Final Report

Team Parental Disappointments

Samuel Zhang, Brian Chen, Eric Chen, Samuel Chu, Kai Breese

**Problem Statement [5pts]**

*1. Formal statement of your problem and model:*

Credit score is the primary way that banks and other financial institutions determine whether or not someone is loan-worthy. However, there are people who are loan-worthy that do not have a credit score. Being able to identify such people would provide a huge source of business for banks.

Subsequently, based on transaction history, the goal of this project is to use an individual’s transaction history to predict whether or not they will default on their loan. The models we ended up using for this project mainly involved a high depth decision tree ensemble and a low depth decision tree ensemble.

*2. Explain dataset and classification problem being solved*

The dataset we are provided is from Prism Data, and it is a collection of time-series data for a set of customers. There is one file containing every single transaction, most notably describing when the transaction took place, the category of the transaction, the amount of the transaction (pos/neg), and the customer id linked to the transaction. There is another file provided containing customer information such as customer id, the final balance, and whether the customer defaulted or not. Lastly, there is a file which maps the numerical transaction categories to strings, allowing viewers to understand what types of transactions they were.

This is a binary classification problem where the goal is to predict whether or not someone will default on their loan based on their transaction history.

*3. Explain the performance metric*

The performance metric that we are using is the AUC-ROC score, where the higher the score, the better. Unlike just pure accuracy, the AUC-ROC score is a good metric for imbalanced datasets, which is what we have in this case (most customers did not default). The AUC-ROC score is the area under the curve of the receiver operating characteristic curve, which is a plot of the true positive rate against the false positive rate. This makes it a good metric for binary classification problems.

*4. importance and relevance of the problem*

The importance of this problem is that it can provide another source of business for banks. If they can identify loan-worthy individuals among those who do not have a credit score, they can provide more loans, making more money. This can also help people who do not have a credit score build up their credit.

**Proposed Approach [5pts]**

Proposed Approach [5pts]

*1. A list of potential strategies and a justification of your approach*

The approaches that we considered more or less all revolved around summarizing the time series with summary statistics on the transaction amount. A large portion of our time was spent on feature engineering these summary statistics, and starting off with a small core set of summary stats (mean, sum, std, min, max) over all transactions for an individual, we then expanded to include not only more types of summary statistics (skew, kurtosis, autocorrelation, zero-crossings, various ratio features, etc.) along with groupbys on the transaction categories and seasonality.

Aside from feature engineering, the main models that we considered were decision tree ensembles, and logistic regression.

*2. What approaches or models are commonly used to solve similar problems*

The approaches that are typically used to deal with variable length time series are quite varied. However, the simplest approach (which we ended up using) is just based upon extracting a fixed set of summary statistics, effectively reducing the multidimensional variable length time series data into "tabular data".

However, something to keep in mind is that this feature representation of summary statistics is almost certainly missing key aspects of the problem. Something like a NN, can almost certainly learn features that are more meaningful that the ones we tried extracting.

On this note, we were also looking for methods that involved working directly with the time-series to prevent information loss, however, the variable nature of the time series ended up being difficult to work with. Most of the time-series architectures we found do not handle the "variable length" aspect well and instead require a "fixed-length" time series. Furthermore, something to note is that the problem is not just using variable length time series (one value), but multidimensional variable length time series (amount, transaction category, date).

A library we tried exploring was pyts, which is a popular library for time series classification, but we were unable to find anything in it that could handle multidimensional variable length time series classification.

*3. How were these approaches adapted for your specific problem*

As mentioned in the first section, we adapted these approaches by summarizing the time series with summary statistics on the transaction amount while performing extra groupbys on seasonality and transaction category (multi-layered index). Then we used these summary statistics as a kind of “tabular dataset” for our models.

**Key Findings in Model Development [5pts]**

*1. A list of what you tried, why you tried it, and whether they worked with evidence.*

*2. Description of how your model(s) evolved over the course of development*

*3. Justifications for decisions made at each stage in development*

The first thing we tried was using a small set of summary statistics (mean, sum, std, min, max) based upon all of the transaction data for a given customer. There was no granularity for things such as transaction category or season. We opted to start without groupbys so that we could try to determine how effective each added feature was. The groupbys end up multiplying the feature count by ~30 for the transaction categories and 4 for the seasons. This is effectively a ~120x'ing of each newly added feature, which would often make it difficult to determine the effectiveness of a newly added feature.

Below are some plots that show how performance changed with a feature set that gradually increased in complexity (model kept fixed):

MODEL:

ensemble = HistGradientBoostingClassifier(

learning\_rate=0.025,

max\_iter=500,

max\_leaf\_nodes=16,

max\_depth=4,

min\_samples\_leaf=25,

l2\_regularization=1,

max\_features=0.5,

max\_bins=25,

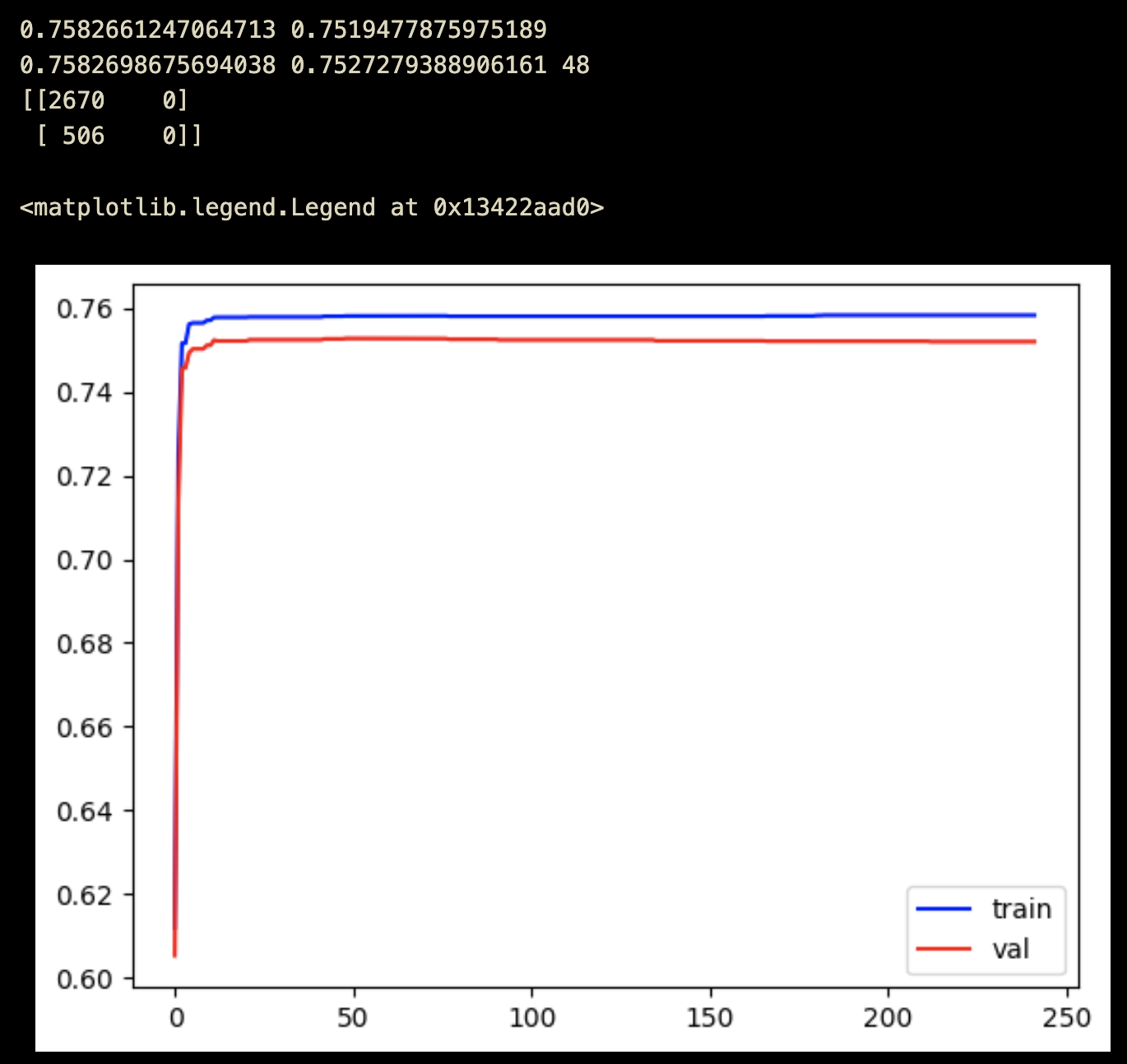
validation\_fraction=0.20,

early\_stopping=True,

random\_state=16,

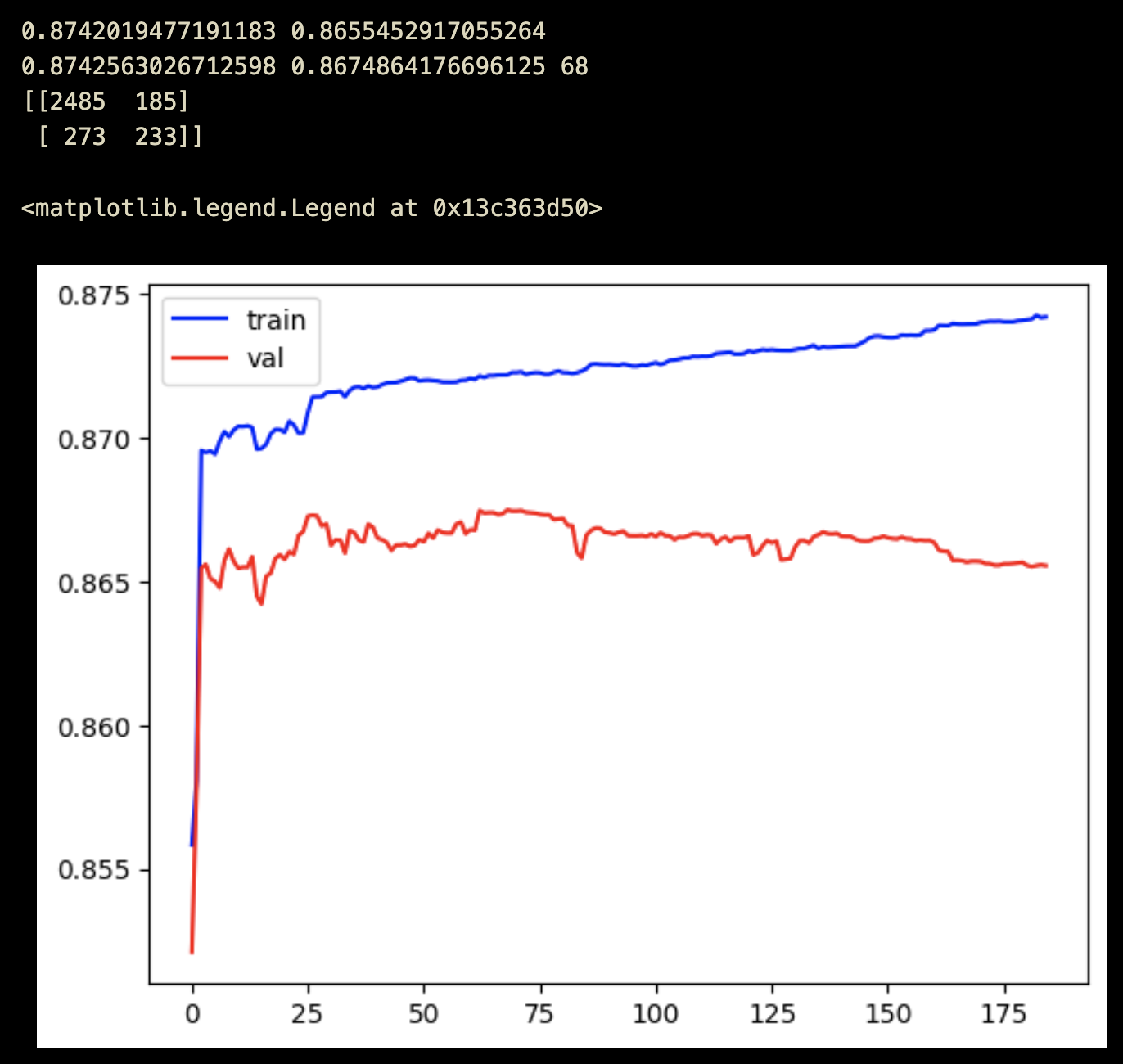
)

'went\_negative', ‘min\_max\_ratio’



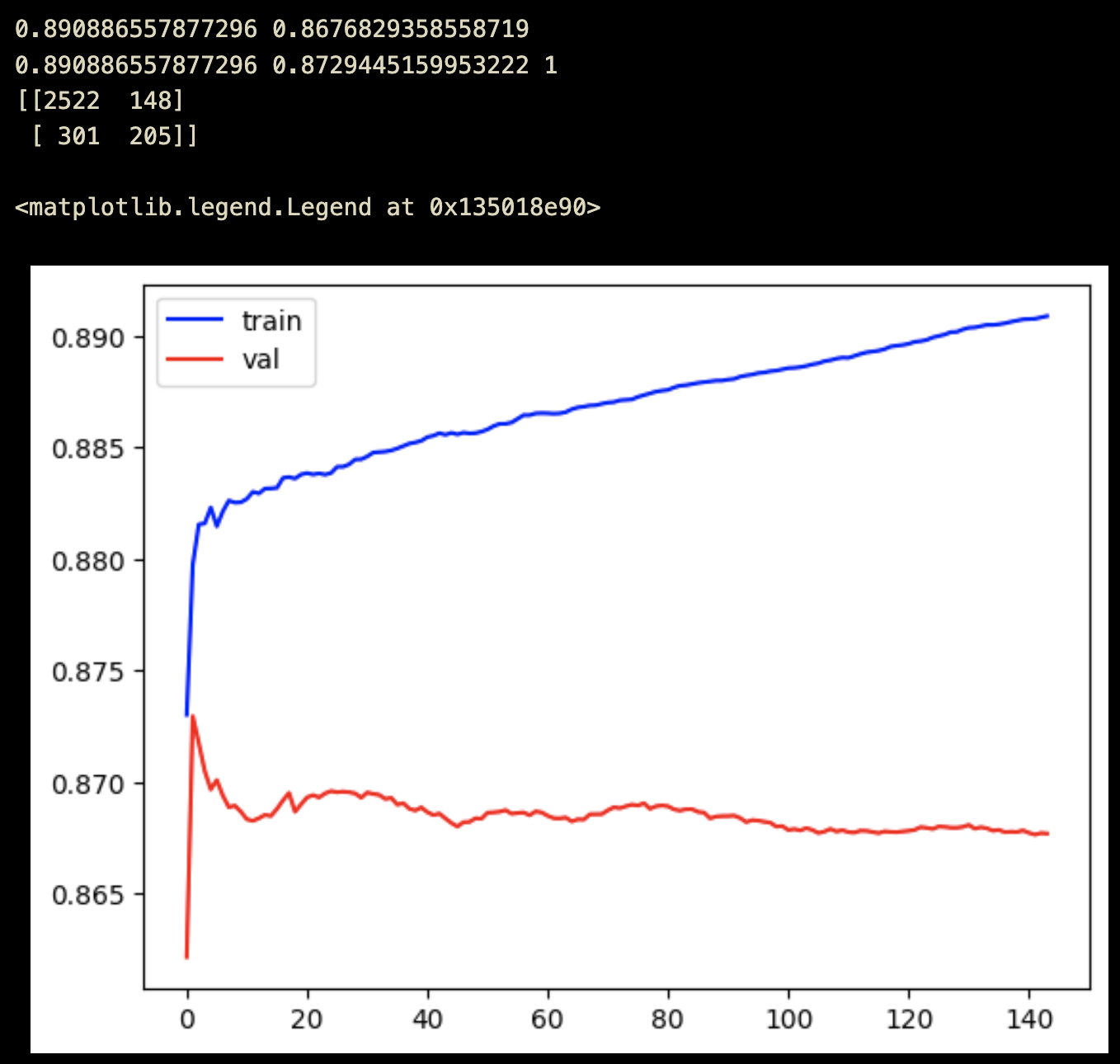
The most important feature within the dataset was the prefix of the customer id, which effectively described the type of loan being taken out. We were surprised to see just how impactful this feature was, as after adding it in, there was a massive performance improvement. The val AUC shot up from being around 0.76 to 0.86. We did not see any AUC gain higher than this throughout the rest of the project.

"went\_negative", "min\_max\_ratio", "prefix"



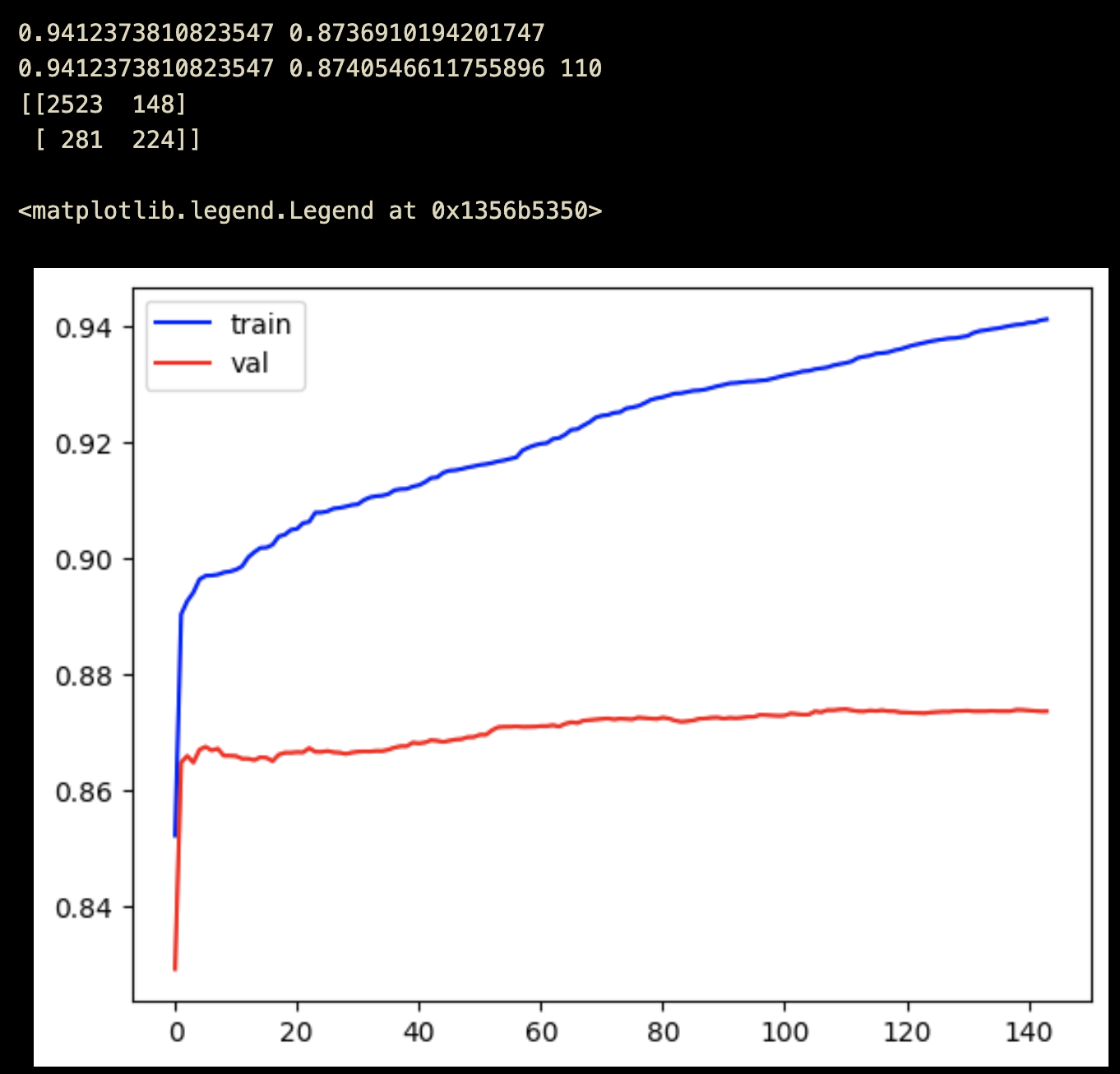
Something important which we observed early on was that adding in features that were highly correlated, led to the model to overfit to noise. Though something to note, is that adding in these features still improved the peak performance. This was expected as despite the introduction of greater noise, there still was still often slightly different information provided through each feature, which the model could leverage.

'min', 'max', 'first', 'last', 'went\_negative', 'first\_last\_ratio', 'min\_max\_ratio', 'prefix'



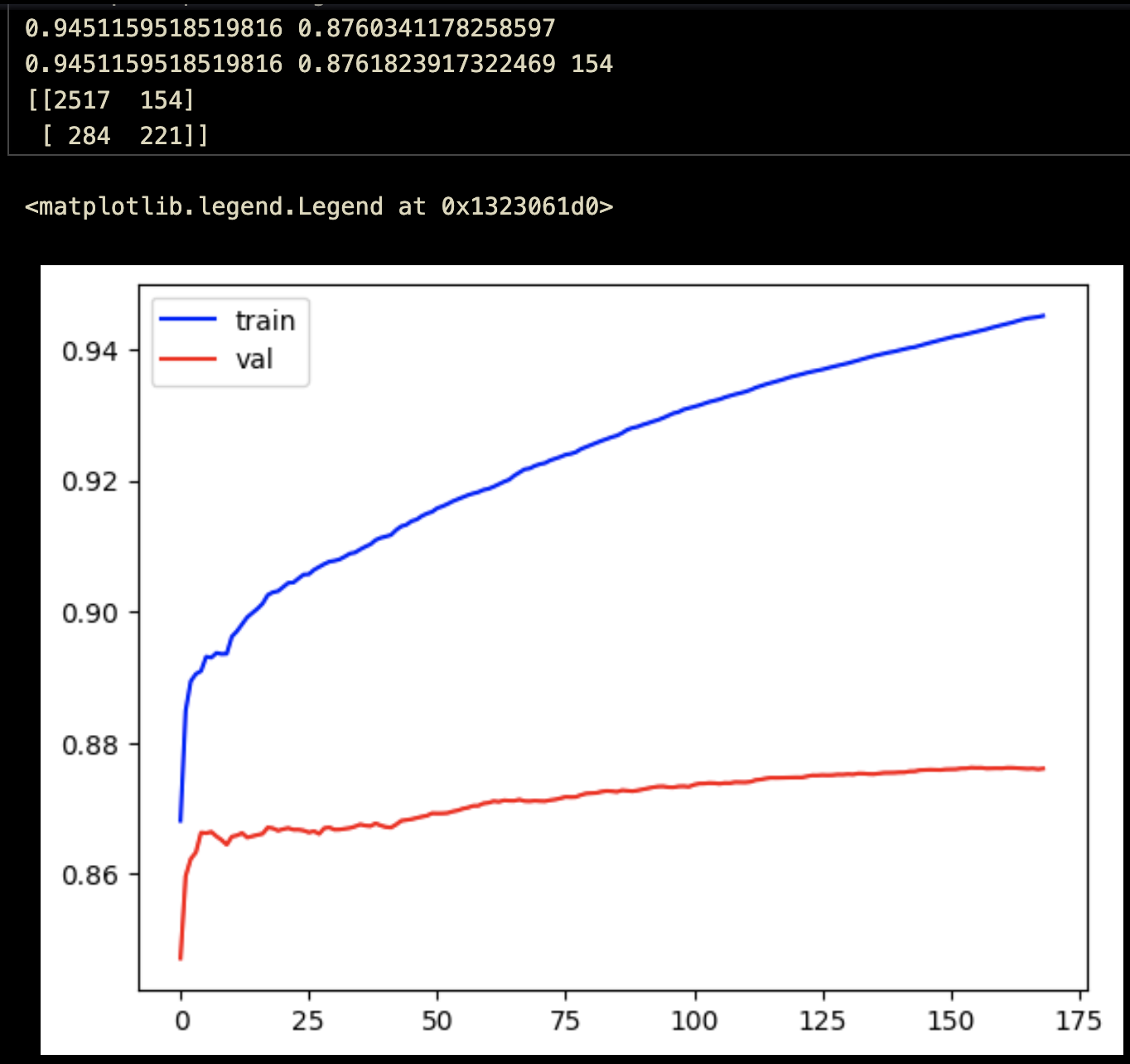
With the slightly improving performance in mind, we figured that adding in more summary statistic features would still continue to improve the performance. This was the case, however, it was clear that there were diminishing returns as the performance only changed from 0.872 -> 0.874 AUC.

'min', 'max', 'first', 'last', 'sum', 'mean', 'count', 'std', 'skew', 'kurtosis', 'autocorr', 'median', 'iqr', 'batdiff\_std', 'went\_negative', 'first\_last\_ratio', 'min\_max\_ratio', 'negative\_positive\_ratio', 'prefix'



At this point, we figured that since the transaction amount only describes the “change” and not “raw amount”, incorporating the total balance information would likely give the model more context to work around. If there’s a 1000 dollar withdrawal, the context and meaning is drastically different if the total balance left after the transaction is 1000000 vs -1000. With this intuition in mind, we calculated said balances and then also computed the same summary statistics off of this information. Effectively, the feature set size just doubled (excluding prefix).

Dataset Shape: (15879, 35) 17 \* 2 + 1 = 35



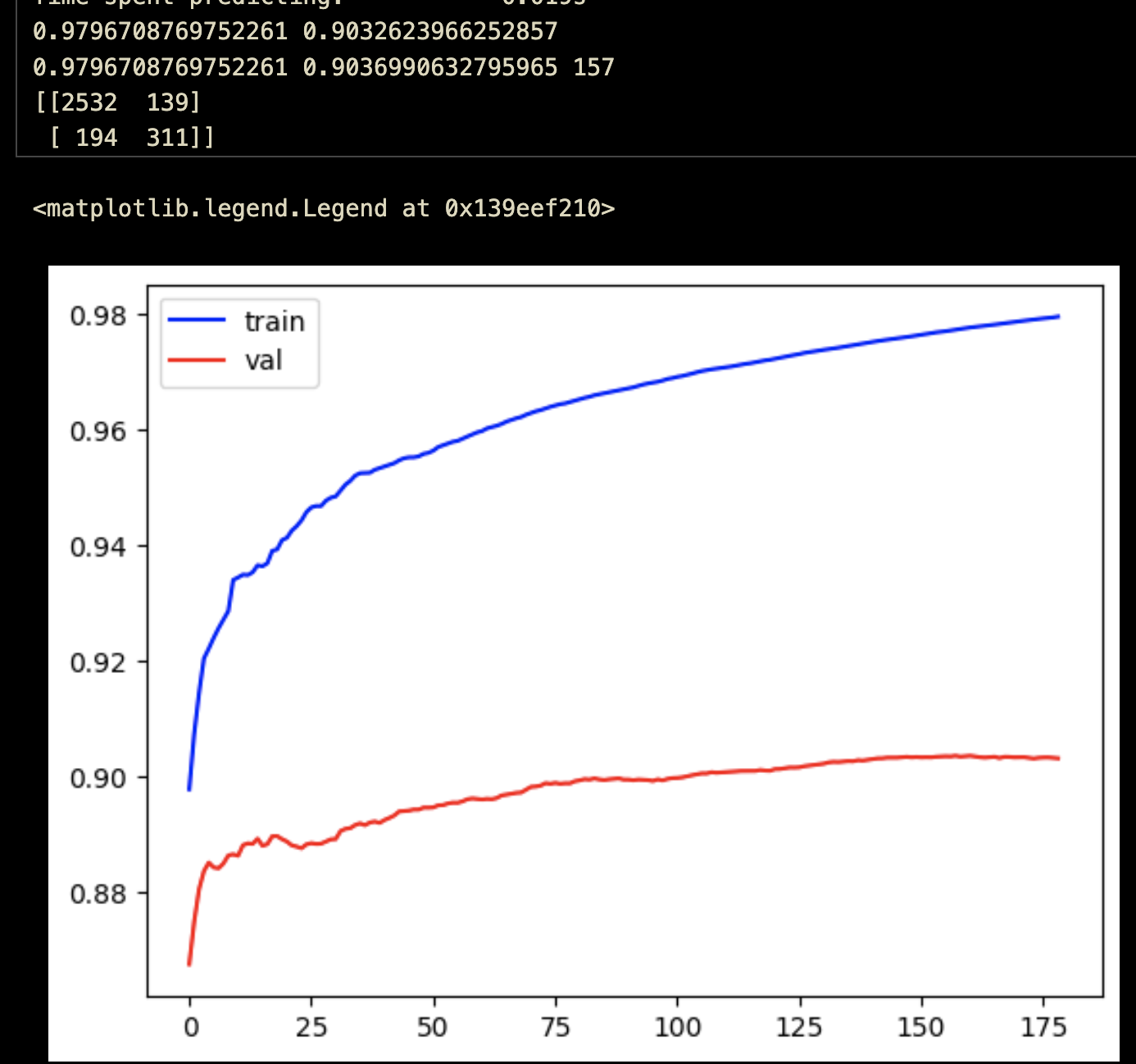
This proved to be effective as the AUC increased from 0.874 -> 0.876. It wasn’t much compared to adding in the prefix, but it still was an improvement.

At this point in time, the only groupby that occurred was based upon the customer\_id. We had yet to consider the transaction category and the season. For the transaction category, there is a ton of information to be discerned. If someone makes a huge payment, the reason why can give insight into their spending habits. It’s a lot more understandable if it’s on something like a medical bill, rather than a luxury item. Furthermore in regards to time/seasonality, when someone spends is an important factor in determining their spending habits. With these ideas in mind, a multi-index groupby was performed also on transaction category and season.

Unsurprisingly, the total feature count ballooned. By just considering the transaction amount, that put us at 2k features, and if also considering the raw balance, that effectively pushed it to 4k features.

When considering the full feature set, that pushed us past 0.90 AUC.

Dataset Shape: (15879, 4369)



We got past 0.90 AUC fairly early on, and from here, we tried a bunch of random things (more ratio features, smaller feature set with logistic regression), but did not manage to achieve any better performance. (Should be noted that the logistic regression models tended to have good recall scores. The ensemble models tended to squeeze out performance through TN rather than TP’s to the dataset imbalance, so if FPs matter more than FNs, the logistic regression model could be a better option.) We hit 0.905 AUC on a 5 fold stratified cross validation by the midpoint presentations/reports, and our best model for the final reveals/submissions clocked in at about 0.903 AUC.

It’s honestly a bit of a shame that we didn’t have access to the NLP data, as that certainly would’ve provided more features to work off of, and push the performance higher. We pretty much spent 3 or 4 weeks at the end, unable to improve the performance further.

**Overview of Final Entries [5pts]**

*1. What did you submit and how did you expect them to fare?*

*2. What were you lower and upper bounds on expected performance?*

*- How did you choose these bounds?*

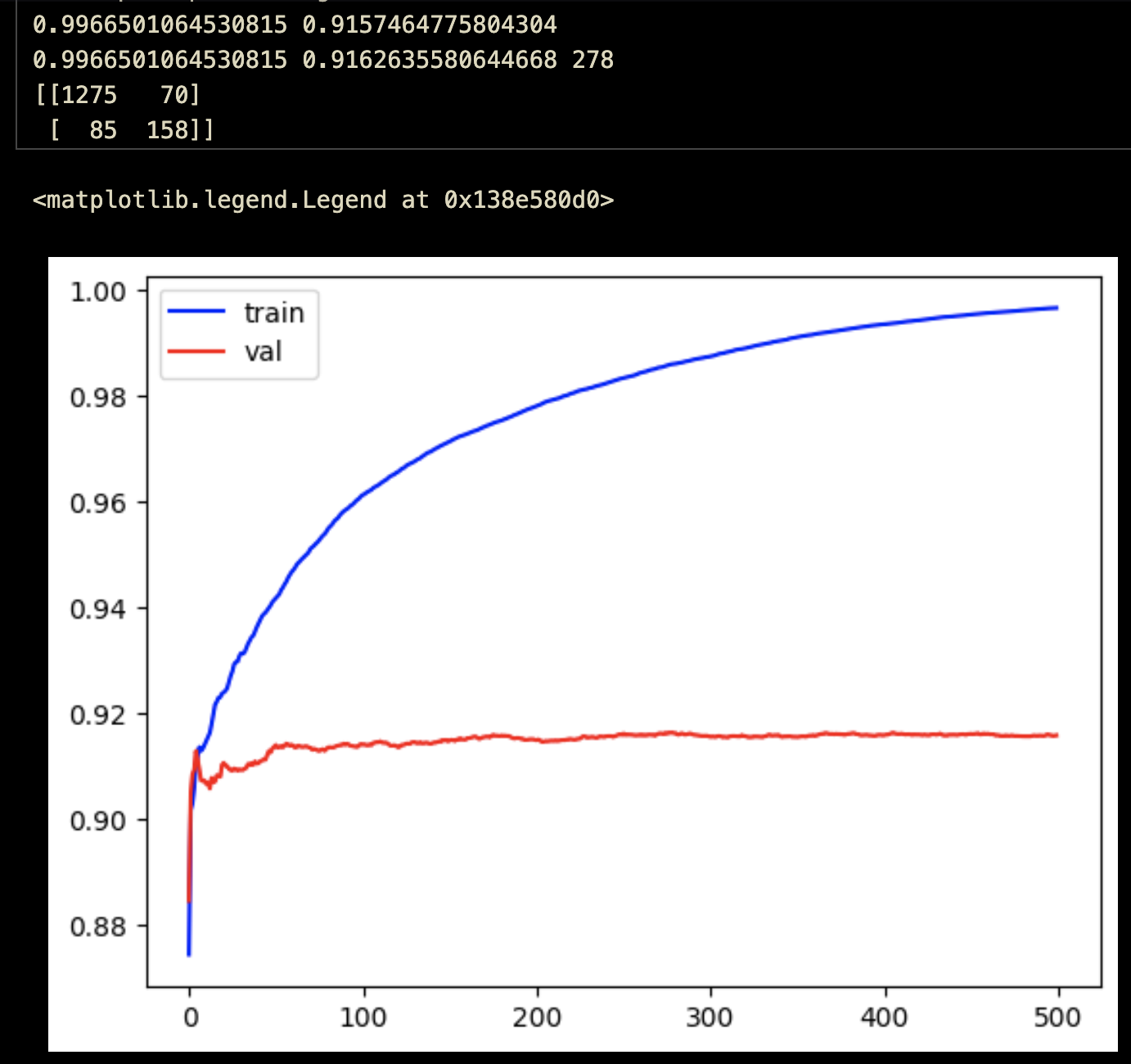
The first model we submitted was based upon a smaller subset of features that had been engineered earlier. It did not include some of the less fruitful summary statistics, and instead of doubling the features with total balance information, instead just keeping the final balance information as a separate feature. The feature set was about 1.5k features total, and the depth being operated at was 6. This model was ranging from 0.88 to 0.92 AUC depending on the splits being used.

This was more or less how we calculated our expected upper and lower bounds. By running 5 fold cross validation many times on different random seeds. And looking at the minimums and maximums.

The fact that performance drastically varied depending on the training dataset was rather concerning. Since we would be retraining the model on the full dataset for the final submission, we didn't know whether the performance on the full dataset would be closer to the upper bound or lower bound. Furthermore, since the test dataset could very well be notably different from the training dataset, we were uncertain how well our validations would generalize to the test dataset.

This ended up providing the main motivation for the lower depth stub model. Essentially, we noticed that when training the high-depth model for hundreds or even thousands of trees, the validation performance would be unstable, and end up drifting away from the training performance. This was a sign that the model was overfitting.

However, when running a low-depth model for hundreds of trees, even when the training auc went to 0.99 and above, the validation auc would not drop. This is likely because of the way low depth boosted decision trees work, where the overfitting cutoffs are EXACTLY around the training data points, and do not provide much if any space around them. Other points in the test dataset have to be EXTREMELY close to a training datapoint in order to be incorrectly classified.

****

\*Validation auc does not drop at all.\*

For the high-depth model, there were concerns on whether when training on the full dataset, we would have the right number of trees. If the trees are too low, the model would not be at its full potential, and if too high, the model would overfit. However, for the low depth stub model, it had the nice property that we could train it for an arbitrary number of iterations (hundreds of millions of trees), and there would be the guarantee that it wouldn't overfit. The training accuracy would be 1, but the validation accuracy would not drop, which is not overfitting. Thus, for the full dataset, the low depth stub model would be guaranteed to be at it's full potential.

Of course, the lower depth came with the cost that it would not be able to pick up on the more complex patterns in the data. However, the benefits of consistency made it an attractive option. For the times we ran the low depth model for 5 fold cross validation, the AUC-ROC score ranged between 0.895 and 0.905. These were what we put as the lower and upper bounds.

**Retrospective Analysis [5pts]**

*1. What went right or wrong in each of your entries*

Low depth: Ended up having the highest AUC score in the class of 0.903. Fell within the expected lower/upper bound range of 0.895-0.905. The other team’s models tended to provide rather optimistic estimates for performance, and that showed on the test dataset. I think the stub model was the only model whose test score was higher than the expected score. I was not expecting it to perform the best, but was pleasantly surprised in the end.

High depth: Was by nature intended to be a high-risk high-reward model. Was meant to be submitted in tandem with the low depth model (the safer, more consistent guarantee), so the two entries both pretty much went as expected. It did not end up performing better than the low-depth model, but given the uncertainty of training on the full dataset, it was not much of a surprise.

*2. In hindsight, how could each of your entries be improved*

Low depth: Used a depth of 4. If there were more submission slots provided, it would’ve been cool to try out a 3 depth and 5 depth ensemble. It’s almost certain that the depth hyperparameter could’ve been better.

High depth: The summary statistic feature set has pretty much been exhausted at this point. The only way I can think of where this model would perform better would be through better hyperparameters. A gridsearch was used, which ended up being very slow, and somehow the idea of a random search did not occur to us, until after the competition had ended. Doing a random search on a larger set of hyperparameters would’ve been a more efficient way to search the hyperparameter space. The idea of random search is essentially that compared to gridsearch which aims to find the best hyperparameter combination (very very slowly), random search aims to find something within the top “100” hyperparameter combinations. After finding a good hyperparameter set with random search, could then use gridsearch to narrow things down further in that subspace.

3. What would be the next steps you would take given more time

The summary statistic method ended up being the best thing we could find. However, that does not mean that it’s the best thing. So much information on the time series is lost through crushing things down to summary stats, and I’m confident that there’s some kind of an architecture that can learn on the multidimensional variable length time series directly. Aside from this, it would’ve been nice to try learning some multidimensional embedding for transaction categories. Our current approach ignores all semantics and just assumes that each category is orthogonal to one another. This is absolutely not the case.