Università degli Studi di Milano

Data Science for Economics

Bayesian optimization for hyperparameter tuning

Students:
Davide Matta
Jing Wang

ACADEMIC YEAR 2023-2024

Contents

1 Introduction **3**

2 Methodology **4**

2.1 Data preparation 4

2.2 Frequestist vs Bayesian 4

3 Algorithms **8**

3.1 XGBoost 8

3.2 Ridge Regression 9

3.3 Neural Networks 10

4 Conclusion **12**

5 Bibliography **13**

1 Introduction

Hyperparameter optimization or tuning in machine learning is the task of selecting the best set of hyperparameters for a machine learning model. They are parameters that are set before the learning process begins and influence the way the algorithm learns.
Hyperparameter optimization involves finding a combination of hyperparameters that results in the best-performing model, typically by minimizing a specific loss function on a validation dataset or through cross-validation. This tuning process aims to find the hyperparameters that lead to a model with the lowest error or highest accuracy, depending on the metric of interest.
Cross-validation is frequently used to evaluate how well the model performs with different sets of hyperparameters, allowing the identification of the set that offers the best generalization to unseen data.
The goal of this project is to compare the Bayesian and Frequentist approaches, which represent two distinct philosophies in statistical inference. The former incorporates prior knowledge and treats parameters as random variables with associated probability distributions. This approach is suitable for dealing with limited data and complex probabilistic models; the latter does not incorporate prior information and treats parameters as fixed, unknown constants. Frequentist methods rely on large sample sizes for accurate parameter estimation. We have tried with different algorithms with these two approaches in order to compare their performance and computational efficiency.
The codes is available in the following page of GitHub: GitHub link

2 Methodology

2.1 Data preparation

In this project, we decided to generate synthetic data using scikit−learn's make classification function, setting up the following parameters:

- n_samples=10000: the total number of samples to generate;
- n_features=20: the total number of features for each sample;
- n_informative=10: the number of informative features; These are the features that contribute to the separation between classes;
- n_redundant=5: these are additional features that are generated as random linear combinations of the informative features;
- n_clusters per class=2: the number of clusters per class. This parameter defines how many subgroups each class will have;
- weights=0.7: the proportions of samples assigned to each class. Here, 0.7 indicates that 70% of the samples will belong to one class, and the remaining 30
- class_sep=0.8: the separation between classes. Higher values result in classes that are more easily separable;

We added some random noise to each element of the generated feature matrix, where each row represents a sample and each column represents a feature. This can be useful for simulating data with some level of randomness or for adding noise to existing data. Each sample is labelled. We divided the data into two parts: 80% for the training set and 20 % for the test set.

2.2 Frequestist vs Bayesian

Our objective is to identify the optimal hyperparameters for our machine learning model, aiming to achieve the highest predictive performance. To accomplish this, we will explore two distinct methodologies: frequentist and Bayesian-based approaches.
Following are the three methods of hyperparameter optimization of machine

learning that we used in order of increasing efficiency:
1. Grid search
2. Random search
3. Bayesian model-based optimization

Grid search is a technique used in machine learning to look through a manually specified subset of the hyperparameter space exhaustively. Unlike random search, which samples hyperparameters randomly, grid search evaluates all possible combinations of them within the specified grid.

Grid search is particularly suitable when the search space is relatively small and the hyperparameters are equally important. It ensures that all combinations of hyperparameters are explored, guaranteeing that the best combination within the search space is found.

Random search is particularly useful when the search space is large or when certain hyperparameters are more influential than others. Since it samples them randomly, it has a higher chance of exploring regions of the search space that might be missed by the previous technique.

Grid and Random search pay no attention to the past results at all and would keep searching across the entire range of the number of estimators even though it is clear the optimal answer (probably) lies in a small region! Bayesian approach, in contrast to random or grid search, keeps track of past evaluation results that it uses to form a probabilistic model mapping hyperparameters to a probability of a score on the objective function, which guides the tuning process. Typically, it takes a set of hyperparameters as input and outputs a measure of the model's performance. In our case, it leads to find the set of hyperparameters with the highest accuracy.

$$p(\text{score} \mid \text{hyperparameters})$$

In the literature, the model used as a proxy for the objective function is commonly referred to as a "surrogate." This surrogate model is denoted as $p(y|x)$, where y represents the target variable (e.g., prediction or performance metric) and x represents the hyperparameters. The surrogate model serves as a simplified and computationally tractable approximation of the true objective function, which may be complex or computationally expensive to evaluate directly.

Bayesian methods leverage this surrogate model to guide the hyperparameter optimization process. Instead of directly optimizing the objective function, Bayesian optimization algorithms focus on finding the next set of hyperparameters to evaluate based on the surrogate model's predictions. Specifically,

they select hyperparameters that are expected to perform best on the surrogate function, thus guiding the search towards regions of the hyperparameter space that are likely to yield improved performance on the true objective function.
Sequential model-based global optimization (SMBO) methods are a formalization of Bayesian optimization. It involves building a surrogate model to approximate the objective function and use it to guide the search for optimal hyperparameters. The optimization process is iterative, updating the surrogate model with new observations and exploring the parameter space in a data-driven manner.
Tree-structured Parzen Estimator (TPE) is a specific type of surrogate model used in SMBO. It constructs probabilistic models to represent the relationship between hyperparameters and their corresponding performance. Unlike direct optimization, TPE uses these models to determine which hyperparameters to try next, making the search process more efficient.It models the probability density of good and bad configurations, allowing it to guide the search for optimal hyperparameters.
The TPE builds a model by applying Bayes rule. Instead of directly representing $p(x|y)$, it instead uses:

$$P(y|x) = \frac{P(x|y) \times P(y)}{P(x)} \tag{1}$$

$p(x|y)$, which is the probability of the hyperparameters given the score on the objective function, in turn is expressed:

$$p(x|y) = \begin{cases} l(x) & \text{if } y < y^* \\ g(x) & \text{if } y \geq y^* \end{cases} \tag{2}$$

where l(x) is the density formed by using the observations x$_i$, such that the corresponding loss function f(x$_i$) was less than y^* and g(x) is the density formed by using the remaining observations.
With Bayes Rule, and a few substitutions, the expected improvement equation (which is the criteria by which the next set of hyperparameters are chosen from the surrogate function and we are trying to maximize) becomes:

$$EI_{y^*}(x) = \frac{\gamma y^* l(x) - l(x) \int_{y^*}^{R} p(y)\, dy}{\gamma l(x) + (1-\gamma) g(x)} \propto (\gamma + \frac{g(x)}{l(x)}(1-\gamma))^{-1} \tag{3}$$

It shows that to maximize improvement we would like points x with high probability under l(x) and low probability under g(x). The tree-structured form of l and g makes it easy to draw many candidates according to l and evaluate them according to g(x)/l(x). On each iteration, the algorithm returns the candidate x^* with the greatest EI.

3 Algorithms

3.1 XGBoost

The first algorithm that we performed on the synthetic data is called XGBoost (eXtreme Gradient Boosting). It is a machine learning technique that builds a predictive model by combining the predictions of multiple individual models, typically decision trees, in an ensemble.
XGBoost uses a customizable objective function that measures the model's performance and guides the optimization process. It also includes several regularization techniques to prevent overfitting and improve generalization performance. These techniques include shrinkage (learning rate), maximum depth of trees and gamma (minimum loss reduction required to make a further partition on a leaf node).
We defined a grid of hyperparameters for tuning this model as following:

- n_estimators: [100, 200, 300, 400, 500],
- max_depth: [3, 6, 9, 12, 15],
- learning_rate: [0.1, 0.01, 0.001, 0.0001].

The values provided represent the ranges of hyperparameters.
The first represents the number of trees in the ensemble.The second controls the maximum depth of each tree.The last one controls the step size at each iteration while moving towards a minimum of the loss function.
We also defined a search space for hyperparameter optimization for Bayesian model, using 'Hyperopt' python library:

- n_estimators: hp.randint('n_estimators', 50, 500),
- max_depth: hp.randint('max_depth', 2, 15),
- learning_rate: hp.loguniform('learning_rate', np.log(0.0001), np.log(0.1))

We established the objective function, which plays a crucial role in discovering the optimal hyperparameters for the XGBoost model. It achieves this by exploring various hyperparameter combinations, training and assessing the model's performance on each configuration. Then, it returns the negative accuracy score, which optimization algorithms aim to minimize.
As we stated in the previous paragraph, we performed grid search, random search and Bayesian based approach to find the optimal hyperparameters of

the XGBoost model for our classification task.
We applied cross validation for choosing the best combination (the same applies to the other algorithms too).
The outcomes are presented in Table 1.
As we can see the combination of hyperparameters found by the the three

Table 1: Summary of Hyperparameter Optimization Results - XGBoost

Method	**Learn. Rate**	**Max Depth**	**N Estim.**	**Accuracy**	**Time**
Grid Search	0.1	9	500	96.45%	7 min 47s
Random Search (20)	0.1	15	500	96.60%	1 min 18s
Bay Opt (50)	0.064	9	437	96.55%	5 min 12s
Bay Opt (20)	0.029	9	443	96.40%	2 min 10s

methods are different between them, but the performance of the models is similar, comparing the test accuracies, they are all around 96.50 % which indicate that they are all performing quite well in terms of the classification task. With three hyperparameters, the grid search needs to explore all possible combinations of these hyperparameters, it results as the most computationally intensive method. While random search, instead of exhaustively evaluating all combinations, selects parameter sets randomly. This randomness allows it to cover a broader range of the search space more quickly, potentially finding good combinations with fewer evaluations.
Bayesian optimization, on the other hand, is quicker than grid search since it requires to estimate a smaller number of combinations, but it is slower than random search even if they run the same number of models, because it requires the additional step of maximizing the objective function.

3.2 Ridge Regression

The second algorithm performed is ridge regression, which is also known as L2 regularization or Tikhonov regularization. It is a linear regression technique used to address multicollinearity (high correlation between predictor variables) and overfitting in models.
The strength of regularization in ridge regression is controlled by the tuning parameter (alpha). A higher value of alpha results in stronger regularization and more significant shrinkage of coefficients.
After standardizing all the features that we have in our dataset, we defined the grid of hyperparameters as following:

- alpha: [0.01, 0.1, 1.0, 10.0, 100.0, 1000.0]
- alpha: hp.loguniform('alpha', np.log(0.01), np.log(1000))

The former provides a list of alpha values for grid search and the latter for Bayesian model based optimization. Since in this case we have only one hyperparameter to be optimized and the algorithm itself is quick, there is no need to apply the random search method. For the same reason, we run Bayesian optimization with a higher number of evaluations. We obtained the outcomes shown in Table 2.

Table 2: Summary of Hyperparameter Optimization Results - Ridge Regression

Method	**Alpha**	**Accuracy**	**Time**
Grid Search	0.01	85.05	267 ms
Bayesian Optimization (100)	46.265	85.10	5.58 s

The optimal hyperparameter is different in the two approaches, the Bayesian optimization achieved an accuracy of 85.10% on the test set, which is slightly better than the grid search performance, but the time spent in the computation is of course higher as number of evaluations is greater.

3.3 Neural Networks

The last algorithm performed is neural networks. They are composed of interconnected nodes, called neurons or units, organized into layers. They can perform computations in parallel, enabling them to handle large amounts of data efficiently. To prevent overfitting and improve generalization performance. As in the previous steps, we defined the hyperparameters to be optimized:

- hidden_dim = trial.suggest_int('hidden_dim', 32, 256)
- lr = trial.suggest_loguniform('lr', 1e-4, 1e-2)
- batch_size = trial.suggest_categorical('batch_size', [32, 64, 128])

The outcomes are shown in Table 3. This time, the Optuna library was used instead of Hyperopt, since it is better integrated with the neural networks tool PyTorch.
We excluded grid search as it is not commonly used in this context. Also, in

Table 3: Summary of Hyperparameter Optimization Results - Neural Networks

Method	Hidden Dim	Lear Rate	Batch Size	Accuracy	Time
Random Search	224	0.00119	32	96.65	2min 21s
Bayesian Optimization	220	0.0048	128	96.00	2min 28s

neural networks random search is not applied using a grid but a search space as in Bayesian optimization.

We have in the two cases distinct combinations of optimized hyperparameters, the models are performing very well on this classification task, achieving 96.65% and 96.00% of test accuracy with similar timing spent in the computation.

4 Conclusion

The experiment showed that the three optimization methods had similar performances with all the three algorithms, which indicates that, as long as we try many different hyperparameters combinations, it is likely that we fall close to the optimal one, regardless of how we get there.
Given a fixed number of combinations to test, the Bayesian approach is slower: we expected it because it requires additional steps when selecting the next set to test.
However, if the ML algorithm has many hyperparameters, the Bayesian approach with a smaller number of combinations may be preferable to the grid search as the XGBoost example shows.
With respect to the computation times, it is worth noting that they also depend on hardware availability and on the efficiency of the libraries in implementing the optimization methods.
In conclusion, the Bayesian optimization, and in particular TPE, has proven capable to perform better than frequentist approaches, especially as the search space becomes larger and as we increase the number of evaluations.

5 Bibliography

1) W. Koehrsen, “A conceptual explanation of bayesian hyperparameter optimization for machine learning,” 2018.
2) J.Bergstra, R. Bardenet, Y. Bengio, B. Kégl, ”Algorithms for Hyper-Parameter Optimization”.