HYPER PARAMETER TUNING

In **machine learning**, a model has **parameters** and **hyperparameters**.

🔹 **Model Parameters** → Learned from the training data

- The algorithm updates them during training.

🔹 **Hyperparameters** Hyperparameters are parameters whose values are set before the learning process begins and control the learning algorithm itself, unlike model parameters which are learned during training.

1. They control how the learning process works.

Feature	Model Parameters	Hyperparameters
Definition	Values learned by the algorithm from training data	Values set manually before training to control learning
Who sets them?	Learned automatically by optimization algorithm	Set by user / tuning method
Changes during training?	✅ Yes (updated in each iteration)	❌ No (fixed during training run)
Examples (Logistic Regression)	Weights for each feature (e.g., w1, w2, ..., wn)	C, penalty, solver
Examples (Decision Tree)	Split thresholds, tree structure	max_depth, min_samples_leaf
Examples (SVM)	Support vectors, weights	C, kernel, gamma
Optimization	Found via training (gradient descent, etc.)	Found via tuning (grid search, random search, etc.)

Why hyperparamaeters are important?

- **Improves model performance**: Optimal hyperparameter settings can significantly increase a model's accuracy and lead to better results on unseen data.

- **Reduces overfitting and underfitting**: Tuning helps balance the model's complexity, ensuring it generalizes well to new data rather than memorizing the training data (overfitting) or being too simplistic to capture patterns (underfitting).
- **Enhances efficiency**: Finding the right hyperparameters can optimize resource utilization, reducing unnecessary computational costs and training time.

HYPERPARAMETER TUNING TECHNIQUES

1. Manual Search

- You try values based on intuition.

2. **Grid Search:**

   Explores a predefined range of hyperparameter values by trying every possible combination.

- Pros: Guaranteed to find the best combination within the defined search space.
- Cons: Can be computationally expensive and time-consuming, especially with a large number of hyperparameters or values.

3. **Random Search:**

   Randomly samples hyperparameter combinations from a defined search space. It can be more efficient than grid search, especially in high-dimensional spaces.

- Pros: Faster than grid search and can explore a broader range of values.
- Cons: No guarantee of finding the globally optimal combination.

4. **Bayesian Optimization:**

   This is a more intelligent approach that builds a probabilistic model to predict which hyperparameter combinations are most likely to yield the best results. It learns from past evaluations to inform its next choice, making it more efficient than brute-force methods.

- Pros: Finds a good hyperparameter combination in fewer iterations, making it suitable for models with expensive training times.
- Cons: Can be complex to set up and, because it is sequential, cannot be easily parallelized like grid or random search.

Process of Hyperparameter Tuning:

- **Define Search Space:** Specify the range or set of possible values for each hyperparameter.
- **Choose Evaluation Metric:** Select a metric (e.g., accuracy, F1-score, loss) to evaluate the model's performance for different hyperparameter combinations.
- **Select Tuning Strategy:** Choose a method like grid search, random search, or Bayesian optimization.
- **Execute Trials:** Run multiple training trials, each with a different set of hyperparameters, and evaluate the model's performance using the chosen metric.
- **Identify Optimal Hyperparameters:** Select the hyperparameter combination that yields the best performance on the evaluation metric.

1. Logistic Regression

Logistic Regression is simpler, but still has key hyperparameters:

Important Hyperparameters:

1. **C (inverse of regularization strength)**
   - Default = 1.0
   - Smaller C → stronger regularization (simpler model, avoids overfitting).
   - Larger C → weaker regularization (fits training data more closely).
2. **Penalty (type of regularization)**
   - Options: 'l1', 'l2', 'elasticnet', 'none'
   - 'l2' (Ridge) is default and most common.
   - 'l1' (Lasso) leads to sparsity (some coefficients = 0).
   - 'elasticnet' = mix of L1 + L2.
3. **Solver (optimization algorithm)**
   - 'liblinear': good for small datasets & L1/L2.
   - 'saga': supports large datasets, L1 + elasticnet.
   - 'lbfgs': fast, supports only L2.
4. **max_iter (max iterations)**
   - Sometimes logistic regression needs more iterations to converge, so tune this (e.g., 100 → 1000).

🔹 2. Support Vector Machine (SVM)

SVMs have crucial hyperparameters, and tuning them well makes a big difference.

Important Hyperparameters:

1. **C (regularization parameter)**
   - Similar meaning as in logistic regression.
   - Small C → larger margin, allows misclassifications (prevents overfitting).
   - Large C → smaller margin, fewer misclassifications (risk of overfitting).
2. **Kernel**
   - Defines the decision boundary shape:
     - 'linear' → straight-line boundaries.
     - 'rbf' (Gaussian) → nonlinear, most common.
     - 'poly' → polynomial decision boundary.
     - 'sigmoid' → like neural networks.
3. **Gamma (for RBF/poly kernels)**
   - Controls influence of a single training point.
   - Low gamma → smooth, far-reaching influence (underfit).
   - High gamma → very sharp, narrow influence (overfit).
4. **Degree (if using polynomial kernel)**
   - Degree of the polynomial.
   - Typically tuned if you select kernel='poly'.

🔹 3. Decision Tree

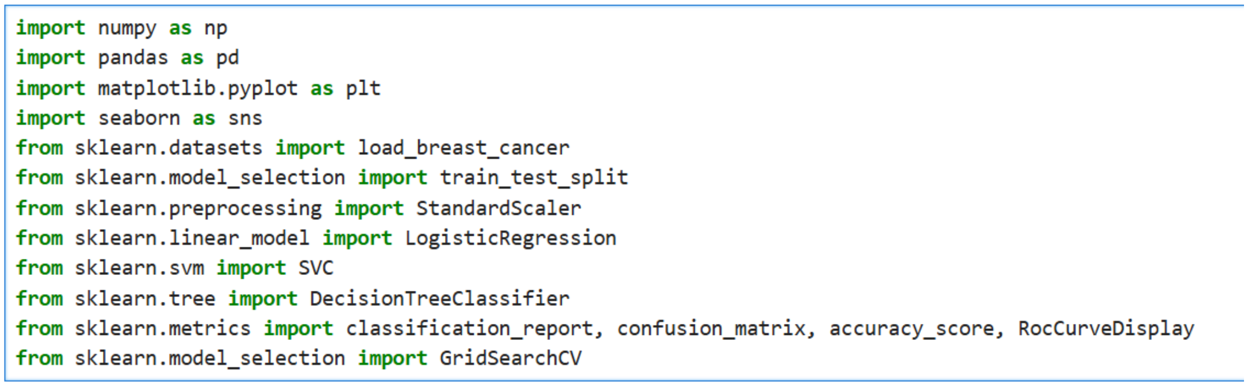
Decision trees are **highly sensitive to hyperparameters** — tuning prevents overfitting.

Important Hyperparameters:

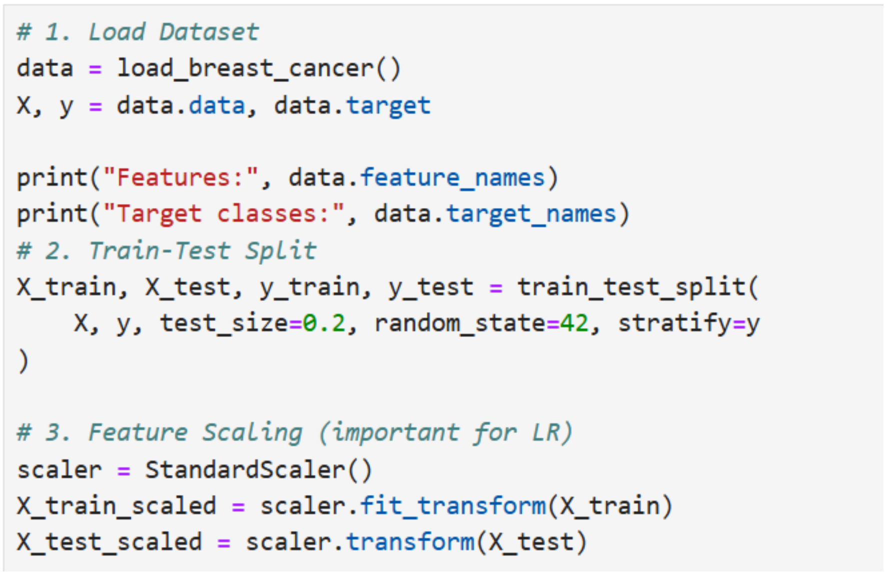
1. **max_depth**
   - Maximum depth of the tree.
   - Small = underfitting, large = overfitting.
2. **min_samples_split**
   - Minimum samples required to split a node.

   - Larger values → fewer splits → simpler tree.

3. **min_samples_leaf**
   - Minimum samples required in a leaf node.
   - Prevents creating very small leaves.
4. **max_features**
   - Number of features to consider when looking for the best split.
   - Options: None (all), 'sqrt', 'log2', or a fixed number.
5. **criterion**
   - Function to measure split quality:
     - 'gini' → Gini Impurity (default).
     - 'entropy' → Information Gain.
6. **max_leaf_nodes**
   - Maximum number of leaf nodes.
   - Controls tree complexity.
7. **class_weight**
   - Useful for imbalanced data (e.g., fraud detection).



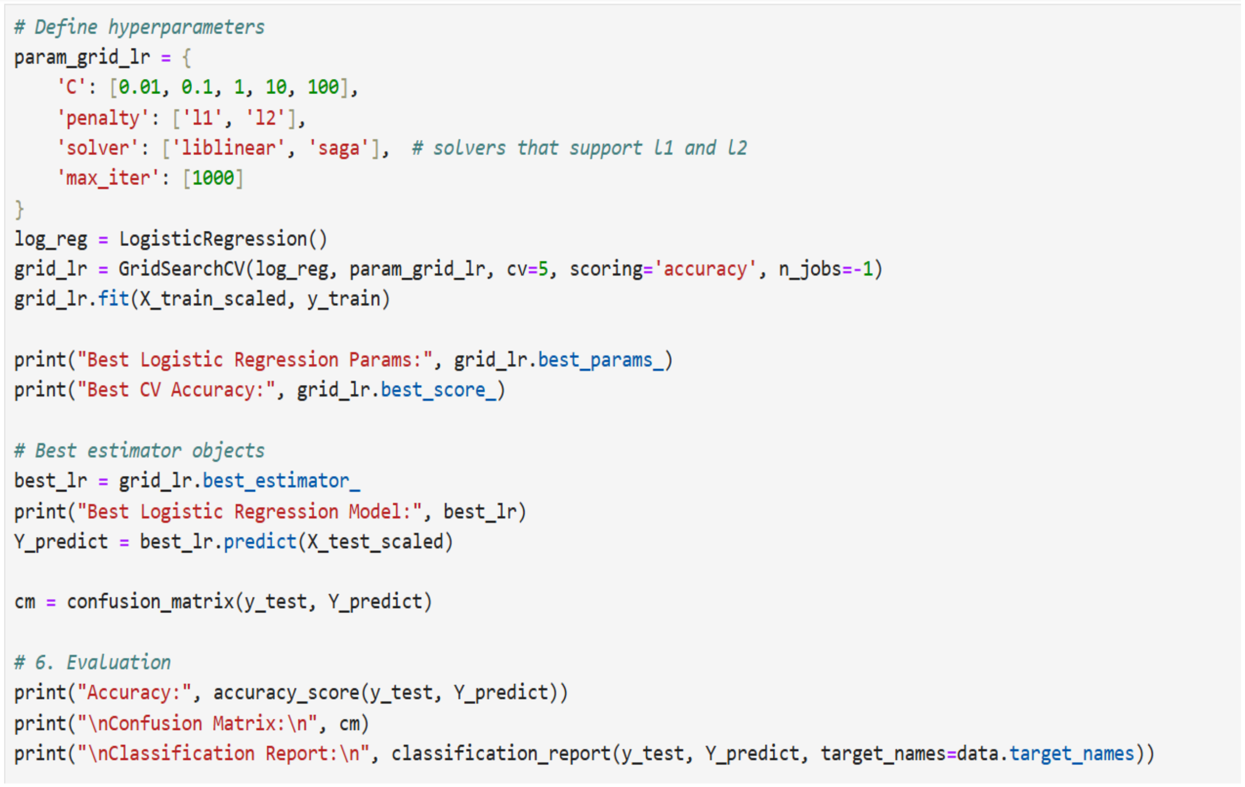
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, RocCurveDisplay
from sklearn.model_selection import GridSearchCV
```



```python
# 1. Load Dataset
data = load_breast_cancer()
X, y = data.data, data.target

print("Features:", data.feature_names)
print("Target classes:", data.target_names)
# 2. Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# 3. Feature Scaling (important for LR)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```



```python
# Define hyperparameters
param_grid_lr = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear', 'saga'],  # solvers that support L1 and L2
    'max_iter': [1000]
}
log_reg = LogisticRegression()
grid_lr = GridSearchCV(log_reg, param_grid_lr, cv=5, scoring='accuracy', n_jobs=-1)
grid_lr.fit(X_train_scaled, y_train)

print("Best Logistic Regression Params:", grid_lr.best_params_)
print("Best CV Accuracy:", grid_lr.best_score_)

# Best estimator objects
best_lr = grid_lr.best_estimator_
print("Best Logistic Regression Model:", best_lr)
Y_predict = best_lr.predict(X_test_scaled)

cm = confusion_matrix(y_test, Y_predict)

# 6. Evaluation
print("Accuracy:", accuracy_score(y_test, Y_predict))
print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, Y_predict, target_names=data.target_names))
```

```python
#SVM
param_grid_svm = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'rbf'],
    'gamma': ['scale', 'auto', 0.01, 0.1, 1]
}

svm = SVC()
grid_svm = GridSearchCV(svm, param_grid_svm, cv=5, scoring='accuracy', n_jobs=-1)
grid_svm.fit(X_train_scaled, y_train)

print("Best SVM Params:", grid_svm.best_params_)
print("Best CV Accuracy:", grid_svm.best_score_)

# Best estimator objects
best_svm = grid_svm.best_estimator_
print("The best SVM model :",best_svm)
Y_predict = best_svm.predict(X_test_scaled)

cm = confusion_matrix(y_test, Y_predict)

# 6. Evaluation
print("Accuracy:", accuracy_score(y_test, Y_predict))
print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, Y_predict, target_names=data.target_names))
```

```python
param_grid_dt = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 3, 5, 7, 10],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2']
}

dt = DecisionTreeClassifier(random_state=42)
grid_dt = GridSearchCV(dt, param_grid_dt, cv=5, scoring='accuracy', n_jobs=-1)
grid_dt.fit(X_train_scaled, y_train)

print("Best Decision Tree Params:", grid_dt.best_params_)
print("Best CV Accuracy:", grid_dt.best_score_)

# Best estimator objects
best_dt = grid_dt.best_estimator_
print("The best Tree model:",best_dt)
Y_predict = best_dt.predict(X_test_scaled)

cm = confusion_matrix(y_test, Y_predict)

# 6. Evaluation
print("Accuracy:", accuracy_score(y_test, Y_predict))
print("\nConfusion Matrix:\n", cm)
print("\nClassification Report:\n", classification_report(y_test, Y_predict, target_names=data.target_names))
```