IIT BOMBAY

Lab Assignment #6
Points: 5

Main TAs: *Vedang Dhirendra Asgaonkar, Vishal Tapase, Sanjeev Kumar*

Course: *CS 335* – Instructor: *Preethi Jyothi*
Due date: *September 11, 2023*

General Instructions

1. Download lab6.tgz from Moodle for CS 335, and extract the file to get `lab6`.
2. The folders `q1/` and `q2/` within `lab6` contain all the files needed to implement Perceptron and SVM classifiers, respectively.
3. Your final submission is due on Moodle on or before **11.59 pm on Sept 11, 2023**.
4. For your final submission, create [rollnumber].tgz with the following internal directory structure:

```
[rollnumber]/
  |
  +-q1/
  |  |
  |  +- perceptron_template.py
  +-q2/
  |  +- svm_template.py
  |  +-images/
  |  |  +- svm-C-1.png
  |  |  +- svm-C-10.png
  |  |  +- svm-C-100.png
  |  |  +- svm-kernel.png
```

5. You will get 2 points if we are able to successfully run `perceptron_template.py` to generate the decision boundaries for all 100 epochs using the vanilla and averaged Perceptrons and create gifs using `make_gif.py`. You will get 1 point if the test accuracy using the vanilla and averaged perceptrons match our accuracies (or are within a tolerance limit). Our test accuracies are roughly 85% and 90% from the vanilla and averaged perceptron classifiers, respectively. You will get 2 points for correctly computing the margins and submitting the required `png` files for q2.

Q1: Perceptron Classifier

For this problem, you will implement a single-layer perceptron classifier. `produce_sample_dataset.py` generates a toy dataset with inputs $\mathbf{x} \in \mathbb{R}^2$ and binary labels $y \in \{-1, 1\}$. Recall, the Perceptron weight update rule for an example $(\mathbf{x}, y)$ is

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t + \eta y \mathbf{x}$$

where $\mathbf{w} \in \mathbb{R}^3$ and η is a learning rate hyperparameter (that we set as 1 in class). Append a 1 at the end of each $\mathbf{x}$ to allow for the bias term in $\mathbf{w}$ to be computed.

Implement the function `fit` inside the class `Perceptron` and do the following for a fixed number of epochs set in `epochs`. *Note this is different from what we studied in class.* If $\mathbf{w}^t$ is the weight vector at the start of epoch number t, then compute the predictions for all training examples using $\mathbf{w}^t$ and compute the sum of $\eta y_i \mathbf{x}_i$ across all $\mathbf{x}_i$'s that result in a misclassification using $\mathbf{w}^t$. If the latter sum is saved in Δ, then the updated weight vector will be:

$$\mathbf{w}^{t+1} \leftarrow \mathbf{w}^t + \Delta$$

This weight update can be done in a single line of code. The vanilla Perceptron classifier then returns the final weight vector after a predefined number of `epochs`.

Exponential moving average (EMA)-variant of the Perceptron: Instead of the final vector, one could also return an averaged weight vector computed across multiple epochs. We suggest a variant that keeps track of the exponential moving average of the weight vector:

$$\mathbf{a}^{t+1} = \lambda \mathbf{a}^t + (1 - \lambda)\mathbf{w}^{t+1} \tag{1}$$

where $\mathbf{w}^t$ is the weight vector at epoch t of the training and $\mathbf{a}^0 = \mathbf{w}^0$. The parameter λ controls how quickly the moving average reacts to changes in $\mathbf{w}$. Higher the value of λ, greater will be its inertia. The exponential moving average can be used in place of the final weight vector to perform the final prediction on the test set. This is seen to give more stable results, as illustrated by our task. Implement both the update for the moving average and the prediction function within `fit`. The `plot_decision_boundary` method stores images of the linear decision boundary changing over time for both the final weight and moving average based predictions. You can then run `q1/make_gif.py` to make an animation out of these images and see the decision boundaries change over time. Notice how the decision boundary is much more stable when the average is used.

Complete all the TODOs marked in `q1/perceptron_template.py`. You should also print the test accuracy using both the final weight vector and the averaged weight vector.

> Note that `imageio==2.27.0` needs to be installed for the script `q1/make_gif.py` to run successfully.

Q2: SVMs with Kernels

This task involves implementing support vector (SV) classifiers for two toy tasks by maximizing the margin. You will also plot the decision boundaries as well as margins learned by the support vector (SV) classifiers. All the code for this question is in the template file `q2/svm_template.py`.

There are two tasks in this question, within functions `svm_task1` and `svm_task2` that use the data files `q2/data1.csv` and `q2/data2.csv`, respectively. Complete all the TODOs marked in `q2/svm_template.py`.

Complete the following functions:

1. `svm_task_1()`: Train an SVM using `sklearn`'s SVC implementation(`https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html`) with a linear kernel to fit the data in `q1/data1.csv`. Plot the decision boundary learned by the SVM classifier for three different values of C (=1,10,100), along with its margins, and save these files within `q2/images` as "svm-C-1.png", "svm-C-10.png" and "svm-C-100.png", respectively.

2. `svm_task_2()`: Train an SVM with a non-linear kernel (polynomial or RBF) with different *C* values to fit the data in `q2/data2.csv`. Visualize the data in `q2/data2.csv` using the function `plot_data`. This will help you assess what kernel might be appropriate to invoke with the SVM classifier. Plot the learned decision boundary and save the file in `q2/images/svm-kernel.png`.