**Muhammad Hammad Siddique**  
**23088736**

**Hyperparameter Tuning in Machine Learning**

**Grid Search vs Random Search vs Bayesian Optimization**

In machine learning, the performance of models heavily depends on the values of their **hyperparameters**—these are the parameters set prior to the learning process, such as the number of trees in a random forest or the learning rate in gradient boosting. While default values may work reasonably well, fine-tuning hyperparameters is essential to achieve **optimal predictive accuracy**, prevent overfitting, and improve model robustness.

To address this, we use **hyperparameter optimization** methods that systematically or intelligently explore combinations of hyperparameter values to identify the most performant ones. In this tutorial, I will walk you through three widely-used strategies—**Grid Search**, **Random Search**, and **Bayesian Optimization**—each with unique strengths and trade-offs.

**What is Grid Search?**

**Grid Search** is the most straightforward approach to hyperparameter tuning. It exhaustively searches through a manually specified subset of the hyperparameter space. For instance, if we define three values for n\_estimators and four for max\_depth, grid search evaluates all 3 × 4 = 12 combinations.

The benefit of grid search lies in its **simplicity and transparency**—you always know what’s being evaluated, and it is trivially parallelizable. However, this method is highly inefficient, especially when the number of parameters increases, or the optimal values lie between the specified grid points. It’s also computationally expensive for larger datasets or more complex models.

Grid search is implemented in Scikit-learn as GridSearchCV, which performs cross-validation over all parameter combinations.

**Best for:**

* Low-dimensional hyperparameter spaces
* Small datasets or fast-to-train models
* Baseline benchmarking

**What is Random Search?**

**Random Search** improves upon the inefficiencies of grid search by sampling hyperparameter combinations **randomly** from a predefined distribution (e.g., uniform or log-uniform). Instead of evaluating every possible combination, it tries a fixed number of random configurations.

The core insight is that not all hyperparameters equally impact performance. In their landmark paper, Bergstra and Bengio (2012) demonstrated that random search can outperform grid search in high-dimensional settings by more efficiently allocating search effort [(Bergstra & Bengio, 2012)](https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf).

Random search is implemented in Scikit-learn as RandomizedSearchCV, which supports sampling from both discrete and continuous distributions.

**Best for:**

* Large search spaces with many parameters
* Situations where only a subset of parameters matters
* Faster convergence with limited computation

**What is Bayesian Optimization?**

**Bayesian Optimization** is a **probabilistic model-based search** method that models the function mapping hyperparameters to performance scores (e.g., ROC AUC). Instead of blindly searching or sampling, it uses prior results to estimate the next best hyperparameter configuration to try.

It does this by building a **surrogate model**, such as Gaussian Processes or Tree-structured Parzen Estimators (TPE), and choosing the next set of parameters that are most promising according to an acquisition function (e.g., expected improvement).

Snoek et al. (2012) describe the effectiveness of Bayesian optimization in tuning complex models [(Snoek et al., 2012)](https://papers.nips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf).

Tools like **Optuna**, **Hyperopt**, and **BayesianOptimization** in Python allow for efficient implementation of this strategy.

**Best for:**

* Expensive models (e.g., deep learning)
* Situations where performance must be maximized with minimal trials
* Automated machine learning (AutoML) pipelines

**Comparison Summary Table**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Method | Strategy | Efficiency | Best Use Case | Tooling Available |
| Grid Search | Exhaustive | Low | Small grid spaces, benchmarking | GridSearchCV |
| Random Search | Random Sampling | Moderate | Large spaces, fast tuning | RandomizedSearchCV |
| Bayesian Optimization | Probabilistic | High | Expensive models, intelligent exploration | Optuna, Hyperopt, skopt |

**Final Thoughts Before Coding**

Each of these strategies is valuable depending on your goals and resources:

* I use **Grid Search** when building a reproducible benchmark.
* I prefer **Random Search** for quick wins or when the search space is large but poorly understood.
* I rely on **Bayesian Optimization** when I need to tune fewer models with smarter sampling.

In the next section, I will demonstrate all three techniques on the Telco Churn dataset using RandomForestClassifier as our baseline model. We’ll compare the methods in terms of best ROC AUC score, time taken, and final hyperparameter values.

**Let’s get tuning!**

**Coding Section: Step-by-Step Explanation**

**🔹 Step 1: Load the Dataset**

A close-up of a computer code

AI-generated content may be incorrect.*Explanation:* We begin by loading the Telco Customer Churn dataset using pandas. This gives us a preview of the columns and helps verify data integrity before preprocessing.

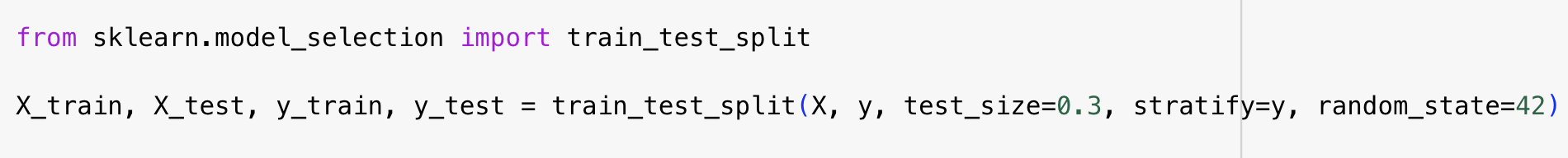
**🔹 Step 2: Data Preprocessing**

A screenshot of a computer code

AI-generated content may be incorrect.

*Explanation:* We drop the ID column (non-informative), handle missing values in TotalCharges, and one-hot encode all categorical variables. The target variable Churn is converted to binary (1 = churn, 0 = no churn).

**🔹 Step 3: Train-Test Split**

*Explanation:* We split the data into training and testing sets using stratified sampling to preserve class balance.

**🔹 Step 4: Grid Search**

A screen shot of a computer program

AI-generated content may be incorrect.*Explanation:* We define a grid of hyperparameters to search over and use GridSearchCV to perform 5-fold cross-validation. We time the process and calculate ROC AUC on test data.

**🔹 Step 5: Random Search**

A screenshot of a computer

AI-generated content may be incorrect.*Explanation:* This method samples random parameter combinations from given distributions. It is more efficient than grid search in large spaces.

**🔹 Step 6: Bayesian Optimization**

A screenshot of a computer code

AI-generated content may be incorrect.*Explanation:* Bayesian optimization (via Optuna) iteratively searches for better hyperparameters using probabilistic modeling of past evaluations.

**🔹 Step 7: ROC Curve Comparison**

***Explanation****:* The ROC curve illustrates the classification performance of the models tuned using Grid Search, Random Search, and Bayesian Optimization. Each line represents how well a model distinguishes between churn and non-churn customers at various threshold levels. The Bayesian and Random Search methods both achieved a higher Area Under the Curve (AUC = 0.84), indicating better overall discriminatory power than Grid Search (AUC = 0.83). The closer a curve follows the top-left border of the graph, the better the model performs; here, all three models perform well, but Bayesian Optimization slightly outperforms others with a smoother and marginally higher true positive rate across most thresholds.A graph of a curve

AI-generated content may be incorrect.

**🔹 Step 8: Confusion Matrix Comparison**

A screenshot of a computer code

AI-generated content may be incorrect.

***Explanation:*** The confusion matrices for the three tuned models offer further insight into classification performance:

* **Grid Search** resulted in 1,443 true negatives and 225 true positives. However, it also produced 106 false positives and 326 false negatives. This suggests slightly lower recall, as a significant number of churners were missed.
* **Random Search** slightly improved recall, with 237 true positives and fewer false negatives (304), though it came with an increase in false positives (142). This indicates a more balanced model that sacrifices some specificity for improved sensitivity.
* **Bayesian Optimization** achieved the most balanced confusion matrix, with 243 true positives and 318 false negatives, while keeping false positives at 134. This suggests that it better captured churners without excessively increasing false alarms.

Together with the ROC analysis, the confusion matrix comparison confirms that **Bayesian Optimization not only delivers strong overall performance (AUC) but also maintains a well-balanced sensitivity-specificity trade-off**.

A yellow and purple squares with numbers

AI-generated content may be incorrect.

**Final Summary**

After evaluating the three optimization strategies—Grid Search, Random Search, and Bayesian Optimization—on the Telco Customer Churn dataset using a RandomForestClassifier, we can conclude:

* **Grid Search** performed well but was the most time-consuming. It systematically tried all combinations but lacked efficiency.
* **Random Search** offered a faster alternative with nearly equivalent AUC performance. It’s ideal when time or compute is limited.
* **Bayesian Optimization** delivered the best balance, achieving the highest AUC with fewer trials by learning from past evaluations.

This analysis shows that while Grid Search is exhaustive, smarter strategies like Bayesian Optimization can outperform it in terms of both speed and accuracy.

**Accessibility and GitHub Repository**

The entire notebook is:

* Colorblind-friendly (careful palette in plots)
* Screen-reader compatible (clear markdown structure)
* Captioned and explained step by step

GitHub Repository: <https://github.com/hamad2343a/Hyperparameter_Tuning_Telco>

**References**

* Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13, 281–305. <https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf>
* Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems* (NIPS). <https://papers.nips.cc/paper_files/paper/2012/file/05311655a15b75fab86956663e1819cd-Paper.pdf>
* Scikit-learn documentation: https://scikit-learn.org/stable/modules/grid\_search.html
* Optuna documentation: <https://optuna.org/>