In [1]:
```
import numpy
import urllib
import scipy.optimize
import random
from sklearn import linear_model
import gzip
from collections import defaultdict
```

In [2]:
```
def assertFloat(x):
    assert type(float(x)) == float

def assertFloatList(items, N):
    assert len(items) == N
    assert [type(float(x)) for x in items] == [float]*N
```

In [3]:
```
f = open("5year.arff", 'r')
```

In [4]:
```
while not '@data' in f.readline():
    pass

dataset = []
for l in f:
    if '?' in l: # Missing entry
        continue
    l = l.split(',')
    values = [1] + [float(x) for x in l]
    values[-1] = values[-1] > 0 # Convert to bool
    dataset.append(values)
```

In [5]:
```
X = [d[:-1] for d in dataset]
y = [d[-1] for d in dataset]
```

In [6]:
```
answers = {}
```

In [7]:
```
def accuracy(pred, y):
    correct = pred == y
    return sum(correct) / len(correct)
```

In [8]:
```
def rates(predictions, y):
    TP = [a and b for (a,b) in zip(predictions,y)]
    TN = [not a and not b for (a,b) in zip(predictions,y)]
    FP = [a and not b for (a,b) in zip(predictions,y)]
    FN = [not a and b for (a,b) in zip(predictions,y)]

    TP = sum(TP)
    TN = sum(TN)
    FP = sum(FP)
    FN = sum(FN)

    return TP, TN, FP, FN
```

In [9]:
```
def BER(predictions, y):
    TP, TN, FP, FN = rates(predictions, y)

    TPR = 0
    if TP > 0:
        TPR = TP / (TP + FN)
    TNR = 0
    if TN > 0:
        TNR = TN / (TN + FP)
    return 1 - 0.5 * (TPR + TNR)
```

In [10]:
```
### Question 1
```

In [11]:
```
mod = linear_model.LogisticRegression(C=1)
mod.fit(X,y)

pred = mod.predict(X)
```

In [12]:
```
answers['Q1'] = [accuracy(pred,y), BER(pred, y)]
```

In [13]:
```
assertFloatList(answers['Q1'], 2)
```

In [14]:
```
### Question 2
```

In [15]:
```
mod = linear_model.LogisticRegression(C=1, class_weight='balanced')
mod.fit(X,y)

pred = mod.predict(X)
```

In [16]:
```
BER(pred, y)
```
Out[16]:
```
0.20695010677538339
```

In [17]:
```
accuracy(pred, y)
```
Out[17]:
```
0.7829099307159529
```

In [18]:
```
answers['Q2'] = [accuracy(pred,y), BER(pred, y)]
```

In [19]:
```
assertFloatList(answers['Q2'], 2)
```

In [20]:
```
### Question 3
```

In [21]:
```
random.seed(0)
random.shuffle(dataset)
```

In [22]:
```
X = [d[:-1] for d in dataset]
y = [d[-1] for d in dataset]
```

In [23]:
```
Xtrain, Xvalid, Xtest = X[:len(X)//2], X[len(X)//2:(3*len(X))//4], X[(3*len(X))//4:]
ytrain, yvalid, ytest = y[:len(X)//2], y[len(X)//2:(3*len(X))//4], y[(3*len(X))//4:]
```

In [24]:
```
len(Xtrain), len(Xvalid), len(Xtest)
```
Out[24]:
```
(1515, 758, 758)
```

In [25]:
```
mod = linear_model.LogisticRegression(C=1.0, class_weight='balanced')
mod.fit(Xtrain,ytrain)

predTrain = mod.predict(Xtrain)
predValid = mod.predict(Xvalid)
predTest = mod.predict(Xtest)
```

In [26]:
```
BER(predTrain, ytrain)
```
Out[26]:
```
0.19297679580827509
```

In [27]:
```
BER(predValid, yvalid)
```
Out[27]:
```
0.21939006267364469
```

In [28]:
```
BER(predTest, ytest)
```
Out[28]:
```
0.22289628180039145
```

In [29]:
```
answers['Q3'] = [BER(predTrain, ytrain), BER(predValid, yvalid), BER(predTest, ytest)]
```

In [30]:
```
assertFloatList(answers['Q3'], 3)
```

In [31]:
```
### Question 4
```

In [32]:
```
models = {}
bers = {}
bestC = None

for c in [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000]:
    mod = linear_model.LogisticRegression(C=c, class_weight='balanced')
    mod.fit(Xtrain,ytrain)
    predictions = mod.predict(Xvalid)
    ber = BER(predictions, yvalid)
    if bestC == None or ber < bers[bestC]:
        bestC = c
    models[c] = mod
    bers[c] = ber
```

In [33]:
```
answers['Q4'] = list(bers.values())
```

In [34]:
```
assertFloatList(answers['Q4'], 9)
```

In [35]:
```
### Question 5
```

In [36]:
```
predictions = models[bestC].predict(Xtest)
ber = BER(predictions, ytest)
```

In [37]:
```
answers['Q5'] = [bestC, ber]
```

In [38]:
```
assertFloatList(answers['Q5'], 2)
```

In [39]:
```
### Question 6
```

In [40]:
```
f = gzip.open("young_adult_10000.json.gz")
dataset = []
for l in f:
    dataset.append(eval(l))
```

In [41]:
```
dataTrain = dataset[:9000]
dataTest = dataset[9000:]
```

In [42]:
```
dataset[0]
```
Out[42]:
```
{'book_id': '2767052',
 'date_added': 'Wed Jan 13 13:38:25 -0800 2010',
 'date_updated': 'Wed Mar 22 11:46:36 -0700 2017',
 'n_comments': 25,
 'n_votes': 24,
 'rating': 5,
 'read_at': 'Sun Mar 25 00:00:00 -0700 2012',
 'review_id': '248c011811e945eca861f82c51a549291',
 'review_text': 'I cracked and finally picked this up. Very enjoyable quick read - couldn't put it down - it was like crack. \n I'm a bit bothered by the lack of backstory of how Panem and the Hunger Games come about. It is just kind o
f explained away in a few paragraphs and we are left to accept this very strange world where teenagers are pitted int
o an arena each year to kill each other? I was expecting it because I've seen Battle Royale, but would have appreci
ated knowing more of the backstory of how the world could have come into such a odd state. \n I suppose what makes a
 book like this interesting is thinking about the strategy of it all. The players are going to be statistically enour
aged to band together because they will last longer that way, but by definition of course any partnership will be bro
ken, and the drama of how that unfolds is always interesting and full of friendships broken and betrayal. Each chara
cter approached the game in their own way. Some banded together in larger coalitions, some were loners initially and b
anded together later. And some were just loners, like Prefarce. A lot depended on your survival skills could you find
 food and water on your own? Self-dependence is highly valued - and of course our hero was strong there. \n All in all
, a fun read, but I feel kind of dirty for having read it.',
 'started_at': 'Fri Mar 23 00:00:00 -0700 2012',
 'user_id': '8842281e1d1347389f2ab93d60773d4d'}
```

In [43]:
```
userPerItem = defaultdict(set) # Maps an item to the users who rated it
itemsPerUser = defaultdict(set) # Maps a user to the items that they rated
reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)
itemNames = {}
ratingDict = {} # To retrieve a rating for a specific user/item pair

for d in dataTrain:
    user,item = d['user_id'], d['book_id']
    userPerItem[item].add(user)
    itemsPerUser[user].add(item)
    reviewsPerUser[user].append(d)
    reviewsPerItem[item].append(d)
    ratingDict[(user,item)] = d['rating']
```

In [44]:
```
userAverages = {}
itemAverages = {}
ratingMean = []

for u in itemsPerUser:
    rs = [ratingDict[(u,i)] for i in itemsPerUser[u]]
    userAverages[u] = sum(rs) / len(rs)

for i in usersPerItem:
    rs = [ratingDict[(u,i)] for u in usersPerItem[i]]
    itemAverages[i] = sum(rs) / len(rs)

for d in dataTrain:
    ratingMean.append(d['rating'])

ratingMean = sum(ratingMean) / len(ratingMean)
```

In [45]:
```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    if denom == 0:
        return 0
    return numer / denom
```

In [46]:
```
def mostSimilar(i, N):
    similarities = []
    users = usersPerItem[i]
    for i2 in usersPerItem:
        if i2 == i: continue
        sim = Jaccard(users, usersPerItem[i2])
        #sim = Pearson(i, i2) # Could use alternate similarity metrics straightforwardly
        similarities.append((sim,i2))
    similarities.sort(reverse=True)
    return similarities[:10]
```

In [47]:
```
mostSimilar('2767052', 10)
```
Out[47]:
```
[(0.4125, '6148028'),
 (0.3431764705882353, '7260188'),
 (0.15909090909090991, '256683'),
 (0.1375, '1362543'),
 (0.11499252873563218, '11735983'),
 (0.10989010989010989, '13235037'),
 (0.10810810810810811, '28187'),
 (0.10666666666666667, '428262'),
 (0.09676543209876543, '49041'),
 (0.09782608695652174, '41665')]
```

In [48]:
```
answers['Q6'] = mostSimilar('2767052', 10)
```

In [49]:
```
assert len(answers['Q6']) == 10
assertFloatList([s[0] for s in answers['Q6']], 10)
```

In [50]:
```
### Question 7
```

In [51]:
```
def MSE(y, ypred):
    diffs = [(a-b)**2 for (a,b) in zip(y,ypred)]
    return sum(diffs) / len(diffs)
```

In [52]:
```
def predictRating(user,item):
    ratings = []
    similarities = []
    for d in reviewsPerUser[user]:
        i2 = d['book_id']
        if i2 == item: continue
        ratings.append(d['rating'] - itemAverages[i2])
        similarities.append(Jaccard(usersPerItem[item],usersPerItem[i2]))
    if (sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings,similarities)]
        return itemAverages[item] + sum(weightedRatings) / sum(similarities)
    else:
        # User hasn't rated any similar items
        return ratingMean
```

In [53]:
```
alwaysPredictMean = [ratingMean for d in dataTest]
```

In [54]:
```
simPredictions = [predictRating(d['user_id'], d['book_id']) for d in dataTest]
```

In [55]:
```
labels = [d['rating'] for d in dataTest]
```

In [56]:
```
MSE(alwaysPredictMean, labels)
```
Out[56]:
```
1.2237430123456756
```

In [57]:
```
MSE(simPredictions, labels)
```
Out[57]:
```
1.2469091498159586
```

In [58]:
```
answers['Q7'] = MSE(simPredictions, labels)
```

In [59]:
```
assertFloat(answers['Q7'])
```

In [60]:
```
### Question 8
```

In [61]:
```
def predictRating8(user,item):
    ratings = []
    similarities = []
    for d in reviewsPerItem[item]:
        u2 = d['user_id']
        if u2 == user: continue
        ratings.append(d['rating'] - userAverages[u2])
        similarities.append(Jaccard(itemsPerUser[user],itemsPerUser[u2]))
    if (sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings,similarities)]
        return itemAverages[item] + sum(weightedRatings) / sum(similarities)
    else:
        # User hasn't rated any similar items
        return ratingMean
```

In [62]:
```
simPredictions = [predictRating8(d['user_id'], d['book_id']) for d in dataTest]
```

In [63]:
```
MSE(simPredictions, labels)
```
Out[63]:
```
1.2539815675307753
```

In [64]:
```
answers['Q8'] = MSE(simPredictions, labels)
```

In [65]:
```
assertFloat(answers['Q8'])
```

In [66]:
```
f = open('answers_hw2.txt', 'w')
f.write(str(answers) + '\n')
f.close()
```