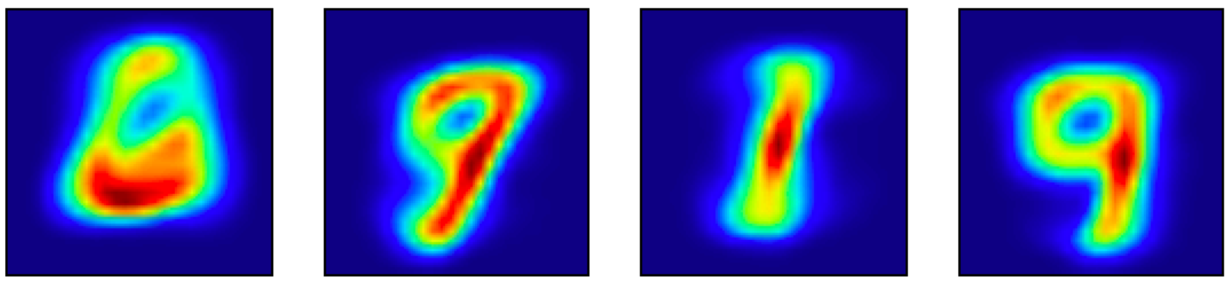
CS 189/289 Introduction to Machine Learning
Homework 7: Unsupervised
Shuhui Huang SID:3032129712

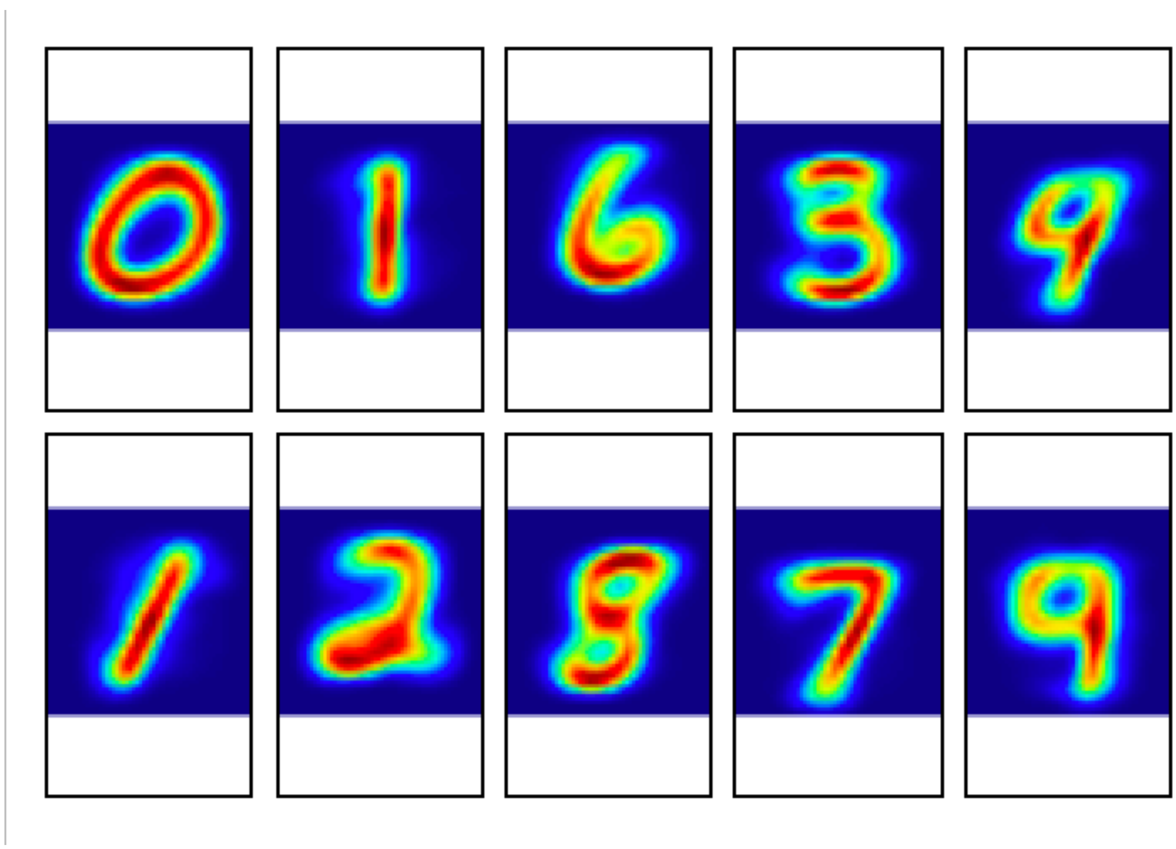
1. k-means clustering

(a) I implement Lloyd's algorithm for solving the k-means objective and the code has been attached in appendix. In my implementation, I initialize the K clusters' centers in "kmeans++ initialization scheme" with K=5, 10, 20 are plotted as follows:

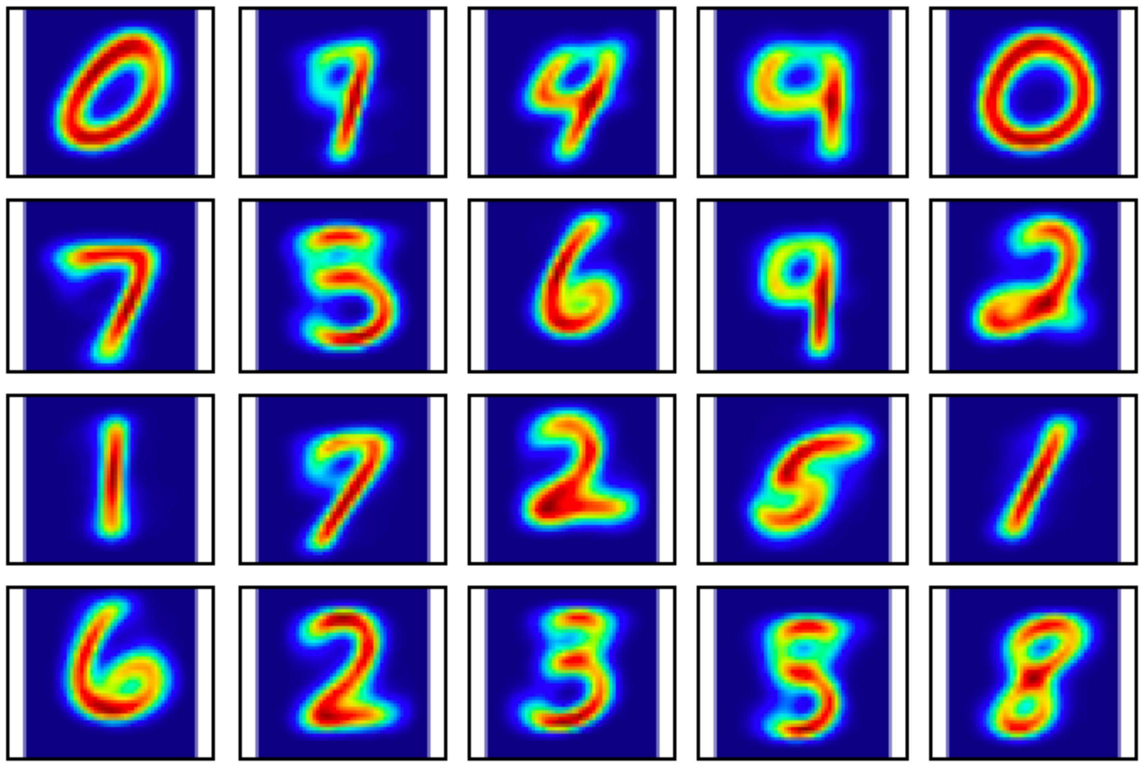
K=5:



K=10:



K=20:



The differences between results with different numbers of cluster centers is that (1) number of pictures in results will be different (2) with cluster centers increase, the same digit is more likely to repeat.

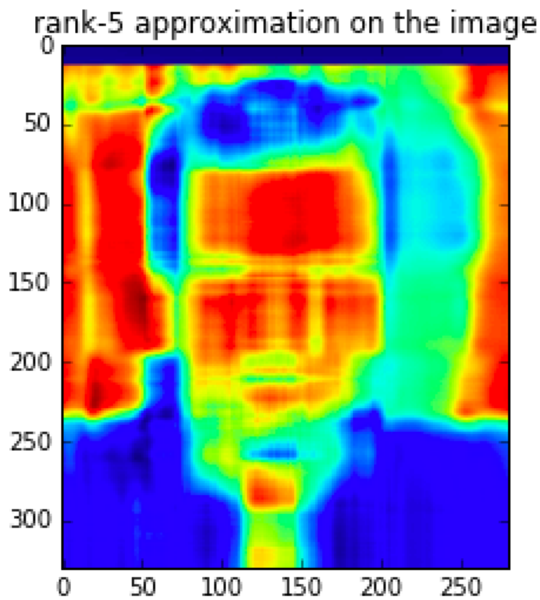
2. Low-Rank Approximation

(a).

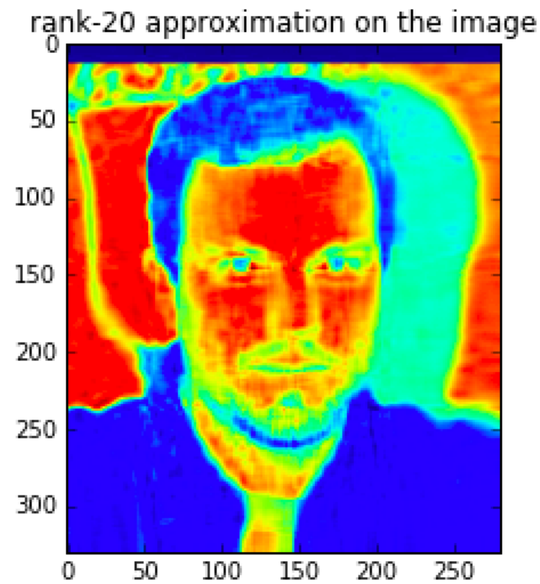
Let D=U$\sum V^T$, $\sum = diag(\sigma_1, \sigma_2, \sigma_3, \dots, \sigma_m)$. So we want to get $\min(\text{rank} \leq \text{r})||\text{D} - \widehat{\text{D}}||_F = \sqrt{\sigma_r + \sigma_{r+1} + \cdots + \sigma_m}$

My code has been attached in the appendix, and rank-5, rank-20, and rank-100 approximation on the image I got is:

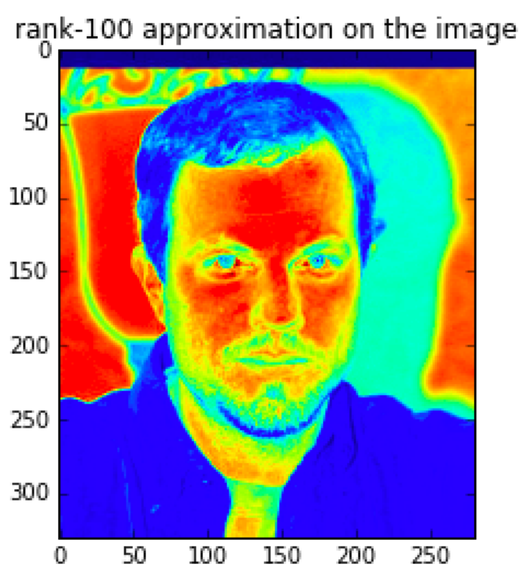
Rank=5:



Rank=20:

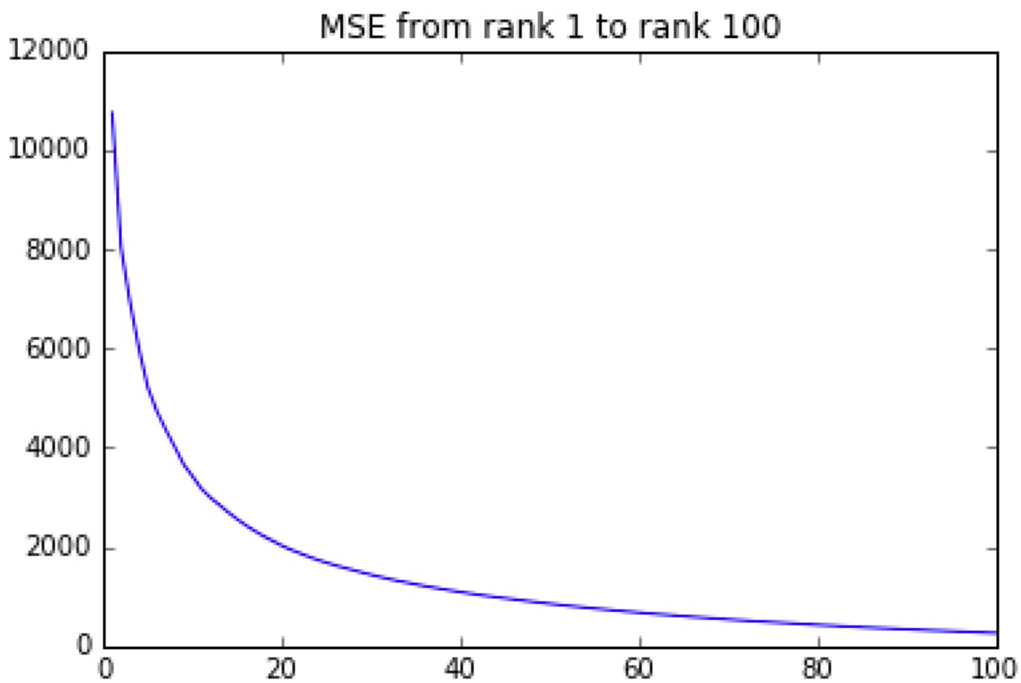


Rank=100:



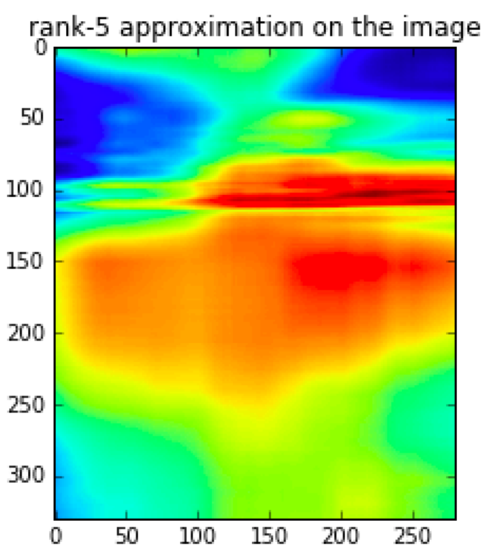
(b).

I use sum of squares of all entries in the matrix followed by a square root as mse, and my code has been attached in the appendix. The plot I got is :

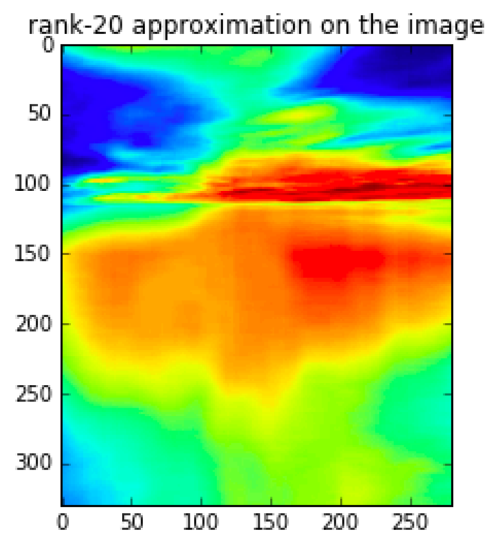


(c) The plot for sky is:

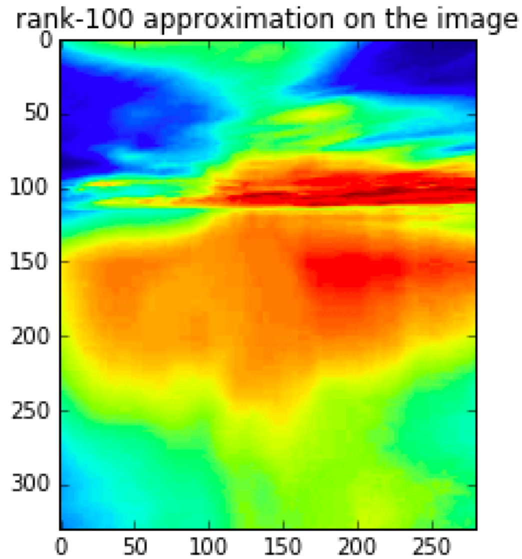
rank=5:



rank=20:



Rank=100:



(d) The lowest rank approximation for face is 50, for sky is 15. The possible reason for such difference is that there are higher differences among pixels in face image, or say, when choose the same rank, face image will lose more information compared with sky image, so the lowest rank for sky is smaller than face. And in matrix, we can say the difference among eigenvalues is higher in sky, so the first 15 biggest eigenvalues and its eigenvectors contains most information, while in face, the first 50 biggest eigenvalues and its eigenvectors contains most information.

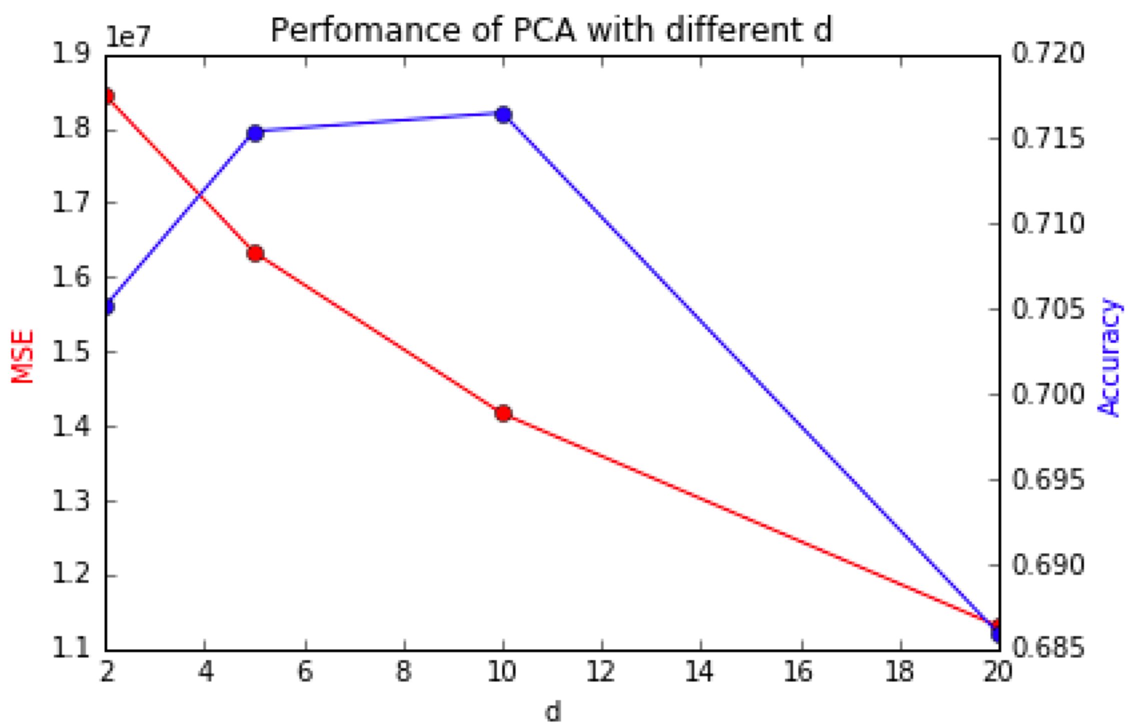
3. Joke Recommender System

3.2 Latent Factor Model

(a). After replacing all missing values with 0 and using PCA to learn the vector representation, I get the following MSEs and prediction accuracies with different value of d (2,5,10,20):

MSE:
[18441623.01788158, 16333384.42019682, 14165432.75799964, 11304007.439729325]

Prediction accuracies:
[0.70514905149051488, 0.71544715447154472, 0.71653116531165306, 0.68590785907859075]



with larger d, MSE will become smaller and smaller, but prediction accuracy might decrease since if d is too large we may over fit the model.

(b) minimize the MSE only on rated joke

If we minimize MSE only on rated joke:

If V_j is fixed, let $\frac{\partial L}{\partial u_i} = 0$, so we can have:

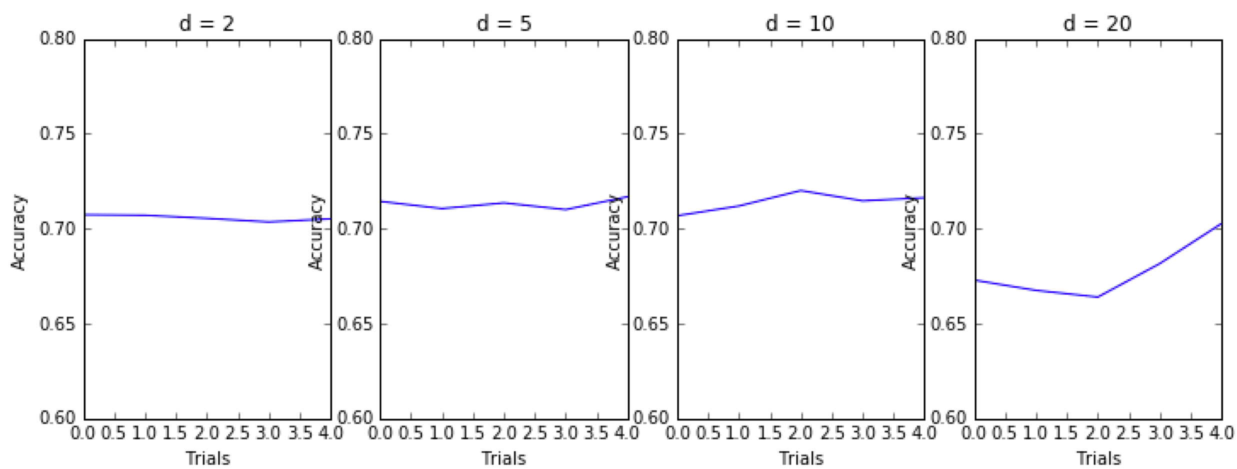
$$u_i = (\sum_{(i,j)\in s} R_{ij}\, V_j^T)(\sum_{(i,j)\in s} V_j V_j^T + \lambda I)^{-1}$$

If u_i is fixed, let $\frac{\partial L}{\partial v_j} = 0$, so we can have:

$$v_i = (\sum_{(i,j)\in s} R_{ij}\, u_i^T)(\sum_{(i,j)\in s} u_i u_i^T + \lambda I)^{-1}$$

So the alternating minimization algorithm is:
1. Random initialize $\{u_i\}$ and $\{v_j\}$.
2. Fix $\{v_j\}$, update $\{u_i\}$ using (1).
3. Fix $\{u_i\}$, update $\{v_j\}$ using (2).
4. If converge, then stop. Otherwise, go to step 2 and repeat.

I use grid search to find the optimal combination of d and λ, the search range for d is [2,5,10,20], and the search range for λ is [0.01,0.1,1,10,100]. The result is as follows:



d=5 and λ = 1 gives the highest validation accuracy.
And the MSE and accuracy with λ = 1 is:

we can see compared with (a), MSE is smaller and accuracy is a little higher.

3.3 Recommending Jokes

I use d=5 and λ = 1 as my final parameters and train on the training data. My code has been attached in the appendix.

My kaggle score is:0.72680

Appendix:
###problem1,kmeans clustering###

```
import scipy.io
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import Grid

image_data = scipy.io.loadmat('/Users/huangshuhui/Google
Drive/study/cs289/hw2017/hw7/hw7_data/mnist_data/images.mat')

####### transform pictures into vector #######

x = np.array(image_data['images'])
image_vector = []
for i in range(x.shape[2]):
   a = x[...,i]
   b = a.flatten()
   image_vector.append(b)
image_vector = np.asarray(image_vector).astype(float)

###### define objective function ######
def J(X,C,mu):
   s = 0
   for i in range(len(C)):
      index = int(C[i])
      s += sum((X[i]-mu[index])**2)
   return s

###### reassignment function ######
def reassign(X,mu):
   C = np.zeros(len(X))
   for i in range(len(X)):
      x = X[i]
      dis_list = np.sum((x-mu)**2,axis=1)
      index = np.argmin(dis_list)
      C[i] = int(index)
```

```
    return C

###### update the mean of clusters ######
def recenter(X,C,mu):
    new_mu = mu
    for k in range(len(mu)):
        new_mu[k] = np.mean(X[C==k],axis=0)
    return new_mu

###### iteration ######
def Kmeans(X,initial_mu,max_iter):
    mu = initial_mu
    C = reassign(X,mu)
    mu = recenter(X,C,mu)
    new_C = reassign(X,mu)
    i=0
    while((new_C!=C).any() and i<=max_iter):
        print(i)
        i+=1
        mu = recenter(X,new_C,mu)
        C = new_C
        new_C = reassign(X,mu)
    return new_C,mu

###### initialize K cluster centers based on Kmeans algorithm #######
def initial_centers(X,k):
    index_0 = np.random.choice(len(X))
    center_0 = X[index_0]
    center_index = [index_0]
    centers = [center_0]
    for m in range(1,k):
        P = np.zeros(len(X))
        for i in range(len(X)):
            x = X[i]
            P[i] = np.min(np.sum((x-centers)**2,axis=1))
        P = P/sum(P)
        new_index = np.random.choice(len(X),p=P)
```

```
        center_index.append(new_index)
        centers.append(X[new_index])
    return centers,center_index

###### plot cluster centers ######

# 5 center
init_mu,mu_index = initial_centers(image_vector,5)
final_c, final_mu = Kmeans(image_vector,init_mu,100)
f, ax = plt.subplots(1, 5,figsize=(12,3))
for i in range(5):
    ax[i].imshow(final_mu[i].reshape(28,28))
    ax[i].get_xaxis().set_visible(False)
    ax[i].get_yaxis().set_visible(False)
plt.show()

# 10 center
init_mu,mu_index = initial_centers(image_vector,10)
final_c, final_mu = Kmeans(image_vector,init_mu,200)
fig = plt.figure()
grid = Grid(fig, rect=111, nrows_ncols=(2,5),axes_pad=0.1)
for i in range(10):
    grid[i].imshow(final_mu[i].reshape(28,28))
    grid[i].get_xaxis().set_visible(False)
    grid[i].get_yaxis().set_visible(False)
plt.show()

# 20 center
init_mu,mu_index = initial_centers(image_vector,20)
final_c, final_mu = Kmeans(image_vector,init_mu,400)
fig = plt.figure()
grid = Grid(fig, rect=111, nrows_ncols=(4,5),axes_pad=0.1)
for i in range(20):
    grid[i].imshow(final_mu[i].reshape(28,28))
    grid[i].get_xaxis().set_visible(False)
    grid[i].get_yaxis().set_visible(False)
plt.show()
```

```
###problem3 Low-Rank Approximation###
from PIL import Image
import numpy as np
from numpy import *
import matplotlib.pyplot as plt
import math

im = Image.open('/Users/huangshuhui/Google
Drive/study/cs289/hw2017/hw7/hw7_data/low-rank_data/face.jpg')
data_origi = np.array(im)
u,sig,v=np.linalg.svd(data_origi)


##rand k approxiamation##
def krankimage(rank,u,sig,v):
    sblank=np.zeros((len(u),len(v)))
    for i in range(rank):
        sblank[i][i]=sig[i]
    newim=np.dot((np.dot(u,sblank)),v)
    plt.imshow(newim)
    plt.title('rank-'+str(rank)+' approximation on the image')

krankimage(5, u,sig,v)
krankimage(20, u,sig,v)
krankimage(100, u,sig,v)

##Plot the Mean Squared Error (MSE)##
mse=[0]*101
for i in range(1,101):
    sblank=np.zeros((len(u),len(v)))
    for j in range(i):
        sblank[j][j]=sig[j]
    newim=np.dot((np.dot(u,sblank)),v)
    mse[i] = math.sqrt(sum((data_origi - newim) ** 2))
mse[0]=None
```

```
plt.plot(mse)
plt.title('MSE from rank 1 to rank 100')

##perform the same rank-5, rank-20, and rank-100 approximation on low rank
data=sky:jpg#

im2 = Image.open('/Users/huangshuhui/Google
Drive/study/cs289/hw2017/hw7/hw7_data/low-rank_data/sky.jpg')
data_origi2 = np.array(im2)
u2,sig2,v2=np.linalg.svd(data_origi2)
krankimage(5, u2,sig2,v2)
krankimage(20, u2,sig2,v2)
krankimage(100, u2,sig2,v2)



###problem3###

import numpy as np
import scipy.io
import matplotlib.pyplot as plt
import csv


data = scipy.io.loadmat('/Users/huangshuhui/Google
Drive/study/cs289/hw2017/hw7/hw7_data/joke_data/joke_train.mat')
joke_train = data['train']
joke_validate = np.loadtxt('/Users/huangshuhui/Google
Drive/study/cs289/hw2017/hw7/hw7_data/joke_data/validation.txt',dtype=int,d
elimiter=",")
joke_train_zero = np.nan_to_num(joke_train)
user_list = np.unique(joke_validate[:,0])
##Latent Factor Model ##

## (1) use PCA to approximate rating matrix
def PCA(train_X_zero, d):
    U, s, V = np.linalg.svd(train_X_zero, full_matrices=False)
```

```
    X_approx = np.zeros((train_X_zero.shape[0],train_X_zero.shape[1]))
    for i in range(d):
        X_approx += s[i]*(U[:,i].reshape(-1,1).dot(V[i,:].reshape(1,-1)))
    return X_approx

def MSE(train_X_nan,X_approx):
    return np.sum((np.nan_to_num(train_X_nan-X_approx))**2)

def predict_accuracy(X_approx):
    u = joke_validate[:,0]-1
    j = joke_validate[:,1]-1
    rate = joke_validate[:,2]
    predicted_rate = X_approx[u,j]>0
    return (sum(rate==predicted_rate)/len(rate))

MSE_ls = []
Accuracy_ls =[]
for d in [2,5,10,20]:
    X_approx = PCA(joke_train_zero,d)
    MSE_ls.append(MSE(joke_train,X_approx))
    Accuracy_ls.append(predict_accuracy(X_approx))
print("The perfomance of PCA with different d")
print("MSE:")
print(MSE_ls)
print("Prediction accuracies:")
print(Accuracy_ls)

f,ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.plot([2,5,10,20], MSE_ls, 'ro-')
ax2.plot([2,5,10,20], Accuracy_ls, 'bo-')
ax1.set_xlabel('d')
ax1.set_ylabel('MSE', color='r')
ax2.set_ylabel('Accuracy', color='b')
plt.title("Perfomance of PCA with different d")
plt.show()
```

```
## (2) minimize MSE only on rated jokes

def new_U(R,V,lamda,d):
    new_U = np.zeros((R.shape[0],d))
    for i in range(new_U.shape[0]):
        num_index = np.isfinite(R[i])
        v = V[:,num_index]
        new_U[i] = np.sum(R[i][num_index].reshape(-
1,1)*v.T,axis=0).dot(np.linalg.inv(v.dot(v.T)+lamda*np.eye(d)))
    return new_U

def new_V(R,U,lamda,d):
    new_V = np.zeros((d,R.shape[1]))
    for j in range(new_V.shape[1]):
        num_index = np.isfinite(R[:,j])
        u = U[num_index,:]
        new_V[:,j] =
np.linalg.inv(u.T.dot(u)+lamda*np.eye(d)).dot(np.sum(R[:,j][num_index].reshape(
-1,1)*U[num_index,:],axis=0).reshape(-1,1)).reshape(1,-1)
    return new_V

def L(U,V,R,lamda):
    return np.sum((np.nan_to_num(U.dot(V)-
R))**2)+lamda*np.sum(U**2)+lamda*np.sum(V**2)

def Iteration(lamda,d,R):
    U = np.random.randn(R.shape[0],d)
    V = np.random.randn(d,R.shape[1])
    L_list = [L(U,V,R,lamda)]
    i=1
    while(len(L_list)<5 or abs(L_list[-1]-L_list[-2])>0.005*L_list[-2]):
        i+=1
        U = new_U(R,V,lamda,d)
        V = new_V(R,U,lamda,d)
        L_list.append(L(U,V,R,lamda))
    return U,V,L_list
```

```
R = joke_train

###### Grid search to find the optimal d and lamda #######

d_ls = [2,5,10,20]
lamda_ls = [0.01,0.1,1,10,100]
#lamda_ls = [1]
acc_all = []
f, axarr = plt.subplots(1, len(d_ls),figsize=(12,4))
for i in range(len(d_ls)):
    d=d_ls[i]
    print(d)
    accuracy_ls = []
    mse_ls=[]
    for lamda in lamda_ls:
        print(lamda)
        U,V,L_list = Iteration(lamda,d,R)
        accuracy_ls.append(predict_accuracy(U.dot(V)))
        mse_ls.append(MSE(joke_train,U.dot(V)))
    print(accuracy_ls)
    print(mse_ls)
    axarr[i].plot(range(len(lamda_ls)), accuracy_ls)
    axarr[i].set_xlabel('Trials')
    axarr[i].set_ylabel('Accuracy')
    axarr[i].set_title('d = '+str(d))
    axarr[i].set_ylim(0.6,0.8)
plt.show()

##plot d-5,lambda=1 mse and acuracy##

f,ax1 = plt.subplots()
ax2 = ax1.twinx()
ax1.plot([2,5,10,20], mse_ls, 'ro-')
ax2.plot([2,5,10,20], accuracy_ls, 'bo-')
ax1.set_xlabel('d')
ax1.set_ylabel('MSE', color='r')
ax2.set_ylabel('Accuracy', color='b')
```

```
plt.title("Minimize the MSE only on rated joke")
plt.show()
###### Final Kaggle submission ######

U,V,L_list = Iteration(1,5,R)
print(predict_accuracy(U.dot(V)))

joke_query = np.loadtxt('/Users/huangshuhui/Google Drive/study/cs289/hw2017/hw7/hw7_data/joke_data/query.txt',dtype=int,delimiter=",")
u = joke_query[:,1]-1
v = joke_query[:,2]-1
predicted_rate = (U.dot(V)[u,v]>0).reshape((-1,1)).astype(int)
result = [[i + 1, predicted_rate[i][0]] for i in range(len(predicted_rate))]
with open('jokes.csv', 'w') as f:
    writer = csv.writer(f)
    writer.writerows(result)
```