Coordinate Descent for Logistic Regression

Zongze Liu

1 High level description of the coordinate descent algorithm

Coordinate descent is an optimization algorithm that successively minimizes an objective function along coordinate axis directions. For each iteration of the algorithm, the algorithm chooses a coordinate direction according to a coordinate selection rule and tries to minimize the objective function on the coordinate hyperplane.

In the optimization problem, we require the loss function $L(\cdot)$, a function of the weight vector $w \in R^d$, to be differentiable and convex. We randomly initialize the weight vector w, for example, according to the normal distribution with mean $= 1$ and variance $= 0.01$.

In each iteration t of coordinate descent, we calculate the gradient of the loss function $\nabla L(w_{t-1})$ and we greedily choose the coordinate i_t such that $\nabla L(w_{t-1})_{i_t}$ has the largest absolute value among all coordinates $j \in \{1, ..., d\}$. Intuitively, we update the weights w in the coordinate axis direction along which the loss function decreases the most. In mathematical notations, this is

$$i_t = \underset{j \in \{1,...,d\}}{\text{argmax}} \, |\nabla L(w_{t-1})_j|.$$

Then the algorithm updates w_{i_t} with the rule

$$w_{i_t} = w_{i_{t-1}} - \alpha_t \nabla L(w_{t-1})_j.$$

The algorithm finds the step size α_t for iteration t using backtracking line search. We first set the maximum step size $\alpha := 1$, shrinkage factor $\beta := 0.9$ and control parameter $c := 0.5$. Then we continue to update $\alpha := \beta * \alpha$ until the inequality

$$L(w_{t-1} - \nabla L(w_{t-1})) \leq L(w_{t-1}) - \alpha c ||\nabla L(w_{t-1})||^2$$

is satisfied. We set the step size for iteration t as $\alpha_t = \alpha$.

2 Convergence

Under the assumption that $L(w) : R^d \to R$ is differentiable and convex and $\nabla L(w)$ is Lipschitz continuous or differentiable, backtracking line search will guarantee the convergence of our coordinate descent algorithm to the optimal loss.

3 Experimental Results

We run our experiments on the wine dataset with labels 0 and 1. This results in a new dataset with size$= 130$. We apply z-score normalization to the dataset so that the scikit-learn logistic regression classifier with the lbfgs solver will converge without regularization. The final loss on the scikit-learn logistic regression classifer is $L^* = 9.480$*1e-5. We will use L^* as our target loss value for the coordinate descent algorithm.
We run the proposed coordinate descent algorithm and a coordinate descent algorithm with random coordinate index selection. We experiment with convergence step size with backtracking line search and a fixed step size $\alpha = 0.08$. We attach the training loss curves of our experiments below. We run the algorithms for 20000 iterations and 120000 iterations respectively. Using backtracking line search, we see asymptotic convergence of both our proposed coordinate descent algorithm and random index coordinate descent algorithm after 120000 iterations. With backtracking line search, our algorithm has final loss $L_1 = 0.00023$ and the random index algorithm has final loss $L_2 = 0.00053$. In fact, we think the precision/error of our algorithm is inverse to the number of iterations. When we turn off backtracking line search and use a fixed step size, we see that both algorithms failed to converge asymptotically even after 120000 iterations. In all case, we see that our algorithm converges significantly faster and has significantly smaller loss value than random index selection (compare the loss curves at 20000 iterations).

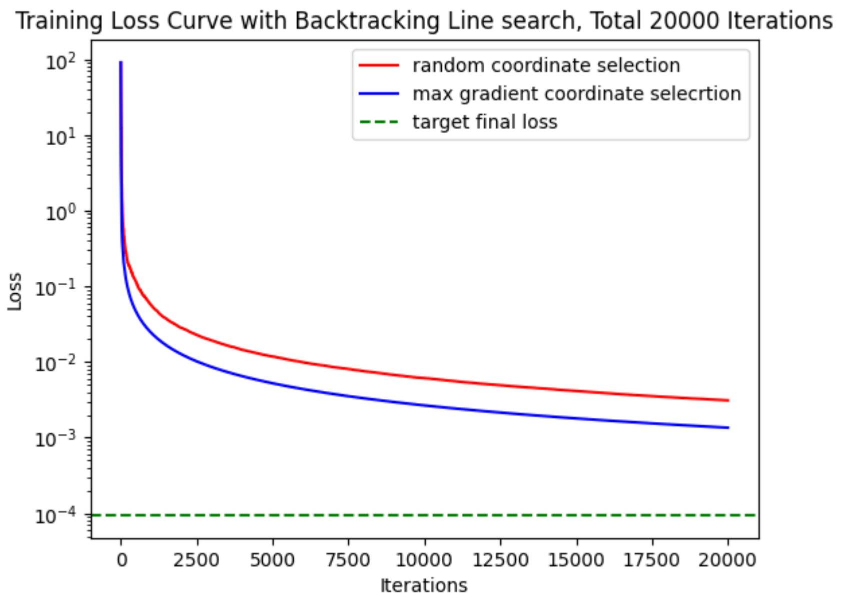


Figure 1: Experiment with backtracking line search. Loss curves of proposed coordinate descent algorithm and coordinate descent with random coordinate selection. Total 80000 iterations.

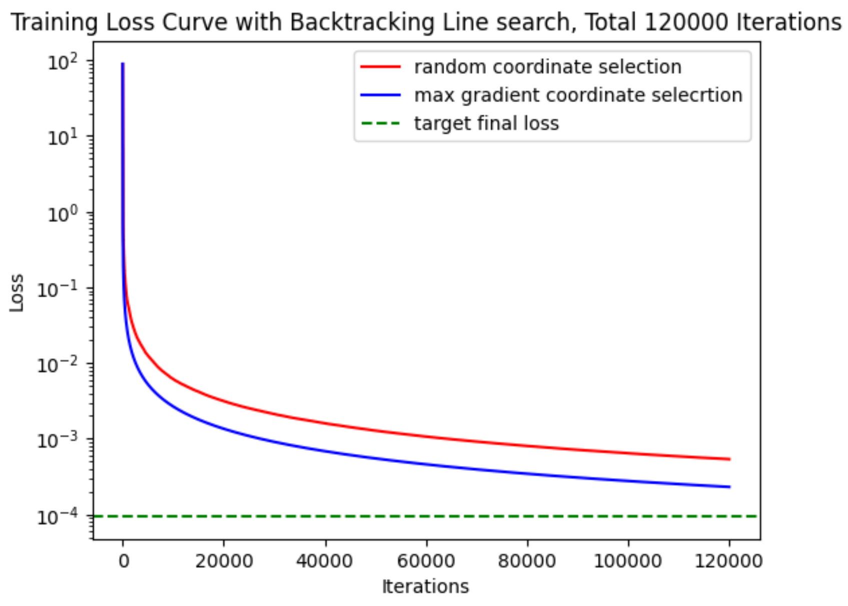


Figure 2: Experiment with backtracking line search. Loss curves of proposed coordinate descent algorithm and coordinate descent with random coordinate selection. Total 120000 iterations.

4 Critical Evaluation

Our algorithm requires the loss function to be both differentiable and convex. We can further improve the algorithm by dropping the differentiability condition and replacing the gradient with a proper subgradient. Then we update the weight with the coordinate of the subgradient with the maximal absolute value. Thus the coordinate descent method can apply to loss functions that are convex but not necessarily differentiable. We can also try to improve our coordinate descent method by updating the weights with a subset or block of gradient coordinates at a time.

5 Sparse coordinate descent

We can try to add a L_1 regularization term to the loss function, adjust the regularization constant λ according to the required sparsity k, and set weights that are smaller than a given threshold (i.e. 1e-8) to zero.

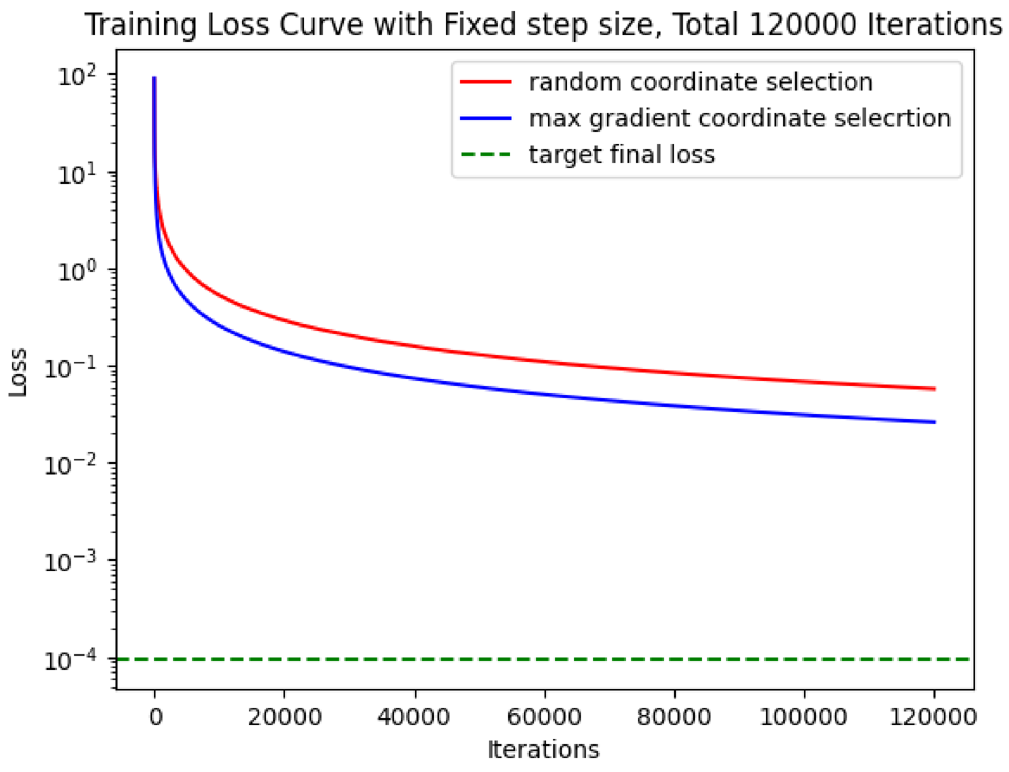


Figure 3: Experiment with fixed step size in coordinate descent. Loss curves of proposed coordinate descent algorithm and coordinate descent with random coordinate selection. Total 120000 iterations.

```
import numpy as np
import matplotlib.pyplot as plt
import sklearn.utils as utils
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_wine
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import log_loss
```

```
data = load_wine()
X = data.data
y = data.target
idx = [i for i in range(len(y)) if y[i] != 2]
X = X[idx]
y = y[idx]
X = StandardScaler().fit_transform(X)
```

```
print(y.shape)
print(X.shape)
```

```
(130,)
(130, 13)
```

```
X, y = utils.shuffle(X, y, random_state = 3)
```

```
print(y[:10])
print(X[0])
```

```
[0 1 1 1 0 0 1 0 0 0]
[ 1.05712867 -0.0890702   0.84298752 -1.11750469  0.0717917   1.33870579
  1.45593625 -1.47451854 -0.09746111  0.76271547 -1.06458328  1.27946422
  0.86830046]
```

```
X = np.hstack((np.ones((len(X),1)),X))
```

```
print(X.shape)
```

```
(130, 14)
```

```
print(X[0])
```

```
[ 1.          1.05712867 -0.0890702   0.84298752 -1.11750469  0.0717917
  1.33870579  1.45593625 -1.47451854 -0.09746111  0.76271547 -1.06458328
  1.27946422  0.86830046]
```

```
logistic_regression = LogisticRegression(penalty=None,fit_intercept=False)
logistic_regression.fit(X,y)
y_pred = logistic_regression.predict_proba(X)
weights = logistic_regression.coef_
logistic_loss = log_loss(y, y_pred, normalize=False)
```

```
print(weights.shape)
```

```
(1, 14)
```

```
print('logistic regression loss=', logistic_loss) #this is sum of all sample log loss, not averaged
```

```
logistic regression loss= 9.480169067119548e-05
```

```
print('logistic regression weights', weights)
```

```
logistic regression weights [[  0.44892414 -10.26521716  -3.64484291  -9.47794415  11.39096
   -1.50385118   0.310449    -3.3173154    1.7470717    1.66698924
   -2.49903833   1.01682515  -5.61306123 -15.49726904]]
```

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

```
def calc_loss(X,y,w):
  y = y.reshape((len(y),1))
  y_pred = sigmoid(X.dot(w.T))
  return -np.sum(np.dot(y.T, np.log(y_pred + 1e-11)) + np.dot((1-y).T, np.log(1 - y_pred + 1e-11)))
```

```
print(calc_loss(X,y,weights))
```

```
    9.480039065715029e-05
```

```
def calc_grad(X, y, w):
  y = y.reshape((len(y),1))
  y_pred = sigmoid(X.dot(w.T))
  return X.T.dot(y_pred - y)
```

```
def backtrack_line_search(X, y, w, direction, beta=0.90):
  alpha = 1.0
  loss = calc_loss(X, y, w)
  norm = np.linalg.norm(direction, ord=2)
  while True:
    lhs = calc_loss(X, y, w - alpha * direction)
    rhs = loss - alpha * 1/2.0 * (norm ** 2)
    if lhs <= rhs:
      break
    alpha = beta * alpha

  return alpha
```

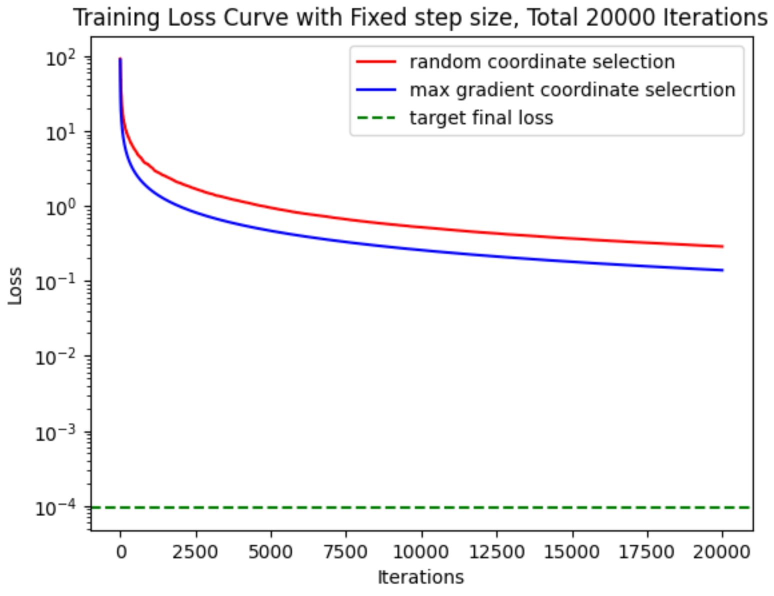
```
def coordinate_descent(X, y, max_iter=20000, rand_idx=False, backtrack_search = True):
  w = np.random.normal(0, 0.01, (1,X.shape[1]))
  #print(w.shape)
  #print(w)
  iterations = [0]
  loss_list = [calc_loss(X,y,w)]
  for i in range(max_iter):
    gradient = calc_grad(X, y, w)
    gradient = gradient.reshape((14,))
    #print(gradient.shape)
    if rand_idx:
      idx = np.random.randint(14)
    else:
      idx = np.argmax(np.absolute(gradient))
    gradient = np.array([gradient[j] if j == idx else 0 for j in range(14)])
    step_size = 0.008
    if backtrack_search:
      step_size = backtrack_line_search(X, y, w, gradient)
    w = w - step_size * gradient
    curr_loss = calc_loss(X, y, w)
    loss_list.append(curr_loss)
    iterations.append(i+1)
  print('Final loss =', loss_list[-1])
  return w, loss_list, iterations
```

```
coordinate_descent(X, y)
```

```
def plot(rand_iters, maxidx_iters, rand_losses, maxidx_losses):
    plt.title('Training Loss Curve with Fixed step size, Total ' + str(len(rand_iters)-1) + ' Iterations' )
    plt.xlabel('Iterations')
    plt.ylabel('Loss')
    plt.semilogy(rand_iters, np.array(rand_losses).reshape(-1, 1), 'r', label='test')
    plt.semilogy(maxidx_iters, np.array(maxidx_losses).reshape(-1, 1), 'b')
    plt.axhline(y=logistic_loss, color='g', linestyle='--')
    plt.legend(('random coordinate selection', 'max gradient coordinate selecrtion', 'target final loss'))
    plt.show()
```

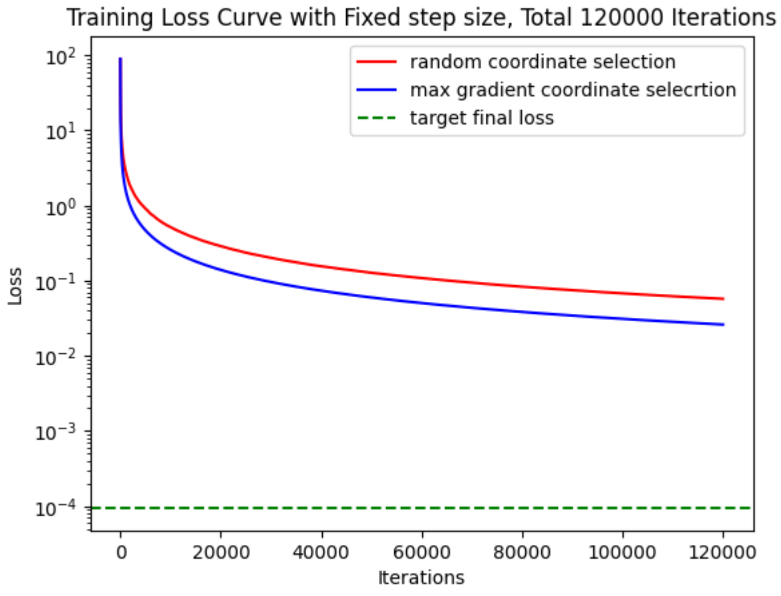
```
def plot_line_search(rand_iters, maxidx_iters, rand_losses, maxidx_losses):
    plt.title('Training Loss Curve with Backtracking Line search, Total ' + str(len(rand_iters)-1) + ' Iterations')
    plt.xlabel('Iterations')
    plt.ylabel('Loss')
    plt.semilogy(rand_iters, np.array(rand_losses).reshape(-1, 1), 'r', label='test')
    plt.semilogy(maxidx_iters, np.array(maxidx_losses).reshape(-1, 1), 'b')
    plt.axhline(y=logistic_loss, color='g', linestyle='--')
    plt.legend(('random coordinate selection', 'max gradient coordinate selecrtion', 'target final loss'))
    plt.show()


w_max, loss_max, iter_max = coordinate_descent(X, y, backtrack_search=False)
w_rand, loss_rand, iter_rand = coordinate_descent(X, y, rand_idx=True, backtrack_search=False)
plot(iter_rand, iter_max, loss_rand, loss_max)
```

```
    Final loss = 0.1388422452897771
    Final loss = 0.28813135964521086
```



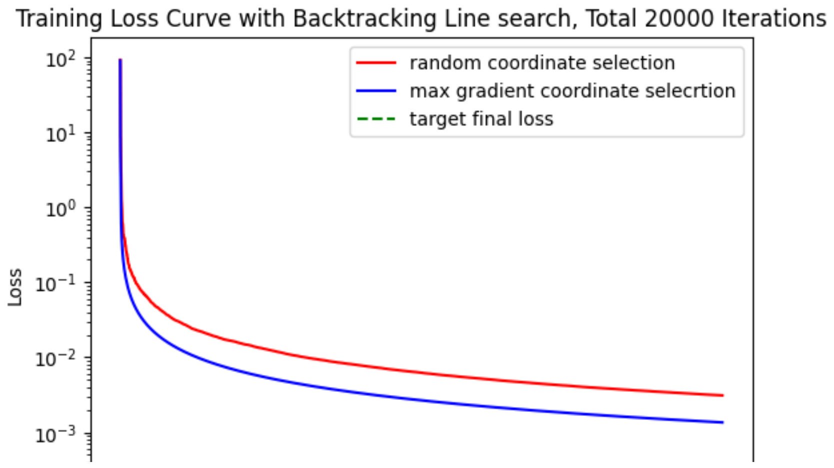
```
w_max, loss_max, iter_max = coordinate_descent(X, y, max_iter = 120000, backtrack_search=False)
w_rand, loss_rand, iter_rand = coordinate_descent(X, y, max_iter = 120000, rand_idx=True, backtrack_search=False)
plot(iter_rand, iter_max, loss_rand, loss_max)
```

```
    Final loss = 0.02608623701730965
    Final loss = 0.05734782412958636
```



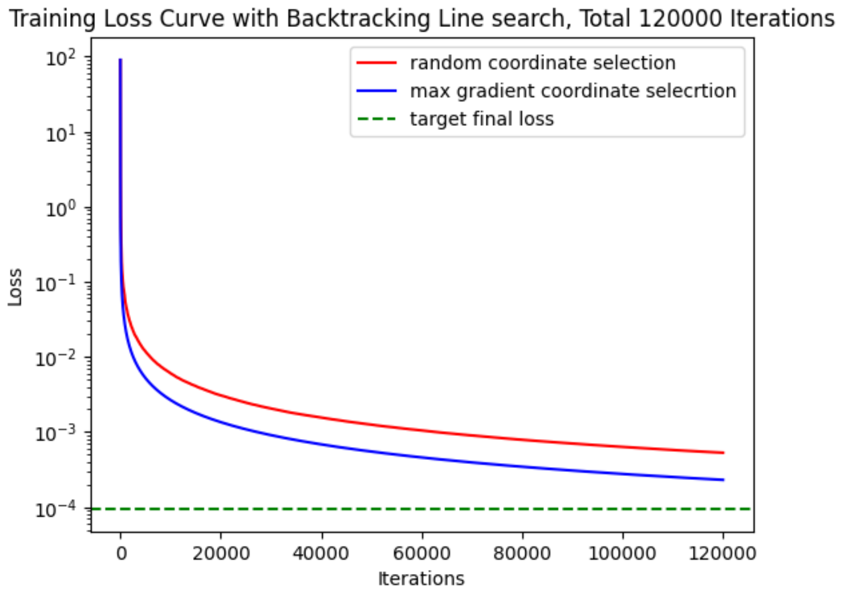
```
w_max, loss_max, iter_max = coordinate_descent(X, y)
w_rand, loss_rand, iter_rand = coordinate_descent(X, y, rand_idx=True)
plot_line_search(iter_rand, iter_max, loss_rand, loss_max)
```

```
Final loss = 0.0013522541548638444
Final loss = 0.003098677246702139
```



```
w_max, loss_max, iter_max = coordinate_descent(X, y, max_iter = 120000)
w_rand, loss_rand, iter_rand = coordinate_descent(X, y,max_iter = 120000, rand_idx=True)
plot_line_search(iter_rand, iter_max, loss_rand, loss_max)
```

```
Final loss = 0.00023065613441494468
Final loss = 0.0005298954637353273
```


Training Loss Curve with Backtracking Line search, Total 120000 Iterations
random coordinate selection
max gradient coordinate selecrtion
target final loss
Loss
Iterations
10^2
10^1
10^0
10^-1
10^-2
10^-3
10^-4