CSCI 5702/7702– Fall 2019

Assignment 4

Large-Scale Machine Learning and Data Streams

Due: 12/04/2019 – 8:59 pm

- Please note that this assignment must be done individually. Your code will be checked against other submissions and other existing resources (such as websites and books) using automatic tools.
- Review the lecture notes before starting with this assignment. Then, fully read this document before starting with the implementation or thinking on the solution.
- If you have technical issues with Spark, please Google the error messages and share the error message alongside the solution that got it fixed on Microsoft Teams, as your classmates may run into the same issues.
- Check Canvas regularly for possible clarifications and updates.
- Part 1 of the assignment is mandatory. Part 2 of the assignment is optional and provides an extra credit opportunity.

Part 1. Implementation of SVM via Gradient Descent (100 points)

Here, you will implement the soft margin SVM using different gradient descent methods as described in the section 12.3.4 of the textbook. To recap, to estimate the $\mathbf{w}$,b of the soft margin SVM, we can minimize the cost:

$$f(\mathbf{w},b) = \frac{1}{2}\sum_{j=1}^{d}(w^{(j)})^2 + C\sum_{i=1}^{n}\max\left\{0, 1 - y_i\left(\sum_{j=1}^{d} w^{(j)}x_i^{(j)} + b\right)\right\}. \quad (1)$$

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.

$$\nabla_{w^{(j)}} f(\mathbf{w},b) = \frac{\partial f(\mathbf{w},b)}{\partial w^{(j)}} = w^{(j)} + C\sum_{i=1}^{n}\frac{\partial L(x_i,y_i)}{\partial w^{(j)}}, \quad (2)$$

where:

$$\frac{\partial L(x_i,y_i)}{\partial w^{(j)}} \stackrel{\text{if}}{=} \begin{cases} 0 & y_i\,(\mathbf{x_i}\cdot\mathbf{w} + b) \geq 1 \\ -y_i x_i^{(j)} & \text{otherwise.} \end{cases}$$

Now, we will implement and compare the following gradient descent techniques:

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,...,d **do**

   Update $w^{(j)} \leftarrow w^{(j)} - \eta \nabla_{w^{(j)}} f(\mathbf{w},b)$ **end for**
   Update $b \leftarrow b - \eta \nabla_b f(\mathbf{w},b)$

   Update $k \leftarrow k + 1$ **end**
   while

where, n is the number of samples in the training data, d is the dimensions of $\mathbf{w}$, η is the learning rate of the gradient descent, and $\nabla_{w^{(j)}} f(\mathbf{w},b)$ is the value computed from computing equation (2) above and $\nabla_b f(\mathbf{w},b)$ is the value computed from your answer in question (a) below.

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

$$\Delta_{\%cost} = \frac{|f_{k-1}(\mathbf{w},b) - f_k(\mathbf{w},b)| \times 100}{f_{k-1}(\mathbf{w},b)}. \quad (3)$$

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

Initialize $\mathbf{w} = \mathbf{0}, b = 0$ and compute $f_0(\mathbf{w},b)$ with these values. **For this method, use** $\eta = 0.0000003, \epsilon = 0.25$

2. **Stochastic gradient descent**: Go through the dataset and update the parameters, one training sample at a time, as follows:

   Randomly shuffle the training data

   i = 1,k = 0

   while convergence criteria not reached **do for**
   j = 1,...,d **do**

   Update $w^{(j)} \leftarrow w^{(j)} - \eta \nabla_{w^{(j)}} f_i(\mathbf{w},b)$ **end for**
   Update $b \leftarrow b - \eta \nabla_b f_i(\mathbf{w},b)$ Update i
   $\leftarrow (i \bmod n) + 1$

   Update $k \leftarrow k + 1$

   end while

where, n is the number of samples in the training data, d is the dimensions of $\mathbf{w}$, η is the learning rate and $\nabla_{w^{(j)}} f_i(\mathbf{w},b)$ is defined for a single training sample as follows:

$$\nabla_{w^{(j)}} f_i(\mathbf{w}, b) = \frac{\partial f_i(\mathbf{w}, b)}{\partial w^{(j)}} = w^{(j)} + C\frac{\partial L(x_i, y_i)}{\partial w^{(j)}}$$

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

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

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

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

Calculate Δ_{cost}, $\Delta_{\%cost}$ at the end of each iteration of the while loop.

Initialize $\Delta_{cost} = 0$, $\mathbf{w} = \mathbf{0}, b = 0$ and compute $f_0(\mathbf{w},b)$ with these values.

For this method, use $\eta = 0.0001, \epsilon = 0.001$.

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**
   $j = 1,\ldots,d$ **do**

   Update $w_{(j)} \leftarrow w_{(j)} - \eta\nabla_{w_{(j)}} f_l(\mathbf{w},b)$ **end for**
   Update $b \leftarrow b - \eta\nabla_b f_l(\mathbf{w},b)$

   Update $l \leftarrow (l + 1)$ mod $((n + batch\ size - 1)/batch\ size)$
   Update $k \leftarrow k + 1$ **end**
   while

where, n is the number of samples in the training data, d is the dimensions of $\mathbf{w}$, η is the learning rate, *batch size* is the number of training samples considered in each batch, and $\nabla_{w_{(j)}} f_l(\mathbf{w},b)$ is defined for a batch of training samples as follows:

$$\nabla_{w^{(j)}} f_l(\mathbf{w}, b) = \frac{\partial f_l(\mathbf{w}, b)}{\partial w^{(j)}} = w^{(j)} + C \sum_{i=l*batch\_size+1}^{min(n,(l+1)*batch\_size)} \frac{\partial L(x_i, y_i)}{\partial w^{(j)}},$$

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

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

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

Calculate Δ_{cost}, $\Delta_{\%cost}$ at the end of each iteration of the while loop.

Initialize $\Delta_{cost} = 0$, $\mathbf{w} = \mathbf{0}, b = 0$ and compute $f_0(\mathbf{w}, b)$ with these values.

For this method, use $\eta = 0.00001, \epsilon = 0.01$, batch size = 20.

(a) [10 Points]

Notice that we have not given you the equation for $\nabla_b f(\mathbf{w}, b)$.

Task: What is $\nabla_b f(\mathbf{w}, b)$ used for the Batch Gradient Descent Algorithm?

(Hint: It should be very similar to $\nabla_{w(j)} f(\mathbf{w}, b)$.)

(b) [90 Points]

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

Use C = 100 for all the techniques. For all other parameters, use the values specified in the description of the technique. **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 dataset provided for Part 1. The dataset contains the following files :

1) features.txt : Each line contains features (comma-separated values) for a single datapoint. It has 6414 datapoints (rows) and 122 features (columns).
2) target.txt : Each line contains the target variable (y = -1 or 1) for the corresponding row in features.txt.

Task: Plot the value of the cost function $f_k(\mathbf{w}, b)$ vs. the number of iterations (k). Report the total time taken for convergence by each of the gradient descent techniques. What do you infer from the plots and the time for convergence?

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

As a sanity check, Batch GD should converge in 10-300 iterations and SGD between 500-3000 iterations with Mini Batch GD somewhere in-between. However, the number of iterations may vary greatly due to randomness. If your implementation consistently takes longer though, you may have a bug.

What to Submit for Part 1

(i) Equation for $\nabla_b f(\mathbf{w}, b)$. [part (a)]

(ii) Plot of $f_k(\mathbf{w}, b)$ vs. the number of updates (k). Total time taken for convergence by each of the gradient descent techniques. Interpretation of plot and convergence times. [part (b)]

(iii) Submit the code on Canvas. [part (b)]

Part 2. Data Streams (Optional, Extra Credit, 50 points)

In this problem, we study an approach to approximate the frequency of occurrences of different items in a data stream. Assume $S = < a_1, a_2, \ldots, a_t >$ is a data stream of items from the set $\{1,2,\ldots,n\}$. Assume for any $1 \le i \le n$, $F[i]$ is the number of times i has appeared in S. We would like to have good approximations of the values $F[i]$ $(1 \le i \le n)$ at all times.

A simple way to do this is to just keep the counts for each item $1 \le i \le n$ separately. However, this will require $O(n)$ space, and in many applications (e.g., think online advertising and counts of user's clicks on ads) this can be prohibitively large. We see in this problem that it is possible to approximate these counts using a much smaller amount of space. To do so, we consider the algorithm explained below.

Strategy

The algorithm has two parameters $\delta, \epsilon > 0$. It picks $log\frac{1}{\delta}$ independent hash functions: $\forall j \in \left[\!\left[1; \left\lceil log\frac{1}{\delta} \right\rceil\right]\!\right], h_j: \{1,2,\cdots,n\} \to \{1,2,\cdots,\left\lceil \frac{e}{\epsilon} \right\rceil\}$, where log denotes natural logarithm. Also, it associates a count $c_{j,x}$ to any $1 \le j \le log\frac{1}{\delta}$ and $1 \le x \le \left\lceil \frac{e}{\epsilon} \right\rceil$. In the beginning of the stream, all these counts are initialized to 0. Then, upon arrival of each a_k $(1 \le k \le t)$, each of the counts $C_{j,h_j}(a_k)$ $(1 \le j \le \left\lceil log\frac{1}{\delta} \right\rceil)$ is incremented by 1.

For any $1 \le i \le n$, we define $\tilde{F}[i] = \min_j\{c_{j,h_j(i)}\}$. We will show that $\tilde{F}[i]$ provides a good approximation to $F[i]$. Note that this algorithm only uses $O(\frac{1}{\epsilon}\log\frac{1}{\delta})$ space.

Hint: for any $1 \le i \le n$: $\tilde{F}[i] \ge F[i]$, in the above algorithm.

Datasets

- **words_stream.txt**: each line of this file is a number corresponding to the ID of a word in the stream.
- **counts.txt**: each line is a pair of numbers separated by a tab. The first number is an ID of a word and the second number is its associated exact frequency count in the stream.
- **hash_params.txt**: each line is a pair of numbers separated by a tab, corresponding to parameters a and for the hash functions (See explanation below).

Instructions

You will not receive any grades for questions a and b below, if your code does not work or generates the results that are different than your answers.

Implement the algorithm and run it on the dataset with parameters $\delta = e^{-5}$, $\epsilon = e \times 10^{-4}$. With this choice of δ you will be using 5 hash functions. The 5 pairs (a,b) that you will need for the hash functions are in hash_params.txt.

Then for each distinct word i in the dataset, compute the relative error $E_r[i] = \frac{\tilde{F}[i]-F[i]}{F[i]}$ and plot these values. The plot should use **a logarithm scale both for the x (i) and the y ($E_r[i]$) axes**, and there should be ticks to allow reading the powers of 10 (e.g. 10^{-1}, 10^0, 10^1 etc...). The plot should have a title, as well as the x and y axes. The exact frequencies $F[i]$ should be read from the counts file. Note that words of low frequency can have a very large relative error. That is not a bug in your implementation, but just a consequence of the bound. Plot $F[i]$ and $\tilde{F}[i]$ in the same way that is described above.

Hash Functions

You may use the following hash function (see example pseudo-code), with p = 123457, a and b values provided in the hash params file and buckets (which is equivalent to $\left\lceil \frac{e}{\epsilon} \right\rceil$) chosen according to the specification of the algorithm. In the provided file, each line gives you a, b values to create one hash function.

```
# Returns hash(x) for hash function given by parameters a, b, p and buckets
def hash(a, b, p, buckets, x)
{
        y = x [modulo] p
        hash_val = (a*y+b) [modulo] p
        return hash_val [modulo] buckets
}
```

This hash function implementation produces outputs of value from 0 to (buckets−1), which is different from our specification in the **Strategy** part. You can either keep the range as {0,…,nbuckets−1}, or add 1 to the hash result so the value range becomes {1,…,buckets}, as long as you stay consistent within your implementation.

Implementation Guidelines

- Implement Part 2 in Python. You can implement Part 2 in PySpark as well, but this is not required.
- Do not hardcode the hash parameters as they should be read from a file.
- Do not hardcode the number of words, sample size, counts, etc. as your code may be tested using a different file.
- You need to define the following parameters in the beginning of your code:
  - dataset_path = '' # path to the sampled dataset
  - counts_path = '' # path to the counts file
  - hash_path = '' # path to hash parameters file
  - delta = e^{-5} #δ
  - epsilon = $e \times (10^{-4})$ #ϵ
  - p

- Your driver function must have the following signature:
  - estimate(dataset_path, counts_path, hash_path, delta, epsilon, p)

What to Submit for Part 2

- Your code for part 2 (.py or .ipynb) [35 points]
- A pdf containing:
  - A plot for $F[i]$ (use a logarithmic scale for both the X and Y axes) [5 points]
  - A plot for $\tilde{F}[i]$ (use a logarithmic scale for both the X and Y axes) [5 points]
  - A plot for $E_r[i]$ (use a logarithmic scale for both the X and Y axes) [5 points]

Submission

You need to submit a .zip file on Canvas, named as your-lastname_your-first-name.zip containing two separate folders (one for each part). Do not zip the sub-folders.

DO NOT INCLUDE EXTRA FILES, SUCH AS THE DATASETS, in your submission.

Please download your assignment after submission and make sure it is not corrupted. We will not be responsible for corrupted submissions and will not be able to take a resubmission after the deadline.

Need Help?

In case you need help with Part 1 of this assignment, please email Evan at evan.stene@ucdenver.edu or go to his office hours (Mondays 1:00 pm-2:00 pm and Wednesdays 11:00 am-12:00 pm).

In case you need help with Part 2 of this assignment, please email Shahab at shahab.helmi@ucdenver.edu or go to his office hours (Wednesdays 11:00 am-1:00 pm).

You are highly encouraged to ask your question on the designated channel for Assignment 4 on Microsoft Teams (not monitored by the TA's). Feel free to help other students with general questions. DO NOT share your solution.