CSCI567 2017 Homework Assignment 3

Due date and time Oct. 22nd (Sunday), 11:59pm

Starting point Your repository will have now a directory 'homework3/'. Please do not change the name of this repository or the names of any files we have added to it. Please perform a `git pull` to retrieve these files. You will find within it:

- A python script `pegasos.py`, which you will be amending by adding code for questions in Sect. 5.
- A script `pegasos.sh`. You will use it to generate output files after you complete `pegasos.py`.
- Dataset: **mnist_subset.json**

Submission Instructions The following will constitute your submission:

- The python script `pegasos.py`, amended with the code you added for Sect. 5. Be sure to commit your changes!
- A PDF report named `firstname_lastname_USCID.pdf`, which contains your solutions for questions in Sect. 1, Sect. 2, and Sect. 3. For your written report, your answers must be typeset with LaTeX and generated as a PDF file. No handwritten submission will be permitted. There are many free integrated LaTeX editors that are convenient to use, please Google search them and choose the one(s) you like the most. This `http://www.andy-roberts.net/writing/latex` seems like a good tutorial.
- A .json files `pegasos.json`, which will be the output of `pegasos.sh`.

Collaboration You may discuss with your classmates. However, you need to write your own solutions and submit separately. Also in your written report, you need to list with whom you have discussed for each problem. Please consult the syllabus for what is and is not acceptable collaboration.

Algorithmic component

1 Kernels

[Recommended maximum time spent: 1 hour]

Consider the following kernel function:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} 1, & \text{if } \boldsymbol{x} = \boldsymbol{x}' \\ 0, & \text{otherwise} \end{cases}, \quad \forall \boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^D. \tag{1}$$

Q1.1 Prove that this is a valid kernel. You can apply the Mercer's theorem mentioned in the lectures (lec10.pdf) and assume that the N points $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$ you pick are distinct (i.e., $\boldsymbol{x}_i \neq \boldsymbol{x}_j$ if $i \neq j$).

What to submit: No more than 10 lines of proof.

Q1.2 Suppose now you are given a training set $\{(\boldsymbol{x}_n \in \mathbb{R}^D, y_n \in \mathbb{R})\}_{n=1}^N$ for a regression problem, where $\boldsymbol{x}_i \neq \boldsymbol{x}_j$ if $i \neq j$. Show that by using this kernel, training a kernel ridge regressor (lec10.pdf) with $\lambda = 0$ will always lead to the training objective of 0—meaning that all the training examples are *predicted accurately* by the learned regressor. The training objective of kernel ridge regression is as follows

$$J(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{K}^T \boldsymbol{K} \boldsymbol{\alpha} - \boldsymbol{y}^T \boldsymbol{K} \boldsymbol{\alpha} + \frac{\lambda}{2} \boldsymbol{\alpha} \boldsymbol{K} \boldsymbol{\alpha} + \frac{1}{2} \boldsymbol{y}^T \boldsymbol{y}, \tag{2}$$

where $\boldsymbol{K} \in \mathbb{R}^{N \times N}$ is the kernel matrix, $\boldsymbol{y} = [y_1, y_2, \cdots, y_N]^T$, and $\boldsymbol{\alpha} \in \mathbb{R}^N$. The learned regressor is

$$f(\boldsymbol{x}) = [k(\boldsymbol{x}, \boldsymbol{x}_1), k(\boldsymbol{x}, \boldsymbol{x}_2), \cdots, k(\boldsymbol{x}, \boldsymbol{x}_N)]\boldsymbol{\alpha}^*, \tag{3}$$

where $\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha})$

What to submit: No more than 5 lines of derivations, plus 1 line to show what $\boldsymbol{\alpha}^*$ is.

Q1.3 Although the learned regressor can accurately predict the value of each training example, it does not generalize to the test data. Specifically, show that for any $\boldsymbol{x}$ with $\boldsymbol{x} \neq \boldsymbol{x}_n, \forall n = 1, 2, \cdots, N$, the predicted value is always 0.

What to submit: No more than 5 lines of derivations.

2 Support Vector Machines

[Recommended maximum time spent: 2 hours]

Consider the dataset consisting of points (x, y), where x is a real value, and $y \in \{-1,\ 1\}$ is the class label. Let's start with three points $(x_1, y_1) = (-1,\ -1),\ (x_3, y_3) = (0,\ 1),\ (x_2, y_2) = (1, -1)$.

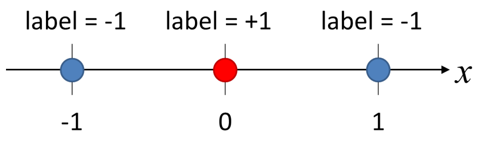


Figure 1: Three data points considered in Q2

Q2.1 Can three points shown in Figure 1, in their current one-dimensional feature space, be perfectly separated with a linear separator? Why or why not?

What to submit: No more than 3 lines of reasoning.

Q2.2 Now we define a simple feature mapping $\phi(x) = [x, x^2]^T$ to transform the three points from one- to two-dimensional feature space. Plot the transformed points in the new two-dimensional feature space (use any package you prefer for the plot, e.g., Matplotlib, PowerPoint). Is there a linear decision boundary that can separate the points in this new feature space? Why or why not?

What to submit: The plot and no more than 3 lines of reasoning.

Q2.3 Given the feature mapping $\phi(x) = [x, x^2]^T$, write down the kernel function $k(x, x')$. Moreover, write down the 3×3 kernel (or Gram) matrix $\boldsymbol{K}$ based on $k(x_i, x_j)$ of the three data points. Verify that $\boldsymbol{K}$ is a positive semi-definite (PSD) matrix. You may want to show this by the definition of PSD matrix: a symmetric $N \times N$ real matrix $\boldsymbol{M}$ is said to be positive semi-definite if the scalar $\boldsymbol{z}^T \boldsymbol{M} \boldsymbol{z}$ is non-negative for every non-zero column vector $\boldsymbol{z}$ of N real numbers.

What to submit: The form of kernel function k, the kernel matrix $\boldsymbol{K}$, and no more than 10 lines of proof to show that $\boldsymbol{K}$ is PSD.

Q2.4 Now you are going to learn a *hard margin* SVM classifier on this dataset with $k(x_i, x_j)$. Recall the primal and dual formulation you learned in lecture 11 (lec11.pdf). Write down the primal and dual formulations of this problem.

What to submit: Primal and dual formulations. Each with no more than 5 lines.

Q2.5 Next, you are going to solve this problem using its dual formulation. Recall the toy example we walked through in lecture 12 (lec12.pdf). You may want to borrow a similar symmetric property to simplify the dual formulation.

What to submit: Solve the dual formulation and report the vector $\boldsymbol{\alpha} \in \mathbb{R}^3$, the weight vector $\boldsymbol{w} \in \mathbb{R}^2$ and the bias $b \in \mathbb{R}$. No more than 10 lines of derivation.

Q2.6 Let $\hat{y} = \boldsymbol{w}^T\phi(\boldsymbol{x}) + b$, where $\boldsymbol{w}$ and b are the weights and the bias you got from the previous question. Draw the decision boundary in the new two-dimensional feature space and circle all support vectors. (Set $\hat{y}$ to 0 to get the decision boundary). Then, draw the decision boundary in the original one-dimensional setting.

What to submit: Two plots: one in the new two-dimensional feature space, and the other in the original one-dimensional feature space.

3 Adaboost for building up a nonlinear classifier

[Recommended maximum time spent: 2 hour]

In the lectures (lec13.pdf), you learned an algorithm, Adaboost, which constructs a strong (binary) classifier based on iteratively adding one weak (binary) classifier into it. A weak classifier is learned to maximize the *weighted training accuracy* at the corresponding iteration, and the weight of each training example is updated after every iteration. Algorithm 1 summarizes the algorithm of Adaboost.

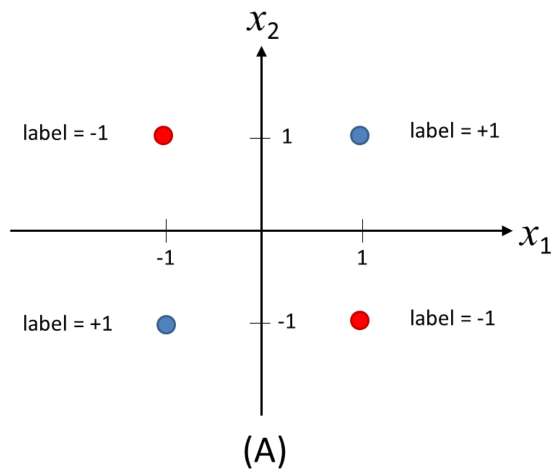
```
Given
$\mathcal{H}$: A set of functions, where $h \in \mathcal{H}$ takes a $D$-dimensional vector as input and outputs
either $+1$ or $-1$.
A training set $\{(\boldsymbol{x}_n \in \mathbb{R}^D, y_n \in \{+1, -1\})\}_{n=1}^N$.

Goal Learn $F(\boldsymbol{x}) = \text{sign}[\sum_{t=1}^T \beta_t f_t(\boldsymbol{x})]$, where $f_t \in \mathcal{H}$ and $\beta_t \in \mathbb{R}$.

Initialization $w_1(n) = \frac{1}{N}, \forall n \in \{1, 2, \cdots, N\}$.

for $t = 1, 2, \cdots, T$ do
    (a) find $f_t = \arg\min_{h\in\mathcal{H}} \sum_n w_t(n)\mathbb{I}[y_n \neq h(\boldsymbol{x}_n)]$
    (b) compute $\epsilon_t = \sum_n w_t(n)\mathbb{I}[y_n \neq f_t(\boldsymbol{x}_n)]$
    (c) compute $\beta_t = \frac{1}{2}\log\frac{1-\epsilon_t}{\epsilon_t}$
    (d) compute $w_{t+1}(n) = \begin{cases} w_t(n)\exp(-\beta_t), & \text{if } y_n = f_t(\boldsymbol{x}_n) \\ w_t(n)\exp(\beta_t), & \text{otherwise} \end{cases}, \forall n \in \{1, 2, \cdots, N\}$
    (e) normalize $w_{t+1}(n) \leftarrow \frac{w_{t+1}(n)}{\sum_{n'} w_{t+1}(n')}, \forall n \in \{1, 2, \cdots, N\}$
end
```

Algorithm 1: The Adaboost algorithm



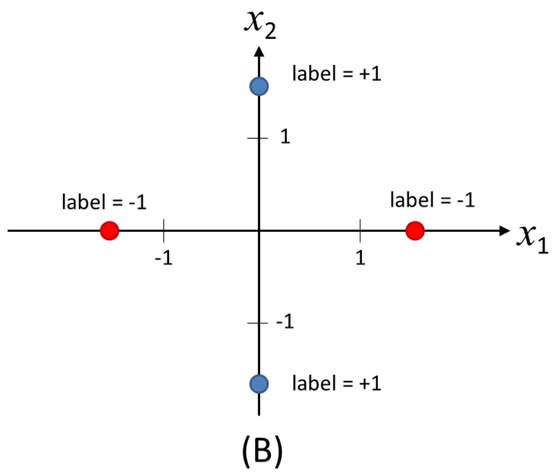


Figure 2: The two training sets considered in Q3: (A) for Q3.1, Q3.2, and (B) for Q3.3, Q3.4, Q3.5, Q3.6.

In this question, you are going to experiment with the learning process of Adaboost, with decision stumps as $\mathcal{H}$. A decision stump $h \in \mathcal{H}$ is a classifier characterized by a triplet $(s \in \{+1, -1\}, b \in \mathbb{R}, d \in \{1, 2, \cdots, D\})$ such that

$$h_{(s,b,d)}(\boldsymbol{x}) = \begin{cases} s, & \text{if } x_d > b, \\ -s, & \text{otherwise.} \end{cases} \tag{4}$$

That is, each decision stump function only looks at a single dimension/entry x_d of the input vector $\boldsymbol{x}$, and check whether x_d is larger than b or not (s decides which label to give if $x_d > b$).

Specifically, you are given a simple 4-example training set (as illustrated in Fig. 2 (A)):

$$(\boldsymbol{x}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, y_1 = +1), \quad (\boldsymbol{x}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, y_2 = -1), \quad (\boldsymbol{x}_3 = \begin{bmatrix} -1 \\ -1 \end{bmatrix}, y_3 = +1), \quad (\boldsymbol{x}_4 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, y_4 = -1)$$

For simplicity, let's consider

$$\mathcal{H} = \{h_{(s,b,d)} \mid s \in \{+1, -1\}, b \in \{-2, -0.5, 0.5, 2\}, d \in \{1, 2\}\}. \tag{5}$$

That is, $\mathcal{H}$ contains only 16 distinct decision stump functions (8 horizontal boundaries and 8 vertical boundaries).

Q3.1 Let's get started to run Adaboost for $T = 3$. At $t = 1$, please find the best decision stump f_1 (i.e., solve step (a) in Algorithm 1). If there are multiple equally best stump functions, just randomly pick **ONE** of them to be f_1. What are the corresponding ϵ_1 and β_1 based on f_1?

What to submit: Write down the triplet (s, b, d) of f_1, and the values of ϵ_1 and β_1. Please round your results to two decimal places (e.g., 1.499 becomes 1.50).

Q3.2 From your results in Q3.1 and Algorithm 1, you will observe something interesting about Adaboost. That is, if $\beta_t = 0$, the corresponding iteration t does not contribute anything to the final decision function $F(\boldsymbol{x})$. This might be fine since we can move on to the next iteration and probably get $\beta_{t+1} \neq 0$ there. Let's have a try. Please write down $w_2(n), \forall n \in \{1, 2, \cdots, N\}$, and check the objective function to solve at $t = 2$ (i.e., solve step (a) in Algorithm 1). You will observe that the objective function at $t = 2$ remains exactly the **SAME** as at $t = 1$. The Adaboost algorithm thus will likely get stuck after iteration $t = 1$.

What to submit: Write down $w_2(n), \forall n \in \{1, 2, 3, 4\}$. Please round your results to two decimal places (e.g., 1.499 becomes 1.50).

Q3.3 From the observations in Q3.2, it seems like Adaboost is not really learning something *strong*. We argue that it is because of the form of decision stumps: each of them can only draw a horizontal or vertical decision boundary in our 2-dimensional case. Can we do any simple trick to solve this issue?

Let's try one trick that you have learned, feature transform. Specifically, we apply a simple linear transformation for every example

$$\boldsymbol{x} \leftarrow \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix} \boldsymbol{x}, \tag{6}$$

which results in a transformed training set

$$(\boldsymbol{x}_1 = \begin{bmatrix} 0 \\ \sqrt{2} \end{bmatrix}, y_1 = +1), \ (\boldsymbol{x}_2 = \begin{bmatrix} -\sqrt{2} \\ 0 \end{bmatrix}, y_2 = -1), \ (\boldsymbol{x}_3 = \begin{bmatrix} 0 \\ -\sqrt{2} \end{bmatrix}, y_3 = +1), \ (\boldsymbol{x}_4 = \begin{bmatrix} \sqrt{2} \\ 0 \end{bmatrix}, y_4 = -1).$$

From now on, let's only consider this new training set (as illustrated in Fig. 2 (B)), but use the same $\mathcal{H}$ defined in eq. (5).

Let's get started again to run Adaboost for $T = 3$, using the new training set. At $t = 1$, please find the best decision stump f_1 (i.e., solve step (a) in Algorithm 1). If there are multiple equally best stump functions, just randomly pick **ONE** of them to be f_1. What are the corresponding ϵ_1 and β_1 based on f_1?

What to submit: Write down the triplet (s, b, d) of f_1, and the values of ϵ_1 and β_1. Please round your results to two decimal places (e.g., 1.499 becomes 1.50).

Q3.4 Now you should see that Adaboost does learn something at $t = 1$ (i.e., $\beta_1 \neq 0$). Let's move on to $t = 2$. Please first write down $w_2(n), \forall n \in \{1, 2, 3, 4\}$ for each example. Then, please find the best decision stump f_2 (i.e., solve step (a) in Algorithm 1). If there are multiple equally best stump functions, just randomly pick **ONE** of them to be f_2. What are the corresponding ϵ_2 and β_2 based on f_2?

What to submit: Write down $w_2(n), \forall n \in \{1, 2, 3, 4\}$, the triplet (s, b, d) of f_2, and the values of ϵ_2 and β_2. Please round your results to two decimal places (e.g., 1.499 becomes 1.50).

Q3.5 Let's move on to $t = 3$. Please first write down $w_3(n), \forall n \in \{1, 2, 3, 4\}$ for each example. Then, please find the best decision stump f_3 (i.e., solve step (a) in Algorithm 1). If there are multiple equally best stump functions, just randomly pick **ONE** of them to be f_3. What are the corresponding ϵ_3 and β_3 based on f_3?

What to submit: Write down $w_3(n), \forall n \in \{1, 2, 3, 4\}$, the triplet (s, b, d) of f_3, and the values of ϵ_3 and β_3. Please round your results to two decimal places (e.g., 1.499 becomes 1.50).

Q3.6 Please combine your results of Q3.3, Q3.4, and Q3.5, and write down your learned $F(\boldsymbol{x})$ in terms of $h_{(s,b,d)}$, β_1, β_2, and β_3. You should plug **VALUES** into $s, b, d, \beta_1, \beta_2, \beta_3$. For example, $F(\boldsymbol{x}) = \text{sign}[0.5 \times h_{(+1,0.5,1)}(\boldsymbol{x}) + 0.5 \times h_{(-1,0.5,1)}(\boldsymbol{x}) + 0.5 \times h_{(-1,2,2)}(\boldsymbol{x})]$. Then please write down the predicted labels of $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \boldsymbol{x}_4$ using $F(\boldsymbol{x})$. How many training examples are correctly labeled by $F(\boldsymbol{x})$?

What to submit: Write down $F(\boldsymbol{x})$, and compute $F(\boldsymbol{x}_1), F(\boldsymbol{x}_2), F(\boldsymbol{x}_3), F(\boldsymbol{x}_4)$, and the number of correctly labeled examples. Please round your results to two decimal places (e.g., 1.499 becomes 1.50).

Q3.7 From Q3.6, you should see that Adaboost with decision stumps can solve a problem similar to XOR in 3 iterations, by using a simple linear transformation (more precisely, 2-dimensional rotation). Note that the decision boundary by $F(\boldsymbol{x})$ is nonlinear, even though that $h \in \mathcal{H}$ can only produce a linear boundary. Also note that, even with the same feature transform, a linear classifier such as logistic regression cannot correctly label every training example in our case.

What to submit: Nothing.

Programming component

4 High-level descriptions

4.1 Dataset

We will use **mnist_subset** (images of handwritten digits from 0 to 9). This is the same subset of the full MNIST that we used for Homework 1 and Homework 2. As before, the dataset is stored in a JSON-formated file **mnist_subset.json**. You can access its training, validation, and test splits using the keys 'train', 'valid', and 'test', respectively. For example, suppose we load **mnist_subset.json** to the variable x. Then, $x['train']$ refers to the training set of **mnist_subset**. This set is a list with two elements: $x['train'][0]$ containing the features of size N (samples) $\times D$ (dimension of features), and $x['train'][1]$ containing the corresponding labels of size N.

4.2 Tasks

You will be asked to implement the linear support vector machine (SVM) for binary classification (Sect. 5). Specifically, you will

- finish implementing the following three functions—`objective_function`, `pegasos_train`, and `pegasos_test`—in `pegasos.py`. Refer to `pegasos.py` and Sect. 5 for more information.
- Run the script `pegasos.sh` after you finish your implementation. This will output `pegasos.json`.
- add, commit, and push (1) the `pegasos.py`, and (2) the `pegasos.json` file that you have created.

As in the previous homework, you are not responsible for loading/pre-processing data; we have done that for you (e.g., we have pre-processed the data to have to class: digits 0–4 and digits 5–9.). For specific instructions, please refer to text in Sect. 5 and the instructions in `pegasos.py`.

4.3 Cautions

Please do not import packages that are not listed in the provided code. Follow the instructions in each section strictly to code up your solutions. **Do not change the output format**. **Do not modify the code unless we instruct you to do so**. A homework solution that does not match the provided setup, such as format, name, initializations, etc., **will not** be graded. It is your responsibility to **make sure that your code runs with the provided commands and scripts on the VM**. Finally, make sure that you **git add, commit, and push all the required files**, including your code and generated output files.

5 Pegasos: a stochastic gradient based solver for linear SVM

In this question, you will build a linear SVM classifier using the Pegasos algorithm [1]. Given a training set $\{(\boldsymbol{x}_n \in \mathbb{R}^D, y_n \in \{1, -1\})\}_{n=1}^N$, the primal formulation of linear SVM is as follows

$$\min_{\boldsymbol{w}} \frac{\lambda}{2}||\boldsymbol{w}||_2^2 + \frac{1}{N}\sum_n \max\{0, 1 - y_n \boldsymbol{w}^T \boldsymbol{x}_n\}. \qquad (7)$$

Instead of turning it into dual formulation, we are going to solve the primal formulation directly with a gradient-base algorithm. *Note that here we include the bias term b into parameter $\boldsymbol{w}$ by appending $\boldsymbol{x}$ with* 1.

In (batch) gradient descent, at each iteration of parameter update, we compute the gradients for all data points and take the average (or sum). When the training set is large (i.e., N is a large number), this is often too computationally expensive (for example, a too large chunk of data cannot be held in memory). Stochastic gradient descent with mini-batch alleviates this issue by computing the gradient on a *subset* of the data at each iteration.

One key issue of using (stochastic) gradient descent to solve eq. (7) is that $\max\{0, z\}$ is not differentiable at $z = 0$. You have seen this issue in Homework 2, where we use the Heaviside function to deal with it. In this question, you are going to learn and implement Pegasos, a representative solver of eq. (7) that applies stochastic gradient descent with mini-batch and explicitly takes care of the non-differentiable issue.

The pseudocode of Pegasos is given in Algorithm 2. At the t-th iteration of parameter update, we first take a mini-batch of data A_t of size K `[step (a)]`, and define $A_t^+ \subset A_t$ to be the subset of samples for which $\boldsymbol{w}_t$ suffers a non-zero loss `[step (b)]`. Next we set the learning rate $\eta_t = 1/(\lambda t)$ `[step (c)]`. We then perform a **two-step parameter update** as follows. We first compute

$(1-\eta_t\lambda)\boldsymbol{w}_t$ and for all samples $(\boldsymbol{x}, y) \in A_t^+$ we add the vector $\frac{y\eta_t}{K}\boldsymbol{x}$ to $(1-\eta_t\lambda)\boldsymbol{w}_t$. We denote the resulting vector by $\boldsymbol{w}_{t+\frac{1}{2}}$ `[step (d)]`. This step can also be written as $\boldsymbol{w}_{t+\frac{1}{2}} = \boldsymbol{w}_t - \eta_t \nabla_t$ where

$$\nabla_t = \lambda \boldsymbol{w}_t - \frac{1}{|A_t|} \sum_{(\boldsymbol{x},y)\in A_t^+} y\boldsymbol{x}$$

The definition of the hinge loss, together with the Heaviside function, implies that ∇_t is the gradient of the objective function on the mini-batch A_t at $\boldsymbol{w}_t$. Last, we set $\boldsymbol{w}_{t+1}$ to be the projection of $\boldsymbol{w}_{t+\frac{1}{2}}$ onto the set

$$B = \{\boldsymbol{w} : ||\boldsymbol{w}|| \leq 1/\sqrt{\lambda}\}$$

This is obtained by scaling $\boldsymbol{w}_{t+1}$ by $\min\{1, \frac{1/\sqrt{\lambda}}{||\boldsymbol{w}_{t+\frac{1}{2}}||}\}$ `[step (e)]`. For details of Pegasos algorithm you may refer to the original paper [1].

```
**Input** A training set $S = \{(\boldsymbol{x}_n \in \mathbb{R}^D, y_n \in \{1,-1\})\}_{n=1}^N$, the total number of iterations $T$,
the batch size $K$, and the regularization parameter $\lambda$.

**Output** The last weight $\boldsymbol{w}_{T+1}$.

**Initialization** Choose $\boldsymbol{w}_1$ s.t. $||\boldsymbol{w}_1|| \leq \frac{1}{\sqrt{\lambda}}$.

for $t = 1, 2, \cdots, T$ do
    (a) Randomly choose a subset of data $A_t \in S$, where $|A_t| = K$
    (b) Set $A_t^+ = \{(\boldsymbol{x}, y) \in A_t : y\boldsymbol{w}_t^T\boldsymbol{x}_t < 1\}$
    (c) Set $\eta_t = \frac{1}{\lambda t}$
    (d) Set $\boldsymbol{w}_{t+\frac{1}{2}} = (1-\eta_t\lambda)\boldsymbol{w}_t + \frac{\eta_t}{K}\sum_{(\boldsymbol{x},y)\in A_t^+} y\boldsymbol{x}$
    (e) Set $\boldsymbol{w}_{t+1} = \min\{1, \frac{1/\sqrt{\lambda}}{||\boldsymbol{w}_{t+\frac{1}{2}}||}\}\boldsymbol{w}_{t+\frac{1}{2}}$
end
```

Algorithm 2: The Pegasos algorithm

Now you are going to implement Pegasos and train a binary linear SVM classifier on the dataset **mnist_subset.json**. You are going to implement three functions—`objective_function`, `pegasos_train`, and `pegasos_test`—in a script named `pegasos.py`. You will find detailed programming instructions in the script.

Q5.1 Finish the implementation of the function `objective_function`, which corresponds to the objective function in the primal formulation of SVM.

Q5.2 Finish the implementation of the function `pegasos_train`, which corresponds to Algorithm 2.

Q5.3 After you train your model, run your classifier on the test set and report the accuracy, which is defined as:

$$\frac{\text{\# of correctly classified test samples}}{\text{\# of test samples}}$$

Finish the implementation of the function `pegasos_test`.

Q5.4 After you complete above steps, run `pegasos.sh`, which will run the Pegasos algorithm for 500 iterations with 6 settings (mini-batch size $K = 100$ with different $\lambda \in \{0.01, 0.1, 1\}$ and $\lambda = 0.1$ with different $K \in \{1, 10, 1000\}$), and output a `pegasos.json` that records the test accuracy and the value of objective function at each iteration during the training process.

What to do and submit: run script `pegasos.sh`. It will generate `pegasos.json`. Add, commit, and push both `pegasos.py` and `pegasos.json` before the due date.

Q5.5 Based on `pegasos.json`, you are encouraged to make plots of the objective function value versus the number of iterations (i.e., a convergence curve) in different settings of λ and k, but you are not required to submit these plots.

References

[1] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, Andrew Cotter *Mathematical Programming, 2011, Volume 127, Number 1, Page 3.* Pegasos: primal estimated sub-gradient solver for SVM.