In [1]:

```
import gzip
from collections import defaultdict
from sklearn import linear_model
import csv
import json
```

In [2]:

```
import numpy as np
import random
```

In [3]:

```
def readGz(path):
    for l in gzip.open(path, 'rt'):
        yield eval(l)
```

In [55]:

```
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['user_id'],d['recipe_id'],d
```

In [5]:

```
def loadCSV(path):
    f = gzip.open(path, 'rt')
    c = csv.reader(f)
    header = next(c)
    dataset = []
    for line in c:
        d = dict(zip(header, line))
        dataset.append(d)
    return dataset
```

Q1)

In [6]:

```
dataset = loadCSV('trainInteractions.csv.gz')
```

In [7]:

```
recipeCount = defaultdict(int)
totalCooked = 0

for user,recipe,d in readCSV("trainInteractions.csv.gz"):
    print(d)
    recipeCount[recipe] += 1
    totalCooked += 1
    break
```

```
{'user_id': '88348277', 'recipe_id': '03969194', 'date': '2004-12-23', 'rati
ng': '5'}
```

In [7]:

```
len(dataset)
```

Out[7]:

```
500000
```

In [22]:

```
train = dataset[:400000]
validation = dataset[400000:]
```

In [23]:

```
userPerRecipeValid = defaultdict(set)
recipePerUserValid = defaultdict(set)
recipeListValid = set([])
newValidationDataset = []

for datum in validation:
    user, recipe = datum['user_id'], datum['recipe_id']
    userPerRecipeValid[recipe].add(user)
    recipePerUserValid[user].add(recipe)
    recipeListValid.add(recipe)

recipeListValid = list(recipeListValid)
recipeListSize = len(recipeListValid)
for datum in validation:
    user, recipe = datum['user_id'], datum['recipe_id']
    newValidationDataset.append((user, recipe, 1))

    while True:
        index = random.randint(0, recipeListSize - 1)
        if recipeListValid[index] not in recipePerUserValid[user]:
            break

    newValidationDataset.append((user, recipeListValid[index], 0))
```

In [25]:

```
len(newValidationDataset)
```

Out[25]:

```
200000
```

In [26]:

```
newValidationDataset[:10]
```

Out[26]:

```
[('90764166', '01768679', 1),
 ('90764166', '44845944', 0),
 ('68112239', '24923981', 1),
 ('68112239', '64197312', 0),
 ('32173358', '57597698', 1),
 ('32173358', '58861769', 0),
 ('30893740', '16266088', 1),
 ('30893740', '78236641', 0),
 ('69780905', '62953151', 1),
 ('69780905', '28584315', 0)]
```

In [35]:

```
### Would-cook baseline: just rank which recipes are popular and which are not, and return

recipeCount = defaultdict(int)
totalCooked = 0

for d in train:
    user, recipe = d['user_id'], d['recipe_id']
    recipeCount[recipe] += 1
    totalCooked += 1

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

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

correct_labels = 0
total_labels = 0
for d in newValidationDataset:
    user, recipe, label = d
    if recipe in return1:
        prediction = 1
    else:
        prediction = 0

    if prediction == label:
        correct_labels += 1
    total_labels += 1

accuracy = correct_labels/total_labels
```

In [36]:

```
accuracy
```

Out[36]:

```
0.61539
```

Q2)

In [50]:

```
thresholds = [0.1, 0.2, 0.3, 0.4, 0.43, 0.45, 0.48, 0.49, 0.5, 0.51, 0.52, 0.53, 0.55, 0.6,
```

In [51]:

```
### Would-cook baseline: just rank which recipes are popular and which are not, and return

recipeCount = defaultdict(int)
totalCooked = 0
best_accuracy = 0
better_threshold = 0

for d in train:
    user, recipe = d['user_id'], d['recipe_id']
    recipeCount[recipe] += 1
    totalCooked += 1

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

for threshold in thresholds:
    return1 = set()
    count = 0
    for ic, i in mostPopular:
        count += ic
        return1.add(i)
        if count > (totalCooked * threshold):
            break

    correct_labels = 0
    total_labels = 0
    for d in newValidationDataset:
        user, recipe, label = d
        if recipe in return1:
            prediction = 1
        else:
            prediction = 0

        if prediction == label:
            correct_labels += 1
        total_labels += 1

    accuracy = correct_labels/total_labels
    print(threshold, accuracy)
    if best_accuracy < accuracy:
        best_accuracy = accuracy
        better_threshold = threshold
```

```
0.1 0.54484
0.2 0.580665
0.3 0.603475
0.4 0.613305
0.43 0.61458
0.45 0.614885
0.48 0.61494
0.49 0.614555
0.5 0.61443
0.51 0.613975
0.52 0.61404
```

```
0.53 0.613185
0.55 0.61228
0.6 0.60929
0.7 0.599965
0.8 0.58739
0.9 0.570195
```

In [52]:

```
print(best_accuracy)
print(better_threshold)
```

```
0.61494
0.48
```

Q3)

In [37]:

```
userPerRecipeTrain = defaultdict(set)
recipePerUserTrain = defaultdict(set)

for datum in train:
    user, recipe = datum['user_id'], datum['recipe_id']
    userPerRecipeTrain[recipe].add(user)
    recipePerUserTrain[user].add(recipe)
```

In [38]:

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

In [39]:

```
def mostSimilar(recipe, N):
    similarities = []
    users = userPerRecipeTrain[recipe]
    for i2 in userPerRecipeTrain:
        if i2 == i: continue
        sim = Jaccard(users, userPerRecipeTrain[i2])

        similarities.append((sim,i2))
    similarities.sort(key=lambda x: x[0], reverse=True)

    return similarities[:N]
```

In [44]:

```
sim_thresholds = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95]
```

In [27]:

```
newValidationDataset[:2]
```

Out[27]:

```
[('90764166', '01768679', 1), ('90764166', '44845944', 0)]
```

In [45]:

```
best_accuracy = 0
best_threshold = 0
for sim_threshold in sim_thresholds:
    correct_labels = 0
    total_labels = 0
    for d in newValidationDataset:
        user, recipe, label = d

        prediction = 0
        for userRecipe in recipePerUserTrain[user]:
            if Jaccard(userPerRecipeTrain[recipe], userPerRecipeTrain[userRecipe]) > sim_th
                prediction = 1
                break

        if prediction == label:
            correct_labels += 1
        total_labels += 1

    accuracy = correct_labels/total_labels
    print(sim_threshold, accuracy)
    if best_accuracy < accuracy:
        best_accuracy = accuracy
        best_threshold = sim_threshold

print(best_accuracy, best_threshold)
```

```
0.0 0.596115
0.1 0.521945
0.2 0.514155
0.3 0.5076
0.4 0.502215
0.5 0.498775
0.6 0.49875
0.7 0.49865
0.8 0.498645
0.9 0.498645
0.95 0.498645
0.596115 0.0
```

Q4)

In [46]:

```
popularity_threshold = 0.48
sim_threshold = 0.1
```

In [51]:

```
### Would-cook baseline: just rank which recipes are popular and which are not, and return

recipeCount = defaultdict(int)
totalCooked = 0
best_accuracy = 0
better_threshold = 0

for d in train:
    user, recipe = d['user_id'], d['recipe_id']
    recipeCount[recipe] += 1
    totalCooked += 1

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


return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > (totalCooked * popularity_threshold):
        break

correct_labels = 0
total_labels = 0
for d in newValidationDataset:
    user, recipe, label = d

    prediction = 0
    if recipe in return1:
        prediction = 1
    else:
        for userRecipe in recipePerUserTrain[user]:
            if Jaccard(userPerRecipeTrain[recipe], userPerRecipeTrain[userRecipe]) > sim_th
                prediction = 1
                break

    if prediction == label:
        correct_labels += 1
    total_labels += 1

accuracy = correct_labels/total_labels
```

In [48]:

```
print(accuracy)
```

```
0.62531
```

Q5)

In [50]:

```python
### Would-cook baseline: just rank which recipes are popular and which are not, and return

recipeCount = defaultdict(int)
totalCooked = 0
best_accuracy = 0
better_threshold = 0

for d in train:
    user, recipe = d['user_id'], d['recipe_id']
    recipeCount[recipe] += 1
    totalCooked += 1

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


return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > (totalCooked * popularity_threshold):
        break

correct_labels = 0
total_labels = 0

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

    prediction = 0
    if recipe in return1:
        prediction = 1
    else:
        for userRecipe in recipePerUserTrain[user]:
            if Jaccard(userPerRecipeTrain[recipe], userPerRecipeTrain[userRecipe]) > sim_th
                prediction = 1
                break
    predictions.write(user + '-' + recipe + ',' + str(prediction) + '\n')

predictions.close()
```

Username: vktiwari33

Q9)

In [151]:

```
import numpy
```

In [124]:

```
import scipy
import scipy.optimize
```

In [128]:

```
dataset = []
```

In [129]:

```
for user,recipe,d in readCSV("trainInteractions.csv.gz"):
    d['rating'] = int(d['rating'])
    dataset.append(d)
```

In [130]:

```
train = dataset[:400000]
valid = dataset[400000:]
```

In [131]:

```
reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)
```

In [132]:

```
for d in train:
    user,item = d['user_id'], d['recipe_id']
    reviewsPerUser[user].append(d)
    reviewsPerItem[item].append(d)
```

In [133]:

```
ratingMean = sum([d['rating'] for d in train]) / len(train)
```

In [143]:

```
labels = [d['rating'] for d in train]
```

In [134]:

```
N = len(train)
nUsers = len(reviewsPerUser)
nItems = len(reviewsPerItem)
users = list(reviewsPerUser.keys())
items = list(reviewsPerItem.keys())
```

In [135]:

```
alpha = ratingMean
```

In [136]:

```
userBiases = defaultdict(float)
itemBiases = defaultdict(float)
```

In [137]:

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

In [157]:

```
def prediction(user, item):
    if user not in userBiases or item not in itemBiases:
        return alpha
    return alpha + userBiases[user] + itemBiases[item]
```

In [139]:

```
def unpack(theta):
    global alpha
    global userBiases
    global itemBiases
    alpha = theta[0]
    userBiases = dict(zip(users, theta[1:nUsers+1]))
    itemBiases = dict(zip(items, theta[1+nUsers:]))
```

In [164]:

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

In [165]:

```
def derivative(theta, labels, dataset, lamb):
    unpack(theta)
    N = len(dataset)
    dalpha = 0
    dUserBiases = defaultdict(float)
    dItemBiases = defaultdict(float)
    for d in dataset:
        u,i = d['user_id'], d['recipe_id']
        pred = prediction(u, i)
        diff = pred - d['rating']
        dalpha += 2/N*diff
        dUserBiases[u] += 2/N*diff
        dItemBiases[i] += 2/N*diff
    for u in userBiases:
        dUserBiases[u] += 2*lamb*userBiases[u]
    for i in itemBiases:
        dItemBiases[i] += 2*lamb*itemBiases[i]
    dtheta = [dalpha] + [dUserBiases[u] for u in users] + [dItemBiases[i] for i in items]
    return numpy.array(dtheta)
```

In [184]:

```
scipy.optimize.fmin_l_bfgs_b(cost,
                             [alpha] + [0.0]*(nUsers+nItems),
                             derivative,
                             args = (labels, train, 1))
```

```
MSE = 0.898807042703075
MSE = 1.4092942879265038
MSE = 0.8985950192911197
MSE = 0.8985951779117449
```

Out[184]:

```
(array([ 4.58067353e+00, -8.58146680e-05, -8.06156759e-06, ...,
        -1.45132190e-06,  1.04630213e-06, -1.45114646e-06]),
 0.8986631878445489,
 {'grad': array([-6.61797442e-06,  2.22831645e-07,  6.45420112e-09, ...,
          6.22533785e-10,  9.15110290e-10,  6.22405709e-10]),
  'task': 'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL',
  'funcalls': 4,
  'nit': 2,
  'warnflag': 0})
```

In [167]:

```
y = []
y_pred = []
for d in valid:
    y.append(d['rating'])
    y_pred.append(prediction(d['user_id'], d['recipe_id']))

print(MSE(y_pred, y))
```

```
0.9094423694728887
```

Q10)

In [168]:

```
largestBu = -100000
largestBuUser = None
smallestBu = 100000
smallestBuUser = None

for u in userBiases:
    if largestBu < userBiases[u]:
        largestBu = userBiases[u]
        largestBuUser = u
    if smallestBu > userBiases[u]:
        smallestBu = userBiases[u]
        smallestBuUser = u
```

In [169]:

```
largestBi = -100000
largestBiItem = None
smallestBi = 100000
smallestBiItem = None

for i in itemBiases:
    if largestBi < itemBiases[i]:
        largestBi = itemBiases[i]
        largestBiItem = i
    if smallestBi > itemBiases[i]:
        smallestBi = itemBiases[i]
        smallestBiItem = i
```

In [170]:

```
print(largestBuUser, smallestBuUser)
```

```
32445558 70705426
```

In [171]:

```
print(largestBiItem, smallestBiItem)
```

```
98124873 29147042
```

Q11)

In [172]:

```
lambdas = [0.1, 0.3, 0.5, 0.7, 0.9, 1.1, 1.2]
```

In [173]:

```
min_mse = 100000
opt_lamb = None
for lamb in lambdas:
    _, mse, _ = scipy.optimize.fmin_l_bfgs_b(cost,
                                 [alpha] + [0.0]*(nUsers+nItems),
                                 derivative,
                                 args = (labels, valid, lamb))
    if min_mse > mse:
        min_mse = mse
        opt_lamb = lamb
```

...

In [174]:

```
print(min_mse)
print(opt_lamb)
```

```
0.904152252029804
1.2
```

In [182]:

```
scipy.optimize.fmin_l_bfgs_b(cost,
                             [alpha] + [0.0]*(nUsers+nItems),
                             derivative,
                             args = (labels, dataset, 1.2))
```

```
MSE = 0.8987313676875054
MSE = 0.8861891996778084
MSE = 0.9367774454503509
MSE = 0.8854927942747061
MSE = 0.8896541050782545
MSE = 0.8888137440324333
MSE = 0.8888688677704336
MSE = 0.8888662342932155
```

Out[182]:

```
(array([ 4.57210042e+00, -3.89869928e-03, -6.25858145e-04, ...,
        -1.13793643e-04,  7.19411161e-05, -1.12289754e-04]),
 0.8933592826848815,
 {'grad': array([-2.94641129e-06, -1.58627819e-07,  2.83147152e-08, ...,
          5.90888053e-09, -3.85861981e-09,  3.81020327e-09]),
  'task': 'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL',
  'funcalls': 8,
  'nit': 6,
  'warnflag': 0})
```

In [183]:

```
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('-')
    predictions.write(u + '-' + i + ',' + str(prediction(u, i)) + '\n')

predictions.close()
```