Section 1 (Regression)

```python
In [25]:
import gzip
import sklearn
from sklearn import linear_model
from collections import defaultdict
import random
import math
import numpy as np
import matplotlib.pyplot as plt

In [5]:
def parse(f):
    for l in open(f):
        yield eval(l)

In [193]:
dataset = list(parse("goodreads_reviews_comics_graphic.json"))

In [7]:
len(dataset)

Out[7]: 542338

In [37]:
dataset[1]

Out[37]: {'user_id': 'bafc2d50014200cda7cb2b6acd60cd73',
          'book_id': '6315584',
          'review_id': '72f1229aba5a88f9e72f0dcdc007dd22',
          'rating': 4,
          'review_text': "I've never really liked Spider-Man. I am, however, a huge fan of the Dresden Files. Jim Butcher is clever and sarcastic and probably the perfect choice to pen a superhero novel. I really enjoyed this book!",
          'date_added': 'Wed Aug 10 06:06:48 -0700 2016',
          'date_updated': 'Fri Aug 12 08:49:54 -0700 2016',
          'read_at': 'Fri Aug 12 08:49:54 -0700 2016',
          'started_at': 'Wed Aug 10 00:00:00 -0700 2016',
          'n_votes': 0,
          'n_comments': 0}
```
In [244]:
    
    def feature(d):
        dayFeat = [0]*7  # One hot encoding of day of week
        dayDict = {"Mon":0, "Tue":1, "Wed":2, "Thu":3, "Fri":4, "Sat":5, "Sun":6}
        dayFeat[dayDict[d['date_added'][:3]]] = 1
        return [1, d['rating'], d['n_comments']] + dayFeat[1:]

In [245]:
    
    X = [feature(d) for d in dataset]
    y = [len(d['review_text']) for d in dataset]

In [246]:
    
    model = sklearn.linear_model.LinearRegression()

In [247]:
    
    model.fit(X,y)

Out[247]:
    LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)

In [248]:
    
    yPred = model.predict(X)

In [249]:
    
    def MSE(predictions, labels):
        differences = [(x-y)**2 for x,y in zip(predictions,labels)]
        return sum(differences) / len(differences)

In [250]:
    
    mse = MSE(yPred, y)

In [251]:
    
    mse

Out[251]:
    624989.9720071985

**Question 1a/2a**

1(a)
Advantages: Simply discarding the outliers is convenient and efficient
Disadvantages: May cause bias to the data with a large loss of information and it may be hard to decide the right range for the outliers.

2(a)
In determining ymin and ymax for the outliers, I use interquartile range (iqr) and any review lengths which are less than 1.5\(iqr\) below the 25th percentile (a1) or larger than 1.5\(iqr\) above the 75th percentile is considered outliers.
I would still use MSE for evaluation.

```python
In [47]:
#2(a)
q1 = np.percentile(y, 25)
q3 = np.percentile(y, 75)
iqr = q3 - q1

y_min = q1 - 1.5 * iqr
y_max = q3 + 1.5 * iqr

# discard instances that are out of the range
Xy = list(zip(X, y))
no_outlier = [d for d in Xy if d[-1] >= y_min and d[-1] <= y_max]
X_no_outlier = [d[0] for d in no_outlier]
y_no_outlier = [d[-1] for d in no_outlier]

In [48]:
model = sklearn.linear_model.LinearRegression()
model.fit(X_no_outlier, y_no_outlier)

In [49]:
mse = MSE(yPred, y_no_outlier)
mse
Out[49]: 93974.44621257373
```

**Question 1b/2b**

**1(b)**

Advantages: Effectively rescale the data without the loss of information
Disadvantages: Sometimes the data may not be in the domain of the transformation function we choose (e.g. zero or negative for log transformation) so we need some extra adjustment to work around.

I will use MSE as evaluation with the log-transformed y as labels.

```python
In [288]:
#2(b)
# transform y
y_transformed = [np.log(d) if d != 0 else 0 for d in y]
```
In [289]:
model = sklearn.linear_model.LinearRegression()
model.fit(X, y_transformed)
yPred = model.predict(X)

In [290]:
model = sklearn.linear_model.LinearRegression()
model.fit(X, y_transformed)
yPred = model.predict(X)

Out[290]: 1.9344344595510008

Question 1c/2c

1(c)
Advantages: Easy to implement and no loss of information
Disadvantages: We will get a label indicating the rough range of the values we want to predict instead of an exact prediction

2(c)
Since this is a classification problem, I will use logistic regression as the model, which will be more accurate than linear regression in classification.
I will use balanced error rate as the evaluation.

In [296]:
#2(c)
#convert y to binary values
median_y = np.median(y)
y_binary = [1 if d > median_y else 0 for d in y]

In [297]:
model = linear_model.LogisticRegression()
model.fit(X, y_binary)
yPred = model.predict(X)

/Users/wangjiayun/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
FutureWarning)
Question 1d/2d

1(d)
Advantage: Robust to outliers without penalizing large errors more and no loss of information
Disadvantages: The objective we choose maybe not convex or not differentiable (e.g. MAE not continuously differentiable), which causes difficulty in implementation

2(d)
I choose to use mean absolute error as the objective and uses the quantile regression with quantile=0.5 for this least absolute deviation problem.
Since I used MAE as the loss function in regression, I will use MAE as the evaluation.
#2(d)

```python
import statsmodels.api as sm
import statsmodels.formula.api as smf
import pandas as pd

def MAE(predictions, labels):
    differences = [abs(x-y) for x,y in zip(predictions, labels)]
    return sum(differences) / len(differences)

data = pd.DataFrame(X, columns = ['X%s' % s for s in range(1, 10)])
data['y'] = y
data[:10]
```

Out[281]:

<table>
<thead>
<tr>
<th></th>
<th>X1</th>
<th>X2</th>
<th>X3</th>
<th>X4</th>
<th>X5</th>
<th>X6</th>
<th>X7</th>
<th>X8</th>
<th>X9</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3825</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>205</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>620</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1171</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3190</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>78</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>626</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>278</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>107</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>467</td>
</tr>
</tbody>
</table>

#quantile regression with quantile = 0.5

```python
formula = "y ~ 1 + " + " + ".join(["X%s" % s for s in range(2, 10)])
formula
```

Out[283]: 'y ~ 1 + X2 + X3 + X4 + X5 + X6 + X7 + X8 + X9'
Question 3

```
In [300]:
    1 mod = smf.quantreg(formula, data)
    2 res = mod.fit(q=.5)

In [301]:
    1 yPred = res.predict()
    2 MAE(yPred, y)

Out[301]: 400.3522762296385
```
3. Since for \(|\theta_0 - y_i|\) we have
\[ |\theta_0 - y_i| = \begin{cases} \theta_0 - y_i, & \theta_0 > y_i \\ y_i - \theta_0, & \theta_0 \leq y_i \end{cases} \]

\[ \Rightarrow \text{We can remove the absolute value sign to make MAE differentiable} \]

Suppose \(\theta_0\) is larger than \(\alpha\) of all \(y\) values and smaller than or equal to \(\beta\) of all \(y\) values. \(\alpha + \beta = N\)

\[ \text{MAE} = f(\theta_0) = \frac{1}{N} \sum_{i=1}^{N} |\theta_0 - y_i| \]
\[ = \frac{1}{N} \left( \sum_{i=1}^{\beta} (\theta_0 - y_i) + \sum_{i=1}^{\alpha} (y_i - \theta_0) \right) \]
\[ = \frac{1}{N} \left( \sum_{i=1}^{\beta} y_i - \sum_{i=1}^{\alpha} y_i + (\alpha - \beta) \theta_0 \right) \]

\[ \frac{df}{d\theta_0} = \frac{1}{N} (\alpha - \beta) = 0 \Rightarrow \alpha = \beta \]

Since \(\alpha + \beta = N\), \(\alpha = \beta = N/2\)

\[ \Rightarrow \text{When MAE is minimized, } \theta_0 \text{ is larger than half of the } y \text{ values and smaller than or equal to half of the } y \text{ values.} \]

\[ \Rightarrow \theta_0 \text{ is the median of all } y \text{ values.} \]
In [ ]: 1

### Question 4

(a) Possible features to collect: Time of the day, day of the week, if the trip is conducted on festival, weather, passenger flow at the start and end location, historical ratings of the driver

(b) We can use one-hot encoding to represent the level of time of the day, day of week, if festival and different types of weather. For example, if we have a 3:15am trip, which is originally indicated as 3:15:00, we will represent it as [0 0 0 1 0 ... 0]. Here we discard the information of minutes and seconds so we are representing the variation on a hourly basis. Passenger flow and ratings are numerical features that can be represented by their original values directly.

A useful transformation of the features could be the distance between start and end location (Euclidean distance between coordinates).

(c) I might set up the problem as a classification task. Trips that are of busy days or start and end at locations of high traffic flow tends to have high tips and vice versa, and people usually have a custom in giving tips (e.g. 20% of the total fee when they are willing to give high and 5% when they decide to give low), so the data tends to be clustered. Therefore, I may first transform the tips (labels) into percentage of the total fee (divide it by the total fee of the corresponding trip) and then dividing it into different levels according to some range.

### Section 2 (Classification)

### Question 5

Ordinary linear regression make predictions according to the best-fit line in terms of MSE, which tends to work well for continuous data but not as well if we have a classification problem of probabilistic data. For example, for the case below when a small portion of the data of a label are far away from the majority, the line tends to be pulled by those points and thus misclassify a lot of instances, while logistic regression will fit the data relatively better and thus higher accuracy.
Question 6

For this experiment, I will use the dataset of section 1 with number of votes, number of comments and day of week as the features, the length of the review text as the label. We will try to predict if the review text length is larger than its median. If so, we label it 1 and else -1. By evaluating the performance of linear regression and logistic regression with accuracy and balanced error rate, we can see that logistic regression outperforms linear regression.
In [175]: 1 #Linear Regression

In [373]: 1 def feature(d):
2     dayFeat = [0]*7 # One hot encoding of day of week
3     dayDict = {"Mon":0, "Tue":1, "Wed":2, "Thu":3, "Fri":4, "Sat":5, "Sun":6}
4     dayFeat[dayDict[d["date_added"][:3]]] = 1
5     return [1, d["n_votes"], d["n_comments"]]+ dayFeat[1:]

In [381]: 1 data_train = dataset[0:len(dataset)//2]
2 data_test = dataset[len(dataset)//2:]
3 median_length = np.median([len(d["review_text"]) for d in dataset])
4 X_train = [feature(d) for d in data_train]
5 y_train = [1 if len(d["review_text"]) > median_length else -1 for d in data_train]
6 X_test = [feature(d) for d in data_test]
7 y_test = [1 if len(d["review_text"]) > median_length else -1 for d in data_test]

In [375]: 1 model = sklearn.linear_model.LinearRegression()
2 model.fit(X_train, y_train)
3 yPred = model.predict(X_test)
4 yPred = [1 if p > 0 else -1 for p in yPred]

In [376]: 1 accuracy = sum(np.array(yPred) == y_test) / len(yPred)
2 print("Accuracy = ", accuracy)

Accuracy = 0.5456744686892676
In [377]:
1 yPred = [p if p != -1 else 0 for p in yPred]
2 y_test = [p if p != -1 else 0 for p in y_test]

3 TP = sum(np.logical_and(yPred, y_test))
4 TN = sum(np.logical_and(np.logical_not(yPred), np.logical_not(y_test)))
5 FP = sum(np.logical_and(yPred, np.logical_not(y_test)))
6 FN = sum(np.logical_and(np.logical_not(yPred), y_test))

7 print("BER = ", 1 - 0.5 * (TP / (TP + FN) + TN / (TN + FP)))

BER =  0.453988228997792

In [176]:
1 #Logistic Regression

In [382]:
1 mod = linear_model.LogisticRegression()
2 mod.fit(X_train, y_train)

/Users/wangjiayun/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)

Out[382]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                     intercept_scaling=1, max_iter=100, multi_class='warn',
                     n_jobs=None, penalty='l2', random_state=None, solver='warn',
                     tol=0.0001, verbose=0, warm_start=False)

In [383]:
1 yPred = mod.predict(X_test)

In [384]:
1 accuracy = sum(np.array(yPred) == y_test) / len(yPred)
2 print("Accuracy = ", accuracy)

Accuracy =  0.5565237914363368
Section 3 (Recommender System)

```python
In [194]:
   # Utility data structures
   reviewsPerUser = defaultdict(list)
   reviewsPerItem = defaultdict(list)
   usersPerItem = defaultdict(set) # U_i from class slides
   itemsPerUser = defaultdict(set) # I_u from class slides

In [195]:
   for d in dataset:
      user, item = d['user_id'], d['book_id']
      reviewsPerUser[user].append(d)
      reviewsPerItem[item].append(d)
      usersPerItem[item].add(user)
      itemsPerUser[user].add(item)

In [196]:
   ratingMean = sum([d['rating'] for d in dataset]) / len(dataset)

In [197]:
   ratingMean

Out[197]: 3.778138356523054

In [198]:
   def Jaccard(s1, s2):
      numer = len(s1.intersection(s2))
      denom = len(s1.union(s2))
      return numer / denom
```

BER = 0.44320594588937323
This function should be re-defined for each of your model variants

```python
In [199]:
def predictRating(user, item):
    ratings = []
    similarities = []
    for d in reviewsPerUser[user]:
        i2 = d['book_id']
        if i2 == item: continue
        ratings.append(d['rating'])
        similarities.append(Jaccard(usersPerItem[item], usersPerItem[i2]))
    if (sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings, similarities)]
        return sum(weightedRatings) / sum(similarities)
    else:
        # User hasn't rated any similar items
        return ratingMean
```

```python
In [200]:
dataset[1]
```

```json
Out[200]:
  {'user_id': 'bafc2d50014200cda7cb2b6ac60cd73',
   'book_id': '6315584',
   'review_id': '72f1229aba5a88f9e72f0dcdc007dd22',
   'rating': 4,
   'review_text': "I've never really liked Spider-Man. I am, however, a huge fan of the Dresden Files. Jim Butcher is clever and sarcastic and probably the perfect choice to pen a superhero novel. I really enjoyed this book!",
   'date_added': 'Wed Aug 10 06:06:48 -0700 2016',
   'date_updated': 'Fri Aug 12 08:49:54 -0700 2016',
   'read_at': 'Fri Aug 12 08:49:54 -0700 2016',
   'started_at': 'Wed Aug 10 00:00:00 -0700 2016',
   'n_votes': 0,
   'n_comments': 0}
```

```python
In [201]:
u,i = dataset[1]['user_id'], dataset[1]['book_id']
predictRating(u, i)
```

```python
Out[201]: 4.44493246042927
```

```python
In [202]:
sample = random.sample(dataset, 1000)
sampleLabels = [d['rating'] for d in sample]
```
In [203]:
# Baseline prediction
alwaysPredictMean = [ratingMean for d in sample]

In [204]:
# Prediction using item-to-item similarity above
cfPredictions = [predictRating(d['user_id'], d['book_id']) for d in sample]

In [205]:
# Baseline accuracy
MSE(alwaysPredictMean, sampleLabels)

Out[205]: 1.274820821512106

In [206]:
# Item-to-item similarity accuracy
MSE(cfPredictions, sampleLabels)

Out[206]: 1.035909467369955

Question 7(a)
Use user-to-user similarity

In [207]:
#predict in terms of user-to-user similarity

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

In [208]:
variant1Predictions = [predictRating(d['user_id'], d['book_id']) for d in sample]
In [209]: MSE(variant1Predictions, sampleLabels)

Out[209]: 1.2450572250218226

**Question 7(b)**

Change Jaccard similarity to user-to-user cosine similarity

```python
In [234]:
def cosine(v1, v2):
    n1, n2 = np.linalg.norm(v1), np.linalg.norm(v2)
    if n1 == 0 or n2 == 0:
        return 0
    return np.dot(v1, v2) / (np.linalg.norm(v1) * np.linalg.norm(v2))
```

```python
In [235]:
def predictRating(user, item):
    ratings = []
similarities = []
    for d in reviewsPerItem[item]:
        u2 = d["user_id"]
        if u2 == user: continue
        ratings.append(d["rating"])
        itemsBoth = itemsPerUser[user].intersection(itemsPerUser[u2])
        if not itemsBoth: continue
        similarities.append(0)
        similarity_user = [d["rating"] for d in reviewsPerUser[user] if d["book_id"] in itemsBoth]
        ratings_u2 = [d["rating"] for d in reviewsPerUser[u2] if d["book_id"] in itemsBoth]
        similarities.append(cosine(np.array(ratings_user), np.array(ratings_u2)))
    if (sum(similarities) > 0):
        weightedRatings = [(x*y) for x,y in zip(ratings, similarities)]
        return sum(weightedRatings) / sum(similarities)
    else:
        # User hasn't rated any similar items
        return ratingMean
```

```python
In [236]: variant2Predictions = [predictRating(d["user_id"], d["book_id"]) for d in sample]
```
In [237]: 1 MSE(variant2Predictions, sampleLabels)

Out[237]: 1.179325343204662

**Question 7(c)**

Subtract the mean rating and use item-to-item Jaccard similarity

In [239]:
    
    def predictRating(user, item):
        ratings = []
        similarities = []
        for d in reviewsPerUser[user]:
            i2 = d['book_id']
            if i2 == item: continue
            i2_mean = np.mean([d['rating'] for d in reviewsPerItem[i2]])
            # subtract the mean rating
            ratings.append(d['rating'] - i2_mean)
            similarities.append(jaccard(usersPerItem[item], usersPerItem[i2]))
        if (sum(similarities) > 0):
            item_mean = np.mean([d['rating'] for d in reviewsPerItem[item]])
            weightedRatings = [(x*y) for x, y in zip(ratings, similarities)]
            return item_mean + sum(weightedRatings) / sum(similarities)
        else:
            # User hasn't rated any similar items
            return ratingMean

In [240]:
    
    variant3Predictions = [predictRating(d['user_id'], d['book_id']) for d in sample]

In [241]:
    
    MSE(variant3Predictions, sampleLabels)

Out[241]: 0.7745985887261152

In [ ]: