In [ ]:
```
from ucimlrepo import fetch_ucirepo
import pandas as pd
import random as ran
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, log_loss
import warnings
warnings.filterwarnings('ignore')

# fetch dataset
wine = fetch_ucirepo(id=109)

# data (as pandas dataframes)
X = wine.data.features
y = wine.data.targets
```

In [ ]:
```
#from sklearn.preprocessing import MinMaxScaler
#scaler = MinMaxScaler()
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_norm = scaler.fit_transform(X)

#picking two cases of wine to do sklearn baseline logistic regression
selected_y = y[y['class'] != 3]
selected_X = X_norm[selected_y.index]
selected_X = np.array(selected_X)
selected_y = np.array(selected_y).flatten() - 1
print(selected_X, selected_y)
```

```
[[ 1.51861254 -0.5622498   0.23205254 ...  0.36217728  1.84791957
   1.01300893]
 [ 0.24628963 -0.49941338 -0.82799632 ...  0.40605066  1.1134493
   0.96524152]
 [ 0.19687903  0.02123125  1.10933436 ...  0.31830389  0.78858745
   1.39514818]
 ...
 [-1.49543397 -0.18523128  1.51142186 ...  0.05506357 -0.2424958
  -0.89450282]
 [-0.77898029 -0.63406285 -0.24314178 ... -0.29592353  0.23773476
  -1.28938005]
 [-1.18661773  1.76269775  0.0492855  ... -0.7346574  -0.05887823
  -0.53147054]] [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

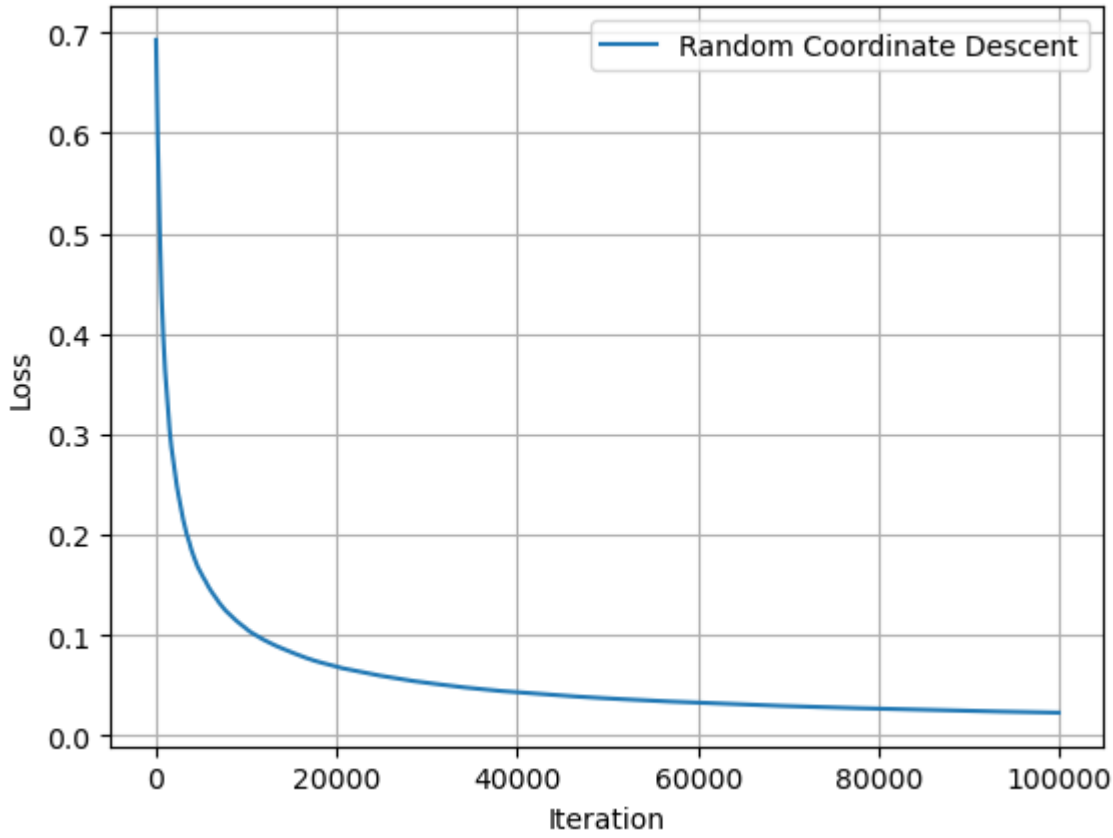
In [ ]:
```
model = LogisticRegression(penalty='none', solver='lbfgs', max_iter=100000)
model.fit(selected_X, selected_y)
logistic_loss = log_loss(selected_y, model.predict_proba(selected_X))

print(f"Logistic Loss L*: {logistic_loss}")
```

```
Logistic Loss L*: 1.5261204098688723e-06
```

In [ ]:
```
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def random_coordinate_descent(X, y, max_iter=100000, learning_rate=0.01):
    m, n = X.shape
    loss_history = []
    iteration_history = []
```

```
    w = np.zeros(n)

    for iter in range(max_iter):
        i = ran.choice(range(n))
        y_pred = sigmoid(X.dot(w))
        diff = y_pred - y
        grad = diff.dot(X[:, i]) / m
        w[i] -= learning_rate * grad

        if iter % 100 == 0:
            loss = log_loss(y, sigmoid(X.dot(w)))
            loss_history.append(loss)
            iteration_history.append(iter)

            if loss < 0.000001:
                break

    return w, loss_history, iteration_history


weights, loss_history, iteration_history = random_coordinate_descent(selecte
```

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(iteration_history, loss_history, label = 'Random Coordinate Descent
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```



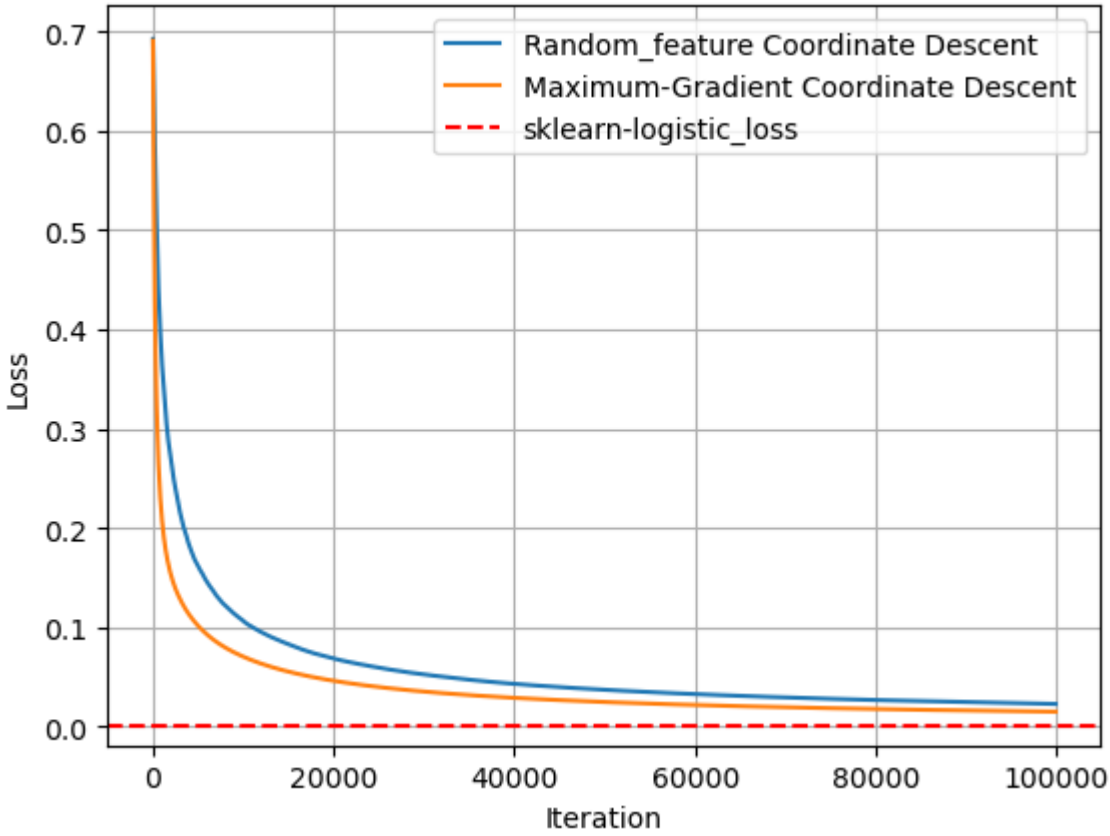
In [ ]:
```
def gradient_coordinate_descent(X, y, max_iter=100000, learning_rate=0.01):

    m, n = X.shape
    loss_history = []
    iteration_history = []
    w = np.zeros(n)
```

```
    for iter in range(max_iter):

        y_pred = sigmoid(X.dot(w))
        diff = y_pred - y
        grad = diff.dot(X) / m

        max_grad_index = np.argmax(np.abs(grad))  # Index of the largest gra
        max_grad = grad[max_grad_index]
        w[max_grad_index] -= learning_rate * max_grad

        if iter % 100 == 0:
            loss = log_loss(y, sigmoid(X.dot(w)))
            loss_history.append(loss)
            iteration_history.append(iter)

            if loss < 0.00001:
                break

    return w, loss_history, iteration_history



# Note: This implementation may require adjustments for very large datasets
# It prioritizes features based on their gradient magnitude and applies coo
# The function tracks the loss history and the number of iterations per feat
# for iterative improvement based on the feature gradients.
```

In [ ]:
```
weights_grad, loss_history_grad, iteration_history_grad = gradient_coordinat
print(len(loss_history), len(iteration_history))
```

```
1000 1000
```

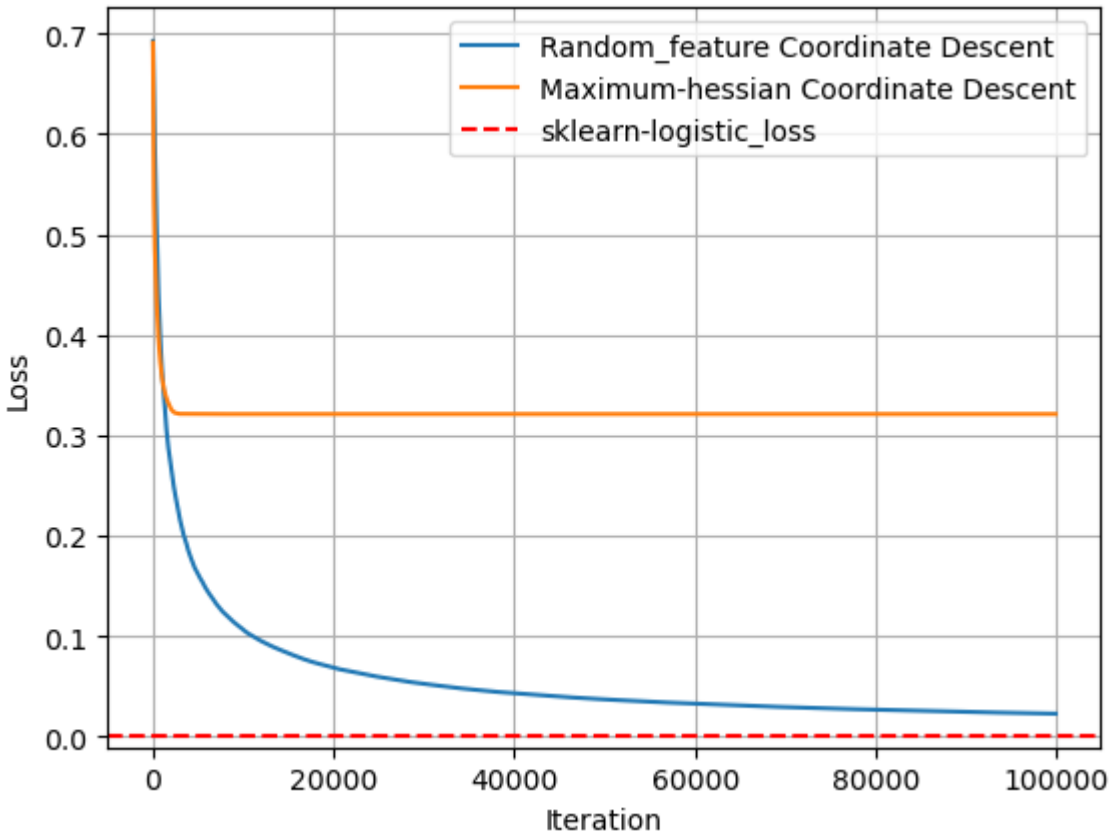
In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(iteration_history, loss_history, label = 'Random_feature Coordinate
plt.plot(iteration_history_grad, loss_history_grad, label = 'Maximum-Gradien
plt.axhline(logistic_loss, color='r', linestyle='--', label = 'sklearn-logis


plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:
```python
def hessian_coordinate_descent(X, y, max_iter=100000, learning_rate=0.01):

    m, n = X.shape
    loss_history = []
    iteration_history = []
    w = np.zeros(n)

    for iter in range(max_iter):

        y_pred = sigmoid(X.dot(w))
        diff = y_pred - y
        grad = diff.dot(X) / m

        S = np.diag(y_pred * (1 - y_pred))  # Diagonal matrix S
        hess = np.diag(X.T.dot(S).dot(X) / m)  # Hessian matrix

        max_hess_index = np.argmax(np.abs(hess))  # Index of the largest gra

        max_hess = grad[max_hess_index]
        w[max_hess_index] -= learning_rate * max_hess

        if iter % 100 == 0:
            loss = log_loss(y, sigmoid(X.dot(w)))
            loss_history.append(loss)
            iteration_history.append(iter)

            if loss < 0.00001:
                break

    return w, loss_history, iteration_history
```

In [ ]:
```python
weights_hess, loss_history_hess, iteration_history_hess = hessian_coordinate
#print(len(loss_history_hess), len(iteration_history_hess))
```

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(iteration_history, loss_history, label = 'Random_feature Coordinate
#plt.plot(iteration_history_grad, loss_history_grad, label = 'Maximum-Gradie
plt.plot(iteration_history_hess, loss_history_hess, label = 'Maximum-hessian
plt.axhline(logistic_loss, color='r', linestyle='--', label = 'sklearn-logis
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```



In [ ]:
```
def adaptive_coordinate_descent(X, y, max_iter=100000, learning_rate=0.01, d

    m, n = X.shape
    loss_history = []
    iteration_history = []
    w = np.zeros(n)

    for iter in range(max_iter):

        y_pred = sigmoid(X.dot(w))
        diff = y_pred - y
        grad = diff.dot(X) / m

        max_grad_index = np.argmax(np.abs(grad))  # Index of the largest gra
        max_grad = grad[max_grad_index]
        w[max_grad_index] -= learning_rate * max_grad

        learning_rate = learning_rate * (decay_rate)

        if iter % 100 == 0:
            loss = log_loss(y, sigmoid(X.dot(w)))
            loss_history.append(loss)
            iteration_history.append(iter)

            if loss < 0.00001:
```

```
                break

    return w, loss_history, iteration_history

weights_adap, loss_history_adap, iteration_history_adap = adaptive_coordinat
```

In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(iteration_history, loss_history, label = 'Random_feature Coordinate
#plt.plot(iteration_history_grad, loss_history_grad, label = 'Maximum-Gradie
plt.plot(iteration_history_adap, loss_history_adap, label = 'adaptive learni
plt.axhline(logistic_loss, color='r', linestyle='--', label = 'sklearn-logis
plt.xlabel('Iteration')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()
```