EE378B Homework 8 Solution

Due to: Zixuan Liu

Question (a)

We generated the synthetic data following the problem setting. The table shows that with n increasing, the ratio of average Q becomes larger. The code for part (a) and (b) is listed below.

Methods	**n=200**	**n=400**	**n=800**	**n=1600**	**n=3200**
PCA	0.905	0.946	0.971	0.986	0.993
NMF	0.909	0.949	0.975	0.987	0.992

Table 1: Results over 10 iterations

```
import numpy as np
import matplotlib.pyplot as plt
from nmf_hw import nmf
import scipy as sp
from scipy.sparse.linalg import svds

d = 200
zeros = np.zeros(d // 2)
ones = np.ones(d // 2)
theta = np.concatenate([zeros, ones])

N = [200, 400, 800, 1600, 3200]
const = np.sqrt(d)
r = 1
Iter = 10
for n in N:
  Q_avg_PCA = 0
  Q_avg_NMF = 0
  for iter in range(Iter):
    np.random.shuffle(theta)
    X = np.zeros((n, d))
    w = np.random.rand(n)
    for i in range(n):
      for j in range(d):
        p = w[i] * theta[j] / const
        Xij = const * np.random.choice(2, p=[1-p, p])
        X[i, j] = Xij

    # PCA
    SIGMA = X.T.dot(X) / n
    U, sigma, VT = sp.sparse.linalg.svds(SIGMA, k=r)
    # U, sigma, VT = np.linalg.svd(SIGMA)
    # print(sigma)
    # print(U.shape)
    u1 = U[:, 0]
    u1_norm = np.linalg.norm(u1)
    theta_norm = np.linalg.norm(theta)
    Q = np.abs(np.dot(u1, theta)) / (u1_norm * theta_norm)
    # print('n =', n, 'Iter', iter, ',Q = ', Q)
    Q_avg_PCA += Q

    # NMF
    X_norm = X / (X.sum(axis=1, keepdims=True) + 1e-10)
    Winit = np.random.rand(n, r) + 0.5
    Hinit = np.random.rand(r, d) + 0.5
    W, H = nmf(X_norm, Winit, Hinit, tol=1e-3, maxiter=100)
```

```python
    # print(H)
    # print(theta)
    h = H[0]
    h_norm = np.linalg.norm(h)
    theta_norm = np.linalg.norm(theta)
    Q = np.abs(np.dot(h, theta)) / (h_norm * theta_norm)
    # print('n =', n, 'Iter', iter, ',Q = ', Q)
    Q_avg_NMF += Q
  print('NMF n =', n, 'Q_avg = ', Q_avg_NMF / Iter)
  print('PCA n =', n, 'Q_avg = ', Q_avg_PCA / Iter)
```

Question (b)

We implemented NMF algorithm following the instruction of homework document. Since the name of algorithm is non-negative matrix factorization, we consider using non-negative matrix H and W as initialization. Also, we observe that the algorithm will gradually converge to some points, therefore we regard $|F(X, H^{t+1}, W^{t+1}) - F(X, H^{t+1}, W^{t+1})| <$ tol as the convergence criterion. Note that we choose 10^{-3} for both synthetic data as well as MNIST test data.
The result of NMF shows that in most settings, it outperforms the PCA algorithm, except for the n=3200 case.

Methods	**n=200**	**n=400**	**n=800**	**n=1600**	**n=3200**
PCA	0.905	0.946	0.971	0.986	0.993
NMF	0.909	0.949	0.975	0.987	0.992

Table 2: Results over 10 iterations

```python
''' Nonnegative Matrix Facotrization'''
import numpy as np
from time import time

def nmf(X, Winit, Hinit, tol, maxiter=100):
  eps_threshold = 1e-10
  W = Winit # (n, r)
  H = Hinit # (r, d)
  initt = time()
  past_score = 0
  for iter in range(maxiter):
    WH = W.dot(H) # (n, d)
    X_WH = X / (WH + eps_threshold) # (n, d)
    X_WH_H = X_WH.dot(H.T) # (n, r)
    W_tilde = W * X_WH_H # (n, r)
    newW = W_tilde / W_tilde.sum(axis=0, keepdims=True) # (n, r)
    newWH = newW.dot(H) # (n, d)
    X_newWH = X / (newWH + eps_threshold) # (n, d)
    W_X_newWH = newW.T.dot(X_newWH) # (r, d)
    H_tilde = H * W_X_newWH # (r, d)
    newH = H_tilde / H_tilde.sum(axis=1, keepdims=True) # (r, d)
    H = newH
    W = newW
    score = F(X, W, H)
    if np.abs(score - past_score) < tol:  break
    past_score = score
  return W, H

def F(X, W, H):
  eps_threshold = 1e-10
  n, d = X.shape
  WH = W.dot(H) # (n, d)
  score = np.sum(X * np.log(WH + eps_threshold) - WH)
  return score
```

Question (c)

We first extract the data, and apply PCA on it. From the plots of the 6 principal vectors, we can see numbers mixed up, for example, we can see a dark blue 6 from the first plot, and a dark blue 3 or 8 in the third plots, and a dark blue 9, a yellow 2 from the fourth plots. These implies that the PCA do extract some useful features from the dataset.

```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
from scipy.sparse.linalg import svds
from nmf_hw import nmf

fname = './mnist_test.csv'
f_outpath = './'
with open(fname, 'rb') as f:
  data = np.loadtxt(f, delimiter=',')

print(data.shape)
print(data[:, 0])

y = data[:, 0]
x = data[:, 1:]
x_img = x.reshape(-1, 28, 28)

N, d = x.shape
r = 6
'''PCA for MNIST'''
SIGMA = x.T.dot(x) / N
U, sigma, VT = sp.sparse.linalg.svds(SIGMA, k=r)
for i in range(r):
  u = U[:, i]
  u_img = u.reshape(28, 28)
  plt.imshow(u_img)
  plt.title('PCA r=' + str(i + 1))
  plt.colorbar(shrink=0.83)
  # plt.show()
  plt.savefig(f_outpath + 'PCA r= ' + str(i + 1) + '.png', format='png', transparent=True, dpi=300, pad_inches = 0)
  plt.clf()
  plt.cla()
```

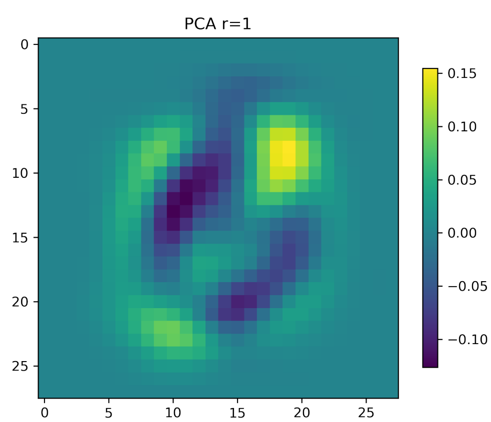

Figure 1: PCA for MNIST: r = 1

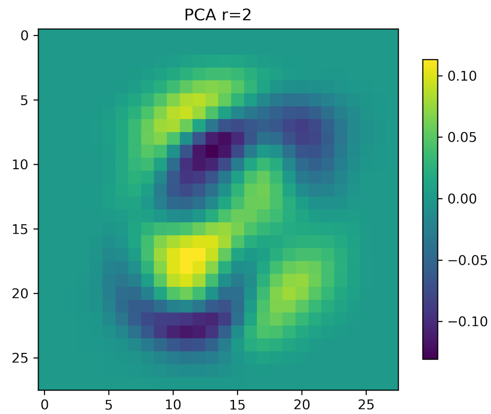

Figure 2: PCA for MNIST: r = 2

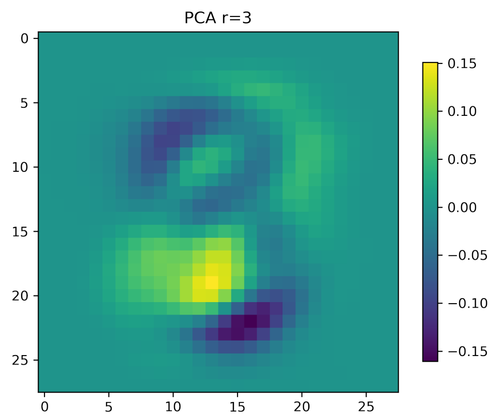

Figure 3: PCA for MNIST: r = 3

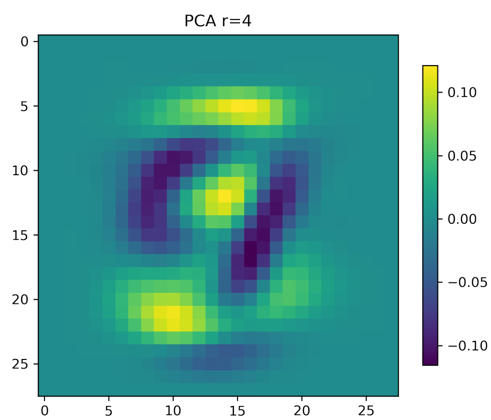

Figure 4: PCA for MNIST: r = 4

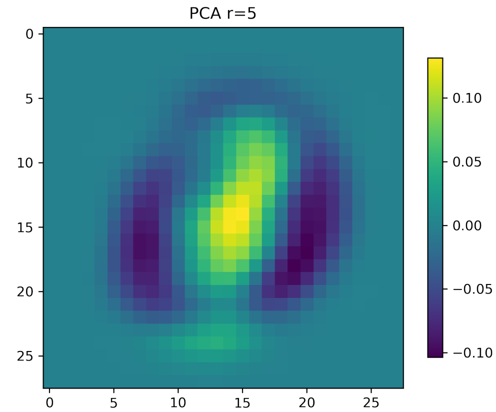

Figure 5: PCA for MNIST: r = 5

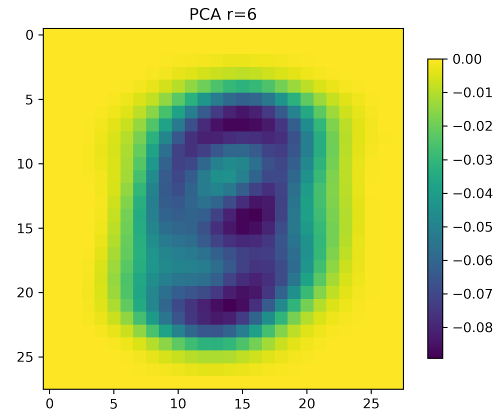

Figure 6: PCA for MNIST: r = 6

Question (d)

We apply the NMF algorithm to MNIST test data. Similar to PCA algorithm, we can see features from the plots. For example, we can see a 'C' like feature in the first plot, and a rough structure of 4 in the second, a line in the third plot, indicating the digit 1, 7, 9. We also see a rough 7 or 9 or 2 in the fourth plot, and a 9 in the fifth plot, and a 1 in the sixth plot. A little bit different from the PCA result is that, NMF tends to extract separate features instead of mixed ones.

```
'''NMF for MNIST'''
x_norm = x / x.sum(1).reshape(-1, 1)
# Normalize the data
Winit = np.random.rand(N, r) + 0.5
Hinit = np.random.rand(r, d) + 0.5
W, H = nmf(x_norm, Winit, Hinit, tol=1e-3, maxiter=500)
print(H.shape)
for i in range(r):
  u = H[i, :]
  u_img = u.reshape(28, 28)
  plt.imshow(u_img)
  plt.title('NMF r=' + str(i + 1))
  plt.colorbar(shrink=0.83)
  # plt.show()
  plt.savefig(f_outpath + 'NMF r= ' + str(i + 1) + '.png', format='png', transparent=True, dpi=300, pad_inches = 0)
  plt.clf()
  plt.cla()
```

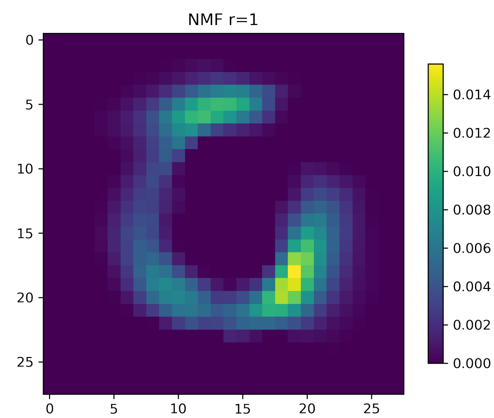

Figure 7: NMF for MNIST: r = 1

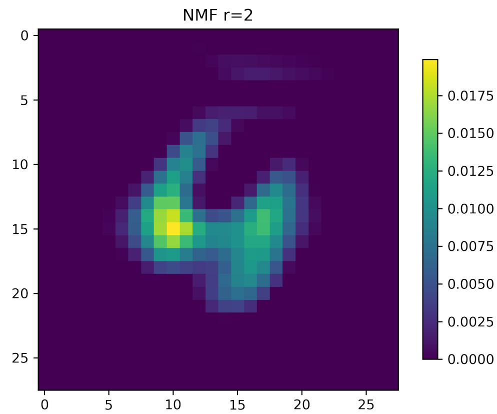

Figure 8: NMF for MNIST: r = 2

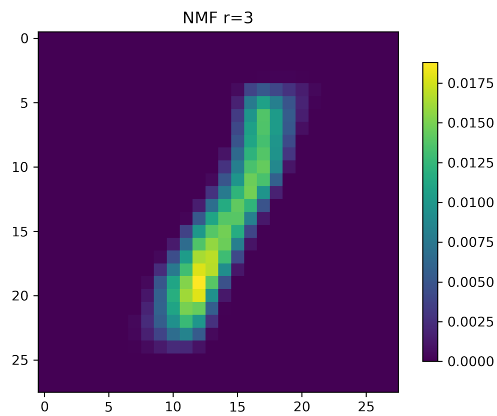

Figure 9: NMF for MNIST: r = 3

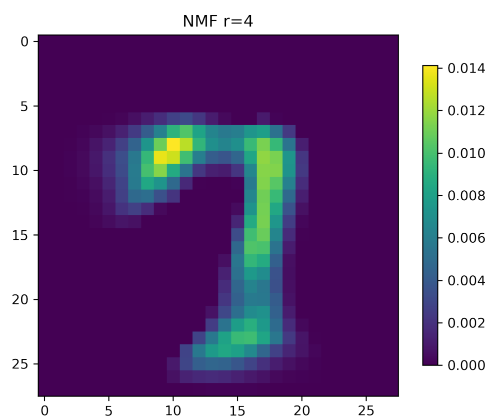

Figure 10: NMF for MNIST: r = 4

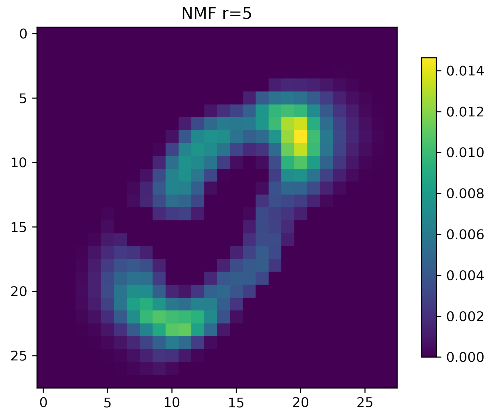

Figure 11: NMF for MNIST: r = 5

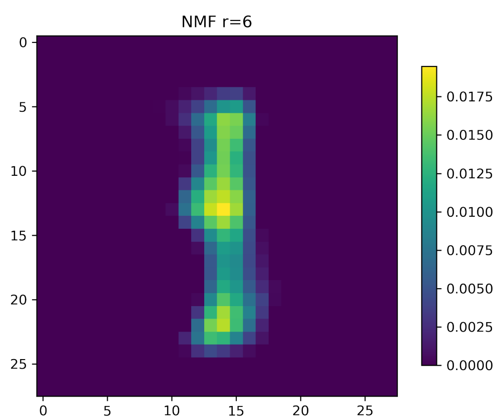

Figure 12: NMF for MNIST: r = 6