Tool Comparison: Optuna vs Hyperopt

Audrey Bertin, Nasko Apostolov, Nelson Evbarunegbe, & Raymond Li

Introduction

For our project, we will be conducting a comparison of two tools designed to help with the process of **hyperparameter tuning** in a machine learning pipeline.

Hyperparameters are user-selected parameters applied to learning algorithms that affect how they are implemented–for example, constraints, weights, learning rates, number of algorithm layers/complexity, etc. Depending on which set of hyperparameters is selected, the results and performance of a particular machine learning algorithm can vary dramatically. The process of hyperparameter tuning is that of finding a good set of parameters for a specific model and dataset combination.

The possible search space is infinite, making this task challenging. Grid search, in which users pre-define a list of possible parameter combinations and test them one by one, has historically been a common way to do this. However, it's very subjective (completely up to the user to select which parameters they want to try), and can be very time consuming for testing many combinations.

Hyperparameter tools have been developed to assist with this, using advanced algorithms and parallel computing to allow users to test out parameters with a more mathematical and faster approach.

In this project, we are interested in comparing how two separate hyperparameter tuning tools work and perform–both in comparison to each other and to the standard grid search.

These tools are:

- **Hyperopt:** a popular Python library for hyperparameter optimization that has been around–and been considered industry standard–for more than a decade.
- **Optuna**: a much newer tool that has some advanced features and is gaining popularity.

Both tools have a UI-based component (either visualizations or a full dashboard) and both have their code publicly available on GitHub:

- https://github.com/hyperopt/hyperopt || https://github.com/optuna/optuna

They also both have publicly available documentation sites:

- http://hyperopt.github.io/hyperopt/ || https://optuna.org

We will evaluate these tools in two separate ways:

1. A **code and UI review**, where we look at the repositories and note things that they are each doing well or not so well, as well as look at the UI features provided by each tool
2. An **experimental evaluation**, where we test an actual hyperparameter tuning process using sample data to compare ease of use and which tool creates the best final model

Code Review

For the code review, we will look at the GitHub pages and documentation sites for each of the tools. We will consider both features involving general repository health, as well as more code-specific items such as non-functional requirements.

For general repository health we will look at the following features of each GitHub repository:

- **Popularity/Use** – how many stars and active users does it have?
- **Community** – how active is the community (how many contributors are there, pull requests, etc).
- **Maintenance** - how often are changes made to the main branch? When was the last release? Are issues/pull requests getting handled or left behind?
- **Dependencies** – how many software dependencies does each project have?
- **Security** – are there any potential security vulnerabilities?

For non-functional requirements, we will look at the following subset:

- **Understandability** – how easy is it for someone unfamiliar with the code to understand what is going on?
- **Testability/Debuggability** - is the code formatted in such a way that it is easy to test features or debug when a problem occurs (modular, throws exceptions, etc)? What does the current test coverage look like?
- **Extensibility/Scalability** - is it easy to add on new features or contribute to the project (since it's open source)? Are the existing features written/provided in such a way that it is easy to implement the code at scale, such as in some sort of production environment at a company, or are external libraries or new features needed?

For each of these categories we will rate the repository's performance on a scale from 1-10. This score is subjective and will be backed up by arguments for both what the repository is doing well, and where it could be improved. At the end, we will share a table summarizing the ratings in each category to compare each tool's strengths and weaknesses (code-wise). We will also sum up the scores to determine an overall best tool based on code quality.

We will also conduct a UI review, where we look at the UI features available in each tool. For Hyperopt, this is several built in plotting functions and for Optuna, this is a full dashboard. For each, we will discuss both the good and bad elements of design, as well as how well those features assist in the parameter tuning process.

Experimental Evaluation

For the experimental evaluation component, we will be comparing three separate options for Hyperparameter tuning: 1) Standard grid search, 2) Using Hyperopt, 3) Using Optuna.

We include grid search to add a second point of comparison – external tool vs features available within the standard machine learning models in Python (rather than solely Tool #1 vs Tool #2).

For each tuning technique, we will run a full machine learning tuning process *two* separate times, once for a classification problem and once for a regression problem. This means a total of 6 different conditions.

For our classification condition, we will be using a dataset to predict whether a patient will be readmitted to the hospital based on data from their prior visit (number of days at hospital, medications given, etc.). This data is from Kaggle, and we have cleaned it in order to prepare it for modeling. A sample of some of the columns in this dataset is shown below:

age_decade	time_in_hospital	n_procedures	n_medications	high_glucose	high_A1C	readmitted
70	8	1	18	0	0	0
70	3	2	13	0	0	0
50	5	0	18	0	0	1

We will predict the variable `readmitted` using all of the other variables. We will use the `RandomForestClassifier` from `scikit-learn` as our machine learning model.

In order to compare the performance of each tool on the classification problem, we will use the following metrics. The first three have to do with the actual performance of the model, while the last is more of a usability metric.

- **Accuracy** - How often the model makes the right prediction overall
- **Precision** – For positive predictions, how many were actually positive? (Catches false positives)
- **Recall** – How many true positives we capture through labeling them positive? (Catches false negatives)
- **Tuning Time** – How long the algorithm takes to run and pick the best parameters

For our regression condition, we will be predicting the CO2 emissions of a vehicle based on a variety of features, such as engine size, transmission type, and fuel type. This data is from Kaggle, and we have cleaned it in order to prepare it for modeling. A sample of several columns in the dataset is included below:

engine_size	cyl	fuel_consump_city	fuel_premium	class_compact	class_truck	co2_emissions
2.0	4	9.9	1	1	0	196
2.4	4	11.2	1	1	0	221
1.5	4	6.0	1	1	0	136

We will predict the variable `emissions` using all of the other variables. We will use the `RandomForestRegressor` from `scikit-learn` as our machine learning model.

In order to compare the performance of each tool on the regression problem, we will use the following metrics. The first three have to do with the actual performance of the model, while the last is more of a usability metric.

- **Mean Absolute Error (MAE)** – Average distance between predicted values and true values (in units of problem)
- **Mean Absolute Percentage Error (MAPE)** – Average distance between predicted values and true values (in *percentage* difference)
- **Mean Squared Error (MSE)** – Average *squared* distance between predicted values and true values; punishes outliers more than MAE
- **Tuning Time** – How long the algorithm takes to run and pick the best parameters

In order to keep processes as similar as possible across all of our conditions, we will plan to do the following:

1. For Optuna and Hyperopt, both tools require us to define an “objective” we are trying to maximize or minimize. We will use the same objective for both tools.
2. We will use the same variables in our dataset (all of them) for all conditions. We will also use the same train test split percentage (80/20).
3. For grid search, we have to subjectively select a handful of parameters for the model to try. For our Optuna and Hyperopt models, we provide a starting point range. We will attempt to ensure that the values we selected for our grid search are included in this range to maintain consistency.

In addition to all of the objective metrics we will be computing, we will also provide a short note on how easy or difficult it was to figure out how to use each tool (Optuna and Hyperopt specifically) on our data.

Related Work/References

We will consider several outside resources as we complete our study:

1. A research paper comparing a variety hyperparameter optimization tools as well as describing some mathematical theory behind how they work. This paper discusses both Hyperopt and Optuna.
2. An official research paper written on Optuna.
3. An official research paper written on Hyperopt.
4. Information on how to complete a grid search in scikit-learn and sample code on implementing this with random forest.
5. Sample code for implementing Optuna and Hyperopt:
    - Optuna #1
    - Optuna #2
    - Hyperopt #1
    - Hyperopt #2