instance as a positive class – the opposite of what it was doing before.

Despite best efforts to create a working Neural Network, I'll conclude that this type of model is simply too complicated for this problem and dataset. It results in overfitted models that simply predict one class or the other. Thus, I will not continue evaluating the ROC curve with this model.

Conclusion:
Unfortunately, in a production environment, none of these models would be good enough to be put into production. However, the value in these models is not lost. In a business setting, it could be valuable to look at the feature importances that the model used in making its predictions –

uncovering potential valuable insights. For example, if the model is placing a lot of weight on a particular feature, it is worth digging into whether the model is simply wrong or if it is in fact onto something interesting.

Here is a feature importances graph on GBM:



Digging into this might be useful for our human teams that are responsible for analyzing and predicting defaults, as they might be able to uncover a pattern or metric that wasn't necessarily obvious before.

Below are the rankings of the different models. I will use the F-1 score on the *test* dataset for these rankings. We will use the harmonic mean of precision and recall because, depending on our business needs/objective (discussed further in the "takeaways" below), we might want to change our threshold and optimize our model for either precision or recall. Additionally, in an imbalanced dataset, the F-1 score is not sensitive to the number of true negatives: making it a better metric when the focus is on predicting default cases.

1. GBM
2. Random Forest
3. Logistic Regression
4. SVM
5. Decision Tree
6. Neural Network

These results suggest that ensemble learning methods (GBM and Random Forest) perform the best on this dataset.

Below I concluded some takeaways I took from this personal project:

Takeaways:

- While most of our best thresholds were at or very close to 0.5, in practice, this might not be the best option. Even though 0.5 basically gives us the best metric between sensitivity and specificity, it might be smarter in our case, given our model's performance and the business context, to decrease our threshold. This will increase our model's recall. Increasing the recall of our model will allow us to not 'miss' any default accounts because the threshold for a positive account (or a default account) is lower. The trade-off to this, however, is that our model will classify more non-default accounts as default accounts. However, we can carefully review accounts classified as default before actually making any consequential decision, and the model saves the team from the work of having to review surely non-default accounts.
- Through this project, I noticed firsthand that hyperparameter tuning generally leads to overfitting or negligible results, unless it is for the purposes of picking a different precision vs. recall trade-off. In other words, it did not increase performance on new data: it merely overfitted the training and validation data. Going forward, I will likely pick "best practice" parameters for these models instead of using GridSearch or manually trying different combinations of parameters.
- Complicated is not always better: A Neural Network was the worst-performing model on this dataset.

Thanks for taking the time to check out my project!
Andrew Dabinett