

# Homework Assignment 8: Ensemble Learning + Clustering

**Due Monday, December 4, 2023 at 11:59 pm EST**

## Description

This assignments consists of two problems that cover some of the algorithms we studied during the semester. In the first problem, you can use any of built-in APIs or libraries. However, in the second problem, you **MUST** implement your own clustering algorithm as directed in the question.

## What to submit

Create a folder: `ps8_xxxx_LastName_FirstName` and add in your solutions (xxxx = python or MATLAB):

`ps8_xxxx_LastName_FirstName/`

- `input/` - input images, videos or other data supplied with the problem set
- `output/` - directory containing output images and other files your code generates
- `ps8.m` or `ps8.py` - code for completing each part, esp. function calls.
- `*.m` or `*.py` Matlab/python function files, or any utility code.
- `ps8_report.pdf` - a PDF file with all output images and text responses

Zip it as `ps8_xxxx_LastName_FirstName.zip`, and submit on Canvas.

## Guidelines

1. Include all the required images in the report to avoid penalty.
2. Include all the textual responses, outputs and data structure values (if asked) in the report.
3. Make sure you submit the correct (and working) version of the code.
4. Include your name and ID on the report.
5. **Comment your code appropriately.**
6. Please avoid late submission. Late submission is not acceptable.
7. Plagiarism is prohibited as outlined in the [Pitt Guidelines on Academic Integrity](#).
  - a. **Please don't share your codes with any of your colleagues.**

## Questions

### 1. Bagging and Handwritten-digits recognition

In this part, you are allowed to use any **MATLAB built-in function or Python machine learning Library**. **However, you're not allowed to use the classification learning or any code that is generated automatically or copied from anywhere.** In this problem, you will use a short sample of the MNIST dataset (HW8\_data1.mat) to build ensemble of classifiers to classify handwritten digits (i.e., 10-class classification problem). The file contains a features matrix  $X$  and labels vector  $y$ . These data can be read directly into your program by using the `load` command. Each row in  $X$  corresponds to one feature vector example. There are 5000 examples in HW8\_data1.mat, where each example is a 20 pixel by 20 pixel grayscale image of the digit. Each pixel is represented by a floating point number indicating the grayscale intensity at that location. The 20 by 20 grid of pixels is “**unrolled**” into a 400-dimensional vector. Each of these examples becomes a single row in our data matrix  $X$ . This gives us a 5000 by 400 matrix  $X$  where every row is a single feature vector for a handwritten digit image. The second part of the training set is a 5000-dimensional vector  $y$  that contains labels for the training set. To make things more compatible with Matlab indexing, where there is no zero index, we have mapped the digit zero to the value ten. Therefore, a “**0**” digit is labeled as “**10**”, while the digits “1” to “9” are labeled as “1” to “9” in their natural order.

- a. Download the mat file to your input directory. Load the data into matlab. Randomly pick 25 images and display them in a 5x5 grid. Save your output figure and include it in your report. Hint: the `resize` function could be useful.
- b. Randomly split the data into 4500 samples training set ( $X_{\text{train}}$  and  $y_{\text{train}}$ ) and 500 samples testing set ( $X_{\text{test}}$  and  $y_{\text{test}}$ ).
- c. Apply bagging on **the training set** to create five equally sized (900 samples each) subsets  $X_1, X_2, X_3, X_4$  and  $X_5$ ; don't forget to save the corresponding label vectors. The samples in  $X_i$  ( $i = 1, 2, 3, 4, 5$ ) should be **randomly** picked from the original training set that you obtained in part b. Please save these subsets to local files (in the input folder as .mat files) and include them along with your submission.
- d. Train a One-vs-One SVM 10-class classifier, with 3<sup>rd</sup> order polynomial kernel, using subset  $X_1$ . Use your trained classifier to:
  - i. Compute the classification error<sup>1</sup> on the training set  $X_1$ .
  - ii. Compute the classification error on the other training subsets, i.e.,  $X_2$  to  $X_5$ .
  - iii. Compute the classification error on the testing set.

---

<sup>1</sup> Error = 1 - Accuracy

- e. Train a KNN ( $K = 3$ ) classifier using subset  $X_2$ . Use your trained classifier to:
  - i. Compute the classification error on the training set  $X_2$ .
  - ii. Compute the classification error on the other training subsets.
  - iii. Compute the classification error on the testing set.
- f. Train a logistic regression classifier using subset  $X_3$ . Use your trained classifier to:
  - i. Compute the classification error on the training set  $X_3$ .
  - ii. Compute the classification error on the other training subsets.
  - iii. Compute the classification error on the testing set.
- g. Train a decision tree classifier using subset  $X_4$ . Use your trained classifier to:
  - i. Compute the classification error on the training set  $X_4$ .
  - ii. Compute the classification error on the other training subsets.
  - iii. Compute the classification error on the testing set.
- h. Train a random forest (with **60** trees) classifier using subset  $X_5$  (Hint: the `TreeBagger` function may be useful). Use your trained classifier to:
  - i. Compute the classification error on the training set  $X_5$ .
  - ii. Compute the classification error on the other training subsets, i.e.,  $X_1$  to  $X_4$ .
  - iii. Compute the classification error on the testing set.
- i. Use the majority voting rule to combine the output of your five classifiers and report the error rate on the testing set.
- j. Your report must document the following:
  - i. Summarize your results.
  - ii. Discussion and comparison of each classifier performance, and your interpretation of the results. Does bagging help?

## 2. K-means clustering and image segmentation

In this problem, you will **implement** the K-means algorithm. You will then use it to perform image clustering (segmentation). **Please note that you are not allowed to use any K-means or segmentation APIs**, instead you will need to implement your own K-means function.

- a. Write a function `[ids, means, ssd] = kmeans_single(X, K, iters)` to implement a basic version of K-means.

### Inputs:

- $X$  is the  $m \times n$  data matrix, where  $m$  is the number of samples and  $n$  is the dimensionality of each feature vector.

- $K$  is the number of clusters to output.
- `iters` is the number of iterations to run for K-means (update centers and assign points)

**Outputs:**

- `ids` is an  $m \times 1$  vector containing the data membership IDs (cluster id) of each sample (denoted by indices ranging from 1 to  $K$ , where  $K$  is the number of clusters).
- `means` is an  $K \times n$  matrix containing the centers/means of each cluster. For example, the first row should be the  $n$ -dimensional vector that contains the center of cluster#1.
- `ssd` is a scalar that gives the final SSD of the clustering. Remember, SSD is the sum of the squared distances between points and their assigned means, summed over all clusters.

To randomly initialize the means: Get the range of the feature space, separately for each feature dimension (compute max and min and take the difference) and use this to request random numbers in that range. Check the documentation for `rand`.

Once you have your initial centers, iterate over the following two steps. The first step is to compute the memberships for each data sample. Matlab's function `pdist2` can help you to efficiently compute distances. Then for each sample, find the min distance and the cluster that gives this min distance. The second step is to recompute the cluster means, simply taking the average across samples assigned to that cluster, for each feature dimension. Repeat for `iters` times.

Hint: You can create a small dataset of your own to test your function. Two dimensional data will be easier to visualize and debug.

- b. K-means is sensitive to the random choice of initial centers. To improve your odds of getting a good clustering, implement a wrapper function `[ids, means, ssd] = kmeans_multiple(X, K, iters, R)` to do  $R$  random initializations/**restarts**, and return the clustering with the lowest SSD.

**Inputs:** (same as `kmeans_single`)

- $R$  is a scalar denoting the number of restarts (different random assignments) to perform.

**Outputs:**

- Same as `kmeans_single`, **but** `ssd` is the lowest SSD across all random restarts.

- c. Now, you will apply clustering to segment and recolor four different images image.

- Download the attached images to your input directory and load them in matlab using `imread` function. This will return a  $H \times W \times 3$  matrix per image, where  $H$  and  $W$  denote height and width, and the image has three channels (R, G, B).
- Convert the images to double format (use the function `im2double`). To avoid a long run of your code, downsample the images (reduce their size) e.g. using `im = imresize(im, [100 100]);`

- To perform segmentation, you need a representation for every image pixel. We will use a three-dimensional feature representation for each pixel, consisting of the R, G and B values of each pixel. The command `X = reshape(im, H*W, 3);` helps to convert the 3D matrix (image) into a 2D matrix (data matrix) with pixels as the rows and channels (features) as the columns. Use the random restarts function (`kmeans_multiple`) you wrote above, to perform clustering over the pixels of the image. Note, you have to perform clustering on each image separately and independently from others.
  - Use your clustering output to recolor each pixel of each image according to their cluster membership. In particular, replace each pixel with the average R, G, B values for the cluster to which the pixel belongs (i.e. recolor using the cluster means). Show the recolored image using `imshow`, but convert it to format `uint8` before displaying.
  - For each image, experiment with different combinations for `K`, `iters`, and `R`. Please use the following values:
    - o `K = 3, 5, 7`
    - o `iters = 7, 13, 20`
    - o `R = 5, 15, 25`
  - **Hint1:** to streamline and modularize your implementation, you may need to implement the segmentation using a function:  

```
[im_out] = Segment_kmeans(im_in, K, iters, R)
```
  - **Hint2:** You may want to test your code on small part of the images before letting it run over the entire image. This will help you to verify your implementation without wasting long execution time.
- d. You report must document your observations, comments, and interpretation of the output. Include only a representative sample of your resultant images from all the combinations in your report. Save all the resultant images in the output folder.