**📘 Logistic Regression on the Iris Dataset**

**With Hyperparameter Tuning: C and penalty**

**🧠 1. Introduction**

**This document demonstrates how to use Logistic Regression, a fundamental classification algorithm, on the well-known Iris dataset. The focus is on hyperparameter tuning using only:**

* **C: Inverse of regularization strength**
* **penalty: Type of regularization (l1 or l2)**

**We use GridSearchCV to find the best parameter combination and evaluate the model using:**

* **Accuracy**
* **Classification Report (Precision, Recall, F1-Score)**
* **Confusion Matrix (visualized using Seaborn)**

**🌼 2. Dataset Description**

**The Iris dataset is a classic multivariate dataset introduced by Ronald Fisher in 1936. It contains:**

* **150 samples (rows)**
* **3 classes (species): *Setosa*, *Versicolor*, *Virginica***
* **4 features (columns):**
  + **Sepal Length (cm)**
  + **Sepal Width (cm)**
  + **Petal Length (cm)**
  + **Petal Width (cm)**

**The target variable is categorical with labels:  
0 = Setosa, 1 = Versicolor, 2 = Virginica**

**We load the dataset and convert it into a Pandas DataFrame for easier manipulation:**

**iris = load\_iris()**

**df = pd.DataFrame(iris.data, columns=iris.feature\_names)**

**df['target'] = iris.target**

**🧹 3. Data Preparation**

**After loading the data, we split it into features (X) and target (y):**

**X = df.drop('target', axis=1)**

**y = df['target']**

**We then split the dataset into training and testing subsets (80% train, 20% test):**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)**

**🔧 4. Model Training and Hyperparameter Tuning**

**We use GridSearchCV to tune:**

* **C: [0.01, 0.1, 1, 10, 100] → controls the regularization strength (smaller = stronger regularization)**
* **penalty: ['l1', 'l2'] → type of regularization**
* **solver: 'liblinear' is chosen since it's compatible with both l1 and l2**

**param\_grid = {**

**'C': [0.01, 0.1, 1, 10, 100],**

**'penalty': ['l1', 'l2'],**

**'solver': ['liblinear']**

**}**

**grid = GridSearchCV(LogisticRegression(max\_iter=200), param\_grid, cv=5)**

**grid.fit(X\_train, y\_train)**

**This performs a cross-validated grid search and selects the best parameters based on accuracy.**

**📈 5. Model Evaluation**

**✅ Best Parameters Found**

**grid.best\_params\_**

**# Example Output: {'C': 1, 'penalty': 'l2', 'solver': 'liblinear'}**

**🎯 Accuracy Score**

**This is the ratio of correctly predicted observations to the total observations:**

**accuracy\_score(y\_test, y\_pred)**

**# Example: 0.9667**

**📋 Classification Report**

**This includes:**

* **Precision: TP / (TP + FP)**
* **Recall: TP / (TP + FN)**
* **F1-score: Harmonic mean of precision and recall**
* **Support: Number of true instances for each class**

**Example:**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**🔢 Confusion Matrix**

**A 3×3 matrix summarizing the prediction results:**

**confusion\_matrix(y\_test, y\_pred)**

**Example output:**

**A number of numbers on a black background

AI-generated content may be incorrect.**

**This means:**

* **All Setosa samples were correctly predicted**
* **1 Virginica was misclassified as Versicolor**

**📊 Visualization**

**We use Seaborn to create a heatmap of the confusion matrix:**

**sns.heatmap(conf\_matrix, annot=True, fmt='d',**

**xticklabels=target\_names,**

**yticklabels=target\_names,**

**cmap='YlGnBu')**

**This helps visually understand which classes are being confused.**

**✅ 6. Summary**

* **Logistic Regression is a robust model for classification problems.**
* **Tuning C and penalty significantly improves performance.**
* **On the Iris dataset, the model achieved ~97% accuracy.**
* **GridSearchCV makes it easy to search for the best parameter combination.**
* **Visualization of the confusion matrix helps in model debugging and evaluation.**