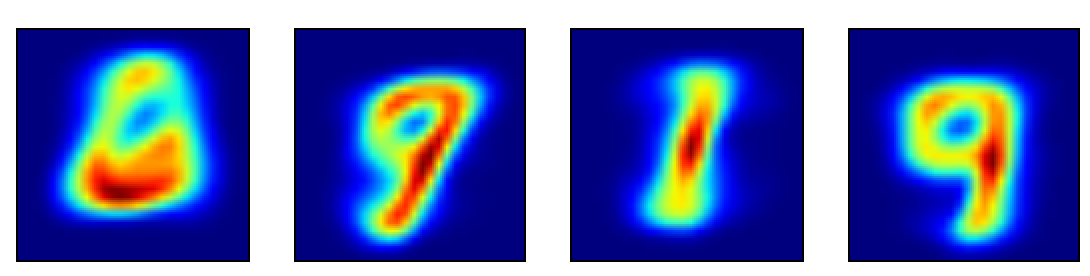
CS 189/289 Introduction to Machine Learning

Homework 7: Unsupervised

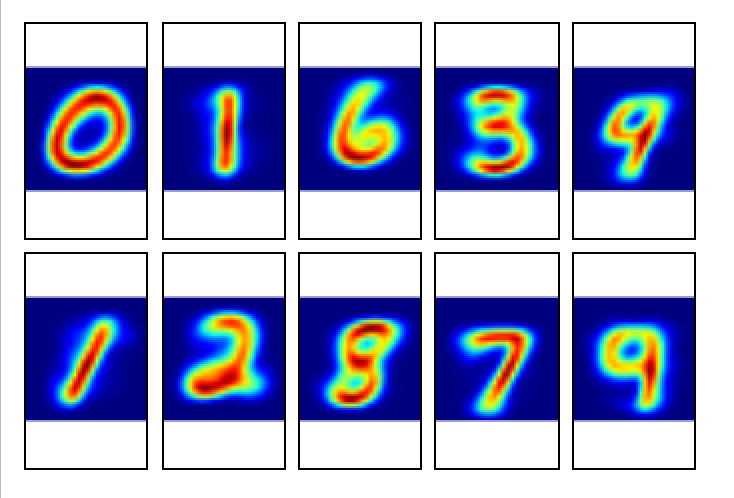
Shuhui Huang SID:3032129712

1. k-means clustering
2. 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:

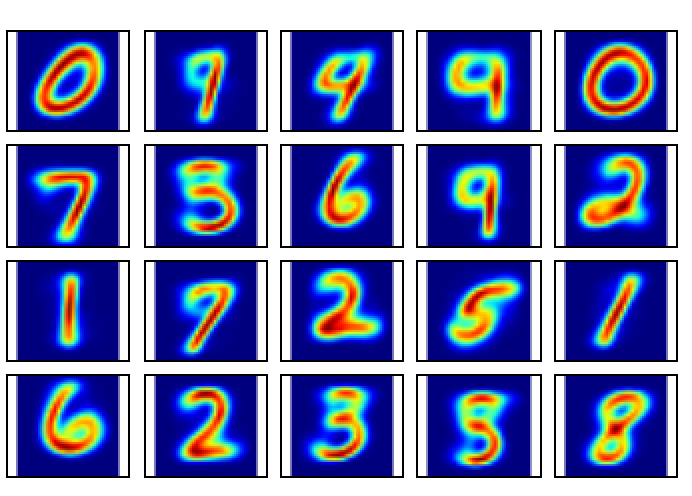
K=5:



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.

1. Low-Rank Approximation

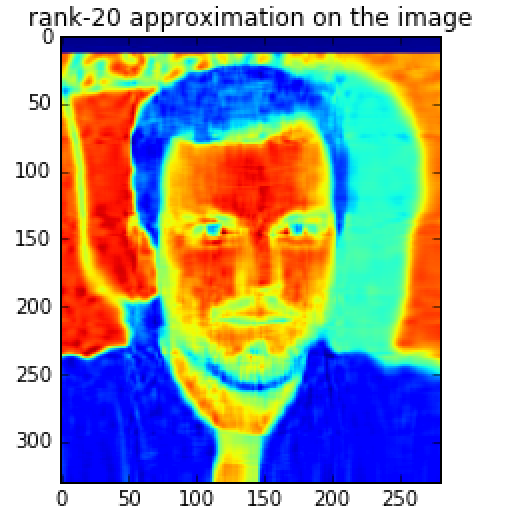
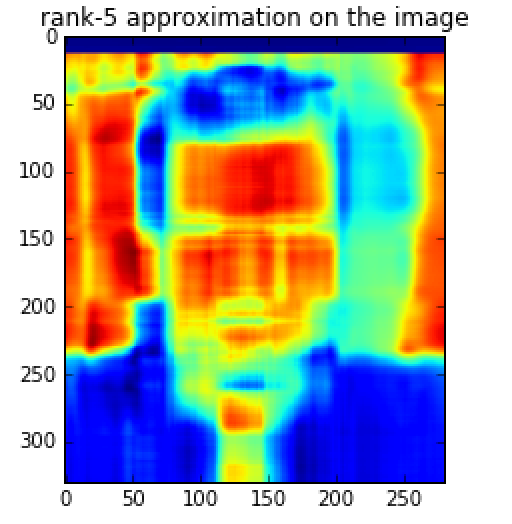
(a).

Let D=, . So we want to get

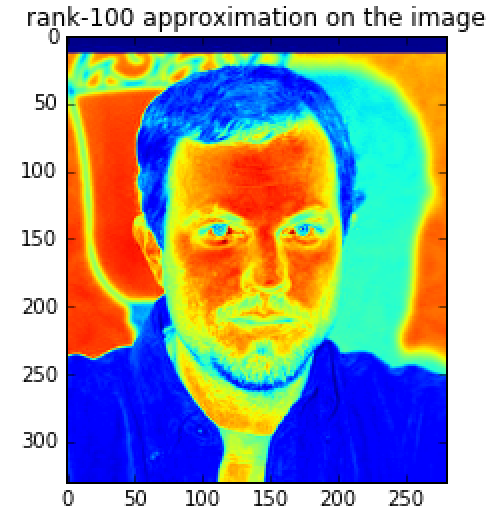
min

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:

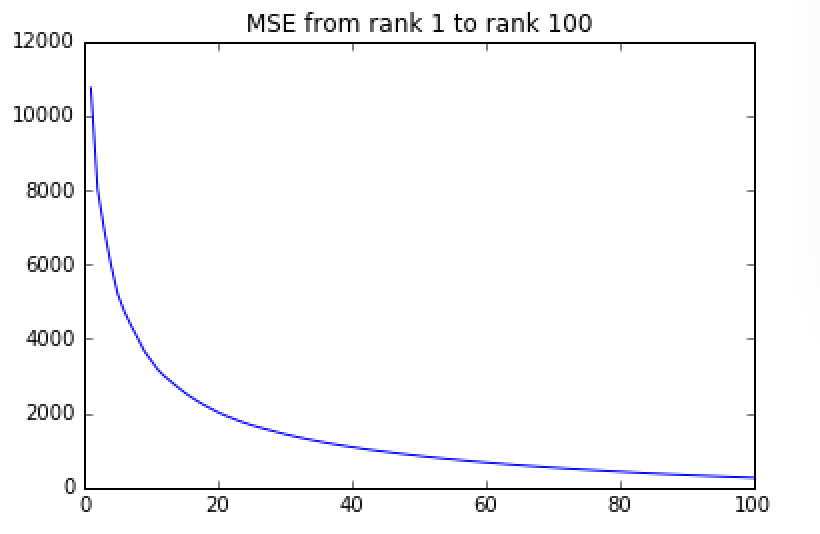


Rank=100:



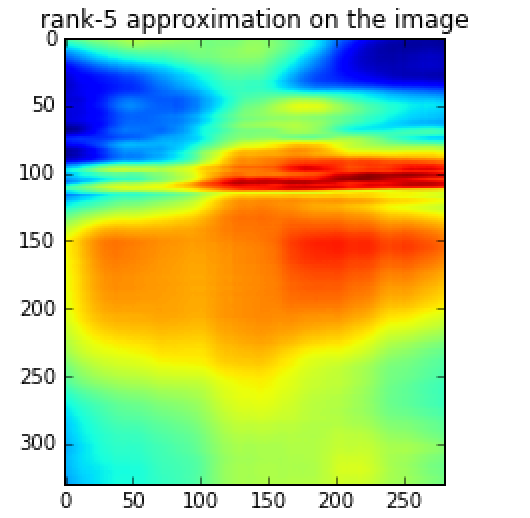
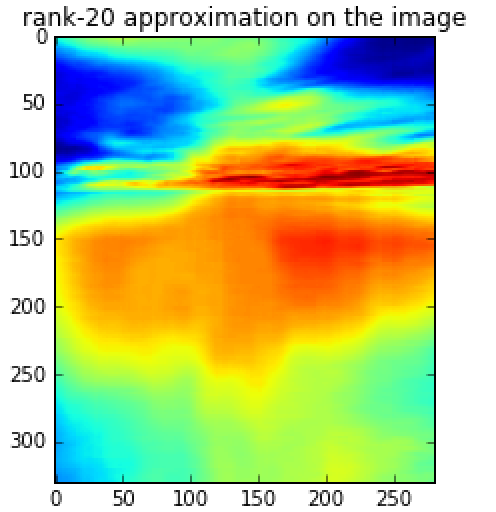
(b).

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 :

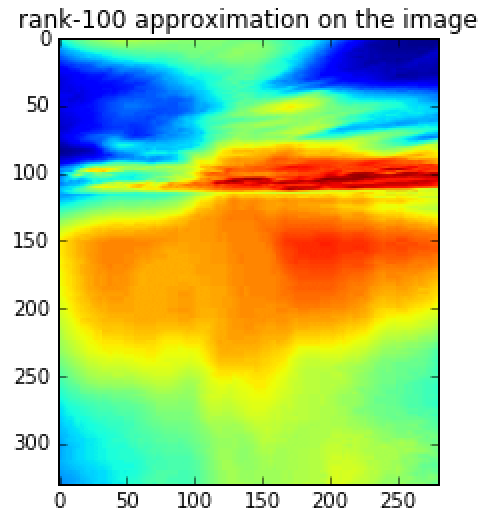


(c) The plot for sky is:

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.

1. Joke Recommender System

3.2 Latent Factor Model

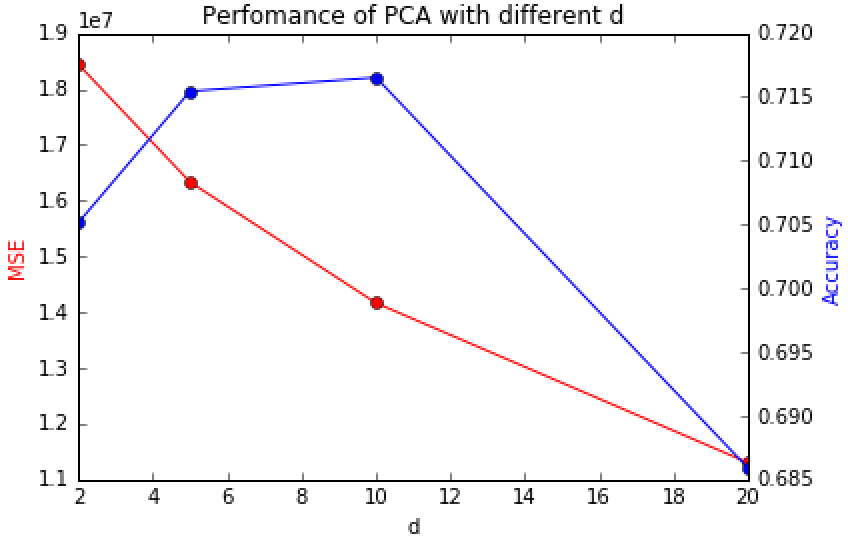
(a). After replacing all missing values with 0 and using PCA to learn the vector representation, I get the following MSEs and prediction accuracies with different value of d (2,5,10,20):

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 is fixed, let , so we can have:

If is fixed, let , so we can have:

So the alternating minimization algorithm is:

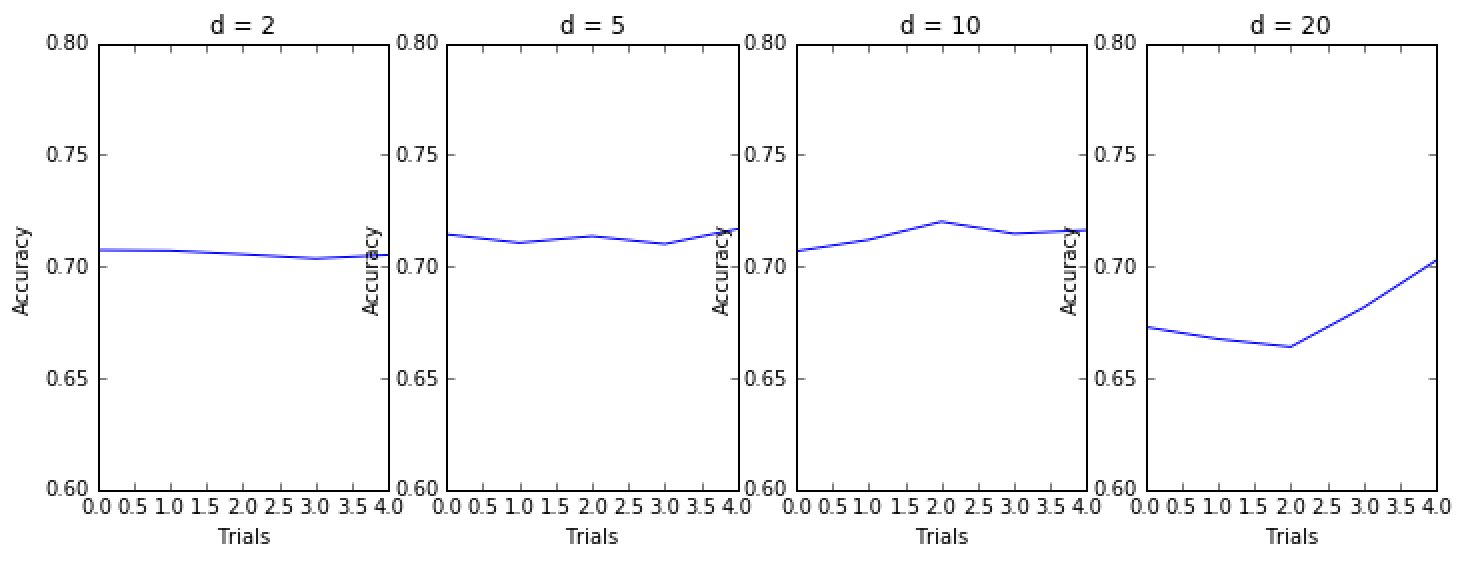
1. Random initialize {} and {}.

2. Fix {}, update {} using (1).

3. Fix {}, update {} using (2).

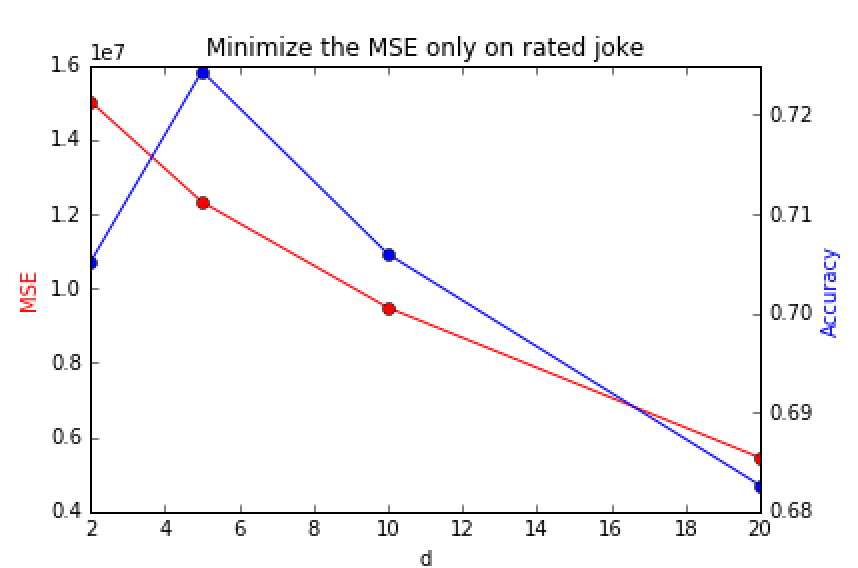
4. If converge, then stop. Otherwise, go to step 2 and repeat.

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,delimiter=",")

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)