Kaggle username: thundering
Task1: visit prediction
Method:
I use KNN and jaccard similarity in my solution.
There are four kinds of data in the test set: (1) both user and business appear in training set (2) only user appears in training set (3) only business appear in the training set (4) user and business don't appear in training set.
For the situation (1) both user and business appear in training set,suppose we want to predict whether (user1 ,business1) is 1 or 0. I combined two method together and use OR to decide the final prediction. In first method, from the training data, I get the set of business that user1 have visited , we call it user1_business. Then I compute the Jaccard similarity between business1 and every user in set user1_business. If the maximum value among these similarity is over 0.8, then by using this method, I predict 1,otherwise predict 0 .
As for calculating the Jaccard similarity between users or between business in this problem, I use categories information to calculate it. Suppose category1 represents the categories of business that user_example1 has visited while category2 represents the categories of business that user_example2 has visited. Jaccard similarity between (user_example1, user_example2) is the intersection of category1 and category2 divided by union of category1 and category2. One disadvantage of this is if user_example1 and user_example2 visited the same kind of restaurant, but user_example1 rated 5 star while user_example2 rated 1 star. In this situation, although they both visited the same kind of business, but to say user_example1 and user_example2 are similar is not proper because their rating to the same business are not consistent. So when I get the categories of business that the user visited, I only choose the categories of business which the user rated equal or higher than the average rating of all business of this user.
In second method, suppose we want to predict whether (user1, business 1) is 1 or 0. First, I calculate the Jaccard similarity between each user who has visited business1 and user1. Then, I set a threshold to choose all the users that are most similar to user1, which is to find the nearest neighborhood of user1. Use similar_business to describe the set of all the business visited by the users who are most similar to user1. Then if the business1 is in the set similar_business, then I predict 1, otherwise, predict 0.
If any of these method predict 1, then the final prediction is 1, otherwise 0.

For situation (2) only users appears in the training set, suppose we want to predict whether (user1, business1) is 1 or 0. For each user in training set, I already got all the categories the user has visited. If this user has visited over 40 different kinds of business, then I suppose this user has visited business1, although I have no information about business1.

For situation (3) only business appears in the training set, suppose we want to predict whether (user1, business1) is 1 or 0. Since user1 doesn't appear in the training set, if the business show up in the popular_business set which is defined as the business that 58% of the whole users have visited. If business1 is in popular_business set, we predict 1 otherwise, predict 0.

For situation (4) users and business don't appear in the training set, we predict 0 directly.

Result: The result is 0.73675 on Kaggle with parameter1= 0.25 (above this value, I regard two users are similar, parameter2 = 0.8 (above this value, I regard two business are similar)

Task2: rating prediction

Method:

I use latent-factor models $f(u,b) = \alpha + \beta u + \beta b + \gamma u * \gamma b$ with k = 1 to solve this problem. The goal is to calculate ：

$\arg\min \alpha\beta\gamma \sum_{u,b} (\alpha + \beta u + \beta b + \gamma u * \gamma b - Ru,b)^2 + \lambda[\sum_u \beta u^2 + \sum_b \beta b^2 + \sum_b \left|\left|\gamma b\right|\right|_2^2 + \sum_u \left|\left|\gamma u\right|\right|_2^2]$

initialize $\alpha, \beta u, \beta b, \gamma u, \gamma b$ to random numbers

step1: fix γb, update $\gamma u = \frac{\sum_{b \in Bu}(Ru,b-\alpha-\beta u-\beta b)\gamma b}{\sum_{b \in Bu} \gamma b^2+\lambda}$

step2: fix γu, update $\gamma b = \frac{\sum_{u \in Ub}(Ru,b-\alpha-\beta u-\beta b)\gamma u}{\sum_{u \in Ub} \gamma u^2+\lambda}$

step3: fix $\gamma u, \gamma b$, update $\alpha = \frac{\sum_{u,b}(Ru,b-\beta u-\beta b-\gamma u\gamma b)}{Ntrain}$

step4: fix $\gamma u, \gamma b, \alpha$, update $\beta u = \frac{\sum_{b \in Bu}(Ru,b-\alpha-\gamma b\gamma u-\beta b)}{\sum_{b \in Bu}+\lambda}$

step5: fix $\gamma u, \gamma b, \alpha, \beta u$, update $\beta b = \frac{\sum_{u \in Ub}(Ru,b-\alpha-\beta u-\gamma b\gamma u)}{\sum_{u \in Ub}+\lambda}$

Do until the MSE converges and change different λ to observe MSE.

Pick the best λ

After we get the value of $\alpha, \beta u, \beta b, \gamma u, \gamma b$, we use $f(u,b) = \alpha + \beta u + \beta b + \gamma u * \gamma b$ to calculate the rating if both u and b appear in training set, $f(u,b) = \alpha + \beta u$ if only u appears in training set, $f(u,b) = \alpha + \beta b$ in only b appears in training set and $f(u,b) = \alpha$ if neither u nor b appear in the training set.

Result:

With $\lambda = 4$, I get the best result. The result is 0.74715 on Kaggle.

Appendix:

Task 1

```
import gzip
from collections import defaultdict
import random
def readGz(f):
    for l in gzip.open(f):
        yield eval(l)
```

```
print('reading data...')
data = list(readGz('train.json.gz'))
print('done')
```

```
reading data...
done
```

```
known_ub_pair = [(d['userID'],d['businessID']) for d in data]  #get known user,business pair from original 200,000 trai
known_user = [d['userID'] for d in data]
known_business = [d['businessID'] for d in data]
iden_user = list(set(known_user))
iden_business = list(set(known_business))
```

```
userRatings = defaultdict(list)
for d in data:
    user,business = d['userID'],d['businessID']
    userRatings[user].append(d['rating'])
userAverage = {}
for u in userRatings:
    userAverage[u] = sum(userRatings[u])/len(userRatings[u])
```

```
#create business-category pairs(duplicate category exist)
busiCat_set = defaultdict(set)
for b in iden_business:
    cat = [d['categories'] for d in data if d['businessID'] == b]
    combine_cat = [j for i in cat for j in i]
    for i in combine_cat:
        busiCat_set[b].add(i)
```

```
#create business_user and user_business pairs
user_business = defaultdict(list)
business_user = defaultdict(list)
for (u,b) in known_ub_pair:
    user_business[u].append(b)
    business_user[b].append(u)
for u in user_business:
    user_business[u] = list(set(user_business[u]))
for b in business_user:
    business_user[b] = list(set(business_user[b]))
```

```
#get the list of popular business
businessCount = defaultdict(int)
totalPurchase = 0
for (u,b) in known_ub_pair:
    businessCount[b] += 1
    totalPurchase += 1
mostPopular = [(businessCount[x],x) for x in businessCount]
mostPopular.sort()
mostPopular.reverse()
popular_business=set()  #set of 58% popular business
count = 0
for ic,i in mostPopular:
    count +=ic
    popular_business.add(i)
    if count>totalPurchase*5.8/10:break
```

```
#get the business category of each user
userCat_set = defaultdict(set)
for r in data:
    if 'categories' in r:
        for c in r['categories']:
            if r['rating']>=userAverage[r['userID']]:
                userCat_set[r['userID']].add(c)
```

```
userCat = defaultdict(set)
for r in data:
    if 'categories' in r:
        for c in r['categories']:
            userCat[r['userID']].add(c)
```

```
predictions = open('task1.txt','w')
for l in open('pairs_Visit.txt'):
    if l.startswith('userID'):
        predictions.write(l)
        continue
    u,b = l.strip().split('-')
    if u in iden_user:
        if b in iden_business:
            similar_business = []
            for elseuser in business_user[b]:
                sim = len(userCat_set[u].intersection(userCat_set[elseuser]))/len(userCat_set[u].union(userCat_set[elseu
                if sim>=0.25:  #similarity beween users
                    for bb in user_business[elseuser]:
                        similar_business.append(bb)
            if b in similar_business:
                pre_predict_1 = 1
            else:
                pre_predict_1 = 0
            ################################
            sim_value = []
            for bbb in user_business[u]:
                sim = len(busiCat_set[b].intersection(busiCat_set[bbb]))/len(busiCat_set[b].union(busiCat_set[b]))
                sim_value.append(sim)
            sim_value.sort()
            sim_value.reverse()
            if len(sim_value)>1 and sim_value[1]>0.8:  #threshold for business in test and that of all the business use
                pre_predict_2 = 1
            else:
                pre_predict_2 = 0
            if pre_predict_1==1 and pre_predict_2 ==1:
                predict = 1
            else:
                predict = 0
```

```
        else:
            if len(userCat[u])>40:                  #
                predict = 1
            else:
                predict = 0
    else:
        if b in iden_business and b in popular_business:
            predict = 1
        else:
            predict = 0#random.randint(0,1)
    predictions.write(u + '-' + b + ',' + str(predict) + '\n')
predictions.close()
```

Task 2

```
import gzip
from collections import defaultdict
def readGz(f):
    for l in gzip.open(f):
        yield eval(l)
print('reading data...')
training = list(readGz('train.json.gz'))
print('done')
```

```
user_iden = [d['userID'] for d in training]
user_iden = list(set(user_iden))
business_iden = [d['businessID'] for d in training]
business_iden = list(set(business_iden))
training_data = [(d['userID'],d['businessID'],d['rating']) for d in training]
u_b_rating = defaultdict(float)
user_business = defaultdict(list)
business_user = defaultdict(list)
for (u,b,r) in training_data:
    u_b_rating[(u,b)] = r
    user_business[u].append(b)
    business_user[b].append(u)
for u in user_business:
    user_business[u] = list(set(user_business[u]))
for b in business_user:
    business_user[b] = list(set(business_user[b]))
beta_user = defaultdict(float)
beta_business = defaultdict(float)
gamma_user = defaultdict(float)
gamma_business = defaultdict(float)
```

```
allRating = []
userRating = defaultdict(list)
businessRating = defaultdict(list)
for (u,b,r) in training_data:
    allRating.append(r)
    userRating[u].append(r)
    businessRating[b].append(r)
globalAverage = sum(allRating)/len(allRating)
userAverage = {}
for u in userRating:
    userAverage[u] = sum(userRating[u])/len(userRating[u])
businessAverage = {}
for b in businessRating:
    businessAverage[b] = sum(businessRating[b])/len(businessRating[b])
```

```
#initialize
from random import randint
lam = 4
alpha = randint(40,45)/10
for u in user_iden:
    beta_user[u] = randint(-20,20)/100
    gamma_user[u] = randint(-20,20)/100
for b in business_iden:
    beta_business[b] = randint(-20,20)/100
    gamma_business[b] = randint(-20,20)/100
```

```
#update
#(1) fix gamma_business, update gamma_user
for iteration in range(2):
    for u in user_iden:
        update_gamma_user = 0
        sum_gamma_b = 0
        for b in user_business[u]:
            update_gamma_user += ( u_b_rating[(u,b)]-alpha-beta_user[u]-beta_business[b] )*gamma_business[b]
            sum_gamma_b += gamma_business[b]**2
        gamma_user[u] = update_gamma_user/( sum_gamma_b + lam)
    #(2) fix gamma_user , update gamma_business
    for b in business_iden:
        update_gamma_business = 0
        sum_gamma_u = 0
        for u in business_user[b]:
            update_gamma_business += ( u_b_rating[(u,b)]-alpha-beta_user[u]-beta_business[b] )*gamma_user[u]
            sum_gamma_u += gamma_user[u]**2
        gamma_business[b] = update_gamma_business/( sum_gamma_u + lam)
    #(3) fix gamma_user and gamma_business, update alpha
    update_alpha = 0
    for (u,b,r) in training_data:
        update_alpha += r - beta_user[u]-beta_business[b]-gamma_user[u]*gamma_business[b]
    alpha = update_alpha / len(training_data)
    #(4) fix gamma_business, gamma_user, alpha , update beta_user
    for u in user_iden:
        update_beta_user = 0
        for b in user_business[u]:
            update_beta_user += u_b_rating[(u,b)] - alpha-beta_business[b]-gamma_user[u]*gamma_business[b]
        beta_user[u] = update_beta_user/(len(user_business[u]) + lam)
    for b in business_iden:
        update_beta_business = 0
        for u in business_user[b]:
            update_beta_business += u_b_rating[(u,b)] - alpha-beta_user[u]-gamma_user[u]*gamma_business[b]
        beta_business[b] = update_beta_business/(len(business_user[b])+lam)
```

```
predictions = open('task2_rating.txt','w')
for l in open('pairs_Rating.txt'):
    if l.startswith('userID'):
        predictions.write(l)
        continue
    u,b = l.strip().split('-')
    if u in user_iden:
        if b in business_iden:
            rating = alpha+beta_user[u]+beta_business[b]+gamma_user[u]*gamma_business[b]
            predictions.write(u+'-'+b+','+str(rating)+'\n')
        else:
            rating = alpha+beta_user[u]
            predictions.write(u+'-'+b+','+str(rating)+'\n')
    else:
        if b in business_iden:
            rating = alpha+beta_business[b]
            predictions.write(u+'-'+b+','+str(rating)+'\n')
        else:
            rating = alpha
            predictions.write(u+'-'+b+','+str(rating)+'\n')
predictions.close()
```