```
#!/usr/bin/env python
# coding: utf-8

# In[55]:


from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

# Load the wine dataset
data = load_wine()
X = data.data
y = data.target

# Filter out only the first two classes for binary classification
binary_filter = y < 2
X_binary = X[binary_filter]
y_binary = y[binary_filter]

# Standardize features by removing the mean and scaling to unit variance
scaler = StandardScaler()
X_std = scaler.fit_transform(X_binary)

# Add intercept term to the feature matrix
X_with_intercept = np.column_stack((np.ones(X_std.shape[0]), X_std))

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

X_train.shape, y_train.shape, X_test.shape, y_test.shape


# In[56]:


# Initialize the logistic regression model with no regularization (C is set to a very
high value)
logistic_model_with_intercept = LogisticRegression(penalty='none', fit_intercept=False,
solver='lbfgs', max_iter=1000)

# Fit the logistic regression model on the training data with intercept
logistic_model_with_intercept.fit(X_train, y_train)

# Calculate the final logistic loss on the training data with intercept
def logistic_loss_with_intercept(w, X, y):
    z = X.dot(w)
    log_likelihood = -y * z + np.log(1 + np.exp(z))
    return np.sum(log_likelihood)

# We need to flatten the coefficients to pass them to the loss function correctly
final_loss_standard_logistic_with_intercept = logistic_loss_with_intercept(
    logistic_model_with_intercept.coef_.flatten(), X_train, y_train
)

final_loss_standard_logistic_with_intercept


# In[57]:


from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss
log_reg = LogisticRegression(C=1e5, solver='lbfgs', max_iter=1000)
log_reg.fit(X_train, y_train)
```

```
predictions_proba = log_reg.predict_proba(X_train)
final_loss_standard_logistic_with_intercept = log_loss(y_train, predictions_proba)
print(f"Final Loss L*: {final_loss}")


# In[58]:


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

def logistic_loss(w, X, y):
    """
    Compute the logistic loss function.
    """
    predictions = sigmoid(X @ w)
    loss = -np.mean(y * np.log(predictions) + (1 - y) * np.log(1 - predictions))
    return loss

def logistic_loss_partial_derivative(w, X, y, i):
    """
    Compute the partial derivative of the logistic loss function with respect to the i-th
coordinate.
    """
    predictions = sigmoid(X @ w)
    error = predictions - y
    partial_derivative = np.mean(error * X[:, i])
    return partial_derivative

# Testing the logistic_loss and logistic_loss_partial_derivative functions
w_test = np.zeros(X_train.shape[1])
print("Logistic Loss on test weights:", logistic_loss(w_test, X_train, y_train))
print("Partial Derivative with respect to the first coordinate:",
logistic_loss_partial_derivative(w_test, X_train, y_train, 0))


# In[59]:


np.random.seed(42)


def update_coordinate(w, X, y, i, learning_rate):
    """
    Update the i-th coordinate of w using the partial derivative of the logistic loss.
    """
    w_new = np.copy(w)
    w_new[i] -= learning_rate * logistic_loss_partial_derivative(w, X, y, i)
    return w_new

def coordinate_descent_lr_decay(X, y, update_rule, initial_lr=0.1, max_iter=1000,
tolerance=1e-5, patience=5, lr_decay=0.9, decay_interval=50):
    w = np.zeros(X.shape[1])
    loss_history = []
    no_improvement_count = 0
    best_loss = np.inf
    learning_rate = initial_lr

    for iteration in range(max_iter):
        w = update_rule(w, X, y, learning_rate)
        loss = logistic_loss(w, X, y)
        loss_history.append(loss)

        # Early stopping logic
        if loss < best_loss - tolerance:
            best_loss = loss
            no_improvement_count = 0
        else:
            no_improvement_count += 1
```

```
        # Learning rate decay
        if iteration % decay_interval == 0 and iteration > 0:
            learning_rate *= lr_decay

        if no_improvement_count >= patience:
            print(f"Early stopping after {iteration + 1} iterations.")
            break

    return w, loss_history

# Define the specific update rules for each coordinate descent variant
def random_update(w, X, y, learning_rate):
    i = np.random.randint(X.shape[1])
    return update_coordinate(w, X, y, i, learning_rate)

def cyclic_update(w, X, y, learning_rate):
    for i in range(X.shape[1]):
        w = update_coordinate(w, X, y, i, learning_rate)
    return w

def greedy_update(w, X, y, learning_rate):
    gradients = np.array([logistic_loss_partial_derivative(w, X, y, i) for i in
range(X.shape[1])])
    i = np.argmax(np.abs(gradients))
    return update_coordinate(w, X, y, i, learning_rate)

# Let's run each method for a fixed number of iterations and record the final loss
iterations = 5000
learning_rate = 0.1

w_random, loss_history_random = random_coordinate_descent(X_with_intercept, y_binary,
iterations, learning_rate)
w_cyclic, loss_history_cyclic = cyclic_coordinate_descent(X_with_intercept, y_binary,
iterations, learning_rate)
w_greedy, loss_history_greedy = greedy_coordinate_descent(X_with_intercept, y_binary,
iterations, learning_rate)



# In[60]:


import matplotlib.pyplot as plt

# Plot the loss history for the random coordinate descent with learning rate decay
plt.figure(figsize=(14, 7))

# Random Coordinate Descent with Learning Rate Decay
plt.plot(loss_history_random, label='Random Coordinate Descent with LR Decay',
color='blue')

# Cyclic Coordinate Descent with Learning Rate Decay
plt.plot(loss_history_cyclic, label='Cyclic Coordinate Descent with LR Decay',
color='red')

# Greedy Coordinate Descent with Learning Rate Decay
plt.plot(loss_history_greedy, label='Greedy Coordinate Descent with LR Decay',
color='green')

# Standard Logistic Regression Solver
# We need to generate a constant line for the standard logistic regression solver loss
standard_loss_line = [final_loss_standard_logistic_with_intercept] *
len(loss_history_random)
plt.plot(standard_loss_line, label='Standard Logistic Regression Loss (L*)', linestyle='-
-', color='black')

plt.title('Loss Decrease Over Iterations (t)')
plt.xlabel('Iteration (t)')
```

```
plt.ylabel('Logistic Loss (L(w_t))')
plt.legend()
plt.grid(True)
plt.savefig("plot.png")
plt.show()


# In[61]:


(loss_history_random[-1], loss_history_cyclic[-1], loss_history_greedy[-1])


# In[ ]:
```