Introduction to Machine Learning (CS771A)
Assignment 1 Report

GROUP 8

Name:	*Roll Number:*
Akashdeep Singh	160072
Prakhar Saxena	160494
Sandarsh Gupta	160620
Vaibhav Mittal	160767
Vasu Bansal	160776

1 Problem 1: Methods Implemented

We have implemented Primal Subgradient Descent and Primal Stochastic Mini-batch Subgradient Descent Method for the Lasso formulation.

$$\arg\min_{w \in \mathbb{R}^d} \|\mathbf{w}\|_1 + \|X\mathbf{w} - \mathbf{y}\|_2^2 \qquad (P1)$$

where $X = [\mathbf{x}^1, \cdots, \mathbf{x}^n]^T \in \mathbb{R}^{n \times d}$ and $y = [y^1, \cdots, y^n]^T \in \mathbb{R}^n$

Gradient Calculation

Gradient for L-1 regularizer: For all $\mathbf{w} = (w_1, \cdots, w_d) \in \mathbb{R}^d$

$$\|\mathbf{w}\| = |w_1| + \cdots + |w_n|$$

so that

$$\frac{\partial \|\mathbf{w}\|}{\partial w_j} = \frac{w_j}{|w_j|}, \qquad w_j \neq 0$$

i.e., for $w_j < 0$; $\frac{\partial \|\mathbf{w}\|}{\partial w_j} = -1$ and for $w_j > 0$; $\frac{\partial \|\mathbf{w}\|}{\partial w_j} = 1$. But when any $w_j = 0$; $\|\mathbf{w}\|$ is not differentiable. Therefore, by subgradient calculus, at $w_j = 0$; $\frac{\partial \|\mathbf{w}\|}{\partial w_j} = \lambda(-1) + (1-\lambda)(1) = 1 - 2\lambda$ where $\lambda \in [0, 1]$

So, at $w_j = 0$; $\frac{\partial \|\mathbf{w}\|}{\partial w_j} \in [-1, 1]$. For simplicity, we have taken this to be zero. Hence,

$$\frac{d\|\mathbf{w}\|}{d\mathbf{w}} = \text{sign}(\mathbf{w})$$

Gradient for Least-sqaured Loss:

$$\begin{aligned} \|X\mathbf{w} - \mathbf{y}\|_2^2 =& (X\mathbf{w} - y)^T (X\mathbf{w} - y) \\ =& \mathbf{w}^T X^T X \mathbf{w} - \mathbf{w}^T X^T y - y^T X \mathbf{w} + y^T y \end{aligned}$$

So,

$$\frac{d\|X\mathbf{w}-\mathbf{y}\|_2^2}{d\mathbf{w}} = 2X^TX\mathbf{w} - X^Ty - (y^TX)^T = 2(X^TX\mathbf{w} - X^Ty)$$

Overall Gradient: From above calculations, the gradient for LASSO formulation P1

$$\nabla(P1) = \text{sign}(\mathbf{w}) + 2(X^TX\mathbf{w} - X^Ty)$$

1.1 Primal Subgradient Descent

Gradient Expression :

$$\nabla(P1) = \text{sign}(\mathbf{w}) + 2(X^TX\mathbf{w} - X^Ty)$$

We used velocity acceleration method in Subgradient descent. At the end of loop, we stored the gradient in a variable $prev\_grad$. The step length (η) chosen was 1. Initial model vector w was initialized to 0. Initial value of $prev\_grad$ was set to 0. Step length (α) for $prev\_grad$ is 0.003. Final step length at each step = $\frac{\eta}{t^2}$ and step length for prev_grad at each step = $\frac{\alpha}{t^{0.3}}$

$$\begin{aligned} \mathbf{w}^{t+1} =& \mathbf{w}^t - \frac{\eta}{t^2}\nabla(P1)^t - \frac{\alpha(prev\_grad)}{t^{0.3}} \\ prev\_grad =& \nabla(P1)^t \end{aligned}$$

1.2 Primal Stochastic Mini-batch Subgradient Descent

The gradient expression is same as for previous part.

$$\nabla(P1) = \text{sign}(\mathbf{w}) + 2(X^TX\mathbf{w} - X^Ty)$$

First we used `array = np.random.randint(0, 799, size = (mini_batchSize, ))` command to get a random list of numbers between 0 and 799 for selecting random rows of X. Then we created new variables `X_ = X[array, :]` and `y_ = y[array]` to select these corresponding rows from X and y. Finally X_ and y_ replaced X and y in the gradient expression and rest was the same. Since Mini-batch results in a very spiky plot of objective function values, we used Velocity acclerated method here also to get a pretty smooth plot. Following the same convention of variables as done for the previous part.

The step length (η) chosen was 0.4. Initial model vector w was initialized to 0. Initial value of prev_grad was set to 0. Step length (α) for prev_grad is 0.0075. Final step length at each step = $\frac{\eta}{t^1}$ and step length for prev_grad at each step = $\frac{\alpha}{t^{1.2}}$ The Mini-batch size used was 100.

2 Hyper-parameters Tuning

We tuned the parameters by optimising for one at a time. First we tuned for step lengths by taking values like 1, 0.1, 0.001, 5. Then after getting a particular value (we got 1 for both cases initially), we varied by adding and subtracting 0.5. Then we moved in the direction which gave us better results and refined the procedure by adding and subtracting 0.1 at a time. We used velocity acceleration method in Subgradient descent. At the end of loop, we stored the gradient in a variable $prev\_grad$.

In a similar fashion, the powers of iteration parameter t in denominator of step length were tuned. For the mini batch size we started out with a batch size of 32 then tried out 64, 128 etc. finally we settled for 100.

3 Convergence Plots

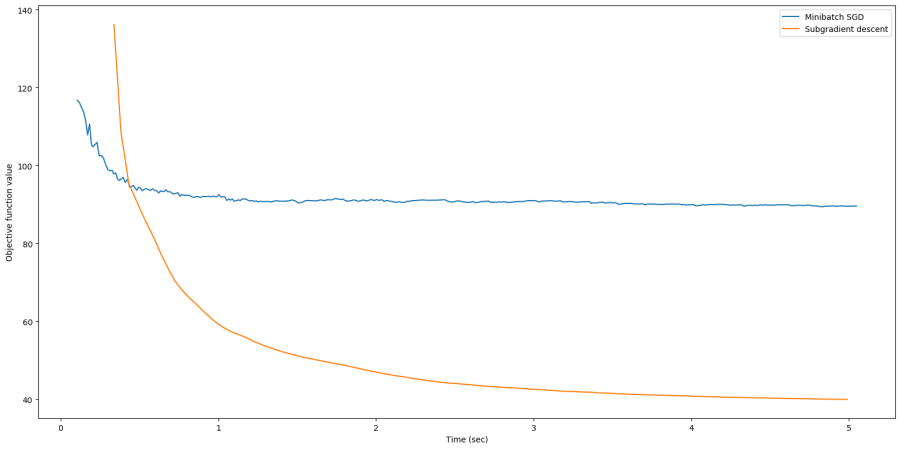

Figure 1: Convergence Plot

Subgradient descent with Velocity acceleration method worked best for us, as can be seen from the plots. It gave us a minimum value of 38.9 of the objective value, compared to 89 obtained by MBSGD. We have removed some initial iterations (6 to 7) for better visualization.

4 Python File Submission

We have uploaded the zipped `submit.py` with the name 'submit' on the following link: `https://github.com/vasubansal1033/CS771_Intro_to_ML/raw/master/submit.zip`