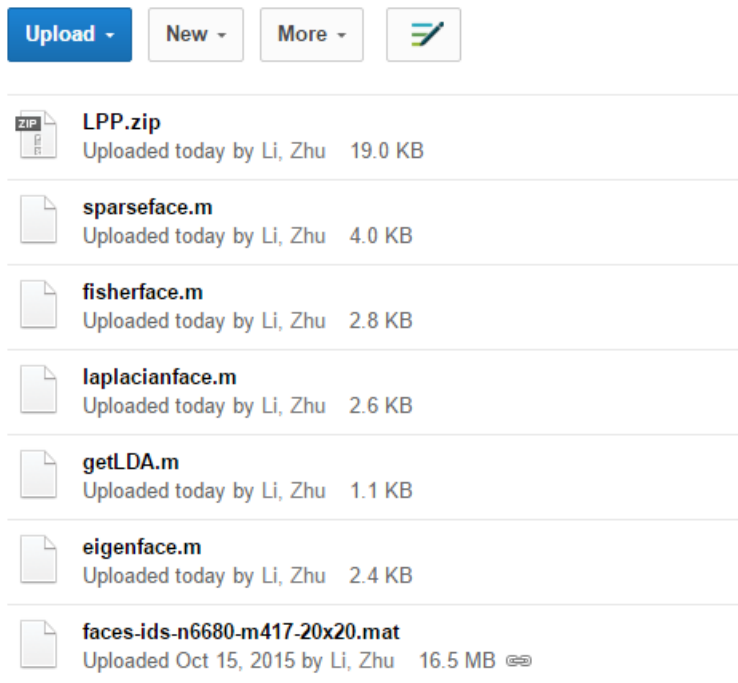


Homework # 3: Subspace Methods in Image Recognition

Description:

The objective of this homework is to consolidate and practice the knowledge and skills we learn on component analysis and subspace learning, especially the PCA, LDA and Graph Embedding. We will do an exercise based on a small face data set, which can be downloaded from:

<https://umkc.box.com/s/p4levwprgins5ctg7fne9stwu7twkpza>



A face data set of 6680 images from 417 subjects, are provided, you should include a special group of 1 subject 10 images from any celebrity on web, to augment the data set.

The basic functions for PCA/LDA/Graph Embedding (LPP) are also provided, as in eigenface.m, fisherface.m, laplacianface.m, and the zipped folder LPP which offers Laplacian embedding solution LPP.m.

The input images are of size 20x20 pels, and students are required to compute the PCA/Eigenface model A_0 : 400 x d_0 , and LDA/Fisherface model on top of the Eigenface model, A_1 : d_0 x d_1 , as well as a Laplacian embedding model, A_2 : d_0 x d_2 .

Then project the face images to these three models, and evaluate the performance as the mean average precision as discussed in the class. Notice that we will have total $m=417+1=418$ subjects, and we need to separate the training from the testing. For each class/subject, we can leave 1 image out for testing, and use the rest for training.

- Also called “average precision at seen relevant documents”
- Determine precision at each point when a new relevant document gets retrieved
- Use $P=0$ for each relevant document that was not retrieved
- Determine average for each query, then average over queries

$$MAP = \frac{1}{N} \sum_{j=1}^N \frac{1}{Q_j} \sum_{i=1}^{Q_j} P(doc_i)$$

with:

Q_j number of relevant documents for query j
 N number of queries
 $P(doc_i)$ precision at i th relevant document

Hint: you just need to compute different index from faces and ids data, to separate training and querying, e.g. .

```
%loading faces, ids, of nx400 and nx1
load faces-ids-n6680-m417-20x20.mat;
n=length(ids);
uid = unique(ids); m = length(uid);
% query data index:
q_idx = zeros(1, m);
for k=1:m
    offs = find(ids==uid(k)); q_idx(k) = offs(1);
end

%training data index
train_idx = setdiff([1, n], q_idx);
```

Deliverable and Grading:

1. Extra data set: pull 10 images from 1 person somewhere (can be your own face images), normalize the images as 20x20 thumbnails, dynamic range scaling to [0, 1], and append it into the face/ids data set, give it id 999. [%10]

10 images of Einstein are included in zip, inside the ExtraDataSet folder. The following is the code to append them:

```

%% Q1
%(COMPLETE)
data = load('faces-ids-n6680-m417-20x20.mat');
folder = sprintf('ExtraDataSet');
images = dir([folder '\*.jpg']);

for j=1:10
    image = imread([folder '\' images(j).name]);
    resized = imresize(image, [20, 20]);
    rescaled = rescale(resized);
    gray = rgb2gray(rescaled);
    extra_data = gray(:);
    data.faces = [data.faces; extra_data];
    data.ids = [data.ids; 999];
end

```

- Matlab/Python Code and plots: function `[A0, eigv]=getEigenfacemodel(faces)` , plot the Eigenface basis, and corresponding eigen values, explain if you keep 8 dimensions, how much information/energy is lost ? [15%]

I created the following function with a flag as a last parameter to avoid plotting anything for question 5.

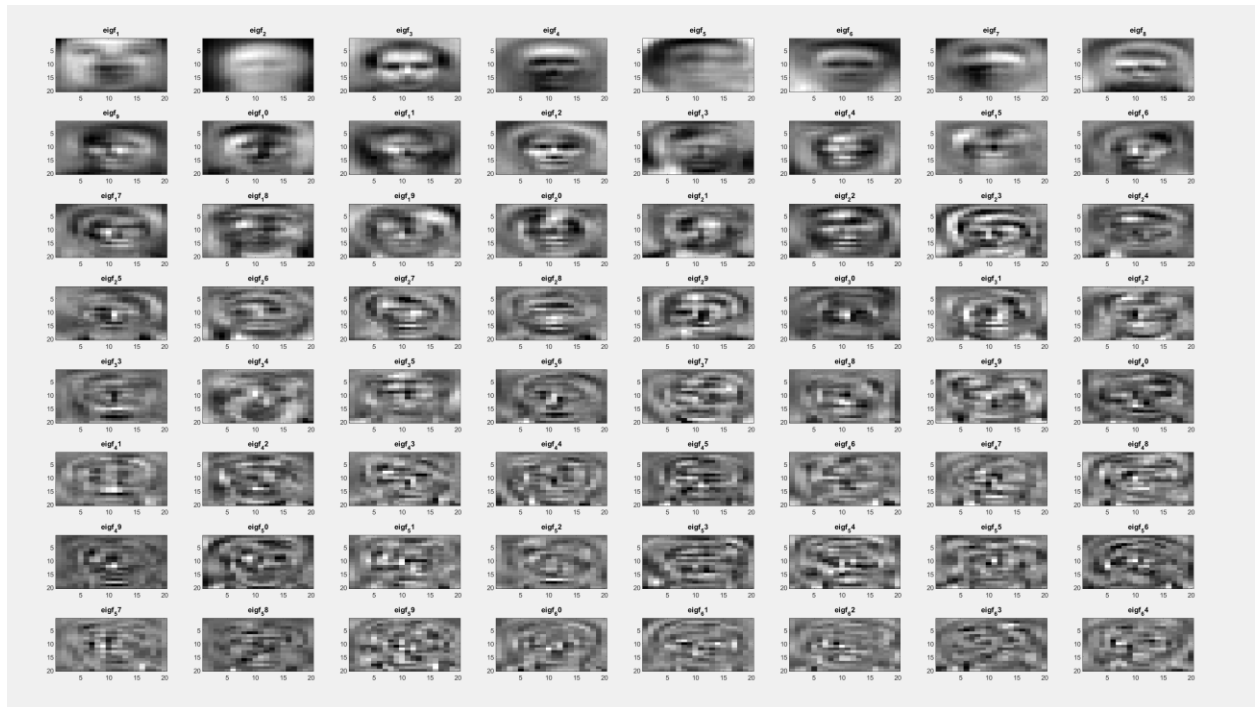
```

Editor - D:\MatlabHomework\HW-3_Subspace Models\getEigenfacemodel.m
getEigenfacemodel.m  getFisherfacemodel.m  getLaplacianfacemodel.m  main.m  getQueryMAPLaplacian.m  getQueryMAPfis

1  function [A0, eigv] = getEigenfacemodel(faces, plotting)
2  [A1, ~, lat] = pca(faces);
3  kd = 64;
4  A0 = A1(:, 1:kd);
5  eigv = lat(1:kd);
6  %eigen values
7  if plotting == 1
8      figure(11);
9      subplot(1,2,1); grid on; hold on; stem(lat, '.');
10     f_eng=lat.*lat;
11     subplot(1,2,2); grid on; hold on;
12     plot(cumsum(f_eng)/sum(f_eng), '-');
13     figure(12);
14     for k=1:kd
15         subplot(8,8,k);
16         colormap('gray'); imagesc(reshape(A1(:,k), [20, 20]));
17         title(sprintf('eigf_%d', k));
18     end
19     figure(13)
20     plot(real(lat), imag(lat), 'r*')
21     xlabel('Real')
22     ylabel('Imaginary')
23     t1 = ['Eigenvalues'];
24     title(t1)
25     display(["information/energy lost: " sum(lat(kd:end))]);
26 end
27 end
28
29

```

The following are eigenface basis with 64 dimensions, as I need it for the 5th question:



The energy loss for 64 dimensions is:

1×2 string array

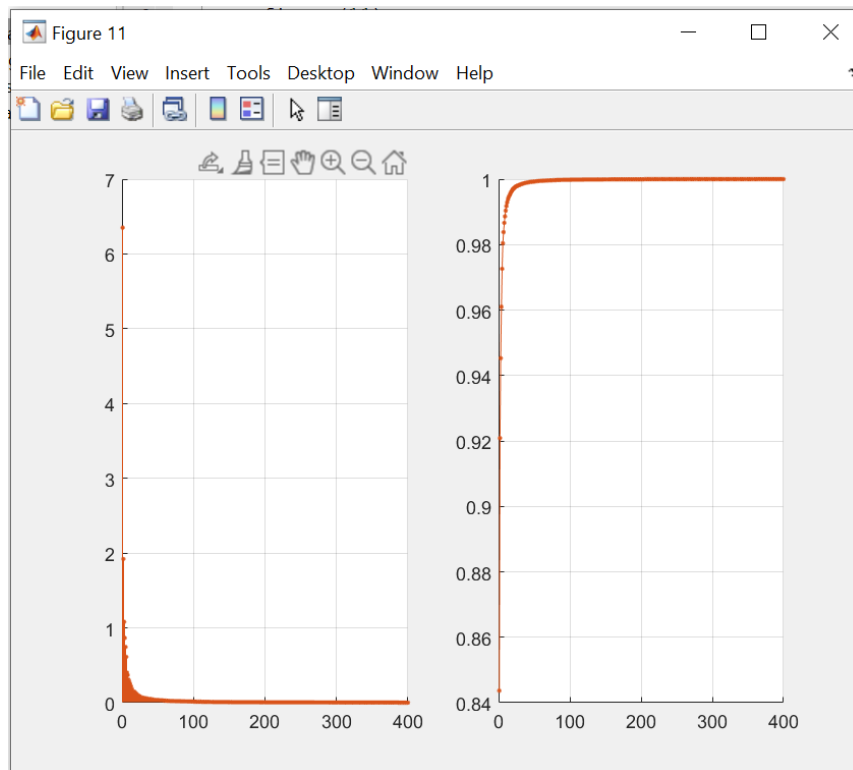
"information/energy lost: " "2.043"

The energy loss for 8 dimensions is:

1×2 string array

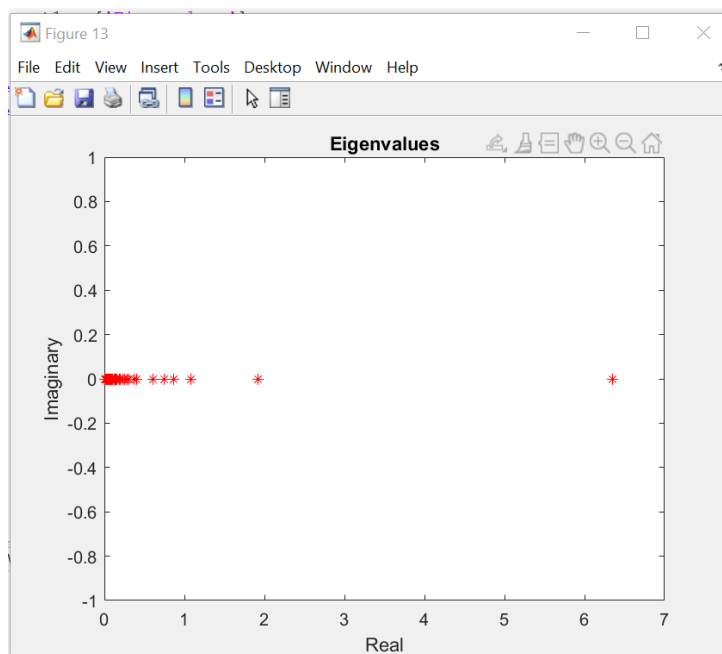
"information/energy lost: " "6.8686"

The following is the energy graph:



This is obvious since we analyze less dimensions our energy loss is greater but the speed is also faster.

Eigen values plot is the following:



3. Matlab/Python Code and plots: function [A1]=getFisherfacemodel(faces, A0, ids) , compute Fisherface model after Eigenface projection (A0), to avoid singularity in covariance matrices, plot Fisherface models [20%]

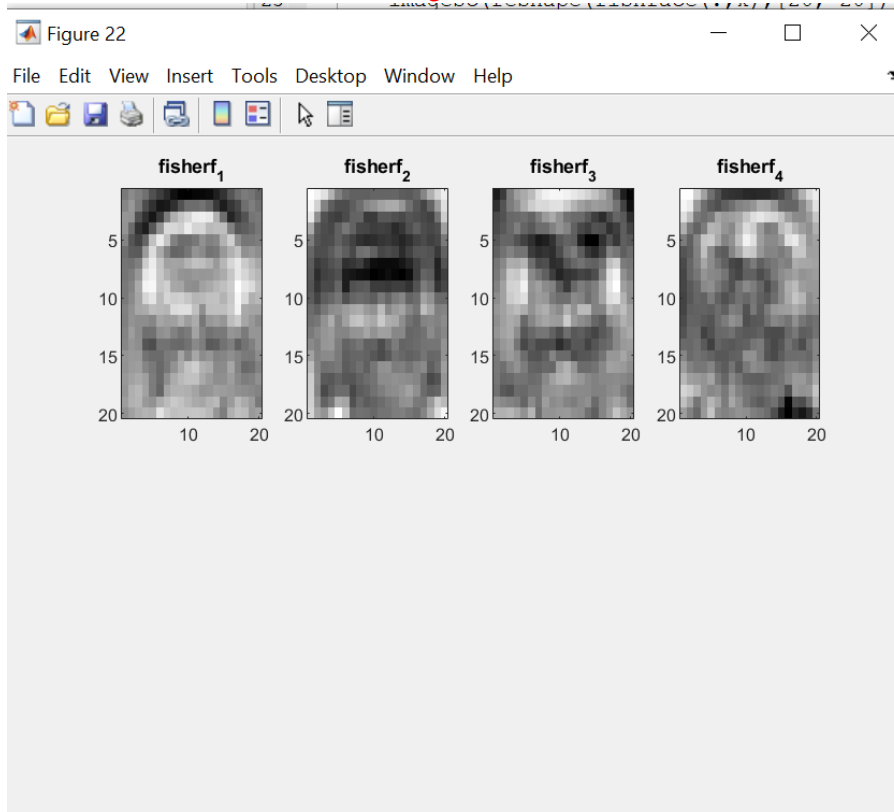
Similarly, to the previous one, I used a flag to avoid plotting in number 5. The following is the code:

```

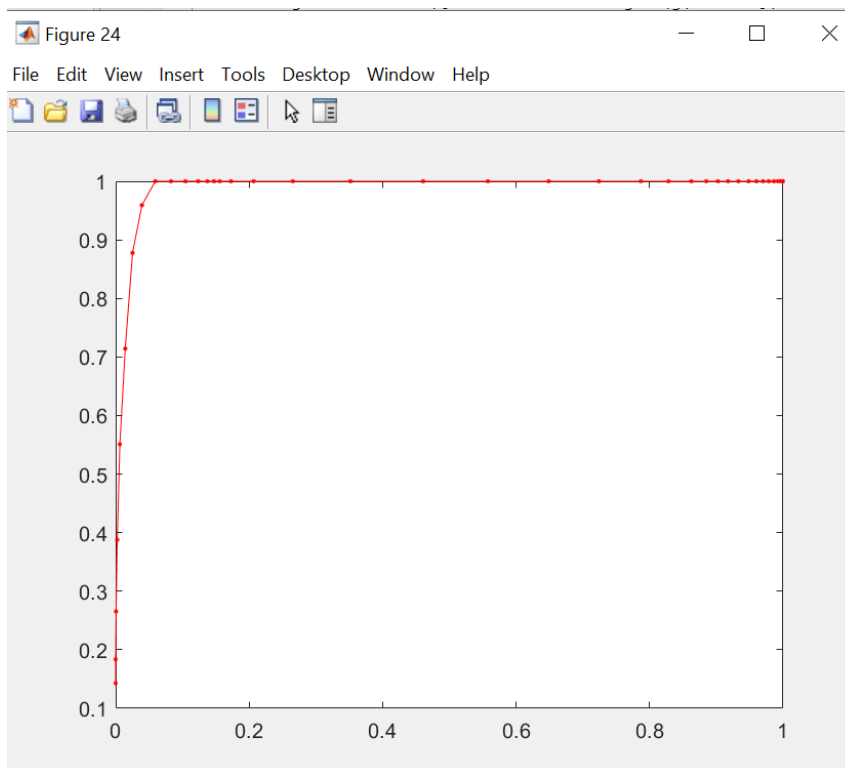
Editor - D:\MatlabHomework\HW-3_Subspace Models\getFisherfacemodel.m
+2  getEigenfacemodel.m  getFisherfacemodel.m  getLaplacianfacemodel.m  main.m  getQueryMAPLaplacian.m  getQueryMAPFisherFace.m  +
1  function [A1] = getFisherfacemodel(faces, A0, ids, kd, plotting)
2      n_face = length(faces);
3      %kd = 64;
4      [A1, ~]=getLDA(faces(1:n_face,:), A0(:,1:kd), ids(1:n_face));
5      if plotting == 1
6          x1 = faces*A0(:,1:kd);
7          x2 = faces*A0(:,1:kd)*A1;
8          %disp(size(ids(1:n_face)));
9
10         f_dist2 = pdist2(x2(1:7,:), x2);
11         %avoiding singularity
12         eigface = eye(400)*A0(:,1:kd);
13         fishface = eye(400)*A0(:,1:kd)*A1;
14         for k=1:4
15             figure(21);
16             subplot(2,4,k); imagesc(reshape(eigface(:,k), [20, 20])); colormap('gray');
17             title(sprintf('Eigf_ %d', k));
18             subplot(2,4,k+4); imagesc(reshape(fishface(:,k), [20, 20])); colormap('gray');
19             title(sprintf('Fisherf_ %d', k));
20         end
21
22         for k=1:4
23             figure(22);
24             subplot(2,4,k);
25             imagesc(reshape(fishface(:,k), [20, 20]));
26             colormap('gray');
27             title(sprintf('fisherf_ %d', k));
28         end
29         figure(23);
30
31         d0 = f_dist2(1:7,1:7); d1=f_dist2(1:7, 8:end);
32         [tp, fp, tn, fn]= getPrecisionRecall(d0(:), d1(:), 40);
33         figure(24);
34
35         plot(fp./(tn+fp), tp./(tp+fn), '-r', 'DisplayName', 'fisher kd=64');
36
37     end
38 end
39

```

Fisher Face models are the following:



Please see the precision graph for fisher face:

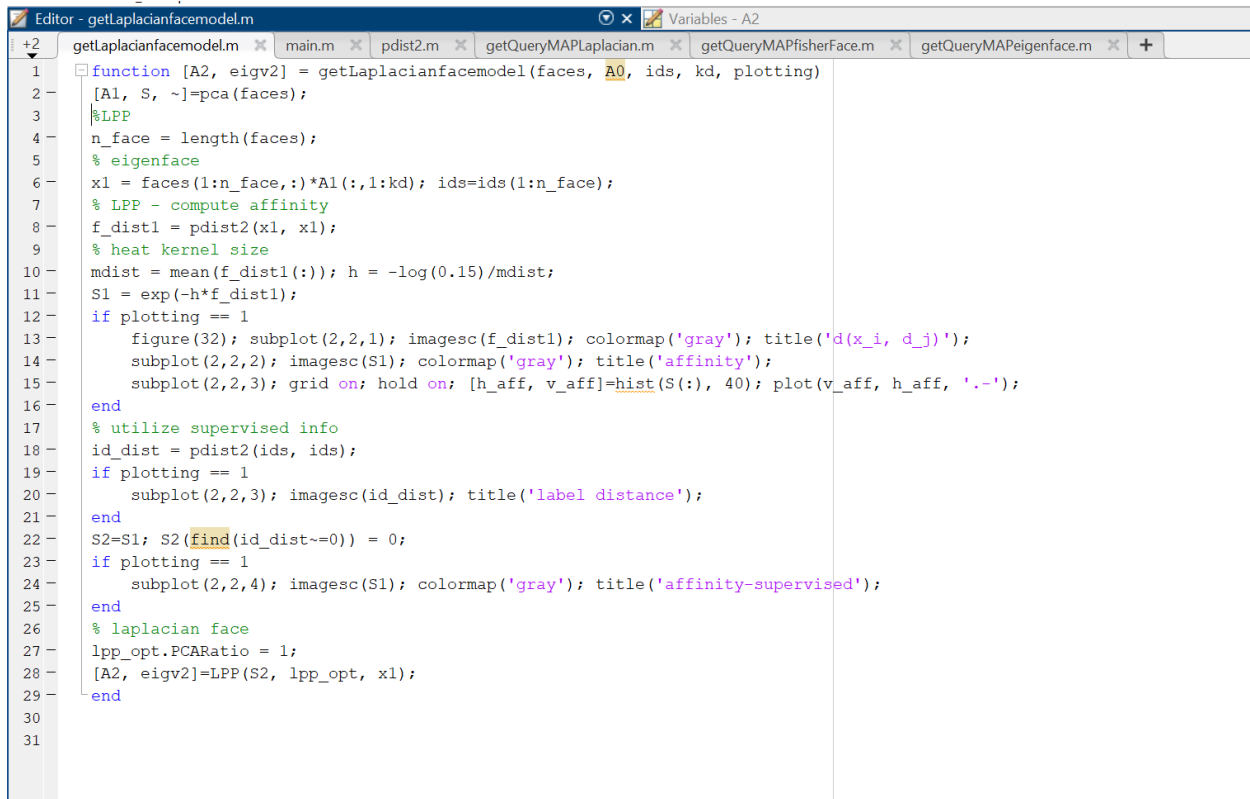


4. Matlab/Python Code and plots: function [A2, S]=getLaplacianfacemodel(faces, A0, ids) , compute Graph Laplacian embedding model after Eigenface projection (A0), output the affinity matrix you computed as S, and plot the affinity matrix, plot the Laplacian face models, justify your choices of parameters in computing affinity matrix. [25%]

Similarly to the previous 2 questions, I used a flag to avoid any plotting in question #5 as this function is going to be utilized there. The following is the code I used to compute the Graph Laplacian embedding model. The heat was set as follows since it showed the best readability:

$$h = -\log(0.15)/\text{mdist}$$

I also used pdist2 to compute the distance between the same train and test datasets.

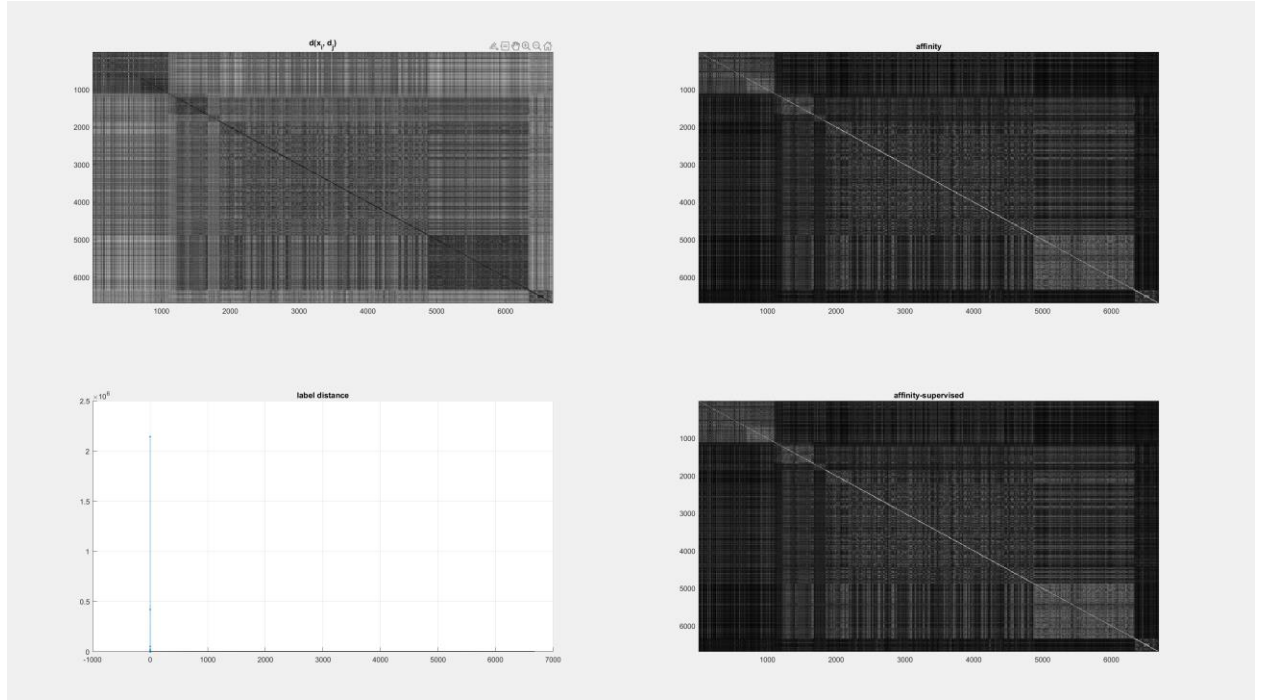


```

Editor - getLaplacianfacemodel.m
+2
getLaplacianfacemodel.m x main.m x pdist2.m x getQueryMAPLaplacian.m x getQueryMAPfisherFace.m x getQueryMAPeigenface.m x
1 function [A2, eigv2] = getLaplacianfacemodel(faces, A0, ids, kd, plotting)
2 [A1, S, ~]=pca(faces);
3 %LPP
4 n_face = length(faces);
5 % eigenface
6 x1 = faces(1:n_face,:)*A1(:,1:kd); ids=ids(1:n_face);
7 % LPP - compute affinity
8 f_dist1 = pdist2(x1, x1);
9 % heat kernel size
10 mdist = mean(f_dist1(:)); h = -log(0.15)/mdist;
11 S1 = exp(-h*f_dist1);
12 if plotting == 1
13     figure(32); subplot(2,2,1); imagesc(f_dist1); colormap('gray'); title('d(x_i, d_j)');
14     subplot(2,2,2); imagesc(S1); colormap('gray'); title('affinity');
15     subplot(2,2,3); grid on; hold on; [h_aff, v_aff]=hist(S(:), 40); plot(v_aff, h_aff, '-');
16 end
17 % utilize supervised info
18 id_dist = pdist2(ids, ids);
19 if plotting == 1
20     subplot(2,2,3); imagesc(id_dist); title('label distance');
21 end
22 S2=S1; S2(find(id_dist~=0)) = 0;
23 if plotting == 1
24     subplot(2,2,4); imagesc(S1); colormap('gray'); title('affinity-supervised');
25 end
26 % laplacian face
27 lpp_opt.PCARatio = 1;
28 [A2, eigv2]=LPP(S2, lpp_opt, x1);
29 end
30
31

```

The following are the affinity matrices for both supervised and not supervised as well as label distance.



5. Matlab/Python code and plots: `[map]=getQueryMAP(q, retrv_ids)`, the function to compute the average precision from a single query, test this against your own data set (should consists at least 2000 face images), and then compute mAP for the new subjects you provided with id=999 . Please plot MAP as function of different choices of, eigenface model dimension of $d_0 = [32, 64]$, and of fisherface/laplacian face dimension of $d_1/d_2 = [8, 16, 24]$. Also indicate your choice of heat kernel size and thresholding in affinity modeling for Laplacian. [%30]

For this question, I utilized all the functions I created in the previous questions. I also create 3 different functions for finding Mean Average Precision for: eigenface, fisherface, and Laplacian. They all can be found in the zipped folder, however, here I will include only 1 function. I used the following parameter for kernel size and thresholding: `lpp_opt.PCARatio = 1`; as I found it gives the most performance.

The following function was used to find Laplacian Mean Average Precision:

```

Editor - getQueryMAPLaplacian.m
+1 getFisherfacemodel.m getLaplacianfacemodel.m main.m getQueryMAPLaplacian.m getQueryMAPFisherFace.m getQueryMAPEigenface.m
1 function [meanAveragePrecision]=getQueryMAPLaplacian(faces, trainIndexes, q, retrievedIDs, kd)
2 trainFace = faces(setdiff(1:size(faces), q), :);
3 [A0, ~]=getEigenfacemodel(trainFace,0);
4 [A2, S] = getLaplacianfacemodel(trainFace, A0, trainIndexes, kd,0);
5 x3 = trainFace*A0(:,1:kd)*A2;
6 for k=1:length(q)
7     dist = pdist2(faces(q(k),:)*A0(:,1:kd)*A2, x3);
8     [~, offsets]=sort(dist);
9     tp = length(find(trainIndexes(offsets(1,1:6))~=retrievedIDs(k))==1);
10    averagePrecision(k) = tp/6;
11 end
12 meanAveragePrecision = mean(averagePrecision);
13 end
14
15

```

And that's how I call this function and plot the results for various dimensions:

```

dimensions = [8, 16, 24];
for i=1:length(dimensions)
    for j=1:length(indexOfQuery)
        meanAveragePrecision(i, j) = getQueryMAPLaplacian(data.faces, trainIndexes(j), indexOfQuery(j), retrievedIDs, dimensions(i));
    end
end
dimensionsForPlotting = 1:1:6;
figure(3);
plot(dimensionsForPlotting, meanAveragePrecision(1,:), '-.k', dimensionsForPlotting, meanAveragePrecision(2,:), ':b', dimensionsForPlotting, meanAveragePrecision(3,:), '-r');
legend('8', '16', '24');
axis([0 6 0 1]);
xlabel('Q');
ylabel('Mean Average Precision');
title('Laplacian');

```

I will save the readers time by not including all the code for other two models, please run my code. I made it so you could run each section consequently to get the results. Meanwhile, the following are the graphs of Mean Average Precision for each technique and for each dimensions mentioned in the question:

