Mini Project 1: Least Squares Based Supervised Classification
ECE 174, Fall 2022
Prof. Piya Pal

Mini Project # 1
Due: November 1 , 11:59 pm

Instructions: In this programming assignment, you will implement two different least squares based multi-class classifiers on MNIST data set, and interpret your results.

- You can write the source code in any language of your choice (Python, MATLAB). The code should be properly commented and you will be required to submit the source code. You should write your own code and not copy from any resource on the internet. If you consult any resource online before writing your own code, you must cite the source, add a comment to your code with a link to the source you got it from. Any uncited code you did not write yourself, or any code which is substantially similar to another student's, will be referred to the Office of Academic Integrity.
- In addition to the source code, you will submit a detailed report that describes the simulations and explains the results using figures and mathematical interpretations. Your report must address all the questions asked in this assignment. We do not have any detailed requirements on the formatting, but we expect the report to be presented in a complete and professional manner, with appropriate labels on items and comments which demonstrate understanding of the results.
- Your grade on the programming assignment will be based on the correctness of the code as well as the quality of the report. Since our evaluation of your code is based on its results as presented in your report, correct code is very important. We have every intention of awarding partial credit, but for instance if a small bug causes your program's output to be unusable throughout the report, we cannot award much credit even if the bug was very small.

---

1 Problem 1

Given $\mathbf{A} \in \mathbb{C}^{m\times n}, \mathbf{y} \in \mathbb{C}^m$, show that $\hat{\mathbf{x}}$ solves $\min_{\mathbf{x}\in\mathbb{C}^n} \|\mathbf{y} - \mathbf{A}\mathbf{x}\|_2$ if it is a solution to the following system of equations:

$$\mathbf{A}^H\mathbf{y} = \mathbf{A}^H\mathbf{A}\mathbf{x}$$

1. Do the above equations always have solution? Justify.
2. How many solutions can they have? Justify.
3. How will you solve the above equations if $\text{rank}(\mathbf{A}) < n$?

2 Problem 2: Least Squares Classifier

2.1 Background on Classifier Design

Reading (Necessary for doing the Programming Assignment):

1. **Least Squares Classifier:** Read Sec. 14.1-14.3 from the textbook to get familiar with least-squares based binary and multi-class classifiers. Pay special attention to Sec. 14.2.2, 14.3.1, 14.3.3 to become familiar with different metrics for evaluating classifier performance (such as confusion matrices).

2. **Dataset:** We will use the well-known MNIST (Mixed National Institute of Standards) database of handwritten digits which contains grayscale images of size 28×28 which can be represented as vectors of size 784. The dataset is provided as `mnist.mat` which you can directly read into MATLAB. For those working in Python, you can refer to the Scipy documentation `https://docs.scipy.org/doc/scipy/reference/generated/scipy.io.loadmat.html` for loading a mat file in Python. The image data is in uint8 format, hence the pixel values will range from 0-255. Don't forget to convert the images to float/double (use built-in functions for this operation) and normalise the values to $[0, 1]$.

Binary Classifier

Consider the binary least-squares classifier where the labels can only take two possible values (corresponding to two classes). Suppose we are given N data points $\{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}$ where $\mathbf{x}_i$ denotes the i^{th} feature vector and $y_i \in \{-1, 1\}$ is the corresponding label for each data point. Given N training data points, we try to find the best linear regression model by solving the following least squares problem

$$\min_{\boldsymbol{\beta}, \alpha} \sum_{i=1}^{N} (y_i - \boldsymbol{\beta}^T \mathbf{x}_i - \alpha)^2 \tag{1}$$

Let $\boldsymbol{\beta}^\star, \alpha^\star$ be the solution of the above problem. Then the **binary least squares classifier** $\hat{f}(\mathbf{x})$ is given by

$$\hat{f}(\mathbf{x}) = \text{sign}(\boldsymbol{\beta}^{\star T} \mathbf{x} + \alpha^\star)$$

for arbitrary test data $\mathbf{x}$. Here $\text{sign}(a) = 1$ for $a \geqslant 0$ and -1 for $a < 0$.

Multi-class Classifier

One-versus-all classifier: Now, we extend this idea to K classes. The **One-versus-all classifier** $\hat{f}(\mathbf{x})$ is given by

$$\hat{f}(\mathbf{x}) = \arg \max_{k=1,\cdots,K} g_k(\mathbf{x}) \tag{2}$$

where $g_k(\mathbf{x}) = \boldsymbol{\beta}_k^T \mathbf{x} + \alpha_k$ is the least squares regression model corresponding to the binary classifier for k^{th} label against the others. That is, for training data $\{(\mathbf{x}_i, y_i), i = 1, 2, \ldots, N\}$ the output of $g_k(\mathbf{x})$ is such that:

$$\text{sign}(g_k(\mathbf{x}_i)) = \begin{cases} 1 & \text{if } y_i = k \\ -1 & \text{if } y_i \neq k \end{cases}$$

Basically, you are supposed to solve (1) K times on the given training data set $\{(\mathbf{x}_i, y_i), i = 1, \ldots, N\}$. Note that you have to transform the original training labels $y_i \in \{1, \cdots, K\}$ to appropriate binary values to solve each of these K problems. The classifier $\hat{f}(\mathbf{x})$ will return the index corresponding to the function in $\{g_k(\mathbf{x})\}_{k=1}^K$ that returns the maximum value for the input $\mathbf{x}$.

One-versus-One classifier: In this part, we design pairwise binary classifier (with parameters $\boldsymbol{\beta}_{i,j}, \alpha_{i,j}$) for every pair of classes i, j $(1 \leqslant i < j \leqslant K)$ as described above. That is, we have a set of binary classifiers $\{\hat{f}_{i,j}(\mathbf{x}), 1 \leqslant i < j \leqslant K\}$ such that the output of the i-versus-j classifier $\hat{f}_{i,j}(\mathbf{x})$ indicates:

$$\hat{f}_{i,j}(\mathbf{x}) = \text{sign}(\boldsymbol{\beta}_{i,j}\mathbf{x} + \alpha_{i,j}) = \begin{cases} 1 & \text{if label} = i \\ -1 & \text{if label} = j \end{cases}$$

Remember that in this case, in order to design each classifier, you will only consider subsets of training data points that have class labels i and j. For obtaining $\boldsymbol{\beta}_{i,j}, \alpha_{i,j}$ you can treat the label corresponding to class i as $+1$ and that corresponding to class j as -1 and then solve (1). This gives us a total of $K(K-1)/2$ classifiers called one-versus-one classifiers.

Given a test feature vector $\mathbf{x}$, let $y_{i,j}$ be the prediction of the i-versus-j classifier, with $y_{i,j} = 1$ meaning that the vector belongs to class i and $y_{i,j} = -1$ meaning it belongs to class j. Depending on the outcomes of the binary classifiers, we increment the "vote" of the appropriate class. The final predicted label is decided by picking the class which has the maximum number of votes after passing $\mathbf{x}$ through all $K(K-1)/2$ classifiers. (You can break ties by choosing the smallest index that achieves the largest number of votes.)

Training and Testing Multi-class Classifiers on MNIST Dataset: For the least-squares classifier, use the training data from the MNIST dataset for training the classifiers (i.e. learning the least square parameters) and use the testing data to test the performance of the classifier. The objective is of course to identify the digits which can take values between $0-9$ (total of $K = 10$ classes) from their corresponding handwritten images. The dataset consists of $N = 60,000$ training images and $10,000$ test images. (**CAUTION:** Do **NOT** mix training data and testing data)

2.2 Tasks

1. Write your own code to implement the (i) one-versus-one and (ii) one-versus-all multi-class classifier using the binary classifier as building blocks. Your code should take appropriately labeled training data as the input, and determine the weights of the constituent binary classifiers as the output.

2. Evaluate the training error of both classifiers using error rate and confusion matrix:

- The error rate is the total number of errors in predicted label divided by the total number of inputs being classified.
- Confusion matrix (ref. Pg. 287 and 298 from the textbook)

3. Evaluate the performance of both multi-class classifiers on the test data. Comment on their performance. How well do they generalize on the test data? Which digits are easy to recognize, which ones are harder?

3 Problem 3: Randomized Feature Based Least Square Classifiers

3.1 Background

Let $\mathbf{h} : \mathbb{R}^d \to \mathbb{R}^L$ be a fixed *feature map* that transforms the data $\mathbf{x} \in \mathbb{R}^d$ (d-dimensional input space) to a feature $\mathbf{h}(\mathbf{x}) \in \mathbb{R}^L$ in the L-dimensional "feature space". Consider the data points $(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$, where $\mathbf{x}_i$ denotes the input data and $y_i \in \{-1, 1\}$ is the corresponding label for the i^{th} data point. Given N training data points, we aim to find the best linear regression model in the feature space, by solving the following least squares problem:

$$\min_{\boldsymbol{\beta}, \alpha} \sum_{i=1}^{N} (y_i - \boldsymbol{\beta}^T \mathbf{h}(\mathbf{x}_i) - \alpha)^2 \tag{3}$$

In Problem 2, we learnt a classifier in the input space, that computes a linear combination of the input variables ($\mathbf{x}_i$'s). Instead, here we aim to learn a linear classifier in the feature space, which may no longer be a linear function of $\mathbf{x}$, depending on the choice of the feature mapping $\mathbf{h}(\mathbf{x})$.

Designing randomized feature maps: For designing the feature mapping $\mathbf{h}(.)$, generate a random matrix $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \cdots, \mathbf{w}_L]^T \in \mathbb{R}^{L \times d}$, and a vector $\mathbf{b} = [b_1, b_2, \cdots, b_L]^T \in \mathbb{R}^L$ with their entries drawn independently from a Gaussian distribution $W_{i,j} \sim \mathcal{N}(0, 1)$ for $1 \leqslant i \leqslant L, 1 \leqslant j \leqslant d$ and $b_i \sim \mathcal{N}(0, 1)$ for $1 \leqslant i \leqslant L$. Then, the feature mapping $\mathbf{h}(\mathbf{x})$ is defined as:

$$\mathbf{h}(\mathbf{x}) := \begin{bmatrix} g(\mathbf{w}_1^T \mathbf{x} + b_1) \\ g(\mathbf{w}_2^T \mathbf{x} + b_2) \\ \vdots \\ g(\mathbf{w}_L^T \mathbf{x} + b_L) \end{bmatrix}$$

Here, $g(.)$ is a real valued function used to compute the feature mappings.

3.2 Tasks

1. Modify the code for Problem 2 to learn the multi-class classifier in the feature space. Use the following non-linear mappings to construct your feature vectors:

   - **Identity Function:** $g(x) = x$
   - **Sigmoid Function:** $g(x) = \frac{1}{1+e^{-x}}$
   - **Sinusoidal Function:** $g(x) = \sin(x)$
   - **Rectified Linear Unit (ReLU) function**: $g(x) = \max(x, 0)$

   Assume that $L = 1000$ for this part. Compare the classification performance for different choices of the feature mapping by changing the non-linearity. Do these feature mappings perform better than the classifier learnt in Problem 2 on (i) Training data (ii) Testing data ? Which feature mapping generalizes well, i.e., continues to perform well on the testing data? Discuss your results.

2. Vary the number of features L, and plot the error rate as a function of the number of features L. Comment on how adding more features affects your classifier design.

3. (**Bonus**) **Robustness of Classifier:** In the previous part, the classifier was trained on "clean training images". However, in practice the classifier may encounter noisy images. Through this problem, we will evaluate how robust is a given classifier, i.e., can it continue to provide correct classification even if the pixels are corrupted by noise?

   Generate a set of noisy test samples $\mathcal{T}_\epsilon = \left\{\mathbf{x}_i^{(\epsilon)}\right\}_{i=1}^{N}$ by adding bounded noise to the test data:

   $$\mathbf{x}_i^{(\epsilon)} = \mathbf{x}_i + \mathbf{n}_i$$

   where $\mathbf{n}_i$ is a noise vector satisfying $\|\mathbf{n}_i\|_2 \leqslant \epsilon$. Pick a one vs. all classifier that you have already trained (trained using clean training data) and test the performance of the classifier on the noise corrupted test data. Compare the error rate for different noise levels ϵ and determine the point where the classifier breaks down (i.e., it performs as bad as a classifier that gives random labels as output). How does the breakdown point of the feature-based classifiers change based on the choice of the non-linearity $g(.)$? Comment on your observations.

4. (**Bonus**) For a fixed input test data point, plot the output of the binary classifiers (without the sign non-linearity) and comment on how the corruption affects the output of your classifiers.

5. (**Bonus**) Repeat part (1) by varying the probability distributions used to generate the matrix $\mathbf{W}$ and $\mathbf{b}$.