Homework Assignment 5: Weighted KNN + PCA and Face Recognition

Due Saturday, October 21st, 2023 at 11:59 pm EST

Description and dataset instructions

This assignment is composed of three parts

Part 1: In class, we learned about weighted nearest neighbor. In the first part of this problem set, you will implement the weighted voting algorithm and study the effect of the bandwidth parameter.

Part 2: In lectures, we discussed PCA for dimensionality reduction and its application for face recognition. The basic idea is that the first few Eigenvectors (of the data covariance matrix) contain most of the variance of the data. So you can use them as a new basis vecors and project your data into this new basis space for dimensionality reduction. Data of small number of features make it easier for ML algorithms to generalize to new data and save computations cost. For this problem set, you will implement the necessary elements to compute the Eigenfaces and apply different classification algorithms to perform face recognition.

Part 3: case study.

What to submit

Download and unzip the `ps5` folder: ps5.zip
Rename it to `ps5_xxxx_LastName_FirstName` and add in your solutions (xxxx = matlab or python):

`ps5_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
- ps5.m or ps5.py- 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 or *.py Matlab/Python function files, or any utility code
- ps5_report.pdf - a PDF file with all output images and text responses

Zip it as `ps5_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.

Questions

1- **Weighted NN**: In this part you will implement and apply the Gaussian weighted neighbors classifier, using the equation in lecture slides, and apply it to some testing examples. For this example, you will need to use the training and testing samples in 'hw4_data3.mat' (from the previous assignment): X_train and y_train are the training example and the corresponding class labels. X_test are the testing features that you are required to predict a class for, and y_test are the ground truth labels that you can use to compute the accuracy.

a. Write a function, `y_predict = weightedKNN(X_train,y_train,X_test,sigma)` that uses all neighbors to make a prediction on the test set, but weighs them according to their distance to the test sample. Use the Euclidian distance as your distance metric. Hint: you may find the MATLAB function `pdist2(cdist` in python`)` useful.
   - `X_train` is an $m \times (n+1)$ features matix, where m is the number of training instances and n is the feature dimension,
   - `y_train` is an $m \times 1$ labels vector for the training instances,
   - `X_test` is an $d \times (n+1)$ feature matrix, where d is the number of test instances,
   - `sigma` is a scaler denoting the bandwidth of the Gaussian weighing function,
   - `y_predict` should be a $d \times 1$ vector that contains the predicted labels for the test instances.

   Function file: `weightedKNN.m` containing function `weightedKNN`

b. Test your function using the provided training matrix (X_train), training labels (y_train), and testing features matrix (X_test). Use the following values for sigma: 0.01, 0.05, 0.2, 1.5, 3.2, and 5. Compute the classification accuracy for each value of sigma.
   Output: a table that list the accuracy vs sigma
   Text output: comment on your results and the effect of sigma.

2- PCA and face recognition

2.0. Data Preprocessing

For part 2, we are going to use the AT&T labs face dataset (Download zip file). There are ten different images of each of 40 distinct subjects. The size of each image is 112x92 pixels, with 256 grey levels per pixel. The images are organized in 40 directories (one for each subject), which have names of the form sX, where X indicates the subject number, person's ID, between 1 and 40. In each of these directories, there are ten different images of that subject, which have names of the form i.pgm, where i is the image number for that subject (between 1 and 10).

Download the AT&T labs face dataset (Download zip file). Extract the dataset to the directory 'input/all' then write a function/script that randomly picks 8 images from each subject folder for training. Save these images under the directory 'input/train/'. You should get a total of 320 images in this directory (you shouldn't use the same file names; otherwise you may get conflicts. Chose

proper naming scheme). Those two images per subject that weren't selected for training should be saved under 'input/test/'. The directory 'input/test' should now contain 80 images.
Hint: A suggested name for each image is "PersonID_ImageNumber". This will help you to parse that file name and get the class label (i.e., person's ID) of each image; later in the assignment.
To make sure that you have handle on your data, read any of your training images, and display it along with the person's ID associated with that image.
Output: A face image as ps5-2-0.png

2.1. PCA analysis

In this part, you will implement the PCA algorithm on face images. You need to write a code to compute the mean image and solve for the eigenfaces.

a. Write the code to read all the images in the training directory. Reshape each image to be represented as one column vector. Construct a matrix T whose columns are the training images (each column is one folded image). The size of T should be 10304 x 320.

   Output: A gray level images showing the values of T `as ps5-1-a.png.` You can use this command in MATLAB: imshow(T,[]) to display matrix T as an image. Similar functions are available for python programmers
   `Hint: Due to the large difference in the dimensions of T, it's tricky to visualize T. You can zoom-in and add several screenshots from different regions in the image and include those screenshots in the report.`

b. Compute the average face vector m. The average face vector is the average of each pixel across all images. Using T, you would need to take the average across each row. Thus, it should be the 10304x1 mean vector computed across the column of T.

   Output: Resize m to 112x92 and display the resultant image (the mean face) as `ps5-2-1-b.png`

   Output (`textual response`):
   - Describe your results.

c. Find the centered data matrix, $A = T - m$, i.e., you subtract the mean vector from each column of T (remember each column in T corresponds to one image). Then define the data covariance matrix $C = AA^T$. Remember the dimension of C has to be 10304x10304.

   Output:
   - image of the covariance matrix as `ps5-2-1-c.png. In you report, use the same approach you used in part a to display different snippets.`

d. Use the Matlab/Python function '`eig`' to compute **only the eigenvalues of $\boldsymbol{A^T A}$**. This will give you 320 values. Recall that each eigenvalue (λ) represent how much variance is retained by the corresponding eigenvector and that the first eigenvector captures the maximum variance. Now, we are going to learn how to decide on how many eigenfaces (eigenvectors) are enough to represent the variance in our training set. To do so, we define the following

$$v(k) = \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{N} \lambda_i}, k = 1, 2, \cdots, N;\ and\ \lambda_m > \lambda_n\ \forall\ n > m$$

where N is the total number of eigenvalues (in our case, the total number of training images). $v(k)$ is the percentage of variance captured by the first k eigenvectors. Compute the values of $v(k)$ and determine the minimum number of eigenvector, K, needed to capture at least 95% of the variance in the training data (i.e., the minimum value of k such that $v(k) \geq 0.95$).
Hint: Before computing $v(k)$, you'll need to sort the eigenvalues in descending fashion, we need to keep those large values, not the residual ones!

Output: The plot of k vs $v(k)$ as `ps5-2-1-d.png`

Output (`textual response`):
- The number of eigenvectors, K, that capture 95% of the variance in the training data

e. Using the value of K you obtained from the previous question, retrieve the K dominant eigenvectors corresponding to the heights K eigenvalues of the covariance matrix C. Save those dominant eigenvectors in a basis matrix U. Those vectors will be used as the basis for PCA dimensionality reduction (i.e., each image will be represented as a weighted sum of those vectors). Now, U defines the reduced eigenface space.
Hint: The MATLAB/Scipy '`eigs`' function is very useful in this regard!

Output: image of the first 9 eigenfaces (you will need to resize the eigenvectors) in one figure as `ps5-2-1-e.png`

Output (`textual response`):
- The dimensions of matrix U
- Describe your results and comment on the eigenfaces you obtained.

2.2. Feature extraction for face recognition

For us to use different classification techniques, we need to extract some features. We could use the 10304 pixel values/image as features of that image, but this is too much. For face recognition, we are going to use the image representation in the reduced eigenface space as our feature vector. Any image can now be represented in the new space as $I = m + w_1 u_1 + w_2 u_2 + \cdots + w_K u_K$, where m is the mean vector from 2.1.b, u_i is the i^{th} column of the basis matrix U, and $w = [w_1, \cdots, w_K]^T$ is the reduced representation of the image I in the reduced eigenface space. Compare the size of the image vector to the size of vector w. Definitely, there is a great deal of dimensionality reduction. As

matrix multiplication, for **one image** I, w can be computed as $w = U^T(I - m)$, remember, I and m are now vectors, not $2D$ matrices.

a. Project all the images in the training folder in the new reduced eigenface space, i.e., find w for each image in the training folder. Construct a matrix W_training where each row in it corresponds to one reduced training image. W_training is the training features matrix. **Hint**, keep track of which subject corresponds to which row in W_training, this will define your **labels vector**, i.e., y_train (you will need that later to train a classifier).

b. Project all the images in the testing folder in the new reduced eigenface space, i.e., find w for each image in the testing folder. Construct a matrix W_testing where each row in it corresponds to one reduced testing image. W_testing is the testing features matrix. **Hint**, keep track of which subject corresponds to which row, this will define your **true_class vector**, i.e., y_test (you will need that later to compute the accuracy of your classifier).

Output (`textual response`):
- The dimensions of W_training and W_testing

2.3. Face recognition

Next, you'll use the training features to train KNN and SVM classifiers and test the resultant classifier using the testing features. For this section, you can use available packages for K-NN and SVM.

a. Train a KNN classifier (you can use built-in functions or libraries) using W_training matrix and the associated labels vector. Test your classifier using samples in W_testing. Use K = 1, 3, 5, 7, 9, and 11 nearest neighbors. Compare the output of the classifier to the true_class labels and compute the classifier accuracy. Accuracy is defined as the number of correctly classified samples divided over the total number of testing samples.

Output (`textual response`):
- Table for your KNN classifier accuracy at the different values K listed above.
- comment on and discuss your results.

b. Use W_training matrix and the associated labels vector (y_train), train six SVM classifiers (you can use built-in functions or libraries). Each classifier must use a different combination of multi-class classification paradigm (one-vs-one and one-vs-all) and kernel (linear, 3rd order polynomial, and Gaussian rbf kernels) from the others.
Make sure to set the correct option for both one-vs-one and one-vs-all. Also, play with the normalization and regularization parameters. Use the default sigma for the rbf kernel.
Keep track of the training time of each classifier model. Compare the output of the classifier to the true_class and compute the accuracy of each classifier.

Output (`textual response`):

1- Table listing the training time for each SVM classifier model.

2- Table listing the testing accuracy of the each SVM classifier model. (You can use tables similar to the ones below for reporting the training time and accuracy)

3- Comment on and discuss your results and compare the performance of different kernels and multi-class classification paradigms.

4- Compare between the performance of KNN and SVM classifiers

Table 2.3.b.1. Training time for each model

	One-vs-one	One-vs-all
Linear		
Polynomial		
RBF		

Table 2.3.b.2. Testing time for each model

	One-vs-one	One-vs-all
Linear		
Polynomial		
RBF		

3- **Case study**: The Biden administration is convinced that the future of vehicular transportation is electric. Accordingly, it is placing a big bet on electric vehicle development and technology. The administration hopes that half of all cars sold by 2030 will be electric. As part of that effort, in late 2022, President Biden used a visit to the Detroit Auto Show to announce that the White House was releasing the first round of funding for a nationwide EV charging network, CNBC reports. The first tranche totals $900 million and will be spent in 35 states to help construct chargers across roughly 53,000 miles of highway.

As a machine-learning engineer, we ask you to develop algorithm to help decision makers on where to place those chargers. **What features would you use for your model? Please explain**