

# hw4ans

May 15, 2021

## 1 Homework 4

```
[2]: import numpy as np
import matplotlib.pyplot as plt
!pip show pystan
```

Name: pystan  
Version: 3.1.1  
Summary: Python interface to Stan, a package for Bayesian inference  
Home-page: <https://mc-stan.org>  
Author: Allen Riddell  
Author-email: riddella@indiana.edu  
License: ISC  
Location: /Users/grja/opt/anaconda3/lib/python3.8/site-packages  
Requires: pysimdjson, numpy, clickit, aiohttp, httpstan  
Required-by:

```
[3]: import stan

import multiprocessing
multiprocessing.set_start_method("fork")

from concurrent.futures import ThreadPoolExecutor as _ThreadPoolExecutor

def _exec_async(func, *args, **kwargs):
    with _ThreadPoolExecutor(max_workers=1) as executor:
        future = executor.submit(func, *args, **kwargs)
    return future.result()

def stan_build(*args, **kwargs): return _exec_async(stan.build, *args, **kwargs)
```

### 1.1 Question 1

```
[4]: Xtr = np.loadtxt("179-hw4-train.csv", dtype = 'float', delimiter = ",")
Xte = np.loadtxt("179-hw4-test.csv", dtype = 'float', delimiter = ",")
```

## 1.2 Question 2

```
[5]: recsys = """
data {
  int<lower=1> M;           // Total number of movies
  int<lower=1> N;           // Total number of users
  int<lower=1> R;           // Total number of ratings
  int<lower=0> K;           // Number of latent dimensions
  int usr[R];              // user id for r'th rating
  int movie[R];            // movie id for r'th rating
  vector<lower=1,upper=10> [R] rating; // vector of rating values (1..10)
}
transformed data {        // transform interval 1..10 to [-3,3]
  vector [R] pred = log( (rating+.5)./(10.5-rating) );
}                          // inverse is: 0.5 + 10./(1+exp(-pred))
parameters {
  vector [N] u;
  vector [M] v;
  matrix [N,K] U;          // latent dimensions
  matrix [M,K] V;
}
model{
  u ~ normal(0,1);
  v ~ normal(0,1);
  to_vector(U) ~ normal(0,1);
  to_vector(V) ~ normal(0,1);
  for (r in 1:R) pred[r] ~
    ↪normal(u[usr[r]]+v[movie[r]]+U[usr[r],]*V[movie[r],]', 0.1);
}
"""

[6]: N,M = Xtr.shape
usr,movie = np.where(Xtr==Xtr) # find non-NaN's
rating = Xtr[usr,movie]

## NOTE: Stan uses 1-based indexing, compared to python's zero-based, so adjust:
recdata = {"M": M, "N": N, "R":len(rating), "K":2,
           "usr":  usr+1,
           "movie": movie+1,
           "rating":rating+1
}

[7]: # You can specify a random seed if you want repeatability...

posterior = stan_build(recsys, data=recdata, random_seed=0)
```

Building...

Building: found in cache, done.

```
[8]: import pickle
with open('stan_recsys.pkl', 'wb') as f: pickle.dump(posterior, f)
```

### 1.3 Question 3

```
[1]: import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

import stan

import multiprocessing
multiprocessing.set_start_method("fork")

from concurrent.futures import ThreadPoolExecutor as _ThreadPoolExecutor

def _exec_async(func, *args, **kwargs):
    with _ThreadPoolExecutor(max_workers=1) as executor:
        future = executor.submit(func, *args, **kwargs)
    return future.result()
```

```
[2]: def stan_sample(model, *args, **kwargs): return _exec_async(model.sample,
↳ *args, **kwargs)
```

```
[3]: Xtr = np.loadtxt("179-hw4-train.csv", dtype = 'float', delimiter = ",")
Xte = np.loadtxt("179-hw4-test.csv", dtype = 'float', delimiter = ",")
```

```
[4]: import pickle
posterior = pickle.load(open('stan_recsys.pkl', 'rb'))
```

```
[5]: fit = stan_sample(posterior, num_chains=1, num_samples=1000)
```

```
Sampling: 0%
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
Sampling: 0% (1/2000)
```

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]



[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

```
Sampling: 95% (1900/2000)
Sampling: 100% (2000/2000)
Sampling: 100% (2000/2000)
Sampling: 100% (2000/2000)
Sampling: 100% (2000/2000), done.
Messages received during sampling:
  Gradient evaluation took 0.000635 seconds
  1000 transitions using 10 leapfrog steps per transition would take 6.35
seconds.
  Adjust your expectations accordingly!
```

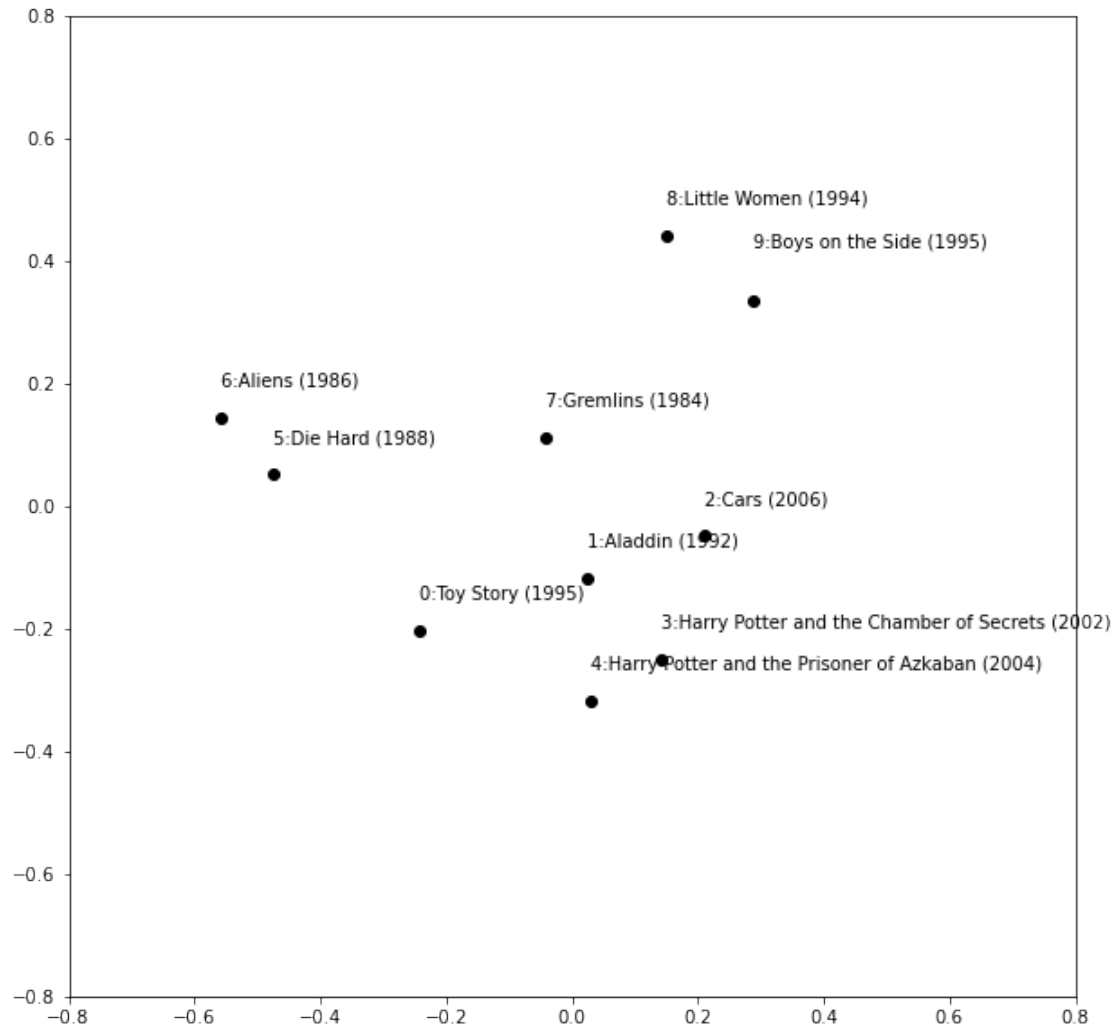
```
[6]: import pickle
      with open('stan_fit.pkl', 'wb') as f: pickle.dump(fit, f)
```

```
[7]: import pickle
      fit = pickle.load(open('stan_fit.pkl', 'rb'))
```

```
[8]: loc = fit["V"].mean(2)
      ms_train = open("179-hw4-train.csv", "r")
      m_name = ms_train.readline().replace("# ", "").split(",")

      plt.figure(figsize=(10, 10))
      for i in range(10):
          plt.plot(loc[i,0],loc[i,1], 'ko'); plt.text(loc[i,0],loc[i,1]+.05, str(i)+":
      ↪"+m_name[i]);
      plt.axis([-0.8, 0.8, -0.8, 0.8])
      plt.show()
```



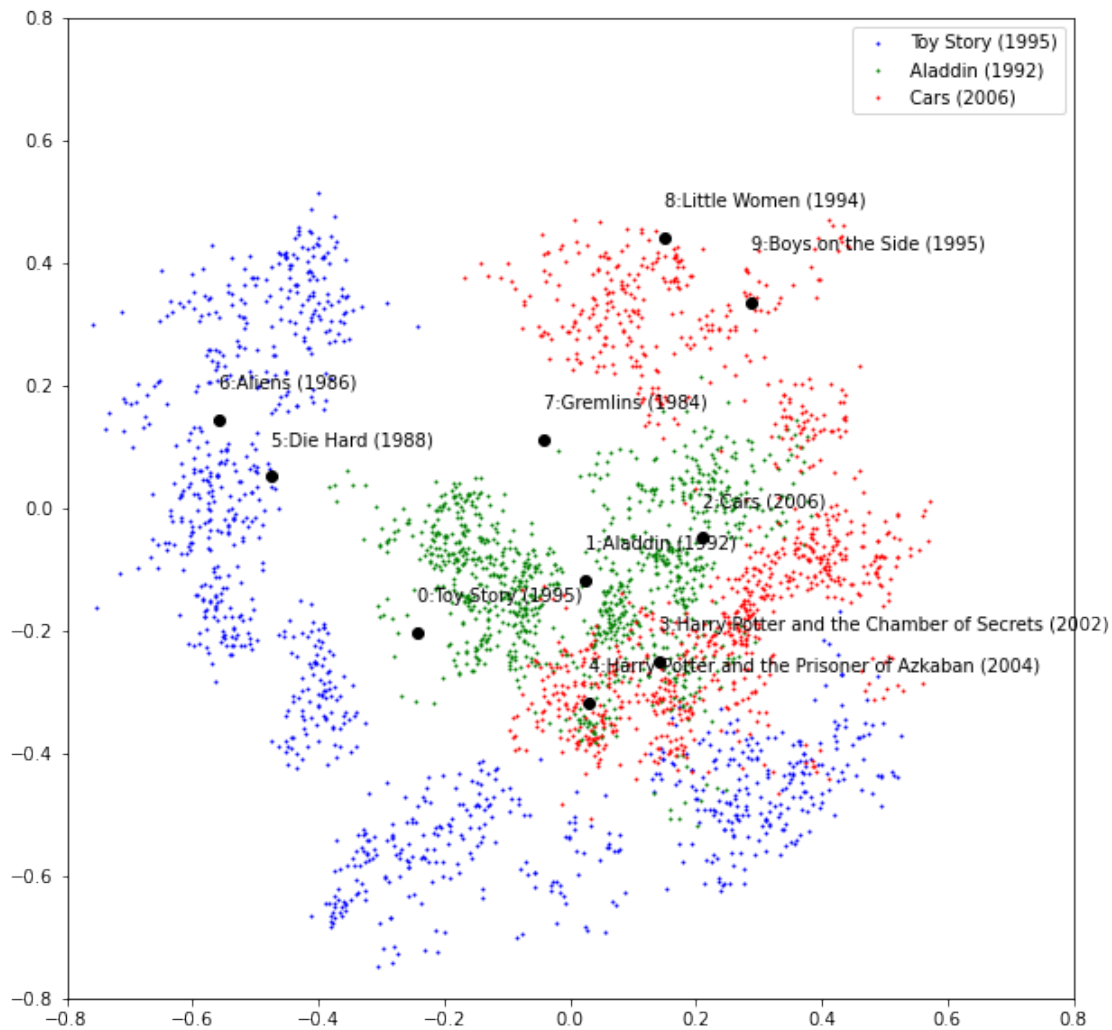


Familiar movies are located closely, like the two Harry Potter films; generally all the familiar films are close to each other.

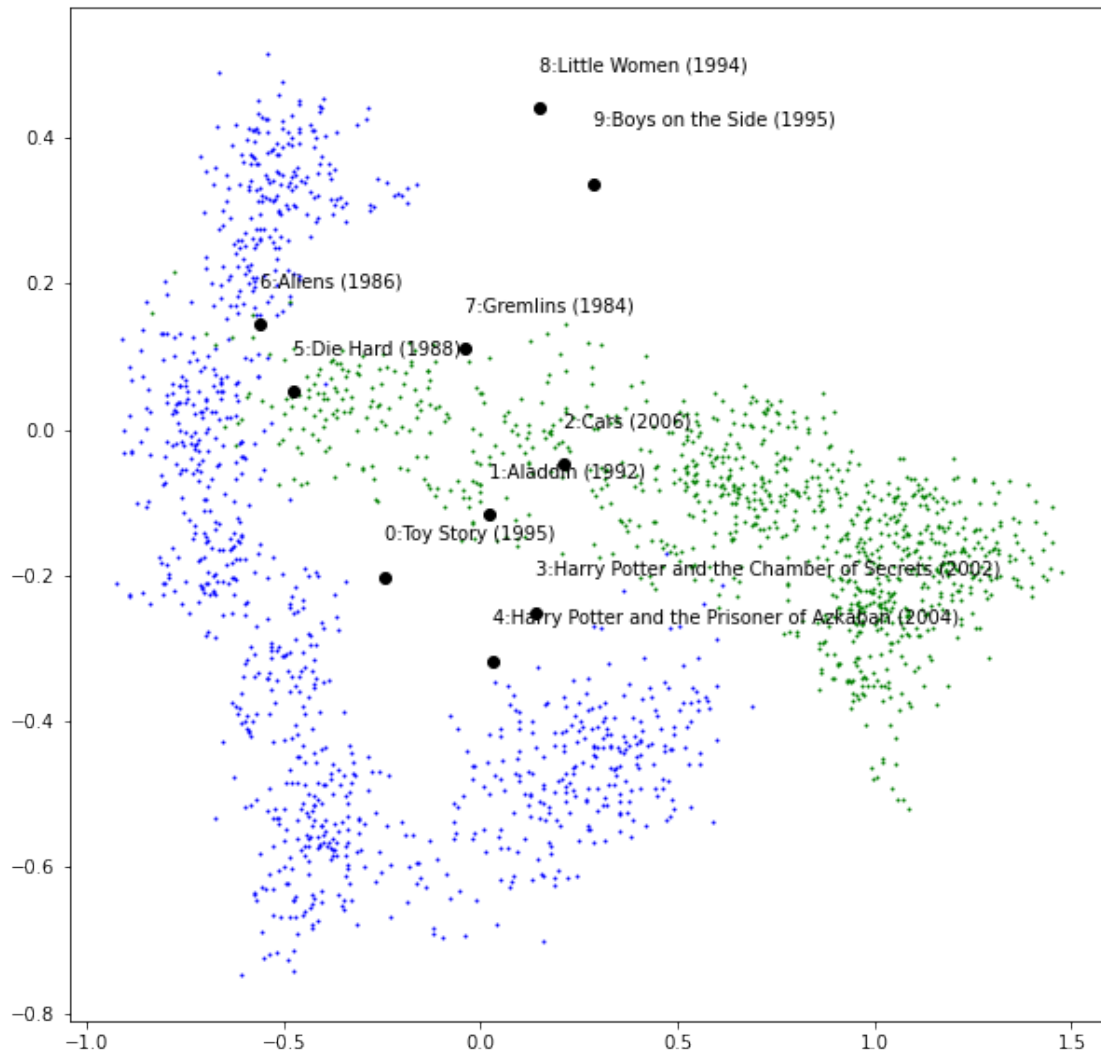
#### 1.4 Question 4

```
[9]: plt.figure(figsize=(10, 10))
plt.plot(fit["V"][0,0,:],fit["V"][0,1,:], 'bo',markersize = 1) # plot movie m's
↳ samples in blue ('b')
plt.plot(fit["V"][1,0,:],fit["V"][1,1,:], 'go',markersize = 1)
plt.plot(fit["V"][2,0,:],fit["V"][2,1,:], 'ro',markersize = 1)
for i in range(10):
    plt.plot(loc[i,0],loc[i,1], 'ko'); plt.text(loc[i,0],loc[i,1]+.05, str(i)+" :
↳ "+m_name[i]);
plt.axis([-0.8, 0.8, -0.8, 0.8])
plt.legend([m_name[0],m_name[1],m_name[2]],loc='upper right')
```

```
plt.show()
```



```
[10]: plt.figure(figsize=(10, 10))
plt.plot(fit["U"][0,0,:],fit["V"][0,1,:], 'bo',markersize = 1)
plt.plot(fit["U"][1,0,:],fit["V"][1,1,:], 'go',markersize = 1)
for i in range(10):
    plt.plot(loc[i,0],loc[i,1], 'ko'); plt.text(loc[i,0],loc[i,1]+.05, str(i)+"":
    ↪ "+m_name[i]);
plt.show()
```



```
[27]: #usr_1 = np.empty(10)
# for i in range(10):
#     sigma = np.exp(-(np.multiply(fit["U"][0,0,:], fit["V"][i,0,:
# → ])+fit["u"][0,0]+fit["v"][i,0]
#                                     +np.multiply(fit["U"][0,1,:], fit["V"][i,1,:])))
#     usr_1[i] = np.mean(0.5+10.0/(1+sigma))
#     if np.isnan(Xtr[0,i]):
#         usr_1[i] = np.nan
#print(usr_1)
Xtr[0], Xtr[1]
```

```
[27]: (array([nan,  7.,  7.,  5.,  6.,  7.,  8.,  4.,  5.,  1.]),
      array([ 5.,  7.,  7.,  7.,  5.,  5.,  5., nan,  5.,  7.]))
```

From the two arrays we can see that they are consistent with the graph, thus meaning that their

placement does match with their relative ratings.

## 1.5 Question 5

```
[14]: total_mean = np.empty([200,10])
      for j in range(200):
          usr_mean = np.empty(10)
          for i in range(10):
              sigma = np.exp(-(np.multiply(fit["U"][j,0,:],fit["V"][i,0,:
→])+fit["u"][j,0]+fit["v"][i,0]
                                     +np.multiply(fit["U"][j,1,:],fit["V"][i,1,:])))
              usr_mean[i] = np.mean(0.5+10.0/(1+sigma))
          total_mean[j] = usr_mean
      total_mean
```

```
[14]: array([[8.7248475 , 7.12785624, 6.4782991 , ..., 6.37176035, 5.26804569,
        4.20126715],
        [7.26181604, 7.33381228, 8.04461068, ..., 6.72596925, 7.75292828,
        7.47588234],
        [9.08385652, 8.62239436, 8.74391774, ..., 7.59386715, 7.1358272 ,
        6.92584842],
        ...,
        [5.6704809 , 4.64311521, 4.75710734, ..., 4.78432143, 5.76319353,
        4.52719618],
        [9.9601504 , 8.97223734, 8.24187979, ..., 8.93322761, 8.31237828,
        6.82832522],
        [9.40533545, 7.37916741, 6.00680635, ..., 7.41639906, 6.23454658,
        4.24420809]])
```

```
[29]: total_var = np.empty([200,10])
      for j in range(200):
          usr_var = np.empty(10)
          for i in range(10):
              sigma = np.exp(-(np.multiply(fit["U"][j,0,:],fit["V"][i,0,:
→])+fit["u"][j,0]+fit["v"][i,0]
                                     +np.multiply(fit["U"][j,1,:],fit["V"][i,1,:])))
              usr_var[i] = np.var(0.5+10.0/(1+sigma))
          total_var[j] = usr_var
      total_var
```

```
[29]: array([[0.01398553, 0.01664058, 0.02213784, ..., 0.01770776, 0.04904399,
        0.04040759],
        [0.04126929, 0.03292784, 0.0306557 , ..., 0.04055495, 0.0552104 ,
        0.06078804],
        [0.01354056, 0.01416333, 0.01409617, ..., 0.02239149, 0.05131963,
        0.05108587],
        ...,
        ...])
```

```
[0.043332 , 0.02181825, 0.03165208, ..., 0.02599672, 0.07920667,
 0.0731548 ],
[0.00468901, 0.02003307, 0.05444853, ..., 0.02062065, 0.05153662,
 0.10242672],
[0.02057592, 0.08299481, 0.11699863, ..., 0.08166395, 0.15594791,
 0.12694121]])
```

```
[16]: mse = np.empty(200)
      for i in range(200):
          usr_diff = np.empty(10)
          for j in range(10):
              if not np.isnan(Xte[i,j]):
                  usr_diff[j]=np.square(Xte[i,j]-total_mean[i,j])
          mse[i] = np.mean(usr_diff)
      mse
```

```
[16]: array([7.57088979e-03, 7.50928532e-03, 3.30682999e+00, 7.71316013e-02,
 3.29419321e-01, 2.38984207e-01, 1.67381735e+00, 6.14064686e-01,
 5.26128805e-01, 1.16853816e-01, 7.19202705e-01, 3.01881739e+00,
 5.27115643e+00, 8.29841766e-01, 5.72209332e+00, 7.47961605e-01,
 5.72209332e+00, 7.47961605e-01, 6.26775371e+00, 1.62197706e+00,
 6.28809409e+00, 1.66714456e+00, 6.27585569e+00, 1.62339904e+00,
 1.83213826e+00, 1.66172314e+00, 1.83213826e+00, 1.87436871e+00,
 2.72122058e+00, 1.91539018e+00, 2.53293160e+00, 2.43370372e+00,
 6.64413759e+00, 2.30979926e+00, 7.15621178e+00, 2.49796868e+00,
 5.62367594e+00, 2.53820831e+00, 5.55769933e+00, 2.46115013e+00,
 5.77920990e+00, 2.46115013e+00, 6.20152366e+00, 2.19933166e+00,
 7.34716951e+00, 2.34733903e+00, 7.24894033e+00, 1.84324129e+00,
 5.90467470e+00, 2.06330289e+00, 5.51278407e+00, 8.74151964e-01,
 5.41308559e+00, 9.58268879e-01, 7.53831459e+00, 1.07296408e+00,
 7.68316543e+00, 1.27502524e+00, 8.12139607e+00, 1.27502524e+00,
 5.13433813e+00, 1.35306120e+00, 5.42923997e+00, 1.46987588e+00,
 5.84353988e+00, 1.53383111e+00, 5.67601188e+00, 1.11888336e+00,
 7.71239757e+00, 1.67797578e+00, 7.91229899e+00, 1.48779634e+00,
 8.09144383e+00, 1.48779634e+00, 7.37870413e+00, 1.35546855e+00,
 7.61039608e+00, 1.84560118e+00, 7.53859083e+00, 1.79603815e+00,
 7.76590408e+00, 2.07093146e+00, 1.03723271e+01, 2.17981100e+00,
 1.03723271e+01, 1.94154667e+00, 8.56178362e+00, 1.99682716e+00,
 8.42571169e+00, 1.92410676e+00, 8.35790525e+00, 1.92410676e+00,
 2.30287013e+00, 9.34160197e-01, 2.10952491e+00, 1.04385815e+00,
 2.37013426e+00, 1.46774879e+00, 3.00291950e+00, 1.08237060e+00,
 3.98174654e+00, 8.40952578e-01, 3.95749186e+00, 2.13301970e+00,
 3.96378417e+00, 2.13301970e+00, 2.93414087e+00, 1.91993327e+00,
 2.95344584e+00, 1.71334373e+00, 3.00111139e+00, 3.51165695e+00,
 2.75636936e+00, 1.85420642e+00, 1.02713147e+00, 1.68881891e+00,
 9.81855861e-01, 2.93277015e+00, 7.47416884e-01, 3.41610176e+00,
 2.37033348e+00, 3.41610176e+00, 2.53545063e+00, 2.94374146e+00,
```

```

2.99635225e+00, 3.54264076e+00, 2.47698687e+00, 3.38475948e+00,
1.65736944e+00, 3.58693609e+00, 1.43170340e+00, 2.30676665e+00,
1.15848596e+00, 3.19887265e+00, 1.16746709e+00, 2.17325399e+00,
1.16746709e+00, 2.25473884e+00, 1.16746709e+00, 1.72222304e+00,
1.43659419e+00, 1.98559369e+00, 1.33224394e+00, 1.89944869e+00,
1.56582338e+00, 1.97250095e+00, 1.27815234e+00, 1.44803629e+00,
1.91880080e+00, 8.21973619e-01, 5.64117454e+00, 8.67763469e-01,
5.63389843e+00, 5.92582796e-01, 5.50654350e+00, 9.10403814e-01,
5.84220370e+00, 9.14859754e-01, 5.28552500e+00, 9.14625975e-01,
5.02375715e+00, 5.90212628e-01, 4.50462625e+00, 5.89522260e-01,
3.90735041e+00, 7.99055721e-01, 1.65041665e+00, 7.99055721e-01,
1.70869839e+00, 7.99055721e-01, 1.64257351e+00, 1.13129523e+00,
2.77689924e+00, 1.78431579e+00, 1.55751061e+00, 1.57445592e+00,
2.20203882e+00, 6.04558566e-01, 2.20203882e+00, 3.43982306e+00,
1.86322804e+00, 3.41412474e+00, 1.88729302e+00, 2.67997530e+00,
1.02001677e+00, 2.67997530e+00, 1.28749189e+00, 5.88741341e+00,
1.82220985e+00, 5.12610161e+00, 1.59665057e+00, 5.04235347e+00,
1.59665057e+00, 5.31183642e+00, 1.46476435e+00, 3.92206162e+00,
1.52727075e+00, 3.27711275e+00, 2.10943896e+00, 1.84544135e+00])

```

```

[17]: mse_ave = np.average(mse)
      mse_ave

```

```

[17]: 2.9465960594296496

```

```

[22]: np.where(total_var == np.min(total_var))

```

```

[22]: (array([65]), array([0]))

```

```

[24]: total_var[65], total_mean[65], Xtr[65]

```

```

[24]: (array([0.00078393, 0.00249704, 0.00496808, 0.00377053, 0.00154407,
              0.00117986, 0.00194749, 0.00479728, 0.02658802, 0.04087975]),
      array([10.22976373,  9.84131161,  9.59961685,  9.82470538, 10.10577692,
              10.21191193, 10.16655833,  9.46794224,  8.6184161 ,  7.99498254]),
      array([ 9.,  9.,  7.,  9.,  9., nan,  9.,  9.,  7., nan]))

```

From the arrays we can see that when we have a low variance, we have lower uncertainty, which means our prediction model works well.

```

[34]: np.where(total_var == np.max(total_var))

```

```

[34]: (array([133]), array([6]))

```

```

[35]: total_var[133], total_mean[133], Xtr[133]

```

```

[35]: (array([0.21706878, 0.02345061, 0.1732932 , 0.12359799, 0.04388355,
              0.84247819, 1.21436224, 0.03578298, 0.10677588, 0.31549469]),

```

```
array([6.86501842, 6.31290901, 6.69958624, 6.22051089, 6.75943805,  
       6.67191844, 6.32421409, 5.81916363, 6.51895788, 5.83650787]),  
array([nan, nan,  5.,  5.,  5., nan, nan, nan,  5., nan]))
```

From the arrays we can see that since most movies are not rated by this user, we have higher uncertainty about the prediction, thus resulting in a high variance.