ECE499/CS499 Statistical Signal Processing (Fall 2021)

Final Project: MNIST Digit Classification

(Due: Noon PST, December 12, Sunday)

Instruction: Students are **not** allowed to use any off-the-shelf machine learning libraries or modules in performing the project tasks.

In this project, we will work with the MNIST handwritten digit image dataset. Each digit image $\mathbf{x}$ is a 28 by 28 grayscale image of a handwritten digit (see Fig. 1 for an example), and it is represented by a vector of 784 pixel values, i.e., $\mathbf{x} \in \mathbb{R}^{784}$. Pixel values are normalized such that they range between 0 and 1. The first 28 entries of $\mathbf{x}$ corresponds to the pixel values of the first row of the image, the next 28 entries correspond to the pixel values of the second row, and so on. The label y for the image $\mathbf{x}$ takes a value from $\{0, 1, ..., 9\}$ that corresponds to the digit displayed in the image.

There are four data files: `MNIST_train_image.mat`, `MNIST_train_label.mat`, `MNIST_test_image.mat`, and `MNIST_test_label.mat`. These are all MATLAB mat files; how to load them is explained in Appendix. Below are the descriptions of these data files:

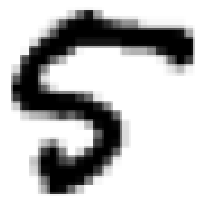
Figure 1: An example image from the digit 5 class

- `MNIST_train_image.mat`: This file contains an 784 by 60,000 array `trainX`. The i-th column of this array is the i-th training image $\mathbf{x}_i$. This array contains total 60,000 training images.
- `MNIST_train_label.mat`: This file contains a 60,000-dimensional vector `trainL`. The i-th entry of this vector is the class label y_i for the i-th training image $\mathbf{x}_i$.
- `MNIST_test_image.mat`: This file contains an 784 by 10,000 array `testX`. Each column of this array is a test image. This array contains total 10,000 test images.
- `MNIST_test_label.mat`: This file contains a 10,000-dimensional vector `testL`. The i-th entry of this vector is the class label for the i-th test image (i.e., the i-th column) in `testX`.

Training and Validation Datasets Partition the set of 60,000 labeled training images from `MNIST_train_image.mat` and `MNIST_train_label.mat` into (i) 50,000 images that will be used for actually training the classifier (from here on, we will refer to this dataset as "training dataset" and denote it by $\mathcal{T}$) and (ii) 10,000 images that will be used for validation, i.e., estimating the test error rate of the trained classifier (we will refer to this dataset as "validation dataset" and denote it by $\mathcal{V}$).

Objective Given the training dataset $\mathcal{T}$ and the validation dataset $\mathcal{V}$, we aim to learn a classifier $\hat{y}(\cdot)$ such that for any digit image $\mathbf{X}$, $\hat{y}(\mathbf{X})$ coincides with the true class label Y with a high probability; more specifically, we aim to find a classifier $\hat{y}(\cdot)$ that minimizes the classification error rate $\Pr(Y \neq \hat{y}(\mathbf{X}))$. Note that only the training dataset and the validation dataset will be used for learning the classifier $\hat{y}(\cdot)$. The test dataset (`MNIST_test_image.mat` and `MNIST_test_label.mat`) will be left out and used only for evaluating the test error rate of the learned classifier $\hat{y}(\cdot)$.

1. (4 points) **Non-parametric Method** Implement a K-Nearest Neighbor (NN) classifier for this classification task. Recall that a K-NN classifier works as follows: given an input variable vector $\mathbf{x} \in \mathbb{R}^{784}$,

   - *Identify the K nearest neighbors:* Compute $\|\mathbf{x} - \mathbf{x}_i\|$ for all $(\mathbf{x}_i, y_i)$'s in the training dataset $\mathcal{T}$. Let $\mathcal{N}_{\mathbf{x}}$ denote the set of K data points in $\mathcal{T}$ that are closest to $\mathbf{x}$ according to the computed distances.
   - *Decision:* Based on $\mathcal{N}_{\mathbf{x}}$, $\hat{y}(\mathbf{x})$ is computed as

     $$\hat{y}(\mathbf{x}) = \arg \max_{c \in \{0,1,2,\ldots,9\}} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{N}_{\mathbf{x}}} \mathbf{1}_{\{y_i = c\}}$$

     In other words, $\hat{y}(\mathbf{x})$ is set to the class that has the maximum number of examples in $\mathcal{N}_{\mathbf{x}}$.

   For a K-NN classifier to be successful, it is important to set K to a proper value. We can determine a proper K value by evaluating the K-NN classifiers with various K values using the validation dataset $\mathcal{V}$. In particular, we can choose the K value that results in the minimum validation error rate, i.e., that minimizes

   $$\frac{1}{|\mathcal{V}|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{V}} \mathbf{1}_{\{\hat{y}(\mathbf{x}_i) \neq y_i\}}$$

   Implement K-NN classifiers with various K values, e.g., $K = 1, 5, 10, 20, 50, 100, 200, 500$. Choose the best K value based on the validation experiment result, and report the resulting test error rate when the chosen K value is used. Plot the training error rate, the validation error rate, and the test error rate versus K, and interpret the results. Provide the codes (Python or MATLAB) that you wrote to perform the aforementioned tasks.

2. (60 points) **Parametric Method** In this question, we will consider the *multinomial logistics regression* model to train a classifier $\hat{y}(\cdot)$ for the digit image classification task.

   (a) (10 points) Provide a clear description of the parametric model assumption.

(b) (10 points) Suppose that we would like to use the cross-entropy loss plus the regularization term $\lambda \cdot \|\boldsymbol{\beta}\|^2$ (where $\|\boldsymbol{\beta}\|^2$ denotes the sum of all squared parameter values) as our loss function. Provide an expression for the loss function $L(\boldsymbol{\beta})$ that clearly shows how the loss function value depends on the training dataset $\mathcal{T}$ and the model parameter values.

(c) (40 points) Implement a Gradient Descent algorithm to minimize the loss function $L(\boldsymbol{\beta})$ you obtained in part (b). Let $\boldsymbol{\beta}^{(i)}$ denote the parameter vector after the i-th iteration of the Gradient Descent algorithm. Plot $L(\boldsymbol{\beta}^{(i)})$ versus i, for $i = 0, 100, 200, 300, ...$ to check whether the loss function value keeps decreasing as we want. If the loss function value does not decrease, you need to fix your implementation to make it work properly. First, double check your code for any bugs. Second, try a different step size because using a too large (or too small) step size can cause a convergence issue for the Gradient Descent method. Note that the training procedure can be performed properly only if you have a working optimization algorithm.

Once you are confident about your implementation of the Gradient Descent method, use it to train the multinomial logistics regression model. Note that the choice of the regularization coefficient λ affects the loss function thereby having an impact on the learned classifier $\hat{y}(\cdot)$. To choose a proper λ value, we will leverage the validation dataset $\mathcal{V}$. We first train multiple multinomial logistics regression models while setting λ to various values (one model for each λ value we try). Then, we evaluate those models using the validation dataset $\mathcal{V}$ and determine the best λ based on the validation experiment results, i.e., choose the λ value that results in the model with the minimum validation error rate. Report the test error rate when the chosen λ is used. Plot the training error rate, the validation error rate, and the test error rate versus λ. Interpret the results. Provide the codes you wrote to perform the aforementioned task.

Appendix A. Loading the data files using MATLAB and Python

If you are using MATLAB, you can load `MNIST_train_image.mat`, `MNIST_train_label.mat`, `MNIST_test_image.mat`, and `MNIST_test_label.mat` simply using the `load` function. For instance, the command line

```
load('MNIST_train_image.mat')
```

will load the training data array `trainX` to the workspace.

If you are using Python, you can use the following script (that uses the `scipy` module) to load the arrays and vectors in those files.

```
import numpy as np
import scipy.io as sio
mat_contents = sio.loadmat('MNIST_train_image.mat')
train_img = mat_contents['trainX']
```

Running the above lines will generate a 784 by $60,000$ `numpy` array `train_img`, which is essentially the same array as the MATLAB array `trainX` saved in `MNIST_train_image.mat`.