

# Task -- Regression

First, let's see how ratings can be predicted as a function of (a) whether a review is a 'verified purchase', and (b) the length of the review (in characters).

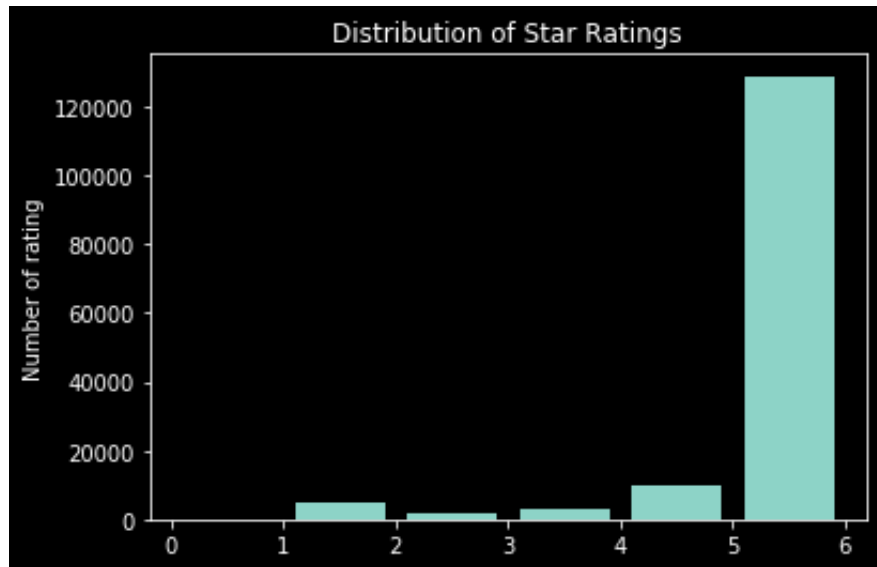
## Problem 1

What is the distribution of ratings in the dataset? That is, how many 1-star, 2-star, 3-star (etc.) reviews are there? You may write out the values or include a simple plot (1 mark).

```
1 # ignore the warnings from Pandas
2 import warnings
3 warnings.filterwarnings("ignore")
4
5 import matplotlib.pyplot as plt
6 import numpy as np
7 import pandas as pd
8
9 # read data and drop NA
10 data = pd.read_csv('./amazon_reviews_us_Gift_Card_v1_00.tsv',
11                    delimiter='\t')
12 print("# of Rows before drop NA value: ")
13 print(len(data))
14 data = data.dropna()
15 print("# of Rows after drop NA value: ")
16 print(len(data))
```

```
1 # of Rows before drop NA value:
2 148310
3 # of Rows after drop NA value:
4 148304
```

```
1 plt.hist(data['star_rating'], bins=[0,1,2,3,4,5,6], rwidth=0.8)
2 plt.xticks(range(0, 7))
3 plt.ylabel('Number of rating')
4 plt.title("Distribution of Star Ratings")
5 plt.show()
```



```
1 for i in range(1, 6):
2     print("The percent of %1d-star: %.1f%%" % (i,
3         (len(data[data['star_rating']==i])/len(data)*100)))
```

```
1 The percent of 1-star: 3.2%
2 The percent of 2-star: 1.1%
3 The percent of 3-star: 2.1%
4 The percent of 4-star: 6.6%
5 The percent of 5-star: 87.0%
```

We can see from the above plot and printout that most of ratings are 5-star (87%) while the least rating is 2-star (only 1%), which means this dataset is extremely unbalanced.

The dataset only has 6 rows containing NA value and we decide to remove them from the data.

## Problem 3

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

$\text{star\_rating} \approx \theta_0 + \theta_1 \times [\text{review is verified}] + \theta_2 \times [\text{review length}]$ .

Report the values of  $\theta_0$ ,  $\theta_1$ , and  $\theta_2$ . Briefly describe your interpretation of these values, i.e., what do  $\theta_0$ ,  $\theta_1$ , and  $\theta_2$  represent? Explain these in terms of the features and labels, e.g. if the coefficient of 'review length' is negative, what would that say about verified versus unverified reviews (1 mark)?

```
1 # Data preprocessing
2 data['verified_purchase_int'] = data.apply(lambda x:
3     int(x['verified_purchase'] == "Y"), axis = 1)
4 data['review_body_length'] = data.apply(lambda x: len(x['review_body']),
5     axis = 1)
6 data['theta_zero'] = 1
```

```

1 # Define My Own Regression
2 def myRegression(featureNames, labelName, data):
3     X, y = data[featureNames], data[labelName]
4     theta, residuals, rank, s = np.linalg.lstsq(X, y)
5     MSE = ((y - np.dot(X, theta))*2).mean()
6     for i in range(len(theta)):
7         print("Theta%d: %.5f" % (i, theta[i]))
8     print("MSE: %.3f" % MSE)
9
10 featureNames = ['theta_zero', 'verified_purchase_int',
11                 'review_body_length']
12 labelName = 'star_rating'
13 myRegression(featureNames, labelName, data)

```

```

1 Theta0: 4.84502
2 Theta1: 0.04987
3 Theta2: -0.00125
4 MSE: 0.651

```

We first convert "verified\_purchase" from "Y" and "N" to "1" and "0" and calculate the length of "review body" in character as features.

After define our own regression, we got three theta values  $\theta_0, \theta_1, \theta_2$  and MSE.

$\theta_0$  is a bias term, which means if there is no "verified\_purchase" and "review\_body" features, the predicted value of rating should be  $\theta_0$ .

$\theta_1$  means the relationship between "verified\_purchase" and "star\_rating". If the purchase is verified, then the rating will increase by  $\theta_1$ .

$\theta_2$  means the relationship between the length of "review\_body" in character and "star\_rating". If the length increase by 1, then the rating will increase by  $\theta_2$ .

In this case, an interesting fact is that  $\theta_2$  is a negative number, which means the more characters you write in your review, the lower rating you will rate this product. This fact is fun but reasonable since people will tend to write some bad reviews to complain when they are unsatisfied than to write some fancy words to praise when they are satisfied.

## Problem 4

Train another predictor that only uses one feature:

$$\text{star rating} \approx \theta_0 + \theta_1 \times [\text{review is verified}]$$

Report the values of  $\theta_0$  and  $\theta_1$ . Note that coefficient you found here might be quite different (i.e., much larger or smaller) than the one from Question 3, even though these coefficients refer to the same feature. Provide an explanation as to why these coefficients might vary so significantly (1 mark).<sup>1</sup>

```

1 featureNames = ['theta_zero', 'verified_purchase_int']
2 labelName = 'star_rating'
3 myRegression(featureNames, labelName, data)

```

```

1 Theta0: 4.57811
2 Theta1: 0.16796
3 MSE: 0.685

```

After removing the length of "review\_body" features, compared with problem 3, the value of  $\theta_0$  decreases from 4.845 to 4.578, and  $\theta_1$  increases from 0.0499 to 0.1679.

For  $\theta_0$ , this time it can be interpreted as the predicted value of rating score when the "verified\_purchase" feature is 0. Compared with problem 3, as we know, if the length of "review\_body" increases, the rating should decrease. So  $\theta_0$  in problem 3 should be bigger than the one in problem 4 since the length of "review\_body" is always bigger than or equal to 0 (so that it can offset the decrease aroused by review body).

For  $\theta_1$ , it still means the relationship between "verified\_purchase" and "star\_rating". If the purchase is verified, then the rating will increase by  $\theta_1$ . But, as we noticed, if "verified\_purchase" is 1, the predicted rating is  $\theta_0 + \theta_1$ , and since  $\theta_0$  decreases a lot, to compensate for this,  $\theta_1$  should increase accordingly.

## Problem 5

Split the data into two fractions – the first 90% for training, and the remaining 10% testing (based on the order they appear in the file). Train the same model as above on the training set only. What is the model's MSE on the training and on the test set (1 mark)?

```

1 def myRegression(featureNames, labelName, dataTrain, dataTest):
2     X, y = dataTrain[featureNames], dataTrain[labelName]
3     theta, residuals, rank, s = np.linalg.lstsq(X, y)
4     print("===== Training =====")
5     MSE = ((y - np.dot(X, theta))*2).mean()
6     for i in range(len(theta)):
7         print("Theta%d: %.5f" % (i, theta[i]))
8     print("MSE: %.3f" % MSE)
9     print("===== Testing =====")
10    X, y = dataTest[featureNames], dataTest[labelName]
11    MSE = ((y - np.dot(X, theta))*2).mean()
12    print("MSE: %.3f" % MSE)
13
14    def trainByRatio(ratio, data, featureNames, labelName):
15        train = data[:int(len(data)*ratio)]
16        test = data[int(len(data)*ratio):]
17        print("===== For ratio ", ratio, "=====")
18        myRegression(featureNames, labelName, train, test)

```

```
1 trainByRatio(0.9, data, featureNames, labelName)
```

```
1 ===== For ratio 0.9 =====
2 ===== Training =====
3 Theta0: 4.43958
4 Theta1: 0.31645
5 MSE: 0.656
6 ===== Testing =====
7 MSE: 0.972
```

## Problem 6

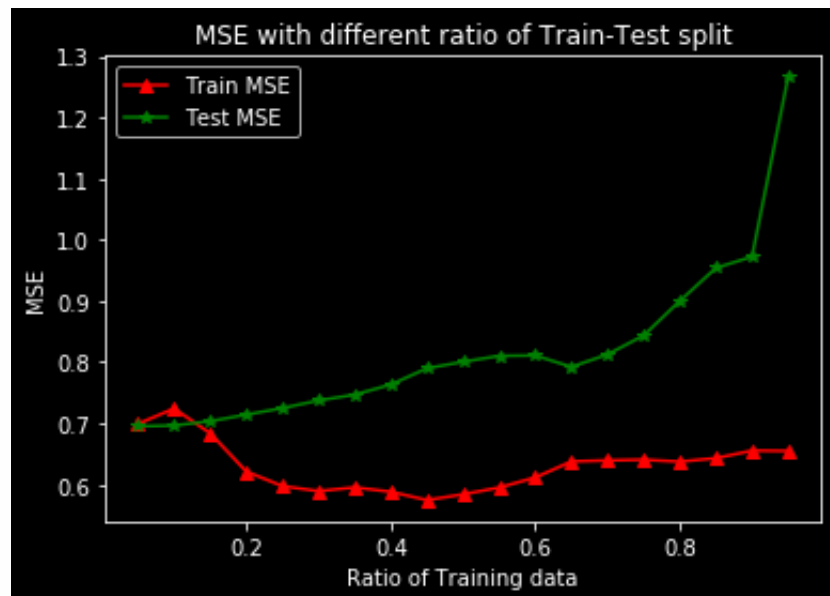
Repeat the above experiment, varying the size of the training and test fractions between 5% and 95% for training (using the complement for testing). Show how the training and test error vary as a function of the training set size (again using a simple plot or table). Does the size of the training set make a significant difference in testing performance? Comment on why it might or might not make a significant difference in this instance (2 marks).

```
1 # To plot a graph, let's revise the function slightly so that we can
  store the MSE in a list
2 def myRegression(featureNames, labelName, dataTrain, dataTest, trainMSE,
  testMSE):
3     # Training
4     X, y = dataTrain[featureNames], dataTrain[labelName]
5     theta, residuals, rank, s = np.linalg.lstsq(X, y)
6     MSE = ((y - np.dot(X, theta))**2).mean()
7     trainMSE.append(MSE)
8     # Testing
9     X, y = dataTest[featureNames], dataTest[labelName]
10    MSE = ((y - np.dot(X, theta))**2).mean()
11    testMSE.append(MSE)
12
13 def trainByRatio(ratio, data, featureNames, labelName, trainMSE,
  testMSE):
14     train = data[:int(len(data)*ratio)]
15     test = data[int(len(data)*ratio):]
16     myRegression(featureNames, labelName, train, test, trainMSE, testMSE)
17
18 trainMSE, testMSE = [], []
19 # ratio from 5% to 95%, step by 5%
20 ratios = [i/100 for i in list(range(5, 100, 5))]
21
22 for ratio in ratios:
23     trainByRatio(ratio, data, featureNames, labelName, trainMSE, testMSE)
24
25 # plot a graph
26 plt.plot(ratios, trainMSE, 'r^-', label='Train MSE')
27 plt.plot(ratios, testMSE, 'g*-', label='Test MSE')
```

```

28 plt.title('MSE with different ratio of Train-Test split')
29 plt.xlabel('Ratio of Training data')
30 plt.ylabel('MSE')
31 plt.legend()
32 plt.show()

```



Yes. As we can see from the plot, the size of the training set makes a significant difference in testing performance. As we increase the training size, however, the test performance decreases. This isn't normal and may be due to the extremely unbalanced nature of this dataset. The star rating label may vary a lot between the training set and testing set as the ratio increases.

The following plot proves our thought.

```

1  def calculatePortionOfFiveStars(ratio, data, trainPortion, testPortion):
2      train = data[:int(len(data)*ratio)]
3      test = data[int(len(data)*ratio):]
4
5      trainPortion.append(len(train[train['star_rating']==1])/len(train)*100)
6      testPortion.append(len(test[test['star_rating']==1])/len(test)*100)
7
8  trainPortion, testPortion = [], []
9  # ratio from 5% to 95%, step by 5%
10 ratios = [i/100 for i in list(range(5, 100, 5))]
11
12 for ratio in ratios:
13     calculatePortionOfFiveStars(ratio, data, trainPortion, testPortion)
14
15 # plot a graph
16 plt.plot(ratios, trainPortion, 'r^-', label='Training data')
17 plt.plot(ratios, testPortion, 'g*-', label='Testing data')
18 plt.title('% of 5-star ratings in Training/Testing data as ratio varies')
19 plt.xlabel('Ratio')
20 plt.ylabel('% of 5-star ratings')
21 plt.legend()

```



## Task -- Classification

In this question we'll alter the prediction from our regression task, so that we are now classifying whether a review is verified. Continue using the 90%/10% training and test sets you constructed previously, i.e., train on the training set and report the error/accuracy on the testing set.

### Problem 8

First, let's train a predictor that estimates whether a review is verified using the rating and the length:

$$p(\text{review is verified}) = \sigma(\theta_0 + \theta_1 \times [\text{star rating}] + \theta_2 \times [\text{review length}])$$

Train a logistic regressor to make the above prediction (you may use a logistic regression library with default parameters, e.g. `linear_model.LogisticRegression()` from `sklearn`). Report the classification accuracy of this predictor. Report also the proportion of labels that are positive (i.e., the proportion of reviews that are verified) and the proportion of predictions that are positive (1 mark).

```

1  # Define My Own Classification
2  from sklearn.linear_model import LogisticRegression
3
4  def myClassification(featureNames, labelName, dataTrain, dataTest):
5      X, y = dataTrain[featureNames], dataTrain[labelName]
6      clf = LogisticRegression().fit(X, y)
7      y_ = clf.predict(X)
8      print("==== Training =====")
9      print("Accuracy: ", clf.score(X, y))

```

```

10     print("Proportion of reviews that are verified: %.2f%%" %
(len(dataTrain[dataTrain[featureNames]==1])/len(dataTrain)*100))
11     print("Proportion of predictions that are positive: %.2f%%" %
(np.mean(y_==1)*100))
12     print("==== Testing =====")
13     X, y = dataTest[featureNames], dataTest[labelName]
14     y_ = clf.predict(X)
15     print("Accuracy: ", clf.score(X, y))
16     print("Proportion of reviews that are verified: %.2f%%" %
(len(dataTest[dataTest[featureNames]==1])/len(dataTest)*100))
17     print("Proportion of predictions that are positive: %.2f%%" %
(np.mean(y_==1)*100))
18
19 def trainByRatio(ratio, data, featureNames, labelName):
20     train = data[:int(len(data)*ratio)]
21     test = data[int(len(data)*ratio):]
22     print("==== For ratio ", ratio, "====")
23     myClassification(featureNames, labelName, train, test)
24
25 featureNames = ['theta_zero', 'star_rating', 'review_body_length']
26 labelName = 'verified_purchase_int'
27 ratio = 0.9
28 trainByRatio(ratio, data, featureNames, labelName)

```

```

1  ===== For ratio  0.9 =====
2  ===== Training =====
3  Accuracy:  0.9511736456062275
4  Proportion of reviews that are verified: 100.00%
5  Proportion of predictions that are positive: 99.96%
6  ===== Testing =====
7  Accuracy:  0.5597734475085968
8  Proportion of reviews that are verified: 100.00%
9  Proportion of predictions that are positive: 99.90%

```

## Problem 9

Considering same prediction problem as above, can you come up with a more accurate predictor (e.g. using features from the text, timestamp, etc.)? Write down the feature vector you design, and report its train/test accuracy (1 mark).

```

1  # Let's do some analysis for other features first
2  print(data['marketplace'].unique())
3  print(data['product_category'].unique())
4  print(data['vine'].unique())

```



```

1 ['US']
2 ['Gift Card']
3 ['N']

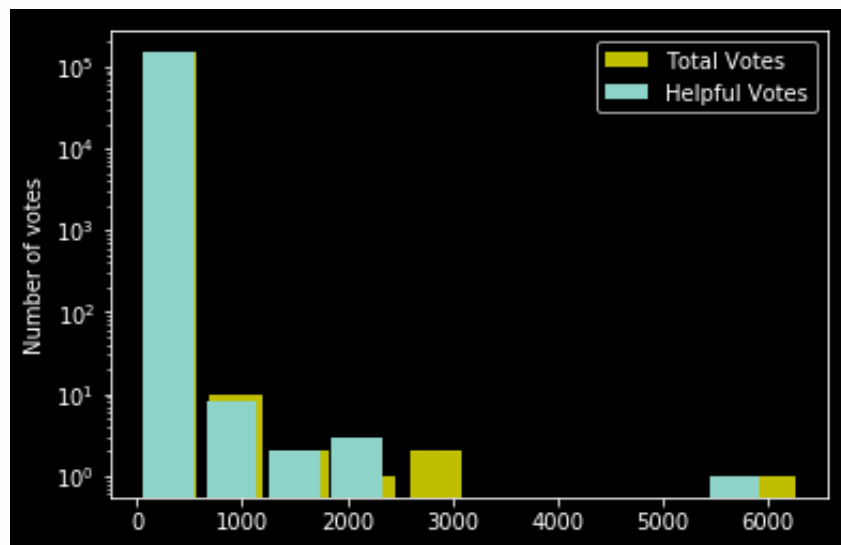
```

As we can see from above, there are no big differences between these features.

```

1 plt.hist(data['total_votes'], log=True, color='y',rwidth=0.8, label='Total
  Votes')
2 plt.hist(data['helpful_votes'], log=True, rwidth=0.8, label='Helpful
  Votes')
3 plt.ylabel('Number of votes')
4 plt.legend()
5 plt.show()

```



```

1 featureNames = ['theta_zero', 'star_rating', 'review_body_length',
  'total_votes', 'helpful_votes']
2 labelName = 'verified_purchase_int'
3 ratio = 0.9
4 trainByRatio(ratio, data, featureNames, labelName)

```

```

1 ===== For ratio 0.9 =====
2 ===== Training =====
3 Accuracy: 0.9512111063660815
4 Proportion of reviews that are verified: 100.00%
5 Proportion of predictions that are positive: 99.96%
6 ===== Testing =====
7 Accuracy: 0.5601105791922325
8 Proportion of reviews that are verified: 100.00%
9 Proportion of predictions that are positive: 99.80%

```

By adding 'total\_votes' and 'helpful\_votes' as features, the accuracy increased a little bit. But that's not enough.

In problem 7 and problem 8, we both noticed that the distribution of our dataset is extremely unbalanced.

One realistic solution is to shuffle the dataset so that the distribution can be more balanced! And the results suddenly become promising. Congrats!

```
1 data = data.sample(frac=1)
2 featureNames = ['theta_zero', 'star_rating', 'review_body_length',
3                 'total_votes', 'helpful_votes']
4 labelName = 'verified_purchase_int'
5 ratio = 0.9
6 trainByRatio(ratio, data, featureNames, labelName)
```

```
1 ===== For ratio 0.9 =====
2 ===== Training =====
3 Accuracy: 0.9102664958456017
4 Proportion of reviews that are verified: 100.00%
5 Proportion of predictions that are positive: 99.62%
6 ===== Testing =====
7 Accuracy: 0.9135594363158249
8 Proportion of reviews that are verified: 100.00%
9 Proportion of predictions that are positive: 99.58%
```