EECS E6892 Bayesian Models for Machine Learning
Homework 1

John Min; jcm2199

February 28, 2014

1 Expectation Maximiation (E.M.) of Probabilistic PCA

$\{x_1, \ldots, x_N\} \in \mathcal{R}^d, x_n \sim N(Wz_n, \sigma^2 I), W \in \mathcal{R}^{d \times k}$. W is unknown and $z_n \sim N(0, I)$.

1.1 Compute posterior of z

$$\begin{aligned}
p(x_1, \ldots, x_N | W^{\text{old}}) &\propto p(x_1, \cdots, x_N | W^{\text{old}}, z_1, \cdots, z_N) p(z_1, \ldots, z_N) \\
&= \prod_{n=1}^{N} \left[\exp\left\{ \frac{(x_n - W^{\text{old}} z_n)^\top (x_n - W^{\text{old}} z_n)}{2\sigma^2} \right\} \exp\left\{ \frac{z_n^2}{2} \right\} \right] \\
&\propto \exp\left\{ -\frac{1}{2} \sum_{n=1}^{N} \left[z_n^\top z_n + \frac{1}{\sigma^2} z_n^\top W_{\text{old}}^\top W_{\text{old}} z_n - \frac{2}{\sigma^2} x_n^\top W_{\text{old}} z_n \right] \right\} \\
&= exp\left\{ -\frac{1}{2\sigma^2} \sum_{n=1}^{N} \left[z_n^\top \left(\sigma^2 I + W_{\text{old}}^\top W_{\text{old}} \right) z_n - 2 x_n^\top W_{\text{old}} z_n \right] \right\} \\
&\sim \mathbf{N}(\mu, \mathbf{\Sigma}) \\
\mu &= \left(\frac{\sigma^2 I + W_{\text{old}}^\top W_{\text{old}}}{\sigma^2 I} \right)^{-1} W_{\text{old}}^\top X \\
\mathbf{\Sigma} &= \frac{\sigma^2 I}{\sigma^2 I + W_{\text{old}}^\top W_{\text{old}}}
\end{aligned}$$

Note:
$-\frac{1}{2}(x - \mu)^\top \mathbf{\Sigma}^{-1} (x - \mu) = -\frac{1}{2} x^\top \mathbf{\Sigma}^{-1} x + x^\top \mathbf{\Sigma}^{-1} \mu +$ const.

Above, $\mu = x^\top W$
$z^\top \mathbf{\Sigma}^{-1} \mu = z^\top W^\top x \Rightarrow \mathbf{\Sigma}^{-1} \mu = W^\top x$

1.2 Take expectation of complete data log-likelihood

$$\begin{aligned}
&\mathrm{E}_{z_n} \left[\ln p\big(x_1, \ldots, x_N | W, z_1, \ldots, z_N\big) \right] = \sum_{n=1}^{N} \left[\exp\left\{ \frac{x_n^2}{2} \right\} - 2\mathrm{E}\big[x_n^\top W z_n\big] + \mathrm{E}\big[z_n^\top W^\top W z_n\big] + \sigma^2 \mathrm{E}\big[z_n^\top z_n\big] \right] \\
&= \sum_{n=1}^{N} \left[\exp\left\{ \frac{x_n^2}{2} \right\} - 2 x_n^\top W \mu + \mathrm{E}\Big[(W z_n)^\top (W z_n) \Big] + \sigma^2 \big(\mu^2 + \Sigma \big) \right] \\
&= \sum_{n=1}^{N} \left[\exp\left\{ \frac{x_n^2}{2} \right\} - 2 x_n^\top W \mu + \big[\operatorname{Tr}(\mu^2 + \Sigma) \big]^\top \big[\operatorname{Tr}(W W^\top) \big] + \sigma^2 \big(\mu^2 + \Sigma \big) \right]
\end{aligned}$$

1.3 Maximize $Q(W, W^{\text{old}})$ w.r.t. W

Find the gradient with respect to W and set to 0:
$\sum_{n=1}^{N} \Big[-2 x_n^\top \mu + 2 \operatorname{Tr}(\mu^2 + \Sigma) W = 0$. (Note: $\nabla_A \operatorname{Tr}(AB) = B^\top$).

W^* aka $W^{\text{new}} = \Big[\operatorname{Tr}(\mu^2 + \Sigma) \Big]^{-1} \bar{X}^\top \mu$ where $\bar{X} = \frac{1}{N} \sum_{n=1}^{N} x_n$

2 Implementation of Probabilistic Matrix Factorization

2.1 Maximum a posteriori (MAP)

Update steps:
$u_i^{\text{MAP}} = \left(\lambda\sigma^2 I + V_i^\top V_i\right)^{-1} V_i^\top m_{u_i}$
$v_j^{\text{MAP}} = \left(\lambda\sigma^2 I + U_j U_j^\top\right)^{-1} U_j^\top m_{v_j}$

Coordinate ascent is used for this MAP implementation where each update step for a particular u_i optimizes the joint log-likelihood for u_i, and the update step for v_j does the same for each v_j. Each update step, whether it be for u_i or for v_j is an $\mathcal{L}_2$ regularized least squares solution, also known as ridge regression.

2.2 Gibbs sampling

Sample: $u_i \sim N(\mu_i, \Sigma_i)$ where $\mu_i = \left(\lambda\sigma^2 I + V_i^\top V_i\right)^{-1} V_i^\top m_{u_i}$, $\Sigma_i = \left(\sigma^2 I + V_i^\top V_i\right)^{-1}$
$v_j \sim (N(\mu_j, \Sigma_j)$ where $\mu_j = \left(\lambda\sigma^2 I + U_j U_j^\top\right)^{-1} U_j^\top m_{v_j}, \Sigma_j = \left(\sigma^2 I + U_j U_j^\top\right)^{-1}$

A burn-in period of 250 iterations is used after which we collect samples every 25 iterations. In each iteration, we sample each u_i from the posterior distribution with parameters that are being updated for each i. Then, we repeat for the v_j's.

2.3 Similar Movies by v_j using d=10

2.3.1 Desperate Measures (1998)

Doors, The (1991); Mary Shelley's Frankenstein (1994); Malice (1993); Bye Bye, Love (1995); Love Affair (1994).

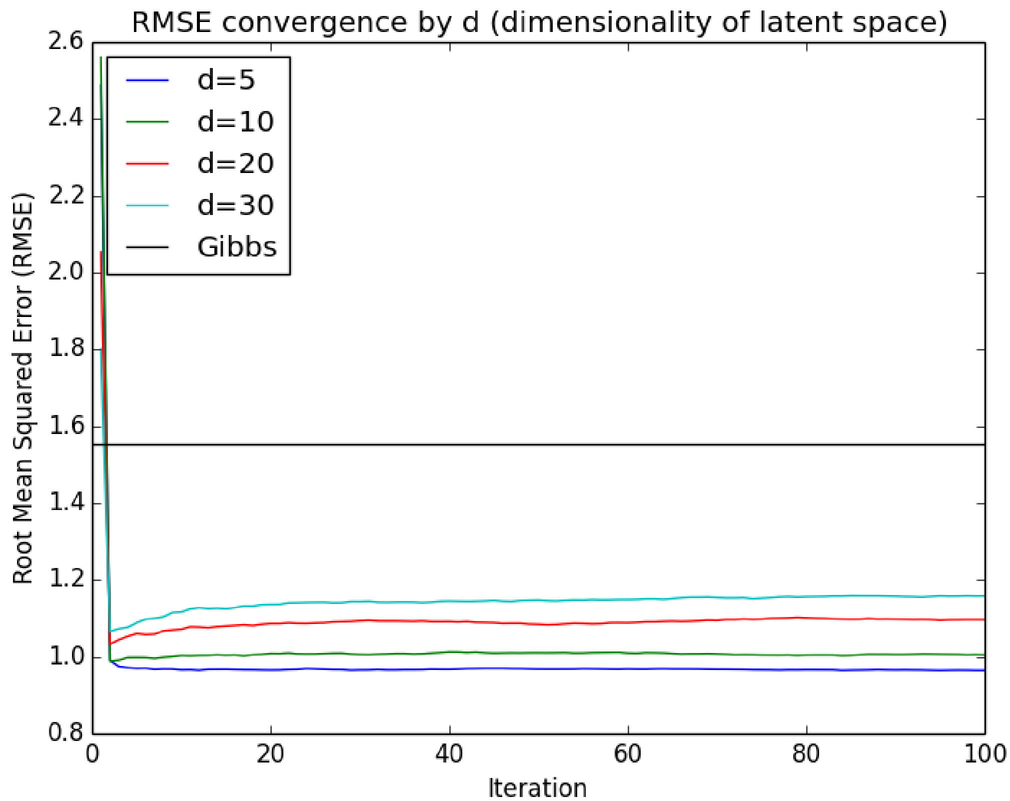
2.3.2 Oliver & Company (1988)

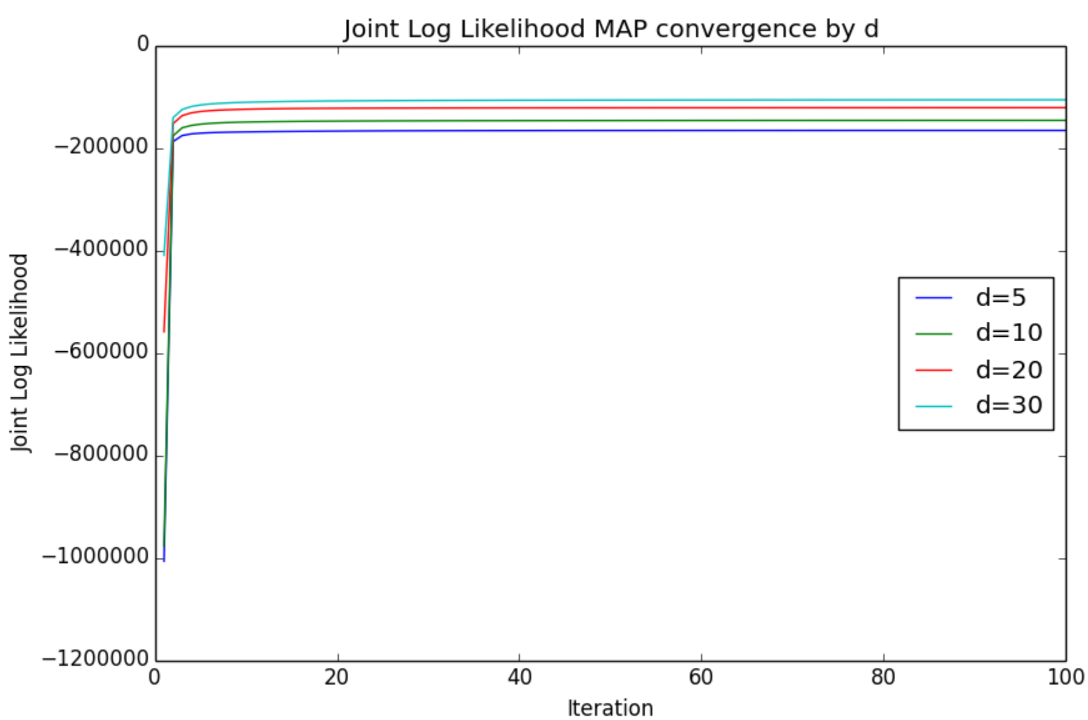
Indian in the Cupboard, The (1995); Prefontaine (1997); Abyss, The (1989); Kid in King Arthur's Court, A (1995); Air Up There, The (1994).

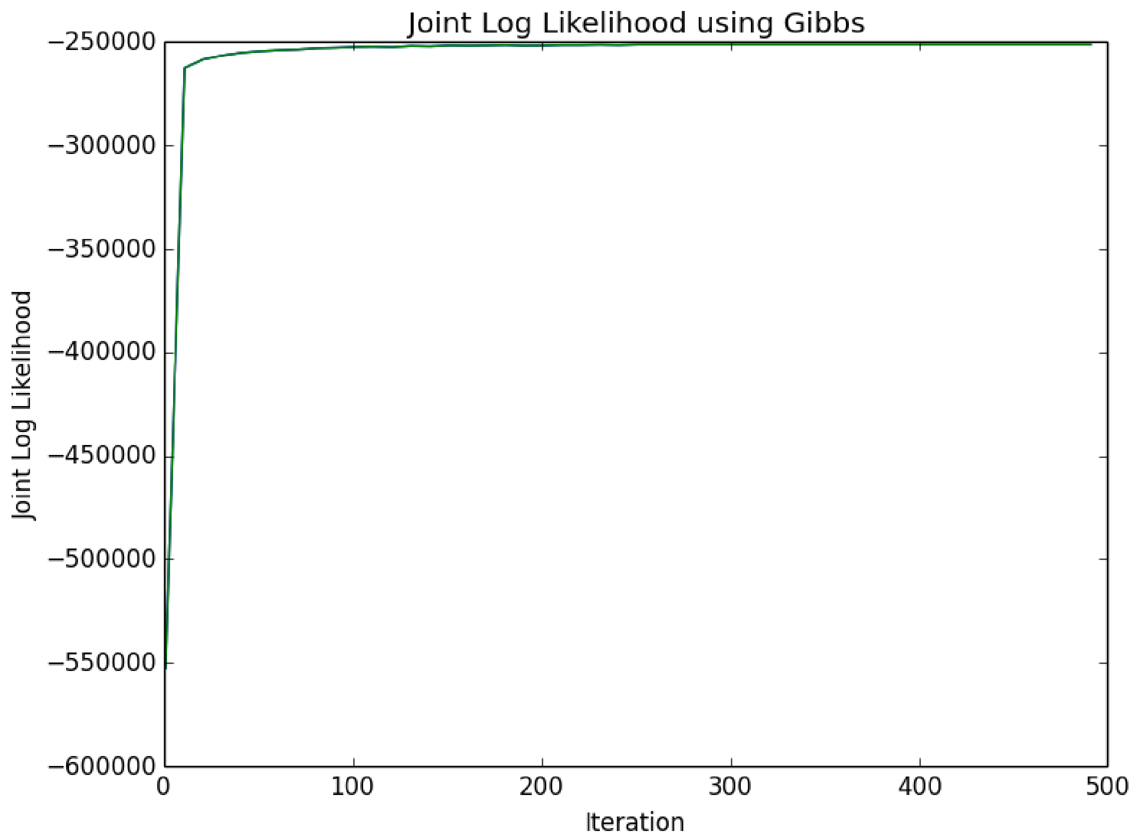
2.3.3 Chinatown (1974)

Manhattan (1979); 8 1/2 (1963); Short Cuts (1993); Annie Hall (1977); Bonnie and Clyde (1967);

2.4 Plots of results







Joint Log Likelihood using Gibbs
Joint Log Likelihood
Iteration
−250000
−300000
−350000
−400000
−450000
−500000
−550000
−600000
0
100
200
300
400
500

2.5 PMF - Python code

```
import numpy as np
import subprocess
import os
import matplotlib.pyplot as plt

def load_data_dict(filename):
    data = dict()
    with open(filename, 'rb') as f:
    for line in f:
        row = line.split(',')
        user, movie, rating = int(row[0])-1, int(row[1])-1, int(row[2])
        data[(user, movie)] = rating
    return data

def metadata(data, base_dir):
    users = list()
    movies = list()
    for user, movie in data.keys():
    users.append(user)
    movies.append(movie)
    users = sorted(list(set(users)))
    movies = sorted(list(set(movies)))
    N = len(users)
    M = int(subprocess.Popen('wc␣-l␣%s/movies.txt'%(base_dir),
    stdout=subprocess.PIPE, shell=True).communicate()[0].split()[0])
    #M = 1682
    return users, movies, N,M

def user_movie_dictionary(data, N, M):
    user_movie_dict = dict()
    for user_id in xrange(N):
    user_movie_dict[user_id] = list()
    for movie_id in xrange(M):
        try:
        data[(user_id, movie_id)]
        user_movie_dict[user_id].append(movie_id)
        except:
        pass
    return user_movie_dict

def movie_user_dictionary(data, N, M):
    movie_user_dict = dict()
    for movie_id in xrange(M):
    movie_user_dict[movie_id] = list()
    for user_id in xrange(N):
        try:
        data[(user_id, movie_id)]
        movie_user_dict[movie_id].append(user_id)
        except:
        pass
    return movie_user_dict

def dict_to_matrix(data_dict, N, M):
    mat = np.zeros((N,M))
    for user, movie in data_dict.keys():
    mat[user, movie] = data_dict[(user, movie)]
    return mat
```

```
#initialization
def initialize_factorization(N, M, d, sigma, lamb):
    U = np.zeros((N,d))
    V = np.zeros((d,M))
    I = np.identity(d)
    mean = np.zeros(d)
    cov = np.power(float(lamb),-1)*I
    for i in xrange(N):
    U[i,:] = np.random.multivariate_normal(mean, cov)
    for j in xrange(M):
    V[:,j] = np.random.multivariate_normal(mean, cov)
    return U, V, I

# user optimization
def update_user_MAP(M_train, U, V, N, I, user_movie_dict):
    for i in xrange(N):
    # compute V_i
    V_i = V[:, user_movie_dict[i]].transpose()
    # compute m_u
    m_u = M_train[i, user_movie_dict[i]]
    # compute u_MAP
    U[i,:] = np.dot(np.linalg.inv(lamb*np.power(float(sigma),2)*I +
        np.dot(V_i.transpose(), V_i)), np.dot(V_i.transpose(), m_u))
    return U

# movie optimizaiton
def update_movie_MAP(M_train, U, V, M, I, movie_user_dict):
    for j in xrange(M):
    # compute U
    U_j = U[movie_user_dict[j], :].transpose()
    # compute m_v
    m_v = M_train[movie_user_dict[j], j]
    # compute v_MAP
    V[:,j] = np.dot(np.linalg.inv(lamb*np.power(float(sigma),2)*I +
        np.dot(U_j, U_j.transpose())), np.dot(U_j, m_v))
    return V

def predict_ratings(U, V):
    M_pred = np.dot(U,V)
    return M_pred

def compute_RMSE(M_pred, test):
    MSE_sum = 0
    N = len(test.keys())
    for i, j in test.keys():
    y_pred = round(M_pred[i,j])
    y = M_test[i,j]
    MSE_sum += np.power(y - y_pred, 2)
    RMSE = np.sqrt(MSE_sum/float(N))
    return RMSE

def log_likelihood(train, U, V, sigma, lamb, N, M):
    data_sum = 0
    u_sum = 0
    v_sum = 0
    for i, j in train.keys():
    data_sum += np.power(train[i,j] - np.dot(U[i,:].transpose(), V[:,j]), 2)
    for i in xrange(N):
    u_sum += np.dot(U[i,:].transpose(), U[i,:])
    for j in xrange(M):
```

```
    v_sum += np.dot(V[:,j].transpose(), V[:,j])
    joint_LL = np.power(sigma,-2)*-.5*data_sum - .5*lamb*u_sum - .5*lamb*v_sum
    return joint_LL

# coordinate ascent
def coordinate_ascent(M_train, test, N, M, d, N_iterations, user_movie_dict, movie_user_dict, sigma, lamb):
    U, V, I = initialize_factorization(N, M, d, sigma, lamb)
    rmse_list = list()
    log_likelihood_list = list()
    for iter in xrange(N_iterations):
    U = update_user_MAP(M_train, U, V, N, I, user_movie_dict)
    V = update_movie_MAP(M_train, U, V, M, I, movie_user_dict)
    M_pred = predict_ratings(U, V)
    rmse = compute_RMSE(M_pred, test)
    rmse_list.append(rmse)
    joint_LL = log_likelihood(train, U, V, sigma, lamb, N, M)
    log_likelihood_list.append(joint_LL)
    return U, V, rmse_list, log_likelihood_list

def plot_RMSE(N_iterations, rmse):
    X = [x+1 for x in xrange(N_iterations)]
    plt.plot(X, rmse)
    plt.xlabel('Iteration')
    plt.ylabel('RMSE')
    plt.show()

## GIBBS SAMPLING
def update_user_Gibbs(M_train, U, V, N, I, user_movie_dict):
    for i in xrange(N):
    # compute V_i
    V_i = V[:, user_movie_dict[i]].transpose()
    # compute m_u
    m_u = M_train[i, user_movie_dict[i]]
    # compute u_MAP
    mean_user = np.dot(np.linalg.inv(lamb*np.power(sigma,2)*I +
        np.dot(V_i.transpose(), V_i)), np.dot(V_i.transpose(), m_u))
    cov_user = np.linalg.inv(lamb*I + np.power(sigma,-2)*np.dot(V_i.transpose(), V_i))
    U[i,:]= np.random.multivariate_normal(mean_user, cov_user)
    return U

def update_movie_Gibbs(M_train, U, V, M, I, movie_user_dict):
    for j in xrange(M):
    # compute U_j
    U_j = U[movie_user_dict[j], :].transpose()
    # compute m_v
    m_v = M_train[movie_user_dict[j], j]
    # compute v_MAP
    mean_movie = np.dot(np.linalg.inv(lamb*np.power(sigma,2)*I +
         np.dot(U_j, U_j.transpose())), np.dot(U_j, m_v))
    cov_movie= np.linalg.inv(lamb*I + np.power(sigma,-2)*np.dot(U_j, U_j.transpose()))
    V[:,j] = np.random.multivariate_normal(mean_movie, cov_movie)
    return V

def initialize_gibbs_dict(test):
    gibbs_dict = dict()
    for i,j in test.keys():
    gibbs_dict[(i,j)] = list()
    return gibbs_dict

def sample_gibbs(train, gibbs_dict, U, V):
```

```
    for i, j in train.keys():
    gibbs_dict[i,j].append(np.dot(U[i,:],V[:,j]))
    return gibbs_dict

def compute_RMSE_Gibbs(gibbs_dict, test):
    MSE_sum = 0
    N = len(test.keys())
    for i, j in test.keys():
    y_pred = round(np.mean(gibbs_dict[i,j]))
    y = M_test[i,j]
    MSE_sum += np.power(y - y_pred, 2)
    RMSE = np.sqrt(MSE_sum/float(N))
    return RMSE

def gibbs(M_train, train, test, N, M, d, sigma, lamb, N_gibbs, burn_in, thinning):
    U, V, I = initialize_factorization(N, M, d, sigma, lamb)
    gibbs_dict = initialize_gibbs_dict(test)
    log_likelihood_list = list()
    for iter in xrange(burn_in):
    U = update_user_Gibbs(M_train, U, V, N, I, user_movie_dict)
    V = update_movie_Gibbs(M_train, U, V, N, I, movie_user_dict)
    if iter%10 == 0:
        joint_LL = log_likelihood(train, U, V, sigma, lamb, N, M)
        log_likelihood_list.append(joint_LL)
    for iter in xrange(N_gibbs - burn_in):
    if iter%10 == 0:
        joint_LL = log_likelihood(train, U, V, sigma, lamb, N, M)
        log_likelihood_list.append(joint_LL)
    if iter%thinning == 0:
        gibbs_dict = sample_gibbs(test, gibbs_dict, U, V)
    rmse = compute_RMSE_Gibbs(gibbs_dict, test)
    return rmse, log_likelihood_list


if __name__ == "__main__":

# parameterization
sigma = np.sqrt(0.25)
lamb = 10
d_list = [10, 20, 30]
d = 10

base_dir = os.path.join(os.getcwd(), 'movie_ratings')
train = load_data_dict(os.path.join(base_dir, 'ratings.txt'))
test = load_data_dict(os.path.join(base_dir, 'ratings_test.txt'))
users, movies, N, M = metadata(train, base_dir)

user_movie_dict = user_movie_dictionary(train, N, M)
movie_user_dict = movie_user_dictionary(train, N, M)

M_train = dict_to_matrix(train, N, M)
M_test = dict_to_matrix(test, N, M)

N_iterations = 100

N_gibbs = 500
burn_in = 250
thinning = 25
```

```
d=5
U5, V5, rmse_5, LL_5 = coordinate_ascent(
    M_train, test, N, M, d, N_iterations, user_movie_dict, movie_user_dict, sigma, lamb)
d=10
U10, V10, rmse_10, LL_10 = coordinate_ascent(
    M_train, test, N, M, d, N_iterations, user_movie_dict, movie_user_dict, sigma, lamb)
d=20
U20, V20, rmse_20, LL_20 = coordinate_ascent(
    M_train, test, N, M, d, N_iterations, user_movie_dict, movie_user_dict, sigma, lamb)
d=30
U30, V30, rmse_30, LL_30 = coordinate_ascent(
    M_train, test, N, M, d, N_iterations, user_movie_dict, movie_user_dict, sigma, lamb)
rmse_gibbs, LL_gibbs = gibbs(
    M_train, train, test, N, M, d, sigma, lamb, N_gibbs, burn_in, thinning)

X_coord = [x+1 for x in xrange(N_iterations)]
X_gibbs = [10*x+1 for x in xrange(len(LL_gibbs))]

plt.plot(X_coord, rmse_5, label='d=5')
plt.plot(X_coord, rmse_10, label='d=10')
plt.plot(X_coord, rmse_20, label='d=20')
plt.plot(X_coord, rmse_30, label='d=30')
plt.axhline(y=rmse_gibbs, color='black', label='Gibbs')
plt.xlabel('Iteration')
plt.ylabel('Root␣Mean␣Squared␣Error␣(RMSE)')
plt.title ('RMSE␣convergence␣by␣d␣(dimensionality␣of␣latent␣space)')
plt.legend(loc=2)
plt.savefig('RMSE.png')
plt.show()

plt.plot(X_coord, LL_5, label='d=5')
plt.plot(X_coord, LL_10, label='d=10')
plt.plot(X_coord, LL_20, label='d=20')
plt.plot(X_coord, LL_30, label='d=30')
plt.xlabel('Iteration')
plt.ylabel('Joint␣Log␣Likelihood')
plt.title('Joint␣Log␣Likelihood␣MAP␣convergence␣by␣d')
plt.legend(loc=5)
plt.savefig('JLL.png')
plt.show()

plt.plot(X_gibbs, LL_gibbs)
plt.xlabel('Iteration')
plt.ylabel('Joint␣Log␣Likelihood')
plt.title('Joint␣Log␣Likelihood␣using␣Gibbs')
plt.save('JLL_Gibbs.png')
plt.show()

U = U10
V = V10

f = open(base_dir+'/movies.txt')
i = 0
movies = dict()
for line in f:
    movie_name = line.split('\n')[0]
    movies[i] = movie_name
    i += 1

import pandas as pd
```

```
import random

n = range(M)
random.shuffle(n)

movie_list = n[:3]

sim_movie_dict = dict()
for m in movie_list:
    sim_movie_dict[m] = list()
    print 'Base Movie: ', movies[m]
    col = range(M)
    col.remove(m)
    dist_Euc = []
    for j in col:
        d = np.linalg.norm(V[:,j] - V[:,m])
        dist_Euc.append(d)
    dist_Euc = pd.Series(dist_Euc)
    dist_Euc.sort() #inplace sort
    for idx in dist_Euc.index[:5]:
        sim_movie_dict[m].append(col[idx])
    print '5 most similar movies:'
    for x in sim_movie_dict[m]:
        print movies[x]
```