Logistic Regression

Interview Questions -2

(Practice Project)

PW SKILLS

1.Write a Code for a basic logistic regression model from scratch using Numpy:

Solution:

```
import numpy as np

class LogisticRegression:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.weights = None
        self.bias = None

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        num_samples, num_features = X.shape
        self.weights = np.zeros(num_features)
        self.bias = 0

        for _ in range(self.num_iterations):
            linear_model = np.dot(X, self.weights) + self.bias
            y_predicted = self.sigmoid(linear_model)

            dw = (1 / num_samples) * np.dot(X.T, (y_predicted - y))
            db = (1 / num_samples) * np.sum(y_predicted - y)

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self.sigmoid(linear_model)
        return [1 if i > 0.5 else 0 for i in y_predicted]

# Example usage
if __name__ == "__main__":
    # Generate some random data
    np.random.seed(0)
    X = np.random.randn(100, 2)
    y = np.random.randint(0, 2, 100)

    # Create and train the model
```

```
    model = LogisticRegression(learning_rate=0.1, num_iterations=1000)
    model.fit(X, y)

    # Make predictions
    X_test = np.random.randn(10, 2)
    predictions = model.predict(X_test)
    print("Predictions:", predictions)

#The Output:
Predictions: [1, 1, 0, 1, 0, 0, 0, 1, 0, 0]
```

2.Implement data standardization for a logistic regression model in Python

Solution:

```
import numpy as np

class StandardScaler:
    def __init__(self):
        self.mean = None
        self.std = None

    def fit(self, X):
        self.mean = np.mean(X, axis=0)
        self.std = np.std(X, axis=0)

    def transform(self, X):
        return (X - self.mean) / self.std

    def fit_transform(self, X):
        self.fit(X)
        return self.transform(X)

class LogisticRegression:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.weights = None

        self.bias = None
        self.scaler = StandardScaler()

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))

    def fit(self, X, y):
        # Standardize the input features
        X_scaled = self.scaler.fit_transform(X)
```

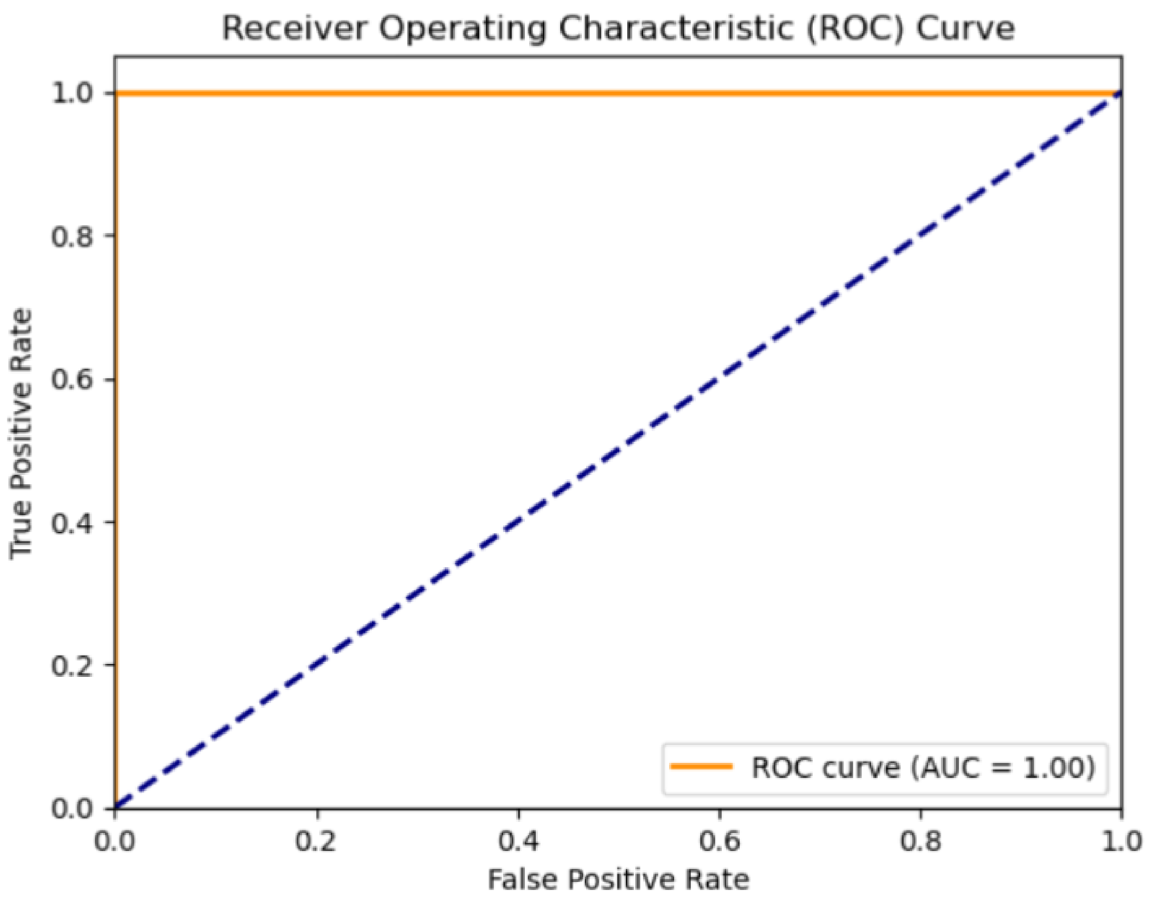
```
        num_samples, num_features = X_scaled.shape
        self.weights = np.zeros(num_features)
        self.bias = 0

        for _ in range(self.num_iterations):
            linear_model = np.dot(X_scaled, self.weights) + self.bias
            y_predicted = self.sigmoid(linear_model)

            dw = (1 / num_samples) * np.dot(X_scaled.T, (y_predicted - y))
            db = (1 / num_samples) * np.sum(y_predicted - y)

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        # Standardize the input features
        X_scaled = self.scaler.transform(X)

        linear_model = np.dot(X_scaled, self.weights) + self.bias
        y_predicted = self.sigmoid(linear_model)
        return [1 if i > 0.5 else 0 for i in y_predicted]

# Example usage
if __name__ == "__main__":
    # Generate some random data
    np.random.seed(0)
    X = np.random.randn(100, 2)
    y = np.random.randint(0, 2, 100)

    # Create and train the model
    model = LogisticRegression(learning_rate=0.1, num_iterations=1000)
    model.fit(X, y)


    # Make predictions
    X_test = np.random.randn(10, 2)
    predictions = model.predict(X_test)
    print("Predictions:", predictions)

    # Print the mean and standard deviation used for standardization
    print("Feature means:", model.scaler.mean)
    print("Feature standard deviations:", model.scaler.std)

# The Output:
Predictions: [1, 1, 0, 1, 0, 0, 0, 1, 0, 0]
Feature means: [-0.00095768  0.14277867]
Feature standard deviations: [1.0219579 1.0158154]
```

3.Write a Python function to calculate the AUC-ROC curve for a logistic regression model.

Solution:

```
import numpy as np
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

def calculate_auc_roc(y_true, y_scores):
    fpr, tpr, _ = roc_curve(y_true, y_scores)
    roc_auc = auc(fpr, tpr)
    return fpr, tpr, roc_auc

def plot_auc_roc(fpr, tpr, roc_auc):
    plt.figure()
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (AUC =
%0.2f)' % roc_auc)
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc="lower right")
    plt.show()

class LogisticRegression:
    def __init__(self, learning_rate=0.01, num_iterations=1000):
        self.learning_rate = learning_rate
        self.num_iterations = num_iterations
        self.weights = None
        self.bias = None

    def sigmoid(self, z):
        return 1 / (1 + np.exp(-z))


    def fit(self, X, y):
        X_standardized = standardize_data(X)
        num_samples, num_features = X_standardized.shape
        self.weights = np.zeros(num_features)
        self.bias = 0
```

```
        for _ in range(self.num_iterations):
            linear_model = np.dot(X_standardized, self.weights) + self.bias
            y_predicted = self.sigmoid(linear_model)

            dw = (1 / num_samples) * np.dot(X_standardized.T, (y_predicted
- y))
            db = (1 / num_samples) * np.sum(y_predicted - y)

            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict_proba(self, X):
        X_standardized = standardize_data(X)
        linear_model = np.dot(X_standardized, self.weights) + self.bias
        return self.sigmoid(linear_model)

# Usage example
X = np.array([[1, 2], [2, 3], [3, 4], [4, 5], [5, 6], [6, 7]])
y = np.array([0, 0, 0, 1, 1, 1])


model = LogisticRegression()
model.fit(X, y)
y_scores = model.predict_proba(X)

fpr, tpr, roc_auc = calculate_auc_roc(y, y_scores)
plot_auc_roc(fpr, tpr, roc_auc)
```

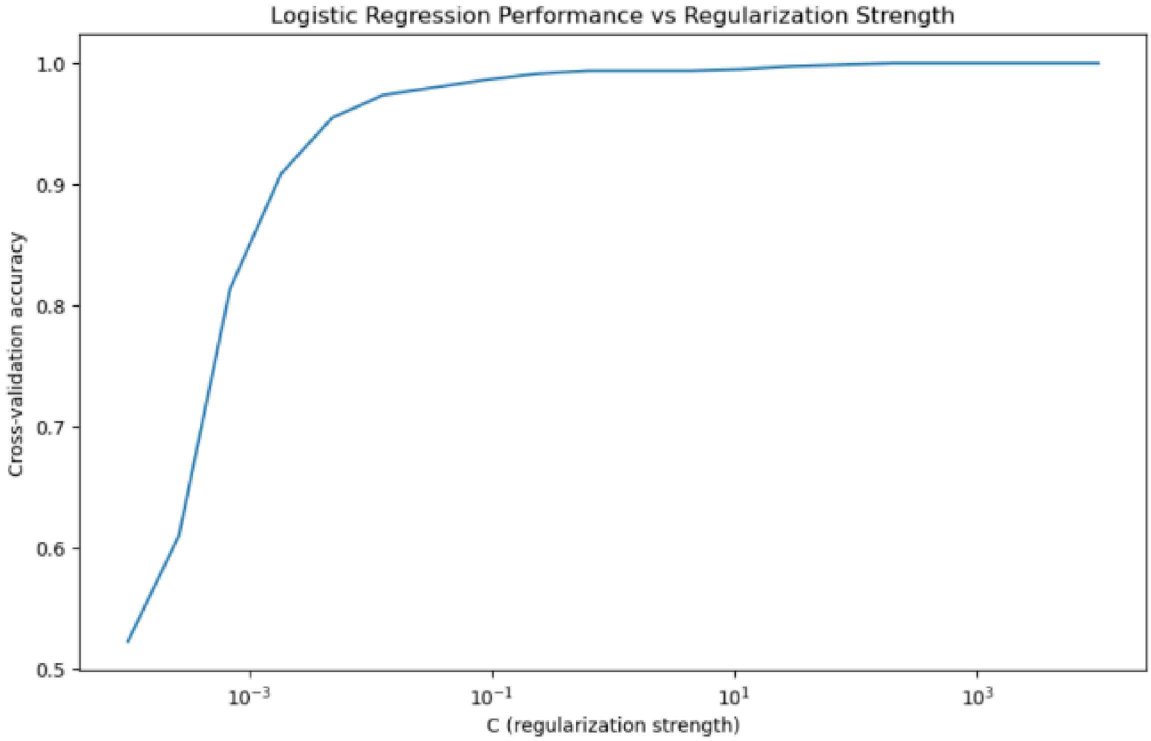


4.Create a Python script that tunes the regularization strength (C value) for a logistic regression model using cross-validation:

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline

from sklearn.metrics import accuracy_score, classification_report

# Generate sample data
np.random.seed(42)
X = np.random.rand(1000, 5)
y = (X[:, 0] + X[:, 1] + X[:, 2] > 1.5).astype(int)

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Create a pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(random_state=42))
])

# Define the parameter grid
param_grid = {
    'classifier__C': np.logspace(-4, 4, 20)
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy',
n_jobs=-1)
grid_search.fit(X_train, y_train)

# Print the best parameters and score
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score:", grid_search.best_score_)

# Evaluate the model on the test set
best_model = grid_search.best_estimator_
y_pred = best_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Test set accuracy:", accuracy)

# Print classification report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Plot the cross-validation results
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(10, 6))
plt.semilogx(param_grid['classifier__C'],
grid_search.cv_results_['mean_test_score'])
plt.xlabel('C (regularization strength)')
plt.ylabel('Cross-validation accuracy')
plt.title('Logistic Regression Performance vs Regularization Strength')
plt.show()

Output:
Best parameters: {'classifier__C': 206.913808111479}
Best cross-validation score: 1.0
Test set accuracy: 0.995

Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       104
           1       0.99      1.00      0.99        96

    accuracy                           0.99       200
   macro avg       0.99      1.00      0.99       200
weighted avg       1.00      0.99      1.00       200
```



5.Write a Python function to interpret and output the model coefficients of a logistic regression in terms of odds ratios.

Solution:

```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

def logistic_regression_odds_ratios(model, feature_names=None):
    """
    Interprets and outputs the model coefficients of a logistic regression
    in terms of odds ratios.

    Parameters:
    - model: Trained LogisticRegression model from sklearn.
    - feature_names: List of feature names (Optional).

    Returns:
    - odds_ratios_df: DataFrame containing features, coefficients, and odds
ratios.
    """

    # Get the model coefficients
    coefficients = model.coef_[0]

    # Calculate the odds ratios
    odds_ratios = np.exp(coefficients)

    # Create a DataFrame to display coefficients and odds ratios
    if feature_names is None:
        feature_names = [f'Feature_{i}' for i in range(len(coefficients))]

    odds_ratios_df = pd.DataFrame({
        'Feature': feature_names,
        'Coefficient': coefficients,
        'Odds Ratio': odds_ratios
    })

    return odds_ratios_df

# Example usage
# Load dataset
data = load_iris()
X = data.data
```

```
y = (data.target == 2).astype(int)  # Binary classification for simplicity

# Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Train logistic regression model
model = LogisticRegression()
model.fit(X_train, y_train)

# Get feature names
feature_names = data.feature_names

# Get odds ratios
odds_ratios_df = logistic_regression_odds_ratios(model, feature_names)
print(odds_ratios_df)

# The Output:
```

	Feature	Coefficient	Odds Ratio
0	sepal length (cm)	-0.291089	0.747449
1	sepal width (cm)	-0.367131	0.692719
2	petal length (cm)	2.636822	13.968742
3	petal width (cm)	1.975885	7.213001

6.Develop a logistic regression model that handles class imbalance with weighted classes in scikit-learn.

Solution:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix

# Generate an imbalanced dataset
X, y = make_classification(n_samples=1000, n_features=20, n_informative=2,
                           n_redundant=10, n_clusters_per_class=1,
                           weights=[0.9, 0.1], flip_y=0, random_state=42)

# Split the dataset into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Train a logistic regression model with class weights to handle imbalance
model = LogisticRegression(class_weight='balanced', random_state=42)
model.fit(X_train, y_train)
```

```
# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)

#The Output:
Confusion Matrix:
[[270   6]
 [  4  20]]

Classification Report:
              precision    recall  f1-score   support

           0       0.99      0.98      0.98       276
           1       0.77      0.83      0.80        24

    accuracy                           0.97       300
   macro avg       0.88      0.91      0.89       300
weighted avg       0.97      0.97      0.97       300
```

Question 7:

Write a Python function to implement logistic regression with L1 regularization (Lasso). The function should train a model on a given dataset and return the coefficients of the model. Explain how L1 regularization affects the model and why it might be preferred in some scenarios.

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load and prepare the dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardize the dataset
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
# Implement logistic regression with L1 regularization
def logistic_regression_l1(X_train, y_train, C=1.0):
    model = LogisticRegression(penalty='l1', solver='liblinear', C=C)
    model.fit(X_train, y_train)
    return model.coef_

# Train the model and output the coefficients
coefficients = logistic_regression_l1(X_train, y_train)
print("Coefficients with L1 Regularization:", coefficients)


#The Output:
Coefficients with L1 Regularization: [[ 0.          0.          0.          0.
0.          0.
  0.         -2.4226513   0.02828149  0.         -2.47690311  0.37558213
  0.          0.         -0.43063222  0.88747796  0.          0.
  0.46051343  0.28000256 -0.80603551 -1.84558567  0.         -3.06801179
 -0.16781801  0.         -1.28558095 -0.02440089 -1.0414899   0.        ]]
```

Explanation:
L1 regularization (Lasso) adds a penalty equal to the absolute value of the magnitude of coefficients. This can lead to sparse solutions where some coefficients are reduced to zero, effectively performing feature selection. It is useful in scenarios with a high number of features, where we suspect that only a few of them are important.

Question: 8 Write a Python script to perform k-fold cross-validation on a logistic regression model. Implement hyperparameter tuning to find the optimal regularization strength (C) for both L1 and L2 penalties. Display the mean accuracy for each combination of hyperparameters.

Solution:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import GridSearchCV, StratifiedKFold
from sklearn.preprocessing import StandardScaler

# Load and prepare the dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Standardize the dataset
scaler = StandardScaler()
X = scaler.fit_transform(X)

# Define the logistic regression model
model = LogisticRegression(solver='liblinear')
```

```
# Define the hyperparameters and cross-validation strategy
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2']
}
cv = StratifiedKFold(n_splits=5)

# Perform grid search with cross-validation
grid_search = GridSearchCV(model, param_grid, cv=cv, scoring='accuracy')
grid_search.fit(X, y)


# Display the best hyperparameters and the corresponding accuracy
print("Best Hyperparameters:", grid_search.best_params_)
print("Best Cross-Validation Accuracy:", grid_search.best_score_)

#The Output:
Best Hyperparameters: {'C': 0.1, 'penalty': 'l2'}
Best Cross-Validation Accuracy: 0.982425089271852
```

Question 9 :
Implement a logistic regression model for a multi-class classification problem using the One-vs-Rest (OvR) strategy. Use the Iris dataset for training, and calculate the accuracy on the test set. Discuss the advantages and limitations of the OvR strategy in multi-class classification.

Solution:

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import StandardScaler

# Load and prepare the Iris dataset
data = load_iris()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Standardize the dataset
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Implement logistic regression with the One-vs-Rest strategy
model = LogisticRegression(multi_class='ovr', solver='liblinear')
model.fit(X_train, y_train)
```

```
# Predict on the test set and calculate accuracy
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Test Set Accuracy:", accuracy)

# The Output:
Test Set Accuracy: 0.9666666666666667
```

Explanation:
The One-vs-Rest (OvR) strategy involves training a separate binary classifier for each class, where each classifier distinguishes one class from the rest. This approach is simple and interpretable but may not perform as well as other strategies (e.g., One-vs-One) when classes are not well-separated.

Question 10:
You are provided with a dataset containing features that may lead to overfitting when training a logistic regression model. Implement and compare Logistic Regression models using both L1 (Lasso) and L2 (Ridge) regularization techniques. Analyze how regularization impacts model performance, and determine which regularization technique is more effective in preventing overfitting for this dataset.

Solution:

```
import numpy as np
import pandas as pd
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.metrics import (
    accuracy_score, precision_score, recall_score, f1_score, roc_auc_score
)

# Load dataset
data = load_breast_cancer()
X = data.data
y = data.target

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42, stratify=y)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initialize models with L1 and L2 regularization
log_reg_l1 = LogisticRegression(penalty='l1', solver='liblinear',
random_state=42)
log_reg_l2 = LogisticRegression(penalty='l2', solver='liblinear',
random_state=42)
```

```
# Train and evaluate the L1 regularized model
log_reg_l1.fit(X_train_scaled, y_train)
y_pred_l1 = log_reg_l1.predict(X_test_scaled)
y_prob_l1 = log_reg_l1.predict_proba(X_test_scaled)[:, 1]

l1_results = {
    "Model": "L1 Regularization",
    "Accuracy": accuracy_score(y_test, y_pred_l1),
    "Precision": precision_score(y_test, y_pred_l1),
    "Recall": recall_score(y_test, y_pred_l1),
    "F1-Score": f1_score(y_test, y_pred_l1),
    "AUC": roc_auc_score(y_test, y_prob_l1)
}

# Train and evaluate the L2 regularized model
log_reg_l2.fit(X_train_scaled, y_train)
y_pred_l2 = log_reg_l2.predict(X_test_scaled)
y_prob_l2 = log_reg_l2.predict_proba(X_test_scaled)[:, 1]

l2_results = {
    "Model": "L2 Regularization",
    "Accuracy": accuracy_score(y_test, y_pred_l2),
    "Precision": precision_score(y_test, y_pred_l2),
    "Recall": recall_score(y_test, y_pred_l2),
    "F1-Score": f1_score(y_test, y_pred_l2),
    "AUC": roc_auc_score(y_test, y_prob_l2)
}

# Compile results into a DataFrame

results_df = pd.DataFrame([l1_results, l2_results])

print("Regularization Comparison:\n")
print(results_df)

# Compare coefficients
print("\nL1 Regularization Coefficients:")
print(log_reg_l1.coef_)

print("\nL2 Regularization Coefficients:")
print(log_reg_l2.coef_)
```

The Output:

```
Regularization Comparison:

               Model  Accuracy  Precision    Recall  F1-Score       AUC
0  L1 Regularization  0.991228   0.986301  1.000000  0.993103  0.996693
1  L2 Regularization  0.982456   0.986111  0.986111  0.986111  0.995701

L1 Regularization Coefficients:
[[ 0.         -0.39717918  0.          0.          0.          0.
   0.         -0.48920021  0.          0.08063071 -2.16784201  0.02456843
   0.          0.         -0.19885608  0.69834099  0.         -0.07261043
   0.          0.19099441 -1.24872145 -1.2054557   0.         -3.52989723
  -0.60972548  0.         -0.67794046 -1.70545338 -0.66377651  0.        ]]

L2 Regularization Coefficients:
[[-0.49311166 -0.55620444 -0.46098799 -0.54817066 -0.19610022  0.66098506
  -0.61843898 -0.7058947  -0.17512683  0.17921326 -1.08792868  0.25162063
  -0.54628003 -0.95809577 -0.16441023  0.65059928  0.1772272  -0.42810634
   0.34924358  0.42338817 -0.94600048 -1.24227216 -0.76480696 -0.97928219
  -0.75956669  0.04956151 -0.82718017 -0.94529551 -0.9287294  -0.18147768]]
```