Part 1: Setting up the Data

MovieLens 1M Dataset

MovieLens 1M movie ratings. Stable benchmark dataset. 1 million ratings from 6000 users on 4000 movies. Released 2/2003.

README.txt

ml-1m.zip (size: 6 MB, checksum)

Permalink: [https://grouplens.org/datasets/movielens/1m/ (https://grouplens.org/datasets/movielens/1m/)](https://grouplens.org/datasets/movielens/1m/)

In [1]:
```
import pandas as pd
from collections import defaultdict

from collections import defaultdict
import scipy
import scipy.optimize
import numpy
import random
```

In [2]:
```
ratings = pd.read_csv("ratings.dat",sep="::", header=None)
movies = pd.read_csv("movies.dat",sep="::", header=None)
users = pd.read_csv("users.dat",sep="::", header=None)
```

```
C:\Users\Ashok_Potti\.conda\envs\env_ashok\lib\site-packages\ipykernel_launch
er.py:1: ParserWarning: Falling back to the 'python' engine because the 'c' e
ngine does not support regex separators (separators > 1 char and different fr
om '\s+' are interpreted as regex); you can avoid this warning by specifying
engine='python'.
  """Entry point for launching an IPython kernel.
C:\Users\Ashok_Potti\.conda\envs\env_ashok\lib\site-packages\ipykernel_launch
er.py:2: ParserWarning: Falling back to the 'python' engine because the 'c' e
ngine does not support regex separators (separators > 1 char and different fr
om '\s+' are interpreted as regex); you can avoid this warning by specifying
engine='python'.

C:\Users\Ashok_Potti\.conda\envs\env_ashok\lib\site-packages\ipykernel_launch
er.py:3: ParserWarning: Falling back to the 'python' engine because the 'c' e
ngine does not support regex separators (separators > 1 char and different fr
om '\s+' are interpreted as regex); you can avoid this warning by specifying
engine='python'.
  This is separate from the ipykernel package so we can avoid doing imports u
ntil
```

In [3]:
```
ratings.shape
```

Out[3]:
```
(1000209, 4)
```

In [4]:
```
movies.shape
```
Out[4]:
```
(3883, 3)
```

In [5]:
```
users.shape
```
Out[5]:
```
(6040, 5)
```

In [6]:
```
ratings.isna().sum()
```
Out[6]:
```
0    0
1    0
2    0
3    0
dtype: int64
```

In [7]:
```
movies.isna().sum()
```
Out[7]:
```
0    0
1    0
2    0
dtype: int64
```

In [8]:
```
users.isna().sum()
```
Out[8]:
```
0    0
1    0
2    0
3    0
4    0
dtype: int64
```

In [9]:
```
ratings.isna().sum()
```
Out[9]:
```
0    0
1    0
2    0
3    0
dtype: int64
```

In [10]:
```
ratings.describe()
```

Out[10]:

	0	1	2	3
count	1.000209e+06	1.000209e+06	1.000209e+06	1.000209e+06
mean	3.024512e+03	1.865540e+03	3.581564e+00	9.722437e+08
std	1.728413e+03	1.096041e+03	1.117102e+00	1.215256e+07
min	1.000000e+00	1.000000e+00	1.000000e+00	9.567039e+08
25%	1.506000e+03	1.030000e+03	3.000000e+00	9.653026e+08
50%	3.070000e+03	1.835000e+03	4.000000e+00	9.730180e+08
75%	4.476000e+03	2.770000e+03	4.000000e+00	9.752209e+08
max	6.040000e+03	3.952000e+03	5.000000e+00	1.046455e+09

In [11]:
```
movies.describe()
```

Out[11]:

	0
count	3883.000000
mean	1986.049446
std	1146.778349
min	1.000000
25%	982.500000
50%	2010.000000
75%	2980.500000
max	3952.000000

In [12]:
```
users.describe()
```

Out[12]:

	0	2	3
count	6040.000000	6040.000000	6040.000000
mean	3020.500000	30.639238	8.146854
std	1743.742145	12.895962	6.329511
min	1.000000	1.000000	0.000000
25%	1510.750000	25.000000	3.000000
50%	3020.500000	25.000000	7.000000
75%	4530.250000	35.000000	14.000000
max	6040.000000	56.000000	20.000000

RATINGS FILE DESCRIPTION

All ratings are contained in the file "ratings.dat" and are in the following format:

UserID::MovieID::Rating::Timestamp

- UserIDs range between 1 and 6040
- MovieIDs range between 1 and 3952
- Ratings are made on a 5-star scale (whole-star ratings only)
- Timestamp is represented in seconds since the epoch as returned by time(2)
- Each user has at least 20 ratings

In [13]:
```
ratings.head(5)
```

Out[13]:

	0	1	2	3
0	1	1193	5	978300760
1	1	661	3	978302109
2	1	914	3	978301968
3	1	3408	4	978300275
4	1	2355	5	978824291

MOVIES FILE DESCRIPTION

Movie information is in the file "movies.dat" and is in the following format:

MovieID::Title::Genres

- Titles are identical to titles provided by the IMDB (including year of release)
- Genres are pipe-separated and are selected from the following genres:
  - Action
  - Adventure
  - Animation
  - Children's
  - Comedy
  - Crime
  - Documentary
  - Drama
  - Fantasy
  - Film-Noir
  - Horror
  - Musical
  - Mystery
  - Romance
  - Sci-Fi
  - Thriller
  - War
  - Western

In [14]:
```
movies.head(5)
```

Out[14]:

	0	1	2
0	1	Toy Story (1995)	Animation|Children's|Comedy
1	2	Jumanji (1995)	Adventure|Children's|Fantasy
2	3	Grumpier Old Men (1995)	Comedy|Romance
3	4	Waiting to Exhale (1995)	Comedy|Drama
4	5	Father of the Bride Part II (1995)	Comedy

USERS FILE DESCRIPTION

User information is in the file "users.dat" and is in the following format:

UserID::Gender::Age::Occupation::Zip-code

All demographic information is provided voluntarily by the users and is not checked for accuracy. Only users who have provided some demographic information are included in this data set.

- Gender is denoted by a "M" for male and "F" for female
- Age is chosen from the following ranges:
  - 1: "Under 18"
  - 18: "18-24"
  - 25: "25-34"
  - 35: "35-44"
  - 45: "45-49"
  - 50: "50-55"
  - 56: "56+"
- Occupation is chosen from the following choices:
  - 0: "other" or not specified
  - 1: "academic/educator"
  - 2: "artist"
  - 3: "clerical/admin"
  - 4: "college/grad student"
  - 5: "customer service"
  - 6: "doctor/health care"
  - 7: "executive/managerial"
  - 8: "farmer"
  - 9: "homemaker"
  - 10: "K-12 student"
  - 11: "lawyer"
  - 12: "programmer"
  - 13: "retired"
  - 14: "sales/marketing"
  - 15: "scientist"
  - 16: "self-employed"
  - 17: "technician/engineer"
  - 18: "tradesman/craftsman"
  - 19: "unemployed"
  - 20: "writer"

In [15]:
```
users.head(5)
```

Out[15]:

	0	1	2	3	4
0	1	F	1	10	48067
1	2	M	56	16	70072
2	3	M	25	15	55117
3	4	M	45	7	02460
4	5	M	25	20	55455

In [16]:
```
dataset=ratings[[0,1,2]]
dataset = dataset.rename(columns={0: "userId", 1: "movieId", 2 : "rating"})
```

In [17]:
```
dataset
```

Out[17]:

	userId	movieId	rating
0	1	1193	5
1	1	661	3
2	1	914	3
3	1	3408	4
4	1	2355	5
...	...	...	...
1000204	6040	1091	1
1000205	6040	1094	5
1000206	6040	562	5
1000207	6040	1096	4
1000208	6040	1097	4

1000209 rows × 3 columns

In [18]:
```
print("Unique Users {}, Unique Moviews {}".format(dataset.userId.nunique(),dat
aset.movieId.nunique()))
```

```
Unique Users 6040, Unique Moviews 3706
```

In [19]:
```
reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)

for idx, row in dataset.iterrows():
    userId,movieId = row['userId'], row['movieId']
    reviewsPerUser[userId].append(row)
    reviewsPerItem[movieId].append(row)
```

In [20]:
```
print(len(reviewsPerUser), len(reviewsPerItem))
```

```
6040 3706
```

In [21]:
```
N = len(dataset)
nUsers = len(reviewsPerUser)
nItems = len(reviewsPerItem)

users = list(reviewsPerUser.keys())
items = list(reviewsPerItem.keys())

userBiases = defaultdict(float)
itemBiases = defaultdict(float)

def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

In [22]:
```
alpha = dataset['rating'].mean()
print("alpha (or mean): {}".format(alpha))

alwaysPredictMean = [alpha] * len(dataset)
labels = dataset['rating']

print("MSE using alpha (or mean) prediction: {}".format(MSE(alwaysPredictMean,
labels)))
```

```
alpha (or mean): 3.581564453029317
MSE using alpha (or mean) prediction: 1.2479152852902136
```

In [23]:
```
userBiases = defaultdict(float)
itemBiases = defaultdict(float)
userGamma = {}
itemGamma = {}

K = 5
```

In [24]:
```
for u in reviewsPerUser:
    userGamma[u] = [random.random() * 0.1 - 0.05 for k in range(K)]

for i in reviewsPerItem:
    itemGamma[i] = [random.random() * 0.1 - 0.05 for k in range(K)]
```

In [25]:

```
def unpack(theta):
    global alpha
    global userBiases
    global itemBiases
    global userGamma
    global itemGamma
    index = 0
    alpha = theta[index]
    index += 1
    userBiases = dict(zip(users, theta[index:index+nUsers]))
    index += nUsers
    itemBiases = dict(zip(items, theta[index:index+nItems]))
    index += nItems
    for u in users:
        userGamma[u] = theta[index:index+K]
        index += K
    for i in items:
        itemGamma[i] = theta[index:index+K]
        index += K
```

In [26]:
```
def inner(x, y):
    return sum([a*b for a,b in zip(x,y)])


def prediction(user, item):
    return alpha + userBiases[user] + itemBiases[item] + inner(userGamma[user
], itemGamma[item])


def cost(theta, labels, lamb):
    unpack(theta)
    predictions = [prediction(d['userId'], d['movieId']) for idx, d in dataset
.iterrows()]
    cost = MSE(predictions, labels)
    print("MSE = " + str(cost))
    for u in users:
        cost += lamb*userBiases[u]**2
        for k in range(K):
            cost += lamb*userGamma[u][k]**2
    for i in items:
        cost += lamb*itemBiases[i]**2
        for k in range(K):
            cost += lamb*itemGamma[i][k]**2
    return cost


def derivative(theta, labels, lamb):
    unpack(theta)
    N = len(dataset)
    dalpha = 0
    dUserBiases = defaultdict(float)
    dItemBiases = defaultdict(float)
    dUserGamma = {}
    dItemGamma = {}
    for u in reviewsPerUser:
        dUserGamma[u] = [0.0 for k in range(K)]
    for i in reviewsPerItem:
        dItemGamma[i] = [0.0 for k in range(K)]
    for idx, d in dataset.iterrows():
        u,i = d['userId'], d['movieId']
        pred = prediction(u, i)
        diff = pred - d['rating']
        dalpha += 2/N*diff
        dUserBiases[u] += 2/N*diff
        dItemBiases[i] += 2/N*diff
        for k in range(K):
            dUserGamma[u][k] += 2/N*itemGamma[i][k]*diff
            dItemGamma[i][k] += 2/N*userGamma[u][k]*diff
    for u in userBiases:
        dUserBiases[u] += 2*lamb*userBiases[u]
        for k in range(K):
            dUserGamma[u][k] += 2*lamb*userGamma[u][k]
    for i in itemBiases:
        dItemBiases[i] += 2*lamb*itemBiases[i]
        for k in range(K):
            dItemGamma[i][k] += 2*lamb*itemGamma[i][k]
```

```
    dtheta = [dalpha] + [dUserBiases[u] for u in users] + [dItemBiases[i] for
i in items]
    for u in users:
        dtheta += dUserGamma[u]
    for i in items:
        dtheta += dItemGamma[i]
    return numpy.array(dtheta)
```

In [27]:
```
MSE(alwaysPredictMean, labels) #Same as our previous baseline
```

Out[27]: 1.2479152852902136

In [28]:
```
x,f,d = scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + # Initialize alpha
                                   [0.0]*(nUsers+nItems) + # Initialize beta
                                   [random.random() * 0.1 - 0.05 for k in rang
e(K*(nUsers+nItems))], # Gamma
                             derivative, args = (labels, 0.001), maxfun = 10,
maxiter = 10)
```

```
MSE = 1.2479127681344244
MSE = 1.2170646604649966
MSE = 5.026653410052432
MSE = 1.2033851469689592
MSE = 1.0529878870454181
MSE = 1.0521619128942303
MSE = 1.0489821231375172
MSE = 1.0186860648373264
MSE = 1.018857594768604
MSE = 1.0199396991226464
MSE = 1.0202800124177205
```

Predicted rating by a user for a movie

In [135]:
```
prediction(6040,1193)
```

Out[135]: 4.044457189238609

In [133]:
```
def get_ten_item(userId):
    user_rating = []
    user_item = []
    for movieId in range(1,nItems):
        try:
            user_rating.append(prediction(userId,movieId))
        except:
            user_rating.append(0)
        user_item.append(movieId)

    zipped = dict(zip(user_item, user_rating))
    return sorted(zipped.items(), key=lambda x: x[1], reverse=True)
```

Top 10 movie recommendation for userID = 3333

In [134]:
```
res = get_ten_item(3333)
movies.loc[[i for i, _ in res[0:10]]]
```

Out[134]:

	0	1	2
318	321	Strawberry and Chocolate (Fresa y chocolate) (...	Drama
260	263	Ladybird Ladybird (1994)	Drama
858	869	Kansas City (1996)	Crime
527	531	Secret Garden, The (1993)	Children's|Drama
1198	1216	Big Blue, The (Le Grand Bleu) (1988)	Adventure|Romance
50	51	Guardian Angel (1994)	Action|Drama|Thriller
2762	2831	Dog of Flanders, A (1999)	Drama
2858	2927	Brief Encounter (1946)	Drama|Romance
593	597	Pretty Woman (1990)	Comedy|Romance
2028	2097	Something Wicked This Way Comes (1983)	Children's|Horror

In [ ]: