In [1]:
```
import gzip
import random
from tqdm import tqdm
from collections import defaultdict
import numpy as np
import time
import scipy
import scipy.optimize
```

In [2]:
```
def readGz(path):
    for l in gzip.open(path, 'rt'):
        yield eval(l)
```

In [3]:
```
def readCSV(path):
    f = gzip.open(path, 'rt')
    f.readline() # Skip Header
    for l in f:
        yield l.strip(). split(',')
```

In [4]:
```
bookData = []
for user, book, rating in readCSV("train_Interactions.csv.gz"):
    bookData.append([user, book, int(rating)])
```

In [5]:
```
bookData[:10]
```

Out[5]:
```
[['u79354815', 'b14275065', 4],
 ['u56917948', 'b82152306', 5],
 ['u97915914', 'b44882292', 5],
 ['u49688858', 'b79927466', 5],
 ['u08384938', 'b05683889', 2],
 ['u13530776', 'b86375465', 4],
 ['u46307273', 'b92838791', 5],
 ['u18524450', 'b35165110', 2],
 ['u69700998', 'b17128180', 5],
 ['u43359569', 'b34596567', 5]]
```

Tasks (Read prediction)

1. Although we have built a validation set, it only consists of positive samples.

For this task we also need examples of user/item pairs that weren't read. For each entry (user,book) in the validation set, sample a negative entry by randomly choosing a book that user hasn't read. Evaluate the performance (accuracy) of the baseline model on the validation set you have built.

Ans:
Accurarcy of validation set: 0.7484

In [66]:
```
# Separate traing and validation set
print(len(bookData))
numTrainSet = 190000
bookDataTrain = bookData[:numTrainSet]
bookDataYTrain =  [1] * len(bookDataTrain)
bookDataValidPos = bookData[numTrainSet:]
bookDataYValidPos = [1] * len(bookDataValidPos)
print(len(bookDataTrain))
print(len(bookDataYTrain))
print(len(bookDataValidPos))
```

```
200000
190000
190000
10000
```

In [67]:
```
bookUniqueIds = set()
for user, book, rating in bookData:
    bookUniqueIds.add(book)
bookUniqueIdsList = list(bookUniqueIds)
#bookUniqueIdsList
len(bookUniqueIdsList)
```

Out[67]: 7170

In [68]:
```
bookReadByUserIds = {}
for user, book, rating in bookData:
    if user in bookReadByUserIds:
        bookReadByUserIds[user].add(book)
    else:
        bookReadByUserIds[user] = set()
        bookReadByUserIds[user].add(book)
#bookReadByUserIds
```

In [69]:
```
def getNegativeEntries():
    bookDataValidNeg = []
    bookDataYValidNeg = [0] * len(bookDataValidPos)
    for user, book, rating in bookDataValidPos:
        #while True:
            #unreadBookId = random.choice(bookUniqueIdsList)
        # For consistent validation set
        for unreadBookId in bookUniqueIdsList:
            if unreadBookId not in bookReadByUserIds[user]:
                bookDataValidNeg.append([user, unreadBookId, "-1"])
                break
    return bookDataValidNeg, bookDataYValidNeg
```

In [70]:
```
bookDataValidNeg, bookDataYValidNeg = getNegativeEntries()
bookDataValid = bookDataValidPos + bookDataValidNeg
bookDataYValid = bookDataYValidPos + bookDataYValidNeg
print(len(bookDataValid))
print(len(bookDataYValid))
```

```
20000
20000
```

In [71]:
```
def writeOutValidationSet(nameTag):
    timestr = time.strftime("%Y%m%d_%H%M%S")
    fileName = "validation_set_" + timestr + "_validMSE_" + nameTag + ".txt"
    outFile = open(fileName, 'w')

    # Write out current validation set
    for data, y in zip(bookDataValid, bookDataYValid):
        outFile.write(data[0] + '-' + data[1] + "," + str(y) + "\n")

    outFile.close()
```

In [72]:
```
def getBaselinePred(Xdata, threshold):
    ### Would-read baseline: just rank which books are popular and which are not, and return '1'
 if a book is among the top-ranked
    bookCount = defaultdict(int)
    totalRead = 0

    for user, book, _ in readCSV("train_Interactions.csv.gz"):
        bookCount[book] += 1
        totalRead += 1

    mostPopular = [(bookCount[x], x) for x in bookCount]
    mostPopular.sort()
    mostPopular.reverse()

    return1 = set()
    count = 0
    for bkc, bkId in mostPopular:
        count += bkc
        return1.add(bkId)
        if count > totalRead/threshold: break

    # Make prediction
    prediction = []
    for uId, bId, rating in Xdata:
        if bId in return1:
            prediction.append(1)
        else:
            prediction.append(0)

    return prediction
```

In [73]:
```
def getAcc(pred, golden):
    correctPredictions = [p==y for p, y in zip(pred, golden)]
    return sum(correctPredictions) / len(golden)
```

In [74]:
```
def getTPR(pred, golden):
    TP = sum([(p and l) for (p,l) in zip(pred, golden)])
    FN = sum([(not p and l) for (p,l) in zip(pred, golden)])
    return TP / (TP + FN)
```

In [75]:
```
def getTNR(pred, golden):
    FP = sum([(p and not l) for (p,l) in zip(pred, golden)])
    TN = sum([(not p and not l) for (p,l) in zip(pred, golden)])
    return TN / (TN + FP)
```

In [76]:
```
def getMetrics(pred, golden):
    TNR = getTNR(pred, golden)
    TPR = getTPR(pred, golden)
    acc = getAcc(pred, golden)
    return (acc, TPR, TNR)
```

In [77]:
```
predBookDataYValid = getBaselinePred(bookDataValid, 2.0)
print(len(predBookDataYValid))
print(len(bookDataYValid))

# Accurarcy
predBookDataValidMSE = getAcc(predBookDataYValid, bookDataYValid)
print(predBookDataValidMSE)
```

```
20000
20000
0.7484
```

In [18]:
```
writeOutValidationSet(str(predBookDataValidMSE))
```

2. The existing 'read prediction' baseline just returns True if the item in question is 'popular,' using a threshold of the 50th percentile of popularity (totalRead/2).

Assuming that the 'non-read' test examples are a random sample of user-book pairs, this threshold may not be the best one. See if you can find a better threshold and report its performance on your validatin set.

Ans:
Threshold = 1.250000, i.e. 80th percentile of popularity
Accurarcy on validation set: 0.899000

In [19]:
```
for thres in np.arange(1, 3, 0.05):
    #predBookDataYTrain = getBaselinePred(bookDataTrain, thres)
    # Accurarcy for training set
    #correctPredictions = [p==y for p, y in zip(predBookDataYTrain, bookDataYTrain)]
    #print("Training: t=%f, acc=%f" % (thres, sum(correctPredictions) / len(bookDataYTrain)) )

    predBookDataYValid = getBaselinePred(bookDataValid, thres)
    # Accurarcy for validation set
    acc, TPR, TNR = getMetrics(predBookDataYValid, bookDataYValid)
    print("Validataion: t=%f, acc=%f, TPR=%f, TNR=%f" % (thres, acc, TPR, TNR) )
```

```
Validataion: t=1.000000, acc=0.500000, TPR=1.000000, TNR=0.000000
Validataion: t=1.050000, acc=0.478050, TPR=0.954800, TNR=0.001300
Validataion: t=1.100000, acc=0.454450, TPR=0.907600, TNR=0.001300
Validataion: t=1.150000, acc=0.433650, TPR=0.866000, TNR=0.001300
Validataion: t=1.200000, acc=0.416400, TPR=0.831500, TNR=0.001300
Validataion: t=1.250000, acc=0.899000, TPR=0.798000, TNR=1.000000
Validataion: t=1.300000, acc=0.881850, TPR=0.763700, TNR=1.000000
Validataion: t=1.350000, acc=0.866700, TPR=0.733400, TNR=1.000000
Validataion: t=1.400000, acc=0.853350, TPR=0.706700, TNR=1.000000
Validataion: t=1.450000, acc=0.841500, TPR=0.683000, TNR=1.000000
Validataion: t=1.500000, acc=0.829650, TPR=0.659300, TNR=1.000000
Validataion: t=1.550000, acc=0.818400, TPR=0.636800, TNR=1.000000
Validataion: t=1.600000, acc=0.808550, TPR=0.617100, TNR=1.000000
Validataion: t=1.650000, acc=0.798950, TPR=0.597900, TNR=1.000000
Validataion: t=1.700000, acc=0.790300, TPR=0.580600, TNR=1.000000
Validataion: t=1.750000, acc=0.782900, TPR=0.565800, TNR=1.000000
Validataion: t=1.800000, acc=0.775400, TPR=0.550800, TNR=1.000000
Validataion: t=1.850000, acc=0.767550, TPR=0.535100, TNR=1.000000
Validataion: t=1.900000, acc=0.761150, TPR=0.522300, TNR=1.000000
Validataion: t=1.950000, acc=0.753950, TPR=0.507900, TNR=1.000000
Validataion: t=2.000000, acc=0.748400, TPR=0.496800, TNR=1.000000
Validataion: t=2.050000, acc=0.742450, TPR=0.484900, TNR=1.000000
Validataion: t=2.100000, acc=0.736100, TPR=0.472200, TNR=1.000000
Validataion: t=2.150000, acc=0.730100, TPR=0.460200, TNR=1.000000
Validataion: t=2.200000, acc=0.724200, TPR=0.448400, TNR=1.000000
Validataion: t=2.250000, acc=0.720400, TPR=0.440800, TNR=1.000000
Validataion: t=2.300000, acc=0.716100, TPR=0.432200, TNR=1.000000
Validataion: t=2.350000, acc=0.711450, TPR=0.422900, TNR=1.000000
Validataion: t=2.400000, acc=0.706750, TPR=0.413500, TNR=1.000000
Validataion: t=2.450000, acc=0.702800, TPR=0.405600, TNR=1.000000
Validataion: t=2.500000, acc=0.699050, TPR=0.398100, TNR=1.000000
Validataion: t=2.550000, acc=0.695450, TPR=0.390900, TNR=1.000000
Validataion: t=2.600000, acc=0.691200, TPR=0.382400, TNR=1.000000
Validataion: t=2.650000, acc=0.687300, TPR=0.374600, TNR=1.000000
Validataion: t=2.700000, acc=0.683650, TPR=0.367300, TNR=1.000000
Validataion: t=2.750000, acc=0.680600, TPR=0.361200, TNR=1.000000
Validataion: t=2.800000, acc=0.677000, TPR=0.354000, TNR=1.000000
Validataion: t=2.850000, acc=0.674550, TPR=0.349100, TNR=1.000000
Validataion: t=2.900000, acc=0.671150, TPR=0.342300, TNR=1.000000
Validataion: t=2.950000, acc=0.668200, TPR=0.336400, TNR=1.000000
```

In [20]:
```
def writeOutBaselinePred(threshold):
    ### Would-read baseline: just rank which books are popular and which are not, and return '1'
 if a book is among the top-ranked
    bookCount = defaultdict(int)
    totalRead = 0

    for user, book, _ in readCSV("train_Interactions.csv.gz"):
        bookCount[book] += 1
        totalRead += 1

    mostPopular = [(bookCount[x], x) for x in bookCount]
    mostPopular.sort()
    mostPopular.reverse()

    return1 = set()
    count = 0
    for bkc, bkId in mostPopular:
        count += bkc
        return1.add(bkId)
        if count > totalRead/threshold: break

    predOutFile = open("predictions_Read.txt", 'w')
    for l in open("pairs_Read.txt", 'r'):
        if l.startswith("userID"):
            #header
            predOutFile.write(l)
            continue
        uId, bId = l.strip().split('-')
        if bId in return1:
            predOutFile.write(uId + '-' + bId + ",1\n")
        else:
            predOutFile.write(uId + '-' + bId + ",0\n")
    predOutFile.close()
```

In [21]:
```
writeOutBaselinePred(1.7)
```

3. A stronger baseline than the one provided might make use of the Jaccard similarity (or another similarity metric). Given a pair (u, b) in the validation set, consider all training items b′ that user u has read.

For each, compute the Jaccard similarity between b and b′, i.e., users (in the training set) who have read ′ b and users who have read b . Predict as 'read' if the maximum of these Jaccard similarities exceeds a threshold (you may choose the threshold that works best). Report the performance on your validation set (1 mark).

Ans:
Choose the threshold with the best accurarcy on validation set: threshold = 0.010000
Accurarcy on the validation set: 0.689850

In [78]:
```
usersPerItem = defaultdict(set)
itemsPerUser = defaultdict(set)
```

In [79]:
```
bookDataTrain[:10]
```

Out[79]:
```
[['u79354815', 'b14275065', 4],
 ['u56917948', 'b82152306', 5],
 ['u97915914', 'b44882292', 5],
 ['u49688858', 'b79927466', 5],
 ['u08384938', 'b05683889', 2],
 ['u13530776', 'b86375465', 4],
 ['u46307273', 'b92838791', 5],
 ['u18524450', 'b35165110', 2],
 ['u69700998', 'b17128180', 5],
 ['u43359569', 'b34596567', 5]]
```

In [80]:
```
for d in bookDataTrain:
    usersPerItem[d[1]].add(d[0])
    itemsPerUser[d[0]].add(d[1])
```

In [81]:
```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer / denom
```

In [82]:
```
def pairSimilarity(u, b):
    similarities = []
    users = usersPerItem[b]
    candidateItems = itemsPerUser[u]
    for b2 in candidateItems:
        if b2 == b: continue
        sim = Jaccard(users, usersPerItem[b2])
        similarities.append((sim,b2))
    similarities.sort(reverse=True)
    return similarities
```

In [83]:
```
def getJaccardPred(Xdata, threshold):
    # Make prediction
    prediction = []
    for uId, bId, rating in Xdata:
        #print("Query: userId: %s, bookId: %s" % (uId, bId))
        #print(itemsPerUser[uId])
        sim = pairSimilarity(uId, bId)
        #print(sim[0][0])

        if sim and sim[0][0] > threshold:
            prediction.append(1)
        else:
            prediction.append(0)
    return prediction
```

In [84]:
```
for thres in np.arange(0, 0.03, 0.001):
    predBookDataYValid = getJaccardPred(bookDataValid, thres)
    # Accurarcy for validation set
    acc, TPR, TNR = getMetrics(predBookDataYValid, bookDataYValid)
    print("Validataion: t=%f, acc=%f, TPR=%f, TNR=%f" % (thres, acc, TPR, TNR) )
```

```
Validataion: t=0.000000, acc=0.665700, TPR=0.920400, TNR=0.411000
Validataion: t=0.001000, acc=0.665700, TPR=0.920400, TNR=0.411000
Validataion: t=0.002000, acc=0.665700, TPR=0.920400, TNR=0.411000
Validataion: t=0.003000, acc=0.671850, TPR=0.919100, TNR=0.424600
Validataion: t=0.004000, acc=0.670100, TPR=0.915600, TNR=0.424600
Validataion: t=0.005000, acc=0.672850, TPR=0.908800, TNR=0.436900
Validataion: t=0.006000, acc=0.677800, TPR=0.898900, TNR=0.456700
Validataion: t=0.007000, acc=0.682100, TPR=0.885500, TNR=0.478700
Validataion: t=0.008000, acc=0.686800, TPR=0.869300, TNR=0.504300
Validataion: t=0.009000, acc=0.689350, TPR=0.848700, TNR=0.530000
Validataion: t=0.010000, acc=0.689850, TPR=0.819700, TNR=0.560000
Validataion: t=0.011000, acc=0.681400, TPR=0.788900, TNR=0.573900
Validataion: t=0.012000, acc=0.679250, TPR=0.756100, TNR=0.602400
Validataion: t=0.013000, acc=0.667650, TPR=0.713700, TNR=0.621600
Validataion: t=0.014000, acc=0.654550, TPR=0.674100, TNR=0.635000
Validataion: t=0.015000, acc=0.639950, TPR=0.627500, TNR=0.652400
Validataion: t=0.016000, acc=0.631200, TPR=0.583500, TNR=0.678900
Validataion: t=0.017000, acc=0.617300, TPR=0.539200, TNR=0.695400
Validataion: t=0.018000, acc=0.604650, TPR=0.496700, TNR=0.712600
Validataion: t=0.019000, acc=0.591600, TPR=0.452600, TNR=0.730600
Validataion: t=0.020000, acc=0.580350, TPR=0.407600, TNR=0.753100
Validataion: t=0.021000, acc=0.567900, TPR=0.370000, TNR=0.765800
Validataion: t=0.022000, acc=0.554250, TPR=0.335200, TNR=0.773300
Validataion: t=0.023000, acc=0.552200, TPR=0.304400, TNR=0.800000
Validataion: t=0.024000, acc=0.543850, TPR=0.272200, TNR=0.815500
Validataion: t=0.025000, acc=0.541250, TPR=0.241200, TNR=0.841300
Validataion: t=0.026000, acc=0.531850, TPR=0.218900, TNR=0.844800
Validataion: t=0.027000, acc=0.524350, TPR=0.202000, TNR=0.846700
Validataion: t=0.028000, acc=0.520700, TPR=0.176800, TNR=0.864600
Validataion: t=0.029000, acc=0.517750, TPR=0.159500, TNR=0.876000
```

In [42]:
```
def writeOutJaccardPred(threshold):
    predOutFile = open("predictions_Read.txt", 'w')
    bookDataTest = []

    # Read Testing set
    for l in open("pairs_Read.txt", 'r'):
        if l.startswith("userID"):
            #header
            predOutFile.write(l)
            continue
        uId, bId = l.strip().split('-')
        bookDataTest.append([uId, bId, -1])

    # Predict by Jaccard
    bookDataYTest = getJaccardPred(bookDataTest, threshold)

    # Write out prediction result
    for data, y in zip(bookDataTest, bookDataYTest):
        predOutFile.write(data[0] + '-' + data[1] + "," + str(y) + "\n")

    predOutFile.close()
```

In [43]:
```
writeOutJaccardPred(0.011)
```

In [44]:
```
writeOutJaccardPred(0.008)
```

In [45]:
```
writeOutJaccardPred(0.013)
```

4. Improve the above predictor by incorporating both a Jaccard-based threshold and a popularity based threshold. Report the performance on your validation set.

Ans:
The proposed methods: to mix the results from various of threshold of the two predictors by AND or OR.
Mix1 (OR):
Accurarcy: 0.86185
TPR: 0.8591
TNR: 0.8646
Threshold of Jaccard prediction: 0.028
Threshold of Baseline prediction: 1.25 (80th percentile of popularity)

Mix2 (AND):
Accurarcy: 0.88385
TPR: 0.7677
TNR: 1.0
Threshold of Jaccard prediction: 0.002
Threshold of Baseline prediction: 1.25 (80th percentile of popularity)

In [85]:
```
mix1Result = []
mix2Result = []

for thresJac in tqdm(np.arange(0, 0.03, 0.002)):
    jacPredBookDataYValid = getJaccardPred(bookDataValid, thresJac)

    for thresBase in np.arange(1, 3, 0.05):
        basePredBookDataYValid = getBaselinePred(bookDataValid, thresBase)

        mix1PredBookDataYValid = []
        mix2PredBookDataYValid = []

        for jacPred, basePred in zip(jacPredBookDataYValid, basePredBookDataYValid):
            if jacPred == basePred:
                mix1PredBookDataYValid.append(jacPred)
                mix2PredBookDataYValid.append(jacPred)
            elif jacPred > basePred:
                mix1PredBookDataYValid.append(jacPred)
                mix2PredBookDataYValid.append(basePred)
            elif basePred > jacPred:
                mix1PredBookDataYValid.append(basePred)
                mix2PredBookDataYValid.append(jacPred)

        acc, TPR, TNR = getMetrics(mix1PredBookDataYValid, bookDataYValid)
        #print("Validataion Mix1: tJaccard=%f, tBaseline=%f, acc=%f, TPR=%f, TNR=%f" % (thresJac,
thresBase, acc, TPR, TNR) )
        mix1Result.append((acc,TPR,TNR,thresJac,thresBase))

        acc, TPR, TNR = getMetrics(mix2PredBookDataYValid, bookDataYValid)
        #print("Validataion Mix2: tJaccard=%f, tBaseline=%f, acc=%f, TPR=%f, TNR=%f" % (thresJac,
thresBase, acc, TPR, TNR) )
        mix2Result.append((acc,TPR,TNR,thresJac,thresBase))
```

```
100%|██████████| 15/15 [04:10<00:00, 16.67s/it]
```

In [86]:
```
mix1Result.sort(reverse=True)
mix2Result.sort(reverse=True)
```

In [87]:
```
mix1Result[:30]
```

Out[87]:
```
[(0.86185, 0.8591, 0.8646, 0.028, 1.2500000000000002),
 (0.85555, 0.8663, 0.8448, 0.026000000000000002, 1.2500000000000002),
 (0.85005, 0.8355, 0.8646, 0.028, 1.3000000000000003),
 (0.84465, 0.8445, 0.8448, 0.026000000000000002, 1.3000000000000003),
 (0.8442, 0.8729, 0.8155, 0.024, 1.2500000000000002),
 (0.83865, 0.8127, 0.8646, 0.028, 1.3500000000000003),
 (0.8346, 0.8244, 0.8448, 0.026000000000000002, 1.3500000000000003),
 (0.8338, 0.8521, 0.8155, 0.024, 1.3000000000000003),
 (0.82975, 0.7949, 0.8646, 0.028, 1.4000000000000004),
 (0.82665, 0.88, 0.7733, 0.022, 1.2500000000000002),
 (0.82635, 0.8079, 0.8448, 0.026000000000000002, 1.4000000000000004),
 (0.82445, 0.8334, 0.8155, 0.024, 1.3500000000000003),
 (0.821, 0.7774, 0.8646, 0.028, 1.4500000000000004),
 (0.8198, 0.8865, 0.7531, 0.02, 1.2500000000000002),
 (0.8184, 0.792, 0.8448, 0.026000000000000002, 1.4500000000000004),
 (0.8174, 0.8615, 0.7733, 0.022, 1.3000000000000003),
 (0.81685, 0.8182, 0.8155, 0.024, 1.4000000000000004),
 (0.8117, 0.7588, 0.8646, 0.028, 1.5000000000000004),
 (0.81155, 0.87, 0.7531, 0.02, 1.3000000000000003),
 (0.8098, 0.7748, 0.8448, 0.026000000000000002, 1.5000000000000004),
 (0.8095, 0.8035, 0.8155, 0.024, 1.4500000000000004),
 (0.8092, 0.8451, 0.7733, 0.022, 1.3500000000000003),
 (0.80425, 0.8554, 0.7531, 0.02, 1.3500000000000003),
 (0.8038, 0.895, 0.7126, 0.018000000000000002, 1.2500000000000002),
 (0.80295, 0.7413, 0.8646, 0.028, 1.5500000000000005),
 (0.80245, 0.8316, 0.7733, 0.022, 1.4000000000000004),
 (0.8023, 0.7598, 0.8448, 0.026000000000000002, 1.5500000000000005),
 (0.8015, 0.7875, 0.8155, 0.024, 1.5000000000000004),
 (0.79815, 0.8432, 0.7531, 0.02, 1.4000000000000004),
 (0.79625, 0.8799, 0.7126, 0.018000000000000002, 1.3000000000000003)]
```

In [88]:
```
mix2Result[:30]
```

Out[88]:
```
[(0.88385, 0.7677, 1.0, 0.002, 1.2500000000000002),
 (0.88385, 0.7677, 1.0, 0.0, 1.2500000000000002),
 (0.8823, 0.7646, 1.0, 0.004, 1.2500000000000002),
 (0.877, 0.754, 1.0, 0.006, 1.2500000000000002),
 (0.86925, 0.7385, 1.0, 0.002, 1.3000000000000003),
 (0.86925, 0.7385, 1.0, 0.0, 1.3000000000000003),
 (0.86785, 0.7357, 1.0, 0.004, 1.3000000000000003),
 (0.86605, 0.7321, 1.0, 0.008, 1.2500000000000002),
 (0.86295, 0.7259, 1.0, 0.006, 1.3000000000000003),
 (0.8558, 0.7116, 1.0, 0.002, 1.3500000000000003),
 (0.8558, 0.7116, 1.0, 0.0, 1.3500000000000003),
 (0.85465, 0.7093, 1.0, 0.004, 1.3500000000000003),
 (0.85255, 0.7051, 1.0, 0.008, 1.3000000000000003),
 (0.8501, 0.7002, 1.0, 0.006, 1.3500000000000003),
 (0.84635, 0.6927, 1.0, 0.01, 1.2500000000000002),
 (0.84385, 0.6877, 1.0, 0.002, 1.4000000000000004),
 (0.84385, 0.6877, 1.0, 0.0, 1.4000000000000004),
 (0.8428, 0.6856, 1.0, 0.004, 1.4000000000000004),
 (0.8403, 0.6806, 1.0, 0.008, 1.3500000000000003),
 (0.8387, 0.6774, 1.0, 0.006, 1.4000000000000004),
 (0.83365, 0.6673, 1.0, 0.01, 1.3000000000000003),
 (0.83285, 0.6657, 1.0, 0.002, 1.4500000000000004),
 (0.83285, 0.6657, 1.0, 0.0, 1.4500000000000004),
 (0.83195, 0.6639, 1.0, 0.004, 1.4500000000000004),
 (0.82925, 0.6585, 1.0, 0.008, 1.4000000000000004),
 (0.828, 0.656, 1.0, 0.006, 1.4500000000000004),
 (0.8222, 0.6444, 1.0, 0.01, 1.3500000000000003),
 (0.82185, 0.6437, 1.0, 0.002, 1.5000000000000004),
 (0.82185, 0.6437, 1.0, 0.0, 1.5000000000000004),
 (0.82105, 0.6421, 1.0, 0.004, 1.5000000000000004)]
```

In [57]:
```
def writeOutMixture1Pred(thresJac, thresBase):
    predOutMix1File = open("predictions_Read_mix1.txt", 'w')
    bookDataTest = []

    # Read Testing set
    for l in open("pairs_Read.txt", 'r'):
        if l.startswith("userID"):
            #header
            predOutMix1File.write(l)
            continue
        uId, bId = l.strip().split('-')
        bookDataTest.append([uId, bId, -1])

    # Predict by Jaccard
    jacPredBookDataYValid = getJaccardPred(bookDataTest, thresJac)
    basePredBookDataYValid = getBaselinePred(bookDataTest, thresBase)
    mix1PredBookDataYValid = []

    for jacPred, basePred in zip(jacPredBookDataYValid, basePredBookDataYValid):
        if jacPred == basePred:
            mix1PredBookDataYValid.append(jacPred)
        elif jacPred > basePred:
            mix1PredBookDataYValid.append(jacPred)
        elif basePred > jacPred:
            mix1PredBookDataYValid.append(basePred)

    # Write out prediction result for mix1 model
    for data, y in zip(bookDataTest, mix1PredBookDataYValid):
        predOutMix1File.write(data[0] + '-' + data[1] + "," + str(y) + "\n")

    predOutMix1File.close()
```

In [58]:
```
def writeOutMixture2Pred(thresJac, thresBase):
    predOutMix2File = open("predictions_Read_mix2.txt", 'w')
    bookDataTest = []

    # Read Testing set
    for l in open("pairs_Read.txt", 'r'):
        if l.startswith("userID"):
            #header
            predOutMix2File.write(l)
            continue
        uId, bId = l.strip().split('-')
        bookDataTest.append([uId, bId, -1])

    # Predict by Jaccard
    jacPredBookDataYValid = getJaccardPred(bookDataTest, thresJac)
    basePredBookDataYValid = getBaselinePred(bookDataTest, thresBase)
    mix2PredBookDataYValid = []

    for jacPred, basePred in zip(jacPredBookDataYValid, basePredBookDataYValid):
        if jacPred == basePred:
            mix2PredBookDataYValid.append(jacPred)
        elif jacPred > basePred:
            mix2PredBookDataYValid.append(basePred)
        elif basePred > jacPred:
            mix2PredBookDataYValid.append(jacPred)

    # Write out prediction result for mix2 model
    for data, y in zip(bookDataTest, mix2PredBookDataYValid):
        predOutMix2File.write(data[0] + '-' + data[1] + "," + str(y) + "\n")

    predOutMix2File.close()
```

In [59]:
```
writeOutMixture2Pred(0.002, 1.05)
```

In [62]:
```
writeOutMixture1Pred(0.028, 1.05)
```

5. To run our model on the test set, we'll have to use the files 'pairs Read.txt' to find the reviewerID/itemID pairs about which we have to make predictions.

Using that data, run the above model and upload your solution to Kaggle. Tell us your Kaggle user name (1 mark). If you've already uploaded a better solution to Kaggle, that's fine too!

Ans:
Display Name: JamesTcl
User Name: k2973363
Email Address: til002@eng.ucsd.edu

(CSE 258 only) Tasks (Rating prediction)

Let's start by building our training/validation sets much as we did for the first task. This time building a validation set is more straightforward: you can simply use part of the data for validation, and do not need to randomly sample non-read users/books.

In [352]:
```
### Rating baseline: compute averages for each user, or return the global average if we've never
 seen the user before

allRatings = []
userRatings = defaultdict(list)

for user,book,r in readCSV("train_Interactions.csv.gz"):
    r = int(r)
    allRatings.append(r)
    userRatings[user].append(r)

globalAverage = sum(allRatings) / len(allRatings)
userAverage = {}
for u in userRatings:
    userAverage[u] = sum(userRatings[u]) / len(userRatings[u])

predictions = open("predictions_Rating.txt", 'w')
for l in open("pairs_Rating.txt"):
    if l.startswith("userID"):
        #header
        predictions.write(l)
        continue
    u,b = l.strip().split('-')

    if u in userAverage:
        predictions.write(u + '-' + b + ',' + str(userAverage[u]) + '\n')
    else:
        predictions.write(u + '-' + b + ',' + str(globalAverage) + '\n')

predictions.close()
```

9. Fit a predictor of the form by fitting the mean and the two bias terms as described in the lecture notes.

Use a regularization parameter of λ = 1. Report the MSE on the validation set.

Ans:
Separate 70% of data as training set, and the rest of 30% of data as validation set.
MSE on the validation set: 1.4613473974773903

In [29]:
```
# Shuffle First?
numTraining = int(len(bookData) * 0.70)
bookDataTrain = bookData[:numTraining]
bookDataValid = bookData[numTraining:]
```

In [30]:
```
reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)

for userId,bookId,r in bookDataTrain:
    reviewsPerUser[userId].append(r)
    reviewsPerItem[bookId].append(r)
```

In [31]:
```
N = len(bookDataTrain)
nUsers = len(reviewsPerUser)
nItems = len(reviewsPerItem)
users = list(reviewsPerUser.keys())
items = list(reviewsPerItem.keys())
```

In [32]:
```
ratingMean = sum([d[2] for d in bookDataTrain]) / len(bookDataTrain)
ratingMean
```

Out[32]:
```
3.896857142857143
```

In [33]:
```
alpha = ratingMean
alpha
```

Out[33]:
```
3.896857142857143
```

In [34]:
```
userBiases = defaultdict(float)
itemBiases = defaultdict(float)
```

In [35]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

In [36]:
```
def prediction(user, item):
    userBias = 0
    itemBias = 0
    if user in userBiases:
        userBias = userBiases[user]
    if item in itemBiases:
        itemBias = itemBiases[item]

    return alpha + userBias + itemBias
```

In [37]:
```
def unpack(theta):
    global alpha
    global userBiases
    global itemBiases
    alpha = theta[0]
    userBiases = dict(zip(users, theta[1:nUsers+1]))
    itemBiases = dict(zip(items, theta[1+nUsers:]))
```

In [38]:
```
def cost(theta, labels, lamb):
    unpack(theta)
    predictions = [prediction(uId, bId) for uId, bId, r in bookDataTrain]
    cost = MSE(predictions, labels)
    #print("MSE = " + str(cost))
    for u in userBiases:
        cost += lamb*userBiases[u]**2
    for i in itemBiases:
        cost += lamb*itemBiases[i]**2
    return cost
```

In [39]:
```
def derivative(theta, labels, lamb):
    unpack(theta)
    N = len(bookDataTrain)
    dalpha = 0
    dUserBiases = defaultdict(float)
    dItemBiases = defaultdict(float)
    for uId, bId, r in bookDataTrain:
        pred = prediction(uId, bId)
        diff = pred - r
        dalpha += 2/N*diff
        dUserBiases[uId] += 2/N*diff
        dItemBiases[bId] += 2/N*diff
    for u in userBiases:
        dUserBiases[uId] += 2*lamb*userBiases[uId]
    for i in itemBiases:
        dItemBiases[bId] += 2*lamb*itemBiases[bId]
    dtheta = [dalpha] + [dUserBiases[uId] for uId in users] + [dItemBiases[bId] for bId in items]
    return np.array(dtheta)
```

In [40]:
```
alwaysPredictMean = [ratingMean for d in bookDataTrain]
```

In [41]:
```
labels = [r for uId,bId,r in bookDataTrain]
```

In [42]:
```
MSE(alwaysPredictMean, labels)
```

Out[42]:
```
1.47996155102053
```

In [43]:
```
scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + [0.0]*(nUsers+nItems), derivative, args = (labels, 1
))
```

Out[43]:
```
(array([ 3.89685714e+00,  1.37862830e-05,  4.43085293e-05, ...,
        -1.35615843e-05,  1.57738446e-05, -3.20604106e-05]),
 1.4798772569227214,
 {'grad': array([ 5.44752911e-05, -2.75429760e-05, -8.84988302e-05, ...,
          2.70969832e-05, -3.15184388e-05,  6.40528352e-05]),
  'task': b'CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH',
  'funcalls': 35,
  'nit': 3,
  'warnflag': 0})
```

In [44]:
```
predictions = [prediction(uId, bId) for uId, bId, r in bookDataValid]
```

In [45]:
```
labelsValid = [r for uId,bId,r in bookDataValid]
```

In [46]:
```
# lambda = 0.001, MSE=1.3215213938145665
MSE(predictions, labelsValid)
```

Out[46]:
```
1.4613473974773903
```

10. Report the user and book IDs that have the largest and smallest values of β

Ans:
user ID with largest values of β: u92864068
user ID with smallest values of β: u11591742
book ID with largest values of β: b76915592
book ID with smallest values of β: b80820222

In [53]:
```
userBiasesList = [(bias, uId) for uId, bias in userBiases.items()]
userBiasesList.sort(reverse=True)
```

In [54]:
```
userBiasesList[0]
```

Out[54]:
```
(0.0004446173289278906, 'u92864068')
```

In [55]:
```
userBiasesList[-1]
```

Out[55]:
```
(-0.0014061416249505658, 'u11591742')
```

In [56]:
```
itemBiasesList = [(bias, bId) for bId, bias in itemBiases.items()]
itemBiasesList.sort(reverse=True)
```

In [57]:
```
itemBiasesList[0]
```

Out[57]:
```
(0.0007608030421973806, 'b76915592')
```

In [58]:
```
itemBiasesList[-1]
```

Out[58]:
```
(-0.0002643354810743335, 'b80820222')
```

11. Find a better value of λ using your validation set. Report the value you chose, its MSE, and upload your solution to Kaggle by running it on the test data

Ans:
Choose the λ with lowest MSE on the validation set, where λ = 0.0000100000
MSE on the validation set: 1.123269

In [60]:
```
def writeOutTestSetPred(lambdaVal):
    fileName = "predictions_Rating_" + str(lambdaVal) + ".txt"
    predictions = open(fileName, 'w')
    for l in open("pairs_Rating.txt"):
        if l.startswith("userID"):
            #header
            predictions.write(l)
            continue
        uId,bId = l.strip().split('-')
        predictions.write(uId + '-' + bId + ',' + str(prediction(uId, bId)) + '\n')

    predictions.close()
```

In [61]:
```
lambdaExpList = range(-7,2)
for exp in lambdaExpList:
    # Train the model
    l = pow(10,exp)
    scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + [0.0]*(nUsers+nItems), derivative, args = (label
s, l))

    # Predict from the model
    predictions = [prediction(uId, bId) for uId, bId, r in bookDataValid]
    labelsValid = [r for uId,bId,r in bookDataValid]
    mse = MSE(predictions, labelsValid)
    print("Lambda: %.10f, MSE of validation set = %f" % (l, mse))
    writeOutTestSetPred(l)
```

```
Lambda: 0.0000001000, MSE of validation set = 1.146492
Lambda: 0.0000010000, MSE of validation set = 1.144550
Lambda: 0.0000100000, MSE of validation set = 1.123269
Lambda: 0.0001000000, MSE of validation set = 1.159703
Lambda: 0.0010000000, MSE of validation set = 1.367648
Lambda: 0.0100000000, MSE of validation set = 1.446096
Lambda: 0.1000000000, MSE of validation set = 1.460314
Lambda: 1.0000000000, MSE of validation set = 1.461343
Lambda: 10.0000000000, MSE of validation set = 1.461462
```

In [ ]: