In [1]:
```
import gzip
import matplotlib.pyplot as plt
import numpy
import random
import scipy
import tensorflow as tf
from collections import defaultdict
from fastFM import als
from scipy.spatial import distance
```

Data is available at http://cseweb.ucsd.edu/~jmcauley/pml/data/. Download and save to your own directory

In [2]:
```
dataDir = "/home/jmcauley/pml_data/"
```

Factorization Machine (fastFM)

Parse the Goodreads comic book data (excluding review text)

In [3]:
```
def parseData(fname):
    for l in gzip.open(fname):
        d = eval(l)
        del d['review_text'] # Discard the reviews to save memory
        d['year'] = int(d['date_added'][-4:]) # Use this for exercises
        yield d
```

In [4]:
```
data = list(parseData(dataDir + "goodreads_reviews_comics_graphic.json.gz"))
```

In [5]:
```
random.shuffle(data)
```

For example...

In [6]:
```
data[0]
```

Out[6]:
```
{'book_id': '15799191',
 'date_added': 'Sun Jun 30 06:12:24 -0700 2013',
 'date_updated': 'Thu Jul 04 08:02:07 -0700 2013',
 'n_comments': 0,
 'n_votes': 0,
 'rating': 4,
 'read_at': 'Thu Jul 04 08:02:07 -0700 2013',
 'review_id': '86b43a448463928ac5d7887b364e2fcd',
 'started_at': 'Sun Jun 30 00:00:00 -0700 2013',
 'user_id': '23852e2647c217deb24964aadb26be64',
 'year': 2013}
```

Utility data structures. Most importantly, each user and item is mapped to an ID from 1 to nUsers/nItems

In [7]:

```
userIDs,itemIDs = {},{}

for d in data:
    u,i = d['user_id'],d['book_id']
    if not u in userIDs: userIDs[u] = len(userIDs)
    if not i in itemIDs: itemIDs[i] = len(itemIDs)

nUsers,nItems = len(userIDs),len(itemIDs)
```

In [8]:

```
nUsers,nItems
```

Out[8]:

```
(59347, 89311)
```

Build the factorization machine design matrix. Note that each instance is a row, and the columns encode both users and items. Other features could straightforwardly be added.

In [9]:

```
X = scipy.sparse.lil_matrix((len(data), nUsers + nItems))
```

In [10]:

```
for i in range(len(data)):
    user = userIDs[data[i]['user_id']]
    item = itemIDs[data[i]['book_id']]
    X[i,user] = 1 # One-hot encoding of user
    X[i,nUsers + item] = 1 # One-hot encoding of item
```

Target (rating) to predict for each row

In [11]:

```
y = numpy.array([d['rating'] for d in data])
```

Initialize the factorization machine

In [12]:

```
fm = als.FMRegression(n_iter=1000, init_stdev=0.1, rank=5, l2_reg_w=0.1, l2_reg_
```

Split data into train and test portions

In [13]:

```
X_train,y_train = X[:400000],y[:400000]
X_test,y_test = X[400000:],y[400000:]
```

Train the model

In [14]:

```
fm.fit(X_train, y_train)
```

Out[14]:

```
FMRegression(init_stdev=0.1, l2_reg=0, l2_reg_V=0.5, l2_reg_w=0.1, n_iter=1000,
             random_state=123, rank=5)
```

Extract predictions on the test set

In [15]:

```
y_pred = fm.predict(X_test)
```

In [16]:

```
y_pred[:10]
```

Out[16]:
```
array([2.23866277, 4.37847526, 3.84866594, 5.03002627, 3.94993096,
       4.36740166, 4.22497778, 4.30029268, 3.28282377, 4.05036905])
```

In [17]:
```
y_test[:10]
```

Out[17]:
```
array([5, 4, 2, 5, 4, 5, 5, 5, 5, 5])
```

In [18]:
```
def MSE(predictions, labels):
    differences = [(x-y)**2 for x,y in zip(predictions,labels)]
    return sum(differences) / len(differences)
```

In [19]:
```
MSE(y_pred, y_test)
```

Out[19]:
```
1.5940755947213534
```

In [ ]:
```

```

Exercises

6.1

Simple example, just incorporating a one-hot encoding of the year (see data extraction in examples above)

In [20]:
```
minYear = min([d['year'] for d in data])
maxYear = max([d['year'] for d in data])
nYears = maxYear - minYear + 1
```

In [21]:
```
minYear, maxYear, nYears
```

Out[21]:
```
(2005, 2017, 13)
```

In [22]:
```
userIDs,itemIDs = {},{}

for d in data:
    u,i = d['user_id'],d['book_id']
    if not u in userIDs: userIDs[u] = len(userIDs)
    if not i in itemIDs: itemIDs[i] = len(itemIDs)

nUsers,nItems = len(userIDs),len(itemIDs)
```

In [23]:
```
X = scipy.sparse.lil_matrix((len(data), nUsers + nItems + nYears))
```

In [24]:
```
for i in range(len(data)):
    user = userIDs[data[i]['user_id']]
    item = itemIDs[data[i]['book_id']]
```

```
    year = data[i]['year'] - minYear
    X[i,user] = 1 # One-hot encoding of user
    X[i,nUsers + item] = 1 # One-hot encoding of item
    X[i,nUsers + nItems + year] = 1 # One-hot encoding of year
```

In [25]:
```
y = numpy.array([d['rating'] for d in data])
```

In [26]:
```
fm = als.FMRegression(n_iter=1000, init_stdev=0.1, rank=5, l2_reg_w=0.1, l2_reg_
```

In [27]:
```
X_train,y_train,data_train = X[:400000],y[:400000],data[:400000]
X_test,y_test,data_test = X[400000:],y[400000:],data[400000:]
```

In [28]:
```
fm.fit(X_train, y_train)
```

Out[28]:
```
FMRegression(init_stdev=0.1, l2_reg=0, l2_reg_V=0.5, l2_reg_w=0.1, n_iter=1000,
             random_state=123, rank=5)
```

In [29]:
```
y_pred_with_features = fm.predict(X_test)
```

In [30]:
```
MSE(y_pred_with_features, y_test)
```

Out[30]:
```
1.6068283763129323
```

6.2

Cold start plots. Count training instances per item (could also measure coldness per user if we had user features).

In [31]:
```
d
```

Out[31]:
```
{'book_id': '31423340',
 'date_added': 'Mon Sep 18 19:24:36 -0700 2017',
 'date_updated': 'Thu Sep 21 04:54:01 -0700 2017',
 'n_comments': 6,
 'n_votes': 13,
 'rating': 4,
 'read_at': 'Thu Sep 21 15:03:21 -0700 2017',
 'review_id': '4f39bbb5aeab832c794a3d2e40943949',
 'started_at': 'Tue Sep 19 20:23:13 -0700 2017',
 'user_id': '1d945500234cbc7a6138a4d017dbfe4b',
 'year': 2017}
```

In [32]:
```
nTrainPerItem = defaultdict(int)
for d in data_train:
    nTrainPerItem[d['book_id']] += 1
```

In [33]:
```
rmsePerNtrainFeatures = defaultdict(list)
rmsePerNtrain = defaultdict(list)
```

```
for d,y,ypf,yp in zip(data_test,y_test,y_pred_with_features,y_pred):
    e2_features = (y-ypf)**2
    e2 = (y-yp)**2
    nt = nTrainPerItem[d['book_id']]
    if nt < 5:
        rmsePerNtrainFeatures[str(nt)].append(e2_features)
        rmsePerNtrain[str(nt)].append(e2)
    elif nt < 10:
        rmsePerNtrainFeatures['5-9'].append(e2_features)
        rmsePerNtrain['5-9'].append(e2)
    elif nt < 20:
        rmsePerNtrainFeatures['10-19'].append(e2_features)
        rmsePerNtrain['10-19'].append(e2)
    else:
        rmsePerNtrainFeatures['20+'].append(e2_features)
        rmsePerNtrain['20+'].append(e2)
```

In [34]:

```
for r in rmsePerNtrain:
    rmsePerNtrainFeatures[r] = sum(rmsePerNtrainFeatures[r]) / len(rmsePerNtrain
    rmsePerNtrain[r] = sum(rmsePerNtrain[r]) / len(rmsePerNtrain[r])
```

In [35]:

```
rmsePerNtrain
```

Out[35]:

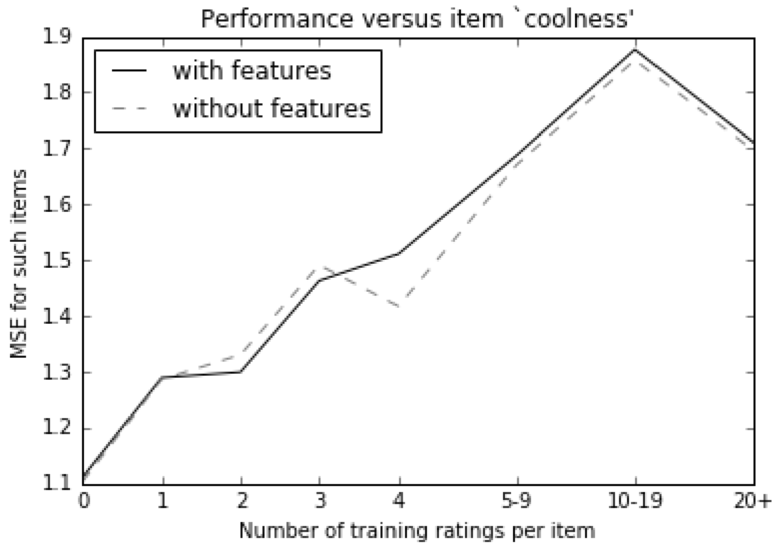
```
defaultdict(list,
            {'0': 1.1034747614259885,
             '1': 1.286213343267511,
             '10-19': 1.8579978486101703,
             '2': 1.3303440529731148,
             '20+': 1.696268806936738,
             '3': 1.4912379121450905,
             '4': 1.4174955135661096,
             '5-9': 1.670982336713272})
```

In [36]:

```
Xlab = ['0','1','2','3','4','5-9','10-19','20+']
X = [0,1,2,3,4,5.5,7,8.5] # Average of above ranges
YF = [rmsePerNtrainFeatures[x] for x in Xlab]
Y = [rmsePerNtrain[x] for x in Xlab]
```

In [37]:

```
plt.xlim(0, max(X))
plt.plot(X,YF,color='k',label='with features')
plt.plot(X,Y,color='grey',linestyle='--',label='without features')
plt.xticks(X,Xlab)
plt.xlabel("Number of training ratings per item")
plt.ylabel("MSE for such items")
plt.title("Performance versus item `coolness'")
plt.legend(loc="best")
plt.show()
```



6.3

Read social data from epinions

In [38]:

```
userIDs = {}
itemIDs = {}
interactions = []

socialTrust = defaultdict(set)

f = open(dataDir + "epinions_data/epinions.txt", 'rb')
header = f.readline()

for l in f:
    try:
        l = l.decode('utf-8')
        l = l.split()
    except Exception as e:
        continue
    i = l[0]
    u = l[1]

    if not u in userIDs: userIDs[u] = len(userIDs)
    if not i in itemIDs: itemIDs[i] = len(itemIDs)
    interactions.append((u,i))

f.close()

f = open(dataDir + "epinions_data/network_trust.txt", 'r')
for l in f:
    try:
        u,_,v = l.strip().split()
    except Exception as e:
        continue
    if u in userIDs and v in userIDs:
        socialTrust[u].add(v)

f.close()
```

In [39]:

```
random.shuffle(interactions)
```

In [40]:

```
items = list(itemIDs.keys())
```

BPR model. First we'll use a regular BPR model just to assess similarity between friends' latent representations. Later we can implement different social sampling assumptions just by passing different samples to the same model.

In [41]:

```
class BPRbatch(tf.keras.Model):
    def __init__(self, K, lamb):
        super(BPRbatch, self).__init__()
        # Initialize variables
        self.betaI = tf.Variable(tf.random.normal([len(itemIDs)],stddev=0.001))
        self.gammaU = tf.Variable(tf.random.normal([len(userIDs),K],stddev=0.001
        self.gammaI = tf.Variable(tf.random.normal([len(itemIDs),K],stddev=0.001
        # Regularization coefficient
        self.lamb = lamb

    # Prediction for a single instance
    def predict(self, u, i):
        p = self.betaI[i] + tf.tensordot(self.gammaU[u], self.gammaI[i], 1)
        return p

    # Regularizer
    def reg(self):
        return self.lamb * (tf.nn.l2_loss(self.betaI) +\
                            tf.nn.l2_loss(self.gammaU) +\
                            tf.nn.l2_loss(self.gammaI))

    def score(self, sampleU, sampleI):
        u = tf.convert_to_tensor(sampleU, dtype=tf.int32)
        i = tf.convert_to_tensor(sampleI, dtype=tf.int32)
        beta_i = tf.nn.embedding_lookup(self.betaI, i)
        gamma_u = tf.nn.embedding_lookup(self.gammaU, u)
        gamma_i = tf.nn.embedding_lookup(self.gammaI, i)
        x_ui = beta_i + tf.reduce_sum(tf.multiply(gamma_u, gamma_i), 1)
        return x_ui

    def call(self, sampleU, sampleI, sampleJ):
        x_ui = self.score(sampleU, sampleI)
        x_uj = self.score(sampleU, sampleJ)
        return -tf.reduce_mean(tf.math.log(tf.math.sigmoid(x_ui - x_uj)))
```

In [42]:

```
def trainingStepBPR(model, interactions):
    Nsamples = 50000
    with tf.GradientTape() as tape:
        sampleU, sampleI, sampleJ = [], [], []
        for _ in range(Nsamples):
            u,i = random.choice(interactions) # positive sample
            j = random.choice(items) # negative sample
            while j in itemsPerUser[u]:
                j = random.choice(items)
            sampleU.append(userIDs[u])
            sampleI.append(itemIDs[i])
            sampleJ.append(itemIDs[j])
```

```
            loss = model(sampleU,sampleI,sampleJ)
            loss += model.reg()
        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients((grad, var) for
                                  (grad, var) in zip(gradients, model.trainable_vari
                                  if grad is not None)
        return loss.numpy()
```

First, train a regular BPR model (no social terms)

In [43]:

```
optimizer = tf.keras.optimizers.Adam(0.1)
modelBPR = BPRbatch(10, 0.00001)
```

In [44]:

```
nTrain = int(len(interactions) * 0.9)
nTest = len(interactions) - nTrain
interactionsTrain = interactions[:nTrain]
interactionsTest = interactions[nTrain:]
```

In [45]:

```
itemsPerUser = defaultdict(list)
usersPerItem = defaultdict(list)
for u,i in interactionsTrain:
    itemsPerUser[u].append(i)
    usersPerItem[i].append(u)
```

In [46]:

```
for i in range(100):
    obj = trainingStepBPR(modelBPR, interactions)
    if (i % 10 == 9): print("iteration " + str(i+1) + ", objective = " + str(obj
```

```
iteration 10, objective = 0.5586551
iteration 20, objective = 0.5336623
iteration 30, objective = 0.5308751
iteration 40, objective = 0.53656584
iteration 50, objective = 0.5435766
iteration 60, objective = 0.543326
iteration 70, objective = 0.54365164
iteration 80, objective = 0.54245454
iteration 90, objective = 0.5428503
iteration 100, objective = 0.5451498
```

In [47]:

```
interactionsTestPerUser = defaultdict(set)
itemSet = set()
for u,i in interactionsTest:
    interactionsTestPerUser[u].add(i)
    itemSet.add(i)
```

In [48]:

```
def AUCu(model, u, N):
    win = 0
    if N > len(interactionsTestPerUser[u]):
        N = len(interactionsTestPerUser[u])
    positive = random.sample(interactionsTestPerUser[u],N)
    negative = random.sample(itemSet.difference(interactionsTestPerUser[u]),N)
    for i,j in zip(positive,negative):
```

```
        si = model.predict(userIDs[u], itemIDs[i]).numpy()
        sj = model.predict(userIDs[u], itemIDs[j]).numpy()
        if si > sj:
            win += 1
    return win/N
```

In [49]:

```
def AUC(model):
    av = []
    for u in interactionsTestPerUser:
        av.append(AUCu(model, u, 10))
    return sum(av) / len(av)
```

In [50]:

```
AUC(modelBPR)
```

Out[50]: 0.6800902434544706

Compute similarities among friends' latent representations

In [51]:

```
sims = []
simFriends = []
while len(sims) < 10000:
    try:
        u,i = random.choice(interactions)
        v = random.sample(socialTrust[u],1)[0] # trust link
        j = random.sample(itemsPerUser[v],1)[0] # friend's item
        k = random.choice(items) # random item
    except Exception as e:
        continue
    s1 = 1 - distance.cosine(modelBPR.gammaI[itemIDs[i]],modelBPR.gammaI[itemIDs
    s2 = 1 - distance.cosine(modelBPR.gammaI[itemIDs[i]],modelBPR.gammaI[itemIDs
    if s1 > 1:
        print("?")
        break
    sims.append(s1)
    simFriends.append(s2)
```

Similarity between randomly chosen pairs of items

In [52]:

```
sum(sims)/len(sims)
```

Out[52]: 0.003534959910223597

Similarity between an item and one consumed by a friend

In [53]:

```
sum(simFriends)/len(simFriends)
```

Out[53]: 0.04061316501272158

(similarity is not particularly high, but still significantly higher than random pairs)

6.4

Implement the social model. Uses the model above, just with different samples.

In [54]:
```
def trainingStepBPRsocial(model, interactions):
    Nsamples = 50000
    with tf.GradientTape() as tape:
        sampleU, sampleI, sampleJ = [], [], []
        while len(sampleU) < Nsamples/2:
            try:
                u,i = random.choice(interactions) # positive sample
                v = random.sample(socialTrust[u],1)[0] # trust link
                j = random.sample(itemsPerUser[v],1)[0] # friend's item
                k = random.choice(items) # negative item
                if j in itemsPerUser[u] or k in itemsPerUser[u]:
                    continue
            except Exception as e:
                continue
            while j in itemsPerUser[u]:
                j = random.choice(items)
            sampleU.append(userIDs[u])
            sampleI.append(itemIDs[i]) # Positive
            sampleJ.append(itemIDs[j]) # greater than social
            sampleU.append(userIDs[u])
            sampleI.append(itemIDs[j]) # Social
            sampleJ.append(itemIDs[k]) # greater than negative

        loss = model(sampleU,sampleI,sampleJ)
        loss += model.reg()
    gradients = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients((grad, var) for
                              (grad, var) in zip(gradients, model.trainable_vari
                              if grad is not None)
    return loss.numpy()
```

In [55]:
```
optimizer = tf.keras.optimizers.Adam(0.1)
modelBPRsocial = BPRbatch(10, 0.00001)
```

In [56]:
```
for i in range(100):
    obj = trainingStepBPRsocial(modelBPRsocial, interactions)
    if (i % 10 == 9): print("iteration " + str(i+1) + ", objective = " + str(obj
```

```
iteration 10, objective = 0.6443011
iteration 20, objective = 0.618395
iteration 30, objective = 0.6267633
iteration 40, objective = 0.6299376
iteration 50, objective = 0.6279181
iteration 60, objective = 0.6186671
iteration 70, objective = 0.6207659
iteration 80, objective = 0.6173624
iteration 90, objective = 0.61636496
iteration 100, objective = 0.60944057
```

In [57]:
```
AUC(modelBPRsocial)
```

Out[57]:
```
0.6322810869785714
```

In [ ]: