Final Project

Due Thursday, April 23, 2020 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 the MATLAB built-in APIs. However, in the second problem, you **MUST** implement your own clustering algorithm as directed in the question.

What to submit

Create a folder: `fp_LastName_FirstName` and add in your solutions:

`fp_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
- fp.m - code for completing each part, esp. function calls; all functions themselves must be defined in individual function files with filename same as function name, as indicated.
- *.m Matlab/Octave function files (one function per file), or any utility code. It's good practice to end your functions with the clause "end"
- fp_LastName_FirstName_debugging.m – one m-file that has all of your codes from all the files you wrote for this assignment. It should be a concatenation of your main script and all of your functions in one file (simply copy all the codes and paste them in this file). In fact, this file in itself can be executed and you can regenerate all of your outputs using it.
- fp_report.pdf - a PDF file with all output images and text responses

Zip it as `fp_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 classification**
   In this problem, you will use the MNIST dataset (http://yann.lecun.com/exdb/mnist/) to build ensemble of classifiers to classify handwritten digits (i.e., 10-class classification problem). **In this part, you are allowed to use any built-in MATLAB library**.

   a. Download the dataset, understand its format. Randomly pick 25 training images and display them in a 5x5 grid.

   b. Apply bagging on the training set to create five equally sized (40,000 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 downloaded. Please save these subsets to local files (in the input folder as .mat files) and include them along with your submission.

   c. Train a One-vs-One SVM 10-class classifier using subset X_1. Use default parameters of the radio-basis function. Use your classifier:

      i. Compute the classification error on the training set X_1.
      ii. Compute the classification error on the MNIST testing set.

   d. Train a One-vs-all SVM 10-class classifier using subset X_2. Use default parameters of the radio-basis function. Use your classifier:

      i. Compute the classification error on the training set X_2.
      ii. Compute the classification error on the MNIST testing set.

   e. Train a three-layer (input, hidden, and output) neural network with a 500 hidden units and a cross-entropy loss function using subset X_3. Use your classifier:

      i. Compute the classification error on the training set X_3.
      ii. Compute the classification error on the MNIST testing set

   f. Train a three-layer (input, hidden, and output) neural network with a 750 hidden units and a cross-entropy loss function using subset X_4. Use your classifier:

      i. Compute the classification error on the training set X_4.
      ii. Compute the classification error on the MNIST testing set

   g. Train a decision tree classifier using subset X_5. Use your classifier:

      i. Compute the classification error on the training set X_5.
      ii. Compute the classification error on the MNIST testing set

h. Use the majority voting rule to combine the output of your five classifiers and report the error rate on the MNIST testing set. A testing sample should be assigned to class i if at least three of your classifiers assign it to class i.

i. 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 API**, 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 your 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 example. For example, the first row should be the n-dimensional vector that contains the center of cluster#1.
- `ssd` is a scaler that gives the final SDD 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 wil 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 and load them in matlab using `imread` function. This will return a HxWx3 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:
     - K = 2, 3, 5, 7
     - Iters = 7, 13, 20
     - R = 5, 15, 25

d. You report must document your observations, comments, and interpretation of the output. Include the resultant image for all the combinations in your report and the output folder.