In [1]:
```
import json
from matplotlib import pyplot as plt
from collections import defaultdict
from sklearn import linear_model
import numpy
import random
import gzip
import math
```

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

In [4]:
```
len(dataset)
```

Out[4]:
```
10000
```

In [5]:
```
answers = {}
```

In [6]:
```
dataset[0]
```

Out[6]:
```
{'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': '248c011811e945ecab8435c31a549291',
 '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 backst
ory of how Panem and the Hunger Games came about. It is just kind of explained away in a few paragraphs and we are left to accept this very strange world where teenager
s are pitted into an arena each year to kill each other? I was expecting it because I've seen Battle Royale, but I would have appreciated 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 g
oing to be statistically encouraged to band together because they will last longer that way, but by definition of course any partnership will be broken, and the drama o
f how that unfolds is always interesting and full of friendships broken and betrayal. Each character approached the game in their own way. Some banded together in large
 numbers, some were lovers initially and banded together later. And some were just loners, like Prakers. A lot depended on your survival skill: 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 i
t.",
 'started_at': 'Fri Mar 23 00:00:00 -0700 2012',
 'user_id': '8842281e1d1347389f2ab93d60773d4d'}
```

In [7]:
```
### Question 1
```

In [8]:
```
def feature(datum):
    return [1] + [datum['review_text'].count('!')]
```

In [9]:
```
X = [feature(d) for d in dataset]
Y = [d['rating'] for d in dataset]
```

In [10]:
```
theta,residuals,rank,s = numpy.linalg.lstsq(X,Y)
```

In [11]:
```
theta
```

Out[11]:
```
array([ 3.68853306,  0.07109019])
```

In [12]:
```
residuals
```

Out[12]:
```
array([ 15231.74749454])
```

In [13]:
```
#MSE
residuals[0] / len(Y)
```

Out[13]:
```
1.5231747494538376
```

In [14]:
```
answers['Q1'] = [theta[0], theta[1], residuals[0] / len(Y)]
```

In [15]:
```
assertFloatList(answers['Q1'], 3)
```

In [16]:
```
### Question 2
```

In [17]:
```
def feature(datum):
    return [1] + [len(datum['review_text']), datum['review_text'].count('!')]
```

In [18]:
```
X = [feature(d) for d in dataset]
Y = [d['rating'] for d in dataset]
```

In [19]:
```
theta,residuals,rank,s = numpy.linalg.lstsq(X,Y)
```

In [20]:
```
theta
```

Out[20]:
```
array([  3.71751281e+00,  -4.12156653e-05,   7.52759173e-02])
```

In [21]:
```
residuals[0] / len(Y)
```

Out[21]:
```
1.5214029246165923
```

In [22]:
```
answers['Q2'] = [theta[0], theta[1], theta[2], residuals[0] / len(Y)]
```

In [23]:
```
assertFloatList(answers['Q2'], 4)
```

In [24]:
```
### Question 3
```

In [25]:
```
def feature(datum, deg):
    feat = [1]
    for i in range(1, deg + 1):
        feat.append(datum['review_text'].count('!')**i)
    return feat


mses = []
for i in range(1,6):
    X = [feature(d, i) for d in dataset]
    Y = [d['rating'] for d in dataset]
    theta,residuals,rank,s = numpy.linalg.lstsq(X,Y)
    print("Degree " + str(i) + ":")
    print("  theta = " + str(theta))
    print("  MSE = " + str(residuals[0] / len(Y)))
    mses.append(residuals[0] / len(Y))
```

```
Degree 1:
  theta = [ 3.68853306  0.07109019]
  MSE = 1.5231747494538376
Degree 2:
  theta = [ 3.6505779   0.13744771 -0.00378271]
  MSE = 1.5044686410463
Degree 3:
  theta = [  3.62678057e+00   2.06720992e-01  -1.22920755e-02   1.51501685e-04]
  MSE = 1.4984655155152
Degree 4:
  theta = [ 3.60559965e+00   2.84522319e-01  -3.02351463e-02   1.04623040e-03
 -1.08917613e-05]
  MSE = 1.4954477332
Degree 5:
  theta = [  3.60026772e+00   3.21702721e-01  -4.17739020e-02   2.02489606e-03
 -3.93946096e-05   2.59479662e-07]
  MSE = 1.4896160954
```

In [27]:
```
answers['Q3'] = mses
```

In [28]:
```
assertFloatList(answers['Q3'], 5)
```

In [29]:
```
### Question 4
```

In [30]:
```
mses = []
for i in range(1,6):
    X = [feature(d, i) for d in dataset]
    Y = [d['rating'] for d in dataset]

    Xtrain = X[:len(X)//2]
    Xtest = X[len(X)//2:]
    Ytrain = Y[:len(Y)//2]
    Ytest = Y[len(Y)//2:]

    mod = linear_model.LinearRegression()
    mod.fit(Xtrain,Ytrain)
    Ypred = mod.predict(Xtest)
    MSE = sum([(yp - ys)**2 for (yp,ys) in zip(Ypred, Ytest)]) / len(Ytest)
    print("Degree " + str(i) + ":")
    print("  test MSE = " + str(MSE))
    mses.append(MSE)
```

```
Degree 1:
  test MSE = 1.52487438599
Degree 2:
  test MSE = 1.49771992593
Degree 3:
  test MSE = 1.48566321903
Degree 4:
  test MSE = 1.47673374681
Degree 5:
  test MSE = 1.48095772728
```

In [31]:
```
answers['Q4'] = mses
```

In [32]:
```
assertFloatList(answers['Q4'], 5)
```

In [33]:
```
### Question 5
```

In [34]:
```
ycopy = Y[:]
```

In [35]:
```
ycopy.sort()
```

In [36]:
```
median = ycopy[len(ycopy)//2]
```

In [37]:
```
median
```

Out[37]:
```
4
```

In [38]:
```
MAE = sum([math.fabs(y - median) for y in Y]) / len(Y)
```

In [39]:
```
MAE
```

Out[39]:
```
0.8923
```

In [40]:
```
answers['Q5'] = MAE
```

In [41]:
```
assertFloat(answers['Q5'])
```

In [42]:
```
### Question 6
```

In [43]:
```
f = open("beer_50000.json")
dataset = []
for l in f:
    if 'user/gender' in l:
        dataset.append(eval(l))
```

In [44]:
```
len(dataset)
```

Out[44]:
```
20483
```

In [45]:
```
X = [[1, d['review/text'].count('!')] for d in dataset]
y = [d['user/gender'] == 'Female' for d in dataset]
```

In [46]:
```
mod = linear_model.LogisticRegression()
mod.fit(X,y)
predictions = mod.predict(X) # Binary vector of predictions
correct = predictions == y # Binary vector indicating which predictions were correct
print(sum(correct) / len(correct))
```

```
0.9849043180758
```

In [47]:
```
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)]
```

In [48]:
```
TP = sum(TP)
TN = sum(TN)
FP = sum(FP)
FN = sum(FN)
```

In [49]:
```
BER = 0.5 * (FP / (TN + FP) + FN / (FN + TP))
```

In [50]:
```
BER
```

Out[50]:
```
0.5
```

In [51]:
```
answers['Q6'] = [TP, TN, FP, FN, BER]
```

In [52]:
```
assertFloatList(answers['Q6'], 5)
```

In [53]:
```
### Question 7
```

In [54]:
```
mod = linear_model.LogisticRegression(class_weight='balanced')
mod.fit(X,y)
predictions = mod.predict(X) # Binary vector of predictions
correct = predictions == y # Binary vector indicating which predictions were correct
print(sum(correct) / len(correct))
```

```
0.8047836102053
```

In [55]:
```
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)]
```

In [56]:
```
TP = sum(TP)
TN = sum(TN)
FP = sum(FP)
FN = sum(FN)
```

In [57]:
```
BER = 0.5 * (FP / (TN + FP) + FN / (FN + TP))
```

In [58]:
```
BER
```

Out[58]:
```
0.45077311342551652
```

In [59]:
```
answers['Q7'] = [TP, TN, FP, FN, BER]
```

In [60]:
```
assertFloatList(answers['Q7'], 5)
```

In [61]:
```
### Question 8
```

In [62]:
```
scores = [x[1] for x in mod.predict_proba(X)]
```

In [63]:
```
sortedScores = list(zip(scores,y))
sortedScores.sort(reverse=True)
sortedLabels = [x[1] for x in sortedScores]
```

In [64]:
```
precs = []
for k in [1,10,100,1000,10000]: # Not efficient, but fine
    precK = sum(sortedLabels[:k]) / k
    print("Precision@" + str(k) + " = " + str(precK))
    precs.append(precK)
```

```
Precision@1 = 0.0
Precision@10 = 0.0
Precision@100 = 0.03
Precision@1000 = 0.033
Precision@10000 = 0.0308
```

In [65]:
```
answers['Q8'] = precs
```

In [66]:
```
assertFloatList(answers['Q8'], 5)
```

In [67]:
```
f = open('answers_hw1.txt', 'w')
f.write(str(answers) + '\n')
f.close()
```

In [ ]:
```

```