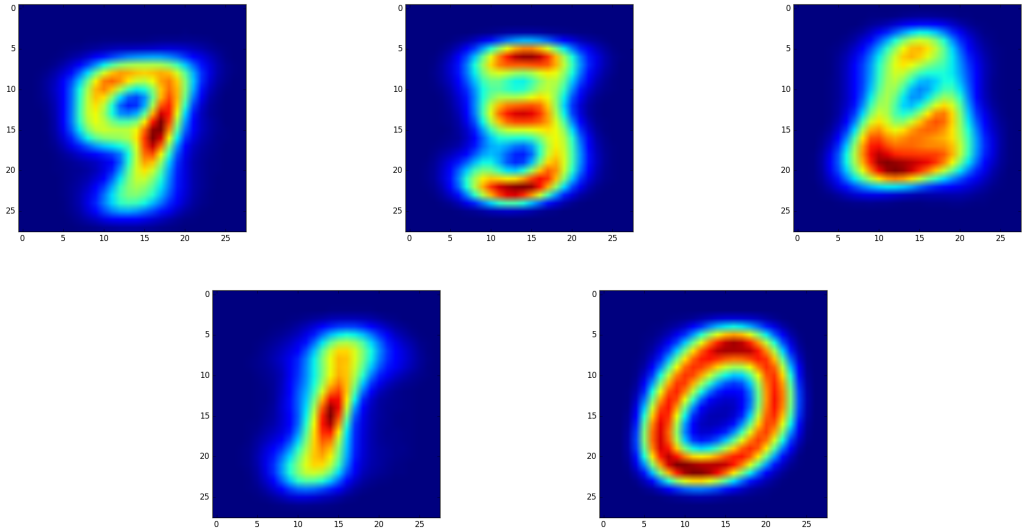
CS189/CS289A – Spring 2017 — Homework 7

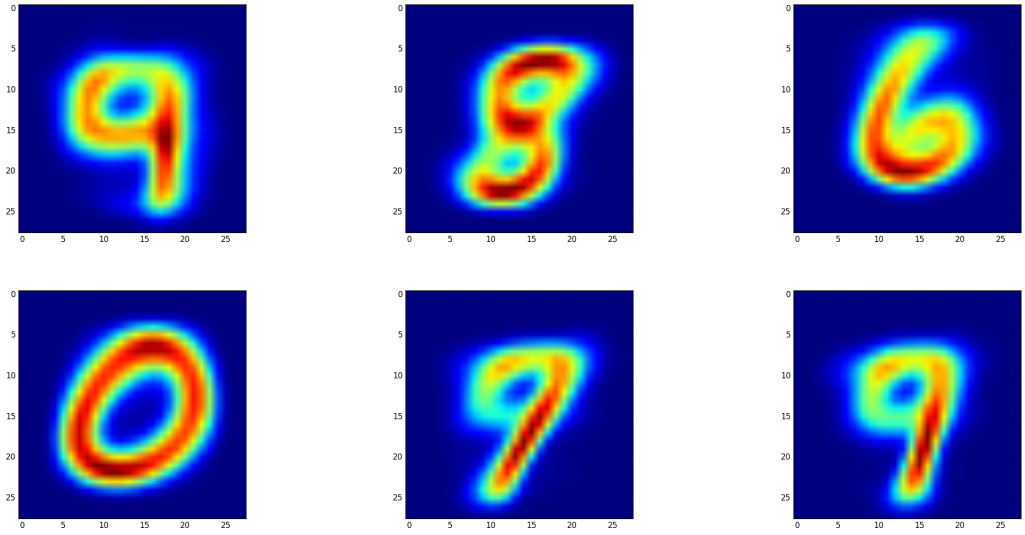
Yaoyang Zhang, SID 3032114788

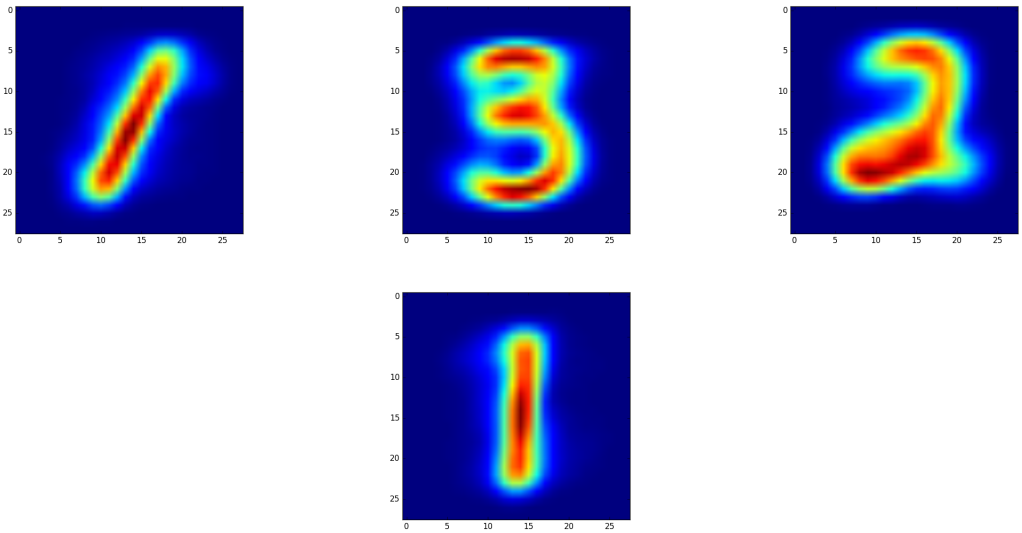
Problem 1: K-means

(a) Visualization of cluster centers with $d = 5$

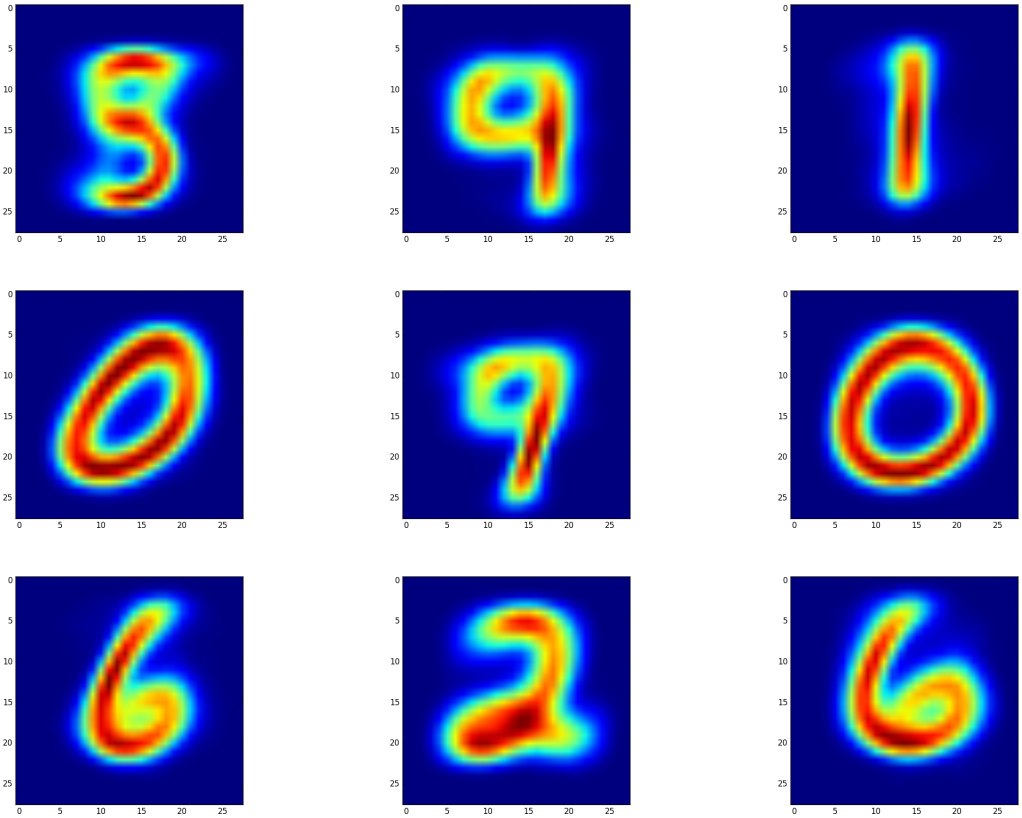


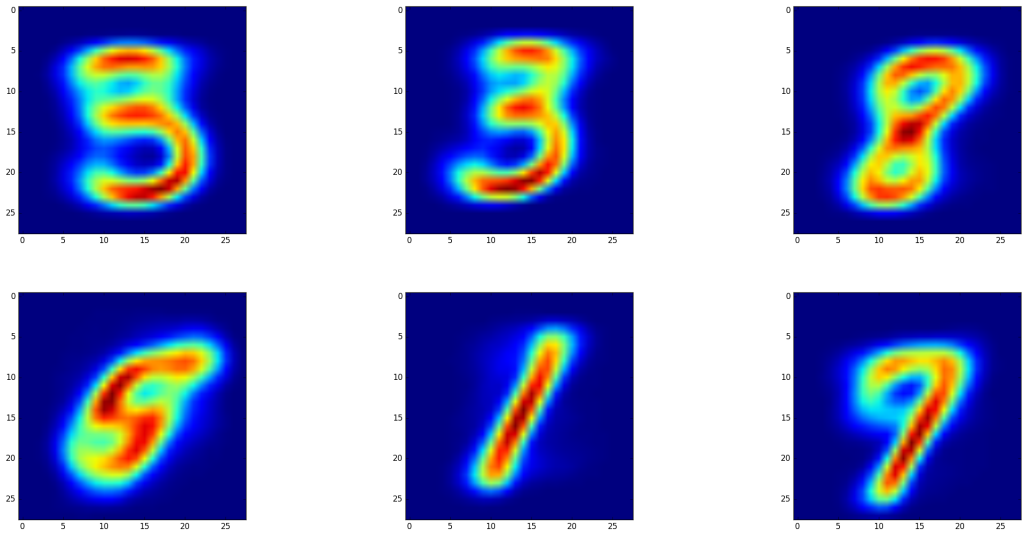
(b) Visualization of cluster centers with $d = 10$





(c) Visualization of cluster centers with $d = 15$





(d) Some observations: when d is small, some cluster centers seem to be a combination of two or more different digits. When d is large, some cluster centers seem to be the same digit but with slight different orientations.

(e) Code for implementation of Lloyd's algorithm

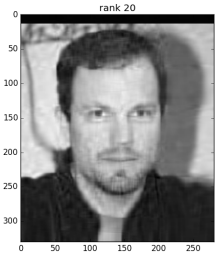
```
import numpy as np
import scipy.io
import matplotlib.pyplot as plt


class kmeans:
    def __init__(self, n_clusters =5):
        self.n_clusters = n_clusters


    def fit(self, X):
        y = np.random.randint(self.n_clusters,size=X.shape[0])
        means = self.calculate_mean(X, y)
        while True:
            old_y  = y
            y = self.update_y(X, means)
            means = self.calculate_mean(X, y)
            print(y)
            print(means)
            if np.linalg.norm(old_y - y) < 1e-10:
                break
        return y, means

    def calculate_mean(self, X, y):
        means = np.zeros((self.n_clusters, X.shape[1]))
```

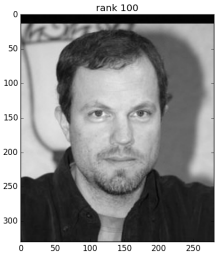
```
        cluster_size = np.zeros(self.n_clusters)
        for i in range(X.shape[0]):
            means[int(y[i])] += X[i]
            cluster_size[int(y[i])] += 1
        means /= cluster_size[:,None]
        return means

    def update_y(self, X, means):
        y = np.zeros((X.shape[0],),dtype=np.int)
        for i in range(X.shape[0]):
            y[i] = self.cloesest_cluster(X[i,:], means)
        return y

    def cloesest_cluster(self, x, means):
        label = -1
        distance = np.inf
        for i in range(self.n_clusters):
            new_dist =np.linalg.norm(x - means[i,:])
            if new_dist < distance:
                distance = new_dist
                label = i
        return label
```

```
data = scipy.io.loadmat('hw7_data/mnist_data/images.mat')
img = data['images']
new_img = np.zeros((60000, 784))
for i in range(60000):
    for j in range(28):
        new_img[i,j*28:(j+1)*28] = img[j,:,i]

km = kmeans(15)
y, means = km.fit(new_img)
for i in range(15):
    plt.imshow(means[i,:].reshape((28,28)))
    plt.savefig('15_clusters_'+str(i)+'.png')
```

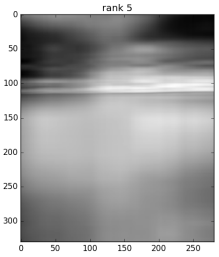
Low-rank Approximation

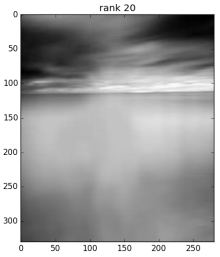
(a) Low rank approximation for face:

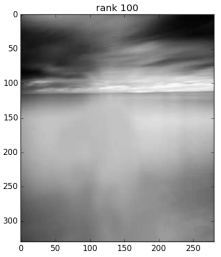




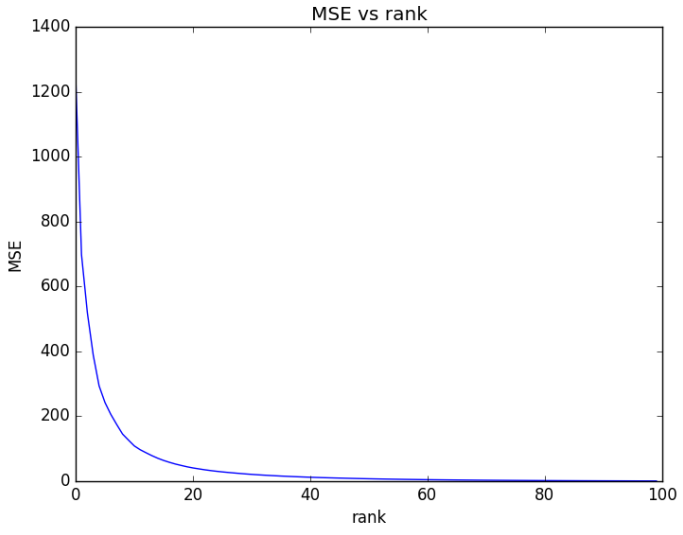
Low rank approximation for sky:



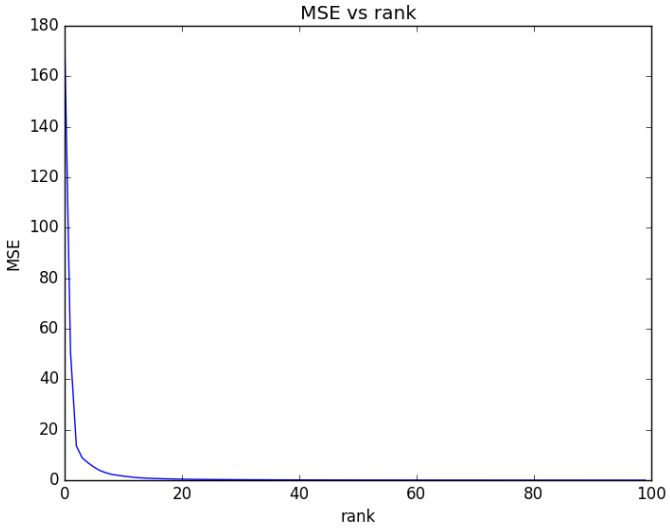




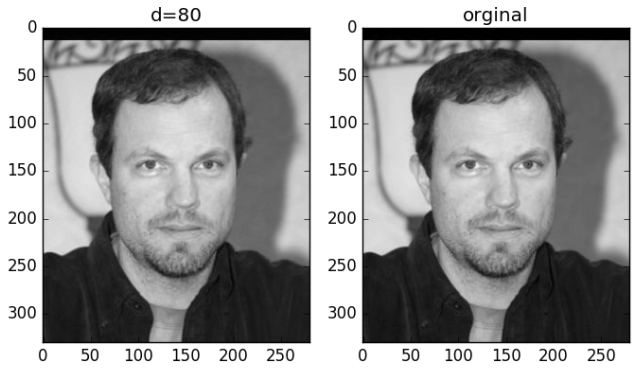
(b) MSE versus d for face

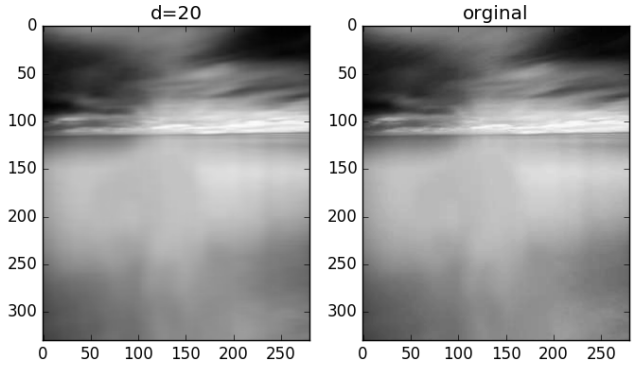


MSE versus d for sky



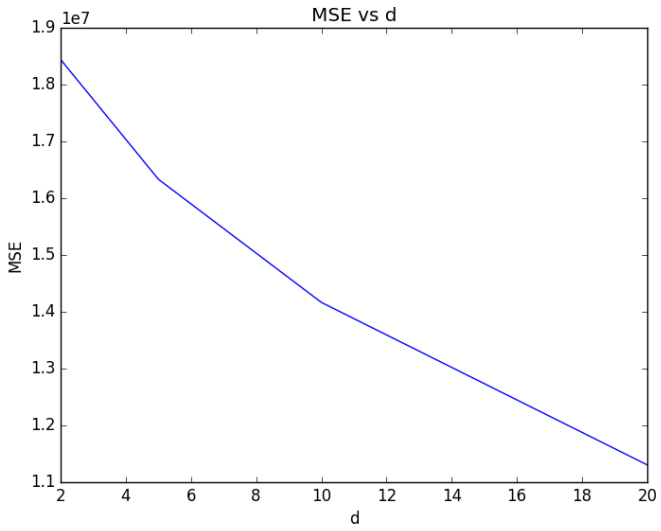
(c) After some experiments, I found out that $d = 20$ is a good approximation for sky and $d = 80$ is a good approximation for face. One possibility that causes the difference is that face is more complex than the sky and have more details that need to be captured by higher rank components.



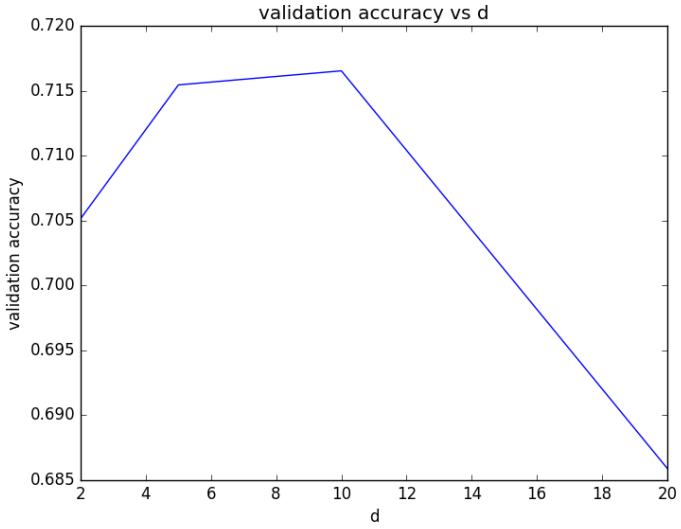

d=20
orginal
0
50
100
150
200
250
300
0
50
100
150
200
250

Joke Recommender System

(a) For SVD implementation, the MSE versus d plot is shown as follows:



The validation accuracy versus d is shown as follows:



The highest validation accuracy is 0.717 and is achieved when $d = 10$. When d goes to 20, MSE drops but validation error increases.

(b) (i) Let $U \in R^{n \times d}$ and $V \in R^{d \times m}$ be the latent representation for users and jokes. Let u_i be the ith row of U and v_j be the jth column of V. The updating rule for u_i and v_j is obtained by setting the partial derivative of L w.r.t u_i and v_j to 0.

$$\frac{\partial L}{\partial u_i} = \sum_{(i,j) \in S} (u_i v_j - R_{ij}) v_j^T + \lambda u_i = 0, \forall i$$

$$\implies u_i = (R_{is}V_s^T)(V_sV_s^T + \lambda I_d)^{-1}$$

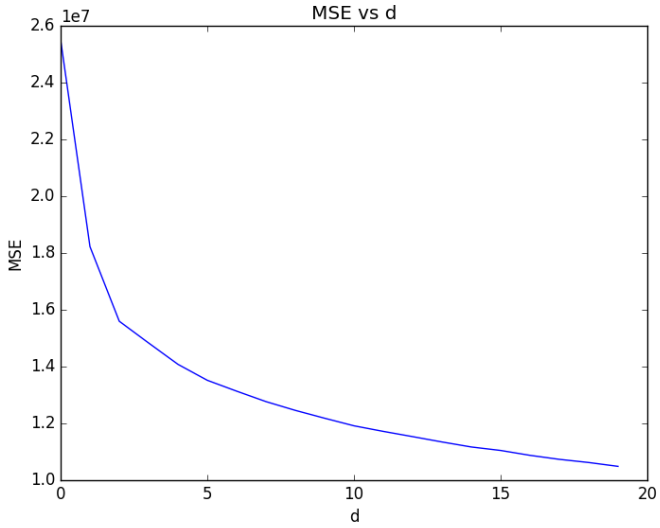
where s are the indices where $(i,j) \in S$, R_{is} is the ith row of R with only those columns whose indices are in s, V_s is V with only those columns whose indices are in s, and I_d is the identity matrix.

$$\frac{\partial L}{\partial v_j} = \sum_{(i,j)\in S} (u_iv_j - R_{ij})u_i^T + \lambda v_j = 0, \forall j$$

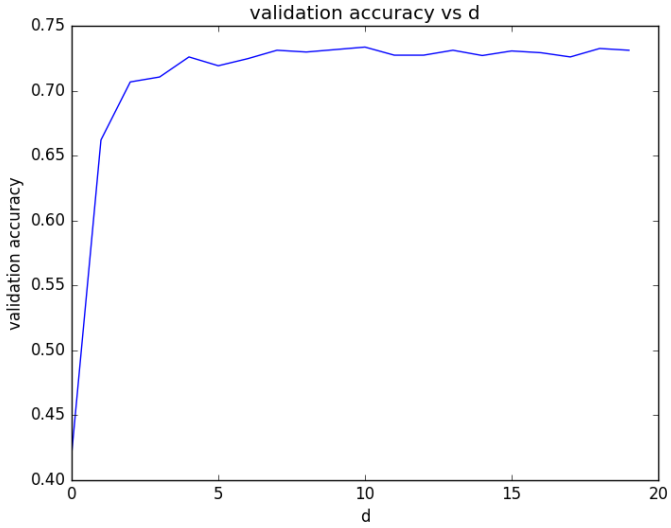
$$\implies v_i = (U_s^TU_s + \lambda I_d)^{-1}U_s^TR_{sj}$$

where s are the indices where $(i,j) \in S$, R_{sj} is the jth column of R with only those rows whose indices are in s, U_s is U with only those rows whose indices are in s, and I_d is the identity matrix.

(ii) I chose the hyper-parameter λ to be 300. The MSE versus d plot is shown as follows:



The validation accuracy versus d is shown as follows:



It can be observed that when using SVD, MSE keeps dropping as d increase but validation accuracy peaks at $d = 10$ and start dropping rapidly after that. When using collaborative filtering, MSE keeps dropping and validation accuracy does not drop as d increases.

(iii) Kaggle display name: YaoyangZhang
Kaggle score: 0.73336

(c) Code for question a

```
import numpy as np
import scipy.io
import math
import matplotlib.pyplot as plt

def low_rank_approximation(D, k):
    U,s,V = np.linalg.svd(D, full_matrices=False)
    s1 = np.zeros((U.shape[1], k))
    s2 = np.zeros((k, V.shape[0]))
    s1[:k,:k] = np.diag(np.power(s[:k], 0.5))
    s2[:k, :k] = np.diag(np.power(s[:k], 0.5))
    return U.dot(s1), s2.dot(V)

def MSE(train_data, U, V, non_nan_set):
    mse = 0
    for item in non_nan_set:
        i = item[0]
        j = item[1]
        mse += (U[i,:].dot(V[:,j])-train_data[i,j])**2
    return mse

def predict(u, v, i, j):
    rating = u[i,:].dot(v[:,j])
```

```
    if rating >=0 :
        return 1
    else:
        return 0
# validation set
valid = []
with open('hw7_data/joke_data/validation.txt') as f:
    for line in f:
        line = line.strip().split(',')
        valid.append(line)

# training data
data = scipy.io.loadmat('hw7_data/joke_data/joke_train.mat')
train_data = data['train']

# non-NaN set
non_nan_set = []
for i in range(train_data.shape[0]):
    for j in range(train_data.shape[1]):
        if math.isnan(train_data[i,j]):
            continue
        non_nan_set.append([i,j])

train_data_zero = np.nan_to_num(train_data)

dimension =[2,5,10,20]
loss = []
error = []
for d in dimension:
    u, v = low_rank_approximation(train_data_zero, d)
    print(u.shape, v.shape)
    loss.append(MSE(train_data, u, v, non_nan_set))
    error_d = 0
    print(d)
    for line in valid:
        predicted_r = predict(u, v, int(line[0])-1,int(line[1])-1)
        if predicted_r != int(line[2]):
            error_d += 1
    error_d /= len(valid)
    error.append(1-error_d)
plt.figure()
plt.plot(dimension, loss)
plt.xlabel('d')
plt.ylabel('MSE')
plt.title('MSE vs d')
plt.savefig('loss_d.png')

plt.figure()
```

```
plt.plot(dimension, error)
plt.xlabel('d')
plt.ylabel('validation accuracy')
plt.title('validation accuracy vs d')
plt.savefig('accu_d.png')
```

Code for question b:

```
import numpy as np
import scipy.io
import math
import csv
import matplotlib.pyplot as plt
from scipy import sparse

# training data
data = scipy.io.loadmat('hw7_data/joke_data/joke_train.mat')
train_data = data['train']


# validation set
valid = []
with open('hw7_data/joke_data/validation.txt') as f:
    for line in f:
        line = line.strip().split(',')
        valid.append(line)


# non-NaN set
non_nan_set = []
for i in range(train_data.shape[0]):
    for j in range(train_data.shape[1]):
        if math.isnan(train_data[i,j]):
            continue
        non_nan_set.append([i,j])

train_data_zero = np.nan_to_num(train_data)



indicator = np.ones(train_data.shape)
for i in range(train_data.shape[0]):
    for j in range(train_data.shape[1]):
        if math.isnan(train_data[i,j]):
            indicator[i, j] = 0
            train_data[i, j] = 0
```

```
def solve_u(u, v, data, indicator, l):
    u_new = np.zeros(u.shape)
    for user in range(u.shape[0]):
        data_user = data[user, :]
        indicator_user = indicator[user, :]
        #  indicator_user = np.diag(indicator_user)
        v_i = v[:, indicator_user.astype(int) == 1]
        data_user = data_user[indicator_user.astype(int) == 1]
        u_user = np.dot(data_user, v_i.T)

        u_user = np.dot(u_user, np.linalg.inv(l * np.identity(u.shape[1]) + v
        u_new[user, :] = u_user
    return u_new


def solve_v(u, v, data, indicator, l):
    v_new = np.zeros(v.shape)
    for review in range(v.shape[1]):
        data_review = data[:, review]
        indicator_review = indicator[:, review]
        # indicator_review = np.diag(indicator_review)
        u_j = u[indicator_review.astype(int) == 1, :]
        data_review = data_review[indicator_review.astype(int) == 1]
        v_review = np.dot(u_j.T, data_review)
        v_review = np.dot(np.linalg.inv(np.dot(u_j.T, u_j) + l * np.identity(
        v_new[:, review] = v_review
    return v_new

def latent_indexing(data, d, indicator, max_step= 20, l = 300):
    u = np.random.uniform(-1, 1, (data.shape[0], d))
    v = np.random.uniform(-1, 1, (d, data.shape[1]))
    for i in range(max_step):
        u = solve_u(u, v, data, indicator, l)
        # print(MSE(train_data, u, v, non_nan_set))
        v = solve_v(u, v, data, indicator, l)
        # print(MSE(train_data, u, v, non_nan_set))
        print(i)
    return u, v

def predict(u, v, i, j):
    rating = u[i,:].dot(v[:,j])
    if rating > 0 :
        return 1
    else:
        return 0

def MSE(train_data, U, V, non_nan_set):
    mse = 0
```

```
    for item in non_nan_set:
        i = item[0]
        j = item[1]
        mse += (U[i,:].dot(V[:,j])-train_data[i,j])**2
    return mse


mses = []
accu = []
for d in range(20):
    u, v = latent_indexing(train_data, d, indicator)
    error_d = 0
    for line in valid:
        predicted_r = predict(u, v, int(line[0]) - 1, int(line[1]) - 1)
        if predicted_r != int(line[2]):
            error_d += 1
    error_d /= len(valid)
    accu.append(1 - error_d)
    mses.append(MSE(train_data, u, v, non_nan_set))

plt.figure()
plt.plot(range(20), mses)
plt.xlabel('d')
plt.ylabel('MSE')
plt.title('MSE vs d')
plt.savefig('new_loss_d.png')

plt.figure()
plt.plot(range(20), accu)
plt.xlabel('d')
plt.ylabel('validation accuracy')
plt.title('validation accuracy vs d')
plt.savefig('new_accu_d.png')


query = []
with open('hw7_data/joke_data/query.txt') as f:
    for line in f:
        line = line.strip().split(',')
        query.append(line)
result = []

for item in query:
    i = int(item[1]) - 1
    j = int(item[2]) - 1
    predicted = predict(u, v, i, j)
    result.append(predicted)
```

```
with open('results.csv', 'wt') as f:
    writer = csv.writer(f, delimiter=',')
    writer.writerow(['Id','Category'])
    for i, line in enumerate(result):
        writer.writerow([str(i+1), str(line)])
```