Logistic Regression Via Coordinate Descent

Chunlin Chen
Department of Electrical and Computer Engineering
University of California, San Diego
chc126@ucsd.edu

Abstract

In convex optimization problems, gradient descent (GD) and stochastic gradient descent (SGD) have been proven effective for differentiable objective functions. In this paper, we pay attention to a different approach, coordinate descent (CD), which updates one certain coordinate of the variable vector at each iteration, and compare the performances under different coordinate selection methods on a classic logistic regression problem.

1 Coordinate Descent Method

We consider a standard unconstrained optimization problem:

$$\min_{\omega \in \mathbb{R}^d} L(\omega) \tag{1}$$

Suppose the objective funtion $L(\cdot) : \mathbb{R}^d \to \mathbb{R}$ is continuous and differentiable everywhere, then a general coordinate descent algorithm can be described as follows(Wright, 2015):

Algorithm 1 General Coordinate Descent Pipeline

1: Initialization: $k \leftarrow 0, \omega_0 \in \mathbb{R}^d$
2: **repeat**
3: Choose index $i_k \in \{1, 2, \cdots, d\}$
4: $\omega^{k+1} \leftarrow \omega^k - \alpha_k [\nabla L(\omega)]_{i_k} \mathbf{e}_{i_k}$
5: $k \leftarrow k + 1$
6: **until** convergence

We use $[\nabla L(\omega)]_i$ to denote the ith component of the gradient $\nabla L(\omega)$, and $\mathbf{e}_i$ denotes the vector with a 1 in the ith coordinate and 0's elsewhere. Note that the update of the value of the selected coordinates is just applying gradient descent with respect to scalars. We will focus more on the selection of coordinates to be updated. The trivial method is to choose coordinates uniformly at random, i.e. random-feature coordinate descent. In the following we will discuss two other methods we implemented.

1.1 Cyclic Coordinate Descent

One natural way of coordinate selection is to choose coordinates in a cyclic ordering:

$$i_k = (k + 1) \mod d, k = 0, 1, 2, \cdots \tag{2}$$

Therefore the cyclic coordinate descent algorithm can be described as follows:

Algorithm 2 Cyclic Coordinate Descent

1: Initialization: $k \leftarrow 0, \omega_0 \in \mathbb{R}^d$
2: **repeat**
3: $i_k = (k + 1) \mod d$
4: $\omega^{k+1} \leftarrow \omega^k - \alpha_k [\nabla L(\omega)]_{i_k} \mathbf{e}_{i_k}$
5: $k \leftarrow k + 1$
6: **until** convergence

1.2 Max Gradient Magnitude Coordinate Descent

Another way of coordinate selection is to choose the coordinate that has the steepest direction of gradient to update, i.e. the coordinate that has the largest absolute value of gradient, which will empirically accelerate the convergence rate:

$$i_k = \underset{i}{argmax} \, |[\nabla L(\omega)]_i| \tag{3}$$

Algorithm 3 Max Gradient Magnitude Coordinate Descent

1: Initialization: $k \leftarrow 0, \omega_0 \in \mathbb{R}^d$
2: **repeat**
3: $i_k = \underset{i}{argmax} \, |[\nabla L(\omega)]_i|$
4: $\omega^{k+1} \leftarrow \omega^k - \alpha_k [\nabla L(\omega)]_{i_k} \mathbf{e}_{i_k}$
5: $k \leftarrow k + 1$
6: **until** convergence

We call this method the max gradient magnitude coordinate descent.

1.3 Convergence

Both of our methods will undoubtedly converge to the optimal loss as long as given sufficient iteration steps, since our methods are essentially applying general gradient descent to a selected vector coordinate every iteration instead of updating the whole, therefore the value of the objective function will decrease after each iteration and will eventually converge to a global optimum when the function is convex. Different coordinate selection methods determine the direction of gradient descent and only effect the convergence rates.

2 Objective Function for Logistic Regression

To simplify the derivation, the bias b is already absorbed into ω in following discussion. For binary labels $y \in \{0, 1\}$, we define the model:

$$\Pr(y = 1|\mathbf{x}) = \frac{1}{1 + e^{-\omega^\top \mathbf{x}}} \tag{4}$$

$$\Pr(y = 0|\mathbf{x}) = 1 - \Pr(y = 1|\mathbf{x}) = \frac{1}{1 + e^{\omega^\top \mathbf{x}}} \tag{5}$$

Above equations can be rewritten as:

$$\Pr(y|\mathbf{x}) = \left(\frac{1}{1 + e^{-\omega^\top \mathbf{x}}}\right)^y \left(\frac{1}{1 + e^{\omega^\top \mathbf{x}}}\right)^{1-y} \tag{6}$$

Given data $(\mathbf{x}^{(1)}, y^{(1)}), \cdots, (\mathbf{x}^{(n)}, y^{(n)}) \in \mathbb{R}^d \times \{0, 1\}$, the loss function for logistic regression is:

$$L(\omega) = -\sum_{i=1}^{n} \ln \Pr(y^{(i)}|\mathbf{x}^{(i)}) = \sum_{i=1}^{n} y^{(i)} \ln\left(1 + e^{-\omega^\top \mathbf{x}^{(i)}}\right) + (1 - y^{(i)}) \ln\left(1 + e^{\omega^\top \mathbf{x}^{(i)}}\right) \tag{7}$$

Take the gradient of $L(\omega)$ with respect to ω:

$$\nabla L(\omega) = \sum_{i=1}^{n} \left(\frac{1}{1 + e^{-\omega^\top \mathbf{x}^{(i)}}} - y^{(i)}\right) \mathbf{x}^{(i)} \tag{8}$$

3 Experimental Results

We test our method on the `wine` dataset. First we run a standard logistic regression solver from `scikit learn` to obtain the optimal loss $L^* = 2.37 \times 10^{-6}$. Then, we run the three methods, random, cyclic, and max gradient magnitude, and record the loss curves. The result are shown in Figure 1.

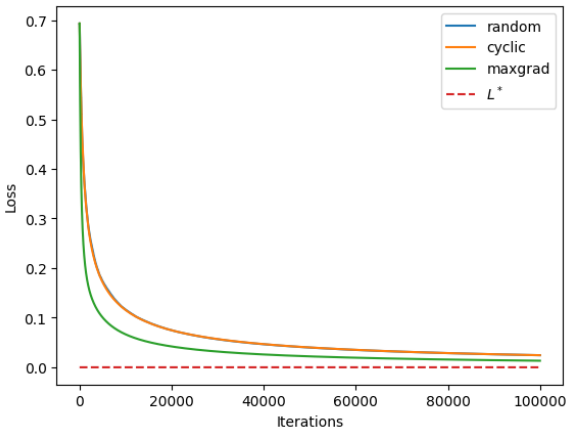

Figure 1: Loss Curves of Different Coordinate Descent Methods.

4 Critical Evaluation

From Figure 1, we can find that the losses of all three methods decrease with iteration step and eventually asymptote to L^*. On top of that, we can see that the loss curve of the max gradient magnitude method has the fastest convergence rate, which indicates the significance of coordinate selection in its effect on convergence rate. The loss curves of cyclic ordering and uniformly random selection are nearly coincident, showing that as the number of iterations are large enough, these two selection methods are essentially traversals of all coordinates and demonstrate relatively low convergence rate.

There exist much scope for improvement of our methods. Note that our loss function does not include the regularization term, which fails to penalize the complexity of ω and could lower the convergence rate. For further improvement, we can introduce regularization into our objective function. Moreover, our methods suppose the objective function must be differentiable everywhere, which is not applicable to some common regularization techniques, e.g. the l_1 or Lasso regularization. In such cases, subgradient methods may be applied to solve this problem.

References

Stephen J Wright. 2015. Coordinate descent algorithms. *Mathematical programming*, 151(1):3–34.

A Code

```
from sklearn.datasets import load_wine
from sklearn.linear_model import
    LogisticRegression
```

```
from sklearn.metrics import log_loss
from sklearn.preprocessing import
    StandardScaler
import numpy as np
import matplotlib.pyplot as plt

data = load_wine()
XTrain, yTrain = data.data[:130], data.
    target[:130]
XTrain = StandardScaler().fit_transform(
    XTrain)

standard_lr = LogisticRegression(penalty
    =None, solver='newton-cg').fit(
    XTrain, yTrain)
loss_opt = log_loss(yTrain, standard_lr.
    predict_proba(XTrain))

def sigmoid(x):
  return 1.0 / (1 + np.exp(-x))

def CD(selection, lr=1e-2, iters=100000)
    :
  X = np.insert(XTrain, 0, 1, axis=1)
  y = yTrain[:, np.newaxis]
  w = np.zeros((X.shape[1], 1))
  losses = []
  loss_0 = log_loss(y, sigmoid(X @ w))
  losses.append(loss_0)
  print('Initial Loss: {}'.format(loss_0
    ))
  iter = 0
  while iter < iters:
    grad = np.sum((sigmoid(X @ w) - y) *
     X, axis=0) / X.shape[0]
    if selection == 'cyclic':
      index = iter % w.shape[0]
    elif selection == 'maxgrad':
      index = np.argmax(np.abs(grad))
    elif selection == 'random':
      index = np.random.randint(0, 14)
    w[index] -= lr * grad[index]
    loss = log_loss(y, sigmoid(X @ w))
    losses.append(loss)
    if (iter+1) % 10000 == 0:
      print('Iteration: {}, Loss: {}'.
    format(iter+1, loss))
    iter += 1
  return losses

loss_cyclic = CD(selection='cyclic')
loss_maxgrad = CD(selection='maxgrad')
loss_rand = CD(selection='random')

step = np.arange(0,len(loss_rand))
loss_std = loss_opt * np.ones(len(
    loss_rand))
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.plot(step[:100000], loss_rand
    [:100000], label='random')
plt.plot(step[:100000], loss_cyclic
    [:100000], label='cyclic')
plt.plot(step[:100000], loss_maxgrad
    [:100000], label='maxgrad')
plt.plot(step[:100000], loss_std
    [:100000], linestyle='--', label='$L
    ^*$')
plt.legend()
plt.show()
```