```
import json
import gzip
import math
from collections import defaultdict
import numpy
from sklearn import linear_model
```

```
# This will suppress any warnings, comment out if you'd like to preserve them
import warnings
warnings.filterwarnings("ignore")
```

```
# Check formatting of submissions
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
```

```
answers = {}
```

```
f = open("spoilers.json.gz", 'r')
```

```
dataset = []
for l in f:
    d = eval(l)
    dataset.append(d)
```

```
f.close()
```

```
# A few utility data structures
reviewsPerUser = defaultdict(list)
reviewsPerItem = defaultdict(list)

for d in dataset:
    u,i = d['user_id'],d['book_id']
    reviewsPerUser[u].append(d)
    reviewsPerItem[i].append(d)

# Sort reviews per user by timestamp
for u in reviewsPerUser:
    reviewsPerUser[u].sort(key=lambda x: x['timestamp'])

# Same for reviews per item
for i in reviewsPerItem:
    reviewsPerItem[i].sort(key=lambda x: x['timestamp'])
```

```
# E.g. reviews for this user are sorted from earliest to most recent
[d['timestamp'] for d in reviewsPerUser['bdd7e561ca59e313b72b8dc30a5b1862e']]
```

```
### 1a
```

```

```

```
answers['Q1a'] = MSE(y,ypred)
```

```
assertFloat(answers['Q1a'])
```

```
### 1b
```

```

```

```
answers['Q1b'] = MSE(y,ypred)
```

```
assertFloat(answers['Q1b'])
```

```
### 2
```

```
answers['Q2'] = []

for N in [1,2,3]:
    # etc.
    answers['Q2'].append(MSE(y,ypred))
```

```
assertFloatList(answers['Q2'], 3)
```

```
### 3a
```

```
def feature3(N, u): # For a user u and a window size of N
```

```
answers['Q3a'] = [feature3(2,dataset[0]['user_id']), feature3(3,dataset[0]['user_id'])]
```

```
assert len(answers['Q3a']) == 2
assert len(answers['Q3a'][0]) == 3
assert len(answers['Q3a'][1]) == 4
```

```
### 3b
```

```
answers['Q3b'] = []

for N in [1,2,3]:
    # etc.
    answers['Q3b'].append(mse)
```

```
assertFloatList(answers['Q3b'], 3)
```

```
### 4a
```

```
globalAverage = [d['rating'] for d in dataset]
globalAverage = sum(globalAverage) / len(globalAverage)
```

```
def featureMeanValue(N, u): # For a user u and a window size of N
```

```
def featureMissingValue(N, u):
```

```
answers['Q4a'] = [featureMeanValue(10, dataset[0]['user_id']), featureMissingValue(10, dataset[0]['user_id'])]
```

```
assert len(answers['Q4a']) == 2
assert len(answers['Q4a'][0]) == 11
assert len(answers['Q4a'][1]) == 21
```

```
### 4b
```

```
answers['Q4b'] = []

for featFunc in [featureMeanValue, featureMissingValue]:
    # etc.
    answers['Q4b'].append(mse)
```

```
assertFloatList(answers['Q4b'], 2)
```

```
### 5
```

```
def feature5(sentence):
```

```
y = []
X = []

for d in dataset:
    for spoiler,sentence in d['review_sentences']:
        X.append(feature5(sentence))
        y.append(spoiler)
```

```

```

```
answers['Q5a'] = X[0]
```

```
answers['Q5b'] = [TP, TN, FP, FN, BER]
```

```
assert len(answers['Q5a']) == 4
assertFloatList(answers['Q5b'], 5)
```

```
### 6
```

```
def feature6(review):
```

```
y = []
X = []

for d in dataset:
    sentences = d['review_sentences']
    if len(sentences) < 4: continue
    X.append(feature6(d))
    y.append(sentences[5][0])

#etc.
```

```
answers['Q6a'] = X[0]
```

```
answers['Q6b'] = BER
```

```
assert len(answers['Q6a']) == 9
assertFloat(answers['Q6b'])
```

```
### 7
```

```
# 50/25/25% train/valid/test split
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:]
```

```
for c in [0.01, 0.1, 1, 10, 100]:
    # etc.
```

```
answers['Q7'] = bers + [bestC] + [ber]
```

```
assertFloatList(answers['Q7'], 7)
```

```
### 8
```

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

```
# 75/25% train/test split
dataTrain = dataset[:15000]
dataTest = dataset[15000:]
```

```
# A few utilities

itemAverages = defaultdict(list)
ratingMean = []

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

for i in itemAverages:
    itemAverages[i] = sum(itemAverages[i]) / len(itemAverages[i])

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

```
reviewsPerUser = defaultdict(list)
usersPerItem = defaultdict(set)

for d in dataTrain:
    u,i = d['user_id'], d['book_id']
    reviewsPerUser[u].append(d)
    usersPerItem[i].add(u)
```

```
# From my HW2 solution, welcome to reuse
def predictRating(user,item):
    ratings = []
    similarities = []
    for d2 in reviewsPerUser[user]:
        i2 = d2['book_id']
        if i2 == item: continue
        ratings.append(d2['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
        if item in itemAverages:
            return itemAverages[item]
        else:
            return ratingMean
```

```

```

```
answers['Q8'] = MSE(predictions, labels)
```

```
assertFloat(answers['Q8'])
```

```
### 9
```

```

```

```
for d in dataTest:
    # etc.
```

```

```

```
answers['Q9'] = [mse0, mse1to5, mse5]
```

```
assertFloatList(answers['Q9'], 3)
```

```
### 10
```

```

```

```
answers['Q10'] = ('describe your solution', itsMSE)
```

```
assert type(answers['Q10'][0]) == str
assertFloat(answers['Q10'][1])
```

```
f = open('answers_midterm.txt', 'w')
f.write(str(answers) + '\n')
f.close()
```

```

```