Let’s proceed with **hyperparameter tuning** for Logistic Regression using **GridSearchCV**. I'll explain each hyperparameter we’re tuning and why it’s important.

**1️⃣ Why Hyperparameter Tuning?**

Hyperparameter tuning allows us to find the best combination of model parameters to optimize performance (e.g., accuracy, precision, recall). For **Logistic Regression**, the key hyperparameters to tune are related to **regularization** and the **solver**.

**2️⃣ Key Hyperparameters to Tune**

**a. C: Inverse of Regularization Strength**

* **What it does**: Controls the amount of regularization. A smaller C value applies stronger regularization, preventing overfitting.
* **Why tune it**: Finding the right balance between underfitting and overfitting is critical for better generalization.
* **Range to try**: [0.01, 0.1, 1, 10, 100].

**b. penalty: Type of Regularization**

* **What it does**: Defines how regularization is applied to the model.
  + **l2**: Ridge regression (default).
  + **none**: No regularization.
* **Why tune it**: Regularization can improve performance by preventing the model from relying too much on a few features. l2 is generally preferred, but testing none ensures we’re not unnecessarily regularizing.
* **Options**: ['l2', 'none'].

**c. solver: Algorithm for Optimization**

* **What it does**: Determines the optimization algorithm used to find the best parameters.
  + **lbfgs**: Recommended for most cases.
  + **liblinear**: Better for small datasets or when using l1 regularization.
  + **saga**: Handles larger datasets and supports both l1 and l2.
* **Why tune it**: Different solvers can handle specific data structures better.
* **Options**: ['lbfgs', 'liblinear', 'saga'].

**d. max\_iter: Maximum Iterations**

* **What it does**: Sets the number of iterations allowed for optimization.
* **Why tune it**: Increase from the default (100) if convergence warnings occur.
* **Fixed Value**: Use 1000.

**3️⃣ GridSearchCV Implementation**

**Code for Hyperparameter Tuning:**

python

CopyEdit

from sklearn.model\_selection import GridSearchCV

from sklearn.linear\_model import LogisticRegression

# Define the parameter grid

param\_grid = {

'C': [0.01, 0.1, 1, 10, 100], # Regularization strength

'penalty': ['l2', 'none'], # Type of regularization

'solver': ['lbfgs', 'liblinear', 'saga'], # Optimization solver

}

# Initialize Logistic Regression with class weights

log\_model = LogisticRegression(random\_state=42, class\_weight='balanced', max\_iter=1000)

# Set up GridSearchCV

grid\_search = GridSearchCV(

estimator=log\_model,

param\_grid=param\_grid,

scoring='f1\_weighted', # Use F1 score to balance precision and recall

cv=5, # 5-fold cross-validation

verbose=2, # Print progress

n\_jobs=-1 # Use all available CPUs

)

# Fit GridSearchCV

grid\_search.fit(X\_train\_scaled, y\_train)

# Print the best parameters and corresponding score

print("Best Parameters:", grid\_search.best\_params\_)

print("Best F1 Score:", grid\_search.best\_score\_)

**4️⃣ Key Hyperparameter Explanations**

| **Hyperparameter** | **Why It’s Tuned** | **Default Value** | **What We’re Testing** |
| --- | --- | --- | --- |
| **C** | Balances regularization | 1.0 | [0.01, 0.1, 1, 10, 100] |
| **penalty** | Determines regularization type | l2 | ['l2', 'none'] |
| **solver** | Optimization algorithm | lbfgs | ['lbfgs', 'liblinear', 'saga'] |
| **max\_iter** | Avoids convergence issues | 100 | Fixed at 1000 |

**5️⃣ Interpreting Results**

1. **Best Parameters**:
   * After running grid\_search.best\_params\_, you’ll see the best combination of hyperparameters.
   * For example: {'C': 0.1, 'penalty': 'l2', 'solver': 'lbfgs'}.
2. **Best Score**:
   * The grid\_search.best\_score\_ shows the cross-validated F1 score for the best parameter combination.

**6️⃣ Next Steps**

1. **Refit the Logistic Regression Model** with the best parameters:

python

CopyEdit

best\_log\_model = grid\_search.best\_estimator\_

best\_log\_model.fit(X\_train\_scaled, y\_train)

1. **Evaluate the Model**: Use the evaluation steps from earlier (classification report, confusion matrix, feature coefficients).
2. **Compare Performance**: Compare the tuned Logistic Regression model with the baseline and alternative models (e.g., Random Forest, Gradient Boosting).