A Comprehensive Guide on Hyperparameter Tuning and its Techniques

Shanthababu Pandian

Last Updated : 14 Nov, 2024

13 min read

55

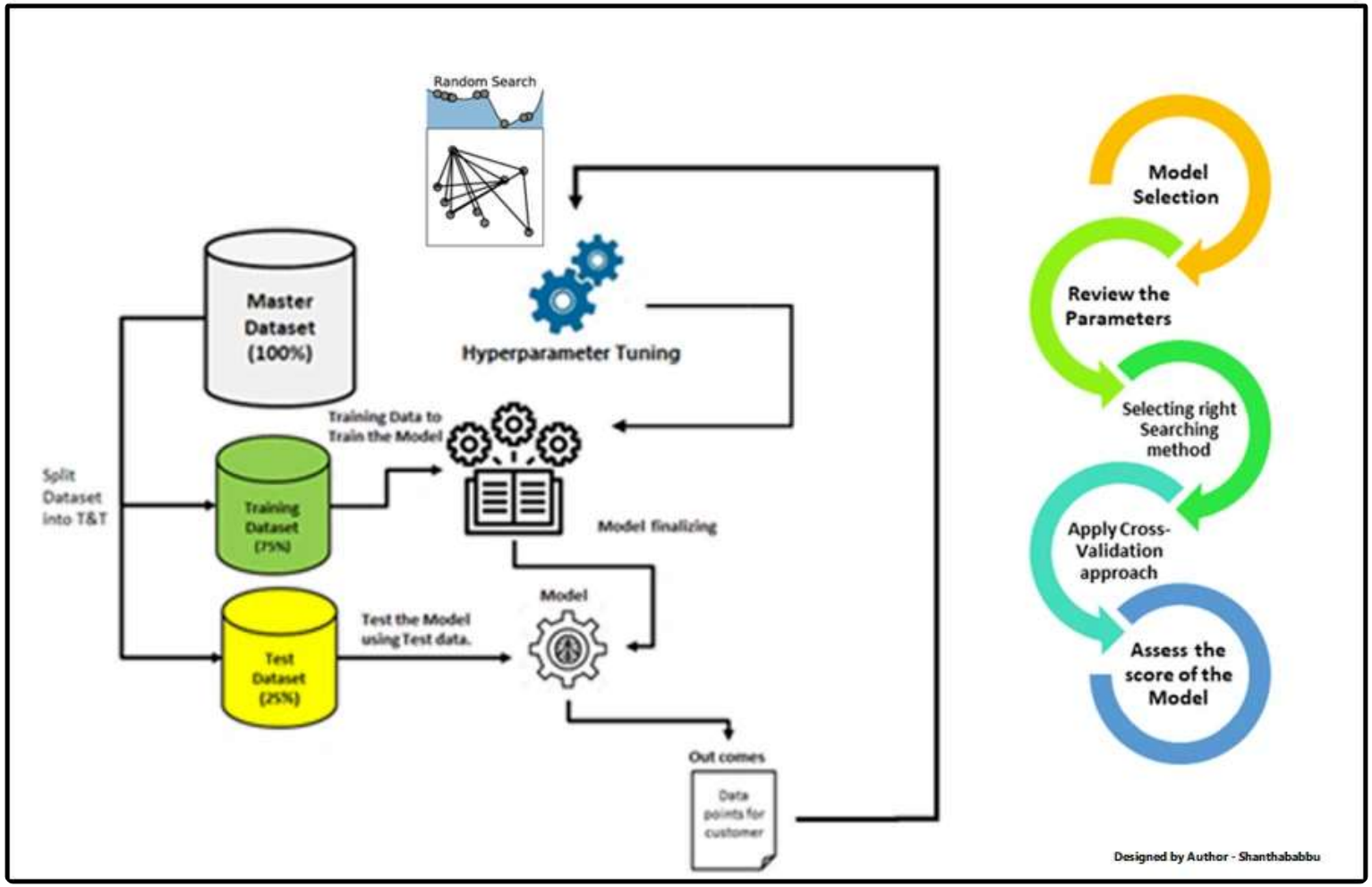
Every ML Engineer and **Data Scientist** must understand the significance of “Hyperparameter Tuning Techniques” while selecting your right machine/deep learning model and improving the performance of the model(s).

Make it simple, for every single machine learning model selection is a major exercise and it is purely dependent on selecting the equivalent set of hyperparameters, and all these are indispensable to train a model. It is always referring to the parameters of the selected model and be remember it cannot be learnt from the data, and it needs to be provided before the model gets into the training stage, ultimately the performance of the machine learning model improves with a more acceptable choice of hyperparameter tuning in machine learning and selection techniques. The main intention of this article is to make you all aware of hyperparameter tuning.

Hyperparameter tuning is basically referred to as tweaking the parameters of the model, which is basically a prolonged process.

Before going into detail, let’s ask some valuable self-questions on hyperparameter tuning in machine learning, I am sure this would help you a lot on this magic word. Personally, I experienced that and explain it here.

In this article, you will explore hyperparameter tuning, including various hyperparameter optimization techniques and methods. Discover how these hyperparameter tuning methods can significantly enhance your machine learning model's performance.



Learning Objectives:

- Understand the importance of hyperparameter tuning for machine learning models.
- Learn the difference between hyperparameters and model parameters.

- Explore various hyperparameter tuning techniques like GridSearchCV, RandomSearchCV, manual search.
- Understand how to prevent data leakage during model training and tuning.

This article was published as a part of the Data Science Blogathon.

Table of contents

1. What are Hyperparameters and How Do They Differ from Model Parameters?
2. ML Life Cycle: Hyperparameter Tuning and its Techniques
3. Hyperparameter Space
4. Data Leakage
5. Steps to Perform Hyperparameter Tuning
    - Train, Test Split Estimator
    - Logistic Regression Classifier
    - KNN (k-Nearest Neighbors) Classifier
    - Support Vector Machine Classifier

What are Hyperparameters and How Do They Differ from Model Parameters?

As we know that there are parameters that are internally learned from the given dataset and derived from the dataset, they are represented in making predictions, classification and etc., These are so-called **Model Parameters,** and they are varying with respect to the nature of the data we couldn't control this since it depends on the data. Like '**m**' and '**C**' in linear equation, which is the value of coefficients learned from the given dataset.

Some set of parameters that are used to control the behaviour of the model/algorithm and adjustable in order to obtain an improvised model with optimal performance is so-called **hyperparameter tuning in machine learning.**

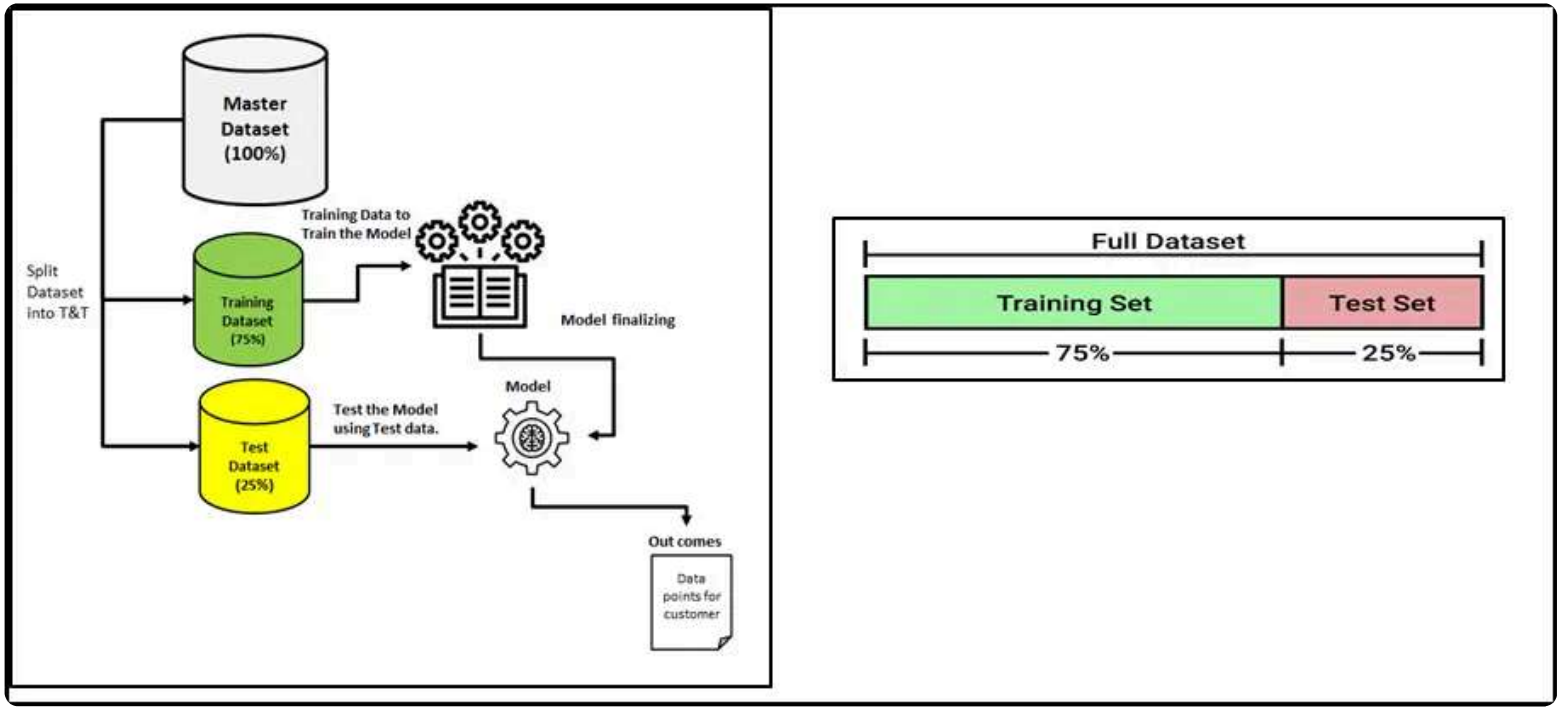
The best model algorithm(s) will sparkle if your best choice of Hyper-parameters.

Checkout this article about the Parameters and Hyperparameters

ML Life Cycle: Hyperparameter Tuning and its Techniques

If you ask me what is **Hyperparameters** in simple words, the one-word answer is **Configuration.**

Without thinking too much, I can say quick Hyperparameter is "**Train-Test Split Ratio (80-20)"** in our simple linear regression model.

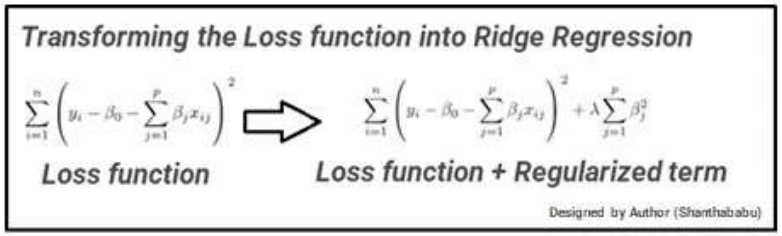


YES! now I can see that, you're really starting to feel what could be HPs and how it would optimize the model. That's why I have mentioned earlier in easy language this is **configuring values**.

Let me give one more example – You can compare this with selecting setting the **font** and its **size** for better readability and clarity while you document your content to be perfect and precise.

Coming back to machine learning and recalling Ridge Regression (L2 Regularization) and Lasso Regression (L1 Regularization), In regularized terms we use to have lambda (λ) I mean the Penalty Factor helps us to get a smooth surface instead of an irregular graph.

This term is used to push the coefficients(β) values near zero in terms of magnitude, For more details please refer to my earlier articles and tutorials on **Study of Regularization Techniques of Linear Models and Its Roles.** This is nothing but hypermeters, crucial for optimizing **model performance.**



For better clarity and understanding, here is one more classical representation for you.

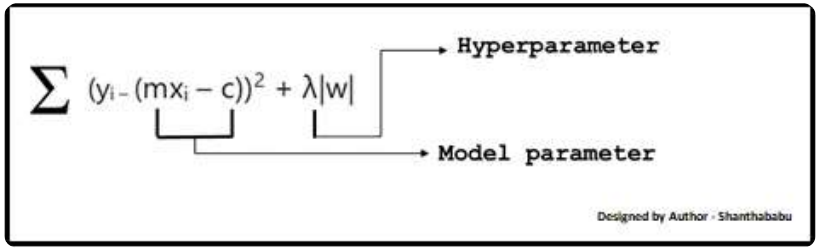


Image designed by the author – Shanthababu

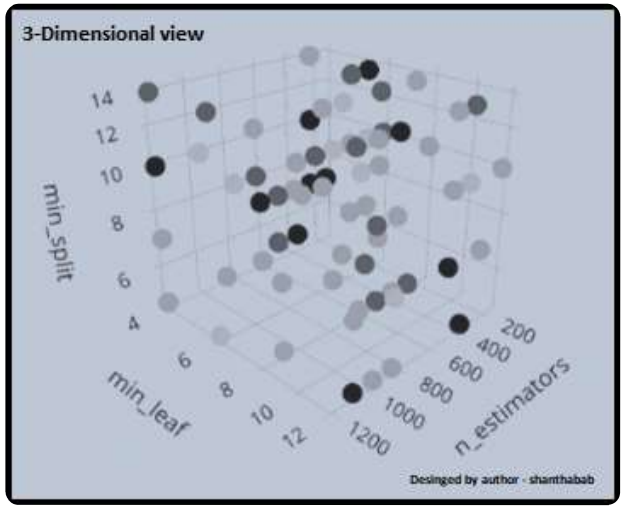
From the above equation, you can understand a better view of **what MODEL and HYPER PARAMETERS** is.

Hyper parameter tuning are supplied as arguments to the model algorithm during initializing them as key, value and their values are picked by the data scientist, who is building the model in iterative mode.

Hyperparameter Space

As we know that there is a list of HPs for any selected algorithm(s) and our job is to figure out the best combination of HPs and to maximize the optimal results by tweaking them strategically, this process will be providing us with the platform for Hyperparameter Space and this combination leads to provide the best optimal results, no doubt in that but finding this combo is not so easy, we have to search throughout the space.

Here every combination of selected HP value is said to be the "**MODEL**" and have to evaluate the same on the spot. For this reason, there are two generic approaches to search effectively in the HP space are **GridSearch CV** and RandomSearch CV, and newer methods like Hyperband are gaining popularity due to their efficiency. Here CV denotes **Cross-Validation**.



Before going to apply the above-mentioned search options on the data/model, we must split the data into ***3 different sets***. I can understand your mind voice, already we are splitting the dataset as Train and Test, now one more track? Yes, there is a valid reason are there, that is nothing but to prevent the "**DATA LEAKAGE"** during Training, Validating and Testing. remember we shouldn't touch the test data set until we move the model into production deployment.

Data Leakage

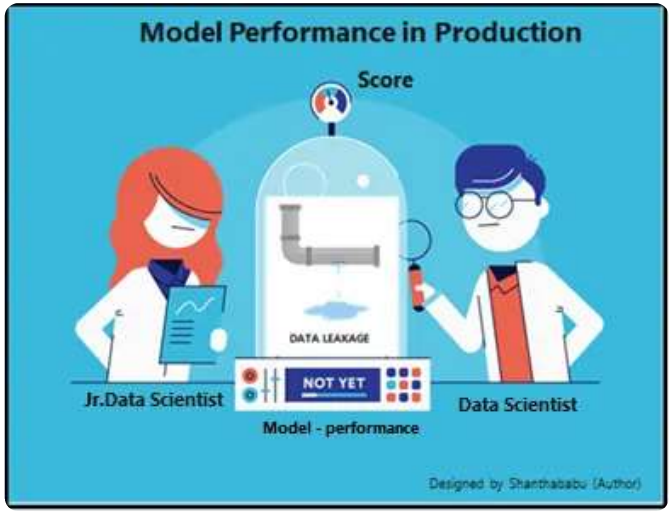
Well! Now quickly will understand what is Data leakage in ML, this is mainly due to not following some of the recommended best practices during the **Data Science/Machine Learning** life cycle. The resulting

is Data Leakage, that's fine, what is the issue here, after successful testing with perfect accuracy followed by training the model then the model has been planned to move into production. At this moment ALL Is Well.

Still, the actual/real-time data is applied to this model in the production environment, you will get poor scores. By this time, you may think that why did this happen and how to fix this. This is all because of the data that we split data into training data and testing subsets. During the training the model has the knowledge of data, which the model is trying to predict, this results in inaccurate and bad prediction outcomes after the model is deployed into production.

Causes of Data Leakage

- Data Pre-processing
- The major root cause is doing all **EDA processes** before splitting the dataset into test and train
- Doing straightforward normalizing or rescaling on a given dataset
- Performing Min/Max values of a feature
- Handling missing values without reserving the test and train
- Removing outliers and Anomaly on a given dataset
- Applying standard scaler, scaling, assert normal distribution on the full dataset

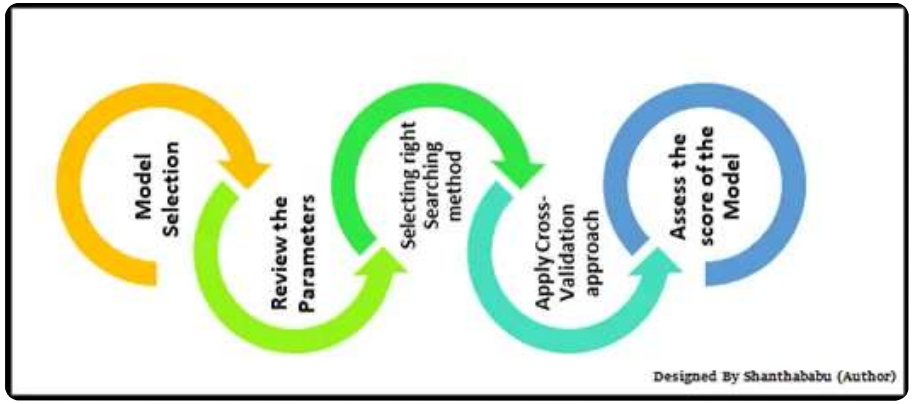


The bottom line is that we should avoid doing anything to our training dataset that involves having knowledge of the test dataset to ensure our model performs as a generalized model in production.

We will go through the available hyperparameters across various algorithms and discuss how to implement these factors to impact the model effectively.

Steps to Perform Hyperparameter Tuning

- Select the right type of model.
- Review the list of parameters of the model and build the HP space
- Finding the methods for searching the hyper parameter tuning
- Applying the cross-validation scheme approach
- Assess the model score to evaluate the model



Now, time to discuss a few hyperparameter tuning in machine learning and their influence on the model.

Train, Test Split Estimator

With the help of this, we use to set the test and train size for the given dataset and along with random state, this is permutations to generate the same set of splits., otherwise you will get a different set of test and train sets, tracing your model during evaluation is bit complex or if we omitted this system will generate this number and leads to unpredictable behaviour of the model. The random state provides the seed, for the random number generator, in order to stabilize the model.

```
train_test_split( X, y, test_size=0.4, random_state=0)
```

Copy Code

Logistic Regression Classifier

The parameter C in **Logistic Regression Classifier** is directly related to the regularization parameter λ but is inversely proportional to C=1/λ.

LogisticRegression(C=1000.0, random_state=0)LogisticRegression(C=1000.0, random_state=0)

KNN (k-Nearest Neighbors) Classifier

As we know the **k-nearest neighbour's algorithm (KNN)** is a non-parametric method used for regression and classification problems. Predominantly this is used for classification problems, in which the number of neighbours and power parameter.

```
KNeighborsClassifier(n_neighbors=5, p=2, metric='minkowski')
```

Copy Code

- n_neighbors is the number of neighbors.
- p is the Minkowski power parameter. When p = 1, it's equivalent to the Manhattan distance, and when p = 2, it represents the Euclidean distance.

Support Vector Machine Classifier

```
SVC(kernel='linear', C=1.0, random_state=0)
```

Copy Code

- The ‘kernel’ parameter specifies the type of kernel to be used in the selected algorithm. For linear classification, ‘kernel’ is set to ‘linear’, while for non-linear classification, it’s set to ‘rbf’.
- C’ represents the penalty parameter, which controls the trade-off between smooth decision boundaries and classifying training points correctly.
- ‘random_state’ is a pseudo-random number generator used to ensure reproducibility of results across different runs.

Decision Tree Classifier

Here, the criterion is the function to measure the quality of a split, max_depth is the maximum depth of the tree, and random_state is the seed used by the random number generator.

DecisionTreeClassifier(criterion=’entropy’, max_depth=3, random_state=0)

Lasso Regression

Lasso(alpha = 0.1) the regularization parameter is alpha.

Principal Component Analysis

```
PCA(n_components = 4)
```

Copy Code

Perceptron Classifier

Perceptron (n_iter=40, eta0=0.1, random_state=0)

- n_iter is the number of iterations,
- eta0 is the learning rate,
- random_state is random number generator.

Understand More about the [Machine Learning Algorithms]

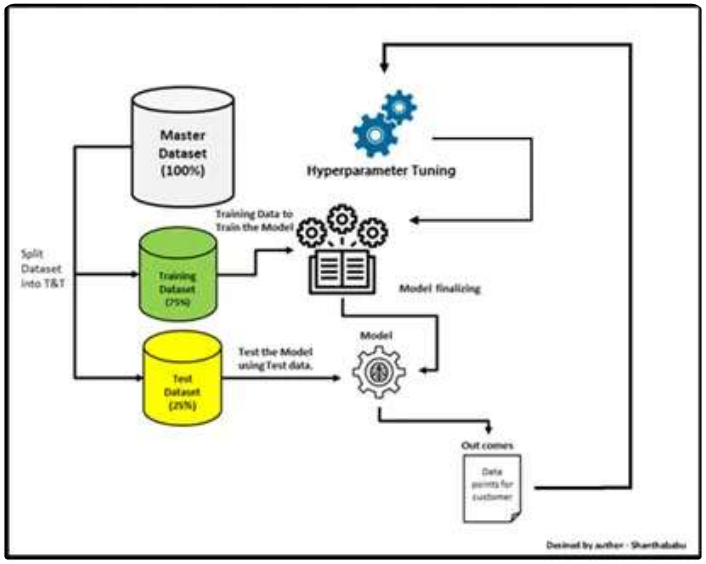
Influencing on Models

Overall, Hyper parameter tuning are influencing the below factors while designing your model. Please remember this.

- Linear Model
  - What degree of polynomial features should use?
- Decision Tree
  - What is the maximum allowed depth?
- What is the minimum number of samples required at a leaf node in the decision tree?
  - Random forest
- How many trees we should include?
  - Neural Network
  - How many neurons we should keep in a layer?
- How many layers, should keep in a layer?
  - Gradient Descent

- What learning rate should we?

So, once we started thinking about introducing the hyper parameter tuning in our model then the overall architecture model would be like below.

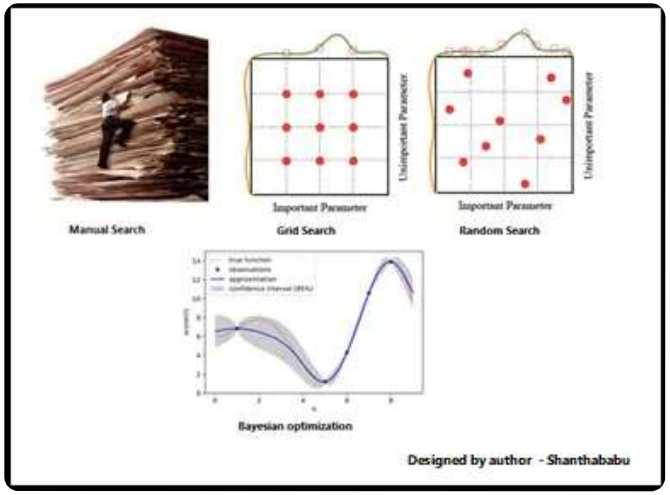


Hyperparameter Optimization Techniques

In the ML world, there are many Hyperparameter optimization techniques are available.

- Manual Search
- Random Search
- Grid Search
- Halving
  - Grid Search

    - Randomized Search
- Automated Hyperparameter tuning
    - Bayesian Optimization
    - Genetic Algorithms
- Artificial Neural Networks Tuning
- HyperOpt-Sklearn
- Bayes Search



Note: When we implement Hyperparameters optimization techniques, we have to have the Cross-Validation techniques as well in the flow because we may not miss out on the best combinations that work on tests and training.

Manual Search

The name itself is self-explanatory that the data scientist can do the experiment with different combinations of hyperparameters and its values for the selected model perform the training and pick up the best model with the best performance and go for testing and move on

to production deployment. Of Course, what you think is absolutely right is that this method will consume immense effort. Utilizing various machine learning frameworks enhances this iterative process significantly.

Let's try this with a simple dataset. This example will serve as a practical tutorial on applying hyperparameter tuning to real-world data.

Copy Code

```
# Import necessary packages
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
df = pd.read_csv("pima-indians-diabetes.csv")
print(df.head())
```

Dataframe ready after load CSV and required libraries for further operations

Copy Code

```
# Train Test Split
#df = df.drop(['name','origin','model_year'], axis=1)
y = df['class']
X = df.drop(['class'],axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=30)
```

Train and Test are done with target and dependent variables identification.

Since we're planning for manual search, I am creating 3 sets for DecisionTreeClassifier and fitting the model:

Copy Code

```
# sets of hyperparameters
params_1 = {'criterion': 'gini', 'splitter': 'best', 'max_depth': 50}
params_2 = {'criterion': 'entropy', 'splitter': 'random', 'max_depth': 70}
params_3 = {'criterion': 'gini', 'splitter': 'random', 'max_depth': 60}
params_4 = {'criterion': 'entropy', 'splitter': 'best', 'max_depth': 80}
params_5 = {'criterion': 'gini', 'splitter': 'best', 'max_depth': 40}
# Separate models
model_1 = DecisionTreeClassifier(**params_1)
model_2 = DecisionTreeClassifier(**params_2)
```

```
model_3 = DecisionTreeClassifier(**params_3)
model_4 = DecisionTreeClassifier(**params_4)
model_5 = DecisionTreeClassifier(**params_5)
model_1.fit(X_train, y_train)
model_2.fit(X_train, y_train)
model_3.fit(X_train, y_train)
model_4.fit(X_train, y_train)
model_5.fit(X_train, y_train)
# Prediction sets
preds_1 = model_1.predict(X_test)
preds_2 = model_3.predict(X_test)
preds_3 = model_3.predict(X_test)
preds_4 = model_4.predict(X_test)
preds_5 = model_5.predict(X_test)
print(f'Accuracy on Model 1: {round(accuracy_score(y_test, preds_1), 3)}')
print(f'Accuracy on Model 2: {round(accuracy_score(y_test, preds_2), 3)}')
print(f'Accuracy on Model 3: {round(accuracy_score(y_test, preds_3), 3)}')
print(f'Accuracy on Model 4: {round(accuracy_score(y_test, preds_4), 3)}')
print(f'Accuracy on Model 5: {round(accuracy_score(y_test, preds_5), 3)}')
```

Output:

Copy Code

```
Accuracy on Model 1: 0.693
Accuracy on Model 2: 0.693
Accuracy on Model 3: 0.693
Accuracy on Model 4: 0.736
Accuracy on Model 5: 0.688
```

Look at the accuracy and its differences with different parameters that we have passed over the list. But this is a tedious job and running behind a number of permutations and combinations and finding the best one, hope you can understand the pain and code management.

Grid-Search

To implement the Grid-Search, we have a Scikit-Learn library called GridSearchCV. The computational time would be long, but it would reduce the manual efforts by avoiding the ‘n’ number of lines of code. Library itself perform the search operations and returns the performing model and its score. In which each model are built for each permutation of a given hyperparameter, internally it would be evaluated and ranked across the given cross-validation folds.

Let’s implement this with the given dataset.

Getting KNeighborsClassifier object for my operation.

```
from sklearn.neighbors import KNeighborsClassifier
knn_clf = KNeighborsClassifier()
```

Copy Code

Assigning my Train and Test spilt to my KNN object

```
knn_clf.fit(X_train, y_train)
```

Copy Code

Output

```
KNeighborsClassifier()
```

Copy Code

Importing other required libraries

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
```

Copy Code

Defining a number of folders for GridSearchCV and assigning TT.

```
gs = GridSearchCV(knn_clf,param_grid,cv=10)
gs.fit(X_train, y_train)
```

Copy Code

Preparing a list of hyperparameters for my further actions with 4 different algorithm:

```
param_grid = {'n_neighbors': list(range(1,9)),'algorithm': ('auto', 'ball_tree', 'kd_tree' , 'brute') }
```

Copy Code

Output

```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),param_grid={'algorithm': ('auto', 'ball_tree', 'kd_tree', 'brute'),'n_neighbo
```

Copy Code

We will print all 4 algorithms for 8 sub-sets.

```
gs.cv_results_['params']
```

Copy Code

Output 32 combinations

```
[{'algorithm': 'auto', 'n_neighbors': 1},
 {'algorithm': 'auto', 'n_neighbors': 2},
 {'algorithm': 'auto', 'n_neighbors': 3},
 {'algorithm': 'auto', 'n_neighbors': 4},
 {'algorithm': 'auto', 'n_neighbors': 5},
 {'algorithm': 'auto', 'n_neighbors': 6},
 {'algorithm': 'auto', 'n_neighbors': 7},
 {'algorithm': 'auto', 'n_neighbors': 8},
 {'algorithm': 'ball_tree', 'n_neighbors': 1},
 {'algorithm': 'ball_tree', 'n_neighbors': 2},
 {'algorithm': 'ball_tree', 'n_neighbors': 3},
 {'algorithm': 'ball_tree', 'n_neighbors': 4},
 {'algorithm': 'ball_tree', 'n_neighbors': 5},
 {'algorithm': 'ball_tree', 'n_neighbors': 6},
 {'algorithm': 'ball_tree', 'n_neighbors': 7},
```

Copy Code

```
 {'algorithm': 'ball_tree', 'n_neighbors': 8},
 {'algorithm': 'kd_tree', 'n_neighbors': 1},
 {'algorithm': 'kd_tree', 'n_neighbors': 2},
 {'algorithm': 'kd_tree', 'n_neighbors': 3},
 {'algorithm': 'kd_tree', 'n_neighbors': 4},
 {'algorithm': 'kd_tree', 'n_neighbors': 5},
 {'algorithm': 'kd_tree', 'n_neighbors': 6},
 {'algorithm': 'kd_tree', 'n_neighbors': 7},
 {'algorithm': 'kd_tree', 'n_neighbors': 8},
 {'algorithm': 'brute', 'n_neighbors': 1},
 {'algorithm': 'brute', 'n_neighbors': 2},
 {'algorithm': 'brute', 'n_neighbors': 3},
 {'algorithm': 'brute', 'n_neighbors': 4},
 {'algorithm': 'brute', 'n_neighbors': 5},
 {'algorithm': 'brute', 'n_neighbors': 6},
 {'algorithm': 'brute', 'n_neighbors': 7},
 {'algorithm': 'brute', 'n_neighbors': 8}]
```

Let’s get the best parameter from the list.

```
gs.best_params_
```
Copy Code

Output

```
{'algorithm': 'auto', 'n_neighbors': 6}
```
Copy Code

As per the Cross-Validation process, will figure out the mean and get the results

```
gs.cv_results_['mean_test_score']
```
Copy Code

Output

```
array([0.68134172, 0.71701607, 0.71331237, 0.71509434, 0.72075472,
       0.73944794, 0.72085954, 0.73392732, 0.68134172, 0.71701607,
       0.71331237, 0.71509434, 0.72075472, 0.73944794, 0.72085954,
       0.73392732, 0.68134172, 0.71701607, 0.71331237, 0.71509434,
       0.72075472, 0.73944794, 0.72085954, 0.73392732, 0.68134172,
       0.71701607, 0.71331237, 0.71509434, 0.72075472, 0.73944794,
       0.72085954, 0.73392732])
```

That's fine. which one is the best accuracy from the above list, this is simple, already we found the best parameter from the list is {'algorithm': 'auto', 'n_neighbors': 6}, So compare the 32 combinations of different parameters and accuracy list. this answer is **0.73944794.** is the highest value among the list and this is the BEST accuracy of the training model.

Best accuracy from training

```
print(gs.score(X_test,y_test))
```

Output

```
0.70129870
```

Random Search

The Grid Search one that we have discussed above usually increases the complexity in terms of the computation flow, So sometimes GS is considered inefficient since it attempts all the combinations of given hyperparameters. But the Randomized Search is used to train the models based on random hyperparameters and combinations. obviously, the number of training models are small column than grid search.

Read More about [Random Forest Algorithm for Beginners]

In simple terms, In Random Search, in a given grid, the list of hyperparameters are trained and test our model on a random combination of given hyperparameters.

Getting RandomForestClassifier object for my operation.

```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier
from scipy.stats import randint as sp_randint
```

Copy Code

Assigning my Train and Test spilt to my RandomForestClassifier object

```
# build a RandomForestClassifier
clf = RandomForestClassifier(n_estimators=50)
```

Copy Code

Specifying the list of parameters and distributions

```
param_dist = {"max_depth": [3, None],
              "max_features": sp_randint(1, 11),
              "min_samples_split": sp_randint(2, 11),
              "min_samples_leaf": sp_randint(1, 11),
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}
```

Copy Code

Defining the sample, distributions and cross-validation

```
samples = 8  # number of random samples
randomCV = RandomizedSearchCV(clf, param_distributions=param_dist, n_iter=samples,cv=3)
```

Copy Code

All parameters are set and, let's do the fit model

```
randomCV.fit(X, y)
print(randomCV.best_params_)
```

Copy Code

Output

```
{'bootstrap': False, 'criterion': 'gini', 'max_depth': 3, 'max_features': 3, 'min_samples_leaf': 7, 'min_samples_split': 8}
```

Copy Code

As per the Cross-Validation process, will figure out the mean and get the results

```
randomCV.cv_results_['mean_test_score']
```

Copy Code

Output

```
array([0.73828125, 0.69010417, 0.7578125 , 0.75911458, 0.73828125,
              nan,        nan, 0.7421875 ])
```

Copy Code

Best accuracy from training

```
print(randomCV.score(X_test,y_test))
```

Copy Code

Output

```
0.8744588744588745
```

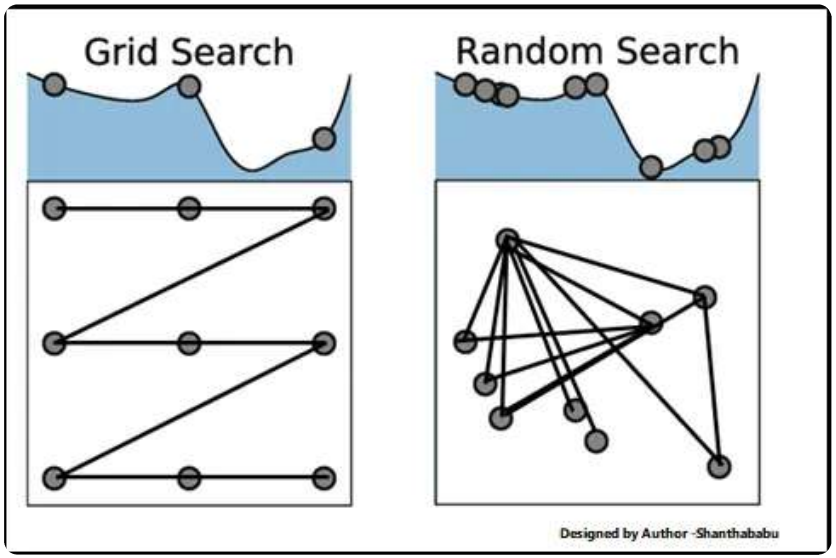
Copy Code

You may have a question, now which technique is best to go. The straight answer is RandomSearshCV, let's see why?

Comparison Study of GridSearchCV and RandomSearshCV

GridSearchCV	RandomSearshCV
Grid is well-defined	Grid is not well defined
Discrete values for HP-params	Continuos values and Statistical distribution
Defined size for Hyperparameter space	No such a restriction
Picks of the best combination from HP-Space	Picks up the samples from HP-Space
Samples are not created	Samples are created and specified by the range and n_iter
Low performance than RSCV	Better performance and result
Guided flow to search for the best combination	The name itself says that, no guidance.

The blow pictorial representation would give you the best understanding of GridSearchCV and RandomSearshCV.



Conclusion

Guys! So far we have discussed in a detailed study of Hyperparameter visions with respect to the Machine Learning point of view, please remember a few things before we go

- Each model has a set of hyperparameters, so we have carefully chosen them and tweaked them during hyperparameter tuning. I mean building the HP space.
- All hyperparameters are NOT equally important and no defined rules for this. try to use continuous values instead of discrete values.
- Make sure to use **K-Fold** while using Hyperparameter tuning to improvise your hyperparameter tuning and coverage of hyperparameter space.
- Go with a better combination for hyperparameters and build strong results.

Hope you like the exploration of hyperparameter tuning! By employing various hyperparameter optimization techniques and methods, you can significantly enhance your model's performance and achieve better results.

Thanks for the time and will connect on different topics. Until then Bye! Cheers!

Key Takeaways:

- Hyperparameter tuning is crucial for selecting the right machine learning model and improving its performance.
- Hyperparameters control the behavior of the model/algorithm, while model parameters are learned from data.
- GridSearchCV and RandomSearchCV are systematic ways to search for optimal hyperparameters.
- Following best practices like proper data splitting is essential to avoid data leakage and overfitting.

Blogathon | Data Analysis | How To Do Hyperparameter Tuning | How To Tune Hyperparameters | Hyper Parameter Tuning | Hyper Parameter Tuning In Machine Learning | Hyperparameter | Hyperparameter Optimization | Hyperparameter Tuning | Hyperparameter Tuning In Machine Learning | Machine Learning Hyperparameter Tuning | Python | Techniques

Shanthababu Pandian

Shanthababu Pandian has over 23 years of IT experience, specializing in data architecting, engineering, analytics, DQ&G, data science, ML, and Gen AI. He holds a BE in electronics and communication engineering and three Master's degrees (M.Tech, MBA, M.S.) from a prestigious Indian university. He has completed postgraduate programs in AIML from the University of Texas and data science from IIT Guwahati. He is a director of data and AI in London, UK, leading data-driven transformation programs focusing on team building and nurturing AIML and Gen AI. He helps global clients achieve business value through scalable data engineering and AI technologies. He is also a national and international speaker, author, technical reviewer, and blogger.

Beginner | Guide | Machine Learning | Python | Python | Technique

Free Courses

Generative AI - A Way of Life

Explore Generative AI for beginners: create text and images, use top AI tools, learn practical skills, and ethics.

Getting Started with Large Language Models

Master Large Language Models (LLMs) with this course, offering clear guidance in NLP and model training made simple.

Building LLM Applications using Prompt Engineering

This free course guides you on building LLM apps, mastering prompt engineering, and developing chatbots with enterprise data.

Improving Real World RAG Systems: Key Challenges & Practical Solutions

Explore practical solutions, advanced retrieval strategies, and agentic RAG systems to improve context, relevance, and accuracy in AI-driven applications.

Microsoft Excel: Formulas & Functions

Master MS Excel for data analysis with key formulas, functions, and LookUp tools in this comprehensive course.

Responses From Readers

What are your thoughts?...

Submit reply

P S Kumar

Excellent explanation about the hyperparameter turning and the technique are explored in altmate manner. Thanks a lot. 👌 👌

Frequently Asked Questions

Q1. How does the objective function impact the process of tuning hyperparameters in machine learning models?

A. The objective function evaluates the performance of a machine learning model at various hyperparameter settings. The main goal during hyperparameter tuning is to find the optimal hyperparameters that maximize or minimize this function, which often requires extensive computational resources to explore the search space effectively.

Q2. What are the advantages of using frameworks like PyTorch or TensorFlow in hyperparameter tuning?What are the advantages of using frameworks like PyTorch or TensorFlow in hyperparameter tuning?

Q3. What strategies can be employed to prevent overfitting during hyperparameter tuning?

Q4. Is GridSearchCV a hyperparameter tuning?

RECOMMENDED ARTICLES

Tune Hyperparameters with GridSearchCV

Hyperparameter Optimization in Machine Learning...

Evaluating Machine Learning Models using Hyperp...

Hyperparameter Tuning Using Randomized Search

A Hands-On Discussion on Hyperparameter Optimiz...

An Effective Approach To Hyper-Parameter Tuning...

Interactive Widget-Based Hyperparameter Tuning ...

Hyperopt: The Alternative Hyperparameter Optimi...

A Beginner’s Guide to Random Forest Hyper...

Tune ML Models in No Time with Optuna

Flagship Courses

GenAI Pinnacle Program | AI/ML BlackBelt Courses

Free Courses

Generative AI | Large Language Models | Building LLM Applications using Prompt Engineering | Building Your first RAG System using LlamaIndex | Stability.AI | MidJourney | Building Production Ready RAG systems using LlamaIndex | Building LLMs for Code | Deep Learning | Python | Microsoft Excel | Machine Learning | Decision Trees | Pandas for Data Analysis | Ensemble Learning | NLP | NLP using Deep Learning | Neural Networks | Loan Prediction Practice Problem | Time Series Forecasting | Tableau | Business Analytics

Popular Categories

Generative AI | Prompt Engineering | Generative AI Application | News | Technical Guides | AI Tools | Interview Preparation | Research Papers | Success Stories | Quiz | Use Cases | Listicles

Generative AI Tools and Techniques

GANs | VAEs | Transformers | StyleGAN | Pix2Pix | Autoencoders | GPT | BERT | Word2Vec | LSTM | Attention Mechanisms | Diffusion Models | LLMs | SLMs | StyleGAN | Encoder Decoder Models | Prompt Engineering | LangChain | LlamaIndex | RAG | Fine-tuning | LangChain AI Agent | Multimodal Models | RNNs | DCGAN | ProGAN | Text-to-Image Models | DDPM | Document Question Answering | Imagen | T5 (Text-to-Text Transfer Transformer) | Seq2seq Models | WaveNet | Attention Is All You Need (Transformer Architecture)

Popular GenAI Models

Llama 3.1 | Llama 3 | Llama 2 | GPT 4o Mini | GPT 4o | GPT 3 | Claude 3 Haiku | Claude 3.5 Sonnet | Phi 3.5 | Phi 3 | Mistral Large 2 | Mistral NeMo | Mistral-7b | Gemini 1.5 Pro | Gemini Flash 1.5 | Bedrock | Vertex AI | DALL.E | Midjourney | Stable Diffusion

Data Science Tools and Techniques

Python | R | SQL | Jupyter Notebooks | TensorFlow | Scikit-learn | PyTorch | Tableau | Apache Spark | Matplotlib | Seaborn | Pandas | Hadoop | Docker | Git | Keras | Apache Kafka | AWS | NLP | Random Forest | Computer Vision | Data Visualization | Data Exploration | Big Data | Common Machine Learning Algorithms | Machine Learning