Assignment 5

LARGE SCALE RECOMMENDATION SYSTEMS

Dexing Xu, Collaborate with Weisi Zhang ¶

Task 1

Please include L2 regularization in your model to ensure that the weights matrices are not too large. Make sure you try different regularization parameters [$\lambda \in (0.001, 0.01, 0.1)$] and select the model that gives you the best RMSE under 5-fold cross-validation.

The latent factor means that how many concepts it could have of the movie, if the latent factor is very large, it may contain more concepts, therefore I choose latent factor = 5 to train the model under different λ.

For λ = 0.001:

```
r1: 1.553604
r2: 1.553028
r3: 1.599136
r4: 1.555939
r5: 1.534617
mean: 1.5423
se: 0.0324
```

For λ = 0.010:

```
r1: 0.196986
r2: 0.151038
r3: 0.150487
r4: 0.145012
r5: 0.162326
mean: 0.1663
se: 0.0347
```

For λ = 0.1:

```
r1: 0.165224
r2: 0.113106
r3: 0.093210
r4: 0.081913
r5: 0.003236
mean: 0.0892
se: 0.0724
```

From the result before, I found that when λ = 0.1 the performance is the best.

Task 2

So far, we used the ratings as they were. We didn't try to remove bias factors. There are several ways of removing bias and building recommendation systems on the deviations. Incorporate bias terms in your factorization model and retrain the recommendation engine. You will repeat 5-fold cross-validation to select the best regularization parameters.

Because in the test set, there are users which model never seen before, so the model has no idea about how to assign user bias to the test matrix, in my model, I only add item bias to the test matrix.

For λ = 0.001:

```
r1: 1.751033
r2: 1.705071
r3: 1.628454
r4: 1.695836
r5: 1.773257
mean: 1.7423
se: 0.0346
```

For λ = 0.010:

```
r1: 0.251332
r2: 0.254785
r3: 0.225916
r4: 0.287983
r5: 0.356832
mean: 0.2876
se: 0.0353
```

For λ = 0.1:

```
r1: 0.169313
r2: 0.112089
r3: 0.096164
r4: 0.082955
r5: 0.005346
mean: 0.0958
se: 0.0548
```

From the result before, I found that when λ = 0.1 the performance is the best.

show some of my code below

In [3]:
```
import numpy as np
from scipy.sparse import rand as sprand
from scipy.sparse import lil_matrix
import torch
from torch.autograd import Variable
import pandas as pd
from sklearn.model_selection import cross_val_predict
from math import sqrt
```

In [2]:
```
def get_movielens_ratings(df):
    n_users = max(df.user_id.unique())
    n_items = max(df.item_id.unique())

    interactions = lil_matrix( (n_users,n_items), dtype=float) #np.zeros
((n_users, n_items))
    for row in df.itertuples():
        interactions[row[1] - 1, row[2] - 1] = row[3]
    return interactions
```

In [ ]:
```
names = ['user_id', 'item_id', 'rating', 'timestamp']
ratings = []
test_ratings = []
for i in range(5):
    df_train = pd.read_csv('ml-10M100K/r'+str(i+1)+'.train', sep='::', n
ames=names,engine='python')
    df_test = pd.read_csv('ml-10M100K/r'+str(i+1)+'.test', sep='::', nam
es=names,engine='python')
    ratings.append(get_movielens_ratings(df_train))
    test_ratings.append(get_movielens_ratings(df_test))
```

In [4]:
```
df_train = pd.read_csv('ml-10M100K/r5.train', sep='::', names=names,engi
ne='python')
df_test = pd.read_csv('ml-10M100K/r5.test', sep='::', names=names,engine
='python')
```

In [5]:
```
ratings = get_movielens_ratings(df_train)
test_ratings = get_movielens_ratings(df_test)
```

In [92]:
```
class MatrixFactorization(torch.nn.Module):

    def __init__(self, n_users, n_items, n_factors=5):
        super().__init__()
        self.user_factors = torch.nn.Embedding(n_users,
                                               n_factors,
                                               sparse=False)
        self.item_factors = torch.nn.Embedding(n_items,
                                               n_factors,
                                               sparse=False)
        # Also should consider fitting overall bias (self.mu term) and b
oth user and item bias vectors
        # Mu is 1x1, user_bias is 1xn_users. item_bias is 1xn_items

    # For convenience when we want to predict a sinble user-item pair.
    def predict(self, user, item):
        # Need to fit bias factors
        return (pred + self.user_factors(user) * self.item_factors(item
)).sum(1)

    # Much more efficient batch operator. This should be used for traini
ng purposes
    def forward(self, users, items):
        # Need to fit bias factors
        return torch.mm(self.user_factors(users),torch.transpose(self.it
em_factors(items),0,1))
```

In [4]:
```
class BiasedMatrixFactorization(torch.nn.Module):

    def __init__(self, n_users, n_items, n_factors=5):
        super().__init__()
        self.user_factors = torch.nn.Embedding(n_users,
                                               n_factors,
                                               sparse=False)
        self.item_factors = torch.nn.Embedding(n_items,
                                               n_factors,
                                               sparse=False)
        self.item_biases = torch.nn.Embedding(n_items,
                                              1,
                                              sparse=False)

    def predict(self, user, item):
        pred = (self.user_factors(user) * self.item_factors(item)).sum(1
)
        pred += self.item_biases(item)
        return pred

    def forward(self, users, items):
        return torch.mm(self.user_factors(users),torch.transpose(self.it
em_factors(items),0,1)) + self.item_biases(items)
```

In [8]:
```
def get_batch(batch_size,ratings):
    # Sort our data and scramble it
    rows, cols = ratings.shape
    p = np.random.permutation(rows)

    # create batches
    sindex = 0
    eindex = batch_size
    while eindex < rows:
        batch = p[sindex:eindex]
        temp = eindex
        eindex = eindex + batch_size
        sindex = temp
        yield batch

    if eindex >= rows:
        batch = p[sindex:rows]
        yield batch
```

In [26]:
```
def run_train_epoch(ratings):
    count = 0
    total_loss = 0
    for i,batch in enumerate(get_batch(BATCH_SIZE, ratings)):
        # Set gradients to zero
        reg_loss_func2.zero_grad()

        # Turn data into variables
        interactions = Variable(torch.FloatTensor(ratings[batch, :].toar
ray()))
        rows = Variable(torch.LongTensor(batch))
        cols = Variable(torch.LongTensor(np.arange(ratings.shape[1])))

        # Predict and calculate loss
        predictions = model2(rows, cols)
        loss = loss_func(predictions, interactions)
        count += 1
        total_loss += loss
        # Backpropagate
        loss.backward()

        # Update the parameters
        reg_loss_func2.step()
    print('train avg loss is %f'%(total_loss/count))
```

In [10]:
```
def run_test_epoch(test_ratings):
    for i,batch in enumerate(get_batch(BATCH_SIZE, ratings)):
        # Turn data into variables
        interactions = Variable(torch.FloatTensor(test_ratings[batch, :]
.toarray()))
        rows = Variable(torch.LongTensor(batch))
        cols = Variable(torch.LongTensor(np.arange(test_ratings.shape[1
])))

        # Predict and calculate loss
        predictions = model2(rows, cols)
        loss = loss_func(predictions, interactions)
        # Backpropagate
        #loss.backward()

        # Update the parameters
        #reg_loss_func.step()
    return sqrt(loss), predictions
```

In [20]:
```
loss_func = torch.nn.MSELoss()
```

In [ ]:
```
weight = [0.001, 0.01, 0.1]
for i in range(4):
    print('for train set %d'%(i+1)+':')
    print('******************************')
    model = MatrixFactorization(ratings[i].shape[0], ratings[i].shape[1
], n_factors=5)
    for w in range(3):
        reg_loss_func = torch.optim.SGD(model.parameters(), lr=0.1, weig
ht_decay = weight[w])
        for j in range(EPOCH):
            print('with λ %.3f'%weight[w] + ' EPOCH %d'%j+":")
            run_train_epoch(ratings[i])

        print('with test set %i'%i)
        loss = run_test_epoch(test_ratings[i])
        print('rmse loss is: %f'%loss+' with λ=%.3f'%weight[w]+' train da
ta %d'%(i+1))
        print('----------------------------')
```

In [ ]:
```
weight = [0.001, 0.01, 0.1]
for i in range(4):
    print('for train set %d'%(i+1)+':')
    print('*****************************')
    model = BiasedMatrixFactorization(ratings[i].shape[0], ratings[i].sh
ape[1], n_factors=4)
    for w in range(3):
        reg_loss_func = torch.optim.SGD(model.parameters(), lr=0.1, weig
ht_decay = weight[w])
        for j in range(EPOCH):
            print('with λ %.3f'%weight[w] + ' EPOCH %d'%j+":")
            run_train_epoch(ratings[i])

        print('with test set %i'%(i+1))
        loss = run_test_epoch(test_ratings[i])
        print('rmse loss is: %f'%loss+' with λ=%.3f'%weight[w]+' train da
ta %d'%(i+1))
        print('---------------------------')
```

In [120]:
```
model = MatrixFactorization(ratings.shape[0], ratings.shape[1], n_factor
s=4)
```

In [12]:
```
model2 = BiasedMatrixFactorization(ratings2.shape[0], ratings2.shape[1],
 n_factors=4)
```

In [21]:
```
reg_loss_func2 = torch.optim.SGD(model2.parameters(), lr=0.1, weight_dec
ay = 0.1)
```

In [121]:
```
reg_loss_func = torch.optim.SGD(model.parameters(), lr=0.1, weight_decay
 = 0.1)
```

In [14]:
```
EPOCH = 30
BATCH_SIZE = 1000 #50
LR = 0.1
```

In [27]:
```
pred = None
test_loss = 0
for i in range(EPOCH):
    print('EPOCH: %d'%i)
    run_train_epoch(ratings2)
```

```
EPOCH: 0
train avg loss is 0.061860
EPOCH: 1
train avg loss is 0.058078
EPOCH: 2
train avg loss is 0.057699
EPOCH: 3
train avg loss is 0.056916
EPOCH: 4
train avg loss is 0.055108
EPOCH: 5
train avg loss is 0.054128
EPOCH: 6
train avg loss is 0.053568
EPOCH: 7
train avg loss is 0.050729
EPOCH: 8
train avg loss is 0.050565
EPOCH: 9
train avg loss is 0.049033
EPOCH: 10
train avg loss is 0.048217
EPOCH: 11
train avg loss is 0.045572
EPOCH: 12
train avg loss is 0.043948
EPOCH: 13
train avg loss is 0.043913
EPOCH: 14
train avg loss is 0.041660
EPOCH: 15
train avg loss is 0.040352
EPOCH: 16
train avg loss is 0.039086
EPOCH: 17
train avg loss is 0.040241
EPOCH: 18
train avg loss is 0.038866
EPOCH: 19
train avg loss is 0.038228
EPOCH: 20
train avg loss is 0.037032
EPOCH: 21
train avg loss is 0.035573
EPOCH: 22
train avg loss is 0.034379
EPOCH: 23
train avg loss is 0.034574
EPOCH: 24
train avg loss is 0.032366
```

In [43]:
```
rst = open('assign5_results.tsv','w+')
```

In [44]:
```
def run_test_batch(test_ratings, rst):
    base_id = 0
    for i,batch in enumerate(get_batch(BATCH_SIZE, test_ratings)):
        # Turn data into variables
        interactions = Variable(torch.FloatTensor(test_ratings[batch, :]
.toarray()))
        rows = Variable(torch.LongTensor(batch))
        cols = Variable(torch.LongTensor(np.arange(test_ratings.shape[1
])))

        # Predict and calculate loss
        predictions = model2(rows, cols)
        loss = loss_func(predictions, interactions)
        print('in batch base id: %d'%base_id)
        base_id = recommend(predictions, base_id)
```

In [45]:
```
def recommend(pred, base_id):
    for user in pred:
        if base_id % 100 == 0:
            print(base_id)
        tops = torch.topk(user,100)[1].data.numpy()
        #print(tops)
        recs = np.setdiff1d(tops, np.array(test_ratings.rows[base_id]),
assume_unique=True)
        #print(recs)
        if len(recs) < 5:
            print('Error!\t', base_id)
        if base_id + 1 >= 57375:
            recs = recs[:5]
            rst.write(str(base_id+1) + '\t' + '\t'.join([str(rec) for re
c in recs]) + '\n')
        base_id += 1
    return base_id
```

In [ ]:
```
run_test_batch(test_ratings,rst)
rst.close()
```