Coordinate Descent

Nidhi Dhamnani

A59012902
CSE 251A: ML - Learning Algorithms
ndhamnani@ucsd.edu

Keywords: Gradient Descent, Stochastic Gradient Descent, Logistic Regression, Steepest Descent, Cyclic Coordinate Descent

Abstract. The goal of this project is to propose a coordinate descent based algorithm to minimize the cost function $L(w)$ for logistic regression.

[1] Motivation

Logistic regression is a statistical model that uses a logistic function to model a binary dependent variable. In this project, we are dealing with binary logistic model which has a dependent variable with two possible values, such as pass/fail which is represented by an indicator variable, where the two values are labeled '-1' and '1'.

In other words, given dataset of pairs (x, y) with $x \in R^d$ and $y \in \{-1, 1\}$, logistic regressor returns the probabilities $P(y = 1|x)$. It uses a linear function $f_w(x) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b$ where $w = (w_1, w_2, \ldots, w_d)$ represents weights vector. Mathematically, $P(y|x) = 1/(1 + e^{-y*f_w(x)}) = 1/(1 + e^{-y*(w.x+b)})$ where w and b are learned from training data. The aim here is to pick w and b that maximize the probabilities $P_{w,b}(Y|X) = \prod_{i=1}^{n} P_{w,b}(y^{(i)}|x^{(i)})$. The problem of maximizing the probability $P_{w,b}(Y|X)$ is equivalent to minimizing the loss function $L(w, b) = -\sum_{i=1}^{n} ln(P_{w,b}(y^{(i)}|x^{(i)})) = \sum_{i=1}^{n} ln(1 + e^{-y^{(i)}(w.x^{(i)}+b)})$ which is also known as log loss.

The goal of this project is to propose a coordinate descent based method to minimize the loss function. Coordinate descent is an optimization algorithm that successively minimizes along coordinate directions to find the minimum of a function. At each iteration, the algorithm determines a coordinate and then minimizes over the corresponding coordinate hyperplane while fixing all other coordinates. It is applicable to both differentiable and non-differentiable functions and is therefore widely used.

For this project, to compare the coordinate descent with the logistic regression, I implemented the log loss for logistic regression and optimized it using stochastic gradient descent method (SGD).

[2] Dataset

The data set is the result of a chemical analysis of wines grown in the some region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. Out of these three classes, only the first two (labelled 0 and 1) are being used for this project. The initial data set had around 30 variables, but only 13 are being used for the analysis.

[3] Description

The proposed coordinate descent algorithm is inspired by stochastic gradient descent. It is based on the ideas of steepest descent and cyclic coordinate descent. In cyclic coordinate descent, one cyclically iterates through the directions, one at a time, minimizing the objective function with respect to each coordinate direction at a time. In steepest descent, the algorithm continues its search in the direction which will minimize/maximize the value of a function.

The high level idea is at each epoch, we go through all the data points and for each data point, we pick the coordinate giving the steepest descent in the gradient. Using the direction of steepest descent, we update weights vector. After one round through the entire dataset, we get the final weight vector for that particular epoch and calculate the log loss. This process is done 5,000 number of times (epochs).

Preprocessing

The data is pre-processed using the following steps:

- Label Conversion: The label '0' in the original data set is converted to '-1' to be able to use the classroom formulas
- Normalization: The data is normalized using sklearn's Standard Scalar normalization which standardizes features by removing the mean and scaling to unit variance. The standard score of a sample x is calculated as:

  $$z = (x^{(i)} - u)/s$$

  where u is the mean of the training samples and s is the standard deviation of the training samples.

Logistic Regressing Using Stochastic Gradient Descent (SGD)

To compare the performance of the coordinate descent algorithm with the logistic regression solver, I implemented my own loss function for logistic regressor using stochastic gradient descent. SGD is an iterative approach for optimizing an objective function with suitable smoothness properties. It replaces the actual gradient (calculated using the entire data set) by an estimate (calculated from a randomly selected subset of the data).

SGD Pseudo Code

Require: X (Training data), y (Training labels), N (Number of training samples)

// Initializing Data Structures
$w_t = [0, 0, \ldots, 0]$ // Initial weights vector (size = 13)
$eta = 0.1$ // Learning rate
$T = 5000$ // Total number of iterations
$log\_loss = []$ // List storing log loss at each epoch
for $c\ in\ \{1, 2, ...T\}$ **do**
  for $i\ in\ \{0, 1, \ldots, (N-1)\}$ **do**
    $p = (eta * y[i])/(1 + e^{y[i]*dot(w_t, X[i])})$ // dot function performs dot product of two vectors
    $v = [p * j\ for\ j\ in\ X[i]]$
    $w_t = add(w_t, v)$ // add function adds two vectors
  end for
  $loss = 0$
  for $i\ in\ \{0, 1, \ldots, (N-1)\}$ **do**
    $loss+ = math.log(1 + math.exp(-1 * y[i] * np.dot(w_t, X[i])))$
  end for
  $log\_loss.append(loss)$
end for
return log_loss

Coordinate Descent Algorithm

The steps performed in each epoch of proposed coordinate descent algorithm are:

- For each data point in the dataset
  - Calculate the gradient vector of log loss at this given point. The gradient vector will have dimension $13 * 1$
  - In the calculated gradient vector find the coordinate index having the maximum value of gradient (in absolute terms)
  - Update the weights vector for the index having maximum gradient value (let's say max_index). The new weight value for index max_index is calculated as as the original value of weight vector for plus eta times the gradient of weights at max_index
- Once we update our weights vector by going through all the data points, we calculate the log loss

Coordinate Descent Pseudo Code

Require: X (Training data), y (Training labels), N (Number of training samples)

// Initializing Data Structures
$w_t = [0, 0, \ldots, 0]$ // Initial weights vector (size = 13)
$eta = 0.1$ // Learning rate
$T = 5000$ // Total number of iterations
$log\_loss = []$ // List storing log loss at each epoch
for $c\ in\ \{1, 2, ...T\}$ **do**
  for $i\ in\ \{0, 1, \ldots, (N-1)\}$ **do**
    $p = (eta * y[i])/(1 + e^{y[i]*dot(w_t, X[i])})$ // dot function performs dot product of two vectors
    $v\_1 = [p * j\ for\ j\ in\ X[i]]$
    $v\_2 = [abs(p * j)\ for\ j\ in\ X[i]]$ // abs function calculates absolute value
    $max\_index = argmax(v\_2)$ // argmax function returns the index of max value
    $w_t[max\_index] += eta * v\_1[max\_index]$
  end for
  $loss = 0$
  for $i\ in\ \{0, 1, \ldots, (N-1)\}$ **do**
    $loss += math.log(1 + math.exp(-1 * y[i] * np.dot(w_t, X[i])))$
  end for
  $log\_loss.append(loss)$
end for
return log_loss

Random Descent Algorithm

The steps followed for random descent are similar to the coordinate descent except instead of using the gradient to decide which coordinate to pick, we select the coordinate randomly.

Random Descent Pseudo Code

Require: X (Training data), y (Training labels), N (Number of training samples)

// Initializing Data Structures
$w_t = [0, 0, \ldots, 0]$ // Initial weights vector (size = 13)

$eta = 0.1$ // Learning rate
$T = 5000$ // Total number of iterations
$log\_loss = []$ // List storing log loss at each epoch
for $c\ in\ \{1, 2, ...T\}$ **do**
  for $i\ in\ \{0, 1, \ldots, (N-1)\}$ **do**
    $p = (eta * y[i])/(1 + e^{y[i]*dot(w_t, X[i])})$ // dot function performs dot product of two vectors
    $v = [p * j\ for\ j\ in\ X[i]]$
    $random\_index = random\_int(0, 12)$
    $w_t[max\_index] += eta * v[random\_index]$
  end for
  $loss = 0$
  for $i\ in\ \{0, 1, \ldots, (N-1)\}$ **do**
    $loss += math.log(1 + math.exp(-1 * y[i] * np.dot(w_t, X[i])))$
  end for
  $log\_loss.append(loss)$
end for
return log_loss

---

[5] Convergence

Coordinate descent belongs to the class of several nonderivative methods used for minimizing differentiable functions. The cost is minimized in one coordinate direction in each iteration. However, the proposed coordinate descent is based on SGD, where we pick a coordinate with maximum gradient and update the weights along that direction. To be able to calculate the gradient, the loss function needs to be continuously differentiable at all the points.

SGD is an approximation of gradient descent where we randomly pick one data point from the whole data set and calculate the gradient at that point at each iteration to reduce the computations. Because we are not calculating the gradient along the entire dataset, instead just using a single point, SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily. The proposed coordinate descent is a further approximation of SGD where we update the weights along one particular coordinate greedly, hence, it is more prone to fluctuations. It also takes more number of epochs to converge because each step's contribution towards the final convergence is less.

In order to find the steepest direction, we can approximate the function via a first-order Taylor expansion: $f(x + d) \approx f(x) + \nabla f(x)$
In order to minimize the loss function we need to find the direction that maximizes $\nabla f(x)$.

Therefore, the proposed coordinate descent method will converge to an optimal value given the loss function is differentiable and has a global minima (i.e. convex) and we pick the step size (η) and the number of epochs wisely.

[6] Experimental Results

To perform the experiments, the number of epochs were fixed for all the three cases.

As shown in Fig. 1 and Fig. 2, the proposed coordinate descent beats the random descent algorithm in terms of rate of convergence. However, stochastic gradient descent outperforms both the random descent and coordinate descent in terms of rate of convergence. As shown in Fig. 2, all the three algorithm converge to same value.

Another point to note is that the convergence varies with the step size for the same number of epochs which is coherent with the results taught in class. The methods converge much faster for $\eta = 0.1$ as shown in Fig. 2 than for $\eta = 0.01$ as shown in Fig. 1.

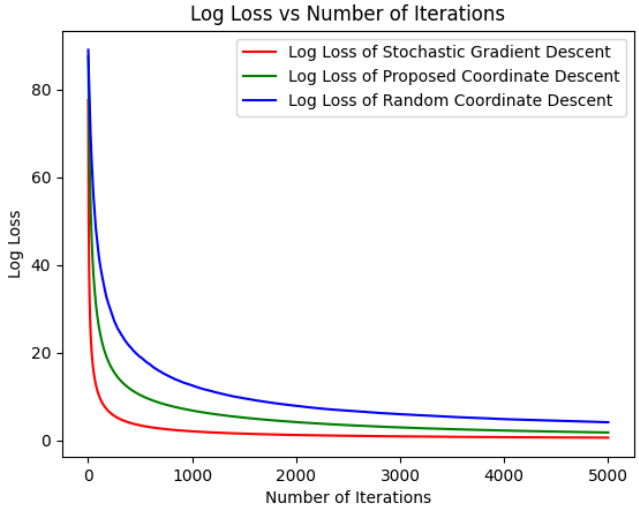

Fig. 1: Log Loss vs No. of Iterations ($\eta = 0.001$)

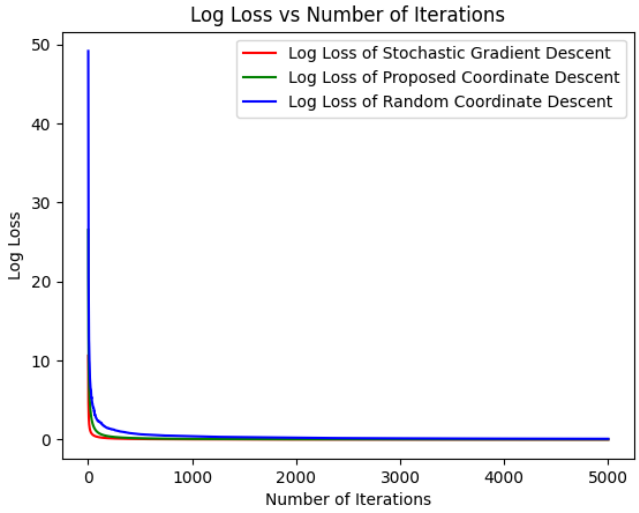

Fig. 2: Log Loss vs No. of Iterations ($\eta = 0.1$)

[7] Conclusions

Gradient descent is computationally intense. To solve this problem, we use stochastic gradient descent. SGD randomly picks one data point from the whole data set at each iteration to reduce the computations enormously. To simplify SGD further, we use coordinate descent.

The proposed coordinate descent is based on cyclic and steepest coordinate descent. In it's current state, it is very slow and takes more epochs to converge to the optimal loss function. It is also sensitive to the choice of eta. The given dataset was relatively very small and had less features/coordinates to

pick from, therefore, the proposed method converged in 5000 epochs. Therefore, there is definitely a room for improvement in the proposed algorithm.

To improve it, we can pick K coordinates and simultaneously update the gradients along the chosen K coordinates to make the updates faster. The current algorithm can also be modified to utilize independence and parallelism. We can also modify the algorithm to work with non differentiable loss function. Therefore, instead on relying on the gradient of the loss function, we can define the change as difference of slope between two data points along each coordinate and pick the coordinate giving the minimum change along the two data points.

References

1. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
2. https://en.wikipedia.org/wiki/Logistic_regression#:~:text=Logistic%20regression%20is%20a%20statistical,a%20form%20of%20binary%20regression)
3. https://en.wikipedia.org/wiki/Stochastic_gradient_descent#:~:text=Stochastic%20gradient%20descent%20(often%20abbreviated,(e.g.%20differentiable%20or%20subdifferentiable)
4. https://archive.ics.uci.edu/ml/datasets/Wine
5. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html
6. http://users.ece.utexas.edu/~cmcaram/EE381V_2012F/Lecture_5_Scribe_Notes.final.pdf
7. https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53
8. https://en.wikipedia.org/wiki/Coordinate_descent
9. https://people.seas.harvard.edu/~yaron/AM221-S16/lecture_notes/AM221_lecture10.pdf

project2_code

February 22, 2022

[15]:
```
from sklearn.datasets import load_wine
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import log_loss
import math
import numpy as np
from sklearn import preprocessing
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import random
```

[16]:
```
data = load_wine()
X = []
y = []
```

[17]:
```
for i in range(130):
    X.append(data['data'][i])
    if data['target'][i] == 0:
        y.append(-1)
    if data['target'][i] == 1:
        y.append(1)
```

[18]:
```
scaler = preprocessing.StandardScaler()
X = scaler.fit_transform(X)
```

[19]:
```
w_t = [0]*13
eta = 0.01
count = 0
x_axis = []
y_axis_sgd = []
while count < 5000:
    count += 1
    for i in range(130):
        c = (eta*y[i])/(1+math.exp(y[i]*np.dot(w_t, X[i])))
        v = [c*j for j in X[i]]
        w_t = np.add(w_t, v)

    loss = 0
```

```
    for i in range(130):
        loss += math.log(1+math.exp(-1*y[i]*np.dot(w_t, X[i])))
    y_axis_sgd.append(loss)
    x_axis.append(count)

print(count)
print(w_t)
```

```
5000
[-4.80156685 -1.71843865 -3.64010213  4.5613607  -0.67709047  0.19250325
 -1.16351531  0.60412466  0.57423706 -1.96776244  0.44040263 -2.11995367
 -6.20684003]
```

[20]:
```
print(y_axis_sgd[len(y_axis_sgd)-1])
```

```
0.07695156503157798
```

[21]:
```
y_pred = []
for i in range(130):
    v =np.dot(w_t, X[i])
    if v>0:
        y_pred.append(1)
    else:
        y_pred.append(-1)
```

[22]:
```
accuracy_score(y, y_pred)
```

[22]: 1.0

[23]:
```
w_t = [0]*13
eta = 0.001
count = 0
x_axis = []
y_axis_cd = []
while count < 5000:
    count += 1
    for i in range(130):
        c = (y[i])/(1+math.exp(y[i]*np.dot(w_t, X[i])))
        v = [abs(c*j) for j in X[i]]
        v_new = [c*j for j in X[i]]
        max_index = np.argmax(v)
        w_t[max_index] += eta*v_new[max_index]

    loss = 0
    for i in range(130):
        loss += math.log(1+math.exp(-1*y[i]*np.dot(w_t, X[i])))
    y_axis_cd.append(loss)
```

```
    x_axis.append(count)

print(count)
print(w_t)
```

```
5000
[-3.196220639711379, -1.2013100217919972, -4.3538916274093475,
5.495416105889829, 0.12302392933730259, 0.036840355165491084,
0.4249055940107983, 1.2757959751613541, -0.6859161034696151,
-0.8364580012065207, -0.9476322953708065, -2.3218979204350623,
-2.3631148867993073]
```

[24]:
```
print(y_axis_cd[len(y_axis_cd)-1])
```

```
1.7523728934657863
```

[25]:
```
w_t = [0]*13
eta = 0.001
count = 0
x_axis = []
y_axis_rd = []
while count < 5000:
    count += 1
    for i in range(130):
        c = (y[i])/(1+math.exp(y[i]*np.dot(w_t, X[i])))
        v = [abs(c*j) for j in X[i]]
        v_new = [c*j for j in X[i]]
        max_index = random.randint(0, 12)
        w_t[max_index] += eta*v_new[max_index]

    loss = 0
    for i in range(130):
        loss += math.log(1+math.exp(-1*y[i]*np.dot(w_t, X[i])))
    y_axis_rd.append(loss)
    x_axis.append(count)

print(count)
print(w_t)
```

```
5000
[-1.6677695198116818, -0.5240027793219737, -0.9490688588454765,
1.29836004296092, -0.25812628112798014, -0.07686962553713074,
-0.3543709300539891, 0.1458224980827075, 0.18644950787121145,
-1.0139654098537167, 0.15149546331315983, -0.6214625781299163,
-1.8744128930989876]
```

[26]:
```
print(y_axis_rd[len(y_axis_rd)-1])
```

```
4.041159637588551
```

[27]:
```
plt.plot(x_axis, y_axis_sgd, color='r', label='Log Loss of Stochastic Gradient␣
 ↪Descent')
plt.plot(x_axis, y_axis_cd, color='g', label='Log Loss of Proposed Coordinate␣
 ↪Descent')
plt.plot(x_axis, y_axis_rd, color='b', label='Log Loss of Random Coordinate␣
 ↪Descent')
plt.xlabel('Number of Iterations')
plt.ylabel('Log Loss')
plt.title('Log Loss vs Number of Iterations')
plt.legend()
plt.show()
```