CSCE 5063-001: Assignment 2

Due 11:59pm Friday, March 1, 2019

1 Implementation of SVM via Gradient Descent

You will use the SVM to predict if a person doesn't have diabetes (negative class) or has diabetes (positive class), where the data is from National Institute of Diabetes and Digestive and Kidney Diseases. A total of 8 features will be used, including `Preg` (number of times pregnant), `Pres` (Diastolic blood pressure), `Mass` (body mass index), etc. There are a total number of 760 patients in the data.

The dataset is given in file `data.txt`, which contains the data description, and a matrix with 9 columns and 760 rows. Columns 1-8 are input features, and column 8 is the class label. In this assignment, you don't need to do the normalization and training/testing splitting.

You will implement the soft margin SVM using different gradient descent methods as we learned in class. To recap, to estimate the $\mathbf{w}, b$ of the soft margin SVM, we can minimize the cost function:

$$J(\mathbf{w}, b) = \frac{1}{2}\sum_{j=1}^{n}(w_j)^2 + C\sum_{i=1}^{m}\max\left\{0\ ,\ 1 - y^{(i)}\left(\sum_{j=1}^{n} w_j x_j^{(i)} + b\right)\right\}. \tag{1}$$

In order to minimize the function, we first obtain the gradient with respect to w_j, the jth item in the vector $\mathbf{w}$, as follows:

$$\nabla_{w_j} J(\mathbf{w}, b) = \frac{\partial J(\mathbf{w}, b)}{\partial w_j} = w_j + C\sum_{i=1}^{m}\frac{\partial L(x^{(i)}, y^{(i)})}{\partial w_j}, \tag{2}$$

where:

$$\frac{\partial L(x^{(i)}, y^{(i)})}{\partial w_j} = \begin{cases} 0 & \text{if } y^{(i)}(\mathbf{x}^{(i)}\mathbf{w} + b) \geq 1 \\ -y^{(i)}x_j^{(i)} & \text{otherwise.} \end{cases}$$

We should also compute the gradient with respect to b, which needs to be derived by yourself. Now, we will implement and compare the following gradient descent techniques.

For all methods, use $C = 10$.

1. Batch Gradient Descent

Algorithm 1: Batch Gradient Descent: Iterate through the entire dataset and update the parameters as follows:

$k = 0$;
while *convergence criteria not reached* **do**
  for $j = 1, \dots, n$ **do**
    Update $w_j \leftarrow w_j - \eta \nabla_{w_j} J(\mathbf{w}, b)$;
  Update $b \leftarrow b - \eta \nabla_b J(\mathbf{w}, b)$;
  Update $k \leftarrow k + 1$;

where,
k is the number of iterations,
n is the dimensions of $\mathbf{w}$,
η is the learning rate of the gradient descent, and
$\nabla_{w_j} J(\mathbf{w}, b)$ is the value computed from computing Eq. (2) above and $\nabla_b J(\mathbf{w}, b)$ is the value computed from your answer in question (a) below.

The *convergence criteria* for the above algorithm is $\Delta_{\%cost} < \epsilon$, where

$$\Delta_{\%cost} = \frac{|J_{k-1}(\mathbf{w}, b) - J_k(\mathbf{w}, b)| \times 100}{J_{k-1}(\mathbf{w}, b)}, \tag{3}$$

where,
$J_k(\mathbf{w}, b)$ is the value of Eq. (1) at kth iteration, and
$\Delta_{\%cost}$ is computed at the end of each iteration of the while loop.

Initialize $\mathbf{w} = \mathbf{0}, b = 0$ and compute $J_0(\mathbf{w}, b)$ with these values.

For this method, it is recommended to use $\eta = 0.000000001$ **and** $\epsilon = 0.04$**,** or you can adjust these hyperparameters by yourself until you obtain reasonable results.

2. Stochastic Gradient Descent

Algorithm 2: Stochastic Gradient Descent: Go through the dataset and update the parameters, one training example at a time, as follows:

Randomly shuffle the training data;
$i = 0, k = 0$;
while *convergence criteria not reached* **do**
  for $j = 1, \dots, n$ **do**
    Update $w_j \leftarrow w_j - \eta \nabla_{w_j} J^{(i)}(\mathbf{w}, b)$;
  Update $b \leftarrow b - \eta \nabla_b J^{(i)}(\mathbf{w}, b)$;
  Update $i \leftarrow (i + 1) \mod m$;
  Update $k \leftarrow k + 1$;

where,

k is the number of iterations,
m is the number of examples in the training dataset,
n is the dimensions of $\mathbf{w}$,
i is the training example currently used for computing gradient,
η is the learning rate of the gradient descent, and
$\nabla_{w_j} J^{(i)}(\mathbf{w}, b)$ is defined for a single training example as follows:

$$\nabla_{w_j} J^{(i)}(\mathbf{w}, b) = \frac{\partial J^{(i)}(\mathbf{w}, b)}{\partial w_j} = w_j + C \frac{\partial L(x^{(i)}, y^{(i)})}{\partial w_j}.$$

Note that you will also have to derive $\nabla_b J^{(i)}(\mathbf{w}, b)$, but it should be similar to your solution to question (a) below.

The *convergence criteria* here is $\Delta_{cost}^{(k)} < \epsilon$, where

$$\Delta_{cost}^{(k)} = 0.5 \cdot \Delta_{cost}^{(k-1)} + 0.5 \cdot \Delta_{\%cost},$$

where,
k is the iteration number, and
$\Delta_{\%cost}$ is the same as above (Eq. (3)).

Calculate $\Delta_{cost}^{(k)}, \Delta_{\%cost}$ at the end of each iteration of the while loop.
Initialize $\Delta_{cost}^{(0)} = 0, \mathbf{w} = \mathbf{0}, b = 0$ and compute $J_0(\mathbf{w}, b)$ with these values.

For this method, it is recommended to use $\eta = 0.00000001, \epsilon = 0.0003$, or you can adjust these hyperparameters by yourself until you obtain reasonable results.

3. Mini Batch Gradient Descent where,

Algorithm 3: Mini Batch Gradient Descent: Go through the dataset in batches of predetermined size and update the parameters as follows:

Randomly shuffle the training data;
$l = 0, k = 0$;
while *convergence criteria not reached* **do**
  for $j = 1, \ldots, n$ **do**
    Update $w_j \leftarrow w_j - \eta \nabla_{w_j} J_l(\mathbf{w}, b)$;
  Update $b \leftarrow b - \eta \nabla_b J_l(\mathbf{w}, b)$;
  Update $l \leftarrow (l + 1) \mod ((m + batch\_size - 1)/batch\_size)$;
  Update $k \leftarrow k + 1$;

k is the number of iterations,
m is the number of examples in the training dataset,
n is the dimensions of $\mathbf{w}$,
batch_size is the number of training examples in each batch,
l is the batch currently used for computing gradient,

η is the learning rate of the gradient descent, and
$\nabla_{w_j} J_l(\mathbf{w}, b)$ is defined for a batch of training example as follows:

$$\nabla_{w_j} J_l(\mathbf{w}, b) = \frac{\partial J_l(\mathbf{w}, b)}{\partial w_j} = w_j + C \sum_{i=l \times batch\_size+1}^{min(m,(l+1)\times batch\_size)} \frac{\partial L(x^{(i)}, y^{(i)})}{\partial w_j}.$$

The *convergence criteria* here is $\Delta_{cost}^{(k)} < \epsilon$, where

$$\Delta_{cost}^{(k)} = 0.5 \cdot \Delta_{cost}^{(k-1)} + 0.5 \cdot \Delta_{\%cost},$$

where,
k is the iteration number, and
$\Delta_{\%cost}$ is the same as above (Eq. (3)).

Calculate $\Delta_{cost}^{(k)}, \Delta_{\%cost}$ at the end of each iteration of the while loop.
Initialize $\Delta_{cost}^{(0)} = 0, \mathbf{w} = \mathbf{0}, b = 0$ and compute $J_0(\mathbf{w}, b)$ with these values.

For this method, it is recommended to use $\eta = 0.00000001, \epsilon = 0.004$**,** ***batch_size*** $= 4$**,** or you can adjust these hyperparameters by yourself until you obtain reasonable results.

Question (a)

Notice that we have not given you the equation for $\nabla_b J(\mathbf{w}, b)$.
Task: What is $\nabla_b J(\mathbf{w}, b)$ used for the Batch Gradient Descent Algorithm?
(Hint: It should be very similar to $\nabla_{w_j} J(\mathbf{w}, b)$*.)*

Question (b)

Task: Implement the SVM algorithm for the above mentioned gradient descent techniques.

Use $C = 10$ for all the techniques. **Note:** update w in iteration $i + 1$ using the values computed in iteration i. Do not update using values computed in the current iteration!

Run your implementation on the given dataset.

Task: Plot the value of the cost function $J_k(\mathbf{w}, b)$ vs. the number of iterations (k). Report the total time taken for convergence by each of the gradient descent techniques.

The diagram should have graphs from all the three techniques on the same plot.

For debugging, Batch GD should converge within 100 iterations, SGD within 5,000 iterations, and Mini Batch GD somewhere in-between.

What to submit

1. Equation for $\nabla_b J(\mathbf{w}, b)$.

2. Plot of $J_k(\mathbf{w}, b)$ vs. the number of iterations (k).

3. The convergence times.

4. The source code.