CSE 251A - ML: Learning Algorithms
Project 2: Coordinate Descent

Cindy Chen

Abstract

This paper provides a report on methods of doing coordinate descent to optimize the logistic regression loss.

1. Method selection

The random coordinate descent is built by randomly choosing a dimension to update the weight by its gradient. Doing this with 100000 iterations, we would get the baseline of coordinate descent, which is the randomly selected dimension coordinate descent. We compare the three methods' performance with the baseline of random coordinate descent. Since the logistic regression loss function is convex, we thought that the gradient descent method would be a good fit for optimizing the loss by finding the maximum gradient dimension. We also came up with a second method of using the hessian of each dimension to do coordinate descent. Lastly, we thought to update the weight of coordinate descent using an adaptive learning rate. These are the three methods that we first thought of. We need in our first two methods to be able to have continuous first-order and second-order derivative, for the thrid method we don't specifically need derivatives of the function since we are only changing the learning_rate, we could use the third method in any type of cost function. Below is a detailed description of each method we experienced.

1.1. Maximum gradient coordinate descent method

In this method, we choose the coordinate. Since the logistic regression loss function is convex, we could use the gradient as the direction of updating the coordinates. We thought of choosing the dimension with the largest gradient among the 13 dimensions. Then we do coordinate descent on that chosen dimension with a learning rate of 0.01. After this iteration, we then loop back to calculating the gradient of the 13 dimensions and finding the maximum one to do coordinate descent. Iterate 100000 times to get the loss of our method and we plot it to the graph to compare the loss with the random coordinate descent.

1.2. Maximum hessian coordinate descent method

In this method, we choose the coordinate. We know that the logistic regression loss function is convex, so the hessian would exist for this function, and may help in finding the optimal loss. Similar as the first method of maximum gradient update coordinate descent. Here we use maximum hessian to find the dimension that we want to choose for doing coordinate descent on. We then update the weight using the hessian of that dimension, doing coordinate descent with a learning rate of 0.01 and 100000 iterations. We plot the iteration vs. loss of this method to the graph to compare the performance with the random coordinate descent method.

1.3. Adaptive leaning_rate coordinate descent

In this method, we choose the weight w_i. We update the learning_rate with a small decay in each iteration. Based on using the gradient coordinate descent, as we do more iterations, we will be more likely to reach the global optimal, so we decrease the learning rate to reach closer to the global optimal. We would compare the experimental result with random coordinate descent performance in loss.

2. Pseudocode

Algorithm 1 Maximum gradient coordinate descent

Input: data X, y
$loss\_history$ = [], $weight$ = [array of 0s]
for $i = 0$ in $range(max\_iteration)$ **do**
  find $gradient$ in all features
  find the $max_gradient$ in all features
  Let j = feature index that has $max_gradient$
  update weight of $dimension[j]$
  append the log_loss using the current updated weight
end for
Output: $loss\_history$

3. Convergence

Under what conditions do you think your method converges to the optimal loss? There's no need to prove anything: just give a few sentences of brief explanation.

Algorithm 2 Maximum hessian coordinate descent

Input: data X, y
$loss\_history$ = [], $weight$ = [array of 0s]
for $i = 0$ in $range(max\_iteration)$ **do**
  find $hessiant$ in all features
  find the $max_hessian$ in all features
  Let j = feature index that has $max_hessian$
  update weight of $dimension[j]$ using hessian value
  append the log_loss using the current updated weight
end for
Output: $loss\_history$

Algorithm 3 Adaptive learning_rate coordinate descent

Input: data X, y
$loss\_history$ = [], $weight$ = [array of 0s]
for $i = 0$ in $range(max\_iteration)$ **do**
  update the weight using maximum gradient descent()
  decreases learning_rate by a factor of 0.999999
  append log_loss to the loss_history
end for
Output: $loss\_history$

For the first method, since we update the coordinate in each iteration, and we choose the maximum gradient coordinate as the dimension that is being updated each time, the maximum gradient dimension would be updated and another dimension would be picked in the next iteration. As long as we have enough large number of iterations, our first method would get closer and closer to the global optimal, all the coordinates would have already been updated to get closest to the global optimal.

For the second method, is a failure method that did not pass our baseline, but is still interesting to be considered in the report.

For the third method, our gradient coordinate descent is updating only one dimension at a time, if we do not update the learning_rate, the regular gradient coordinate descent method could have gotten to a loop that never be able to find the global optimal but only circling around the optimal. But here since we decrease the learning_rate in each iteration, we would not get stuck in the loop and would find the convergence as long as we have enough iterations.

4. Experimental results

Figures 1,2 and 3 are the experimental results of my three methods for coordinate descent.

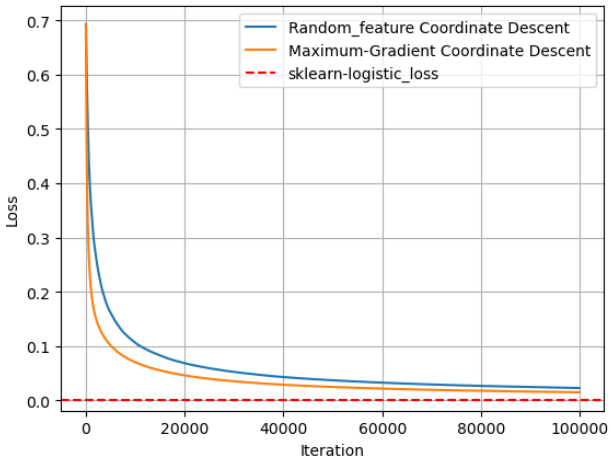


Figure 1. Maximum gradient coordinate descent performance

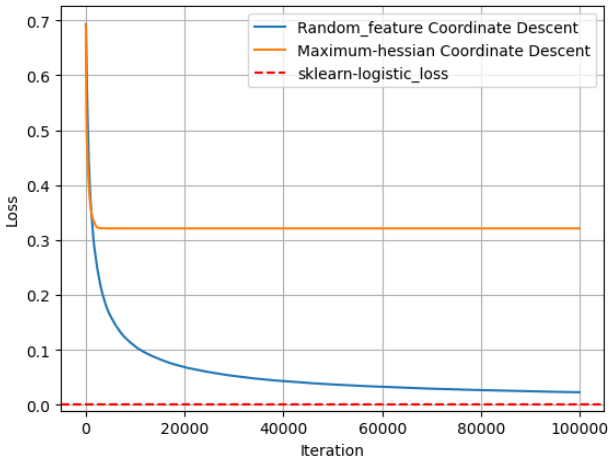


Figure 2. Maximum hessian coordinate descent performance

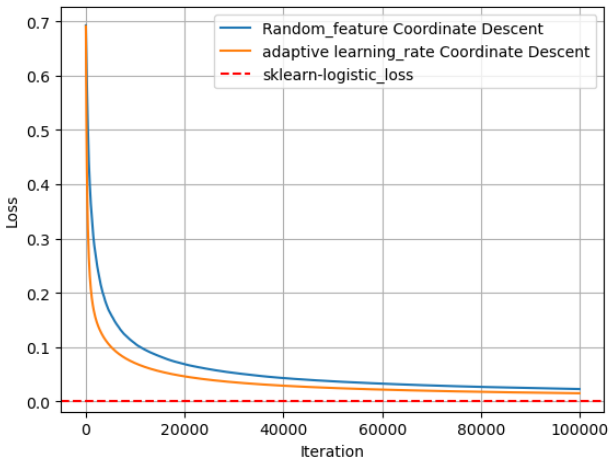


Figure 3. Adaptive learning rate coordinate descent performance

5. Critical evaluation

As we see from figure_1 of experimental results. Using the first method of maximum_gradient coordinate descent method would have a better loss than the baseline of random_feature selection coordinate descent.

From figure_2 we can see that our second method did not pass the baseline of loss. One of the issues might be that even if we update the hessian of that dimension, it would not affect the coordinate a lot and so the maximum_hessian would always be some top dimensions, and the lower dimensions would not have a chance to go through the process.

From figure_3 we can see that the adaptive learning_rate coordinate descent method is better than the random feature coordinate descent.

We can further improve our method with backtracking line search to update the learning_rate, since backtracking line search can help us to fastest find the weight that is nearest to the global optimal and then update the learning_rate adaptively to get closer and closer to the global optimal until convergence.

Also, we can use the correlation of each feature with the target to determine which dimension to choose to do coordinate descent, this might also give us a better performance as we choose the strongest correlated feature and iterate through the features that have more correlation with the target over other features, as long as we have enough iterations to go through all the dimensions, the correlation method would also converge to global optimal.

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 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 1 1]
```

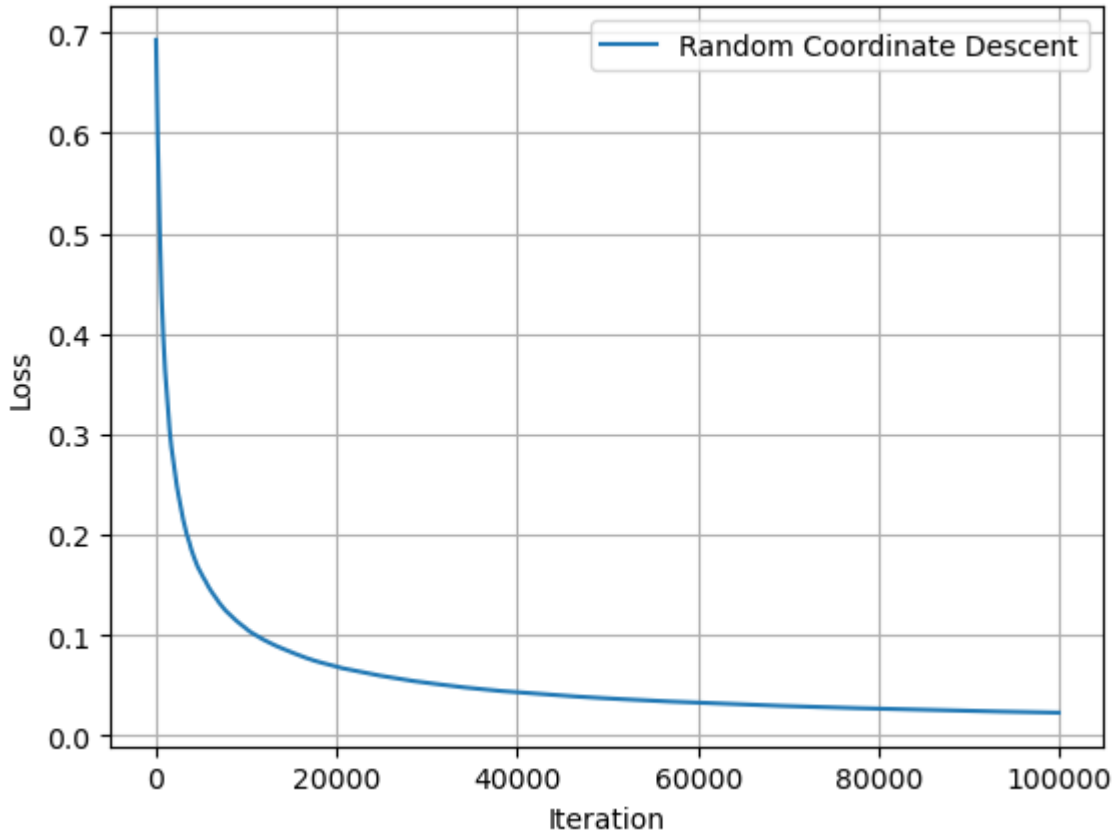
In [ ]:
```
model = LogisticRegression(penalty='none', solver='lbfgs', max_iter=100000)
model.fit(selected_X, selected_y)
logistic_loss = log_loss(selected_y, model.predict_proba(selected_X))

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()
```



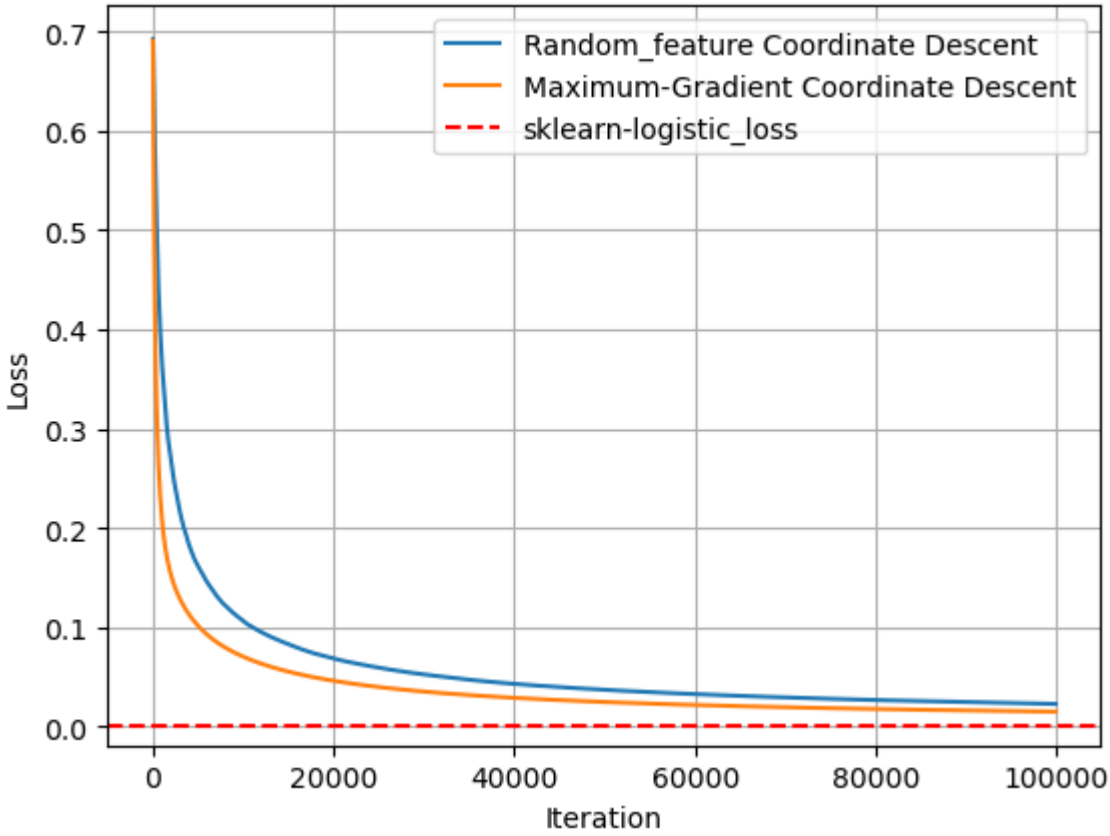
In [ ]:
```
def gradient_coordinate_descent(X, y, max_iter=100000, learning_rate=0.01):

    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
```

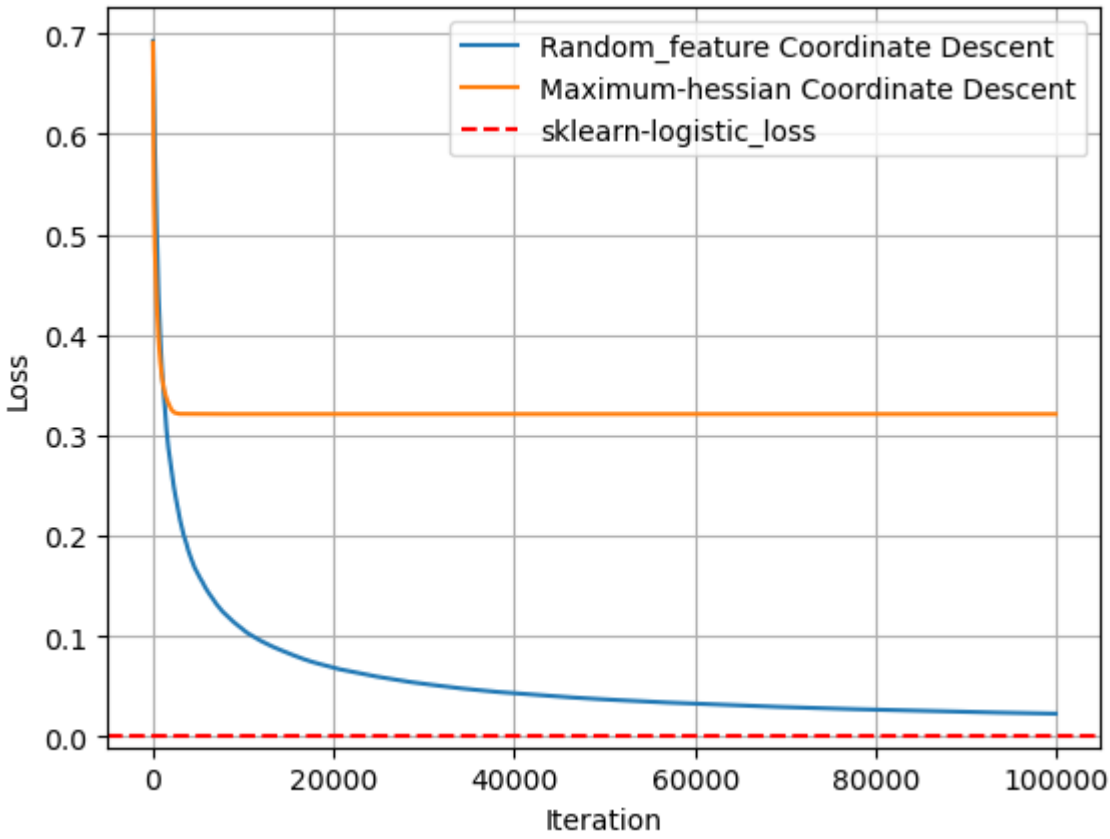
In [ ]:
```
import matplotlib.pyplot as plt
plt.plot(iteration_history, loss_history, label = 'Random_feature Coordinate
plt.plot(iteration_history_grad, loss_history_grad, label = 'Maximum-Gradien
plt.axhline(logistic_loss, color='r', linestyle='--', label = 'sklearn-logis


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()
```