CSE 158, Fall 2019: Homework 1

Name: CUI, HONGJIAN   **PID:** U08398995

Reading the the data

In [1]:
```
import csv
import numpy as np
import matplotlib.pyplot as plt
import string
from collections import defaultdict
```

In [2]:
```
path = "/home/cui/Projects/PycharmProjects/CSE-158/data/amazon_reviews_u
f = open(path)
```

In [3]:
```
reader = csv.reader(f, delimiter = "\t")
header = next(reader)
```

In [4]:
```
dataset = []
for line in reader:
    d = dict(zip(header, line))
    for field in ['helpful_votes', 'star_rating', 'total_votes']:
        d[field] = int(d[field])
    for field in ['verified_purchase', 'vine']:
        if d[field] == 'Y':
            d[field] = True
        else:
            d[field] = False
    dataset.append(d)
```

Tasks - Regression (week 1):

1. What is the distribution of ratings in the dataset?

In [5]:
```
ratingCounts = defaultdict(int)
```

In [6]:
```
for d in dataset:
    ratingCounts[d['star_rating']] += 1
```

Sort the dict to make it arranged by the key value.

In [7]:
```
ratingCounts = sorted(ratingCounts.items(), key=lambda x:x[0])
```

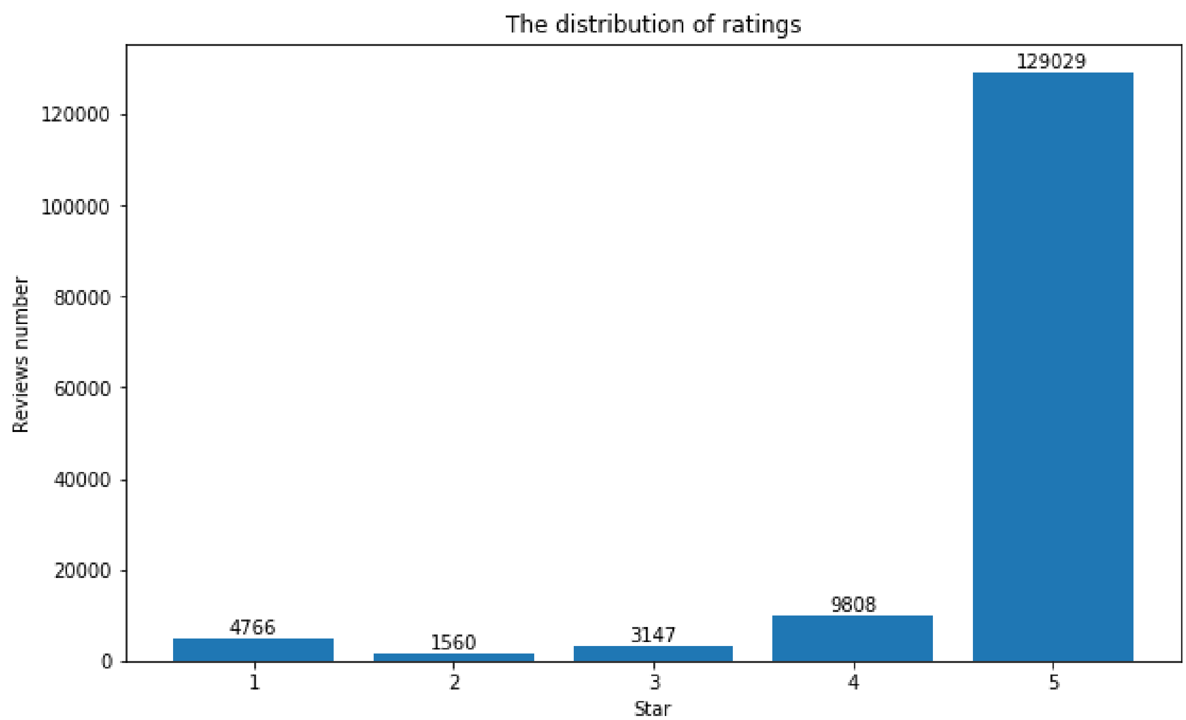
In [8]:
```python
ratingCounts
```
Out[8]:
```
[(1, 4766), (2, 1560), (3, 3147), (4, 9808), (5, 129029)]
```

In [9]:
```python
X = [p[0] for p in ratingCounts]
Y = [p[1] for p in ratingCounts]
```

In [10]:
```python
plt.figure(figsize=(10,6))
plt.bar(X, Y)

for x,y in enumerate(Y):
    plt.text(x + 1, y + 1000, y, ha='center')

plt.title("The distribution of ratings")
plt.xlabel("Star")
plt.ylabel("Reviews number")
plt.show()
```



2. Generate the distribution

In [11]:
```python
verifiedCounts = defaultdict(int)
```

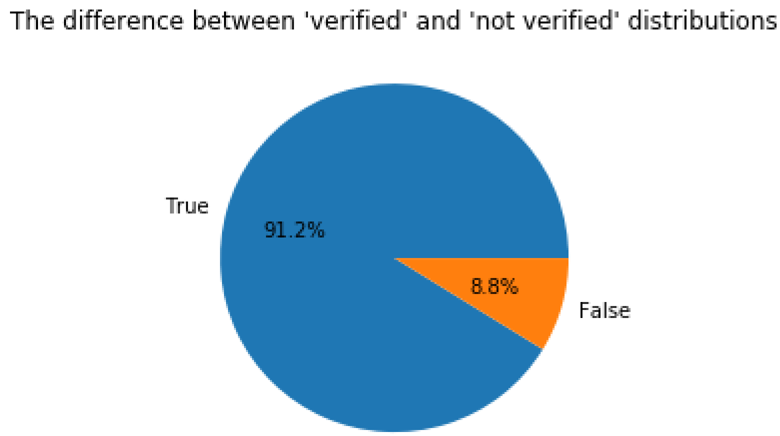
In [12]:
```python
for d in dataset:
    verifiedCounts[d['verified_purchase']] += 1
```

In [13]:
```python
verifiedCounts
```
Out[13]:
```
defaultdict(int, {True: 135289, False: 13021})
```

In [14]:
```
labels = verifiedCounts.keys()
x = verifiedCounts.values()
```

In [15]:
```
plt.axes(aspect = 1)
plt.pie(x = x, labels = labels, autopct = '%.1f%%')
plt.title("The difference between 'verified' and 'not verified' distribu
```

Out[15]:
```
Text(0.5, 1.0, "The difference between 'verified' and 'not verified' d
istributions")
```



3. Train a simple predictor to predict the star rating using two features:

$$star\ rating \approx \theta_0 + \theta_1 \times [reiview\ is\ verified] + \theta_2 \times [review\ length]$$

Report the values of θ_0, θ_1, and θ_2. Briefly describe your interpretationof these values.

In [16]:
```
def feature(datum):
    feat = [1]
    if datum['verified_purchase']==True:
        feat.append(1)
    else:
        feat.append(0)
    feat.append(len(datum['review_body']))

    return feat
```

In [17]:
```
X = [feature(d) for d in dataset]
```

In [18]:
```
X[-5:]
```

Out[18]:
```
[[1, 0, 267], [1, 0, 80], [1, 0, 296], [1, 0, 248], [1, 0, 583]]
```

In [19]:
```
y = [int(d['star_rating']) for d in dataset]
```

In [20]:
```python
y[:5]
```

Out[20]:
```
[5, 5, 5, 1, 5]
```

In [21]:
```python
theta, residuals, rank, s = np.linalg.lstsq(X, y, rcond=None)
```

In [22]:
```python
theta
```

Out[22]:
```
array([ 4.84503504e+00,  4.98580589e-02, -1.24545526e-03])
```

$$\theta_0 = 4.845,\ \theta_1 = 4.986,\ \theta_2 = -1.245$$

If the review is varified (the value of 'verified_purchase' is True), x = 1, else x = 0.

$$label(star\_rating) \approx$$
$$4.845 + 4.986 \times feature_1(verified\_purchase) - 1.245 \times feature_2(len(review\_body))$$

The coefficient of 'review length' is negative, then I look through the review in the dataset.

It seems that the number of good comment words are small. If the review has a big number of words, it may explain some reasons, which is more like a bad comment. So the longer the length of review, the lower the star rating is.

4. Train another predictor that only uses one feature

$$star\ rating \approx \theta_0 + \theta_1 \times [reiview\ is\ verified]$$

Report the values of θ_0 and θ_1 . Note that coefficient you found here might be quite different than the one from Q3, even though these coefficients refer to the same feature. Provide an explanation as to why these coefficients might vary so significantly.

In [23]:
```python
def feature(datum):
    feat = [1]
    if datum['verified_purchase']==True:
        feat.append(1)
    else:
        feat.append(0)

    return feat
```

In [24]:
```python
X = [feature(d) for d in dataset]
```

In [25]:
```python
X[:5]
```

Out[25]:
```
[[1, 1], [1, 1], [1, 1], [1, 1], [1, 1]]
```

In [26]:
```python
y = [int(d['star_rating']) for d in dataset]
```

In [27]:
```
y[:5]
```
Out[27]:
```
[5, 5, 5, 1, 5]
```

In [28]:
```
theta, residuals, rank, s = np.linalg.lstsq(X, y, rcond=None)
```

In [29]:
```
theta
```
Out[29]:
```
array([4.578143  , 0.16793392])
```

$\theta_0 = 4.578, \ \theta_1 = 0.168$

The value of feature "review length" is various, which has a great influence on the data. If we delete this feature, "review is verified" becomes the only factor to influence the result. Besides, most of data from the dataset is verified, so the coefficient of this feature will be small, which means the feature of "review is verified" plays a small role in this predictor.

5. Split the data into two fractions, train the same model as Q4 on the training set only. What is the model's MSE on the training and on the test set?

In [30]:
```
from sklearn import linear_model
```

In [31]:
```
len(X)
```
Out[31]:
```
148310
```

In [32]:
```
split = int(len(X) * 0.9)
```

In [33]:
```
split
```
Out[33]:
```
133479
```

In [34]:
```
X_train = X[:split]
X_test = X[split:]
y_train = y[:split]
y_test = y[split:]
```

In [35]:
```
model = linear_model.LinearRegression()
```

In [36]:
```
model.fit(X_train, y_train)
```
Out[36]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normali
ze=False)
```

In [37]:
```
train_predictions = model.predict(X_train)
test_predictions = model.predict(X_test)
```

In [38]:
```python
def MSE(predictions, targets):
    return ((predictions - targets) ** 2).mean()
```

In [39]:
```python
print ("The MSE of the training set is {:.3f}."
       .format(MSE(train_predictions, y_train)))
```

```
The MSE of the training set is 0.655.
```

In [40]:
```python
print("The MSE of the test set is {:.3f}."
      .format(MSE(test_predictions, y_test)))
```

```
The MSE of the test set is 0.972.
```

6. Using the test set from Q5, report the Mean Absolute Error (MAE) and R^2 coefficient for your predictor.

In [41]:
```python
def MAE(predictions, targets):
    return abs(predictions - targets).mean()
```

In [42]:
```python
print("The MAE of the test set is {:.3f}."
      .format(MAE(test_predictions, y_test)))
```

```
The MAE of the test set is 0.622.
```

In [43]:
```python
def R2(predictions, targets):
    return 1 - ((targets - predictions) ** 2).sum()
            / ((targets - np.mean(targets)) ** 2).sum()
```

In [44]:
```python
print("The R^2 of the test set is {:.3f}."
      .format(R2(test_predictions, y_test)))
```

```
The R^2 of the test set is -0.048.
```

Tasks - Classification (week 2):

8. Train a logistic regressor to make the above prediction.

$$p(review\ is\ verified) \approx \sigma(\theta_0 + \theta_1 \times [star\ rating] + \theta_2 \times [review\ length])$$

In [45]:
```python
def feature(datum):
    feat = [1]
    feat.append(datum['star_rating'])
    feat.append(len(datum['review_body']))

    return feat
```

In [46]:
```python
X = [feature(d) for d in dataset]
```

In [47]:
```
X[:5]
```
Out[47]:
```
[[1, 5, 38], [1, 5, 101], [1, 5, 4], [1, 1, 4], [1, 5, 76]]
```

In [48]:
```
y = [1 if d['verified_purchase'] == True else 0 for d in dataset]
```

In [49]:
```
y[:5]
```
Out[49]:
```
[1, 1, 1, 1, 1]
```

In [50]:
```
split = int(len(X) * 0.9)
```

In [51]:
```
X_train = X[:split]
X_test = X[split:]
y_train = y[:split]
y_test = y[split:]
```

In [52]:
```
model = linear_model.LogisticRegression()
```

In [53]:
```
model.fit(X_train, y_train)
```

```
/home/cui/anaconda3/lib/python3.7/site-packages/sklearn/linear_model/l
ogistic.py:432: FutureWarning: Default solver will be changed to 'lbfg
s' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

Out[53]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept
=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbo
se=0,
                   warm_start=False)
```

In [54]:
```
test_predictions = model.predict(X_test)
```

In [55]:
```
correctPredictionsTest = test_predictions == y_test
```

In [56]:
```
print("The accuracy of the test set is {:.4f}"
      .format(sum(correctPredictionsTest) / len(correctPredictionsTest)
```

```
The accuracy of the test set is 0.5598
```

In [57]:
```
label_pos = y_test.count(1) / len(y_test)
```

In [58]:
```
print("The propotion of labels that are positive is {:.3f}."
      .format(label_pos))
```

```
The propotion of labels that are positive is 0.560.
```

In [59]:
```
prediction_pos = np.sum(test_predictions == 1) / len(test_predictions)
```

In [60]:
```python
print("The propotion of predictions that are positive is {:.3f}."
      .format(prediction_pos))
```

```
The propotion of predictions that are positive is 0.999.
```

9. Considering same prediction problem as above, can you come up with a more accurate predictor?

Write down the feature vector you design, and report its train / test accuracy.

$p(review\ is\ verified) \approx$
$\sigma(\theta_0 + \theta_1 \times [review\ length]) + \theta_2 \times [number\ of\ popular\ words])$

I delete the feature "rating_star" and add a new feature "number of popular words", which is the number of words from the review body included in the popular words list. The computing method is that:

- Computing numbers of all words from the review body in the dataset.
- Filtering the words: delete the blank on both sides of the words, delete the punctuations, lower the words.
- Sort the word counts list by the number of times a word appears.
- Get the top 1000 words as the popularwords list.

In [61]:
```python
wordCounts = defaultdict(int)
punctuation = set(string.punctuation)
```

In [62]:
```python
def wordProcessor(word):
    word = word.lower()
    word = word.strip()
    return "".join(l for l in word if l not in punctuation)
```

In [63]:
```python
for d in dataset:
    c = d['review_body'].split(" ")
    for word in c:
        wordCounts[wordProcessor(word)] += 1
```

In [64]:
```python
popularWords = sorted(wordCounts.items(), key=lambda x:x[1])[-1000:]
```

In [65]:
```python
popularWords[-4:]
```

Out[65]:
```
[('i', 107966), ('gift', 114241), ('the', 130903), ('to', 144318)]
```

In [66]:
```python
popularWords = [word[0] for word in popularWords]
```

In [67]:
```python
popularWords[-5:]
```

Out[67]:
```
['a', 'i', 'gift', 'the', 'to']
```

In [68]:
```
def feature(datum):
    feat = [1]
    feat.append(len(datum['review_body']))

    c = datum['review_body'].split(" ")
    num = 0
    for word in c:
        if wordProcessor(word) in popularWords:
            num += 1
    feat.append(num)

    return feat
```

In [69]:
```
X = [feature(d) for d in dataset]
```

In [70]:
```
X[:5]
```

Out[70]:
```
[[1, 38, 7], [1, 101, 18], [1, 4, 1], [1, 4, 0], [1, 76, 14]]
```

In [71]:
```
y = [1 if d['verified_purchase'] == True else 0 for d in dataset]
```

In [72]:
```
y[:5]
```

Out[72]:
```
[1, 1, 1, 1, 1]
```

In [73]:
```
split = int(len(X) * 0.9)
```

In [74]:
```
X_train = X[:split]
X_test = X[split:]
y_train = y[:split]
y_test = y[split:]
```

In [75]:
```
model = linear_model.LogisticRegression(solver='lbfgs', multi_class='au
```

In [76]:
```
model.fit(X_train, y_train)
```

Out[76]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept
=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verb
ose=0,
                   warm_start=False)
```

In [77]:
```
train_predictions =  model.predict(X_train)
test_predictions = model.predict(X_test)
```

In [78]:
```
correctPredictionsTrain = train_predictions == y_train
correctPredictionsTest = test_predictions == y_test
```

In [79]:
```
print("The accuracy of the training set is {:.4f}"
      .format(sum(correctPredictionsTrain) / len(correctPredictionsTrai
```

```
The accuracy of the training set is 0.9511
```

In [80]:
```
print("The accuracy of the test set is {:.4f}."
      .format(sum(correctPredictionsTest) / len(correctPredictionsTest)
```

```
The accuracy of the test set is 0.5821.
```