Image created with ChatGPT

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

Let's BIM Together
728 followers

September 25, 2024

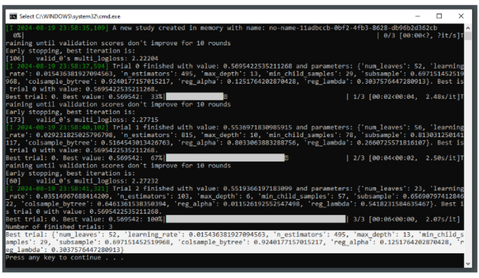
(Part 5 of a 7 series blog. Click here to go to Part 1)

```
        /* LightGbm */
        public static ITransformer> BuildPipeline(MLContext mlContext)
        {
            // Define options for the LightGBM multiclass trainer.
            var options = new LightGbmMulticlassTrainer.Options
            {
                LabelColumnName = "col4",                   //                  Specifies the column containing the label (target).
                FeatureColumnName = "Features",             //                  Specifies the column containing the features.
                NumberOfIterations = 257,                   // ('n_estimators') Sets the number of boosting iterations .
                LearningRate = 0.07254478842758702,         // ('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 .

                Booster = new GossBooster.Options           //                  Define options for the GOSS (Gradient-based One-Side
Sampling) booster.
                {
                    MaximumTreeDepth = 20,                  // ('max_depth')    Sets the maximum depth of the tree .
                    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 = 0.9935359956619561,  // ('reg_alpha')    L1 regularization term .
                    L2Regularization = 0.3224185555357014,  // ('reg_lambda')   L2 regularization term .
                }
            };
```

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!

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



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 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*":

```
MaximumTreeDepth = 20,                  // ('max_depth') Sets the maximum
depth of the tree.
```

Script 6: 2_LIGHTBGM.PY

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

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:

```
@echo off
py -3.9 2_LIGHTGBM.PY
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 parameters in your trainer console application from *Part 4* of this blog.

Go to the next part

Go to the previous part

Part 1 - Introduction & Index

Part 2 - Data Preprocessing Using PowerShell & Python with PyRevit

Part 3 - Data Analysis Using Power BI

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

Part 7 - Outro (Canva flow chart)

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

https://letsbimtogether.com/blog.html