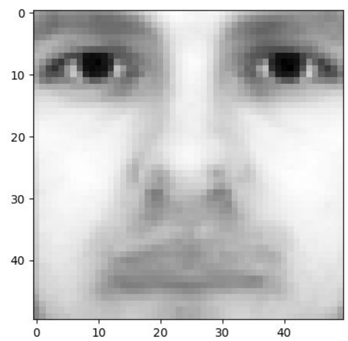
CS 5785 Applied Machine Learning
Assignment 2

Kelly Wang
and Yian Mo
09/23/2017

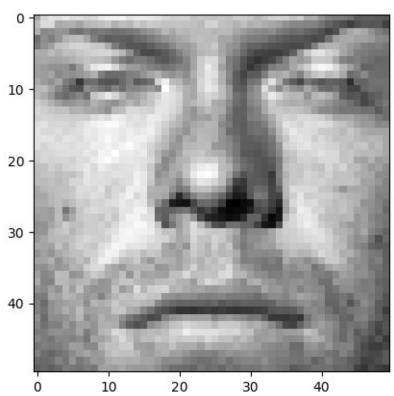
Programming Exercises

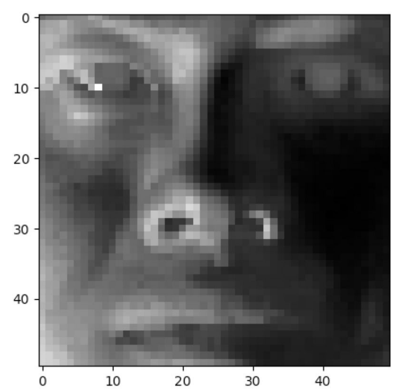
1. Code in faceRecog.py

(c) After processing the data, we compute the average face μ from the training set and plotted it as shown in the following image.

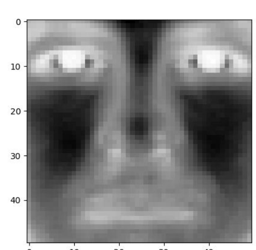
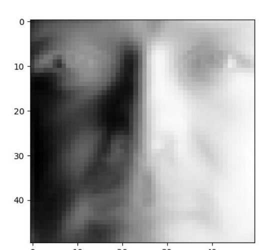
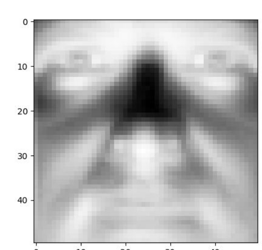


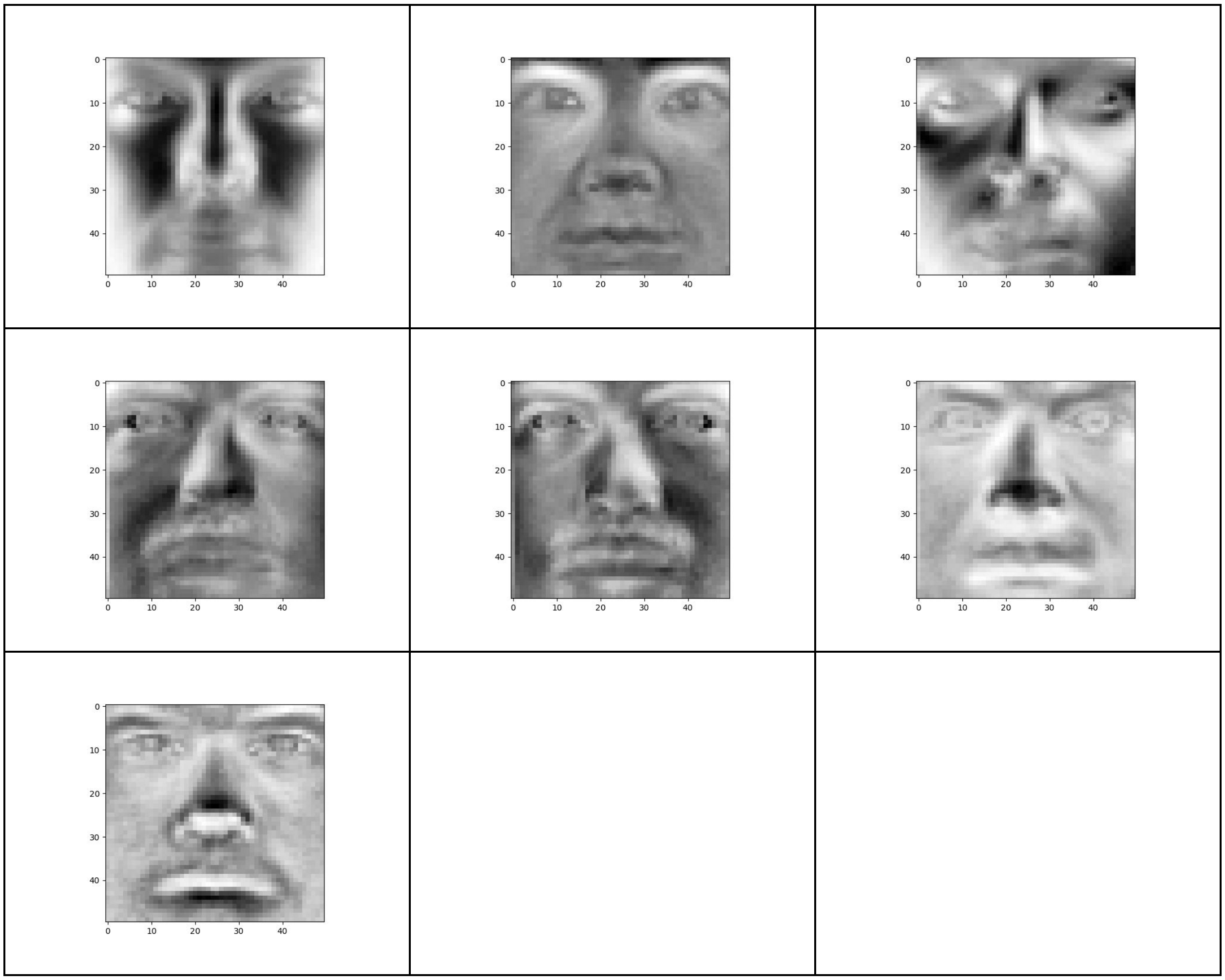
(d) After subtracting the face average μ from training set and testing set, we get two images below. The left one is for training set and the right one is for testing set.



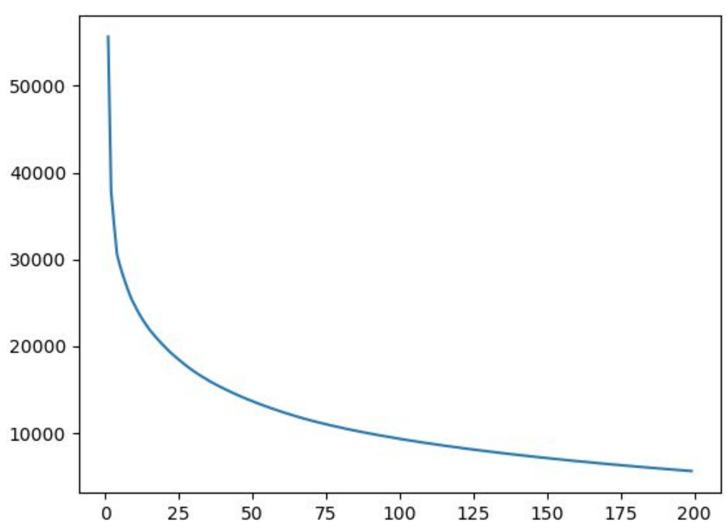


(e)After performing SVD on training set, we get the following first 10 eigenfaces.

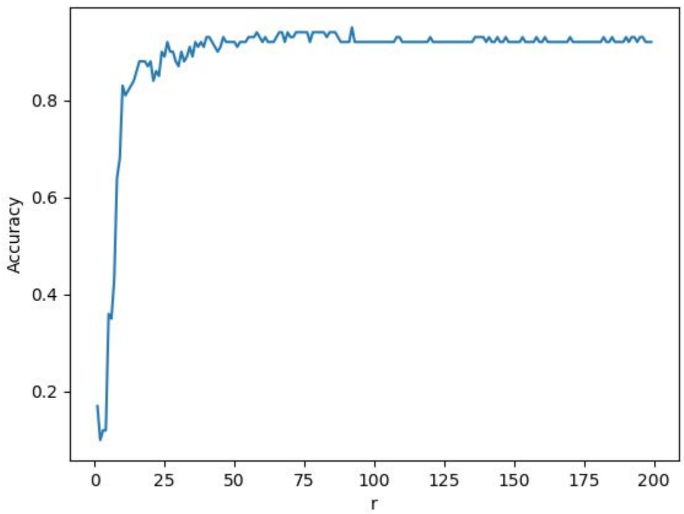


(f) Below is the plot of the rank-r approximation error versus r in range of 1 to 200. From that, we can see that the approximation error dramatically decreases when rank increases.



(g) See function generateFeature()

(h)Using the feature matrix F to train testing set on logistic regression model, we get the accuracy rate plot in the following graph when r=1,2,...,200. At r = 10, the accuracy is found to be 0.83.



2. **What's Cooking (code in cooking.py)**

(b) By looping through each sample, we found that there are 39774 samples in the training dataset, 20 categories (cuisines), and 6714 unique ingredients in total.

(d) The average classification accuracy using gaussian prior is 0.379493793821, while the accuracy using bernoulli prior is found to be 0.683587657646.

(e) Bernoulli prior performs better than the gaussian prior because in this case, our feature vector are discontinuous binary (0 or 1), suggesting whether the sample has the gradient or not. Therefore, we don't need to use the gaussian prior which works better for the continuous features.

(f) If we use the Logistic Regression model to perform the 3 fold cross-validation, the accuracy is 0.775758670409, which is better than both gaussian and bernoulli.

(g) Logistic regression model is then used to train on the whole training dataset, and the returned accuracy is 0.78338.

Written Exercises

4.1. $L(a) = a^T B a - \lambda (a^T W a - 1)$

$$\frac{dL}{da} = 2a^T B^T - 2\lambda a^T W^T = 0.$$

$$a^T B^T = \lambda a^T W^T$$

$$Ba = \lambda W a.$$

$$(W^{-1} B) a = \lambda a.$$

Therefore, we can see it is an eigenvalue problem.

4.2. (1). $\delta_1(x) < \delta_2(x)$.

$$x^T\Sigma^{-1}\mu_1 - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \log\pi_1 < x^T\Sigma^{-1}\mu_2 - \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 + \log\pi_2.$$

$$x^T\Sigma^{-1}(\mu_2-\mu_1) > \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \log\pi_2 - \log\pi_1$$

$$x^T\Sigma^{-1}(\mu_2-\mu_1) > \frac{1}{2}\mu_2^T\Sigma^{-1}\mu_2 - \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 + \log\frac{N_1}{N} - \log\frac{N_2}{N}.$$

(2) to minimize $\sum_{i=1}^{N}(y_i-\beta_0-\beta^Tx_i)^2 = (Y-\beta_0 1 - X\beta)^T(Y-\beta_0 1-X\beta)$

we compute $\frac{d}{d\beta}$ and $\frac{d}{d\beta_0}$.

$$\begin{cases} \frac{d}{d\beta} = 2X^TX\beta - 2X^TY + 2\beta_0 X^T 1 = 0. \\ \frac{d}{d\beta_0} = 2N\beta_0 - 2 1^T(Y-X\beta) = 0. \end{cases}$$

$$\Rightarrow \hat{\beta}_0 = \frac{1}{N}1^T(Y-X\beta). \quad ①$$

$$\Rightarrow 2X^TX\beta - X^TY + \left(\frac{1}{N}1^TY - \frac{1}{N}1^TX\beta\right)X^T1 = 0.$$

$$\Rightarrow \left(X^TX - \frac{1}{N}X^T1\,1^TX\right)\beta = X^TY - \frac{1}{N}X^T1\,1^TY. \quad ☆$$

let t_i be the target labels,

U_i be the n element vector with j-th element 1 if the j-th observation is class i, and zero otherwise.

then $X^TY - \frac{1}{N}X^T1\,1^TY = t_1N_1\mu_1 + t_2N_2\mu_2 - \frac{1}{N}(N_1\mu_1+N_2\mu_2)(t_1N_1+t_2N_2)$

$$= \frac{N_1N_2}{N}(t_1-t_2)(\mu_1-\mu_2).$$

$$X^TX - \frac{1}{N}X^T1\,1^TX = (N-2)\Sigma + N_1\mu_1\mu_1^T + N_2\mu_2\mu_2^T - \frac{1}{N}X^T1\,1^TX.$$

$$= (N-2)\Sigma + \frac{N_1N_2}{N}(\mu_2-\mu_1)(\mu_2-\mu_1)^T$$

$$= (N-2)\Sigma + \frac{N_1 N_2}{N} \overset{\Sigma_B}{\cancel{(\mu_2-\mu_1)(\mu_2-\mu_1)^T}}.$$

therefore, our ☆ equation becomes.

$$\left((N-2)\Sigma + \frac{N_1 N_2}{N}\Sigma_B\right)\beta = \frac{N_1 N_2}{N}(t_1-t_2)(\mu_1-\mu_2)$$

$$= \frac{N_1 N_2}{N}\left(-\frac{N}{N_1}-\frac{N}{N_2}\right)(\mu_1-\mu_2)$$

$$= N(\mu_2-\mu_1)$$

(3). we know $\Sigma_B\beta = (\mu_2-\mu_1)(\mu_2-\mu_1)^T\beta$

$$= \lambda(\mu_2-\mu_1)$$

So $\Sigma_B\beta$ is in same direction with $(\mu_2-\mu_1)$

So $\beta = \Sigma^{-1}(\mu_2-\mu_1)\lambda$.

$\Rightarrow \beta \propto \Sigma^{-1}(\mu_2-\mu_1)$

(4) based on (2),

we computed using t_1 and t_2 for the most part and only brings in the value of t_1 and t_2 in the end. Therefore, it holds as long as $\mu_2 \neq \mu_1$.

(5). frome (2), we get equation ①.

$$\beta_0 = \frac{1}{N} 1^T(Y - X\beta)$$

$$= \frac{1}{N}(t_1N_1 + t_2N_2) - \frac{1}{N}1^T X\beta.$$

Since $t_1N_1 + t_2N_2 = 0$.

$$\Rightarrow \beta_0 = -\frac{1}{N}1^T X\beta = -\frac{1}{N}(N_1\mu_1^T + N_2\mu_2^T)\beta.$$

$$\therefore f(x) = \beta_0 + \beta^T = -\frac{1}{N}(N_1\mu_1^T + N_2\mu_2^T)\beta + x^T\beta.$$

$$= \frac{1}{N}(Nx^T - N_1\mu_1^T - N_2\mu_2^T)\beta$$

$$= \frac{1}{N}(N x^T - N_1\mu_1^T - N_2\mu_2^T)\,\lambda\Sigma^{-1}(\mu_2-\mu_1)$$

(based on (3))

Therefore, it is different from LDA rule

$$f(x)>0 \Rightarrow \frac{1}{N}(N x^T - N_1\mu_1^T - N_2\mu_2^T)\,\lambda\Sigma^{-1}(\mu_2-\mu_1)>0$$

$$x^T\lambda\Sigma^{-1}(\mu_2-\mu_1) > (N_1\mu_1^T + N_2\mu_2^T)\,\lambda\Sigma^{-1}(\mu_2-\mu_1)$$

Written Exercise Q3

September 27, 2017

```
In [104]: import numpy as np

          M = [[1,0,3],[3,7,2],[2,-2,8],[0,-1,1],[5,8,7]]
          M = np.array(M, dtype='float')

          MTM = np.matmul(np.transpose(M),M)
          MMT = np.matmul(M, np.transpose(M))

          print 'MTM:\n', MTM
          print 'MMT:\n',MMT
```

```
MTM:
[[  39.   57.   60.]
 [  57.  118.   53.]
 [  60.   53.  127.]]
MMT:
[[  10.    9.   26.    3.   26.]
 [   9.   62.    8.   -5.   85.]
 [  26.    8.   72.   10.   50.]
 [   3.   -5.   10.    2.   -1.]
 [  26.   85.   50.   -1.  138.]]
```

```
In [107]: eigenvalues_MTM,eigenvectors_MTM = np.linalg.eig(MTM)
          eigenvalues_MMT,eigenvectors_MMT = np.linalg.eig(MMT)
          print 'eigenvalues of MTM are:\n',eigenvalues_MTM
          print 'eigenvectors of MTM are:\n',eigenvectors_MTM
          print 'eigenvalues of MMT are:\n',eigenvalues_MMT
          print 'eigenvectors of MMT are:\n',eigenvectors_MMT
```

```
eigenvalues of MTM are:
[  2.14670489e+02   9.32587341e-15   6.93295108e+01]
eigenvectors of MTM are:
[[ 0.42615127  0.90453403 -0.01460404]
 [ 0.61500884 -0.30151134 -0.72859799]
 [ 0.66344497 -0.30151134  0.68478587]]
eigenvalues of MMT are:
[  2.14670489e+02  -8.88178420e-16   6.93295108e+01  -3.34838281e-15
```

```
   7.47833227e-16]
eigenvectors of MMT are:
[[-0.16492942 -0.95539856  0.24497323 -0.54001979 -0.78501713]
 [-0.47164732 -0.03481209 -0.45330644 -0.62022234  0.30294097]
 [-0.33647055  0.27076072  0.82943965 -0.12704172  0.2856551 ]
 [-0.00330585  0.04409532  0.16974659  0.16015949  0.43709105]
 [-0.79820031  0.10366268 -0.13310656  0.53095405 -0.13902319]]
```

In [109]:
```
eigenvectors_MMT_trans = np.transpose(eigenvectors_MMT)
eigenvectors_MMT_trans[2] = np.negative(eigenvectors_MMT_trans[2]) #taking the negati
U = np.transpose(np.array([eigenvectors_MMT_trans[0],eigenvectors_MMT_trans[2]]))
print 'U:\n',U

D = np.array([eigenvalues_MTM[0], eigenvalues_MTM[2]])
D = np.diag(np.sqrt(D))
print 'D:\n', D

eigenvectors_MTM_trans = np.transpose(eigenvectors_MTM)
V = np.negative([eigenvectors_MTM_trans[0],eigenvectors_MTM_trans[2]])
print 'V:\n', V

tmp = np.matmul(D,V)
x = np.matmul(U,tmp)
print 'The reconstructed M:\n', x
print 'The reconstructed M from SVD is almost the same with the original M matrix'
```

```
U:
[[-0.16492942  0.24497323]
 [-0.47164732 -0.45330644]
 [-0.33647055  0.82943965]
 [-0.00330585  0.16974659]
 [-0.79820031 -0.13310656]]
D:
[[ 14.65163776   0.        ]
 [  0.           8.32643446]]
V:
[[-0.42615127 -0.61500884 -0.66344497]
 [ 0.01460404  0.72859799 -0.68478587]]
The reconstructed M:
[[ 1.05957729  2.97232071  0.20641116]
 [ 2.88975625  1.4999211   7.16934764]
 [ 2.20171904  8.063796   -1.45863887]
 [ 0.04128223  1.05957729 -0.93573059]
 [ 4.96762859  6.38498523  8.51790055]]
The reconstructed M from SVD is almost the same with the original M matrix
```

In [110]:
```
# u,s,v = np.linalg.svd(M)
```

In [103]:
```
# get one-dimensional approximation of matrix M
newD = np.matrix(D[0,0])
tmp = np.matmul(newD,np.matrix([V[0,:]]))
oneDApproxi = np.matmul(np.matrix(U[:,0].reshape(5,1)),tmp)
print 'one-dimensional approximation of matrix M is \n',oneDApproxi
```

```
one-dimensional approximation of matrix M is
[[ 1.02978864  1.48616035  1.60320558]
 [ 2.94487812  4.24996055  4.58467382]
 [ 2.10085952  3.031898    3.27068057]
 [ 0.02064112  0.02978864  0.0321347 ]
 [ 4.9838143   7.19249261  7.75895028]]
```

In [ ]: