PA2: Coordinate Descent

Sijie Wang
siw019@ucsd.edu

Abstract

In this project we consider a standard unconstrained optimization problem: min $L(w)$, where $L(\cdot)$ is some cost function and $w \in R^d$. A Coordinate Descent algorithm can be used to successively minimize along coordinate directions to find the minimum loss of the function. The algorithm was implemented and tested on the wine data set.

1 High-level Description

The main idea of my coordinate descent algorithm is update the coordinate with the largest absolute value of the gradient in each iterate.

Initialize w first.

Then use the greedy method to choose an index of w, such that the absolute value of the gradient is maximized.

$$i_t = \arg\max_{1 \le k \le d} |\nabla_k L(w^{t-1})| \tag{1}$$

After choosing an index, update the the value of w_i using the following formula:

$$w_i^t = w_i^{t-1} - \alpha \nabla_i L(w^{t-1}) \tag{2}$$

where α is the step size
Repeat above steps until the maximum iteration limit is reached

Note that the Coordinate Descent can work with any cost function

2 Convergence

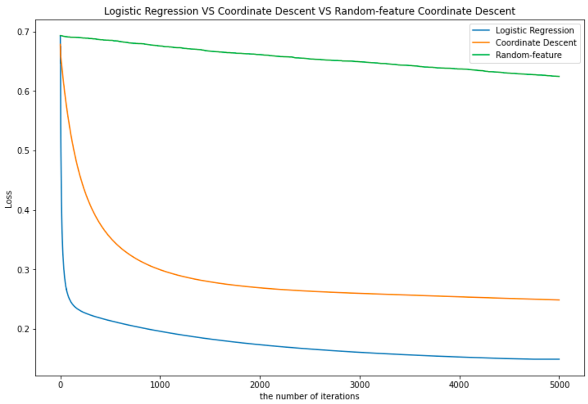
Ideally, the optimal loss can be converged when the largest absolute value of the gradient is equal to 0, which means that no matter which index is chosen, the value of w can no longer be changed, and the loss cannot be reduced any more.

3 Experimental Results

Using **scikit-learn LogisticRegression** as a standard logistic regression solver, we can get the final loss is 0.06347

To make a fair comparison, we run all three algorithms including Logistic Regression, my Coordinate Descent and Random-feature Coordinate Descent for the same number of iterations (max iter = 5,000). For my Coordinate Descent and Random-feature Coordinate Descent algorithm , I initialize w as the zero vector of shape (14,1)

The result is shown below:



4 Critical Evaluation

My coordinate descent scheme in (1) sets the step size to be fixed. For further improvement, I'll explore how to dynamically change the step size to increase the efficiency of loss reduction, such as updating the step size during iteration by calculating derivative

Besides, implementing Block Coordinate Descent [1] may be useful to furthermore improve, which take advantage of choosing a block of coordinates rather than a single coordinate at each iterate.

References

[1] Paul Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. 2001.

In [1]:

```
from sklearn.datasets import load_wine
import numpy as np
from sklearn import linear_model
import matplotlib
import matplotlib.pyplot as plt
```

In [2]:

```
wine_data, wine_labels = load_wine(True)
wine_data = wine_data[:130]
wine_labels = np.expand_dims(wine_labels[:130], axis = 1)
```

```
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.p
y:67: FutureWarning: Pass return_X_y=True as keyword args. From versio
n 0.25 passing these as positional arguments will result in an error
  warnings.warn("Pass {} as keyword args. From version 0.25 "
```

In [3]:

```
print(wine_data)
print(len(wine_data))
```

```
[[1.423e+01 1.710e+00 2.430e+00 ... 1.040e+00 3.920e+00 1.065e+03]
 [1.320e+01 1.780e+00 2.140e+00 ... 1.050e+00 3.400e+00 1.050e+03]
 [1.316e+01 2.360e+00 2.670e+00 ... 1.030e+00 3.170e+00 1.185e+03]
 ...
 [1.179e+01 2.130e+00 2.780e+00 ... 9.700e-01 2.440e+00 4.660e+02]
 [1.237e+01 1.630e+00 2.300e+00 ... 8.900e-01 2.780e+00 3.420e+02]
 [1.204e+01 4.300e+00 2.380e+00 ... 7.900e-01 2.570e+00 5.800e+02]]
130
```

Standard Logistic Regression Solver

In [4]:

```
def loss_Calculator(wine_data, wine_labels, weight, bias):
    prediction = 1.0 / (1 + np.exp(-wine_data.dot(weight.T) + bias))
    loss = 0
    for i in range(len(wine_labels)):
        if wine_labels[i] == 0:
            loss = loss - np.log(1-prediction[i])
        else:
            loss = loss - np.log(prediction[i])
    loss = loss / len(wine_data)
    return loss
```

In [5]:

```
lr_loss_arr = [0]*5000
iterations = []
for i in range(5000):
    log_reg = linear_model.LogisticRegression(C = 1e10, max_iter = i, solver = 'sag'
    log_reg.fit(wine_data,wine_labels)
    weight = log_reg.coef_
    bias = log_reg.intercept_
    loss = loss_Calculator(wine_data, wine_labels, weight, bias)
    lr_loss_arr[i] = loss[0]
    iterations.append(i)
```

```
y:329: ConvergenceWarning: The max_iter was reached which means the co
ef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.p
y:72: DataConversionWarning: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of y to (n_samples, ), for e
xample using ravel().
  return f(**kwargs)
/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_sag.p
y:329: ConvergenceWarning: The max_iter was reached which means the co
ef_ did not converge
  warnings.warn("The max_iter was reached which means "
/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.p
y:72: DataConversionWarning: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of y to (n_samples, ), for e
xample using ravel().
  return f(**kwargs)
/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_sag.p
y:329: ConvergenceWarning: The max_iter was reached which means the co
ef_ did not converge
```

In [6]:

```
from sklearn.metrics import log_loss
log = linear_model.LogisticRegression().fit(wine_data,wine_labels)
final_loss = log_loss(wine_labels,log.predict_proba(wine_data))
print('final loss L* is', final_loss)
```

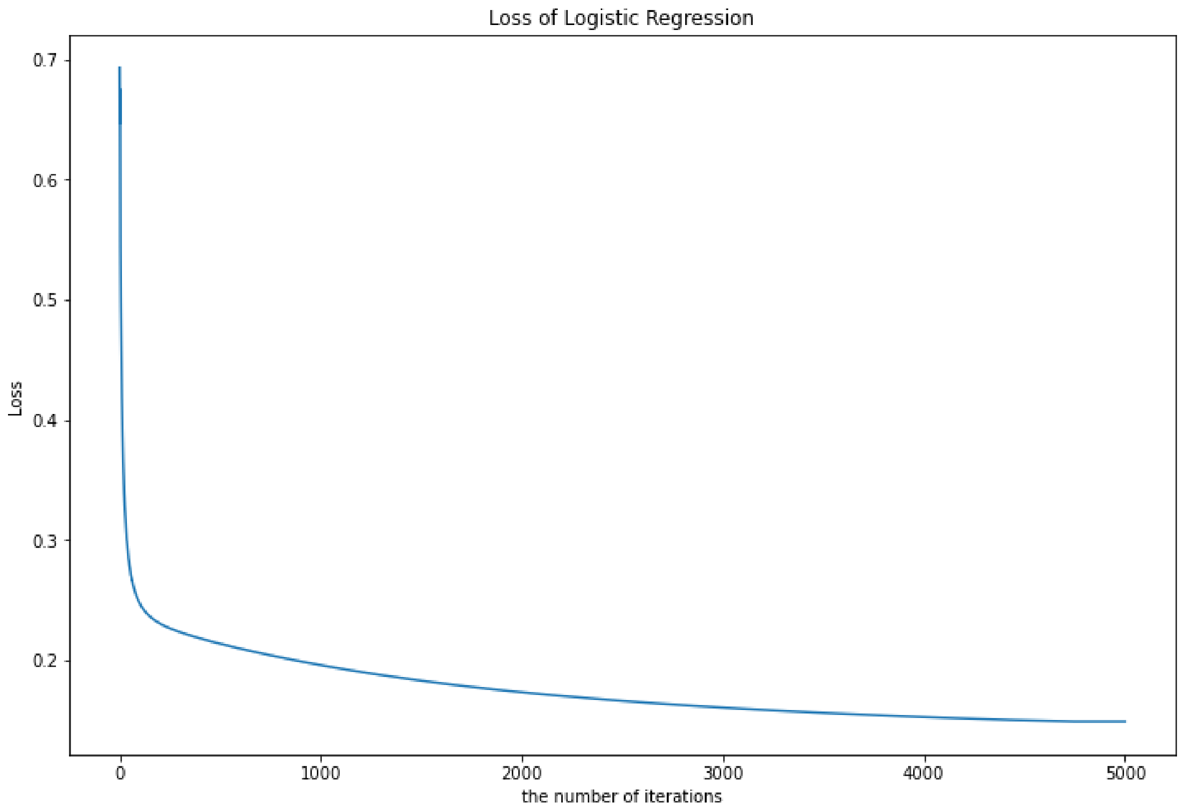
```
final loss L* is 0.06346882678105432

/opt/anaconda3/lib/python3.8/site-packages/sklearn/utils/validation.p
y:72: DataConversionWarning: A column-vector y was passed when a 1d ar
ray was expected. Please change the shape of y to (n_samples, ), for e
xample using ravel().
  return f(**kwargs)
/opt/anaconda3/lib/python3.8/site-packages/sklearn/linear_model/_logis
tic.py:762: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as show
n in:
    https://scikit-learn.org/stable/modules/preprocessing.html (http
s://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic
-regression (https://scikit-learn.org/stable/modules/linear_model.html
#logistic-regression)
  n_iter_i = _check_optimize_result(
```

In [7]:

```
x = [i for i in range(len(lr_loss_arr))]
plt.figure(figsize = (12,8))
plt.plot(x, lr_loss_arr)
plt.xlabel('the number of iterations')
plt.ylabel('Loss')
plt.title('Loss of Logistic Regression')
plt.show()
```



My Coordinate Descent Method

In [8]:

```
def loss_Calculator_CD(wine_data, wine_labels, weight):
    prediction = 1.0 / (1 + np.exp(-wine_data.dot(weight)))
    loss = 0
    for i in range(len(wine_labels)):
        if wine_labels[i] == 0:
            loss = loss - np.log(1-prediction[i])
        else:
            loss = loss - np.log(prediction[i])
    loss = loss / len(wine_labels)
    return loss
```

In [9]:

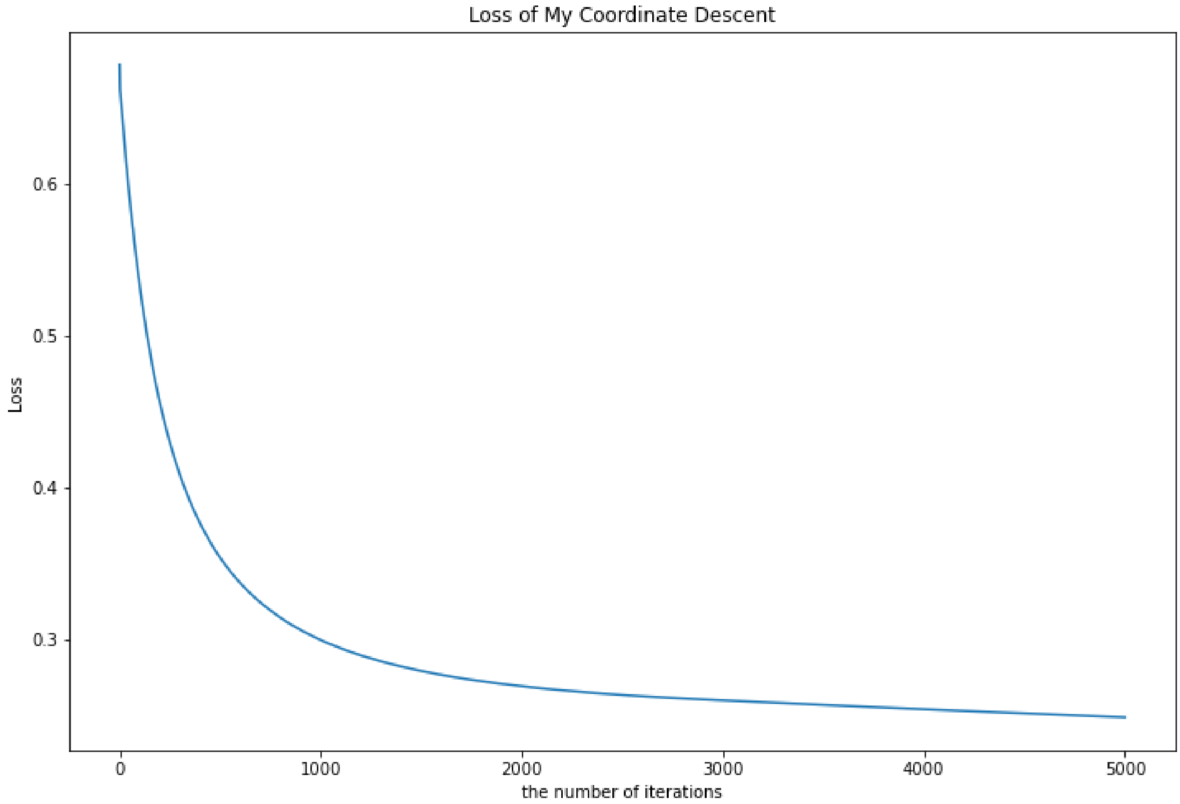
```
def my_fit(wine_data, wine_labels, step_size, maximum):
    wine_data = np.insert(wine_data, 0, 1, axis = 1)
    weight = np.zeros((len(wine_data[0]),1))
    loss_arr = []
    counter = 0
    iterations = []
    while counter < maximum:
        prediction = 1.0 / (1 + np.exp(-wine_data.dot(weight)))
        derivate = np.sum((prediction - wine_labels) * wine_data, axis=0) / len(wine
        weight_idx = np.argmax(np.abs(derivate))
        weight[weight_idx] = weight[weight_idx] - step_size * derivate[weight_idx]
        loss = loss_Calculator_CD(wine_data,wine_labels,weight)
        loss_arr.append(loss)
        iterations.append(counter)
        counter = counter + 1
    return loss_arr, iterations
```

In [10]:

```
my_loss_arr,iterations = my_fit(wine_data, wine_labels, step_size = 1e-5, maximum =
```

In [11]:

```
plt.figure(figsize = (12,8))
plt.plot(iterations, my_loss_arr)
plt.xlabel('the number of iterations')
plt.ylabel('Loss')
plt.title('Loss of My Coordinate Descent')
plt.show()
```



Random-feature Coordinate Descent

In [12]:

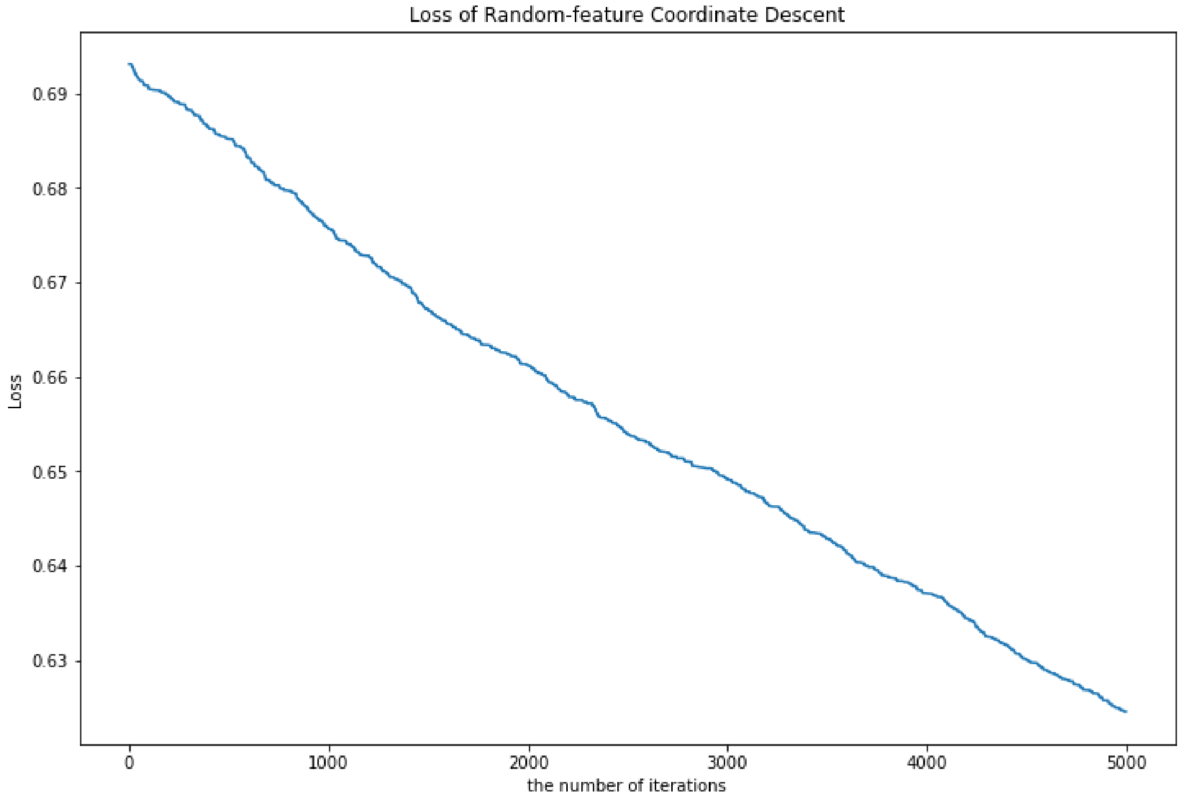
```
def random_fit(wine_data, wine_labels, step_size, maximum):
    wine_data = np.insert(wine_data, 0, 1, axis = 1)
    weight = np.zeros((len(wine_data[0]),1))
    loss_arr = []
    counter = 0
    iterations = []
    while counter < maximum:
        prediction = 1.0 / (1 + np.exp(-wine_data.dot(weight)))
        derivate = np.sum((prediction - wine_labels) * wine_data, axis=0) / len(wine
        weight_idx = np.random.randint(0, 13)
        weight[weight_idx] = weight[weight_idx] - step_size * derivate[weight_idx]
        loss = loss_Calculator_CD(wine_data,wine_labels,weight)
        loss_arr.append(loss)
        iterations.append(counter)
        counter = counter + 1
    return loss_arr, iterations
```

In [13]:

```
random_loss_arr, iterations = random_fit(wine_data, wine_labels, step_size = 1e-4, m
```

In [14]:

```
plt.figure(figsize = (12,8))
plt.plot(iterations, random_loss_arr)
plt.xlabel('the number of iterations')
plt.ylabel('Loss')
plt.title('Loss of Random-feature Coordinate Descent')
plt.show()
```



In [15]:

```
plt.figure(figsize = (12,8))
logReg_loss, = plt.plot(iterations, lr_loss_arr)
my_loss, = plt.plot(iterations, my_loss_arr)
random_loss, = plt.plot(iterations, random_loss_arr)
plt.legend([logReg_loss, my_loss, random_loss], ['Logistic Regression', 'Coordinate
plt.xlabel('the number of iterations')
plt.ylabel('Loss')
plt.title('Logistic Regression VS Coordinate Descent VS Random-feature Coordinate De
plt.show()
```