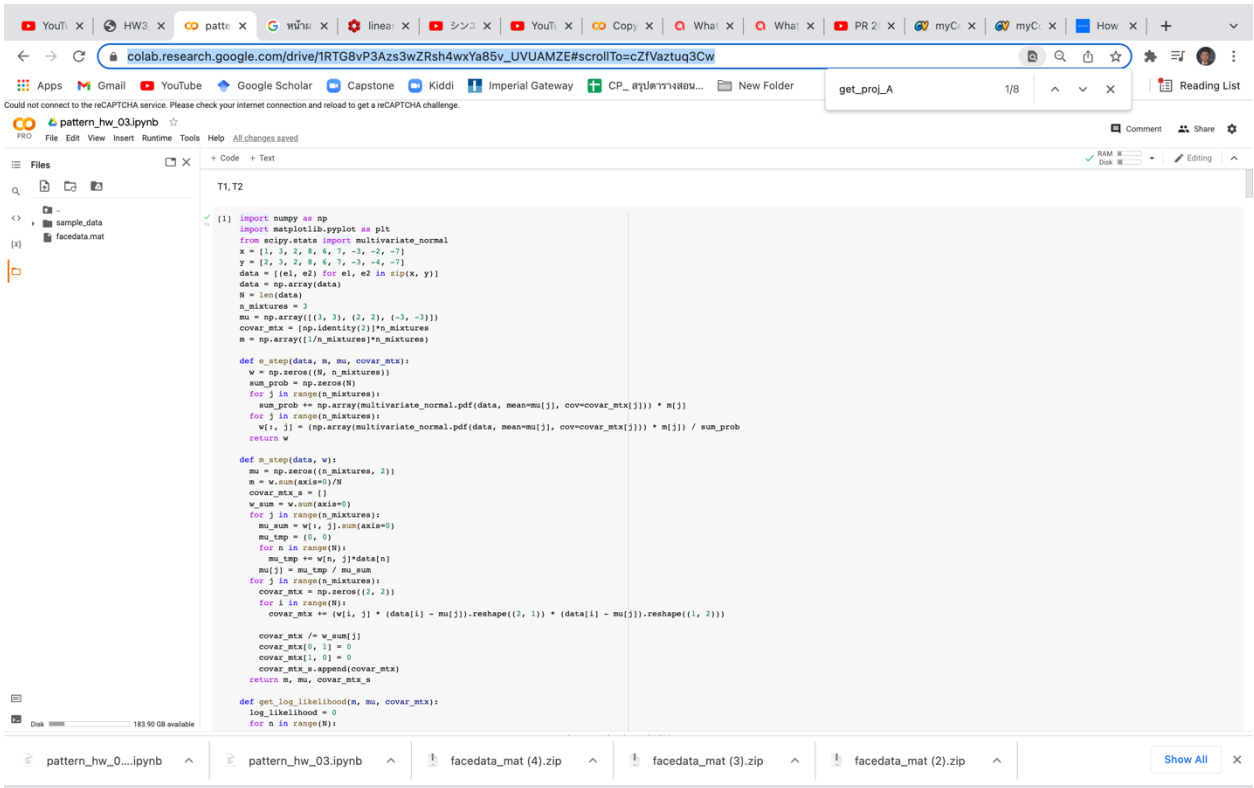
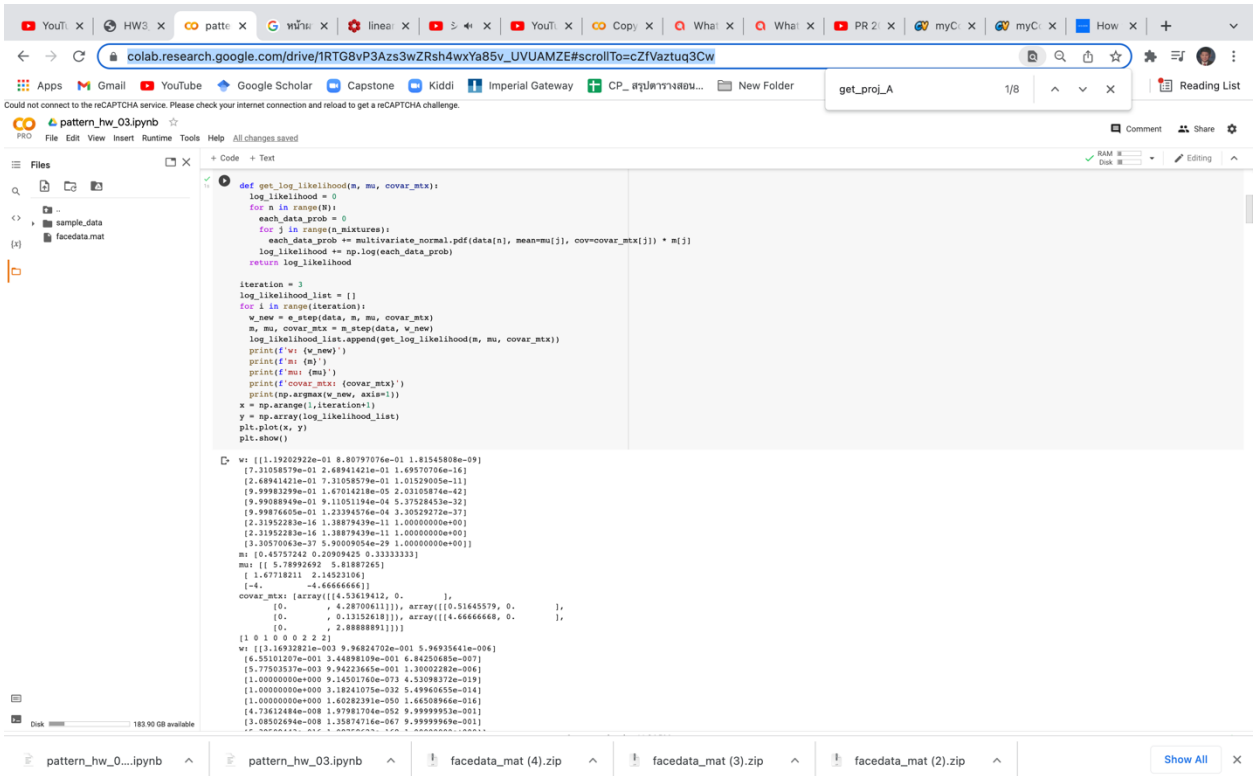


colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb

T1, T2

```
[1] import numpy as np
    import matplotlib.pyplot as plt
    from scipy.stats import multivariate_normal
    x = [1, 3, 2, 8, 6, 7, -3, -2, -7]
    y = [2, 3, 2, 8, 6, 7, -3, -4, -7]
    data = [(e1, e2) for e1, e2 in zip(x, y)]
    data = np.array(data)
    N = len(data)
    n_mixtures = 3
    mu = np.array([(3, 3), (2, 2), (-3, -3)])
    covar_mtx = [np.identity(2)]*n_mixtures
    m = np.array([1/n_mixtures]*n_mixtures)

    def e_step(data, m, mu, covar_mtx):
      w = np.zeros((N, n_mixtures))
      sum_prob = np.zeros(N)
      for j in range(n_mixtures):
        sum_prob += np.array(multivariate_normal.pdf(data, mean=mu[j], cov=covar_mtx[j])) * m[j]
      for j in range(n_mixtures):
        w[:, j] = (np.array(multivariate_normal.pdf(data, mean=mu[j], cov=covar_mtx[j])) * m[j]) / sum_prob
      return w

    def m_step(data, w):
      mu = np.zeros((n_mixtures, 2))
      m = w.sum(axis=0)/N
      covar_mtx_s = []
      w_sum = w.sum(axis=0)
      for j in range(n_mixtures):
        mu_sum = w[:, j].sum(axis=0)
        mu_tmp = (0, 0)
        for n in range(N):
          mu_tmp += w[n, j]*data[n]
        mu[j] = mu_tmp / mu_sum
      for j in range(n_mixtures):
        covar_mtx = np.zeros((2, 2))
        for i in range(N):
          covar_mtx += (w[i, j] * (data[i] - mu[j]).reshape((2, 1)) * (data[i] - mu[j]).reshape((1, 2)))

        covar_mtx /= w_sum[j]
        covar_mtx[0, 1] = 0
        covar_mtx[1, 0] = 0
        covar_mtx_s.append(covar_mtx)
      return m, mu, covar_mtx_s

    def get_log_likelihood(m, mu, covar_mtx):
      log_likelihood = 0
      for n in range(N):
```



colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb

```
    def get_log_likelihood(m, mu, covar_mtx):
      log_likelihood = 0
      for n in range(N):
        each_data_prob = 0
        for j in range(n_mixtures):
          each_data_prob += multivariate_normal.pdf(data[n], mean=mu[j], cov=covar_mtx[j]) * m[j]
        log_likelihood += np.log(each_data_prob)
      return log_likelihood

    iteration = 3
    log_likelihood_list = []
    for i in range(iteration):
      w_new = e_step(data, m, mu, covar_mtx)
      m, mu, covar_mtx = m_step(data, w_new)
      log_likelihood_list.append(get_log_likelihood(m, mu, covar_mtx))
      print(f'w: {w_new}')
      print(f'm: {m}')
      print(f'mu: {mu}')
      print(f'covar_mtx: {covar_mtx}')
      print(np.argmax(w_new, axis=1))
    x = np.arange(1,iteration+1)
    y = np.array(log_likelihood_list)
    plt.plot(x, y)
    plt.show()
```

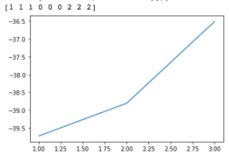
```
w: [[1.19202922e-01 8.80797076e-01 1.81545808e-09]
 [7.31058579e-01 2.68941421e-01 1.69570706e-16]
 [2.68941421e-01 7.31058579e-01 1.01529005e-11]
 [9.99983299e-01 1.67014218e-05 2.03105874e-42]
 [9.99088949e-01 9.11051194e-04 5.37528453e-32]
 [9.99876655e-01 1.23394576e-04 3.30529272e-37]
 [2.31952283e-16 1.38879439e-11 1.00000000e+00]
 [2.31952283e-16 1.38879439e-11 1.00000000e+00]
 [3.30570063e-37 5.90009054e-29 1.00000000e+00]]
m: [0.45757242 0.20909425 0.33333333]
mu: [[ 5.78992692  5.81887265]
 [ 1.67718211  2.14523106]
 [-4.         -4.66666666]]
covar_mtx: [array([[4.53619412, 0.        ],
       [0.        , 4.28700611]]), array([[0.51645579, 0.        ],
       [0.        , 0.13152618]]), array([[4.66666668, 0.        ],
       [0.        , 2.88888891]])]
[1 0 1 0 0 0 2 2 2]
w: [[3.16932821e-003 9.96824702e-001 5.96935641e-006]
 [6.55101207e-001 3.44898109e-001 6.84250685e-007]
 [5.77503537e-003 9.94223665e-001 1.30002282e-006]
 [1.00000000e+000 9.14501760e-073 4.53098372e-019]
 [1.00000000e+000 3.18241075e-032 5.49960655e-014]
 [1.00000000e+000 1.60282391e-050 1.66508966e-016]
 [4.73612484e-008 1.97981704e-052 9.99999953e-001]
 [3.08502694e-008 1.35874716e-067 9.99999969e-001]
```

colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

pattern_hw_03.ipynb

```
w: [[1.19202922e-01 8.80797076e-01 1.81545808e-09]
 [7.31058579e-01 2.68941421e-01 1.69570706e-16]
 [2.68941421e-01 7.31058579e-01 1.01529005e-11]
 [9.99983299e-01 1.67014218e-05 2.03105874e-42]
 [9.99088949e-01 9.11051194e-04 5.37528453e-32]
 [9.99876605e-01 1.23394576e-04 3.30529272e-37]
 [2.31952283e-16 1.38879439e-11 1.00000000e+00]
 [2.31952283e-16 1.38879439e-11 1.00000000e+00]
 [3.30570063e-37 5.90009054e-29 1.00000000e+00]]
m: [0.45757242 0.20909425 0.33333333]
mu: [[ 5.78992692  5.81887265]
 [ 1.67718211  2.14523106]
 [-4.         -4.66666666]]
covar_mtx: [array([[4.53619412, 0.        ],
       [0.        , 4.28700611]]), array([[0.51645579, 0.        ],
       [0.        , 0.13152618]]), array([[4.66666668, 0.        ],
       [0.        , 2.88888891]])]
[1 0 1 0 0 0 2 2 2]
w: [[3.16932821e-003 9.96824702e-001 5.96935641e-006]
 [6.55101207e-001 3.44898109e-001 6.84250685e-007]
 [5.77503537e-003 9.94223665e-001 1.30002282e-006]
 [1.00000000e+000 9.14501760e-073 4.53098372e-019]
 [1.00000000e+000 3.18241075e-032 5.49960655e-014]
 [1.00000000e+000 1.60282391e-050 1.66508966e-016]
 [4.73612484e-008 1.97981704e-052 9.99999953e-001]
 [3.08502694e-008 1.35874716e-067 9.99999969e-001]
 [5.39509443e-016 1.08758623e-168 1.00000000e+000]]
m: [0.40711618 0.25954961 0.33333421]
mu: [[ 6.27176215  6.27262711]
 [ 1.72091544  2.14764812]
 [-3.99998589 -4.6666488 ]]
covar_mtx: [array([[2.94672736, 0.        ],
       [0.        , 2.93847196]]), array([[0.49649261, 0.        ],
       [0.        , 0.12584815]]), array([[4.66673088, 0.        ],
       [0.        , 2.88900236]])]
[1 0 1 0 0 0 2 2 2]
w: [[9.82897443e-005 9.99896667e-001 5.04302423e-006]
 [2.45965474e-001 7.54033295e-001 1.23094211e-006]
 [3.18013803e-004 9.99680992e-001 9.94302212e-007]
 [1.00000000e+000 9.43430508e-076 3.14527994e-019]
 [1.00000000e+000 1.86592722e-033 4.19732750e-014]
 [1.00000000e+000 1.37418755e-052 1.08248377e-016]
 [5.61748867e-013 6.98460925e-055 1.00000000e+000]
 [3.64921259e-013 1.02517456e-070 1.00000000e+000]
 [1.03044593e-025 1.72424695e-176 1.00000000e+000]]
m: [0.36070909 0.30595677 0.33333414]
mu: [[ 6.6962644   6.69629468]
 [ 1.91071238  2.27383436]
 [-3.99998673 -4.6666501 ]]
covar_mtx: [array([[1.73961067, 0.        ],
       [0.        , 1.73929602]]), array([[0.62898406, 0.        ],
       [0.        , 0.1988491 ]]), array([[4.66672942, 0.        ],
       [0.        , 2.88899545]])]
```



colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

pattern_hw_03.ipynb

[1 0 1 0 0 0 2 2 2]
w: [[3.16932821e-003 9.96824702e-001 5.96935641e-006]
 [6.55101207e-001 3.44898109e-001 6.84250685e-007]
 [5.77503537e-003 9.94223665e-001 1.30002282e-006]
 [1.00000000e+000 9.14501760e-073 4.53098372e-019]
 [1.00000000e+000 3.18241075e-032 5.49960655e-014]
 [1.00000000e+000 1.60282391e-050 1.66508966e-016]
 [4.73612484e-008 1.97981704e-052 9.99999953e-001]
 [3.08502694e-008 1.35874716e-067 9.99999969e-001]
 [5.39509443e-016 1.08758623e-168 1.00000000e+000]]
m: [0.40711618 0.25954961 0.33333421]
mu: [[6.27176215 6.27262711]
 [1.72091544 2.14764812]
 [-3.99998589 -4.6666488]]
covar_mtx: [array([[2.94672736, 0.   ],
       [0.   , 2.93847196]]), array([[0.49649261, 0.   ],
       [0.   , 0.12584815]]), array([[4.66673088, 0.   ],
       [0.   , 2.88900236]])]
[1 0 1 0 0 0 2 2 2]
w: [[9.82897443e-005 9.99896667e-001 5.04302423e-006]
 [2.45965474e-001 7.54033295e-001 1.23094211e-006]
 [3.18013803e-004 9.99680992e-001 9.94302212e-007]
 [1.00000000e+000 9.43430508e-076 3.14527994e-019]
 [1.00000000e+000 1.86592722e-033 4.19732750e-014]
 [1.00000000e+000 1.37418755e-052 1.08248377e-016]
 [5.61748867e-013 6.98460925e-055 1.00000000e+000]
 [3.64921259e-013 1.02517456e-070 1.00000000e+000]
 [1.03044593e-025 1.72424695e-176 1.00000000e+000]]
m: [0.36070909 0.30595677 0.33333414]
mu: [[6.6962644   6.69629468]
 [1.91071238 2.27383436]
 [-3.99998673 -4.6666501]]
covar_mtx: [array([[1.73961067, 0.   ],
       [0.   , 1.73929602]]), array([[0.62898406, 0.   ],
       [0.   , 0.1988491]]), array([[4.66672942, 0.   ],
       [0.   , 2.88899545]])]
[1 1 1 0 0 0 2 2 2]

colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb

T3, T4

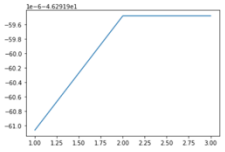
```
[2] n_mixtures = 2
    mu = np.array([[3, 3], [-3, -3]])
    covar_mtx = [np.identity(2)]*n_mixtures
    m = np.array([1/n_mixtures]*n_mixtures)
    log_likelihood_list_2 = []
    iteration = 3
    for i in range(iteration):
      w_new = e_step(data, m, mu, covar_mtx)
      m, mu, covar_mtx = m_step(data, w_new)
      log_likelihood_list_2.append(get_log_likelihood(m, mu, covar_mtx))
      print(f'w: {w_new}')
      print(f'm: {m}')
      print(f'mu: {mu}')
      print(f'covar_mtx: {covar_mtx}')
    x = np.arange(1,iteration+1)
    y = np.array(log_likelihood_list_2)
    plt.plot(x, y)
    plt.show()
```

```
w: [[9.99999985e-01 1.52299795e-08]
 [1.00000000e+00 2.31952283e-16]
 [1.00000000e+00 3.77513454e-11]
 [1.00000000e+00 2.03109266e-42]
 [1.00000000e+00 5.38018616e-32]
 [1.00000000e+00 3.10570063e-37]
 [2.31952283e-16 1.00000000e+00]
 [2.31952283e-16 1.00000000e+00]
 [3.30570063e-37 1.00000000e+00]]
m: [0.66666666 0.33333334]
mu: [[ 4.50000001  4.66666667]
 [-3.99999997 -4.66666663]]
covar_mtx: [array([[6.91666665, 0.        ],
       [0.        , 5.88888889]]), array([[4.66666677, 0.        ],
       [0.        , 2.8888891 ]])]
w: [[9.99879274e-01 1.20725832e-04]
 [9.99999741e-01 2.59403362e-07]
 [9.99975922e-01 2.40783341e-05]
 [1.00000000e+00 9.39286607e-19]
 [1.00000000e+00 7.41043154e-14]
 [1.00000000e+00 2.98366370e-16]
 [2.41448223e-04 9.99758552e-01]
 [1.52869075e-04 9.99847131e-01]
 [5.22429300e-09 9.99999995e-01]]
m: [0.66669436 0.33330564]
mu: [[ 4.49961311  4.66620178]
 [-3.99993241 -4.66651231]]
covar_mtx: [array([[6.91944755, 0.        ],
       [0.        , 5.89275124]]), array([[4.66806942, 0.        ],
```


colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb


```
[2]  [2.31952283e-16 1.00000000e+00]
 [3.30570063e-37 1.00000000e+00]]
m: [0.66666666 0.33333334]
mu: [[ 4.50000001  4.66666667]
 [-3.99999997 -4.66666663]]
covar_mtx: [array([[6.91666665, 0.        ],
       [0.        , 5.88888889]]), array([[4.66666677, 0.        ],
       [0.        , 2.8888891 ]])]
w: [[9.99879274e-01 1.20725832e-04]
 [9.99999741e-01 2.59403362e-07]
 [9.99975922e-01 2.40783341e-05]
 [1.00000000e+00 9.39286607e-19]
 [1.00000000e+00 7.41043154e-14]
 [1.00000000e+00 2.98366370e-16]
 [2.41448223e-04 9.99758552e-01]
 [1.52869075e-04 9.99847131e-01]
 [5.22429300e-09 9.99999995e-01]]
m: [0.66669436 0.33330564]
mu: [[ 4.49961311  4.66620178]
 [-3.99993241 -4.66651231]]
covar_mtx: [array([[6.91944755, 0.        ],
       [0.        , 5.89275124]]), array([[4.66806942, 0.        ],
       [0.        , 2.89103318]])]
w: [[9.99878589e-01 1.21411383e-04]
 [9.99999738e-01 2.61769575e-07]
 [9.99975770e-01 2.42295830e-05]
 [1.00000000e+00 9.63496021e-19]
 [1.00000000e+00 7.54839047e-14]
 [1.00000000e+00 3.04963776e-16]
 [2.42792697e-04 9.99757207e-01]
 [1.53838319e-04 9.99846162e-01]
 [5.28882554e-09 9.99999995e-01]]
m: [0.66669453 0.33330547]
mu: [[ 4.49961084  4.66619903]
 [-3.99993206 -4.66651141]]
covar_mtx: [array([[6.91946372, 0.        ],
       [0.        , 5.8927741 ]]), array([[4.66807754, 0.        ],
       [0.        , 2.89104566]])]
```



Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

get_proj_A 1/8

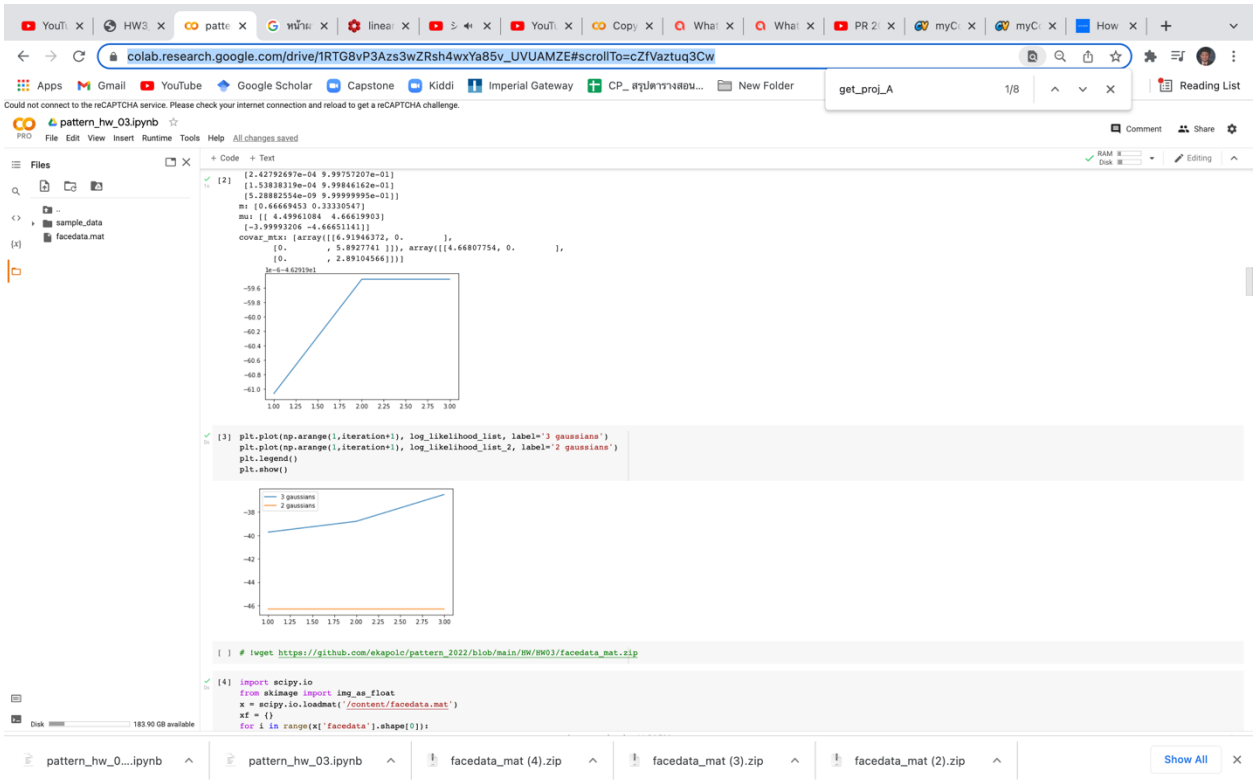
Apps Gmail YouTube Google Scholar Capstone Kiddi Imperial Gateway CP_ สรุปตารางสอน... New Folder Reading List

Files: sample_data, facedata.mat

Disk 183.90 GB available

pattern_hw_0....ipynb | pattern_hw_03.ipynb | facedata_mat (4).zip | facedata_mat (3).zip | facedata_mat (2).zip | Show All

The 3-estimator GMM is better than 2-estimator GMM in all iterations in terms of log-likelihood.



colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

pattern_hw_03.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Files

sample_data

facedata.mat

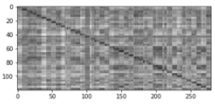
T5

```
[5] def euclidean_dis(a, b):
      dis = np.sqrt(np.sum(np.power(a.reshape(-1) - b.reshape(-1), 2)))
      return dis
    print(euclidean_dis(xf[0, 0], xf[0, 1]))
    print(euclidean_dis(xf[0, 0], xf[1, 0]))
```

```
10.037616294165492
8.173295099737281
```

T6

```
def get_similartity_mtx(T, D):
  m = len(T)
  n = len(D)
  A = np.zeros((m, n))
  for i in range(m):
    for j in range(n):
      A[i, j] = np.sqrt(np.sum(np.power(T[i].reshape(-1) - D[j].reshape(-1), 2)))
  plt.imshow(A, cmap="gray")
  plt.show()
  return A

def get_T_D(xf):
  T = []
  D = []
  for i in range(40):
    for j in range(10):
      if j <= 2:
        T.append(xf[i, j])
      else:
        D.append(xf[i, j])
  return T, D

T, D = get_T_D(xf)
A = get_similartity_mtx(T, D)
```



Disk 183.90 GB available

pattern_hw_0....ipynb | pattern_hw_03.ipynb | facedata_mat (4).zip | facedata_mat (3).zip | facedata_mat (2).zip | Show All

colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.

pattern_hw_03.ipynb

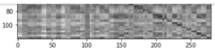
File Edit View Insert Runtime Tools Help All changes saved

Files

sample_data

facedata.mat

[6]



T7

```
[131] print(f'In my opinion, the black square in [5:10, 5:10] represents the similarity between the images which come from the same people.')
```

```
In my opinion, the black square in [5:10, 5:10] represents the similarity between the images which come from the same people.
```

T8

```
[15] def get_tp_fa(A, t):
       test_num = A.shape[1]
       train_num = A.shape[0]
       tp = 0
       tn = 0
       fp = 0
       fn = 0
       for test_idx in range(test_num):
         people_face_dis_list = [A.max()]*(train_num//3)
         for train_idx in range(train_num):
           people_face_dis_list[train_idx//3] = min(people_face_dis_list[train_idx//3], A[train_idx, test_idx])
         for idx, dis in enumerate(people_face_dis_list):
           if dis < t and test_idx//7 == idx:
             tp += 1
           elif dis >= t and test_idx//7 == idx:
             fn += 1
           elif dis < t and test_idx//7 != idx:
             fp += 1
           else:
             tn += 1
       tp_rate = tp/(tp + fn)
       fa_rate = fp/(fp + tn)
       return tp_rate, fa_rate
```

```
[132] tp_rate, fa_rate = get_tp_fa(A, 10)
      print(f'At t=10, tp_rate: {tp_rate}')
      print(f'At t=10, fa_rate: {fa_rate}')
```

```
At t=10, tp_rate: 0.9964285714285714
At t=10, fa_rate: 0.4564102564102564
```

T9

Disk 183.90 GB available

pattern_hw_0....ipynb | pattern_hw_03.ipynb | facedata_mat (4).zip | facedata_mat (3).zip | facedata_mat (2).zip | Show All

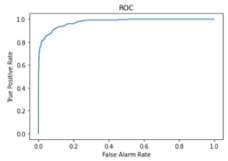
colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb

```
At t=10, tp_rate: 0.9964285714285714
At t=10, fa_rate: 0.4564102564102564
```

T9

```
[133] def plot_roc(A, num=1000):
        min_thres = A.min()
        max_thres = A.max()
        thres = np.linspace(min_thres, max_thres, num=num)
        tp_rate_list = []
        fa_rate_list = []
        all_rate = []
        for t in thres:
          tp_rate, fa_rate = get_tp_fa(A, t)
          tp_rate_list.append(tp_rate)
          fa_rate_list.append(fa_rate)
          all_rate.append((t, tp_rate, fa_rate))
        tp_rate_list, fa_rate_list = np.array(tp_rate_list), np.array(fa_rate_list)
        plt.title('ROC')
        plt.plot(fa_rate_list, tp_rate_list)
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Alarm Rate')
        plt.show()
        return all_rate
      all_rate = plot_roc(A)
      print(f'The lowest and the highest value of A should be the minimum and the maximum thresholds, respectively.')
```




```
The lowest and the highest value of A should be the minimum and the maximum thresholds, respectively.
```

T10

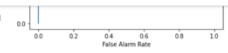
```
[147] for t, tp_rate, fa_rate in all_rate:
        if abs(tp_rate - 1 + fa_rate) <= 4e-3:
          print(f'EER: {t}')
```

Disk 183.90 GB available

pattern_hw_0....ipynb | pattern_hw_03.ipynb | facedata_mat (4).zip | facedata_mat (3).zip | facedata_mat (2).zip | Show All

colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb




```
The lowest and the highest value of A should be the minimum and the maximum thresholds, respectively.
```

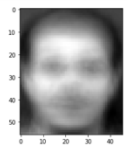
T10

```
[147] for t, tp_rate, fa_rate in all_rate:
        if abs(tp_rate - 1 + fa_rate) <= 4e-3:
          print(f'EER: {t}')
          print(f'tp rate: {tp_rate}')
          print(f'fa rate: {fa_rate}')
        elif abs(fa_rate-0.0010) <= 9e-5:
          print(f'recall rate: {tp_rate} at False Alarm rate: 10%')
```

```
recall rate: 0.5428571428571428 at False Alarm rate: 10%
EER: 8.084041408785595
tp rate: 0.9071428571428571
fa rate: 0.08891941391941392
```

T11

```
[19] T_np = np.array(T)
     D_np = np.array(D)
     average_face = np.average(T_np, axis=0)
     plt.imshow(average_face, cmap="gray")
     plt.show()
```



T12

```
[149] X_tmp = np.concatenate((T_np, D_np), axis=0)
```

pattern_hw_0....ipynb | pattern_hw_03.ipynb | facedata_mat (4).zip | facedata_mat (3).zip | facedata_mat (2).zip | Show All

T12

```
[149] X_tmp = np.concatenate((T_np, D_np), axis=0)
      X = (X_tmp - average_face).reshape((len(X_tmp), 56*46)).T
      cov_mtx = X.dot(X.T)
      print(f'The dimension of covariance matrix is {cov_mtx.shape}')
      print(f'The rank of covariance matrix is {np.linalg.matrix_rank(cov_mtx)}')
```

```
The dimension of covariance matrix is (2576, 2576)
The rank of covariance matrix is 399
```

T13

```
[150] gram = X.T.dot(X)
      print(f'The size of Gram matrix is {gram.shape}')
      print(f'The rank of Gram matrix is {np.linalg.matrix_rank(gram)}')
      print(f'We expect to get {len(X_tmp)-1} non-zero values')
```

```
The size of Gram matrix is (400, 400)
The rank of Gram matrix is 399
We expect to get 399 non-zero values
```

T14

```
[161] def is_symmetric(A):
        is_symmetric = True
        for i in range(A.shape[0]):
          if not is_symmetric:
            break
          for j in range(A.shape[1]):
            if i == j:
              continue
            elif A[i, j] != A[j, i]:
              is_symmetric = False
              break
        return True
      is_sym = is_symmetric(gram)
      print(f'Gram is a symmetric matrix: {is_sym}')
      explain = """
      X -> mxn matrix
      X.T -> nxm matrix
      gram[i, j] is computed by X.T[i, :].dot(X[:, j])
      and gram[j, i] is computed by X.T[j, :].dot(X[:, i])
      we also know that a.dot(b) equals b.dot(a)
      In addition, X.T[i, :] = X[:, i] and X[:, j] = X.T[j, :].
```


colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw
Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.
pattern_hw_03.ipynb
File Edit View Insert Runtime Tools Help All changes saved

[150] gram = X.T.dot(X)
      print(f'The size of Gram matrix is {gram.shape}')
      print(f'The rank of Gram matrix is {np.linalg.matrix_rank(gram)}')
      print(f'We expect to get {len(X_tmp)-1} non-zero values')

The size of Gram matrix is (400, 400)
The rank of Gram matrix is 399
We expect to get 399 non-zero values

T14

```
def is_symmetric(A):
  is_symmetric = True
  for i in range(A.shape[0]):
    if not is_symmetric:
      break
    for j in range(A.shape[1]):
      if i == j:
        continue
      elif A[i, j] != A[j, i]:
        is_symmetric = False
        break
  return True
is_sym = is_symmetric(gram)
print(f'Gram is a symmetric matrix: {is_sym}')
explain = """
X -> mxn matrix
X.T -> nxm matrix
gram[i, j] is computed by X.T[i, :].dot(X[:, j])
and gram[j, i] is computed by X.T[j, :].dot(X[:, i])
we also know that a.dot(b) equals b.dot(a)
In addition, X.T[i, :] = X[:, i] and X[:, j] = X.T[j, :].
As a result, gram is a symmetric matrix.
"""
print(explain)
```

```
Gram is a symmetric matrix: True

X -> mxn matrix
X.T -> nxm matrix
gram[i, j] is computed by X.T[i, :].dot(X[:, j])
and gram[j, i] is computed by X.T[j, :].dot(X[:, i])
we also know that a.dot(b) equals b.dot(a)
In addition, X.T[i, :] = X[:, i] and X[:, j] = X.T[j, :].
As a result, gram is a symmetric matrix.
```

pattern_hw_0....ipynb pattern_hw_03.ipynb facedata_mat (4).zip facedata_mat (3).zip facedata_mat (2).zip Show All

colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb

T15

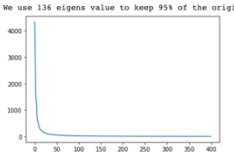
```
X_tmp = np.concatenate((T_np, D_np), axis=0)
X = (X_tmp - average_face).reshape((len(X_tmp), 56*46)).T
gram = X.T.dot(X)
eigen_val, eigen_vec = np.linalg.eigh(gram)
eigen_val = eigen_val[::-1]
eigen_vec = eigen_vec[:, ::-1]
non_zero_eigen = len([a for a in eigen_val if a >= 1e-4])
print(f'There are {non_zero_eigen} of non zero eigen values')
```

```
There are 399 of non zero eigen values
```

T16

```
all_var = np.sum(eigen_val)
proportion_var = all_var*.95
var_sum = 0
for idx, eigen_value in enumerate(eigen_val):
  var_sum += eigen_value
  if var_sum >= proportion_var:
    print(f'We use {idx+1} eigens value to keep 95% of the original variance.')
    break
plt.plot(np.arange(len(eigen_val)),np.array(eigen_val))
plt.show()
```

```
We use 136 eigens value to keep 95% of the original variance.
```



T17, T18

```
v = X.dot(eigen_vec)
```

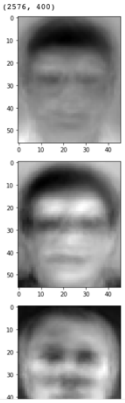
colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb

T17, T18

```
v = X.dot(eigen_vec)
print(v.shape)
new_v = []
for i in range(v.shape[1]):
  v_size = np.linalg.norm(v[:, i], axis=0)
  new_v.append(v[:, i]/v_size)
new_v = np.array(new_v)
for i in range(10):
  plt.imshow(new_v[i].reshape((56, 46)), cmap="gray")
  plt.show()
```

```
(2576, 400)
```



colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb

```
[117]
```

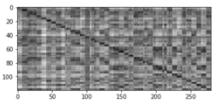
```
[158] explain = """
      I think in the first eigen face, it captures the shirt of the person as the color of the shirt is brightest among other pixels.
      Regarding the second eigen face, it reflects the forehead of a person.
      After observing the original images, I think it is quite reasonable that why these two eigen images are the ones that have the highest variances.
      """
      print(explain)
```

```
I think in the first eigen face, it captures the shirt of the person as the color of the shirt is brightest among other pixels.
Regarding the second eigen face, it reflects the forehead of a person.
After observing the original images, I think it is quite reasonable that why these two eigen images are the ones that have the highest variances.
```

T19

```
[121] k = 10
      def get_proj_A(new_v, X, k):
        print(new_v.shape)
        print(X.shape)
        p = new_v[:k, :].dot(X)
        # print(p.shape)
        proj_T = [p[:, e] for e in range(len(T_np))]
        proj_D = [p[:, e+len(T_np)] for e in range(len(D_np))]
        proj_A = get_similartity_mtx(proj_T, proj_D)
        return proj_A
      proj_A = get_proj_A(new_v, X, k)
```

```
(400, 2576)
(2576, 400)
```



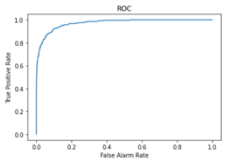
```
[26] all_rate = plot_roc(proj_A)
```

pattern_hw_0....ipynb | pattern_hw_03.ipynb | facedata_mat (4).zip | facedata_mat (3).zip | facedata_mat (2).zip | Show All

colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb


```
[26] all_rate = plot_roc(proj_A)
```




```
[27] for t, tp_rate, fa_rate in all_rate:
       if abs(tp_rate - 1 + fa_rate) <= 1e-3:
         print(f'EER: {t}')
         print(f'tp rate: {tp_rate}')
         print(f'fa rate: {fa_rate}')
       elif abs(fa_rate-0.0010) <= 8e-6:
         print(f'recall rate: {tp_rate} at False Alarm rate: 10%')
```

```
recall rate: 0.4964285714285144 at False Alarm rate: 10%
```

```
     def get_EER(all_rate, thres = 5e-3):
       found = False
       for t, tp_rate, fa_rate in all_rate:
         if abs(tp_rate - 1 + fa_rate) <= thres:
           print(f'EER: {t}')
           print(f'tp rate: {tp_rate}')
           print(f'fa rate: {fa_rate}')
           found = True
           return fa_rate
       if not found:
         thres *= 5
         for t, tp_rate, fa_rate in all_rate:
           if abs(tp_rate - 1 + fa_rate) <= thres:
             print(f'EER: {t}')
             print(f'tp rate: {tp_rate}')
             print(f'fa rate: {fa_rate}')
             found = True
             return fa_rate
       return None
```

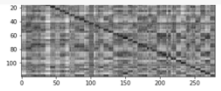
pattern_hw_0....ipynb | pattern_hw_03.ipynb | facedata_mat (4).zip | facedata_mat (3).zip | facedata_mat (2).zip | Show All

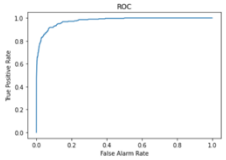
colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb

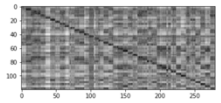
T20

```
[31] k_list = [5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
     eer_list = []
     for k in k_list:
       proj_A = get_proj_A(new_v, X, k)
       all_rate = plot_roc(proj_A)
       eer_list.append((k, get_EER(all_rate)))
```





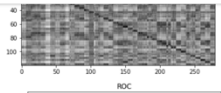

```
EER: 5.32466894402587
tp rate: 0.9178571428571428
fa rate: 0.07756410256410257
```

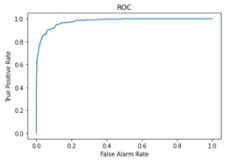


pattern_hw_0....ipynb | pattern_hw_03.ipynb | facedata_mat (4).zip | facedata_mat (3).zip | facedata_mat (2).zip | Show All

colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb






```
EER: 5.5541144674255225
tp rate: 0.9142857142857143
fa rate: 0.08571428571428572
```

```
[32] eer_list_tmp = np.array([e for _, e in eer_list])
     best_k = eer_list[np.argmin(eer_list_tmp)][0]
     print(f'best eer is when k = {best_k}')

     best eer is when k = 13
```

T21

```
[33] print(f'The rank must be {min(40-1, 400-40)}')

     The rank must be 39
```

T22

```
[164] proj_PCA = new_v[:39, :].dot(X)
      global_mean = np.mean(proj_PCA, axis=1)
      C = 40
      class_mean = np.zeros((40, 10, 39))
      proj_T = np.array([proj_PCA[:, e] for e in range(len(T_np))])
      proj_D = np.array([proj_PCA[:, e+len(T_np)] for e in range(len(D_np))])
      for i in range(len(proj_T)):
        class_mean[i//3, i%3, :] = proj_T[i]
```

pattern_hw_0....ipynb | pattern_hw_03.ipynb | facedata_mat (4).zip | facedata_mat (3).zip | facedata_mat (2).zip | Show All

colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb

T21

```
[33] print(f'The rank must be {min(40-1, 400-40)}')
```

```
The rank must be 39
```

T22

```
[164] proj_PCA = new_v[:39, :].dot(X)
      global_mean = np.mean(proj_PCA, axis=1)
      C = 40
      class_mean = np.zeros((40, 10, 39))
      proj_T = np.array([proj_PCA[:, e] for e in range(len(T_np))])
      proj_D = np.array([proj_PCA[:, e+len(T_np)] for e in range(len(D_np))])
      for i in range(len(proj_T)):
        class_mean[i//3, i%3, :] = proj_T[i]
      for i in range(len(proj_D)):
        class_mean[i//7, i%7+3, :] = proj_D[i]
      within_class_mean = np.mean(class_mean, axis=1)
      S_B = np.zeros((39, 39))
      S_W = np.zeros((39, 39))
      for c in range(C):
        diff = (within_class_mean[c] - global_mean).reshape((39, 1))
        S_B += diff.dot(diff.T)
      for c in range(C):
        for i in range(3):
          diff = (proj_T[c*3+i] - within_class_mean[c]).reshape((39, 1))
          S_W += diff.dot(diff.T)
        for j in range(7):
          diff = (proj_D[c*7+i] - within_class_mean[c]).reshape((39, 1))
          S_W += diff.dot(diff.T)
      eigen_val_LDA, eigen_vec_LDA = np.linalg.eigh(np.matmul(np.linalg.inv(S_W), S_B))
      non_zero = len([e for e in eigen_val_LDA if e >= 1e-10])
      S_W_inv_S_B_sym = is_symmetric(np.matmul(np.linalg.inv(S_W), S_B))
      print(f'We can still use numpy.linalg.eigh as the S_W inverse x S_B is symmetric: {S_W_inv_S_B_sym}')
      print(f'There are {non_zero} non-zero values in eigen values')
      eigen_val_LDA = eigen_val_LDA[::-1]
      eigen_vec_LDA = eigen_vec_LDA[:, ::-1]
```

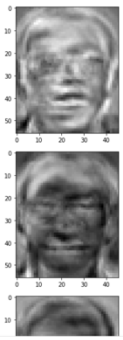
```
We can still use numpy.linalg.eigh as the S_W inverse x S_B is symmetric: True
There are 22 non-zero values in eigen values
```

T23

```
[165] explain = """
      It is completely different from PCA projection since the eigen face of the LDA belongs to some people,
      while the PCA's eigenfaces look like the the average of people faces.
      """
      print(explain)
```

```
It is completely different from PCA projection since the eigen face of the LDA belongs to some people,
while the PCA's eigenfaces look like the the average of people faces.
```

```
v_LDA = eigen_vec_LDA.dot(new_v[:39, :])
for i in range(10):
  plt.imshow(v_LDA[i].reshape((56, 46)), cmap="gray")
  plt.show()
```



pattern_hw_0....ipynb | pattern_hw_03.ipynb | facedata_mat (4).zip | facedata_mat (3).zip | facedata_mat (2).zip | Show All

colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

pattern_hw_03.ipynb

T23

```
[165] explain = """
      It is completely different from PCA projection since the eigen face of the LDA belongs to some people,
      while the PCA's eigenfaces look like the the average of people faces.
      """
      print(explain)
```

```
It is completely different from PCA projection since the eigen face of the LDA belongs to some people,
while the PCA's eigenfaces look like the the average of people faces.
```

```
v_LDA = eigen_vec_LDA.dot(new_v[:39, :])
for i in range(10):
  plt.imshow(v_LDA[i].reshape((56, 46)), cmap="gray")
  plt.show()
```

pattern_hw_0....ipynb | pattern_hw_03.ipynb | facedata_mat (4).zip | facedata_mat (3).zip | facedata_mat (2).zip | Show All

colab.research.google.com/drive/1RTG8vP3Azs3wZRsh4wxYa85v_UVUAMZE#scrollTo=cZfVaztuq3Cw

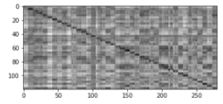
get_proj_A 1/8

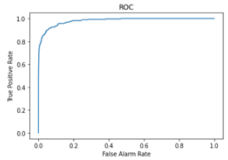
pattern_hw_03.ipynb

T24

```
[124] proj_A_LDA = get_proj_A(v_LDA, X, 39)
      all_rate = plot_roc(proj_A_LDA)
      get_EER(all_rate, thres = 5e-3)
```

```
(39, 2576)
(2576, 400)
```





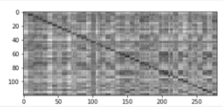

```
EER: 6.653314539085665
tp rate: 0.925
fa rate: 0.07454212454212454
0.07454212454212454
```

```
for t, tp_rate, fa_rate in all_rate:
  if abs(fa_rate-0.0010) <= 8e-6:
    print(f'recall rate: {tp_rate} at False Alarm rate: 10%')
```

```
recall rate: 0.5857142857142857 at False Alarm rate: 10%
```

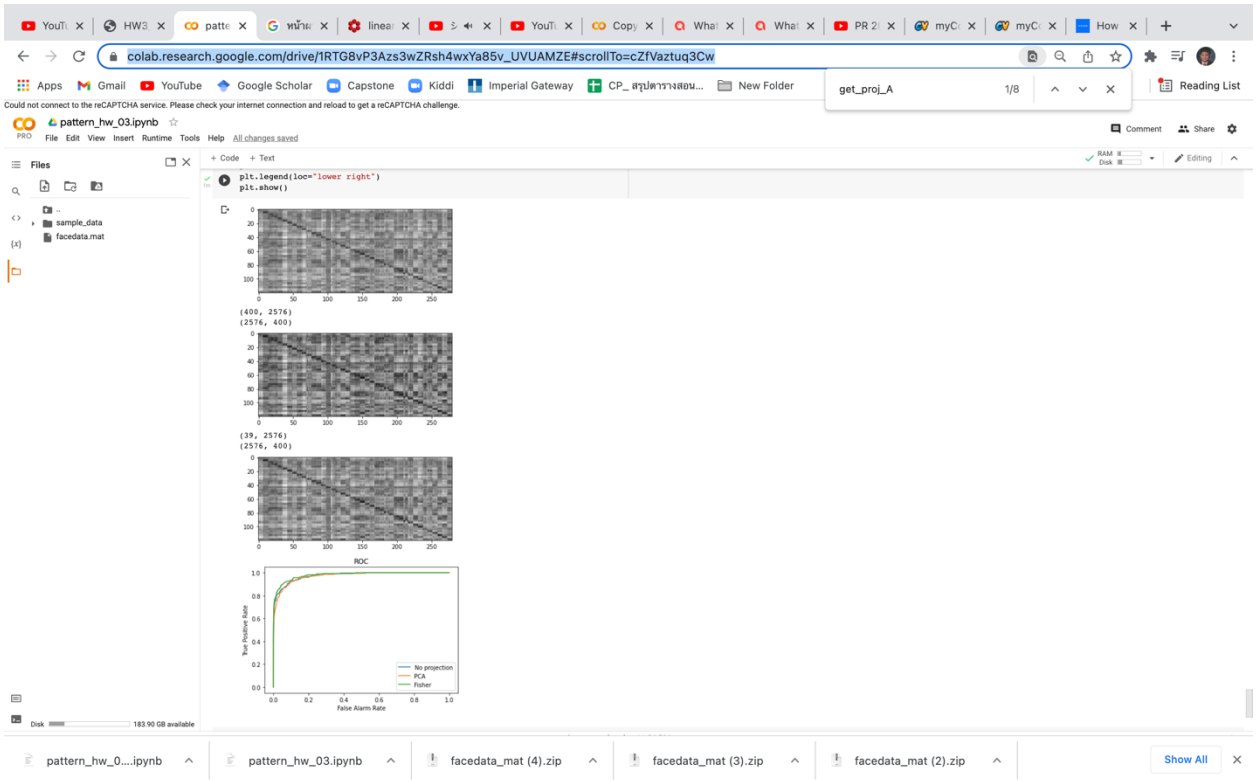
T25

```
A = get_similartity_mtx(T, D)
A_PCA = get_proj_A(new_v, X, k=10)
A_LDA = get_proj_A(v_LDA, X, 39)
num = 1000
min_thres = min(A.min(), A_PCA.min(), A_LDA.min())
max_thres = max(A.max(), A_PCA.max(), A_LDA.max())
thres = np.linspace(min_thres, max_thres, num=num)
tp_rate_list = []
fa_rate_list = []
tp_rate_list_PCA = []
fa_rate_list_PCA = []
tp_rate_list_LDA = []
fa_rate_list_LDA = []
for t in thres:
  tp_rate, fa_rate = get_tp_fa(A, t)
  tp_rate_list.append(tp_rate)
  fa_rate_list.append(fa_rate)

  tp_rate_PCA, fa_rate_PCA = get_tp_fa(A_PCA, t)
  tp_rate_list_PCA.append(tp_rate_PCA)
  fa_rate_list_PCA.append(fa_rate_PCA)

  tp_rate_LDA, fa_rate_LDA = get_tp_fa(A_LDA, t)
  tp_rate_list_LDA.append(tp_rate_LDA)
  fa_rate_list_LDA.append(fa_rate_LDA)

tp_rate_list, fa_rate_list = np.array(tp_rate_list), np.array(fa_rate_list)
tp_rate_list_PCA, fa_rate_list_PCA = np.array(tp_rate_list_PCA), np.array(fa_rate_list_PCA)
tp_rate_list_LDA, fa_rate_list_LDA = np.array(tp_rate_list_LDA), np.array(fa_rate_list_LDA)
plt.title('ROC')
plt.plot(fa_rate_list, tp_rate_list, label="No projection")
plt.plot(fa_rate_list_PCA, tp_rate_list_PCA, label="PCA")
plt.plot(fa_rate_list_LDA, tp_rate_list_LDA, label="Fisher")
plt.ylabel('True Positive Rate')
plt.xlabel('False Alarm Rate')
plt.legend(loc="lower right")
plt.show()
```



Disk 183.90 GB available

pattern_hw_0....ipynb | pattern_hw_03.ipynb | facedata_mat (4).zip | facedata_mat (3).zip | facedata_mat (2).zip | Show All



Looking along the false alarm rate axis, we can see that the Fisher method is better than the others clearly. This may be due to the Fisher algo. incorporate the class of the data into its consideration. However, it seems that the PCA at k=10 underperforms the no projection method. To my mind, this is owing to the number of eigen vectors which were used was too small.