He, Jiaqi
University of California, San Diego

November 10, 2017

HOMEWORK 3

1 Task (Visit Prediction)

Problem 1. Although we have built a validation set, it only consists of positive samples. For this task we also need examples of user/business pairs that weren't visited. Build such a set by randomly sampling users and businesses until you have 100,000 non-visited user/business pairs. This random sample combined with your 100,000 validation reviews now corresponds to the complete validation set for the visit prediction task. Evaluate the performance (accuracy) of the baseline model on the validation set you have built (1 mark).

Answer: First step is to divide 200,000 data into two parts for training and validation.
Secondly, in order to generate 100,000 non-visited user/business pairs, we need a user set and a business set containing all recorded users and businesses. Then we randomly pick one user and one business and check whether this pair has ever existed in 200,000 data. If not, we then successfully generate one pair. Keep doing this until we have 100,000 non-visited pairs.
Lastly, evaluate the performance (accuracy) of the baseline model on the validation set.
The accuracy is 0.648705.
Below is the code for this problem.

```
import gzip
from collections import defaultdict
import numpy as np
import random
import math

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

data_set = []
u2b = {}
users = set([])
businesses = set([])
for l in readGz("train.json.gz"):
    user,business = l['userID'],l['businessID']
    users.add(user)
    businesses.add(business)
    if user not in u2b:
        u2b[user] = set([business])
    else:
        u2b[user].add(business);
    data_set.append([user, business, 1])

train_set = data_set[:int(len(data_set)/2)]
```

```
validate_set = data_set[int(len(data_set)/2):]

# randomly create 100000 negative samples
users = list(users)
businesses = list(businesses)
neg_samples = []
while len(neg_samples) < 100000:
    selected_user = users[int(np.random.rand()*len(users))]
    selected_business = businesses[int(np.random.rand()*len(businesses))]
    if selected_business not in u2b[selected_user]:
        neg_samples.append([selected_user, selected_business, 0])

validate_set = validate_set + neg_samples

# use baseline model to train
businessCount = defaultdict(int)
totalPurchases = 0

i = 1;
for l in readGz("train.json.gz"):
    if i <= 100000:
        user,business = l['userID'],l['businessID']
        businessCount[business] += 1
        totalPurchases += 1
    else:
        break

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

return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > totalPurchases/2: break

prediction = []
correct_number = 0
for elem in validate_set:
    if elem[1] in return1:
        prediction.append(1)
        if elem[2] == 1:
            correct_number += 1
    else:
        prediction.append(0)
        if elem[2] == 0:
            correct_number += 1

accuracy = correct_number/len(validate_set)

print(accuracy)
```

Problem 2. The existing 'visit prediction' baseline just returns True if the business in question is 'popular,' using a threshold of the 50th percentile of popularity (totalVisits/2). Assuming that the 'non-visited' test examples are a random sample of user-visit pairs, is this particular threshold value the best? If not, see if you can fi

nd a better one (and report its performance), or if so, explain why it is the best (1 mark).

Answer: By trying different percentiles, we can get accuracy data in Table 1.

Table 1: Accuracy on the Validation Set with Different Percentiles

Percentile	1/4	1/3	1/2	2/3	3/4
Accuracy	0.596	0.618	0.648	0.652	0.639

By checking the table, we can know that setting percentile to be approximately 2/3 would be better. This ratio is vary close to **golden ratio**: 0.618, and if we try this percentile, we can get a better result: 0.653435. Golden ratio wins.

Problem 3. Users may tend to repeatedly visit business of the same type. Build a baseline that returns 'True' if a user has visited a business of the same category before (at least one category in common), or zero otherwise (1 mark).

Answer: In this problem, we need to keep track of each user's visiting history to help determine whether he/she has ever visited a business of same category.
First, for each user-business pair in the training set, we add the categories of that business into two dictionaries: u2t(user to type) and b2t(business to type). By doing so, we are storing the categories that each user would visit and each business possesses.
Second, validate the model. For each user-business pair, we retrive the categories that this user would visit and the categories that this business possess, we check if these two sets have any intersections. If there exists some intersections, then we say this user would visit this business, return 'True'.
Hence, the accuracy of this model on the validation set is: 0.650525.
Below is the code for this problem:

```
import gzip
from collections import defaultdict
import numpy as np
import random
import math

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

data_set = []
u2b = {}
users = set([])
businesses = set([])
u2t = {}
b2t = {}
i = 1
for l in readGz("train.json.gz"):
    user,business,types = l['userID'],l['businessID'],l['categories']
    users.add(user)
    businesses.add(business)
    if user not in u2b:
        u2b[user] = set([business])
    else:
        u2b[user].add(business);
    data_set.append([user, business, 1])
    if i <=100000:
        i += 1
        # update user -> types that they visit
        if user not in u2t:
            u2t[user] = set(types)
        else:
            for t in types:
                u2t[user].add(t)
        # update business -> types that they belong to
        if business not in b2t:
            b2t[business] = set(types)
        else:
            for t in types:
```

```
                b2t[business].add(t)


train_set = data_set[:int(len(data_set)/2)]
validate_set = data_set[int(len(data_set)/2):]

# randomly create 100000 negative samples
users = list(users)
businesses = list(businesses)
neg_samples = []
while len(neg_samples) < 100000:
    selected_user = users[int(np.random.rand()*len(users))]
    selected_business = businesses[int(np.random.rand()*len(businesses))]
    if selected_business not in u2b[selected_user]:
        neg_samples.append([selected_user, selected_business, 0])

validate_set = validate_set + neg_samples

# predict the results
prediction = []
correct_number = 0
for elem in validate_set:
    if elem[0] in u2t and elem[1] in b2t:
        types_user = u2t[elem[0]]
        types_business = b2t[elem[1]]
        size1 = len(types_user)
        size2 = len(types_business)
        types_user = types_user.union(types_business)
        size3 = len(types_user)
        if size1 + size2 != size3:
            prediction.append(1)
            if elem[2] == 1:
                correct_number += 1
        else:
            prediction.append(0)
            if elem[2] == 0:
                correct_number += 1
    else:
        prediction.append(0)
        if elem[2] == 0:
            correct_number += 1

accuracy = correct_number/len(validate_set)

print(accuracy)
```

Problem 4. To run our model on the test set, we'll have to use the files 'pairs Visit.txt' to find the userID/businessID 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!

Answer: My Kaggle user name is: **jih417**, as is shown in Fig. 1.

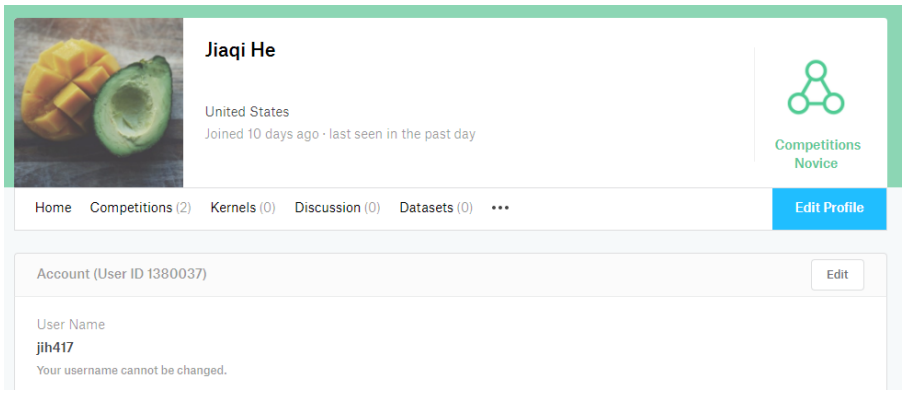

Figure 1: Kaggle profile info

If using the model from problem 3, the score is 0.66490 on Kaggle. We can definitely improve this result by building a collaborative filtering model, which will be solved in assignment 1.

Below is the code for this problem.

```
import gzip
from collections import defaultdict
import numpy as np
import random
import math

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

data_set = []
u2b = {}
users = set([])
businesses = set([])
u2t = {}
b2t = {}
for l in readGz("train.json.gz"):
    user,business,types = l['userID'],l['businessID'],l['categories']
    users.add(user)
    businesses.add(business)
    if user not in u2b:
        u2b[user] = set([business])
    else:
```

```
        u2b[user].add(business);
    # update user -> types that they visit
    if user not in u2t:
        u2t[user] = set(types)
    else:
        for t in types:
            u2t[user].add(t)
    # update business -> types that they belong to
    if business not in b2t:
        b2t[business] = set(types)
    else:
        for t in types:
            b2t[business].add(t)
    data_set.append([user, business, 1])

print (len(users))
print (len(businesses))
# predict the results
predictions = open("predictions_Visit.txt", 'w')
for l in open("pairs_Visit.txt"):
    if l.startswith("userID"):
        #header
        predictions.write(l)
        continue
    u,i = l.strip().split('-')
    if u in u2t and i in b2t:
        types_user = u2t[u]
        types_business = b2t[i]
        size1 = len(types_user)
        size2 = len(types_business)
        types_user = types_user.union(types_business)
        size3 = len(types_user)
        if size1 + size2 != size3:
            predictions.write(u + '-' + i + ",1\n")
        else:
            predictions.write(u + '-' + i + ",0\n")
    else:
        predictions.write(u + '-' + i + ",0\n")

predictions.close()
```

2 Task (Rating Prediction)

Problem 5. What is the performance of a trivial predictor

$$rating(user, item) = \alpha \tag{1}$$

on the validation set, and what is the value of α (1 mark)?

Answer: This model predicts each user-item pair to be the average rating.
First, we acquire training data and validation data. On the training set, we add up all ratings and calculate the average rating, namely α in the model.
For validation, we just predict α ratings and calculate MSE.
Hence, $\alpha = 4.18703$, and the MSE on the validation set is 0.7483437444993984.

Below is the code for this problem.

```
import gzip
from collections import defaultdict
import math

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

allRatings = []
userRatings = defaultdict(list)
validate_data = []
i = 1
for l in readGz("train.json.gz"):
    if i <= 100000:
        user,business = l['userID'],l['businessID']
        allRatings.append(l['rating'])
        userRatings[user].append(l['rating'])
    else:
        user,business,rate = l['userID'],l['businessID'],l['rating']
        validate_data.append([user, business, rate])
    i+=1

globalAverage = sum(allRatings) / len(allRatings)

print(globalAverage)

MSE = 0
for d in validate_data:
    MSE += (d[2]-globalAverage)**2
MSE /= 100000
print(MSE)
```

Problem 6. Fit a predictor of the form

$$rating(user, item) \approx \alpha + \beta_{user} - \beta_{item} \tag{2}$$

by fitting the mean and the two bias terms as described in the lecture notes. Use a regularization parameter of $\lambda = 1$. Report the MSE on the validation set (1 mark).

Answer: This model is more complex than the previous one.
The optimization problem of this model is

$$argmin_{\alpha,\beta} \sum_{u,i} (\alpha + \beta_u + \beta_i - R_{u,i})^2 + \lambda [\sum_u \beta_u^2 + \sum_i \beta_i^2] \tag{3}$$

If we differentiate this expression, we can acquire the following:

$$\alpha = \frac{\sum_{u,i \in train} (R_{u,i} - (\beta_u + \beta_i))}{N_{train}} \tag{4}$$

$$\beta_u = \frac{\sum_{i \in I_u} (R_{u,i} - (\alpha + \beta_i))}{\lambda + |I_u|} \tag{5}$$

$$\beta_i = \frac{\sum_{u \in U_i} (R_{u,i} - (\alpha + \beta_u))}{\lambda + |U_i|} \tag{6}$$

We need to repeat the updates above until convergence.
After the training, we can try our model on the validation set. The MSE on the validation set is: 0.6456944316928961.

Below is the code for this problem.

```
import gzip
from collections import defaultdict
import math

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

alpha = 0
beta_u = defaultdict(list)
beta_i = defaultdict(list)
users = set([])
businesses = set([])
u2b = {}
b2u = {}
train_data = []
validate_data = []
i = 1
for l in readGz("train.json.gz"):
    user,business,rate = l['userID'],l['businessID'],l['rating']
```

```
    if i <= 100000:
        train_data.append([user, business, rate])
        if user not in u2b:
            u2b[user] = set([(business,rate)])
            users.add(user)
        else:
            u2b[user].add((business, rate))
        if business not in b2u:
            b2u[business] = set([(user,rate)])
            businesses.add(business)
        else:
            b2u[business].add((user,rate))
    else:
        validate_data.append([user, business, rate])
    i+=1
    beta_u[user] = 0
    beta_i[business] = 0

users = list(users)
businesses = list(businesses)
target = businesses[0]

def iterate(lamda):
    global alpha
    # update alpha
    numerator = 0
    for d in train_data:
        numerator += (d[2] - beta_u[d[0]] - beta_i[d[1]])
    alpha = numerator/len(train_data)
    # update beta_u
    for user in users:
        total = 0
        for elem in u2b[user]:
            total += (elem[1] - alpha - beta_i[elem[0]])
        beta_u[user] = total/(lamda + len(u2b[user]))
    # update beta_i
    for b in businesses:
        total = 0
        for elem in b2u[b]:
            total += (elem[1] - alpha - beta_u[elem[0]])
        beta_i[b] = total/(lamda + len(b2u[b]))


for x in range(20):
    iterate(1)

# validate
mean_sq_error = 0
for d in validate_data:
    mean_sq_error += (alpha + beta_u[d[0]] + beta_i[d[1]] - d[2])**2
mean_sq_error = mean_sq_error/100000
print("MSE is equal to ", mean_sq_error)
```

Problem 7. Report the user and item IDs that have the largest and smallest values of β (1 mark).

Answer: By checking β_u and β_i, we can easily find the user and item with the largest and smallest values.
The user that has the largest value of β is 'U357799541' with β = 1.1643513853119356;
The user that has the largest value of β is 'U417838537' with β = -2.8318415410925115;
The item that has the largest value of β is 'B093985406' with β = 1.1719607471576514;
The item that has the smallest value of β is 'B241777680' with β = -2.2312453963097525.

Below is the code for this problem.

```
beta_max_u = 0
beta_max_i = 0
max_u = []
max_i = []

beta_min_u = 0
beta_min_i = 0
min_u = []
min_i = []

for user in users:
    if beta_u[user] > beta_max_u:
        max_u = [user]
        beta_max_u = beta_u[user]
    elif beta_u[user] == beta_max_u:
        max_u.append(user)
    if beta_u[user] < beta_min_u:
        min_u = [user]
        beta_min_u = beta_u[user]
    elif beta_u[user] == beta_min_u:
        min_u.append(user)

for b in businesses:
    if beta_i[b] > beta_max_i:
        max_i = [b]
        beta_max_i = beta_i[b]
    elif beta_i[b] == beta_max_i:
        max_i.append(b)
    if beta_i[b] < beta_min_i:
        min_i = [b]
        beta_min_i = beta_i[b]
    elif beta_i[b] == beta_min_i:
        min_i.append(b)

print(max_u, beta_max_u)
print(max_i, beta_max_i)
print(min_u, beta_min_u)
print(min_i, beta_min_i)
```

Problem 8. 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 (1 mark).

Answer: By trying different λ, we can obtain data in Fig. 2.

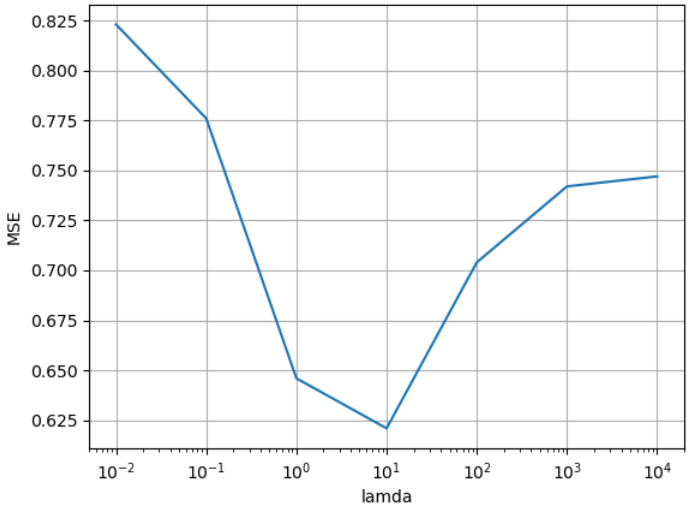

Figure 2: Kaggle profile info

Hence, it is a good choice to select $\lambda = 10$. This time the MSE on the validation set is 0.6214475097509096.

Below is the code that has already been modified to make predictions on test file.

```
import gzip
from collections import defaultdict
import math

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

alpha = 0
beta_u = defaultdict(list)
beta_i = defaultdict(list)
users = set([])
businesses = set([])
u2b = {}
b2u = {}
train_data = []
i = 1
for l in readGz("train.json.gz"):
```

```
    user,business,rate = l['userID'],l['businessID'],l['rating']
    beta_u[user] = 0
    beta_i[business] = 0
    train_data.append([user, business, rate])
    if user not in u2b:
        u2b[user] = set([(business,rate)])
        users.add(user)
    else:
        u2b[user].add((business, rate))
    if business not in b2u:
        b2u[business] = set([(user,rate)])
        businesses.add(business)
    else:
        b2u[business].add((user,rate))


users = list(users)
businesses = list(businesses)

def iterate(lamda):
    global alpha
    # update alpha
    numerator = 0
    for d in train_data:
        numerator += (d[2] - beta_u[d[0]] - beta_i[d[1]])
    alpha = numerator/len(train_data)
    # update beta_u
    for user in users:
        total = 0
        for elem in u2b[user]:
            total += (elem[1] - alpha - beta_i[elem[0]])
        beta_u[user] = total/(lamda + len(u2b[user]))
    # update beta_i
    for b in businesses:
        total = 0
        for elem in b2u[b]:
            total += (elem[1] - alpha - beta_u[elem[0]])
        beta_i[b] = total/(lamda + len(b2u[b]))
        # if b == target:
        #     print(beta_i[target])

for x in range(30):
    iterate(10)
# mean_sq_error = 0
# for d in validate_data:
#     mean_sq_error += (alpha + beta_u[d[0]] + beta_i[d[1]] - d[2])**2
# mean_sq_error = math.sqrt(mean_sq_error)
# mean_sq_error = mean_sq_error/100000
# print(mean_sq_error)

idx = 1
predictions = open("predictions_Rating.txt", 'w')
for l in open("pairs_Rating.txt"):
    if l.startswith("userID"):
        predictions.write(l)
        continue
    u,i = l.strip().split('-')
```

```
76      if u not in users:
77          if i not in businesses:
78              predictions.write(u + '-' + i + ',' + str(alpha) + '\n')
79          else:
80              predictions.write(u + '-' + i + ',' + str(alpha + beta_i[i]) + '\n')
81      else:
82          if i not in businesses:
83              predictions.write(u + '-' + i + ',' + str(alpha + beta_u[u]) + '\n')
84          else:
85              predictions.write(u + '-' + i + ',' + str(alpha + beta_u[u] + beta_i[i]) +
                                                                    '\n')
86              # if idx < 10:
87              #     print(alpha," ", beta_u[u]," ", beta_i[i])
88              #     idx+=1
89
90  predictions.close()
```

Submitted by He, Jiaqi on November 10, 2017.