In [1]:

```python
import numpy
import urllib
import scipy.optimize
import random
from collections import defaultdict # Dictionaries with default values
import nltk
import string
from nltk.stem.porter import *
from sklearn import linear_model
import ast
import gzip
def Jaccard(book1,book2,bookAllUser):
    s1=[]
    for d in bookAllUser[book1]:
        s1.append(d['userID'])
    s2=[]
    for d in bookAllUser[book2]:
        s2.append(d['userID'])
    s1=set(s1)
    s2=set(s2)
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer / denom
f = gzip.open("train_Interactions.csv.gz", 'rt', encoding="utf8")
header = f.readline()
header = header.strip().split(',')
datatrain = []
datavalid = []
count=0
lenAll = 200000
for line in f:
    fields = line.strip().split(',')
    d = dict(zip(header, fields))
    if count <lenAll*0.95 :
        datatrain.append(d)
    else:
        datavalid.append(d)
    count=count+1

userReadBook = defaultdict(list)
bookAllUser  = defaultdict(list)

for d in datatrain:
    user,book,r =d['userID'],d['bookID'],d['rating']
    userReadBook[user].append(d)
    bookAllUser[book].append(d)
ratingMean = sum([float(d['rating']) for d in datatrain]) / len(datatrain)

N = len(datatrain)
nUsers = len(userReadBook)
nBooks = len(bookAllUser)
users = list(userReadBook)
books = list(bookAllUser)

alpha = ratingMean

userBiases = defaultdict(float)
bookBiases = defaultdict(float)

def prediction(user, book):
    return alpha + userBiases[user] + bookBiases[book]

def predictRating(user,item):
    ratings = []
    similarities = []
    for d in userReadBook[user]:
        i2 = d['bookID']
        if i2 == item: continue
        ratings.append(d['rating'])
        similarities.append(Jaccard(item,i2,bookAllUser))
    if (sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings,similarities)]
        return sum(weightedRatings) / sum(similarities)
    else:
        # User hasn't rated any similar items
        return ratingMean

def unpack(theta):
    global alpha
    global userBiases
    global bookBiases
    alpha = theta[0]
    userBiases = dict(zip(users, theta[1:nUsers+1]))
    bookBiases = dict(zip(books, theta[1+nUsers:]))

def cost(theta, labels, lamb):
    unpack(theta)
    predictions = [prediction(d['userID'], d['bookID']) for d in datatrain]
    cost = MSE(predictions, labels)
    print("MSE = " + str(cost))
    for u in userBiases:
        cost += lamb*userBiases[u]**2
    for i in bookBiases:
        cost += lamb*bookBiases[i]**2
    return cost

def derivative(theta, labels, lamb):
    unpack(theta)
    N = len(datatrain)
    dalpha = 0
    dUserBiases = defaultdict(float)
    dbookBiases = defaultdict(float)
    for d in datatrain:
        u,i = d['userID'], d['bookID']
        pred = prediction(u, i)
        diff = pred - float(d['rating'])
        dalpha += 2/N*diff
        dUserBiases[u] += 2/N*diff
        dbookBiases[i] += 2/N*diff
    for u in userBiases:
        dUserBiases[u] += 2*lamb*userBiases[u]
    for i in bookBiases:
        dbookBiases[i] += 2*lamb*bookBiases[i]
    dtheta = [dalpha] + [dUserBiases[u] for u in users] + [dbookBiases[i] for i in books]
    return numpy.array(dtheta)

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

labels = [float(d['rating']) for d in datatrain]
scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + [0.0]*(nUsers+nBooks),derivative, args = (labels,2e-5))
predictions = open("predictions_Rating.txt", 'w')
for l in open("pairs_Rating.txt"):
  if l.startswith("userID"):
    #header
    predictions.write(l)
    continue
  user,book = l.strip().split('-')
  if user in users and book in books:
        prediction_valid=prediction(user,book)
  else:

        prediction_valid=predictRating(user,book)
  predictions.write(user + '-' + book + ","+str(prediction_valid)+"\n")
predictions.close()
#my kaggle name is HumphreySD, and my score is 1.14656, my email address is haz013@eng.ucsd.edu
#I choose lamuda as 2e-5,MSE is 0.92566
```

```
MSE = 1.4735475011336192
MSE = 1.4560931393014562
MSE = 1.392284346401447
MSE = 8.08588938870108
MSE = 1.3695427812836598
MSE = 1.202888290365505
MSE = 1.201333071387938
MSE = 1.195213051215806
MSE = 1.1723466623360814
MSE = 1.0588858605572056
MSE = 1.0109866093480053
MSE = 0.9792061553479576
MSE = 0.9641404735702934
MSE = 0.952691455209057
MSE = 0.9382161957927877
MSE = 0.9324830107667381
MSE = 0.9310949475578871
MSE = 0.9306016200670771
MSE = 0.9519124072854905
MSE = 0.9305585881131831
MSE = 0.9297759227007407
MSE = 0.9285850480621985
MSE = 0.9270178994992827
MSE = 0.9263009232898817
MSE = 0.9257156932469571
MSE = 0.9249547536300587
MSE = 0.9256445353993737
MSE = 0.9260333750758728
MSE = 0.9262261853216237
MSE = 0.9262792851467087
MSE = 0.9319734642514613
MSE = 0.9262812564150997
MSE = 0.9261590468682027
MSE = 0.9258838932958376
MSE = 0.9257510823640084
MSE = 0.9252038589117146
MSE = 0.9255926057178915
MSE = 0.9256206677681702
MSE = 0.9256531516906248
MSE = 0.9256757115957949
MSE = 0.9256977359736026
MSE = 0.9256076963007648
MSE = 0.925651556577265
MSE = 0.9256600242778392
```

In [ ]: