Homework 4

Due April 14, 2013 8pm Chicago time

Computer problem: Detecting Alzheimer's disease from brain images

- In this homework, you will solve a problem that pertains to the cutting-edge problem of detecting Alzheimer's disease (AD). AD is a severe brain disease experienced by elderly people. In this disease, the patient experiences loss of memory, and major changes in speaking, personality, and ability to function generally.
- I am providing you with images that were taken using an imaging method called PET, which measures glucose metabolism (utilization of sugar) within the brain. In AD patients, key brain regions stop function normally, and thus stop consuming resources at the normal rate.
- I am providing you with images of 42 different patients in the file 'DATA.mat'. Some of these patients have Alzheimer's and some are normal. Your goal is to train a classifier to tell them apart.
- Each brain image was originally a 3D array of pixel values (128x128x91), where each pixel represents the rate of glucose metabolism. To make it a little easier for you, the data have been reformatted into a data matrix X with dimensions 42 (patients) x 1,490,944 (pixels).
- The corresponding class labels are in array y ('AD'=Alzheimer's, 'NL'=normal).
- Many of the pixels are uninteresting to us (e.g., pixels outside the brain), so I am also providing you with a "mask", a binary image that tells which pixels can be ignored. A value of 1 means a pixel should be included, 0 means it should be excluded.

a) Apply the **mask** (mask_matrix) to the images to create feature vectors of lower dimension. Note that each of the 42 images in X has already been transformed from 128x128x91 to 1x1,490,944 (each row in X), while mask_matrix is still in the original format (i.e., you will have to first transform mask_matrix and then apply it to X).

b) **Center and scale (i.e., standardize) X**, i.e., subtract column mean from each column and divide each column by its standard deviation.

c) Compute the **PCA** basis vectors and eigenvalues using the **SVD** method.
Tip: To avoid 'out of memory' errors (which you might encounter because of the size of X), use the "economy size" decomposition:
```
[U,S,V] = svd(X,'econ');
```
If X is mxn with m< n, only the first m columns of V are computed and S is mxm.

d) Compute the **principal component scores** for the whole data set in X, i.e., the representation of X in the principal component space.

e) Verify that the **eigenvalues** are in descending order, convert them to %VAF values, then graph them.

f) Keep only the first 2 components: now X will be 42x2. Make a scatter plot of the principal components. To make the **scatter plot** you can use the following command:
```
gscatter(SCORES(:,1),SCORES(:,2),y);
```
where SCORES is the representation of X in the principal component space and y is the label vector.

g) Train a **linear discriminant classifier** on the entire dataset and test it on the same data (training set = testing set). (Remember, this is a bad idea, but we're doing it to see what happens). Use the ML decision threshold (same as MAP with equal prior probabilities).

h) Compute the following **performance indexes** of the classifier:

- **Confusion Matrix**: it is a 2x2 matrix where the rows show the predicted outcomes and the columns are the true labels:
  ConfusionMatrix = [TP FP; FN TN];
  where
  TP = True Positive, means number of cases where an AD patient was classified correctly;
  TN = True Negative, means number of cases where a normal sample classified correctly;
  FP = False Positive, means number of cases where a normal patient classified as AD;
  FN = False Negative, means number of cases where an AD patient classified as normal;
- **Misclassification rate** = (FP+FN)/(TP+TN+FP+FN);
- **True Positive Fraction (Detection Probability)** = TP/(TP+FN)
- **False Positive Fraction** (**False Alarm Rate**) = FP/(FP + TN).

i) Now train and test **a linear discriminant classifier** within a **two-fold cross validation**. Remember that in two-fold cross validation you have to split the entire dataset into two subsets S1 and S2; during the first iteration S1 will be used as training set and S2 as test set, whereas during the second iteration S2 will be used as training set and S1 as test set.

j) Compute the **average performance** of the classifier, i.e., the mean and the standard deviation of each performance index computed in part i.

k) **Compare performance results** obtained from the classifiers in parts g and i. What do you observe? Is this what you expect?

l) **Plot a 2D scatter plot of all the data in the PCA domain, along with the decision boundary**.

m) In this problem **the discriminant vector** in the PCA domain corresponds to a vector in the original space that is a brain image (and can be displayed like an image). To make the brain image version of the dicriminant vector:
   1. Use the discriminant vector from part g.
   2. Transform the discriminant vector back into image space. The transformation is performed as follows:
      ```
      EigImg = Eivec2*DV;
      ```
      Where Eivec2 are the eigenvectors associated with the 2 largest eigenvalues and DV is the discriminant vector

3. Use the mask matrix to map the pixels in the vector `EigImg` back to the original image dimension (i.e., the dimension of each brain image before applying the mask). To do so:
   i. transform the mask matrix into a vector. The size of this vector should be 1,490,944 x 1:
   ```
   mask_vec = mask_matrix(:)';
   ```
   ii. Create a vector of the same dimension of the vectorized mask (i.e., 1,490,944 x 1) of all NaN:
   ```
   Img_vec = repmat(nan,1,length(mask_vec));
   ```
   Where mask_vec is the vectorized mask matrix
   iii. Mask_vec has a total number of '1' equal to the length of Img_vec; find the indexes where mask_vec == 1 (i.e., pixels to keep) and substitute the NaN values in Img_vec at those indexes with the values of EigImg:
   ```
   idx = find(mask_vec == 1);
   Img_vec(idx) = EigImg;
   ```
   iv. Reshape Img_vec to the original brain images size (128x128x91). You can use the Matlab command reshape to do so.
   v. Display some slices of the image (e.g., slice 40). You can use the command imagesc:
   ```
   imagesc(IMG(:,:,40));
   ```
   where IMG is the reshaped Img_vec of size 128 x 128x 91.
   vi. Tip (optional): if you want to see all the 91 slices, make a loop for i = 1:Nslices (wher Nslices is 91) and at each iteration display slice(i) (i.e., imagesc(IMG(:,:,i))), pausing for about 0.25 seconds between each iteration.
   vii. Explain in words the meaning of what the images represent.

n) Even in part i where we used cross validation, we were still cheating (we used information about the test set during training). Can you spot the mistake? I designed the assignment this way to make it easier for you, but in real life you must avoid cheating!