AMAT 592 Assignment 2

- This assignment is done by MATLAB. **Compress all code files into a .zip file and submit through Blackboard.**
- **Due: May 6, 11:59 pm. Late homeworks will not be accepted.**

1. The handwritten digit dataset `mnist5k.mat` is modified from the original MNIST gray-scale image dataset, where samples of digit 9 belong to class 1 and otherwise class -1. It contains a training set `Xtr` with labels `Ytr` and a testing set `Xte` with labels `Yte`. There are 5000 sample images in both training and testing sets and each sample is stored a vector of 784 gray-scale pixel values between 0 and 255. We do binary classification using logistic regression:

$$\min_{\mathbf{w}, w_0} f(\mathbf{w}, w_0) := \frac{1}{n} \sum_{i=1}^{n} \log\left(1 + \exp((-y^{(i)}(\mathbf{w}^\top \mathbf{x}^{(i)} + w_0))\right) + \lambda \|\mathbf{w}\|^2 \qquad (1)$$

   (a) Visualize the first 9 training images in `Xtr` using MATLAB built-in function `imshow`. Dispaly the 9 images as a 3×3 tabular in the same figure using `subplot`. Note that you need to `reshape` each sample into a 28×28 matrix before visualization.

   (b) Write a function named as `logit.m` to implement gradient decent algorithm for the logistic regression problem (1) using **a constant step size** η. The input arguments of `logit` include `Xtr`, `Ytr`, `Yte`, `Yte`, the constant step size η, and the regularization parameter λ. The output of `logit` should be the training accuracy, test accuracy, and the objective value at each iteration (stored as a vector). You should adopt a proper stopping criterion for gradient decent implementation. Call your function `logit.m` by choosing proper values of η and λ. Note that your main file is supposed to be separated from `logit.m`.

   For this part, you need to:

      i. Print out the final training accuracy and test accuracy (A reasonable test accuracy should be $> 94\%$)

      ii. Plot the curve for objective value vs. iteration number. The x-axis should be iteration number t (starting from 1 to wherever the algorithm was terminated). The y-axis should be corresponding objective value $f(\mathbf{w}^t, w_0^t)$.

Hint: To make your MATLAB implementation fast, you should use matrix/vector operations whenever possible to avoid `for` loop.

2. In this problem, we use K-means clustering to compress RGB color image. Read the MATLAB built-in image `peppers.png` by the command

   `I = imread('peppers.png')`,

   which returns a 3-D matrix `I` of size $384 \times 512 \times 3$. The image has 384×512 pixels with each pixel having 3 values for the R(ed)G(reen)B(lue) channels respectively. Each pixel is viewed as a 3-D data point.

   Note that the data type of `I` is `uint8`. Make sure to convert the data type to float by `double(I)` before clustering. We cluster all the 384×512 data points using K-means and obtain k centroids $\mu_1, \ldots, \mu_k$. Then the original image can be compressed by replacing each pixel with the centroid of its cluster, so that compressed image only contains k different colors. The built-in function `kmeans` implements K-means++ by default. Set the argument '`MaxIter`' = 500 in `kmeans`.

   You need to visualize 3 compressed images for $k = 5, 20, 100$ as well as the original one. Make sure to convert the data type back to `uint8` before visualization. Display them as a 2×2 tabular in the same figure using `subplot` function, and `title` each subfigure with, e.g. '$k = 5$' or 'Original'.

3. In this problem, we use PCA to reduce the dimension of raw face images. Load the data `face.mat`, and we will have the variable `X` which is the data matrix of size 400×10304, where each row vector represents a gray-scale image originally of 112×92 pixels.

   First of all, center the data points (i.e., row vectors) in `X` by subtracting their mean `mu` from each row. Denote the preprocessed data matrix by variable `X_0`. Apply PCA to `X_0` and reduce data's dimension to $k = 350$. You can use the following command (taking the i-th image as an example) to recover the image:

   `Recon = X_0(i,:)*V_k*V_k'+ mu`,

   where `V_k= V(1:k,:)` contains the first k principal components. Recall the set of principal components can be computed via built-in function `svd` for SVD. Remember to `reshape` the vector `Recon` into a 112×92 matrix to show the image.

   Pick any image from `X`, and show the effect of PCA by comparing the original image and the recovered images side by side using `subplot`. Give an title to each subfigure.