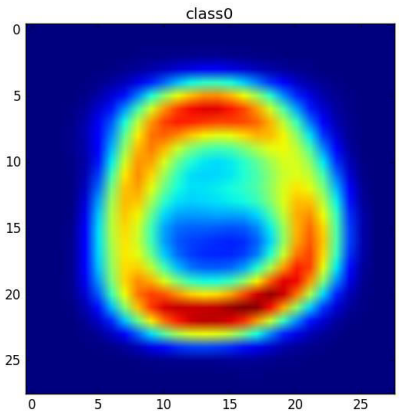
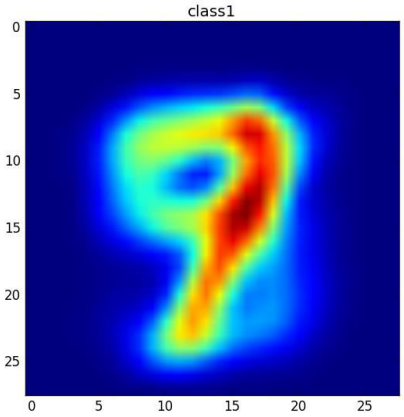
CS189 HW 7

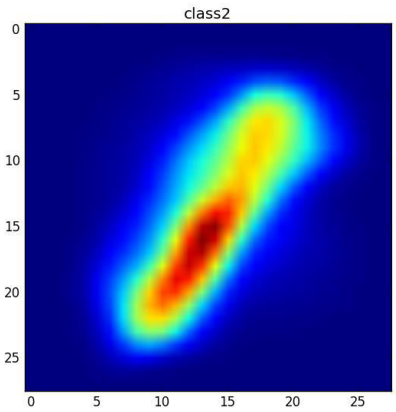
Yong-Chan Shin

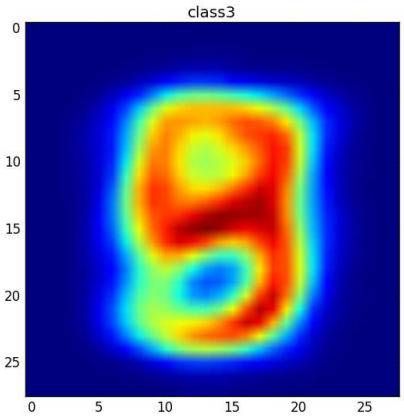
1. K-means clustering

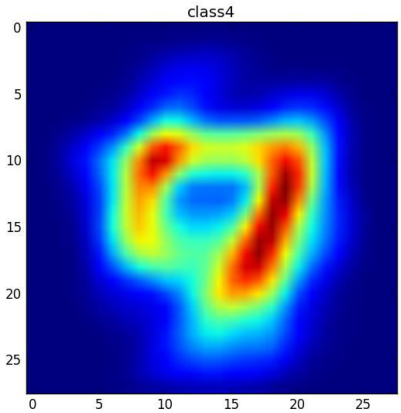
   (a)

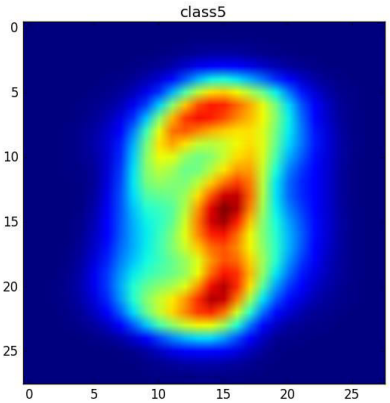


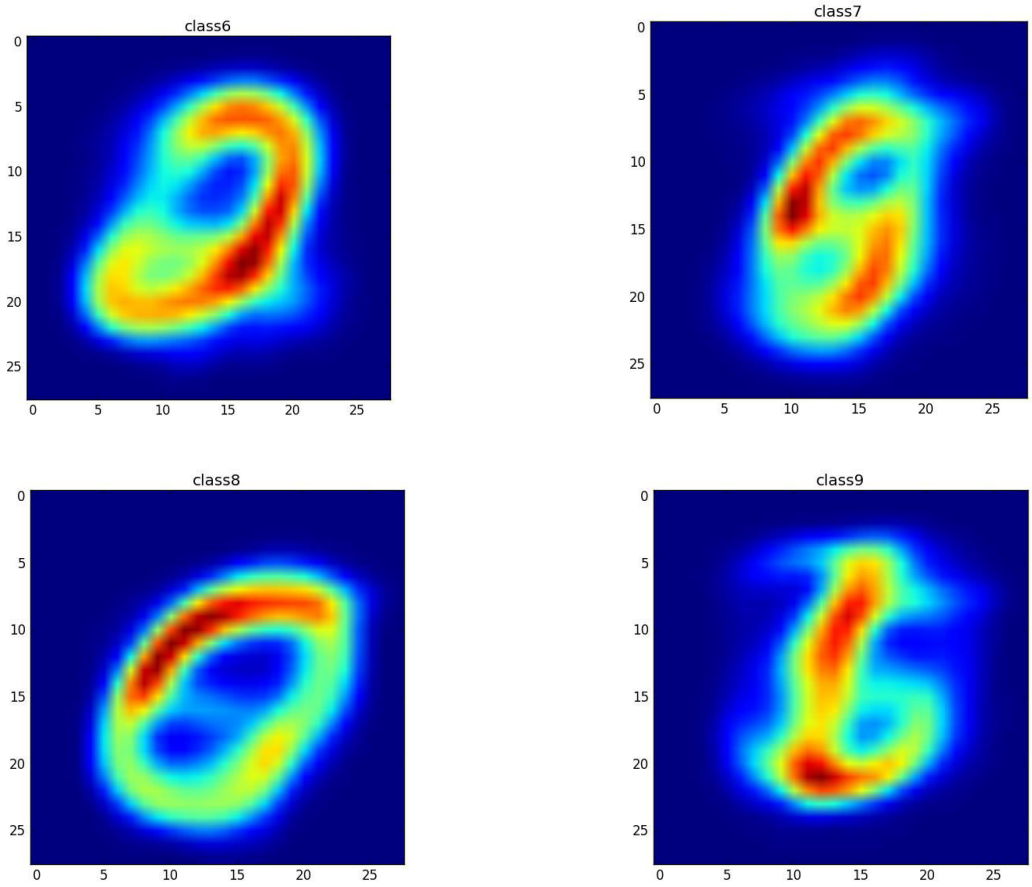












Most of the cluster centers look quite close to its class digit, but some of them look different. It seems to be due to the initial value of the centers.

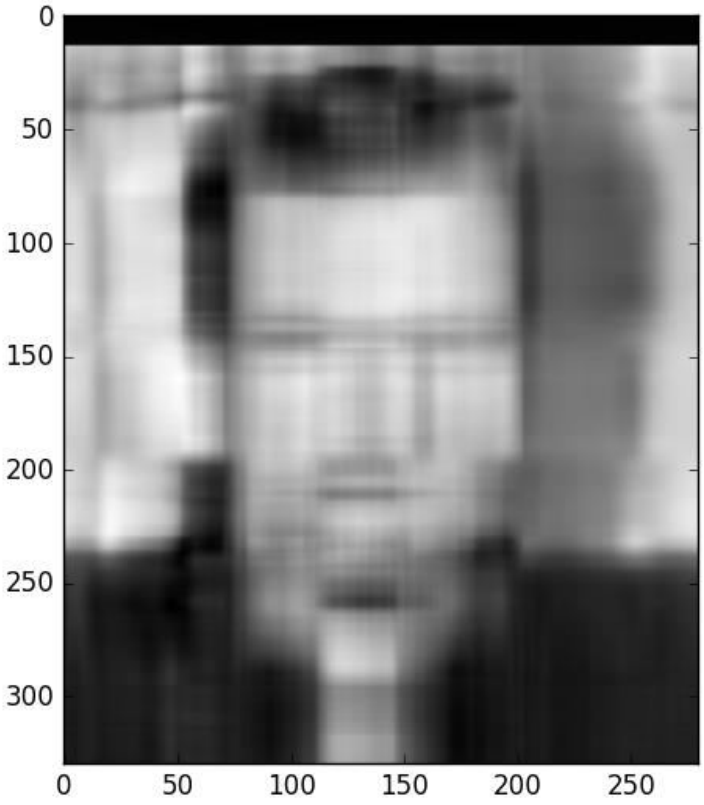
<Q1 code>

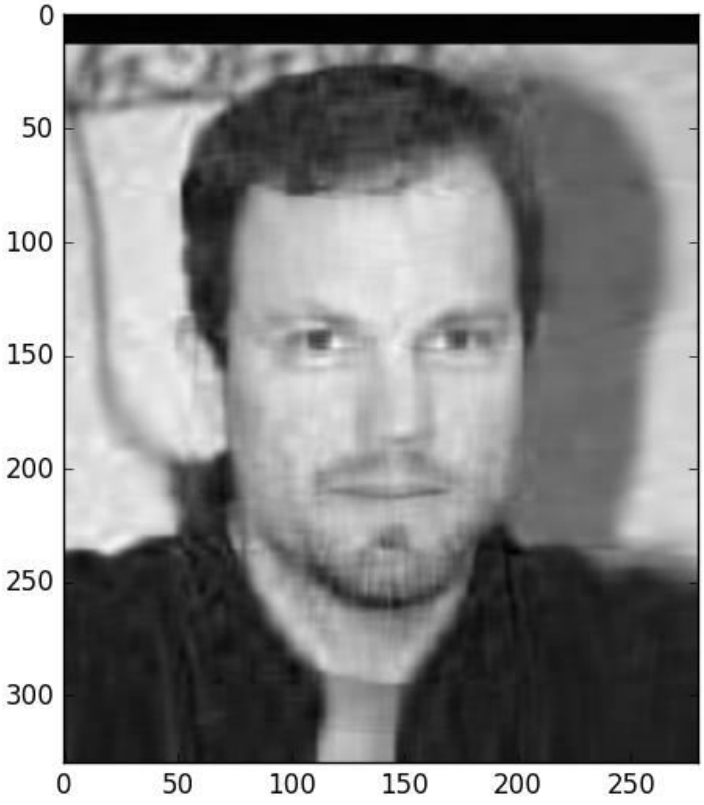
```
def kmeans(data, k):
        #initialize mu with random numbers
    mu = [np.random.rand(data[0].shape[0])*255/2.0 for i in range(k)]
    k_classes = [[] for i in range(k)]
    changed = True
    while changed:
        for sample in data:
            index, closest, dif   = 0, mu[0], float('inf')
            for i in range(len(mu)):
                dist = np.linalg.norm(mu[i] - sample)
                if dist < dif:
                    index, closest, dif = i, mu[i], dist

            k_classes[index].append(sample)

        #update centroid
        unchanged_count = 0
        for i in range(len(mu)):
            new_mean = np.mean(k_classes[i], axis=0)
            #print("new_mean",new_mean)
            #print("old mean",mu[i])
            if new_mean.all() == mu[i].all():
                unchanged_count+=1
            mu[i] = new_mean
```

```
            if unchanged_count == k:
                        changed = False

        return mu, k_classes


def q1():
            train_data = scipy.io.loadmat('hw7_data/mnist_data/images.mat')
            train_raw_set = train_data['images'].T
            train_set = []
            for i in range(len(train_raw_set)):
                  train_set.append(train_raw_set[i].T.ravel())
            train_set = np.array(train_set)


            k = 10
            mu, classes = kmeans(train_set, k)
            for i in range(k):
                  plt.imshow(mu[i].reshape(28,28))
                  print("class", i, "count:", len(classes[i]))
                  plt.title("class" + str(i))
                  #plt.savefig("class" + str(i) + ".png")
                  plt.show()
```

2. Low-Rank Approximation (From the first to the third, the image is based on rank=5, 20, and 100)

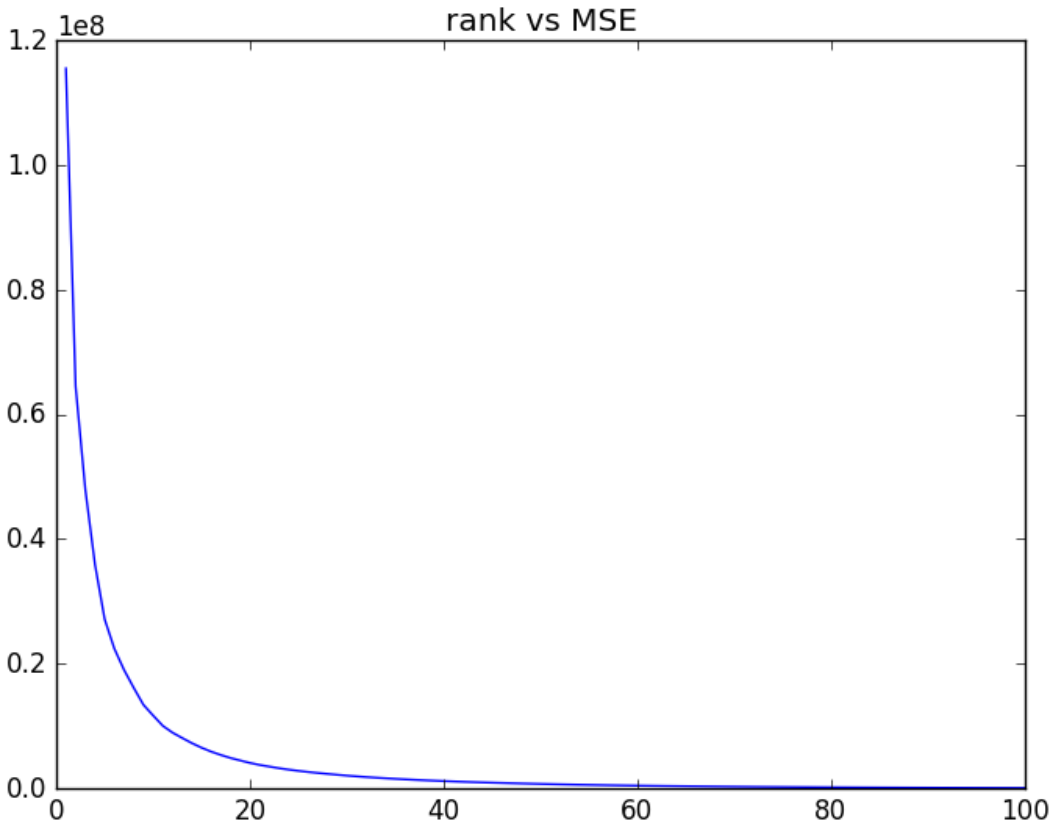
(a)



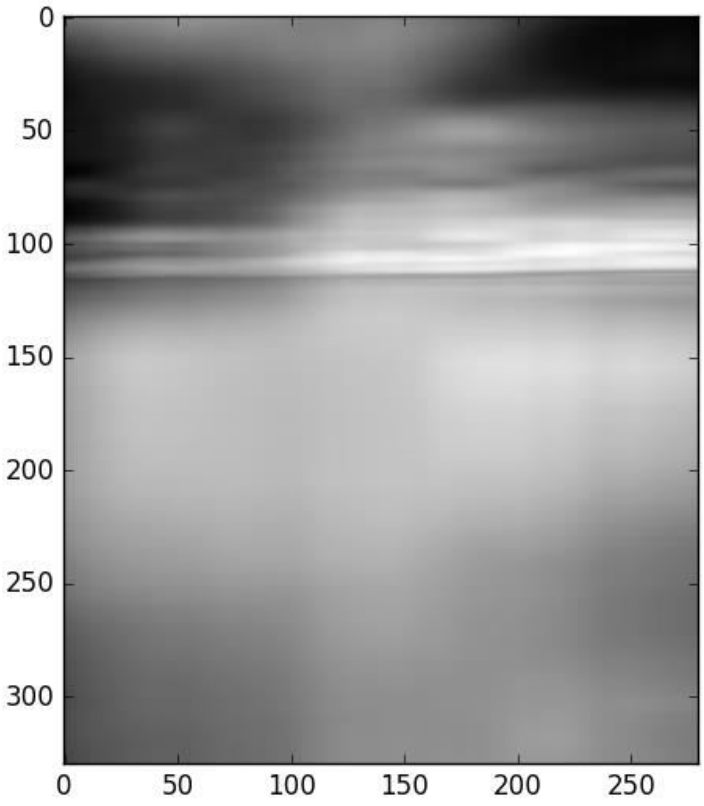

0
50
100
150
200
250
300
0
50
100
150
200
250

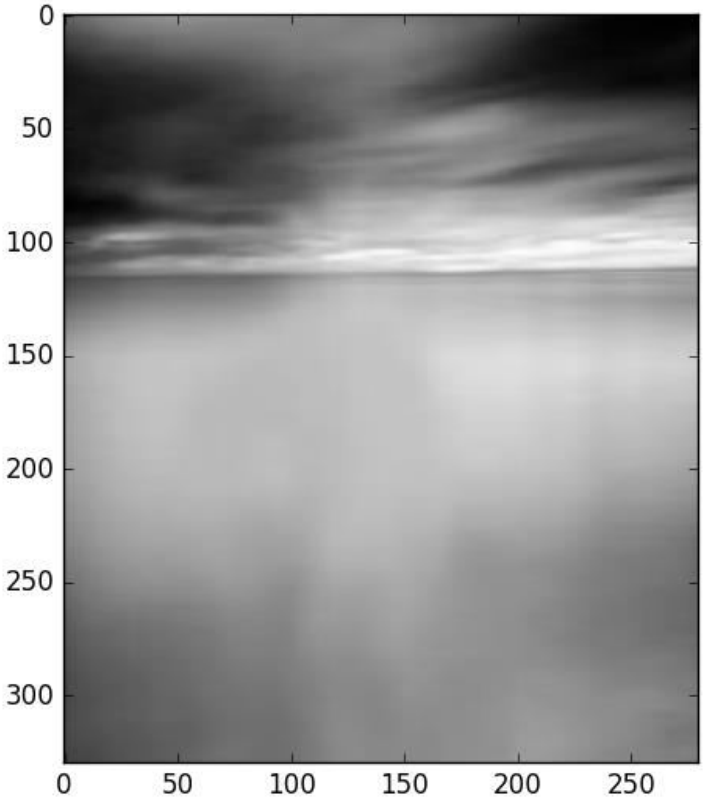

0
50
100
150
200
250
300
0
50
100
150
200
250

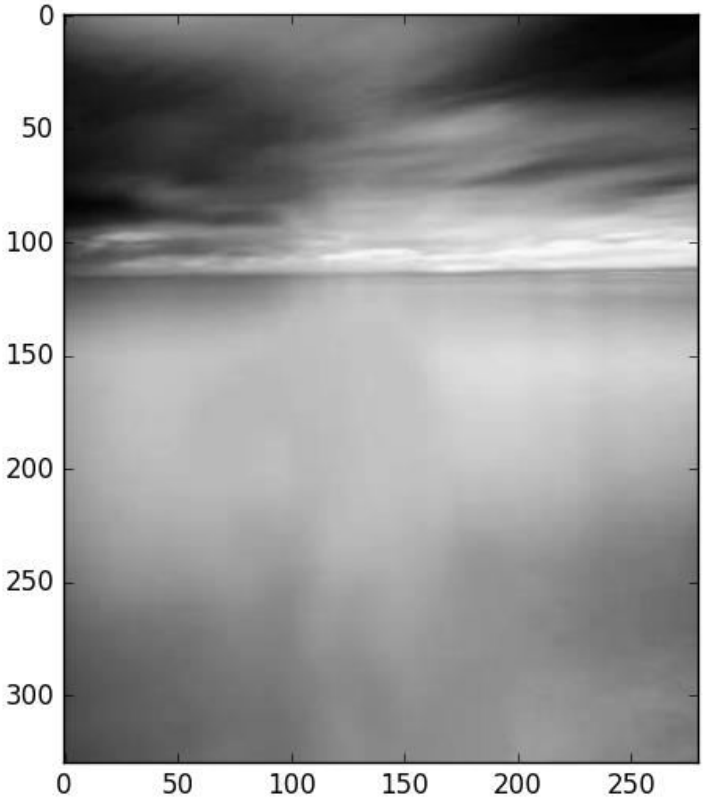
(b)



(c)




0
50
100
150
200
250
300
0
50
100
150
200
250



(d) It was hard to differentiate between the original image and low rank approximation from rank 15 for sky image. For face image, the lowest rank was 60.

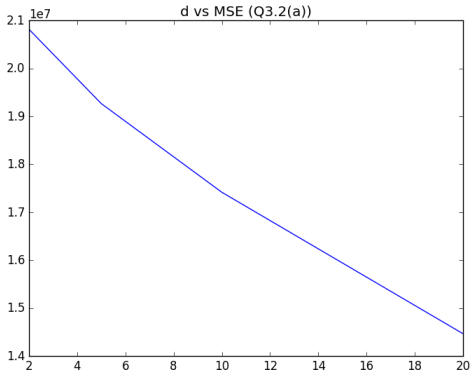
<Q3 code>

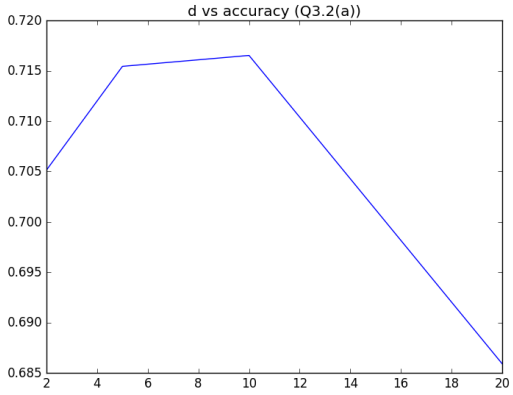
```
def q2():
    def q2a(face,U,s,V):
        rank_list = range(50,60)
        #rank_list = [5,20,100]
        for rank in rank_list:
            S = np.diag(np.append(s[:rank], np.zeros(len(s)-rank)))
            face = (U.dot(S)).dot(V)
            plt.title("rank:"+str(rank))
            plt.imshow(face, cmap='gray')
            plt.show()
            #face_new_img.show()
    def q2b(face,U,s,V):
        rank_list = range(1,101)
        MSE_list = []
        for rank in rank_list:
            S = np.diag(np.append(s[:rank], np.zeros(len(s)-rank)))
            D_hat = (U.dot(S)).dot(V)
            tot = 0
            for i in range(D_hat.shape[0]):
                for j in range(D_hat.shape[1]):
                    tot+=(face[i,j]-D_hat[i,j])**2
            MSE_list.append(tot)
        plt.title("rank vs MSE")
        plt.plot(rank_list,MSE_list)
```

```
            plt.show()


face_img = Image.open("hw7_data/low-rank_data/face.jpg")
face = np.array(face_img)
sky_img = Image.open("hw7_data/low-rank_data/sky.jpg")
sky = np.array(sky_img)
U_face, s_face, V_face = np.linalg.svd(face, full_matrices=False)
# print("Ufaceshape",U_face.shape)
# print("sface len",len(s_face))
# print("V_faceshape",V_face.shape)
U_sky, s_sky, V_sky = np.linalg.svd(sky, full_matrices=False)
q2a(face,U_face,s_face,V_face)
q2b(face,U_face,s_face,V_face)
q2a(sky,U_sky, s_sky, V_sky) # Question 2 (c)
```

3. Joke Recommender System

(a) (i)

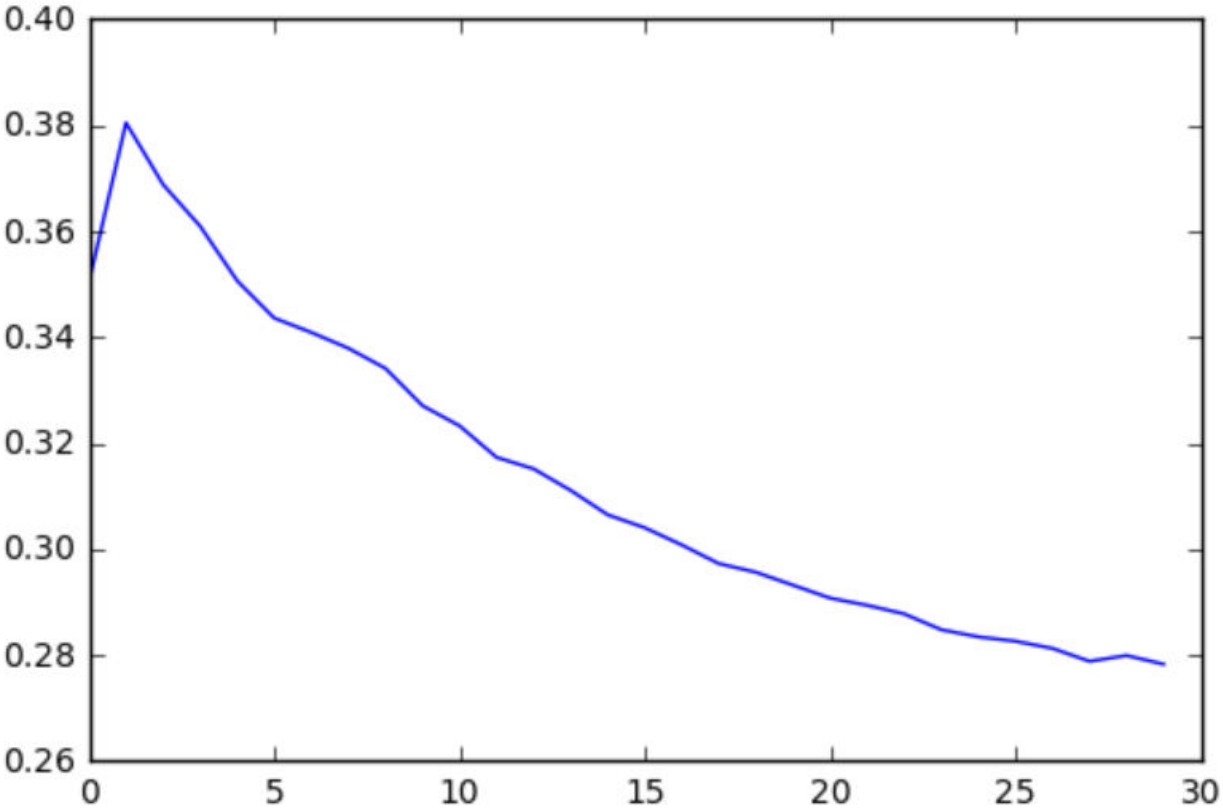





```
validation score for k=2: 0.7051490514905149
validation score for k=5: 0.7154471544715447
validation score for k=10: 0.7165311653116531
validation score for k=20: 0.6859078590785908
```

As d increases, MSE always decreases. However, accuracy goes down for k=20, which seems to be due to overfit.

(ii) MSE decreases as we perform more iteration, which gives similar graph with part (a). While for (a), the accuracy decreases for some large rank, the accuracy continuously goes up for (b), as the graph below says (the graph below was measuring the error rate, and the error rate continuously goes down except for the very initial iteration).



3.3 Kaggle ID: YONG-CHAN_SHIN

Score: 0.72807

<Q3 code>

```
def q3a():
    # 3.1
    train_set = scipy.io.loadmat('hw7_data/joke_data/joke_train.mat')['train']
    train_mat = np.array(train_set[:,:])
    #i,j,s
    validation = genfromtxt('hw7_data/joke_data/validation.txt', delimiter=',')
    validation[validation == 0] = -1
    validation = validation - np.array([1,1,0]) # i,j,s
    #id,i,j
    query = genfromtxt('hw7_data/joke_data/query.txt', delimiter=',')
    query = query - np.array([0,1,1])
    # 3.2 (a)
    nan_index = np.isnan(train_mat)
    train_mat[nan_index] = 0
    U, s, V = np.linalg.svd(train_mat, full_matrices=False)
    d_list = [2,5,10,20]
    error_a=[]
    score_a=[]
    for d in d_list:
        S = np.diag(np.append(s[:d], np.zeros(len(s)-d)))
        pred = U.dot(S.dot(V))
        error = MSE(pred,train_mat)
        error_a.append(error)
        v_score = compute_score(validation,pred)
```

```
            print("validation score:",v_score)
            score_a.append(v_score)
        plt.title("d vs MSE (Q3.2(a))")
        plt.plot(d_list,error_a)
        plt.show()
        plt.title("d vs accuracy (Q3.2(a))")
        plt.plot(d_list,score_a)
        plt.show()

def q3b():
        train_set = scipy.io.loadmat('hw7_data/joke_data/joke_train.mat')
        train_mat = train_set['train']

        #load validation set.
        valid_data = genfromtxt('hw7_data/joke_data/validation.txt', delimiter=',')
        valid_data[valid_data == 0] = -1
        valid_data = valid_data - np.array([1,1,0])

        train_mat_zero = np.array(train_mat[:,:])
        train_mat_zero_nan_index = np.isnan(train_mat_zero)
        train_mat_zero[train_mat_zero_nan_index] = 0

        U, s, V = np.linalg.svd(train_mat_zero, full_matrices=False)
        nan_list = np.isnan(train_mat)
```

```
class latent_factor_model:
    def __init__(self, lamb):
        self.lamb = lamb
        self.u = np.random.random(U.shape)
        self.v = np.random.random(V.shape)

    def grad_u(self):
        vv = []
        for j in range(len(self.v)):
            vv.append(np.outer(self.v[j], self.v[j]))

        for i in range(len(self.u)):
            inv = sum([vv[j] for j in range(len(self.v)) if not nan_list[i,j]])
            inv += self.lamb*np.identity(len(inv))
            vR = sum([self.v[j]*train_mat[i,j] for j in range(len(self.v)) if not nan_list[i,j]])
            self.u[i] =  np.linalg.inv(inv).dot(vR)

    def grad_v(self):
        uu = []
        for i in range(len(self.u)):
            uu.append(np.outer(self.u[i], self.u[i]))

        for j in range(len(self.v)):
            inv = sum([uu[i] for i in range(len(self.u)) if not nan_list[i,j]])
```

```
                inv += self.lamb*np.identity(len(inv))
                uR = sum([self.u[i]*train_mat[i,j] for i in range(len(self.u)) if not nan_list[i,j]])
                self.v[j] =  np.linalg.inv(inv).dot(uR)

        def train(self):
                self.grad_u()
                self.grad_v()

        def predict(self, user, joke):
                return self.u[int(user)].dot(self.v[int(joke)])

lfm = latent_factor_model(10)
lfm.train()
errors = []
for i in range(30):
        print("Epoch", i)
        curr_error = 0
        predictions = np.array([[0 for _ in range(train_mat.shape[1])] for _ in range(train_mat.shape[0])])
        for example in valid_data:
                user, joke, actual = example[0], example[1], example[2]
                pred = lfm.predict(user,joke)
                predictions[int(user),int(joke)] = pred
                curr_error += np.sign(pred) != np.sign(actual)*1
```

```
                curr_error = curr_error/len(valid_data)
                errors.append(curr_error)
                print("MSE:",MSE(predictions,train_mat))
                print(errors[i])
                lfm.train()

        query = genfromtxt('hw7_data/joke_data/query.txt', delimiter=',')
        query = query - np.array([0,1,1])

        predictions = []
        for row in query:
                score = lfm.predict(row[1], row[2])
                predictions.append((score > 0)*1)
        preds = np.array(predictions)

        table = {"Category":preds.astype(int),"Id":np.arange(1,len(preds)+1)}
        output = pd.DataFrame(data=table)
        output.to_csv("kaggle_ycls_hw7.csv", index=False)
        print("csv created")
        plt.plot(errors)

def compute_score(validation,pred):
        correct_count = 0
        for i,j,s in validation:
                prediction = pred[int(i),int(j)]
```

```
                #print("pred:",pred[int(i),int(j)])
                if np.sign(int(s))==np.sign(prediction)*1:
                        correct_count+=1
        return float(correct_count)/len(validation)


def MSE(pred,train_mat):
        tot=0
        for i in range(train_mat.shape[0]):
                for j in range(train_mat.shape[1]):
                        if not np.isnan(train_mat[i,j]):
                                tot+=(pred[i,j]-train_mat[i,j])**2
        return tot
```

<Full code>

```
import scipy.io
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import random
from PIL import Image
from numpy import genfromtxt
#from scipy.ndimage import imread


def main():
	#q1()
	#q2()
	q3a()
	#q3b()


def kmeans(data, k):
	#initialize mu with random numbers
    mu = [np.random.rand(data[0].shape[0])*255/2.0 for i in range(k)]
    k_classes = [[] for i in range(k)]
    changed = True
    while changed:
	for sample in data:
	    index, closest, dif    = 0, mu[0], float('inf')
```

```
            for i in range(len(mu)):
                dist = np.linalg.norm(mu[i] - sample)
                if dist < dif:
                    index, closest, dif = i, mu[i], dist

            k_classes[index].append(sample)

        #update centroid
        unchanged_count = 0
        for i in range(len(mu)):
            new_mean = np.mean(k_classes[i], axis=0)
            #print("new_mean",new_mean)
            #print("old mean",mu[i])
            if new_mean.all() == mu[i].all():
                unchanged_count+=1
            mu[i] = new_mean
        if unchanged_count == k:
                changed = False

    return mu, k_classes


def q1():
        train_data = scipy.io.loadmat('hw7_data/mnist_data/images.mat')
        train_raw_set = train_data['images'].T
        train_set = []
```

```
        for i in range(len(train_raw_set)):
            train_set.append(train_raw_set[i].T.ravel())
        train_set = np.array(train_set)


        k = 10
        mu, classes = kmeans(train_set, k)
        for i in range(k):
            plt.imshow(mu[i].reshape(28,28))
            print("class", i, "count:", len(classes[i]))
            plt.title("class" + str(i))
            #plt.savefig("class" + str(i) + ".png")
            plt.show()


def q2():
        def q2a(face,U,s,V):
                rank_list = range(50,60)
                #rank_list = [5,20,100]
                for rank in rank_list:
                        S = np.diag(np.append(s[:rank], np.zeros(len(s)-rank)))
                        face = (U.dot(S)).dot(V)
                        plt.title("rank:"+str(rank))
                        plt.imshow(face, cmap='gray')
                        plt.show()
                        #face_new_img.show()
        def q2b(face,U,s,V):
```

```
        rank_list = range(1,101)
        MSE_list = []
        for rank in rank_list:
                S = np.diag(np.append(s[:rank], np.zeros(len(s)-rank)))
                D_hat = (U.dot(S)).dot(V)
                tot = 0
                for i in range(D_hat.shape[0]):
                        for j in range(D_hat.shape[1]):
                                tot+=(face[i,j]-D_hat[i,j])**2
                MSE_list.append(tot)
        plt.title("rank vs MSE")
        plt.plot(rank_list,MSE_list)
        plt.show()


face_img = Image.open("hw7_data/low-rank_data/face.jpg")
face = np.array(face_img)
sky_img = Image.open("hw7_data/low-rank_data/sky.jpg")
sky = np.array(sky_img)
U_face, s_face, V_face = np.linalg.svd(face, full_matrices=False)
# print("Ufaceshape",U_face.shape)
# print("sface len",len(s_face))
# print("V_faceshape",V_face.shape)
U_sky, s_sky, V_sky = np.linalg.svd(sky, full_matrices=False)
q2a(face,U_face,s_face,V_face)
q2b(face,U_face,s_face,V_face)
```

```
    q2a(sky,U_sky, s_sky, V_sky) # Question 2 (c)


def q3a():
    # 3.1
    train_set = scipy.io.loadmat('hw7_data/joke_data/joke_train.mat')['train']
    train_mat = np.array(train_set[:,:])
    #i,j,s
    validation = genfromtxt('hw7_data/joke_data/validation.txt', delimiter=',')
    validation[validation == 0] = -1
    validation = validation - np.array([1,1,0]) # i,j,s
    #id,i,j
    query = genfromtxt('hw7_data/joke_data/query.txt', delimiter=',')
    query = query - np.array([0,1,1])
    # 3.2 (a)
    nan_index = np.isnan(train_mat)
    train_mat[nan_index] = 0
    U, s, V = np.linalg.svd(train_mat, full_matrices=False)
    d_list = [2,5,10,20]
    error_a=[]
    score_a=[]
    for d in d_list:
        S = np.diag(np.append(s[:d], np.zeros(len(s)-d)))
        pred = U.dot(S.dot(V))
        error = MSE(pred,train_mat)
        error_a.append(error)
```

```
            v_score = compute_score(validation,pred)
            print("validation score:",v_score)
            score_a.append(v_score)
        plt.title("d vs MSE (Q3.2(a))")
        plt.plot(d_list,error_a)
        plt.show()
        plt.title("d vs accuracy (Q3.2(a))")
        plt.plot(d_list,score_a)
        plt.show()

def q3b():
        train_set = scipy.io.loadmat('hw7_data/joke_data/joke_train.mat')
        train_mat = train_set['train']

        #load validation set.
        valid_data = genfromtxt('hw7_data/joke_data/validation.txt', delimiter=',')
        valid_data[valid_data == 0] = -1
        valid_data = valid_data - np.array([1,1,0])

        train_mat_zero = np.array(train_mat[:,:])
        train_mat_zero_nan_index = np.isnan(train_mat_zero)
        train_mat_zero[train_mat_zero_nan_index] = 0

        U, s, V = np.linalg.svd(train_mat_zero, full_matrices=False)
        nan_list = np.isnan(train_mat)
```

```
class latent_factor_model:
    def __init__(self, lamb):
        self.lamb = lamb
        self.u = np.random.random(U.shape)
        self.v = np.random.random(V.shape)

    def grad_u(self):
        vv = []
        for j in range(len(self.v)):
            vv.append(np.outer(self.v[j], self.v[j]))

        for i in range(len(self.u)):
            inv = sum([vv[j] for j in range(len(self.v)) if not nan_list[i,j]])
            inv += self.lamb*np.identity(len(inv))
            vR = sum([self.v[j]*train_mat[i,j] for j in range(len(self.v)) if not nan_list[i,j]])
            self.u[i] =  np.linalg.inv(inv).dot(vR)

    def grad_v(self):
        uu = []
        for i in range(len(self.u)):
            uu.append(np.outer(self.u[i], self.u[i]))

        for j in range(len(self.v)):
```

```
                    inv = sum([uu[i] for i in range(len(self.u)) if not nan_list[i,j]])
                    inv += self.lamb*np.identity(len(inv))
                    uR = sum([self.u[i]*train_mat[i,j] for i in range(len(self.u)) if not nan_list[i,j]])
                    self.v[j] =  np.linalg.inv(inv).dot(uR)

        def train(self):
            self.grad_u()
            self.grad_v()

        def predict(self, user, joke):
            return self.u[int(user)].dot(self.v[int(joke)])

lfm = latent_factor_model(10)
lfm.train()
errors = []
for i in range(30):
    print("Epoch", i)
    curr_error = 0
    predictions = np.array([[0 for _ in range(train_mat.shape[1])] for _ in range(train_mat.shape[0])])
    for example in valid_data:
        user, joke, actual = example[0], example[1], example[2]
        pred = lfm.predict(user,joke)
        predictions[int(user),int(joke)] = pred
```

```
                        curr_error += np.sign(pred) != np.sign(actual)*1
                curr_error = curr_error/len(valid_data)
                errors.append(curr_error)
                print("MSE:",MSE(predictions,train_mat))
                print(errors[i])
                lfm.train()

        query = genfromtxt('hw7_data/joke_data/query.txt', delimiter=',')
        query = query - np.array([0,1,1])

        predictions = []
        for row in query:
                score = lfm.predict(row[1], row[2])
                predictions.append((score > 0)*1)
        preds = np.array(predictions)

        table = {"Category":preds.astype(int),"Id":np.arange(1,len(preds)+1)}
        output = pd.DataFrame(data=table)
        output.to_csv("kaggle_ycls_hw7.csv", index=False)
        print("csv created")
        plt.plot(errors)

def compute_score(validation,pred):
        correct_count = 0
        for i,j,s in validation:
```

```
                prediction = pred[int(i),int(j)]
                #print("pred:",pred[int(i),int(j)])
                if np.sign(int(s))==np.sign(prediction)*1:
                        correct_count+=1
        return float(correct_count)/len(validation)


def MSE(pred,train_mat):
        tot=0
        for i in range(train_mat.shape[0]):
                for j in range(train_mat.shape[1]):
                        if not np.isnan(train_mat[i,j]):
                                tot+=(pred[i,j]-train_mat[i,j])**2
        return tot


main()
```