CSE 258 Assignment 1

Jiaqi He

November 19, 2017

1 Visit Prediction

1.1 Problem

Predict given a (user,business) pair from 'pairs_Visit.txt' whether the user visited the business (really, whether it was one of the business they reviewed). Accuracy will be measured in terms of the categorization accuracy (1 minus the Hamming loss). The test set has been constructed such that exactly 50% of the pairs correspond to visited business and the other 50% do not. Performance will be measured in terms of the fraction of correct classifications.

1.2 Solution

1.2.1 Baseline - Based on Ratio

Find the most popular businesses that account for 50% of visits in the training data. Return '1' whenever such a business is seen at test time, '0' otherwise.

1.2.2 Baseline - Based on Type

Users may tend to repeatedly visit business of the same type. Build a baseline that returns '1' if a user has visited a business of the same category before (at least one category in common), or '0' otherwise.

The accuracy of this model on the test set on Kaggle is 0.66490.

1.2.3 Latent Factor Model

We can also use Latent Factor Model to predict visit. However, this model doesn't perform that well for this visit prediction. Hence, we don't give implementation details about latent factor model here. More details about Latent Factor Model would be covered in next problem: rating prediction.
Our goal is the following:

$$visit(user, item) \simeq \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i \tag{1}$$

where α represents the global average visit, β_u represents visit deviation of user u, β_i represents visit deviation of business i, γ_u and $\gamma_i$represents latent factors.
Our prediction threshold is set to be 0.5, namely, if prediction is greater than 0.5, we then predict '1'; if prediction is less than or equal to 0.5, we predict '0'.
In such a way, we get 0.70231 score on Kaggle. Below is the code for this approach.

```
import gzip
from collections import defaultdict
import math
import numpy as np
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 = []
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, 1])
    if user not in u2b:
        u2b[user] = set([(business,1)])
        users.add(user)
    else:
        u2b[user].add((business, 1))
    if business not in b2u:
        b2u[business] = set([(user,1)])
        businesses.add(business)
    else:
        b2u[business].add((user,1))

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])
train_data = train_data[:100000]
train_data = train_data + neg_samples
# matrix factorization

k = 10
user_map = {}
business_map = {}
idx_user = 0            #the number of different users
idx_business = 0        #the number of different businesses
for user in users:
    user_map[user] = idx_user
    idx_user += 1
for business in businesses:
    business_map[business] = idx_business
    idx_business += 1
p = np.random.rand(idx_user, k)*math.sqrt(5.0/k)      # user
q = np.random.rand(idx_business, k)*math.sqrt(5.0/k) # business

def iterate(lamda):
    global alpha
    # update alpha
    numerator = 0
    for d in train_data:
        u = user_map[d[0]]
        i = business_map[d[1]]
        numerator += (d[2] - beta_u[d[0]] - beta_i[d[1]] - np.dot(p[u,],q[i,]))
    alpha = numerator/len(train_data)
```

```
    # update beta_u
    for user in users:
        u = user_map[user]
        total = 0
        for elem in u2b[user]:
            i = business_map[elem[0]]
            total += (elem[1] - alpha - beta_i[elem[0]] - np.dot(p[u,],q[i,]))
        beta_u[user] = total/(lamda + len(u2b[user]))
    # update beta_i
    for b in businesses:
        i = business_map[b]
        total = 0
        for elem in b2u[b]:
            u = user_map[elem[0]]
            total += (elem[1] - alpha - beta_u[elem[0]] - np.dot(p[u,],q[i,]))
        beta_i[b] = total/(lamda + len(b2u[b]))
    # update p_u
    for user in users:
        u = user_map[user]
        temp_p = [0] * k
        temp_p = np.array(temp_p)
        denominator = 0
        for elem in u2b[user]:
            i = business_map[elem[0]]
            temp_p = (elem[1] - alpha - beta_u[user] - beta_i[elem[0]])*q[i,] + temp_p
            denominator += sum(q[i,]**2)
        denominator += lamda
        p[u,] = temp_p/denominator
    # update q_i
    for b in businesses:
        i = business_map[b]
        temp_q = [0] * k
        temp_q = np.array(temp_q)
        denominator = 0
        for elem in b2u[b]:
            u = user_map[elem[0]]
            temp_q = (elem[1] - alpha - beta_u[elem[0]] - beta_i[b])*p[u,] + temp_q
            denominator += sum(p[u,]**2)
        denominator += lamda
        q[i,] = temp_q/denominator

for x in range(20):
    iterate(7)
    print(alpha)


threshold = 0.5
predictions = open("predictions_Visit1.txt", 'w')
for l in open("pairs_Visit.txt"):
    if l.startswith("userID"):
        predictions.write(l)
        continue
    u,i = l.strip().split('-')
    if u not in users:
        if i not in businesses:
            if alpha > threshold:
                predictions.write(u + '-' + i + ',1\n')
            else:
                predictions.write(u + '-' + i + ',0\n')
        else:
            if alpha + beta_i[i] > threshold:
```

```
                predictions.write(u + '-' + i + ',1\n')
            else:
                predictions.write(u + '-' + i + ',0\n')
    else:
        if i not in businesses:
            if alpha + beta_u[u] > threshold:
                predictions.write(u + '-' + i + ',1\n')
            else:
                predictions.write(u + '-' + i + ',0\n')
        else:
            u_idx = user_map[u]
            i_idx = business_map[i]
            predicted_rating = alpha + beta_u[u] + beta_i[i] + np.dot(p[u_idx,],q[i_idx,]
                                                                      )
            if predicted_rating > threshold:
                predictions.write(u + '-' + i + ',1\n')
            else:
                predictions.write(u + '-' + i + ',0\n')

predictions.close()
```

1.2.4 k-Nearest Neighbor Algorithm

k-Nearest Neighbor Algorithm is one of the most powerful recommendation methods.Based on the history of businesses that a user visited and the history of visits to a business, this model predicts new possible visits.
The idea is as follows:

1. For each user-business pair (u, b) that we need to predict, we first find this user's k nearest neighbors, namely the top k users who behave similarly the most as this user.We then have a clustering of users with size of k. We denote this clustering as $U_{similar}$.
2. Then we start to find the history of visits of this clustering. We denote the set of all businesses that are visited by this clustering as $B_{visited}$.
3. If the business b in the pair is within $B_{visited}$, we then return '1', otherwise we return '0'.

Here we use cosine similarity to find similar users. Cosine similarity is defined as

$$sim(A, B) = \frac{A \cdot B}{\parallel A \parallel \parallel B \parallel} \tag{2}$$

For two users, we get their history of visits. Hence, $A \cdot B$ is the intersection of these two sets, and $\parallel$ A $\parallel$ is the square root of the size A, and $\parallel$ B $\parallel$ is the square root of the size B.
In this problem, we first set $k = 50$, namely we want to find top 50 users that are similar to the user we want to predict. Then we gather all businesses that these 50 users visit to be $B_{visited}$.
Finally, we check if b is in $B_{visited}$. For this model, we achieve 0.75380 score on Kaggle.
If we set $k = 500$, namely we want to find top 500 users that are similar to the user we want to predict. Then we gather all businesses that these 500 users visit to be $B_{visited}$. Then we achieve **0.87024** score on Kaggle, which is better than the previous case.
Below is the code for this model.

```
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)

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)

users = list(users)

def findneighbour(user, users, u2b, k):
    visited = u2b[user]
    neighbours = []
    for cur_user in users:
        if cur_user == user:
            continue
        else:
            cur_visited = u2b[cur_user]
            intersec = visited.intersection(cur_visited)
            if len(intersec) == 0:
                continue
            else:
                similarity = len(intersec)/math.sqrt(len(visited)*len(cur_visited)) #
                                                                    cosine similarity
                neighbours.append([similarity, cur_user])
    neighbours.sort()
    neighbours.reverse()
    neighbours = neighbours[:k]
    return neighbours

######################## Approach 1 ###################################
# def predict(neighbours, u2b, business):
#     numerator = 0
#     denumerator = 0
#     for i in range(len(neighbours)):
#         user = neighbours[i][1]
#         sim = neighbours[i][0]
#         denumerator += sim
#         if business in u2b[user]:
#             numerator += sim
#     if numerator/denumerator > 0.5:
#         return True
#     else:
#         return False
#
# 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 u2b:
#          neighbours = findneighbour(u, users, u2b, 5) # when set 50, the score is 0.
                                                    75380
#          if predict(neighbours, u2b, i):
#              predictions.write(u + '-' + i + ",1\n")
#          else:
#              predictions.write(u + '-' + i + ",0\n")
#      else:
#          predictions.write(u + '-' + i + ",0\n")
#
# predictions.close()

###################### Approach 2 ####################################
def predict(neighbours, u2b, business):
    for i in range(len(neighbours)):
        user = neighbours[i][1]
        if business in u2b[user]:
            return True
    return False

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 u2b:
        neighbours = findneighbour(u, users, u2b, 500) # when set 50, the score is 0.
                                                        75380
        if predict(neighbours, u2b, i):
            predictions.write(u + '-' + i + ",1\n")
        else:
            predictions.write(u + '-' + i + ",0\n")
    else:
        predictions.write(u + '-' + i + ",0\n")

predictions.close()
```

2 Rating Prediction

2.1 Problem

Predict people's star ratings as accurately as possible, for those (user,item) pairs in 'pairs_Rating.txt'. Accuracy will be measured in terms of the (root) mean-squared error (RMSE).

2.2 Solution

2.2.1 Baseline - Global Average

Return the global average rating, or the user's average if we have seen them before in the training data.

$$rating(user, item) = \alpha \tag{3}$$

2.2.2 Baseline - Linear Model

Fit a predictor of the form

$$rating(user, item) \simeq \alpha + \beta_u + \beta_i \tag{4}$$

by fitting the mean and the two bias terms as described in the lecture notes.

2.2.3 Latent Factor Model

Our goal is the following:

$$rating(user, item) \simeq \alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i \tag{5}$$

where α represents the global average rating, β_u represents rating deviation of user u, β_i represents rating deviation of business i, γ_u and $\gamma_i$represents latent factors.
In this model, we not only consider independent factors like β_u, β_i, but also consider interactive factors $\gamma_u, and \gamma_i$, which would yield more accurate model.
The steps are as follows:

1. Set initial values for $\alpha, \beta_u, \beta_i, \gamma_u, \gamma_i$. To begin with, we can utilize results in Homework 3, we know that the global average is roughly 4.2, so we set $\alpha = 4.2$ to begin. For β_u, β_i, they are set to be 0 for each user and item. For γ_u, γ_i, we use random function to create a matrix to start.
2. Iterate and update parameters until convergence.
3. Use this latent factor model to make predictions.

Our optimization goal is

$$argmin_{\alpha,\beta,\gamma} \sum_{u,i} (\alpha + \beta_u + \beta_i + \gamma_u \cdot \gamma_i - R_{u,i})^2 + \lambda [\sum_u \beta_u^2 + \sum_i \beta_i^2 + \sum_i \parallel \gamma_i \parallel_2^2 + \sum_u \parallel \gamma_u \parallel_2^2] \tag{6}$$

If we take the derivative of the equation 6 seperately in terms of $\alpha, \beta_u, \beta_i, \gamma_u, \gamma_i$, we would get

$$\alpha = \frac{\sum_{u,i}(R_{u,i} - \beta_u - \beta_i - \gamma_u \cdot \gamma_i)}{N_{train}} \tag{7}$$

$$\beta_u = \frac{\sum_{i \in I_u}(R_{u,i} - \alpha - \beta_i - \gamma_u \cdot \gamma_i)}{\lambda + |I_u|} \tag{8}$$

$$\beta_i = \frac{\sum_{u \in U_i}(R_{u,i} - \alpha - \beta_u - \gamma_u \cdot \gamma_i)}{\lambda + |U_i|} \tag{9}$$

$$\gamma_u = \frac{\sum_{i \in I_u}(R_{u,i} - \alpha - \beta_u - \beta_i)\gamma_i}{\lambda + \sum_{i \in I_u} \gamma_i^2} \tag{10}$$

$$\gamma_i = \frac{\sum_{u \in U_i}(R_{u,i} - \alpha - \beta_u - \beta_i)\gamma_u}{\lambda + \sum_{u \in U_i} \gamma_u^2} \tag{11}$$

For each iteration, we modify $\alpha, \beta_u, \beta_i, \gamma_u, \gamma_i$ using equations above. We keep iterating them until convergence.

When we first set $\lambda = 1$ and iteration times to be 100, we have found that the model starts to converge at approximately 50 iterations. And the MSE on the training set is 0.27, which is quite small. However, when we upload this model's prediction to Kaggle, we get 0.81 score, which is quite unacceptable!

This indicates that this model is **overfitting**. Hence, we need to increase λ a bit to overcome overfitting. When setting $\lambda = 5$, we get 0.75216 score on Kaggle, which is a good signal that we have indeed alleviated the bad effects of overfitting. After many attempts, we find that setting $\lambda = 7$ is a wise choice, and we achieve **0.74963** score on Kaggle.

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 = 4.2
beta_u = defaultdict(list)
beta_i = defaultdict(list)
users = set([])
businesses = set([])
u2b = {}
b2u = {}
train_data = []
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))


# matrix factorization
import numpy as np
k = 1
user_map = {}
business_map = {}
idx_user = 0            #the number of different users
idx_business = 0        #the number of different businesses
for user in users:
    user_map[user] = idx_user
    idx_user += 1
for business in businesses:
    business_map[business] = idx_business
    idx_business += 1
```

```
p = np.random.rand(idx_user, k)*math.sqrt(5.0/k)      # user
q = np.random.rand(idx_business, k)*math.sqrt(5.0/k) # business

def iterate(lamda):
    global alpha
    # update alpha
    numerator = 0
    for d in train_data:
        u = user_map[d[0]]
        i = business_map[d[1]]
        numerator += (d[2] - beta_u[d[0]] - beta_i[d[1]] - np.dot(p[u,],q[i,]))
    alpha = numerator/len(train_data)
    # update beta_u
    for user in users:
        u = user_map[user]
        total = 0
        for elem in u2b[user]:
            i = business_map[elem[0]]
            total += (elem[1] - alpha - beta_i[elem[0]] - np.dot(p[u,],q[i,]))
        beta_u[user] = total/(lamda + len(u2b[user]))
    # update beta_i
    for b in businesses:
        i = business_map[b]
        total = 0
        for elem in b2u[b]:
            u = user_map[elem[0]]
            total += (elem[1] - alpha - beta_u[elem[0]] - np.dot(p[u,],q[i,]))
        beta_i[b] = total/(lamda + len(b2u[b]))
    # update p_u
    for user in users:
        u = user_map[user]
        temp_p = [0] * k
        temp_p = np.array(temp_p)
        denominator = 0
        for elem in u2b[user]:
            i = business_map[elem[0]]
            temp_p = (elem[1] - alpha - beta_u[user] - beta_i[elem[0]])*q[i,] + temp_p
            denominator += sum(q[i,]**2)
        denominator += lamda
        p[u,] = temp_p/denominator
    # update q_i
    for b in businesses:
        i = business_map[b]
        temp_q = [0] * k
        temp_q = np.array(temp_q)
        denominator = 0
        for elem in b2u[b]:
            u = user_map[elem[0]]
            temp_q = (elem[1] - alpha - beta_u[elem[0]] - beta_i[b])*p[u,] + temp_q
            denominator += sum(p[u,]**2)
        denominator += lamda
        q[i,] = temp_q/denominator

for x in range(30):
    iterate(7)
    if x%10 == 0:
        print(alpha)

error = 0
for l in readGz("train.json.gz"):
    user,business,r = l['userID'],l['businessID'],l['rating']
```

```
    u = user_map[user]
    i = business_map[business]
    error += (r-np.dot(q[i,],p[u,])-alpha-beta_u[user]-beta_i[business])**2
print(error/200000)

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('-')
    if u not in users:
        if i not in businesses:
            predictions.write(u + '-' + i + ',' + str(alpha) + '\n')
        else:
            predictions.write(u + '-' + i + ',' + str(alpha + beta_i[i]) + '\n')
    else:
        if i not in businesses:
            predictions.write(u + '-' + i + ',' + str(alpha + beta_u[u]) + '\n')
        else:
            u_idx = user_map[u]
            i_idx = business_map[i]
            predicted_rating = alpha + beta_u[u] + beta_i[i] + np.dot(p[u_idx,],q[i_idx,]
                                                                      )
            if predicted_rating > 5.0:
                predicted_rating = 5.0
            predictions.write(u + '-' + i + ',' + str(predicted_rating) + '\n')

predictions.close()
```