Boğaziçi University
Department of Computer Engineering
Instructor: İnci M. Baytaş
Volunteering Teaching Assistant: Ege Onur Taga

Spring 2023
Assignment 1

CMPE 462 Assigment 1

The coding questions should be implemented in Python. You are not allowed to use built-in functions whenever a question asks you to implement a model from scratch. Jupyter Notebooks and relevant libraries, such as **numpy** and **matplotlib**, can be used.

You may merge your answers to coding and theoretical questions into a Jupyter Notebook with markdown cells. You should ensure mathematical expressions follow a consistent and clear notation. You are advised to use LaTeX. If you cannot use LaTex, then please make sure to use a proper math format. If you want to submit your code and solutions separately, you must provide .py files with instructions on reproducing the results for each question. If your answers cannot be reproduced, points will be deducted.

1. *Perceptron Learning Algorithm.* Consider the perceptron in two dimensions: $h(\mathbf{x}) = \text{sign}\left(\mathbf{w}^T\mathbf{x}\right)$ where $\mathbf{w} = [w_0, w_1, w_2]^T$ and $\mathbf{x} = [1, x_1, x_2]^T$. Technically, $\mathbf{x}$ has three coordinates, but we call this perceptron two-dimensional because the first coordinate is fixed at 1.

   (a) (5 points) Mathematically express the separating decision boundary as $x_2 = ax_1 + b$. What are the slope a and intercept b in terms of w_0, w_1, w_2?

   (b) (10 points) Let's create our own target function f and a linearly separable data set D. Consider $d = 2$ so that you can visualize the problem. First, choose a random line in the plane as your target function, where one side of the line maps to $+1$ and the other maps to -1. Choose the inputs $\mathbf{x}_i$ of the data set as random points in the plane, and evaluate the target function on each $\mathbf{x}_i$ to get the corresponding output y_i. Generate a dataset of size 20 following this procedure. Plot the examples $\{(\mathbf{x}_i, y_i)\}$ as well as the target function f on a plane. Please use different markers for the examples from different classes.

   (c) (10 points) Implement the perceptron learning algorithm from scracth and run it on the data set above. Report the number of updates that the algorithm takes before converging. Plot the examples $\{(\mathbf{x}_i, y_i)\}$, the target function f, and the final hypothesis g in the same figure. Comment on whether f is close to g.

   (d) (5 points) Repeat everything in (c) with another data set of size $1,000$ using the same data generation procedure. Compare your results with (c). Did you observe a difference in convergence? Discuss your observations and their reasons.

2. *Linear Regression*

   We will use the **3D Shapes** dataset from Google's DeepMind, which can be found here `https://github.com/deepmind/3d-shapes`. In the data folder provided to you, you will find 10,000 images under the **3dshapes-train** folder. The orientation values are included in the **orientations-train.npy** file. You need to use the numpy library to load this file which is simply an array of size 10,000. Each image contains a shape with a particular orientation, orientations can take 15 different values, linearly spaced between $[-30, 30]$. These will serve as our target labels.

   (a) (5 points) Convert the training images into grayscale, flatten each image into a vector of length 4096 (64x64) and load them into a numpy array. You will obtain a matrix of size 10,000x4096. You might need to use an additional library for this, e.g. **Pillow.**

   (b) (10 points) Please pose predicting the orientation of images as a linear regression problem. Implement and train a linear regression model from scratch. Compute the solution $w^\star$. You can also find a test set consisting of 1000 images under the **3dshapes-test** folder. Use the solution vector you obtained to make predictions on the test data. Report the Root Mean Square Error (MSE) between your predictions and target values in the **orientations-test.npy** file.

(c) (10 points) Using the entire image as the feature vector requires a large data matrix that complicates the computations. To alleviate this issue, review the feature extraction literature or design your own features to come up with a more compact representation per image. Apply the technique of your choice to obtain the feature vectors for each image and explain the reasons behind your choice. Train the linear regression model with your new features and report the MSE on the test features. Did you improve your previous results, or are the current results worse? Discuss the possible reasons. Do not forget to cite your references.

3. **Logistic Regression** Please download the files with handwritten digits, including only 1 and 5. You can use np.load() to load the following npy files; training data (train_data.npy), training labels (train_labels.npy), test data (test_data.npy), and test labels (test_labels.npy). Each row of train_data and test_data represents one data point. train_data should be a $1,561 \times 256$ matrix and test_data should be a 424×256 matrix. Each data point has 256 grayscale values between -1 and 1. The 256 pixels correspond to a 16×16 image. train_labels and test_labels are 1561 and 424 dimensional arrays, respectively. Label 1 is for digit 1, and label -1 is for digit 5.

   (a) (10 points) Extract the two features discussed in the class (symmetry and average intensity) to distinguish 1 and 5.

   (b) (5 points) Provide 2-D scatter plots of your features for training and test data (Now your data matrix will be N×2). For each data example, plot the two features with a red $\times$ if it is a 5 and a blue $\circ$ if it is a 1.

   (c) (15 points) Please write the expression of the gradient of the logistic loss. Implement logistic regression classifier from scratch. Give separate plots of the training and test data, together with the separators. Report train E_{in} and test E_{test} errors with train and test classification accuracies.

   (d) (15 points) Train a regularized logistic regression: $\min_{\mathbf{w}} E(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2$, where $E(\mathbf{w})$ is the logistic loss. Change your gradient descent algorithm accordingly and train a regularized model. This time, do not extract symmetry and average intensity features but use flattened images as your input vector. Report the best λ using cross-validation. Report train and test classification accuracies. Did regularization improve the performance in the case of using the raw images as inputs? Discuss.