CSE 258 - Assignment 1

Kaggle username: NanShao

1. Purchase prediction

The model I chose to use was to use jaccard similarity to find both the most similar user and item for the user and item we want to predict. This model ended up with a accuracy of 0.664 on kaggle. I used the whole dataset for building KNN model for both user and item.

- Jaccard similarity

The model computes the jaccard similarity between all users and all items. The jaccard similarity is based on what categories a user has bought before(what categories this item belongs to). According to the formula of jaccard similarity: J(A,B) = |A and B| / |A or B|, we can get the similarity of users and items.

To predict whether a specific item will be bought by a specific user, we first find the K most similar items, and chect whether they has been bought by user, if the answer is yes, it is very possible that this item will be bought. Likewise, we find the K most similar users, and check whether they have bought this item, if yes, it is very possible that this user will buy this item.

- Computing cost

When computing similarities for each user and each item, we have to compare a item(or user) with all other items(or users). We roughly have 10k + users and items, so it is really costly for computations. and it becomes an infeasible. So I basically did some improvement: I defined that the similar user must have bought at least one item in common(the similar item must have been bought by at least one user in common). In this way, I massively reduce the time needed to calculate similarities of user and item.

- Parameter selection

To find the best K of similar user and item, I did several experiments on it. Finally, the best Ks of them are 786 and 171 respectively.

- Future work

Due to the limited time and resources, the accuracy of my model is not that perfect. I think there are some aspects that I can work on to make improvements. Firstly, I can add more features into this prediction problem like the gender of users, the keyword in reviews and some global features like popularity of items. Moreover, I should get a balance between these features, give them different weights to differentiate the importance.

- Relevant code

```
# get similarity between two items
def get_simi(i1,i2):
    # just use Jaccard similarity: J(A,B) = |A and B| / |A or B|
    a_and_b = 0
    a_or_b = 0
    for cate in item_info[i1]:
        if cate in item_info[i2]:
            a_and_b += 1
        else:
            a_or_b += 1
    a_or_b += len(item_info[i2])
    return a_and_b / (a_or_b * 1.0)


# input: user, item, item_relation, item_users, k
def will_purchase(u,i,i_r,i_u,k):
    if (u not in user_info) or (i not in item_info):
        return random.randint(0,1)
    # find k closet item, if they has been bought by u, i will bought by u
    friends = []
    for f in i_r[i]:
        friends.append(f)
    friends = sorted(friends, key=lambda x: x[1], reverse=True)
    index = 0
    if k > len(friends):
        k = len(friends)
    while index < k:
        item = friends[index][0]
        if u in i_u[item]:
            return 1
        index += 1
    return 0


# find the best k
klist = [1,2,5,10, 164, 166, 168, 169, 170, 171, 172, 173, 174, 175]
accs = []
for k in klist:
    acc = 0
    '''
    for d in test_data[:1000]:
        acc += will_purchase(d[0],d[1],item_relation,item_users,k)
    '''
    for d in test_data[1000:]:
        acc += (1-will_purchase(d[0],d[1],item_relation,item_users,k))
```

```
    acc /= (1000 * 1.0)
    accs.append(acc)
```

2. Rating prediction

I used the latent factor model to get the best result in rating prediction. The final score was 1.114.

Latent factor model:

Prediction = alpha + beta*user + beta*item + gamma*user + gamma*item

alpha: the average value of rating for all the training data

beta_user: the user rating bias between personal rating tendency and average rating value

beta_item: the item rating bias between items received rating and average rating value

gamma*user(gamma*item): randomly initialized with dimension K

1. I first initicalize the alpha, beta*user and beta*item to be 0.
2. Train the Latent factor model without gamma, get the beta*user and beta*item.
3. Fix the beta*user and beta*item(based on the local optimal result), I initialize the gamma*user and gamma*item to be random float number between -0.5 and 0.5, with normalized distribution(with the best dimension K = 1).
4. Run gradient descent with different lambda(best lambda = 6.5), get the best parameters.

```
def fun88(lamb):
    times = 50
    for i in range(times):
        alp = 0
        for uu in userrating:
            for bb in userrating[uu]:
                alp += userrating[uu][bb] - (beta_user[uu] + beta_item[bb] + g_u[u
u]*g_i[bb])
        alp /= len(train)
        for uu in userrating:
            beta_user[uu] = 0
            for bb in userrating[uu]:
                beta_user[uu] += userrating[uu][bb] - (alp + beta_item[bb] + g_u[u
u]*g_i[bb])
            beta_user[uu] /= (lamb + len(userrating[uu]))
        for bb in businessrating:
            beta_item[bb] = 0
            for uu in businessrating[bb]:
```

```
                beta_item[bb] += businessrating[bb][uu] - (alp + beta_user[uu] + g
_u[uu]*g_i[bb])
            beta_item[bb] /= (lamb + len(businessrating[bb]))
        for uu in userrating:
            g_u[uu] = np.random.normal(0.0,.1)#random.uniform(-.5,.5)
            for bb in userrating[uu]:
                g_u[uu] += g_i[bb] * (userrating[uu][bb] - (alp + beta_user[uu] +
beta_item[bb] + g_u[uu]*g_i[bb]))
            g_u[uu] /= lamb
        for bb in businessrating:
            g_i[bb] = np.random.normal(0.0,.1)#random.uniform(-.5,.5)
            for uu in businessrating[bb]:
                g_i[bb] += g_u[uu] * (businessrating[bb][uu] - (alp + beta_user[uu
] + beta_item[bb] + g_u[uu]*g_i[bb]))
            g_i[bb] /= lamb
        mse = 0
        for uu in userrating:
            for bb in userrating[uu]:
                mse += (alp + beta_user[uu] + beta_item[bb] - userrating[uu][bb])
**2
        mse /= len(train)

    validation_user = defaultdict(lambda: defaultdict(int))
    validation_business = defaultdict(lambda: defaultdict(int))
    for d in valid:
        uu, bb = d['reviewerID'], d['itemID']
        validation_user[uu][bb] = d['rating']
        validation_business[bb][uu] = d['rating']
    mse = 0
    for uu in validation_user:
        for bb in validation_user[uu]:
            mse += ((alp + (beta_user[uu] if uu in beta_user else 0) +
                   (beta_item[bb] if bb in beta_item else 0) +
                     (g_u[uu] if uu in g_u else 0) * (g_i[bb] if bb in g_i else 0)
 -
                    validation_user[uu][bb]) **2)

    mse /= len(valid)
    return mse, alp, beta_user, beta_item, g_u, g_i


# find the best lamb
lambdas = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0, 10000.0]
mse = []
for l in lambdas:
    mse.append(fun88(l))
for i in range(len(mse)):
    m = mse[i]
```

```
    l = lambdas[i]
    print str(m) +'\t'+ str(l)
```