Dynamic HyperParameter Optimisation with Optuna

By: Daud Waqas

This is a quick project centred around exploring how institutions and researchers may approach hyper parameter optimisation in cases of high model and system complexities, especially since traditional approaches such as **Grid Search** and **Random Search** present some notable limitations. **Grid Search**, while exhaustive, becomes *computationally expensive* as the number of hyperparameters (and their potential combinations) increases. On the other hand, **Random Search** provides a more efficient alternative but lacks systematic exploration, making it unreliable in scenarios w*here it is important to understand the "thought process" behind reaching a set criteria of optimised hyperparameters* (think of industries such as quantitative trading, healthcare, etc). To address these challenges, a more dynamic and adaptive solution was required. This report explores the effectiveness of **Optuna**, an open-source hyperparameter optimization library that *intelligently selects hyperparameters based on prior evaluations*, without exhausting combinations deemed to be inefficient, making it significantly more efficient and effective than traditional methods.

First, its necessary to cover a high level overview of both Grid Search and Random Search, and their associated limitations:

- **Grid Search** is a brute-force method that systematically tests *all possible hyperparameter combinations within predefined value-sets*. While this guarantees that the optimal combination is tested, it quickly becomes an inefficient use of compute and resources as the number of parameters and possible values increases. For example, with 10 hyperparameters, each having three possible values, the total number of trials required would be ***3^10*** = ***59,049***, making it impractical for large-scale applications.

- **Random Search** improves upon Grid Search by *selecting hyperparameter values randomly from a specified range rather than exhaustively testing all combinations*. This method can be more computationally efficient, as it does not require testing every possible set of hyperparameters, and will eventually land on a "feasible" result. However, it introduces an element of chance—optimal configurations may be missed due to the random nature of the search. As the number of hyperparameters increases, the likelihood of finding an optimal combination without an excessive number of trials decreases significantly. Since there is also **no guarantee of the hyperparameter combination being "optimal"**, it poses a problem to industries which highly value interpretability.

Given the setbacks of these 2 methods, many other open-source libraries decided to incorporate machine learning (and lightweight deep learning) algorithms to give the best of 2 worlds; that is, a *non-exhaustive method for hyperparameter optimisation that is also highly interpretable*. Libraries like Optuna and RayTune are examples of this, though for this paper I'll be focusing on Optuna.

Optuna is a Python-based framework designed to optimise hyperparameters efficiently by leveraging *Bayesian optimisation techniques, trial pruning, and an adaptive search process*. By using an iterative approach, **Optuna** continuously refines its search space as trails progress, and over time priorities hyperparameter combinations and ranges that show promise while eliminating trails deemed to be unproductive.

The 2 most critical mechanisms in this whole process is most definitely the **Tree-Structured Parzen Estimator (TPE)** and **trail pruning**.

The **Tree-structured Parzen Estimator (TPE)** is a Bayesian optimisation algorithm that *dynamically suggests new hyperparameter values based on prior trial results*. Unlike Grid and Random Search, which select values blindly, TPE models hyperparameter distributions and *actively shifts the focus of the optimisation onto the more promising areas within the search space*. The process follows this systematic process:

- **Data collection,** in the form of storing performance results for different hyperparameter settings.
- **The segmentation of trials,** making sure that past trials are split based on whether they are "good" or "bad" based on a performance threshold.
- **Distribution modelling,** which is the construction of probability distributions for both groups, prioritising hyperparameters associated with better performance while occasionally testing less-explored values *(to ensure that it does not miss potential opportunities within the search space)*.
- And **new parameter suggestion/s,** which is a comprehensive process of selecting new hyperparameter values that *balance both exploration and exploitation*, improving the search efficiency.

The function of the **Tree-structured Parzen Estimator (TPE)** is what gives **Optuna** a clear advantage over **Random Search**; Optuna's usage of the Bayesian optimisation algorithm within its TPE means that there is legitimate thought process and "reason" behind each subsequent trail, ensuring that the search space continuously narrows as each trail progresses.

Trail pruning is another essential optimisation feature in Optuna which *discards unpromising trials early, saving computational resources*. This process works similarly to early stopping in training, though is a lot more involving as it is utilised to determine the "unfavourable" ranges within the search space. I won't be going into a lot of detail regarding the mathematical processes which cover how trail pruning determines these ranges, but for those seeking reference it should be noted that the trail pruning within Optuna actively utilises a:

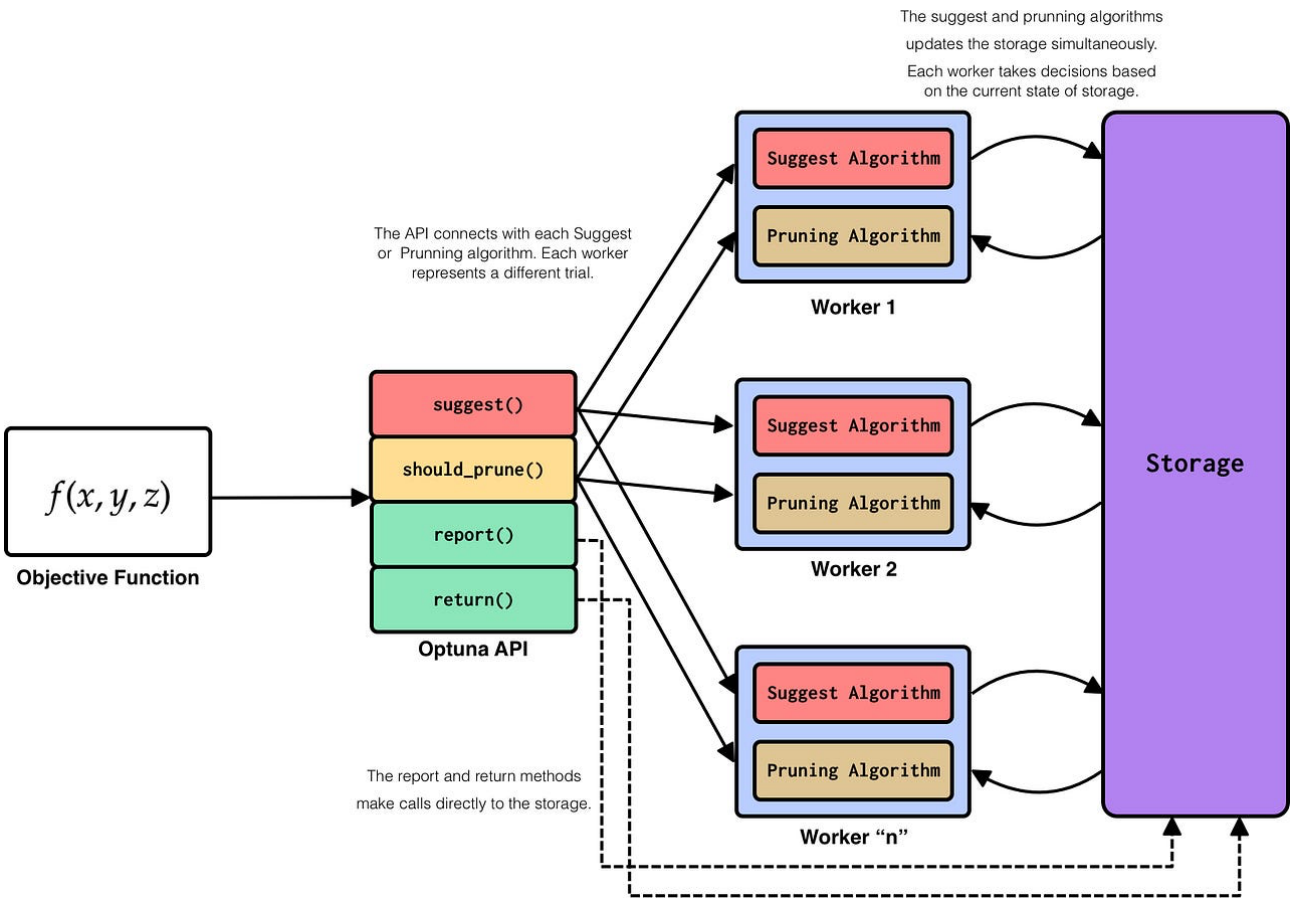
- Median Pruner,
- Hyperband,
- Successive Halving,
- and a Threshold Pruner

While the process involves:

- **Evaluating ongoing trials** by assessing performance during execution.
- **Comparison with previous results,** which actively checks whether the current trial performs worse than past trials at the same stage.
- **Early termination,** which eliminates trials that are unlikely to yield optimal results.

The function of **trail pruning** is what gives **Optuna** a clear advantage over **Grid Search**; the pruning of underperforming trails means that, unlike Grid Search, Optuna can yield great results *without exhausting the entire search space*, saving computational resources and time that could be directed to more efficient research.

The below diagram gives a great representation of how **Optuna** works, *where the input of the model, search space and number of trails is shown as the* ***objective function***:



Article: OPTUNA: A Flexible, Efficient and Scalable Hyperparameter Optimization Framework
Author: Fernando López
Link: *https://towardsdatascience.com/optuna-a-flexible-efficient-and-scalable-hyperparameter-optimization-framework-d26bc7a23fff*

The following pages will contain some visualisations of **Optuna**; the model being optimised is a **Categorical Boost (CatBoost) Classifier on the Adult UCI Dataset**, and the visualisations will be complimented with their equivalents for both **Grid Search** and **Random Search**.

It should be noted that the purpose of this study is not to best optimise the CatBoost model, but rather to observe how **Grid Search**, **Random Search** and **Optuna** handle the search space.

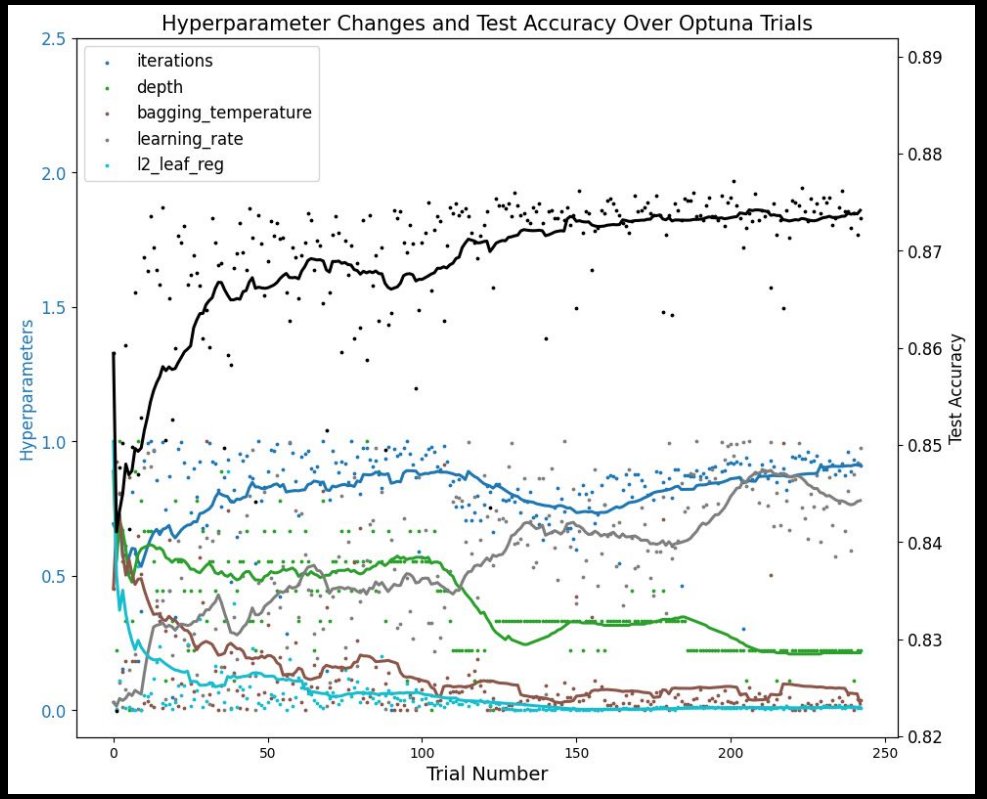
Visualisations

Now, when it comes to observing the effects and benefits of **Optuna**, it isn't really necessary to do a demo; can observe the effects of the **Tree-structured Parzen Estimator** and the **Pruning** through visualisations of **Optuna**, including the hyperparameter values it proposes and the final results it gives. This would also allow us to compare **Optuna**, from a practical standpoint, to the traditional **GridSearch** and **RandomSearch** methodologies.

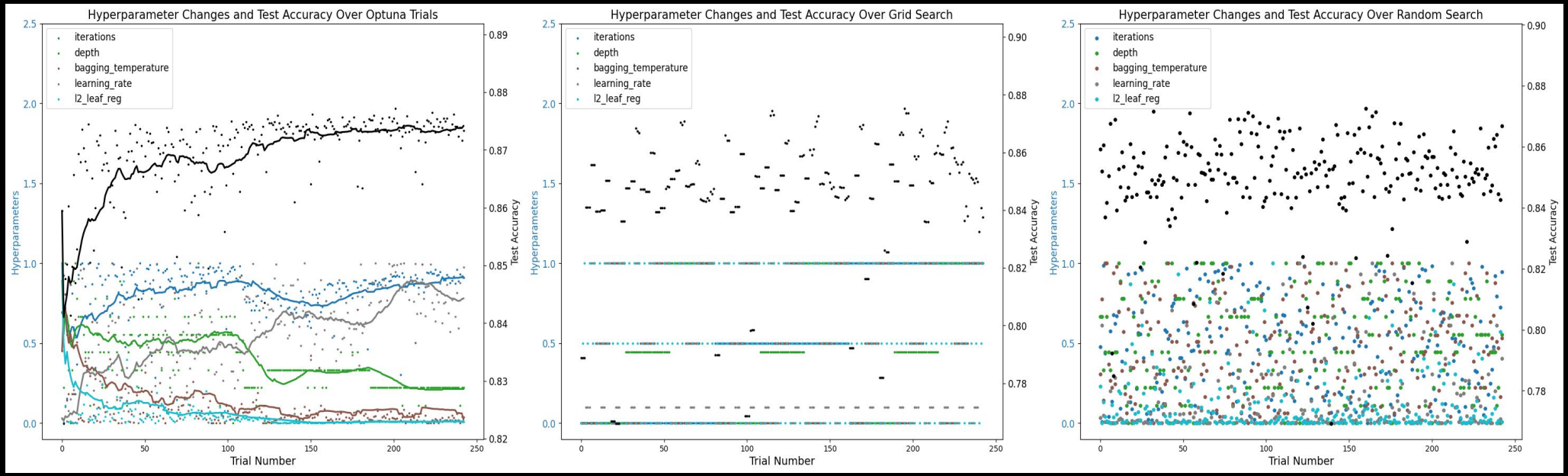
A visual analysis on ***Optuna***

Now, the plot given here is a representation of the **accuracy results** for each trial, where the **black** points *represent the accuracy* and the **colored** points *represent the given hyperparameter values for that specific trial*. From this graph, we can deduce that:

- After about **150** trials, Optuna has achieved a status quo according to the moving average line, with little to no improvement afterwards.
- The spread of **accuracy points (black)** is **converging** in as the number of trials increases *(ie Optuna is learning from prev. trails)*.
- The convergence or lack of convergence within the hyperparameter values is indicative of the effect (or lack of effect) of that hyperparameter on the accuracy.



*Comparison between **Optuna**, **Grid Search** and **Random Search***



Practical deduction based on the visuals

The **convergence** which **Optuna** provides does something which *neither Grid Search nor Random Search does*; it helps make any hyperparameter optimisations more **understandable** while also providing a **guarantee**. When working in a more critical environment, such as healthcare or banking, it is critical for data scientists to not just understand the model but also have a guarantee that their hyperparameter ranges are optimal. **Random Search**, in this sense, simply isn't a viable solution. Of course, data scientists could just increase the number of trials to increase the likelihood of getting a good configuration, but there will never be a **guarantee** that the given "best results" are actually fully optimal, since **Random Search** takes random samples without any clear purpose. **Grid Search** has a different problem, and that has to do with the **"exhaustion"**. Like what was explored beforehand, in complex scenarios with a high number of hyperparams, Grid Search becomes needlessly expensive and just ends up being not worth it.

Optuna fixes both of these issues; through **visualisations** like what I did in previous slides, data scientists *can determine why a specific combination of hyperparameters* was chosen (including observing the magnitude of effect that a specific hyperparam may have). Not only that, but due to the **TPE** and **pruning** that **Optuna** implements in its "learning", there is also high likelihood that **Optuna** *will require less trials versus **Grid Search** and its "exhaustion"*, making it more **efficient** in most complex scenarios *(high number of hyperparams, etc)*.

Final Considerations

- It is not always a good idea to use **Optuna**; in certain cases where only quick optimisations are needed *(such as cases where data scientists are more concerned about **data augmentation** than hyperparam tuning),* it is much more worthwhile to set up a simple **Grid Search** with a **3x3 configuration** *(3 hyperparams w/ 3 values each)* that would only take **27 trails** to exhaust. Please note that it does take time for **Optuna's TPE** and **Pruning** to actually converge *(I would say this is at least a minimum of 70/80 trials)*, so it's only really useful in cases where hyperparam optimisation is a large point for concern.
- **Optuna** isn't just used for hyperparameter optimisation; even though this is the most common use case, it *may sometimes be used for testing the effect of **different magnitudes** of **data augmentation***, especially in deep learning tasks like image classification. This is due to the **objective function** system in Optuna being open-ended in terms of what the value it proposes actually relates to.
- In theory, for the sake of more serious projects, it is pretty much **not recommended whatsoever** to use **Random Search**; the fact that it randomly samples without any explanation or clear purpose makes it highly risky for any production or critical workflow *(though it should be fine for more casual or simple envir)*.