

Image created with ChatGPT

Using Machine Learning to Predict the Next Command in Revit _ Part 5: Parameter Optimization Algorithms Using Python with Optuna







Select C:\WINDOWS\system32\cmd.exe

(Part 5 of a 7 series blog. Click here to go to Part 1)

```
/* LightGbm */
             public static IEstimator<ITransformer> BuildPipeline(MLContext mlContext)
            {
                // Define options for the LightGBM multiclass trainer.
                var options = new LightGbmMulticlassTrainer.Options
150.
                    LabelColumnName = "col4",
                                                               11
                                                                                          Specifies the column containing the Label (target).
                   FeatureColumnName = "Features",
                                                                                          Specifies the column containing the features.
                                                              //
                                                                                        Sets the number of boosting iterations .
                   NumberOfIterations = 257,
                                                              // ('n_estimators')
                   LearningRate = 0.07254478842750792,
                                                              // ('learning_rate')
                                                                                         Sets the Learning rate .
                   NumberOfLeaves = 47,
                                                               // ('num_Leaves')
                                                                                          Sets the maximum number of Leaves in each tree .
                   MinimumExampleCountPerLeaf = 20,
                                                               // ('min_child_samples') Minimum number of samples per leaf .
157.
159.
                    Booster - new GossBooster.Options
                                                                                          Define options for the GOSS (Gradient-based One-Side
     Sampling) booster.
169.
                                                                                          Sets the maximum depth of the tree .
161.
                        MaximumTreeDepth - 20,
                                                               // ('max_depth')
                       SubsampleFraction = 0.6453484735118924, // ('subsample')
                                                                                         Fraction of data to be used for training .
                        FeatureFraction = 0.9707222345576305, // ('colsample_bytree')
                                                                                         Fraction of features to be used in each iteration .
                        L1Regularization = 0.9953559556619561, // ('reg_alpha')
                                                                                          L1 regularization term .
                        L2Regularization = 0.32241853553570154, // ('reg_lambda')
165.
                                                                                          L2 regularization term .
166.
167.
                };
```

In the code snippet above, which is from *Part 4* of this blog (the console trainer application), you can see several *LightGBM* parameters. These include the learning rate, number of leaves, and more—all of which can be adjusted to influence your model's performance. I spent a considerable amount of time manually tweaking these parameters, and it was not a pretty process. My method involved lowering a parameter, checking the accuracy of predictions on a test set then increasing the parameter if the accuracy improved. I'd repeat this back-and-forth adjustment until I found a sweet spot for that single parameter. While this method works for fine-tuning one parameter, there are nine parameters in the code snippet. Good luck dialing them all in manually!

configurations of parameters and identify which combination yields the best results. I could have done this within *Visual Studio C#* using *AutoML*. However, I struggled with it early in the project and ended up taking a detour. Although I initially wanted to avoid *Python*, I eventually turned to *Optuna*: a *Python* library that can easily optimize parameter values. With *Optuna*, you can specify constraints for the parameters and define the number of trial runs, then it will return the best trial. For *AG Feeling Lucky*, I used 100 trial runs to find the optimal parameter values. Below is a screenshot of three trial runs (fitting them into the console was a challenge):

Fortunately, there's a better way: using optimization algorithms. These algorithms test various

```
-08-19 23:58:35,109] A new study created in memory with name: no-name-11adbccb-0bf2-4fb3-8628-db96b2d362cb
                                                                                                                        | 0/3 [00:00<?, ?it/s]T
raining until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[106] valid_0's multi_logloss: 2.22204
[I 2024-08-19 23:58:37,594] Trial 0 finished with value: 0.5695422535211268 and parameters: {'num_leaves': 52, 'learning
rate': 0.015436381927094563, 'n_estimators': 495, 'max_depth': 13, 'min_child_samples': 29, 'subsample': 0.697151452519
968, 'colsample_bytree': 0.9240177157015217, 'reg_alpha': 0.1251764202870428, 'reg_lambda': 0.3037576447280913}. Best is
trial 0 with value: 0.5695422535211268.
Best trial: 0. Best value: 0.569542: 33%
                                                                                                              1/3 [00:02<00:04, 2.48s/it]T
raining until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[173] valid_0's multi_logloss: 2.27715
[I 2024-08-19 23:58:40,102] Trial 1 finished with value: 0.5536971830985915 and parameters: {'num_leaves': 56, 'learning
rate': 0.029231825025796798, 'n_estimators': 815, 'max_depth': 10, 'min_child_samples': 78, 'subsample': 0.813031250141_
117, 'colsample_bytree': 0.5164543013426763, 'reg_alpha': 0.8033063883288756, 'reg_lambda': 0.2660725571816107). Best is
trial 0 with value: 0.5695422535211268.
Best trial: 0. Best value: 0.569542: 67%|
                                                                                                              2/3 [00:04<00:02, 2.50s/it]T
raining until validation scores don't improve for 10 rounds
Early stopping, best iteration is:
[60] valid_0's multi_logloss: 2.27232
                                  Trial 2 finished with value: 0.5519366197183099 and parameters: {'num_leaves': 23, 'learning
_rate': 0.03514967688414209, 'n_estimators': 103, 'max_depth': 6, 'min_child_samples': 57, 'subsample': 0.65690797412846
22, 'colsample_bytree': 0.6461361538350394, 'reg_alpha': 0.011526192552547498, 'reg_lambda': 0.5418231584635467}. Best i
s trial 0 with value: 0.5695422535211268.
Best trial: 0. Best value: 0.569542: 100%
                                                                                                             3/3 [00:06<00:00, 2.07s/it]
Number of finished trials: 3
Best trial: {'num_leaves': 52, 'learning_rate': 0.015436381927094563, 'n_estimators': 495, 'max_depth': 13, 'min_child_s
amples': 29, 'subsample': 0.697151452519968, 'colsample_bytree': 0.9240177157015217, 'reg_alpha': 0.1251764202870428, 'r
eg_lambda': 0.3037576447280913}
Press any key to continue . . .
```

exactly. This discrepancy is part of the "silly detour" I mentioned. Since I broke this task into two different development environments, Visual Studio C# ML.NET and Python with Optuna, I matched the parameter names using a combination of Google, ChatGPT (there may have been a hallucination or two that slipped through), and some common sense. For example, in ML.NET, the maximum tree depth is referred to as MaximumTreeDepth, whereas in Optuna, it's max_dept":

The trial highlighted in white represents the best result with all its parameters listed. If you

cross-check these with ML.NET's LightGBM parameters, you'll notice that not all names match

```
depth of the tree.
```

// ('max_depth') Sets the maximum

Link to source code: https://pastebin.com/rySSUjcB

Script 6: 2_LIGHTBGM.PY

MaximumTreeDepth = 20,

To get this to work, create a new folder with any name and place the following files in it: 2_LIGHTGBM.PY, DATA-COOKED.txt, and a batch file called RUN.bat. If you have Python

3.9 installed, the contents of the batch file should be:

```
PAUSE

Now, all you need to do is run this batch file, and the Optuna script will display its progress.

Once it's done, take the optimal values it provides and plug them into the corresponding
```

Go to the next part

Go to the previous part

@echo off

Part 1 - Introduction & Index

Part 2 - Data Preprocessing Using PowerShell & Python with PyRevit

Part 3 - Data Analysis Using Power BI

parameters in your trainer console application from Part 4 of this blog.

Part 4 - Model Trainer Application Using C# ML.NET PowerShell Notepad++

Part 5 - Parameter Optimization Algorithms Using Python with Optuna

Part 6 - Revit Plugin Using C# ML.NET & Revit API

https://letsbimtogether.com/blog.html

Part 7 - Outro (Canva flow chart)

Download AG Feeling Lucky Plugin for Revit & Trainer/Analysis Console Application here: