Report Assignment 1

Name: Linbin Yang

Student ID: A53277054

Kaggle: linbinisme

Contents:

Task one

This task requires us to make purchase predictions. For this question, I adopt collaborating filtering method. I used three kinds of methods (Cosine, Jaccard and Pearson) to measure the similarity between users or items. The Pseudocode below shows how I solve this problem. My final results is 0.60614 which is higher than the baseline. My best model's score so far is 0.63549, and this model just uses categories to predict purchase, that is, if the item's categories is one subset of purchase categories that this user has, we predict true.

[code 1]

```
def function (UserID_input, ItemID_input){
If UserID_input exists and ItemID_input exists{
    compare similarity between the items this user used to buy and the input item.
    If similarity is larger than 0 return True
    return False
}
return True
}
```

For similarity measurement, I try jaccard similarity and cosine similarity. Here I would introduce how I compute the jaccard similarity:

Instead of using really sparse vector, I define two defaultdict for user and item:

user1: {item1, item2, item3}

user2: {item3, item2}

We found that both user1 and user2 buy item2 and item3 and all together they buy 3 items, so for jaccard similarity, the result is 2/3 = 0.6666.

Using this method avoids storing sparse matrix in memory and iterating over all dataset and program runs very fast.

Task two

This task requires us to make rating predictions. I have two model for this problem. One is to use naïve bayesian model, which assume that the user feature is independent between each other:

$$f(u,i) = \alpha + \beta_u + \beta_i$$

Where alpha represents the average rating score of all items in the training set, beta_u measures how user u's score toward item I would deviate from the average score. beta_i is similar to beta_u.

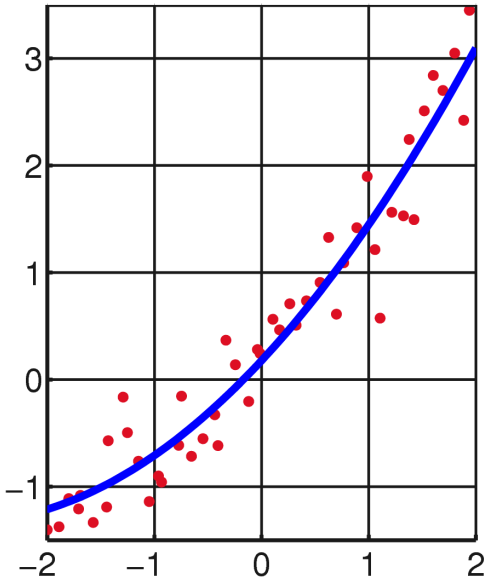
The objective functions for this model is:

$$Obj = \sum_{u,i} (\alpha + \beta_u + \beta_i - R_{u,i})^2 + \lambda(\sum_u {\beta_u}^2 + \sum_i {\beta_i}^2)$$

The optimization method for this Objective function is **Alternating Least Squares (ALS)**. We just repeating iterations on the derivate equations below:

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

Why this optimization methods would work here? Here is my explanation to this: the objective function for this model is convex, that is to say, this objective function is sure to have one global optimum. We want to minimize the Objective function as much as possible and the condition is that all its derivative equations equal to zero.

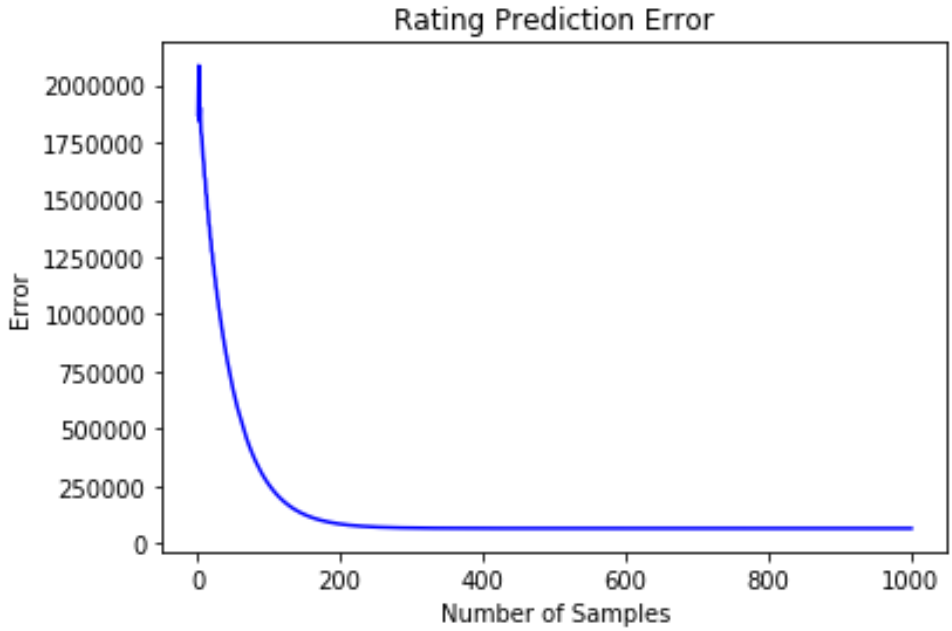


G1 Alternating Least Square Method

We start by initialize alpha, beta_u and beta_i randomly, we can see from we apply this method to ensure that the data would finally obey the true distribution (converge) and the final coefficients are the result we want. The graph below (See G1) can give you hints on how ALS works in coefficients inference in one convex model.

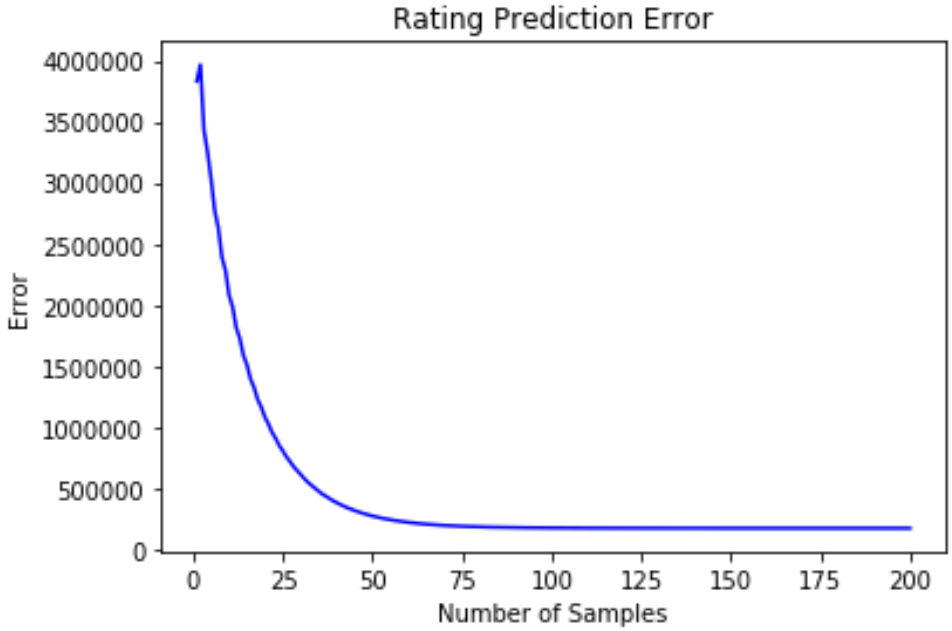
During the experiment part, I found that when I tune lam = 6.6 and train my model on the entire 20W dataset. I got score 1.18 on the Kaggle. If I use 10W as training set

and 10W as validation set, this model achieves MSE 1.28 on validation set. The graph below shows how this model converges on lam = 6.6.



G2 Alternating Least Square Method lam=6.6

One thing I found that is very interesting is that when you set different lam for beta_u and beta_i, your performance on Kaggle could increase a lot! Here, I set lam_u for beta_u as 3.0 and set lam_i for beta_i as 5.0. My performance on Kaggle just increase to 1.14, ranked top 100!



G3 Alternating Least Square Method lam_u=3.0 lam_i=5.0

I think it is because we penalized beta_u and beta_i separately, we actually enhance the users' inclinations in this model by setting lam_u as just 3.0. That is to say,

user's inclination influences more on the final rating results compared with the item. And that is why our model have a better performance after this change.

Another model I use for this task is latent-factor model which takes the relation between users and items into considerations. The equations is:

$$f(u,i) = \alpha + \beta_u + \beta_i + \lambda_u * \lambda_i$$

Where lam_u and lam_i are m X k and k X n matrix. k is the hyper parameters, m is the number of users and n is the number of items.

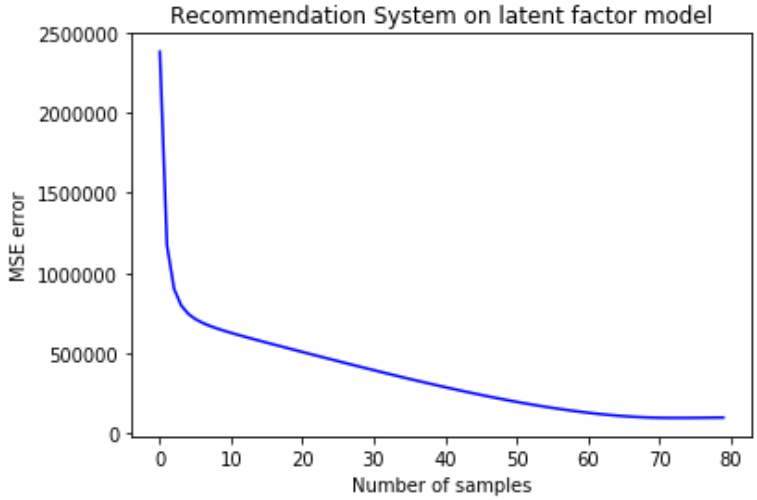
Our objective function now becomes:

$$Obj = \sum_{u,i}(\alpha + \beta_u + \beta_i + \lambda_u * \lambda_i - R_{u,i})^2 + \lambda(\sum_u {\beta_u}^2 + \sum_i {\beta_i}^2 + \sum_u ||\lambda_u||_2^2 + \sum_i ||\lambda_i||_2^2)$$

Here, we use **Stochastic Gradient Descent (SGD)** to optimize this objective function. The derivative function for lam_u[k] is as follows:

$$\frac{dObj}{d\lambda_{uk}} = \sum_{i \in I_u} 2 * (\alpha + \beta_u + \beta_i + \lambda_u * \lambda_i - R_{u,i}) * \lambda_{ik} + 2 * \lambda * \lambda_{uk}$$

For this problem, we need to define learning rate, lam and k all by ourselves. After trying many times, I set k = 1, lam = 6.6 and learning_rate = 0.000166. I got good performance on validation dataset compared with model 1 for task two, that is the MSE on validation set is 1.17, compared with 1.28 of the model 1. The graph below shows how latent factor model converges on the 10W training data.



G4 Stochastic Gradient Descent for latent factor model

Code for Task 1

In [70]:

```
import gzip
import numpy as np
import matplotlib.pyplot as plt
from collections import defaultdict
from mpl_toolkits.mplot3d import Axes3D
```

In [94]:

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

In [95]:

```
for elem in readGz("/Users/linbinyang/Desktop/course-fall2018/cse258/data/assi
gnment1/train.json.gz"):
    print (elem)
    break
```

```
{'reviewTime': '09 26, 2013', 'reviewText': "The model in this pic
ture has them rolled up at the top because they are actually very
high waisted! that's my only complaint though, because they are ve
ry good quality, and fit really well! I am 5'2&amp;#34; 120lbs with th
ick thighs and i love them i can't wait to wear them out!", 'helpf
ul': {'nHelpful': 0, 'outOf': 0}, 'reviewerID': 'U490934656', 'rev
iewHash': 'R798569390', 'categories': [['Clothing, Shoes & Jewelry
', 'Women'], ['Clothing, Shoes & Jewelry', 'Novelty, Costumes & Mo
re', 'Novelty', 'Clothing', 'Women', 'Leggings']], 'unixReviewTime
': 1380153600, 'itemID': 'I402344648', 'rating': 4.0, 'summary': '
High Waisted', 'categoryID': 0}
```

In [147]:

```
all_set = []
user_set = set()
item_set = set()
for elem in readGz("/Users/linbinyang/Desktop/course-fall2018/cse258/data/assi
gnment1/train.json.gz"):
    all_set.append(elem)
    user_set.add(elem['reviewerID'])
    item_set.add(elem['itemID'])
```

In [167]:

```
# we use cosine similarity measure similarity between users and items
user = defaultdict(list)
item = defaultdict(list)
for elem in all_set:
    item_R = []
    user_R = []
    user_R.append(elem['reviewerID'])
    item_R.append(elem['itemID'])
    user_R.append(elem['rating'])
    item_R.append(elem['rating'])
    user[elem['reviewerID']].append(item_R)
    item[elem['itemID']].append(user_R)
```

In [171]:

```
print (f"The total number of users is {len(user)}")
```

```
The total number of users is 39239
```

In [149]:

```
print (f"The total number of items is {len(item)}")
```

```
The total number of items is 19914
```

In [123]:

```
def CalculatePetersonSimilarity(user1, user2, user, user_set):
    res = 0
    r_u1_avg = sum([item[1] for item in user[user1]])/len(user[user1])
    r_u2_avg = sum([item[1] for item in user[user2]])/len(user[user2])
    item_intersec = []
    for item_1 in user[user1]:
        for item_2 in user[user2]:
            if item_1[0] == item_2[0]:
                item_intersec.append(item_1)
                break
    if len(item_intersec) == 0:
        return res
    else:
        for elem in item_intersec:
            res = res + (elem[1] - r_u1_avg) * (elem[1] - r_u2_avg)
    deno1 = 0
    for elem in item_intersec:
        deno1 = deno1 + (elem[1] - r_u1_avg)**2
    deno2 = 0
    for elem in item_intersec:
        deno2 = deno2 + (elem[1] - r_u2_avg)**2
    return res/((deno1*deno2)**(1/2))
```

In [172]:

```
def CalculateJacardSimilarity_user(user1, user2, user, user_set):
    if user1 not in user or user2 not in user:
        return 0
    item1 = user[user1]
    item2 = user[user2]
    similarity = 0
    for each_item_1 in item1:
        item_id = each_item_1[0]
        for each_item_2 in item2:
            if each_item_2[0] == item_id:
                similarity = similarity + 1
                break
    item_set = set()
    for item in item1:
        item_set.add(item[0])
    for item in item2:
        item_set.add(item[0])
    return similarity/len(item_set)
```

In [82]:

```
def CalculateJacardSimilarity_item(item1, item2, item, item_set):
    if item1 not in item_set:
        return 1
    user1 = item[item1]
    user2 = item[item2]
    similarity = 0
    for each_user_1 in user1:
        user_id = each_user_1[0]
        for each_user_2 in user2:
            if each_user_2[0] == user_id:
                similarity = similarity + 1
                break
    user_set = set()
    for user_i in user1:
        user_set.add(user_i[0])
    for user_i in user2:
        user_set.add(user_i[0])
    return similarity/len(user_set)
```

In [173]:

```
def BuilderSimilarityUser(user_give, user, user_set):
    res = []
    for user_each in user_set:
        if user_each == user_give:
            continue
        if CalculateJacardSimilarity_user(user_give, user_each, user, user_set
) > 0:
            res.append(user_each)
    return res
```

In [ ]:

```
user_sim = defaultdict(list)
for any_user in user_set:
    user_sim[any_user] = BuilderSimilarityUser(any_user, user, user_set)
```

In [ ]:

```
def Judge(u, i, user_sim, item):
    for each_user in user_sim[u]:
        for each_item in item[each_user]:
            if i == each_item:
                flag = True
                return flag
    return False
```

In [143]:

```
def PredictBuy_v3(user_g, item_g, item, user, item_set, user_set):
    if user_g not in user_set or item_g not in item_set:
        return False
    else:
        for each_user in item[item_g]:
            if CalculateJacardSimilarity_user(user_g, each_user[0], user, user
_set) > 0.2:
                return True
    return False
```

In [117]:

```
print (CalculatePetersonSimilarity('U507366950', 'U507366950', user, user_set)
)
```

```
[['I464613034', 5.0], ['I872967861', 5.0], ['I506949867', 5.0], ['
I723155560', 5.0], ['I029346709', 5.0], ['I264463508', 2.0], ['I07
5704956', 5.0], ['I847796665', 5.0], ['I787056386', 2.0], ['I22595
5774', 2.0], ['I323568513', 5.0], ['I241584112', 5.0], ['I93033965
0', 3.0]]
1.0
```

In [144]:

```
f = open("/Users/linbinyang/Downloads/assignment1/pairs_Purchase.txt")
f_1 = open("output_pp20.txt","w")
index = 0
while 1:
    line = f.readline().strip()
    if not line:
        break
    if index == 0:
        f_1.write(line+'\n')
        index = index + 1
        continue
    u = line.split("-")[0]
    i = line.split("-")[1]
    flag = Judge(u, i, user_sim, item)
    if flag:
        f_1.write(u + '-' + i + ',' + '1'+'\n')
    else:
        f_1.write(u + '-' + i + ',' + '0'+'\n')
f_1.close()
f.close()
```

Code for Task 2

In [2]:

```
import gzip
import numpy as np
import matplotlib.pyplot as plt
from collections import defaultdict
from mpl_toolkits.mplot3d import Axes3D
```

In [3]:

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

In [84]:

```
training_set = []
validation_set = []
all_set = []
i = 1
for elem in readGz("/Users/linbinyang/Desktop/course-fall2018/cse258/data/assignment1/train.json.gz"):
    if i <= 100000:
        training_set.append(elem)
        i = i + 1
    elif i <= 200000:
        validation_set.append(elem)
        i = i + 1
    all_set.append(elem)
```

In [106]:

```
# We have build our training data and validation data
user = defaultdict(list)
item = defaultdict(list)
U_I_R = []
for elem in training_set:
    unit = []
    item_R = []
    user_R = []
    unit.append(elem['reviewerID'])
    user_R.append(elem['reviewerID'])
    unit.append(elem['itemID'])
    item_R.append(elem['itemID'])
    unit.append(elem['rating'])
    user_R.append(elem['rating'])
    item_R.append(elem['rating'])
    U_I_R.append(unit)
    user[elem['reviewerID']].append(item_R)
    item[elem['itemID']].append(user_R)
```

In [6]:

```
def inner_multiply(x, y):
    return sum([x[i]*y[i] for i in range(len(x))])
```

In [7]:

```
def ObjectFunction(alpha, beta_u, beta_i, gamma_u, gamma_i, lam, U_I_R, user,
item):
    res = 0
    for elem in U_I_R:
        res = res + (alpha + beta_u[elem[0]] + beta_i[elem[1]] + inner_multipl
y(gamma_u[elem[0]], gamma_i[elem[1]]) - elem[2])**2
    for elem in user:
        res = res + lam*beta_u[elem] * beta_u[elem]
        res = res + lam*sum([unit_gamma_u*unit_gamma_u for unit_gamma_u in gam
ma_u[elem]])
    for elem in item:
        res = res + lam*beta_i[elem] * beta_i[elem]
        res = res + lam*sum([unit_gamma_i*unit_gamma_i for unit_gamma_i in gam
ma_i[elem]])
    return res
```

In [68]:

```
def ObjectPrime(alpha, beta_u, beta_i, gamma_u, gamma_i, lam, U_I_R, user, ite
m):
    res = []
    d_alpha = 0
    d_beta_u = defaultdict(int)
    d_beta_i = defaultdict(int)
    d_gamma_u = defaultdict(list)
    d_gamma_i = defaultdict(list)
    for elem in U_I_R:
        d_alpha = alpha + 2*(beta_u[elem[0]] + beta_i[elem[1]] + inner_multipl
y(gamma_u[elem[0]], gamma_i[elem[1]])-elem[2])
    for key_u in beta_u:
        d_beta_u[key_u] = 0
        for item_unit in user[key_u]:
            d_beta_u[key_u] = d_beta_u[key_u] + 2*(alpha + beta_u[key_u] + bet
a_i[item_unit[0]] + inner_multiply(gamma_u[key_u], gamma_i[item_unit[0]])-item
_unit[1])
        d_beta_u[key_u] = d_beta_u[key_u] + 2*lam*beta_u[key_u]
    for key_i in beta_i:
        d_beta_i[key_i] = 0
        for user_unit in item[key_i]:
            d_beta_i[key_i] = d_beta_i[key_i] + 2*(alpha + beta_u[user_unit[0]
] + beta_i[key_i] + inner_multiply(gamma_u[user_unit[0]], gamma_i[key_i]) - us
er_unit[1])
        d_beta_i[key_i] = d_beta_i[key_i] + 2*lam*beta_i[key_i]
    for user_unit in gamma_u:
        unit_gamma_u = [0]*len(gamma_u[user_unit])
        for i in range(len(gamma_u[user_unit])):
            for item_unit in user[user_unit]:
                unit_gamma_u[i] = unit_gamma_u[i] + 2*(alpha+beta_u[user_unit]
+beta_i[item_unit[0]]+inner_multiply(gamma_u[user_unit], gamma_i[item_unit[0]]
)-item_unit[1])*gamma_i[item_unit[0]][i]
            unit_gamma_u[i] = unit_gamma_u[i] + 2*lam*gamma_u[user_unit][i]
        d_gamma_u[user_unit] = unit_gamma_u
    for item_unit in gamma_i:
        unit_gamma_i = [0]*len(gamma_i[item_unit])
        for i in range(len(gamma_i[item_unit])):
            for user_unit in item[item_unit]:
                unit_gamma_i[i] = unit_gamma_i[i] + 2*(alpha+beta_u[user_unit[
0]]+beta_i[item_unit]+inner_multiply(gamma_u[user_unit[0]], gamma_i[item_unit]
)-user_unit[1])*gamma_u[user_unit[0]][i]
            unit_gamma_i[i] = unit_gamma_i[i] + 2*lam*gamma_i[item_unit][i]
        d_gamma_i[item_unit] = unit_gamma_i
    res.append(d_alpha)
    res.append(d_beta_u)
    res.append(d_beta_i)
    res.append(d_gamma_u)
    res.append(d_gamma_i)
    return res
```

In [71]:

```
# update parameters for each iterations
def updateParameter(d_res, lr, alpha, beta_u, beta_i, gamma_u, gamma_i):
    coff = []
    alpha = alpha - lr*d_res[0]
    for key_u in beta_u:
        beta_u[key_u] = beta_u[key_u] - lr*d_res[1][key_u]
    for key_i in beta_i:
        beta_i[key_i] = beta_i[key_i] - lr*d_res[2][key_i]
    for user_unit in gamma_u:
        for i in range(len(gamma_u[user_unit])):
            gamma_u[user_unit][i] = gamma_u[user_unit][i] - lr*d_res[3][user_u
nit][i]
    for item_unit in gamma_i:
        for i in range(len(gamma_i[item_unit])):
            gamma_i[item_unit][i] = gamma_i[item_unit][i] - lr*d_res[4][item_u
nit][i]
    coff.append(alpha)
    coff.append(beta_u)
    coff.append(beta_i)
    coff.append(gamma_u)
    coff.append(gamma_i)
    return coff
```

In [120]:

```
# initialize parameters
# Initial beta_u and beta_i
coff = []
lam = 6.6
lr = 0.00166
beta_u = defaultdict(int)
beta_i = defaultdict(int)
alpha = 0
gamma_u = defaultdict(list)
gamma_i = defaultdict(list)
for key_u in user:
    beta_u[key_u] = 0
    gamma_u[key_u] = np.random.randn(1).tolist()
for key_i in item:
    beta_i[key_i] = 0
    gamma_i[key_i] = np.random.randn(1).tolist()
coff.append(alpha)
coff.append(beta_u)
coff.append(beta_i)
coff.append(gamma_u)
coff.append(gamma_i)
```

In [112]:

```
# training process
def train(iteration_num, coff, lam, lr, U_I_R, user, item):
    res_list = []
    for i in range(iteration_num):
        res = ObjectFunction(coff[0], coff[1], coff[2], coff[3], coff[4], lam,
U_I_R, user, item)
        if (res < 100000):
            lr = lr*0.99
            # learning rate decay
        if i%10 == 0:
            print (f"MSE = {res}")
            res_list.append(res)
        d_res = ObjectPrime(coff[0], coff[1], coff[2], coff[3], coff[4], lam,
U_I_R, user, item)
        coff = updateParameter(d_res, lr, coff[0], coff[1], coff[2], coff[3],
coff[4])
    number_of_samples = range(0, int(iteration_num/10))
    plt.title('Recommendation System on latent factor model')
    plt.xlabel('Number of samples')
    plt.ylabel('MSE error')
    plt.plot(number_of_samples, res_list, color='blue')
    plt.show()
    return coff
```

In [1]:

```
# coff = train(800, coff, lam, lr, U_I_R, user, item)
```

In [122]:

```
print (coff[0])
```

```
4.509273496281763
```

In [81]:

```
def computeMSE(coff, validation_set):
    res = []
    real_label = []
    for elem in validation_set:
        if elem['reviewerID'] in coff[1] and elem['itemID'] in coff[2]:
            unit_res = coff[0] + coff[1][elem['reviewerID']] + coff[2][elem['i
temID']] + inner_multiply(coff[3][elem['reviewerID']], coff[4][elem['itemID']]
)
            res.append(unit_res)
            real_label.append(elem['rating'])
    return sum([abs(res[i] - real_label[i])**2 for i in range(len(res))])/len(
res)
```

In [124]:

```
predictions = open("predictions_Rating4.txt", 'w')
for l in open("/Users/linbinyang/Downloads/assignment1/pairs_Rating.txt"):
    if l.startswith("reviewerID"):
        predictions.write(l)
        continue
    u,i = l.strip().split('-')
    if u in coff[1] and i in coff[2]:
        predictions_n = coff[0] + coff[1][u] + coff[2][i] + inner_multiply(cof
f[3][u], coff[4][i])
    else:
        predictions_n = coff[0]
    if predictions_n > 5:
        predictions.write(u + '-' + i + ',' + str(5) + '\n')
    else:
        predictions.write(u + '-' + i + ',' + str(predictions_n) + '\n')
predictions.close()
```

In [123]:

```
print (computeMSE(coff, validation_set))
```

```
1.1752082586198969
```