In [1]:
```
import gzip
import math
import scipy.optimize
import numpy
import string
import random
import csv
import matplotlib.pyplot as plt
from collections import defaultdict
```

In [2]:
```
# read csv from zip file
def readGz(path):
    for l in gzip.open(path, 'rt'):
        yield eval(l)

def readCSV(path):
    f = gzip.open(path, 'rt')
    c = csv.reader(f)
    header = next(c)
    for l in c:
        d = dict(zip(header,l))
        yield d
```

In [3]:
```
# produce the dataset
dataset = list(readCSV("trainInteractions.csv.gz"))
dataset[0]
```

Out[3]:
```
{'user_id': '88348277',
 'recipe_id': '03969194',
 'date': '2004-12-23',
 'rating': '5'}
```

In [47]:
```
# split the dataset
train = dataset[:400000]
validation = dataset[400000:]
```

Task (Cook/Make prediction)

Question 1

Although we have built a validation set, it only consists of positive samples. For this task we also need examples of user/item pairs corresponding to recipes that weren't cooked. For each entry (user,recipe) in the validation set, sample a negative entry by randomly choosing a recipe that user hasn't cooked.1 Evaluate the performance (accuracy) of the baseline model on the validation set you have built (1 mark).

In [5]:
```
recipes_per_user = defaultdict(set)
users_per_recipe = defaultdict(set)
recipes = set()

for d in dataset:
    user, recipe = d['user_id'], d['recipe_id']
    recipes_per_user[user].add(recipe)
    users_per_recipe[recipe].add(user)
    recipes.add(recipe)
```

In [6]:
```
validation[0]
```

Out[6]:
```
{'user_id': '90764166',
 'recipe_id': '01768679',
 'date': '2011-09-10',
 'rating': '5'}
```

In [7]:
```
# build new validation set
from tqdm.notebook import tqdm
neg_validation = list()
for v in tqdm(validation):
    v['cooked'] = 1
    random_recipe = random.sample(recipes - recipes_per_user[v['use
    neg_v = ({'user_id': v['user_id'], 'recipe_id': random_recipe,
    neg_validation.append(neg_v)
```

100% 100000/100000 [29:15<00:00, 56.98it/s]

In [48]:
```
for neg_v in neg_validation:
    validation.append(neg_v)
```

In [49]:
```
# check if the new validation set have 200,000 lines
len(validation)
```

Out[49]: 200000

In [10]:
```
def baseline(train, validation, threshold):
    recipe_count = defaultdict(int)
    total_cooked = 0

    for t in train:
        recipe_count[t['recipe_id']] += 1
        total_cooked += 1

    most_pop = [(recipe_count[x], x) for x in recipe_count]
    most_pop.sort()
    most_pop.reverse()

    correct = 0
    return1 = set()
    count = 0

    for ic, i in most_pop:
        count += ic
        return1.add(i)
        if count > total_cooked/threshold:
            break


    for v in validation:
        if v['recipe_id'] in return1:
            correct += (v['cooked'] != 0)
        else:
            correct += (v['cooked'] == 0)

    acc = (correct/len(validation))
    return acc
```

In [11]:
```
print("the accuracy of the baseline model on the new validation set
```

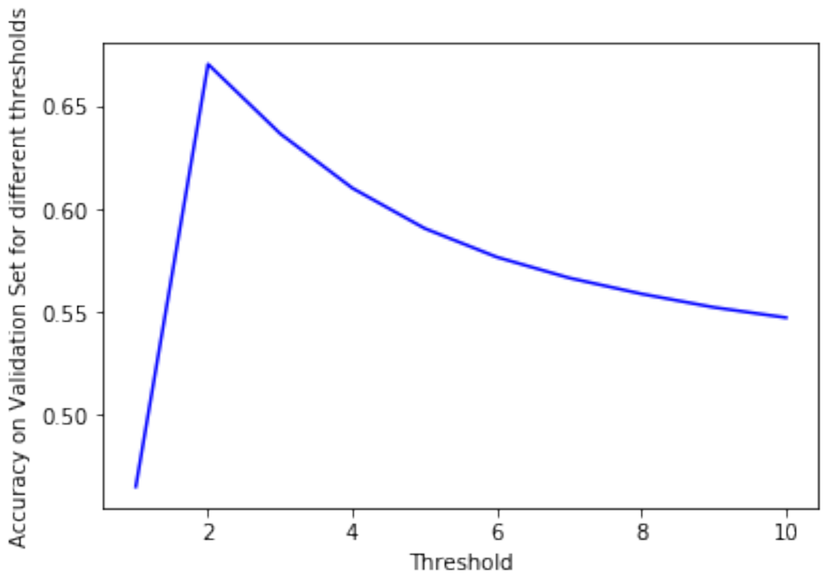
```
the accuracy of the baseline model on the new validation set is:
0.670285
```

Question2

The existing 'made/cooked prediction' baseline just returns True if the item in question is 'popular,' using a threshold of the 50th percentile of popularity (totalCooked/2). Assuming that the 'non-made' test examples are a random sample of user-recipe pairs, this threshold may not be the best one. See if you can find a better threshold and report its performance on your validation set (1 mark).

In [12]:
```
def pipline(thresholds):
    accuarcy = []
    for threshold in thresholds:
        accuarcy.append(baseline(train, validation, threshold))

    plt.plot(thresholds, accuarcy, 'b-')
    plt.xlabel('Threshold')
    plt.ylabel('Accuracy on Validation Set for different thresholds
    plt.show()
    print('When threshold is %.3f has the best accuracy %.3f' % (th
```

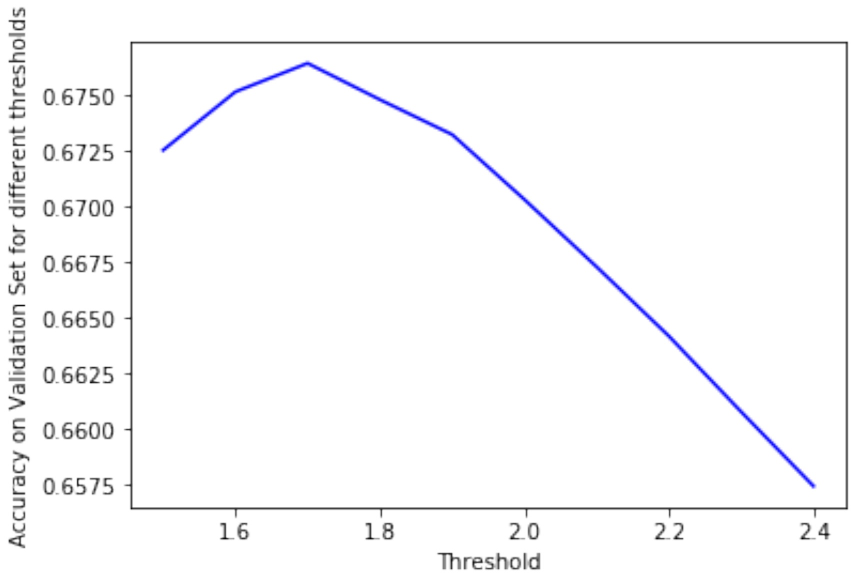
In [50]:
```
# find the best accuarcy when threshold from 1 to 10
thresholds = [i for i in range(1, 11)]
pipline(thresholds)
```




```
When threshold is 2.000 has the best accuracy 0.670
```

We can find that when threshold is around 2 has the best accuracy. Therefore, we find the best accuracy between 1.5 to 2.5

In [13]:
```
thresholds = [i/10 for i in range(15, 25)]
pipline(thresholds)
```




```
When threshold is 1.700 has the best accuracy 0.676
```

Question 3

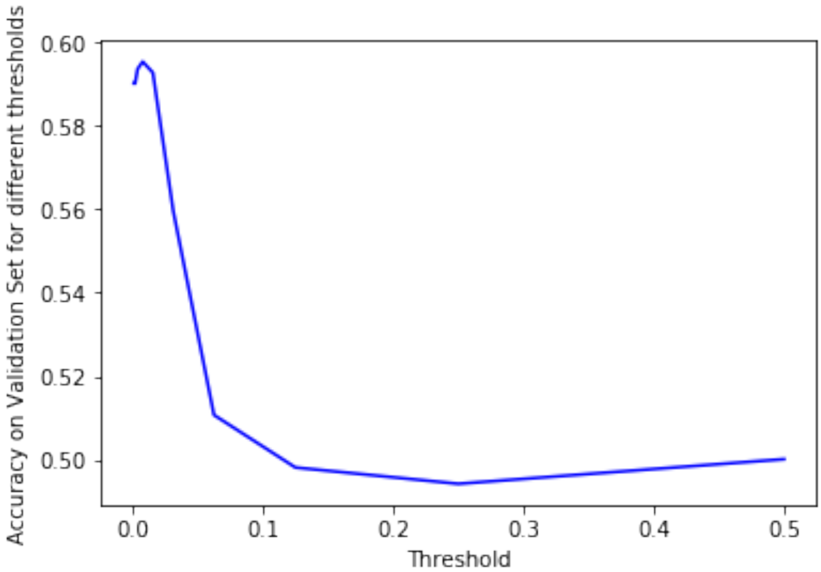
An alternate baseline than the one provided might make use of the Jaccard similarity (or another similarity metric). Given a pair (u, g) in the validation set, consider all training items g' that user u has cooked. For each, compute the Jaccard similarity between g and g', i.e., users (in the training set) who have made g and users who have made g'. Predict as 'made' if the maximum of these Jaccard similarities exceeds a threshold (you may choose the threshold that works best). Report the performance on your validation set (1 mark).

In [14]:
```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    if denom == 0:
        return 0
    return numer / denom
```

In [15]:
```
def findJaccard(train, validation):
    t_user_recipes = defaultdict(set)
    t_recipe_users = defaultdict(set)
    for t in train:
        user, recipe = t['user_id'], t['recipe_id']
        t_user_recipes[user].add(recipe)
        t_recipe_users[recipe].add(user)

    thresholds = [1/2**i for i in range(1, 11)]
    acc = []

    for threshold in thresholds:
        print('Evaluating on threshold %.3f ...' % threshold)
        correct = 0
        for v in validation:
            v_user, v_recipe = v['user_id'], v['recipe_id']
            jac = 0
            if v_user in t_user_recipes:
                user_cook = t_user_recipes[v_user]
                for recipe in user_cook:
                    if v_recipe in t_recipe_users:
                        temp = Jaccard(t_recipe_users[v_recipe], t_
                        if jac <= temp:
                            jac = temp

            if jac > threshold:
                correct += (v['cooked'] != 0)
            else:
                correct += (v['cooked'] == 0)
        acc1 = correct/len(validation)
        acc.append(acc1)

    plt.plot(thresholds, acc, 'b-')
    plt.xlabel('Threshold')
    plt.ylabel('Accuracy on Validation Set for different thresholds
    plt.show()
    print('When threshold is %.3f has the best accuracy %.3f' % (th
```

In [16]:
```
findJaccard(train, validation)
```

```
Evaluating on threshold 0.500 ...
Evaluating on threshold 0.250 ...
Evaluating on threshold 0.125 ...
Evaluating on threshold 0.062 ...
Evaluating on threshold 0.031 ...
Evaluating on threshold 0.016 ...
Evaluating on threshold 0.008 ...
Evaluating on threshold 0.004 ...
Evaluating on threshold 0.002 ...
Evaluating on threshold 0.001 ...
```




```
When threshold is 0.008 has the best accuracy 0.595
```

Question 4

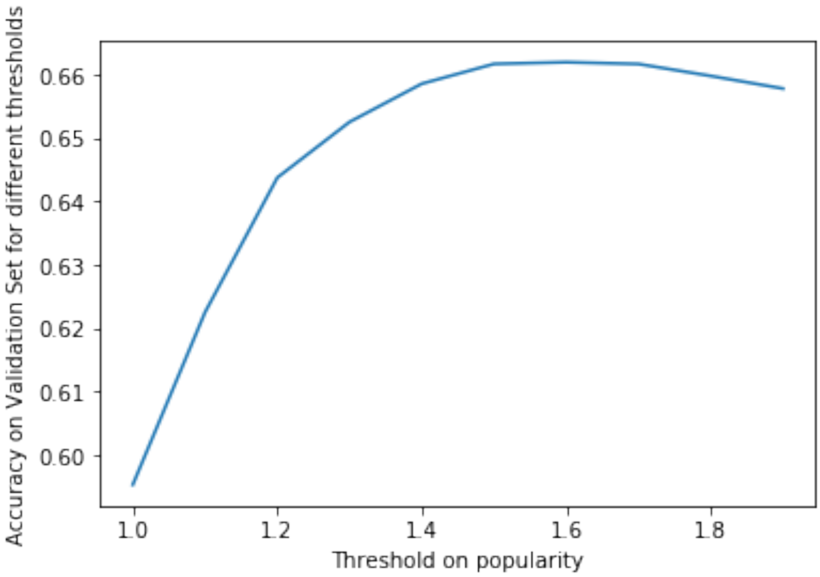
In [17]:
```
def ensemble(threshold_pop = 1.7, threshold_jac = 0.008):

    t_user_recipes = defaultdict(set)
    t_recipe_users = defaultdict(set)
    for t in train:
        user, recipe = t['user_id'], t['recipe_id']
        t_user_recipes[user].add(recipe)
        t_recipe_users[recipe].add(user)

    recipe_count = defaultdict(int)
    total_cooked = 0

    for t in train:
        recipe_count[t['recipe_id']] += 1
        total_cooked += 1

    most_pop = [(recipe_count[x], x) for x in recipe_count]
    most_pop.sort()
    most_pop.reverse()

    return1 = set()
    count = 0

    for ic, i in most_pop:
        count += ic
        return1.add(i)
        if count > total_cooked/threshold_pop:
            break

    correct = 0
    for v in validation:
        v_user, v_recipe = v['user_id'], v['recipe_id']
        jac = 0
        if v_user in t_user_recipes:
            user_cook = t_user_recipes[v_user]
            for recipe in user_cook:
                if v_recipe in t_recipe_users:
                    temp = Jaccard(t_recipe_users[v_recipe], t_reci
                    if jac <= temp:
                        jac = temp

        if jac > threshold_jac and v_recipe in return1:
            correct += (v['cooked'] != 0)
        else:
            correct += (v['cooked'] == 0)

    return correct/len(validation)
```

In [18]:
```
print("Accuracy is %.3f when use popular threshold and jaccard thre
```

```
Accuracy is 0.662 when use popular threshold and jaccard threshold
```

Tune on popularity

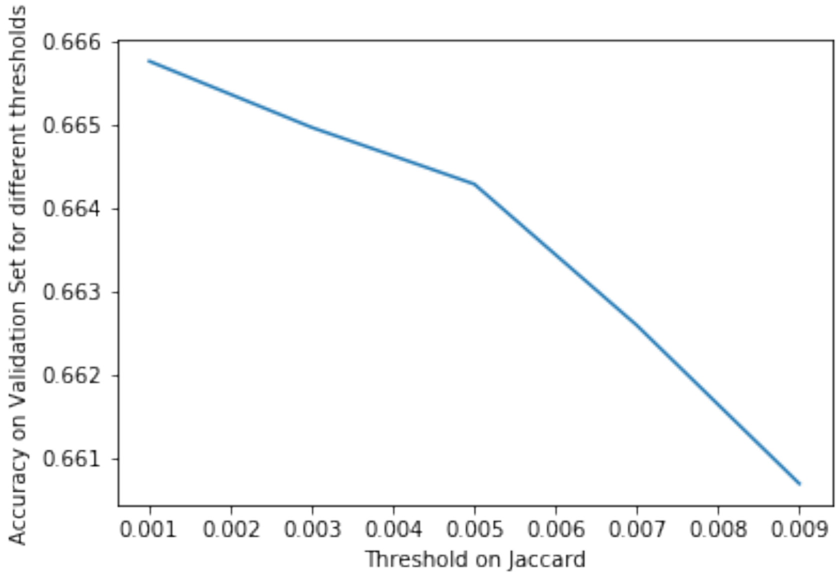
In [19]:
```
threshold_pops = [i/10 for i in range(10, 20)]
acc = []
for threshold_pop in threshold_pops:
    acc.append(ensemble(threshold_pop, threshold_jac = 0.008))

plt.plot(threshold_pops, acc)
plt.xlabel('Threshold on popularity')
plt.ylabel('Accuracy on Validation Set for different thresholds')
plt.show()

print('When threshold is %.3f has the best accuracy %.3f' % (thresh
```




```
When threshold is 1.600 has the best accuracy 0.662
```

Tune on Jaccard

In [20]:
```
threshold_jacs = [i/10000 for i in range(10, 110, 20)]
acc = []
for threshold_jac in threshold_jacs:
    acc.append(ensemble(1.7, threshold_jac))

plt.plot(threshold_jacs, acc)
plt.xlabel('Threshold on Jaccard')
plt.ylabel('Accuracy on Validation Set for different thresholds')
plt.show()
print('When threshold is %.3f has the best accuracy %.3f' % (thresh
```




```
When threshold is 0.001 has the best accuracy 0.666
```

Therefore, we can find when popular threshold is 1.6 and Jaccard similarity threshold is 0.001, we can have the best accuracy, which is 0.666.

In [21]:
```
print("Accuracy is %.3f when use popular threshold and jaccard thre
```

```
Accuracy is 0.666 when use popular threshold and jaccard threshold
```

Question 5

In [22]:
```
len(train)
```

Out[22]:
```
400000
```

In [23]:
```
len(validation)
```

Out[23]: 200000

In [24]:
```
threshold_pop = 1.7
threshold_jac = 0.005

t_user_recipes = defaultdict(set)
t_recipe_users = defaultdict(set)
for t in train:
    user, recipe = t['user_id'], t['recipe_id']
    t_user_recipes[user].add(recipe)
    t_recipe_users[recipe].add(user)

recipe_count = defaultdict(int)
total_cooked = 0

for t in train:
    recipe_count[t['recipe_id']] += 1
    total_cooked += 1

most_pop = [(recipe_count[x], x) for x in recipe_count]
most_pop.sort()
most_pop.reverse()

return1 = set()
count = 0

for ic, i in most_pop:
    count += ic
    return1.add(i)
    if count > total_cooked/threshold_pop:
        break

predictions = open("predictions_Made.txt", 'w')
for l in open("stub_Made.txt"):
    if l.startswith("user_id"):
        #header
        predictions.write(l)
        continue


    u,i = l.strip().split('-')
    jac = 0
    if u in t_user_recipes:
        user_cook = t_user_recipes[u]
        for recipe in user_cook:
            if i in t_recipe_users:
                temp = Jaccard(t_recipe_users[i], t_recipe_users[re
                if jac <= temp:
                    jac = temp
```

```
        if i in return1 and jac > threshold_jac:
            predictions.write(u + '-' + i + ",1\n")
        else:
            predictions.write(u + '-' + i + ",0\n")
predictions.close()
```

Submission has been submitted on Kaggle.
My Kaggle id is: 2AM_official
My accuracy for submitting on Kaggle is: 0.69740

Tasks (Rating prediction)

Quesiton 9

In [25]:
```
# use Gradient Descent to make prediction
import pandas as pd
import scipy
import scipy.optimize
import numpy as np

def splitDataset(datapath):
    f = gzip.open(datapath, 'rt')
    data = pd.read_csv(f)
    # in this task, I prefer to user 450000 as the train set and 50
    train, valid = data[:400000], data[400000:]
    return data, train, valid
```

In [26]:
```
data, train, valid = splitDataset("trainInteractions.csv.gz")
```

In [27]:
```
def prediction(user, item):
    return alpha + user_biases[user] + item_biases[item]
```

In [28]:
```
def MSE(predictions, labels):
    differences = [(x - y) ** 2 for x, y in zip(predictions,labels)
    return sum(differences) / len(differences)
```

In [29]:
```
def unpack(theta):
    global alpha
    global user_biases
    global item_biases
    alpha = theta[0]
    user_biases = dict(zip(users, theta[1:n_users+1]))
    item_biases = dict(zip(items, theta[1+n_users:]))
```

In [30]:
```
def cost(theta, labels, lamb):
    unpack(theta)
    predictions = [prediction(d['user_id'], d['recipe_id']) for ind
    cost = MSE(predictions, labels)
    print("MSE = " + str(cost))
    for u in user_biases:
        cost += lamb * user_biases[u]**2
    for i in item_biases:
        cost += lamb * item_biases[i]**2
    return cost
```

In [31]:
```
def derivative(theta, labels, lamb):
    unpack(theta)
    N = len(train)
    d_alpha = 0
    d_user_biases = defaultdict(float)
    d_item_biases = defaultdict(float)
    for index, d in train.iterrows():
        u,i = d['user_id'], d['recipe_id']
        pred = prediction(u, i)
        diff = pred - d['rating']
        d_alpha += 2/N * diff
        d_user_biases[u] += 2/N * diff
        d_item_biases[i] += 2/N * diff
    for u in user_biases:
        d_user_biases[u] += 2 * lamb * user_biases[u]
    for i in item_biases:
        d_item_biases[i] += 2 * lamb * item_biases[i]
    d_theta = [d_alpha] + [d_user_biases[u] for u in users] + [d_it
    return np.array(d_theta)
```

In [32]:
```
len(train)
```

Out[32]:
```
400000
```

In [33]:
```
# when lambda is 1
lamb = 1

rating_mean = train['rating'].mean()
alpha = rating_mean

labels = train['rating']

user_biases = defaultdict(float)
item_biases = defaultdict(float)

users = list(set(train['user_id']))
items = list(set(train['recipe_id']))
n_users = len(users)
n_items = len(items)

scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + [0.0]*(n_users+n_items
                             derivative, args = (labels, lamb))
```

```
MSE = 0.8987313599958769
MSE = 0.8856358581692616
MSE = 0.8985952813610849
MSE = 0.8985952329948594
```

Out[33]:
```
(array([4.58067734e+00, 3.97387833e-05, 1.04820554e-05, ...,
        6.34873333e-07, 2.09635790e-06, 2.09675329e-06]),
 0.8986631878143104,
 {'grad': array([ 5.03931794e-07, -1.74043442e-08, -1.52204830e-09
, ...,
         -1.59579119e-09, -3.03656101e-10, -3.09086612e-10]),
  'task': b'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL',
  'funcalls': 4,
  'nit': 2,
  'warnflag': 0})
```

In [34]:
```
len(valid)
```

Out[34]:
```
100000
```

In [35]:
```
# Validating
predictions = []
for index, d in valid.iterrows():
    u, i = d['user_id'], d['recipe_id']
    if u in user_biases and i in item_biases:
        predictions.append(prediction(u, i))
    elif u in user_biases:
        predictions.append(alpha + user_biases[u])
    elif u in user_biases:
        predictions.append(alpha + item_biases[i])
    else:
        predictions.append(rating_mean)

print("MSE on validation set is %.3f" % MSE(predictions, valid['rat
```

```
MSE on validation set is 0.909
```

Question 10

In [36]:
```
print("max user: %s , max value: %f" % (max(user_biases, key = user
print("max book: %s , max value: %f" % (max(item_biases, key = item
print("min user: %s , min value: %f" % (min(user_biases, key = user
print("min book: %s , min value: %f" % (min(item_biases, key = item
```

```
max user: 32445558 , max value: 0.003670
max book: 98124873 , max value: 0.000209
min user: 70705426 , min value: -0.001295
min book: 29147042 , min value: -0.000285
```

Question 11

In [43]:
```
# change lambda to find the best accuracy
lamb = 10**(-3)

rating_mean = train['rating'].mean()
alpha = rating_mean

labels = train['rating']

user_biases = defaultdict(float)
item_biases = defaultdict(float)

users = list(set(train['user_id']))
items = list(set(train['recipe_id']))
n_users = len(users)
n_items = len(items)

scipy.optimize.fmin_l_bfgs_b(cost, [alpha] + [0.0]*(n_users+n_items
                             derivative, args = (labels, lamb))
```

```
MSE = 0.8987313599958769
MSE = 0.8856358581692616
MSE = 1.0654370778402464
MSE = 0.8838486849225675
MSE = 0.8783950770605176
MSE = 0.8774757265689982
MSE = 0.8739908659816205
MSE = 0.8608684471037029
MSE = 0.8570174630119913
MSE = 0.8538133491814515
MSE = 0.8536437933444463
MSE = 0.8537540776882084
MSE = 0.85373239822389
MSE = 0.8536762891686595
MSE = 0.8536408279443795
MSE = 0.8536434992235896
MSE = 0.8536432119023515
MSE = 0.8536518992124711
MSE = 0.8536433250721075
```

Out[43]:
```
(array([ 4.54414380e+00,  3.90084044e-02,  1.09833262e-02, ...,
        -1.43174176e-04,  2.16969659e-03,  2.38693740e-03]),
 0.8704667939165373,
 {'grad': array([-2.98698808e-06,  6.52551061e-08, -8.98493329e-09
, ...,
          2.15544371e-09, -1.01221827e-09, -4.68002195e-09]),
  'task': b'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL',
  'funcalls': 19,
  'nit': 15,
  'warnflag': 0})
```

In [44]:
```
predictions = []
for index, d in valid.iterrows():
    u, i = d['user_id'], d['recipe_id']
    if u in user_biases and i in item_biases:
        predictions.append(prediction(u, i))
    elif u in user_biases:
        predictions.append(alpha + user_biases[u])
    elif u in user_biases:
        predictions.append(alpha + item_biases[i])
    else:
        predictions.append(rating_mean)

print("MSE on validation set is %.3f" % MSE(predictions, valid['rat
```

```
MSE on validation set is 0.873
```

In [45]:
```
predictions = open("predictions_Rated.txt", 'w')
for l in open("stub_Rated.txt"):
    if l.startswith("user_id"):
    #header
        predictions.write(l)
        continue

    u, i = l.strip().split('-')

    if int(u) in user_biases and int(i) in item_biases:
        predictions.write(u + '-' + i + ',' + str(prediction(int(u)
    elif int(u) in user_biases:
        predictions.write(u + '-' + i + ',' + str(alpha + user_bias
    elif int(i) in item_biases:
        predictions.write(u + '-' + i + ',' + str(alpha + item_bias
    else:
        predictions.write(u + '-' + i + ',' + str(alpha) + '\n')

predictions.close()
```

The test data is uploaded to Kaggle.
My Kaggle id is: 2AM_official
My Score is: 0.88260

In [ ]: