Bosi Cheng
A532712697

Purchase prediction

This task is to Predict given a (user, item) pair from ‘pairs_Purchase.txt’ whether the user purchased the item (really, whether it was one of the products they reviewed).

In homework3, we predicted the similar stuffs by if the item is "popular". In the assignment, I improved it with Jaccard algorithm. By calculating the Jaccard distance we can determine the relevance between a user and an item (in what extent they're related).

The result is "yes" or "no" so I have to determine a boundary value. Firstly I calculated the mean of all the distances and then I tried several values around it. The 0.183 gave me the largest correction ratio.

```
import gzip

from collections import defaultdict

import random




def readGz(f):

    for l in gzip.open(f):

        yield eval(l)



### Would-purchase baseline: just rank which businesses are popular and which are not, and return '1' if a

business is among the top-ranked

def jaccard_similarity(list1, list2):

        s1 = set(list1)
```

```
    s2 = set(list2)
    return len(s1.intersection(s2)) / len(s1.union(s2))


user_catagory = defaultdict(list)
item_catagory = defaultdict(list)
business=defaultdict(list)


for l in readGz("train.json.gz"):
    user, item, catagories = l['reviewerID'], l['itemID'], l['categories']
    business[user].append(item)
    for category in catagories:
        for label in category:
            if label not in user_catagory[user]:
                user_catagory[user].append(label)
            if label not in item_catagory[item]:
                item_catagory[item].append(label)



predictions = open("predictions_Purchase.txt", 'w')
accur=0
```

```
for l in open("pairs_Purchase.txt"):
    if l.startswith("reviewerID"):
        #header
        predictions.write(l)
        continue
    u,i = l.strip().split('-')
    dis=jaccard_similarity(user_catagory[u],item_catagory[i])
    # if count_pos>=count_neg:
    predictions.write(u + '-' + i + ","+ str(1 if dis>0.183 else 0) +"\n")


predictions.close()
```

Rating prediction

Applying latent factor model to implement this project.

Firstly, retrieve the data and initialize the variations. And then run lfm_main function with iteration times and lambda. I tried several iteration times and found the rather good lambda pick. Then predict the rating with predict function. If b_u has the user and b_i has the item it calculates the result or it simply returns the mean value alpha.

```
import gzip
from collections import defaultdict
import numpy
from random import sample, randint
```

```
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt



def readGz(f):
    for l in gzip.open(f):
        yield eval(l)



def predict(user, item, b_u, b_i, a):
    try:
        u = userIds[user]
        i = itemIds[item]
        return a + b_u[u] + b_i[i]
    except KeyError:
        return a



def lfm_main(x_train, b_u, b_i, a, iterate_times, labmda):
    for _ in range(iterate_times):
        for j in range(len(x_train)):
            u = x_train[j][0]
            i = x_train[j][1]
```

```
        userItems = userItem[u]

        b_u[u] = (- a * len(userItems) - b_i[userItems].sum() + R[u,userItems].sum()) / (labmda +
len(userItems))

        itemUsers = itemUser[i]

        b_i[i] = (- a * len(itemUsers) - b_u[itemUsers].sum() + R[itemUsers, i].sum()) / (labmda +
len(itemUsers))

    return a, b_i, b_u



data = list(readGz('train.json.gz'))



users = list(set([d['reviewerID'] for d in data]))

items = list(set([d['itemID'] for d in data]))

num_users = len(users)

num_items = len(items)

userIds = dict(zip(users, range(num_users)))

itemIds = dict(zip(items, range(num_items)))



R = numpy.zeros((num_users, num_items))

nums = numpy.zeros((num_users, num_items))

userItem = defaultdict(list)
```

```
itemUser = defaultdict(list)

for d in data:

    u = userIds[d['reviewerID']]

    i = itemIds[d['itemID']]

    R[u][i] = d['rating']

    nums[u][1] += 1

    userItem[u].append(i)

    itemUser[i].append(u)



nums[nums == 0] = 1

R /= nums



x_train = numpy.array([[userIds[d['reviewerID']], itemIds[d['itemID']]] for d in data])

y_train = numpy.array([d['rating'] for d in data])



b_u = numpy.random.random((num_users,))

b_i = numpy.random.random((num_items))

a = y_train.mean()



a, b_i, b_u= lfm_main(x_train, b_u, b_i, a, 20, 6.4)
```

```
predictions = open("predictions_Rating.txt", 'w')

for l in open("pairs_Rating.txt"):

    if l.startswith("reviewerID"):

        predictions.write(l)

        continue

    u,i = l.strip().split('-')

    p = predict(u, i , b_u, b_i, a)

    predictions.write(u + '-' + i + ',' + str(p) + '\n')

predictions.close()
```