IFT 6390 - Data Challenge 2

Axel Bogos
SID: 20091252
Kaggle username: axelbog
Kaggle team name: honest work
DIRO
Université de Montréal
Montreal, QC
axel.bogos@umontreal.ca

Pulkit Madan
SID: 20213197
Kaggle username: pulkit117
Kaggle team name: honest work
DIRO
Université de Montréal
Montreal, QC
pulkit.madan@umontreal.ca

Introduction

While food scarcity and crop supply are perhaps some of humanity's oldest challenges, the ongoing pandemic-induced shortages and the prospect of a rapidly changing climate makes addressing these challenges in novel ways especially relevant. As extreme weather events are expected to grow both in frequency and intensity (NOAA National Centers for Environmental Information 2021), leveraging satellite data and tools developed by the machine learning community in order to assess the state of agricultural land might help to both measure the impact of events such as wildfires or flooding on cropland, track population displacement and ultimately improve food security. The CropHarvest dataset (Tseng et al. 2021) aims at lowering the barrier of entry for ML practitioners interested in the aforementioned challenges. As part of the second data challenge of IFT-6390, an exploration of the dataset and model selection and optimization was conducted. In this document, the pre-processing of the datasets, the models explored, the results obtained and finally a discussion of future endeavours and a reflection on our results will be conducted.

Pre-Processing & Feature Engineering

The dataset consists of 18 distinct type of observations, each over 12 months. 13 of these distinct observations are from satellites, with columns named *B2_X B3_X B4_X B5_X B6_X B7_X B8_X B8A_X B9_X B11_X B12_X, VV_X, VH_X* where X is the month number $\in \{1, 2, \ldots, 12\}$. The remaining columns *temperature_2m_X, total_precipitation_X, elevation_X, slope_X, NDVI,_X* are also obtained from satellites but named after the specific meteorological data observed. All columns were numeric and complete, hence relatively minimal pre-processing was needed. Two feature processing lane were explored, besides standard scaling.

Dimensionality Reduction

Co-linear reduction Upon inspection, more than 124 features were found to be at least 90% correlated with another feature beyond this feature set; however experimental trials have shown reducing the dataset to this independently linear feature set resulted in general performance decrease. Upon reflection, it is to be expected that different satellite data may correlate each other across months while still providing some relevant label information, hence we decided not to further explore this technique of dimensionality reduction.

Temporal Rolling Window Aggregation Based on a general intuition of cropland being correlated with cyclical time of years (eg. spring and summer in the Northern Hemisphere or wet season in equatorial regions etc…), aggregating each satellites data over 4-months windows was deemed an interesting lane of exploration for feature engineering and dimensionality reduction. We compare our cross-validated F1 scores using this aggregated representation and the full-feature set representation in table 2.

Standard Scaling

Unless otherwise mentioned, all experiments have been run on a scaled version of the dataset, that is each feature have a mean of 0 and a standard deviation of 1.

Missing Values

As the dataset was complete with no feature having any missing values, no process had to be done to impute values or drop certain rows.

Methods

Cross-Validation & Train-Test Split

5-fold cross-validation was used across the board for model selection and optimization.

Train-Test Split

As the test set is hosted on Kaggle and 5-fold cross-validation was used across the board, no particular ratio of train-val-test split was used.

Random Forest

Algorithm Description The Random Forest algorithm is an ensembling method that relies on the aggregation-or the *bagging*- of numerous simple learners, in this case decision trees, where each simple learner is trained on a bootstrapped sample of our data set. The final class is determined by a majority-vote of these simple learners. Although simple in nature, this method is experimentally quite successful and forms a good baseline for our stronger models. In the end, the random forest ended up being among the best models both for the cross-validated F1 scores and on the final test set.

Optimization Methodology As the purpose of this model was to serve as a baseline, a hyper-parameter optimization was not conducted; as such default hyper-parameters from the *sklearn* library were used. These hyperparameters are shown in table 4.

XGB

Algorithm Description Both XGB and LGBM are implementations of the extreme gradient boosting decision trees algorithm (and successors of Adaboost). While like random forests this algorithm is also based on ensembling, it uses boosting instead of bagging. This consists of iteratively training simple learners (in this case a decision tree) and *reweighting* the cost associated with misclassified data points, increasing or decreasing this cost depending on whether it was misclassified or not by the simple learner. XGB and LGBM differ in the implementation rather than the theoretical background and motivation. They notably differ in how they implement the search for a new split in the decision tree; for our purpose this results in slightly different sets hyper-parameters to tune for each model and faster execution by the LGBM models.

Optimization Methodology Hyper-parameter search was done with *Optuna*, a hyper-parameter search optimization framework which accelerates the process by pruning unpromising trials, allowing to search a larger hyper-parameter space. Each trial was evaluated through a 5-fold negative log-loss. 150 trials were conducted, for a total of 750 fits. The hyper-parameter space searched can be found in table 5, while the best parameter set found can be seen in table 1. A plot of the optimization history and of hyper-parameter importance can be found in figure 2 and figure 1 respectively.

Table 1: XGB Hyper-parameters

Hyper-parameter	Value
learning_rate	0.11
n_estimators	150
reg_lambda	1.0
reg_alpha	0.6667
colsample_bytree	0.625
booster	gbtree

LGBM

Algorithm Description As this extreme gradient boosting decision tree algorithm was already described in the *XGB - Algorithm Description* section, we discuss it no more here.

Optimization Methodology The Optuna optimization framework was also used here. Each trial was evaluated through a 5-fold negative log-loss. 150 trials were conducted, for a total of 750 fits. The hyper-parameter space searched can be found in table 7, while the best parameter set found can be seen in table 6. A plot of the optimization history and of hyper-parameter importance can be found in figure 4 and figure 3 respectively.

Multilayer Perceptron

Algorithm Description We differentiate here between the multilayer perceptron and the Pytorch neural network mainly because of their uses rather than their design; while both rely on the same conceptual model (that is, a neural network with at least 1 hidden layer), we detail these two models separately as what we refer to as the 'multilayer perceptron' is an sklearn version of a neural network which was ultimately used as a model in a stacked classifier, while the second neural network was built using Pytorch and evaluated independently. They also slightly differ in the architecture due to different optimization strategies. Upon completion of the optimization process, an architecture of a single hidden layer with 132 neurons was settled on as this setup achieved the highest cross-validated F1 score.

Optimization Methodology An *Optuna* optimization was used for the *sklearn* version of the neural network; the main hyper-parameters to decide upon being the number of hidden layers and their respective number of neurons.The hyper-parameter space searched can be found in table 9, while the best parameter set found can be seen in table 8.

Pytorch Neural Network

Algorithm Description As we have already discussed the conceptual design of the neural network in the previous section, we do not further discuss it here. However, by using Pytorch, some design strategies differ; notably in the choice of the optimizer and the evaluation methods.

Optimization Methodology An Adam optimizer was used, conjointly with early stopping and manual trials of different architectures (namely variations in the number of layers and number of neurons per layer). To vary the experiment from the MLP, a larger model with more parameters was used than in the case of the MLP; a final model with one hidden layer of 512 neurons was settled on. To control over fitting, early stopping was used; nevertheless, few other hyper-parameters were tweaked beside the learning rate initialization, the hidden layer sizes and the early stopping tolerance in terms of epochs with no decreasing loss. An overview of these hyper-parameters can be found in table 10

Stacked Classifier

Algorithm Description A stacked classifier consists of using multiple models (which may be individually optimized) and for which the outputs constitute the input to a final model which is trained directly on the probability outputs of the former models. The overall stacked classifier and each of its constituent are trained with a 5-fold cross-validation, allowing the final meta-learning model to capture strengths and weaknesses of each model. Our optimized LGBM, XGB and MLP were used as the primary models, while the final meta-learner was a logistic regression model.

Optimization Methodology No particular hyper-parameter optimization was done for this model as it relied on the previously mentioned optimization of the MLP, LGBM and XGB models.

Results

On the processed subset of CropHarvest (Tseng et al. 2021), we compare the performance of one bagging classifier (Random Forest), two boosting classifiers (XGB, LGBM), 2 implementations of neural network (MLP and a Pytorch neural network) and a stacked classifier. As the training dataset is imbalanced in the target classes, with number of *non-crop* samples almost twice as that of *crop* samples, we use cross-validated F1 score as our performance evaluation metric, instead of accuracy. Hyper-parameter optimization, on the other hand was optimized as a minimization of a log-loss. Table 2 shows that the XGB and stacked classifier models outperform the other models on the original dataset. We also observe a trend where each model performed similarly or worse on the aggregated dataset as compared with the original dataset. In Table 3, we compare the performance of our models on the private and public sets on Kaggle. On the public dataset, we observe that LGBM and stacked classifier achieve same F1 score as the *Best TA model*, whereas XGB outperforms all other classifiers. On the private dataset, the random forest classifier surprisingly outperformed other classifiers by achieving an F1 score of 0.99779, which is second highest on the Kaggle leader-board, while the XGB and stacked classifier still performed quite well, outperforming the best TA baseline again.

Table 2: Cross-validated F1 scores

Model	*Original data*	*Aggregated data*
MLP	0.869	0.869
Neural Network	0.8776	NA
Random Forest	0.885	0.88
LGBM	0.886	0.882
XGB	**0.89**	0.878
Stacked Classifier	**0.89**	0.882

Table 3: F1 scores on test sets

Model	*Public*	*Private*
Neural network	0.905	0.87599
MLP	0.926	0.91027
Best TA Model	0.99516	0.98043
LGBM	0.99516	0.99558
Stacked Classifier	0.99516	0.99669
Random Forest	0.99516	**0.99779**
XGB	**0.99757**	0.99669

Discussion

To evaluate our models, we utilized F1 score as our metric. Additionally, to test the generalizing capacity of our models and to obtain better estimate of our models capacity, we use sklearn's *StratifiedKfold()* method which uses stratified folds such that each fold has the same number of samples from each class thus overcoming the class imbalance present in our training data. To establish a baseline, we used random forest classifier with sklearn's default parameters. Next, we tested the effectiveness of two implementations of a neural networks; sklearn's MLP and Pytorch. As we observed that cross-validated F1 score was lower than our baseline on both the networks and both had a strong tendency towards over fitting our data, we changed our approach and started experimenting with boosted decision tree classifiers: XGBoost(XGB) and LGBM. Both of the methods performed better than the baseline in terms of cross-validated F1 score. Furthermore, the XGB outperformed the LGBM which may be attributed to the difference in the way it finds a new split in the decision tree. Finally, to further increase the generalizing performance of our classifier, we implemented a stacked classifier, the setup of which is detailed in the *Methods* section. We assume that the individual classifiers present in the stacked classifier capture different representations from the data and thus, produced a strong final representation which achieves a tied-top (with XGB) F1 score of 0.89.

We also tested our models on the dataset obtained by temporal rolling window aggregation to measure its effectiveness and observe that all our models produce inferior results when compared with the original dataset (Table 2). We believe that since we are taking the average of four months to aggregate the data for each sensor, we might be losing some crucial variations present in this window. Given inferior performance, we utilize the original dataset for further experiments. Nevertheless, we do think there might be possible refinements of this dimensionality reduction as it also has the potential to smooth out macro noisy observations.

The evaluation on the Kaggle test dataset produces similar trends as observed during training, with rather a strange exception of the random forest classifier. Both neural networks achieved F1 scores that were worse than the Best TA Model whereas all our other models are able to either match or beat this baseline, suggesting a strong generalizing performance and an overall good fit of the problem with tree-based methods. While XGB performs the best on the public dataset, random forest classifier with the default parameters surprisingly outperformed all other methods on the private dataset.

Future work: Even though our models achieved very high scores on the test data, they don't explicitly exploit the time-series nature of the data. With the sensors collecting data every month, exploring models such as recurrent neural networks or LSTMs which explicitly exploit the sequential nature of data and store and utilize the previous state(s), would be an interesting direction for future work. Another direction of exploration can be in the domain of data aggregation where we explore ways to aggregate the data without losing important variations. Furthermore, as random forests ended up being the top-performing model, exploring the optimization of its hyper-parameters and/or of related models such as

ExtraTrees would likely lead to a slight improvement; that being said, with the current scores being very high it may or may not be the case.

Statement of Contributions

I hereby state that all the work presented in this report is that of the authors.

References

NOAA National Centers for Environmental Information. 2021. U.s. billion-dollar weather and climate disasters.

Tseng, G.; Zvonkov, I.; Nakalembe, C. L.; and Kerner, H. 2021. Cropharvest: A global dataset for crop-type classification. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Appendix

Table 4: Random Forest Hyper-parameters

Hyper-parameter	Value
n_estimators	100
min_samples_split	2
min_samples_leaf	1
max_depth	None
max_features	auto
bootstrap	True

Table 5: XGB Optuna Hyper-parameters Search Space

Hyper-parameter	Range	Step
n_estimators	[100, 4000]	Uniform
booster	[gbtree, gblinear, dart]	Categorical
scale_pos_weight	[1, 10]	Uniform
lambda	[1e-8, 1]	Log-Uniform
alpha	[1e-8, 1]	Log-Uniform
max_depth[1]	[1, 12]	Uniform
eta[1]	[1e-8, 1]	Log-Uniform
gamma[1]	[1e-8, 1]	Log-Uniform
grow_policy[1]	[depthwise, lossguide]	Categorical
sample_type[2]	[uniform, weighted]	Categorical
normalize_type[2]	[tree, forest]	Categorical
rate_drop[2]	[1e-8, 1]	Log-Uniform
skip_drop[2]	[1e-8, 1]	Log-Uniform

[1] Applicable for gbtree and gblinear boosters
[2] Applicable for dart booster

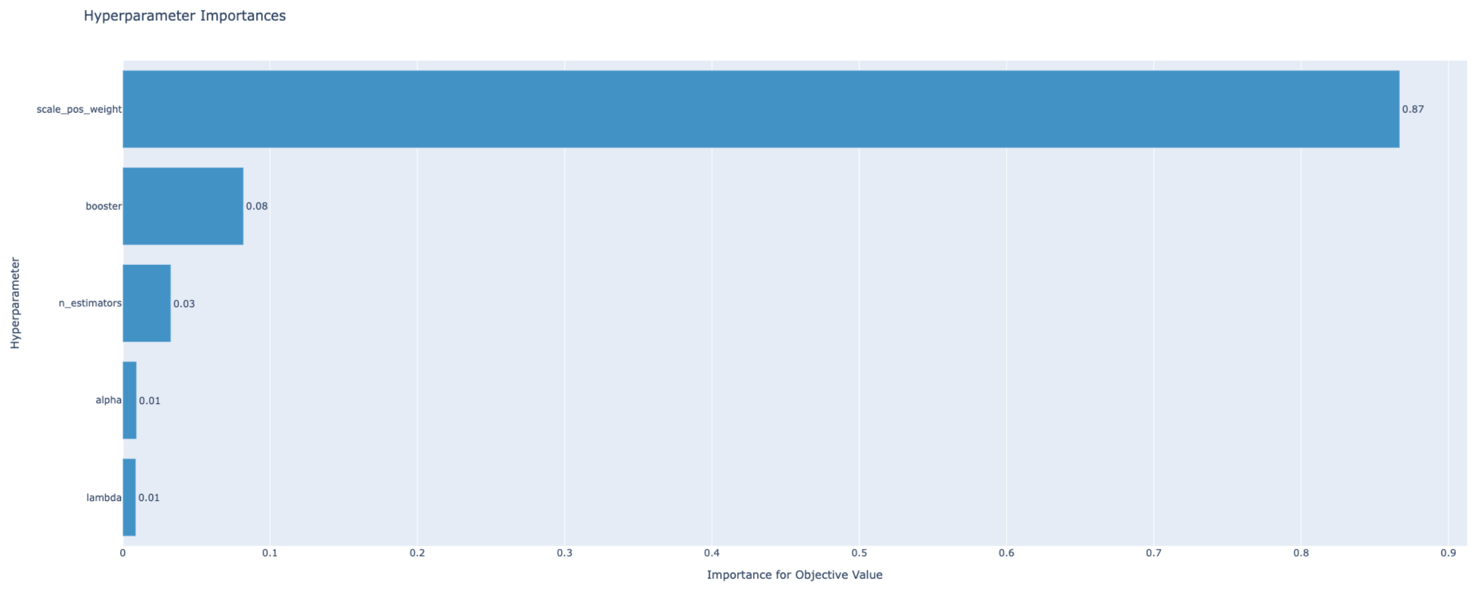

Figure 1: XGB Hyperparameter Importance by Mean Decrease in Impurity.

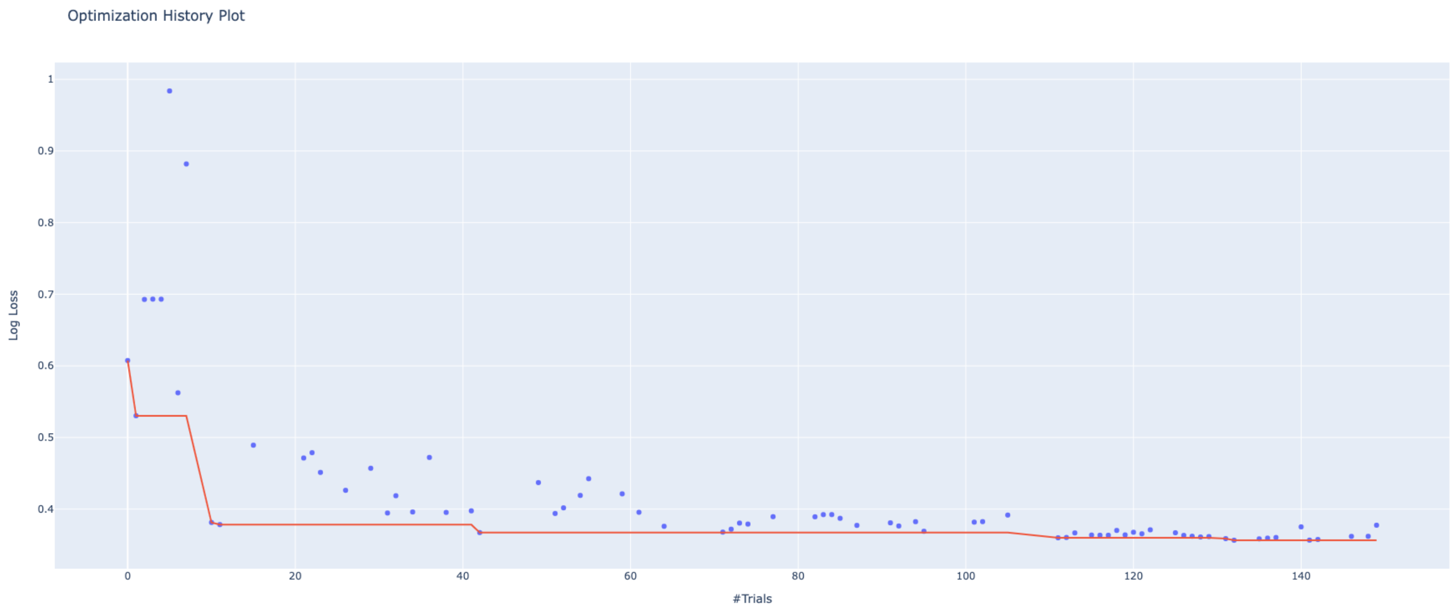

Figure 2: XGB Optuna Optimization History Plot

Table 6: LGBM Hyper-parameters

Hyper-parameter	Value
n_estimators	4600
learning_rate	0.22904
num_leaves	728
max_depth	11
min_data_in_leaf	300
lambda_l1	5
lambda_l2	30
min_gain_to_split	0.056522
bagging_fraction	0.8
feature_fraction	0.4

Table 7: LGBM Optuna Hyper-parameters Search Space

Hyper-parameter	Range	Step
n_estimators	[100, 5000]	100
learning_rate	[0.01, 0.3]	Uniform
num_leaves	[8, 3000]	20
max_depth	[3, 12]	Uniform
min_data_in_leaf	[20, 10000]	100
lambda_l1	[0, 100]	5
lambda_l2	[0, 100]	5
min_gain_to_split	[0, 15]	Uniform
bagging_fraction	[0.2, 1]	0.1
feature_fraction	[0.2, 1]	0.1

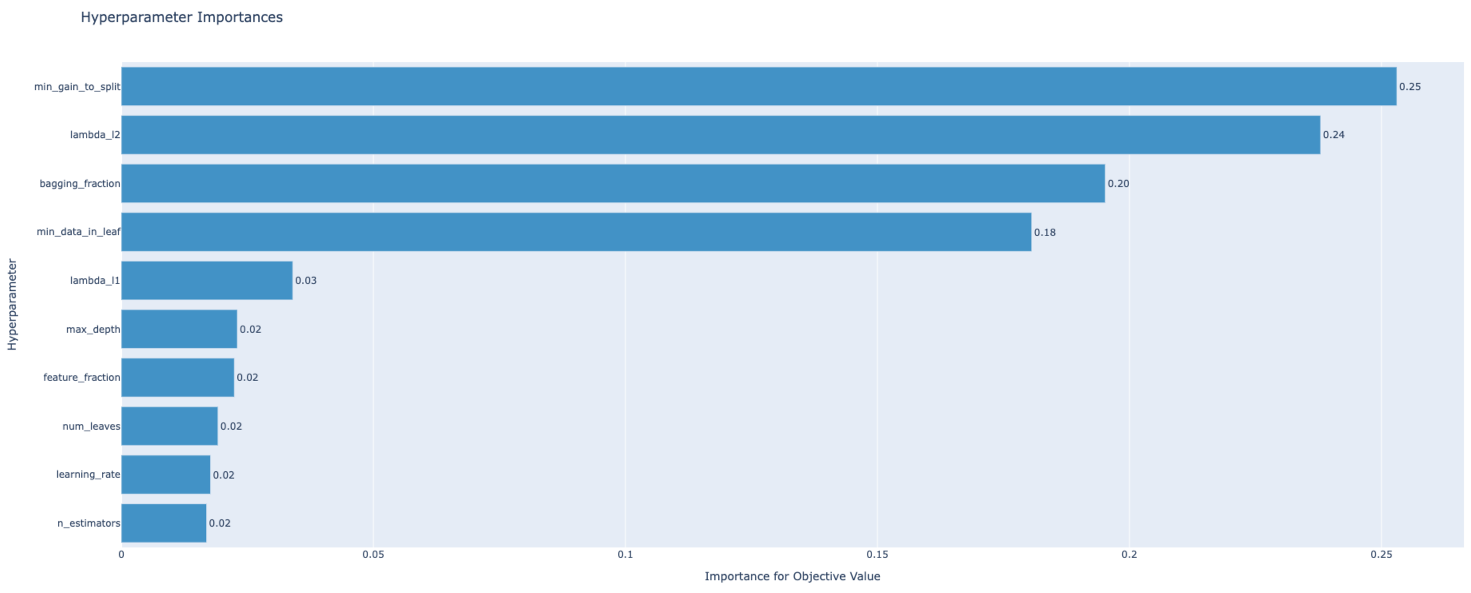

Figure 3: LGBM Hyperparameter Importance by Mean Decrease in Impurity.

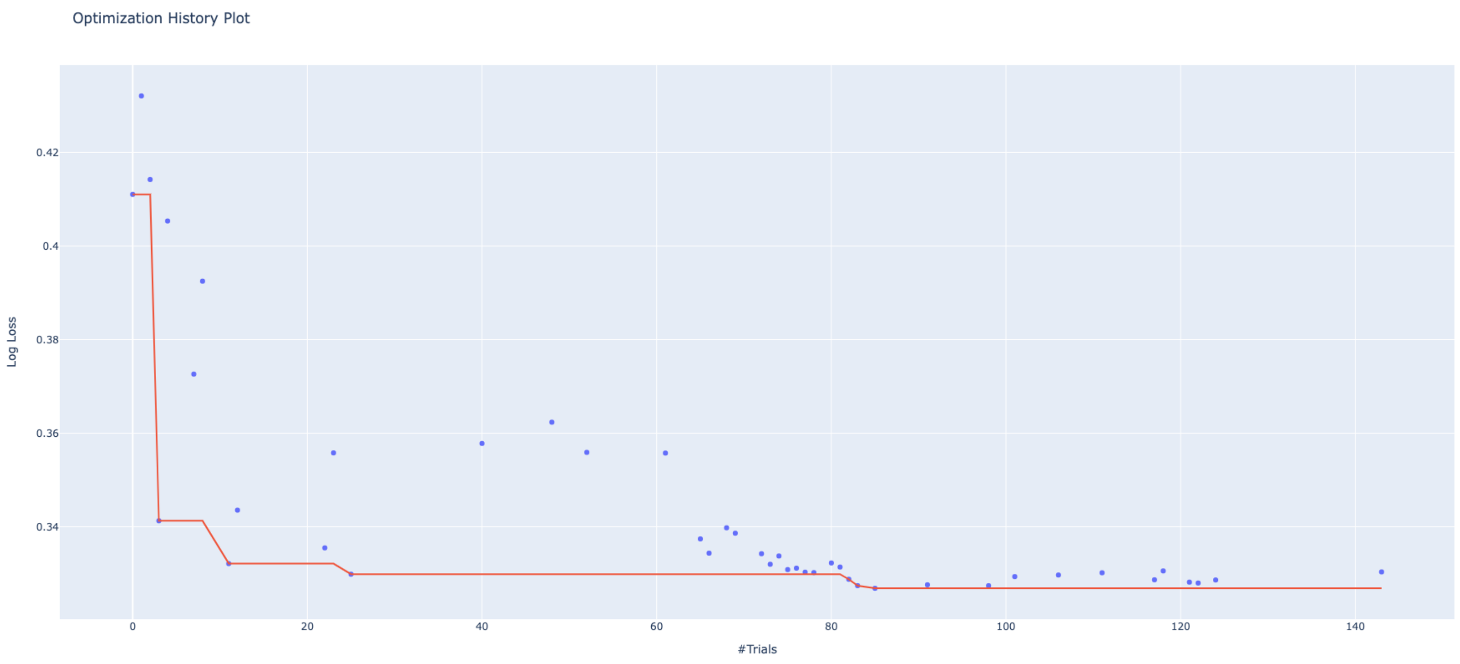

Figure 4: LGBM Optuna Optimization History Plot

Table 8: MLP Hyper-parameters

Hyper-parameter	Value
hidden_layer_sizes	(132)
learning_rate_init	1.70162e-05
learning_rate	adaptive
momentum	0.542808

Table 9: MLP Optuna Hyper-parameters Search Space

Hyper-parameter	Range	Step
hidden_layer_sizes (# of layers)	[1, 5]	1
hidden_layer_sizes (# of neurons)	[1, 150]	1
learning_rate_init	[1e-5, 0.3]	Loguniform
learning_rate	[adaptive, constant, invscaling]	Categorical
momentum	[0, 0.99]	Uniform

Table 10: Pytorch Neural Network Hyper-parameters

Hyper-parameter	Value
learning_rate_init	1e-3
optimizer	Adam
early stopping tolerance	3
activation function	ReLU