Problem Set 4

Please read the homework submission policies here.

Assignment Submission All students should submit their assignments electronically via GradeScope. No handwritten work will be accepted. Math formulas **must** be typeset using LATEX or other word processing software that supports mathematical symbols (E.g. Google Docs, Microsoft Word). Simply sign up on Gradescope and use the course code MP8KGN. Please use your UW NetID if possible.

For the non-coding component of the homework, you should upload a PDF rather than submitting as images. We will use Gradescope for the submission of code as well. Please make sure to tag each part correctly on Gradescope so it is easier for us to grade. There will be a small point deduction for each mistagged page and for each question that includes code. Put all the code for a single question into a single file and upload it. Only files in text format (e.g. .txt, .py, .java) will be accepted. **There will be no credit for coding questions without submitted code on Gradescope, or for submitting it after the deadline**, so please remember to submit your code.

Coding You may use any programming languages and standard libraries, such as NumPy and PySpark, but you may not use specialized packages and, in particular, machine learning libraries (e.g. sklearn, TensorFlow), unless stated otherwise. Ask on the discussion board whether specific libraries are allowed if you are unsure.

Late Day Policy All students will be given two no-questions-asked late periods, but only one late period can be used per homework and cannot be used for project deliverables. A late-period lasts 48 hours from the original deadline (so if an assignment is due on Thursday at 11:59 pm, the late period goes to the Saturday at 11:59pm Pacific Time).

Academic Integrity We take academic integrity extremely seriously. We strongly encourage students to form study groups. Students may discuss and work on homework problems in groups. However, each student must write down the solutions and the code independently. In addition, each student should write down the set of people whom they interacted with.

Discussion Group (People with whom you discussed ideas used in your answers):

On-line or hardcopy documents used as part of your answers:

I acknowledge and accept the Academic Integrity clause.

*(Signed)*_____

1 Implementation of SVM via Gradient Descent (30 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, given a dataset of n samples $\mathcal{D} = \left\{ \left(\boldsymbol{x}^{(i)}, y^{(i)} \right) \right\}_{i=1}^{n}$, where every d-dimensional feature vector $\boldsymbol{x}^{(i)} \in \mathbb{R}^d$ is associated with a label $y^{(i)} \in \{-1, 1\}$, to estimate the parameters $\boldsymbol{\theta} = (\boldsymbol{w}, b)$ of the soft margin SVM, we can minimize the loss function:

$$
\begin{aligned}
f(\boldsymbol{w}, b; \mathcal{D}) &= \frac{1}{2} \|\boldsymbol{w}\|_2^2 + C \sum_{(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathcal{D}} \max \left\{ 0, 1 - y^{(i)}(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)} + b) \right\} \\
&= \frac{1}{2} \|\boldsymbol{w}\|_2^2 + C \sum_{(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathcal{D}} L(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})
\end{aligned}
$$

In order to minimize the function, we first obtain the gradient with respect to $\boldsymbol{\theta}$. The partial derivative with respect to w_j, the j-th entry in the vector $\boldsymbol{w}$, is:

$$
\partial_{w_j} f(\boldsymbol{w}, b; \mathcal{D}) = \frac{\partial f(\boldsymbol{w}, b; \mathcal{D})}{\partial w_j} = w_j + C \sum_{(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathcal{D}} \frac{\partial L(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})}{\partial w_j} \tag{1}
$$

where

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

and the partial derivative with respect to b is:

$$
\partial_b f(\boldsymbol{w}, b; \mathcal{D}) = \frac{\partial f(\boldsymbol{w}, b; \mathcal{D})}{\partial b} = C \sum_{(\boldsymbol{x}^{(i)}, y^{(i)}) \in \mathcal{D}} \frac{\partial L(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})}{\partial b} \tag{2}
$$

where

$$
\frac{\partial L(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta})}{\partial b} = \begin{cases} 0 & \text{if } y^{(i)} \left(\boldsymbol{w} \cdot \boldsymbol{x}^{(i)} + b \right) \geq 1 \\ -y^{(i)} & \text{otherwise.} \end{cases}
$$

Since the direction of the gradient is the direction of steepest ascent of the loss function, gradient descent proceeds by iteratively taking small steps along the direction opposite to the direction of gradient. The general framework of gradient descent is given in Algorithm 1.

Algorithm 1 General Gradient Descent

Parameters: learning rate η, batch size β.

1: Randomly shuffle the training data $\qquad\qquad\qquad\qquad$ ▷ Only for SGD/MBGD
2: $k \leftarrow 0$
3: **for** $t = 1, 2, \dots$ **do**
4: $\qquad B \leftarrow \left\{ \left(x^{(i)}, y^{(i)} \right) : \beta k + 1 \leq i \leq \min\{\beta(k+1), n\} \right\}$
5: $\qquad$ **for** $j = 1, \dots, d$ **do**
6: $\qquad\qquad w_j^{(t)} \leftarrow w_j^{(t-1)} - \eta \cdot \partial_{w_j} f(\boldsymbol{w}^{(t-1)}, b^{(t-1)}; B)$ $\qquad$ ▷ Computed by equation 1
7: $\qquad$ **end for**
8: $\qquad b^{(t)} \leftarrow b^{(t-1)} - \eta \cdot \partial_b f(\boldsymbol{w}^{(t-1)}, b^{(t-1)}; B)$ $\qquad$ ▷ Computed by equation 2
9: $\qquad k \leftarrow (k + 1 \mod \lceil n/\beta \rceil)$
10: $\qquad$ **if** convergence criteria reached **then**
11: $\qquad\qquad$ **break**
12: $\qquad$ **end if**
13: **end for**

Task: Implement the SVM algorithm using the following gradient descent variants.

For all the variants use $C = 100$, $\boldsymbol{w}^{(0)} = \boldsymbol{0}$, $b^{(0)} = 0$. For all other parameters, use the values specified in the description of the variant.

Note: update the parameters $\boldsymbol{w}$ and b on iteration t using the values computed on iteration $t - 1$. Do not update using values computed in the current iteration!

1. **Batch Gradient Descent (BGD)**: When the $\beta = n$, in every iteration the algorithm uses the entire dataset to compute the gradient and update the parameters.

   As a *convergence criterion* for batch gradient descent we will use $\Delta_{\%loss}^{(t)} < \varepsilon$, where

   $$\Delta_{\%loss}^{(t)} = \frac{|f(\boldsymbol{w}^{(t-1)}, b^{(t-1)}; \mathcal{D}) - f(\boldsymbol{w}^{(t)}, b^{(t)}; \mathcal{D})|}{f(\boldsymbol{w}^{(t-1)}, b^{(t-1)}; \mathcal{D})} \times 100 \qquad (3)$$

   Set $\eta = 3 \cdot 10^{-7}$, $\varepsilon = 0.25$.

2. **Stochastic Gradient Descent (SGD)**: When $\beta = 1$, in every iteration the algorithm uses one training sample at a time to compute the gradient and update the parameters.

   As a *convergence criterion* for stochastic gradient descent we will use $\Delta_{loss}^{(t)} < \varepsilon$, where

   $$\Delta_{loss}^{(t)} = \tfrac{1}{2}\Delta_{loss}^{(t-1)} + \tfrac{1}{2}\Delta_{\%loss}^{(t)}, \qquad (4)$$

   t is the iteration number, $\Delta_{\%loss}^{(t)}$ is same as above (equation 3) and and $\Delta_{loss}^{(0)} = 0$.
   Use $\eta = 0.0001$, $\varepsilon = 0.001$.

3. **Mini-Batch Gradient Descent (MBGD)**: In every iteration the algorithm uses mini-batches of β samples to compute the gradient and update the parameters.

As a *convergence criterion* for mini-batch gradient descent we will use $\Delta_{loss}^{(t)} < \varepsilon$, where $\Delta_{loss}^{(t)}$ is the same as above (equation 4) and $\Delta_{loss}^{(0)} = 0$

Use $\eta = 10^{-5}$, $\varepsilon = 0.01$ and $\beta = 20$.

Task: Run your implementation on the data set in `svm/data`. The data set contains the following files :

1. `features.txt` : Each line contains the features (comma-separated values) of a single sample. It has 6414 samples (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 loss function $f(\boldsymbol{w}^{(t)}, b^{(t)}; \mathcal{D})$ vs. the iteration number t starting from $t = 0$. 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, it may have a bug.

What to submit

(i) Plot of $f(\boldsymbol{w}^{(t)}, b^{(t)}; \mathcal{D})$ vs. the number of updates (t). Total time taken for convergence by each of the gradient descent techniques. Interpretation of plot and convergence times. [part (a)]

(ii) Submit the code to Gradescope. [part (a)]

2   Data Streams I (40 points)

You are an astronomer at the Space Telescope Science Institute in Baltimore, Maryland, in charge of the *petabytes* of imaging data they recently obtained. According to the news report linked in the previous sentence, "...*The amount of imaging data is equivalent to two billion selfies, or 30,000 times the total text content of Wikipedia. The catalog data is 15 times the volume of the Library of Congress.*"

This data stream has images of everything out there in the universe, ranging from stars, galaxies, asteroids, to all kinds of awesome exploding/moving objects. Your task is to determine the approximate frequencies of occurrences of different (unique) items in this data stream.

We now introduce our notation for this problem. Let $S = \langle a_1, a_2, \ldots, a_t \rangle$ be the given data stream of length t. Let us denote the items in this data stream as being from the set $\{1, 2, \ldots, n\}$. For any $1 \leq i \leq n$, we denote $F[i]$ to be the number of times i has appeared in S. Our goal is then to have good approximations of the values $F[i]$ for all $1 \leq i \leq n$ at all times.

The naïve way to do this is to just keep the counts for each item $1 \leq i \leq n$ separately. However, this will require $\mathcal{O}(n)$ space which, in our application, is clearly infeasible. We shall see that it is possible to approximate these counts using a much smaller amount of space. To do so, we consider the algorithm explained below.

Algorithm. The algorithm has two parameters δ and $\varepsilon > 0$, and $\left\lceil \log \frac{1}{\delta} \right\rceil$ *independent* hash functions

$$h_j : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, \left\lceil \frac{e}{\varepsilon} \right\rceil\}.$$

Note that in this problem, log denotes the natural logarithm. For each bucket b of each hash function j, the algorithm has a counter $c_{j,b}$ that is initialized to zero.

As each element i arrives in the data stream, it is hashed by each of the hash functions, and the count for the j-th hash function $c_{j,h_j(i)}$ is incremented by 1.

For any $1 \leq i \leq n$, we define $\widetilde{F}[i] = \min_j \{c_{j,h_j(i)}\}$ as our estimate of $F[i]$.

Task. The goal is to show that $\widetilde{F}[i]$ as defined above provides a good estimate of $F[i]$.

(a) [3 Points]

What is the memory usage of this algorithm (in Big-$\mathcal{O}$ notation)? Give a one or two line justification for the value you provide.

(b) [4 Points]

Justify that for any $1 \leq i \leq n$:
$$\widetilde{F}[i] \geq F[i].$$

(c) [11 Points]

Prove that for any $1 \leq i \leq n$ and $1 \leq j \leq \lceil \log(\frac{1}{\delta}) \rceil$:

$$\mathbb{E}\left[c_{j,h_j(i)} \right] \leq F[i] + \frac{\varepsilon}{e} t,$$

where, as mentioned, t is the length of the stream.

(d) [11 Points]

Prove that:

$$\mathbb{P}\left[\widetilde{F}[i] \leq F[i] + \varepsilon t\right] \geq 1 - \delta.$$

Hint: Use Markov inequality and the independence of hash functions.

Based on the proofs in parts (b) and (d), it can be inferred that $\widetilde{F}[i]$ is a good approximation of $F[i]$ for any item i such that $F[i]$ is not very small (compared to t). In many applications (*e.g.*, when the values $F[i]$ have a heavy-tail distribution), we are indeed only interested in approximating the frequencies for items which are not too infrequent. We next consider one such application.

(e) [11 Points]

Warning. This implementation question requires substantial computation time Python implementation reported to take 15min - 1 hour. Therefore, we advise you to start early.

Dataset. The dataset in **streams/data** contains the following files:

1. `words_stream.txt` Each line of this file is a number, corresponding to the ID of a word in the stream.

2. `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.

3. `words_stream_tiny.txt` and `counts_tiny.txt` are smaller versions of the dataset above that you can use for debugging your implementation.

4. `hash_params.txt` Each line is a pair of numbers separated by a tab, corresponding to parameters a and b which you may use to define your own hash functions (See explanation below).

Instructions. Implement the aforementioned algorithm and run it on the dataset with parameters $\delta = e^{-5}$, $\varepsilon = e \times 10^{-4}$. (Note: with this choice of δ you will be using 5 hash functions - the 5 pairs (a, b) that you'll need for the hash functions are in `hash_params.txt`). Then for each distinct word i in the dataset, compute the relative error $E_r[i] = \frac{\widetilde{F}[i] - F[i]}{F[i]}$ and plot these values as a function of the exact word frequency $\frac{F[i]}{t}$. **You do not have to implement the algorithm in Spark.**

The plot should use a logarithm scale both for the x and the y 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 axis labels. 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 we proved in question (a).

Answer the following question by reading values from your plot: What is an approximate condition on a word frequency in the document to have a relative error below $1 = 10^0$?

Hash functions. You may use the following hash function (see example pseudocode), with $p = 123457$, a and b values provided in the hash params file and n_buckets (which is equivalent to $\left\lceil \frac{e}{\varepsilon} \right\rceil$) 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 n_buckets
def hash_fun(a, b, p, n_buckets, x) {
    y = x [modulo] p
    hash_val = (a*y + b) [modulo] p
    return hash_val [modulo] n_buckets
}
```

Note: This hash function implementation produces outputs of value from 0 to (n_buckets $-$ 1), which is different from our specification in the **Algorithm** part. You can either keep the range as $\{0, ..., \text{n\_buckets} - 1\}$, or add 1 to the hash result so the value range becomes $\{1, ..., \text{n\_buckets}\}$, as long as you stay consistent within your implementation.

What to submit

(i) Expression for the memory usage of the algorithm and justification. [part (a)]

(ii) Proofs for parts (b)-(d).

(iii) Log-log plot of the relative error as a function of the frequency. Answer for which word frequencies is the relative error below 1. [part (e)]

(iv) Submit the code to Gradescope. [part (e)]

3   Data Streams II (30 points)

We are in the same setup as the previous part, working with the stream $S = \langle a_1, a_2, \ldots, a_t \rangle$ consisting of items from the set $\{1, 2, \ldots, n\}$; the frequency of element i is again denoted by $F(i)$. Impressed by the quality of your estimator for frequency of occurences of different objects in the telescope's data stream, you have been now tasked with estimating more sophisticated summary statistics from this stream. We'll now explore how to estimate the

sum of *squared* frequencies of all the items in the data stream. That is, we wish to estimate $M = \sum_{i=1}^{n} F(i)^2$. Here's a proposed algorithm.

Algorithm.

- Fix a function $h : [n] \rightarrow \{\pm 1\}$ that associates each item in the data stream with a random sign.

- Initialize $Z = 0$.

- Every time an element j appears in the data stream, add $h(j)$ to Z. That is, increment Z if $h(j) = +1$ and decrement Z if $h(j) = -1$.

- After processing all the t elements in the data stream, return the estimate $X = Z^2$

Note that since we fix h *before* we receive the data stream, an element j is treated consistently every time it shows up: Z is either incremented every time j shows up or is decremented every time j shows up. In the end, element j contributes $h(j) \cdot F(j)$ to the final value of Z.

(a) [12 Points]

Prove that

$$\mathbb{E}_h[X] = M$$

(b) [18 Points]

Prove that

$$\mathrm{Var}(X) \leq 4M^2$$

Note that this bound is not at all tight. A tighter bound can be easily obtained by a more careful derivation similar to the one required for proving the requested bound.

Thus, part (a) shows that the estimator designed in the given algorithm is *unbiased*, while part (b) gives a bound on its variance.

What to submit

(i) Proof that $\mathbb{E}_h[X] = M$. [parts (a)]

(ii) Proof that $\mathrm{Var}(X) \leq 4M^2$. [part (b)]