In [1]:
```
import numpy as np
import ast
import urllib
import random
import scipy.optimize
from sklearn import linear_model # For performing logistic regression
```

In [2]:
```
def parseDataFromFile(fname):
    for l in open(fname):
        yield ast.literal_eval(l)
```

In [3]:
```
data = list(parseDataFromFile("C:\\Users\\ramasarma\\Documents\\UCSD\\Fall 2020\\CSE 258\\Homework1\\fantasy_10000.js
```

In [4]:
```
def feature(datum):
    max_len = -1
    for d in datum:
        max_len = max(len(d['review_text']), max_len)
    print("Max length is {}".format(max_len))
    feat = [[1, len(d['review_text'])/max_len] for d in datum]
    return feat
```

In [5]:
```
X_2 = np.matrix(feature(data))
y_2 = np.asarray([d['rating'] for d in data])
```

```
Max length is 14306
```

Training for Question 2

In [6]:
```
theta_2, residuals_2, rank_2, s_2 = np.linalg.lstsq(X_2, y_2, rcond=-1)
print(theta_2)

#Cross verification
X = np.matrix(X_2)
y = np.matrix(y_2)
np.linalg.inv(X.T*X)*X.T*y.T
```

```
[3.68568136 0.98335392]
```

Out[6]:
```
matrix([[3.68568136],
```

```
        [0.98335392]])
```

In [7]:
```
y_prediction_2 = np.asarray(X.dot(theta_2))
print(y_prediction_2.shape)
print(y_2.shape)
```

```
(1, 10000)
(10000,)
```

In [8]:
```
MSE_2 = np.mean(np.square(y_prediction_2 - y_2))
print(MSE_2)
```

```
1.5522086622355378
```

Due to normalization of the second feature based on the length of the max review, we get the value of theta 1 very close to 1

$$\theta_0 = 3.68568136, \theta_1 = 0.98335392, MSE = 1.5522086622355378$$

Training for question 3

In [9]:
```
def feature_q3(datum):
    max_len = -1
    for d in datum:
        max_len = max(len(d['review_text']), max_len)
    feat = [[1, (len(d['review_text'])/max_len), float(d['n_comments'])] for d in datum]
    return feat
```

In [10]:
```
X_3 = np.matrix(feature_q3(data))
y_3 = [d['rating'] for d in data]
```

In [11]:
```
theta_3, residuals_3, rank_3, s_3 = np.linalg.lstsq(X_3, y_3, rcond = -1)
print(theta_3)

#Cross verification
X = np.matrix(X_3)
y = np.matrix(y_3)
np.linalg.inv(X.T*X)*X.T*y.T
```

```
[ 3.68916737  1.08497776 -0.03279289]
```

Out[11]:
```
matrix([[ 3.68916737],
        [ 1.08497776],
        [-0.03279289]])
```

In [12]:
```
y_prediction_3 = X_3.dot(theta_3)
MSE = np.mean(np.square(y_prediction_3 - y_3))
print(MSE)
```

```
1.5498351692774583
```

$$\theta_0 = 3.68916737, \theta_1 = 1.08497776, \theta_2 = -0.03279289, MSE = 1.5498351692774581$$

By adding a new feature in the form of the number of comments, we are making rating depend on another factor in the equation. This component (number of comments) is negatively correlated to the rating as θ_2 is negative. To compensate for that, the coefficient for the length of the review, θ_1 has increased

Training for question 4 (a) - First order polynomial

In [13]:
```
def feature_q4(datum):
    max_len = -1
    for d in datum:
        max_len = max(len(d['review_text']), max_len)
    print("Max length = {}".format(max_len))
    feat = [[1, len(d['review_text'])/max_len] for d in datum]
    return feat
```

In [14]:
```
X_4a = np.matrix(feature_q4(data))
y_4a = [d['rating'] for d in data]

theta_4a, residuals_4a, rank_4a, s_4a = np.linalg.lstsq(X_4a, y_4a,rcond=-1)
print(theta_4a)
y_prediction_4a = X_4a.dot(theta_4a)
MSE_4a = np.mean(np.square(y_prediction_4a - y_4a))
print(MSE_4a)

#Cross verification
X = np.matrix(X_4a)
```

```
y = np.matrix(y_4a)
np.linalg.inv(X.T*X)*X.T*y.T
```

```
Max length = 14306
[3.68568136 0.98335392]
1.5522086622355378
```

Out[14]:
```
matrix([[3.68568136],
        [0.98335392]])
```

$$\theta_0 = 3.65975869, \theta_1 = 0.98335392, MSE = 1.5506567696339388$$

Training for Question 4 (b) - Quadratic polynomial

In [15]:
```
def feature_q4(datum):
    max_len = -1
    for d in datum:
        max_len = max(len(d['review_text']), max_len)
    print("Max length = {}".format(max_len))
    feat = [[1, len(d['review_text'])/max_len, (len(d['review_text'])/max_len) ** 2] for d in datum]
    return feat
```

In [16]:
```
X_4b = np.matrix(feature_q4(data))
y_4b = [d['rating'] for d in data]

theta_4b, residuals_4b, rank_4b, s_4b = np.linalg.lstsq(X_4b, y_4b,rcond=-1)
print(theta_4b)
y_prediction_4b = X_4b.dot(theta_4b)
MSE_4b = np.mean(np.square(y_prediction_4b - y_4b))
print(MSE_4b)

#Cross verification
X = np.matrix(X_4b)
y = np.matrix(y_4b)
np.linalg.inv(X.T*X)*X.T*y.T
```

```
Max length = 14306
[ 3.65975869  1.8395413  -2.62503319]
1.5506567696339388
```

Out[16]:
```
matrix([[ 3.65975869],
        [ 1.8395413 ],
        [-2.62503319]])
```

Answer to question 4(b) - Quadratic polynomial

$$\theta_0 = 3.65975869, \theta_1 = 1.8395413, \theta_2 = -2.62503319, MSE = 1.5506567696339388$$

Training for Question 4 (c) - Third degree polynomial

In [17]:

```
def feature_q4(datum):
    max_len = -1
    for d in datum:
        max_len = max(len(d['review_text']), max_len)
    print("Max length = {}".format(max_len))
    feat = [[1, len(d['review_text'])/max_len, (len(d['review_text'])/max_len) ** 2, \
             (len(d['review_text'])/max_len) ** 3] for d in datum]
    return feat
```

In [18]:

```
X_4c = np.matrix(feature_q4(data))
y_4c = [d['rating'] for d in data]
theta_4c, residuals_4c, rank_4c, s_4c = np.linalg.lstsq(X_4c, y_4c,rcond=-1)
print(theta_4c)
y_prediction_4c = X_4c.dot(theta_4c)
MSE_4c = np.mean(np.square(y_prediction_4c - y_4c))
print(MSE_4c)

#Cross verification
X = np.matrix(X_4c)
y = np.matrix(y_4c)
np.linalg.inv(X.T*X)*X.T*y.T
```

```
Max length = 14306
[ 3.63659658  2.8884065  -8.48042966  6.12504475]
1.549798532380553
```

Out[18]:

```
matrix([[ 3.63659658],
        [ 2.8884065 ],
        [-8.48042966],
        [ 6.12504475]])
```

Answer to question 4(c) - Cubic polynomial

$$\theta_0 = 3.63659658, \theta_1 = 2.8884065, \theta_2 = -8.48042966, \theta_3 = 6.12504475, MSE = 1.549798532380553$$

Training for Question 4(d) - Biquadratic polynomial

In [19]:
```
def feature_q4(datum):
    max_len = -1
    for d in datum:
        max_len = max(len(d['review_text']), max_len)
    #print("Max length = {}".format(max_len))
    feat = [[1, (len(d['review_text']) / max_len), (len(d['review_text']) / max_len) ** 2, \
             (len(d['review_text']) / max_len) ** 3, (len(d['review_text']) / max_len) ** 4] for d in datum]
    return feat
```

In [20]:
```
X_4d = np.matrix(feature_q4(data))
#print(X_4d[:12])
y_4d = [d['rating'] for d in data]
theta_4d, residuals_4d, rank_4d, s_4d = np.linalg.lstsq(X_4d, y_4d,rcond=-1)
print(theta_4d)
y_prediction_4d = X_4d.dot(theta_4d)
MSE_4d = np.mean(np.square(y_prediction_4d - y_4d))
print(MSE_4d)

#Cross verification
X = np.matrix(X_4d)
y = np.matrix(y_4d)
np.linalg.inv(X.T*X)*X.T*y.T
```

```
[  3.64736873   2.20419719  -1.80763945 -11.6451833   12.21844408]
1.5496291324524714
```

Out[20]:
```
matrix([[  3.64736873],
        [  2.20419719],
        [ -1.80763945],
        [-11.6451833 ],
        [ 12.21844408]])
```

Answer to Question 4(d) - Biquadratic polynomial

$$\theta_0 = 3.64736873, \theta_1 = 2.20419719, \theta_2 = -1.80763945, \theta_3 = -11.6451833, \theta_4 = 12.21844408$$

$$MSE = 1.5496291324524718$$

In [21]:
```
def feature_q4(datum):
    max_len = -1
    for d in datum:
        max_len = max(len(d['review_text']), max_len)
    #print("Max length = {}".format(max_len))
    feat = [[1, len(d['review_text']) / max_len, (len(d['review_text']) / max_len) ** 2, \
             (len(d['review_text']) / max_len) ** 3, (len(d['review_text']) / max_len) ** 4, \
             (len(d['review_text']) / max_len) ** 5] for d in datum]
    return feat
```

In [22]:
```
X_4e = np.matrix(feature_q4(data))
y_4e = [d['rating'] for d in data]
theta_4e, residuals_4e, rank_4e, s_4e = np.linalg.lstsq(X_4e, y_4e,rcond=-1)
print(theta_4e)
y_prediction_4e = X_4e.dot(theta_4e)
MSE_4e = np.mean(np.square(y_prediction_4e - y_4e))
print(MSE_4e)

#Cross verification
X = np.matrix(X_4e)
y = np.matrix(y_4e)
np.linalg.inv(X.T*X)*X.T*y.T
```

```
[  3.6441158    2.47396326  -5.65441081   5.55309592 -15.94637484
  14.68100179]
1.5496142023298694
```

Out[22]:
```
matrix([[  3.6441158 ],
        [  2.47396326],
        [ -5.65441081],
        [  5.55309592],
        [-15.94637484],
        [ 14.68100179]])
```

Answer to Question 4(e) - Fifth degree polynomial

$$\theta_0 = 3.6441158, \theta_1 = 2.47396326, \theta_2 = -5.65441081, \theta_3 = 5.55309592, \theta_4 = -15.94637484, \theta_5 = 14.68100179$$

$$MSE = 1.5496142023298691$$

In [23]:
```
random.shuffle(data)
length = len(data)
```

```
train_data = data[:length // 2]
print(len(train_data))
test_data = data[length // 2:]
print(len(test_data))
```

```
5000
5000
```

Question 5 (a) - After separation of Train and test data, cubic polynomial

In [24]:

```
def feature_q5(datum):
    max_len = -1
    for d in datum:
        max_len = max(len(d['review_text']), max_len)
    #print("Max length = {}".format(max_len))
    feat = [[1, len(d['review_text'])/max_len] for d in datum]
    return feat

X_5a = np.matrix(feature_q5(train_data))
y_5a = [d['rating'] for d in train_data]

theta_5a, residuals_5a, rank_5a, s_5a = np.linalg.lstsq(X_5a, y_5a,rcond=-1)
print("Training set theta is {}".format(theta_5a))
y_prediction_5a = X_5a.dot(theta_5a)
MSE_5a = np.mean(np.square(y_prediction_5a - y_5a))
print("Training set MSE is {}".format(MSE_5a))

X_test_5a = np.matrix(feature_q5(test_data))
y_test_5a = [d['rating'] for d in test_data]

theta_test_5a, residuals_test_5a, rank_test_5a, s_test_5a = np.linalg.lstsq(X_test_5a, y_test_5a, rcond = -1)
print("Test set theta is {}".format(theta_test_5a))
y_prediction_5a = X_test_5a.dot(theta_test_5a)
MSE_test_5a = np.mean(np.square(y_prediction_5a - y_5a))
print("Test set MSE is {}".format(MSE_test_5a))
```

```
Training set theta is [3.65620837 1.16446036]
Training set MSE is 1.5556330691585833
Test set theta is [3.71606714 0.74397102]
Test set MSE is 1.5663203941592045
```

In [25]:

```
def feature_q5(datum):
```

```python
    max_len = -1
    for d in datum:
        max_len = max(len(d['review_text']), max_len)
    feat = [[1, len(d['review_text'])/max_len, (len(d['review_text'])/max_len) ** 2] for d in datum]
    return feat

X_5b = np.matrix(feature_q5(train_data))
y_5b = [d['rating'] for d in train_data]

theta_5b, residuals_5b, rank_5b, s_5b = np.linalg.lstsq(X_5b, y_5b,rcond=-1)
print("Training set Theta is {}".format(theta_5b))
y_prediction_5b = X_5b.dot(theta_5b)
MSE_5b = np.mean(np.square(y_prediction_5b - y_5b))
print("Training set MSE is {}".format(MSE_5b))

X_test_5b = np.matrix(feature_q5(test_data))
y_test_5b = [d['rating'] for d in test_data]

theta_test_5b, residuals_test_5b, rank_test_5b, s_test_5b = np.linalg.lstsq(X_test_5b, y_test_5b, rcond = -1)
print("Test set Theta is {}".format(theta_test_5b))
y_prediction_5b = X_test_5b.dot(theta_test_5b)
MSE_test_5b = np.mean(np.square(y_prediction_5b - y_5b))
print("Test set MSE is {}".format(MSE_test_5b))
```

```
Training set Theta is [ 3.61574284  2.42947402 -3.58046569]
Training set MSE is 1.5518445548960906
Test set Theta is [ 3.70523471  1.11188901 -1.19085165]
Test set MSE is 1.567226201652561
```

In [26]:

```python
def feature_q5(datum):
    max_len = -1
    for d in datum:
        max_len = max(len(d['review_text']), max_len)
    feat = [[1, len(d['review_text'])/max_len, (len(d['review_text'])/max_len) ** 2, \
             (len(d['review_text'])/max_len) ** 3] for d in datum]
    return feat

X_5c = np.matrix(feature_q5(train_data))
y_5c = [d['rating'] for d in train_data]

theta_5c, residuals_5c, rank_5c, s_5c = np.linalg.lstsq(X_5c, y_5c,rcond=-1)
print("Training set Theta is {}".format(theta_5c))
```

```
y_prediction_5c = X_5c.dot(theta_5c)
MSE_5c = np.mean(np.square(y_prediction_5c - y_5c))
print("Training set MSE is {}".format(MSE_5c))

X_test_5c = np.matrix(feature_q5(test_data))
y_test_5c = [d['rating'] for d in test_data]

theta_test_5c, residuals_test_5c, rank_test_5c, s_test_5c = np.linalg.lstsq(X_test_5c, y_test_5c, rcond = -1)
print("Test set Theta is {}".format(theta_test_5c))
y_prediction_5c = X_test_5c.dot(theta_test_5c)
MSE_test_5c = np.mean(np.square(y_prediction_5c - y_5c))
print("Test set MSE is {}".format(MSE_test_5c))
```

```
Training set Theta is [ 3.58864168  3.61502431 -9.94455584  6.43271937]
Training set MSE is 1.5506903039600453
Test set Theta is [ 3.68556838  2.0039438  -6.20165702  5.24590139]
Test set MSE is 1.5696894680660314
```

In [27]:

```
def feature_q5(datum):
    max_len = -1
    for d in datum:
        max_len = max(len(d['review_text']), max_len)
    feat = [[1, len(d['review_text'])/max_len, (len(d['review_text'])/max_len) ** 2, \
             (len(d['review_text'])/max_len) ** 3, (len(d['review_text'])/max_len) ** 4] for d in datum]
    return feat

X_5d = np.matrix(feature_q5(train_data))
y_5d = [d['rating'] for d in train_data]

theta_5d, residuals_5d, rank_5d, s_5d = np.linalg.lstsq(X_5d, y_5d, rcond = -1)
print("Training set Theta is {}".format(theta_5d))
y_prediction_5d = X_5d.dot(theta_5d)
MSE_5d = np.mean(np.square(y_prediction_5d - y_5d))
print("Training set MSE is {}".format(MSE_5d))

X_test_5d = np.matrix(feature_q5(test_data))
y_test_5d = [d['rating'] for d in test_data]

theta_test_5d, residuals_test_5d, rank_test_5d, s_test_5d = np.linalg.lstsq(X_test_5d, y_test_5d, rcond=-1)
print("Test set Theta is {}".format(theta_test_5d))
y_prediction_5d = X_test_5d.dot(theta_test_5d)
```

```
MSE_test_5d = np.mean(np.square(y_prediction_5d - y_5d))
print("Test set MSE is {}".format(MSE_test_5d))
```

```
Training set Theta is [  3.5865655    3.74295108 -11.15260653   9.5554212   -2.1165591 ]
Training set MSE is 1.5506836965095383
Test set Theta is [  3.70878715   0.51503494   8.52285396 -34.32007284  26.714403  ]
Test set MSE is 1.571150789556881
```

In [28]:

```
def feature_q5(datum):
    max_len = -1
    for d in datum:
        max_len = max(len(d['review_text']), max_len)
    feat = [[1, len(d['review_text'])/max_len, (len(d['review_text'])/max_len) ** 2, \
             (len(d['review_text'])/max_len) ** 3, (len(d['review_text'])/max_len) ** 4, \
             (len(d['review_text'])/max_len) ** 5] for d in datum]
    return feat

X_5e = np.matrix(feature_q5(train_data))
y_5e = [d['rating'] for d in train_data]

theta_5e, residuals_5e, rank_5e, s_5e = np.linalg.lstsq(X_5e, y_5e, rcond = -1)
print("Training set theta is {}".format(theta_5e))
y_prediction_5e = X_5e.dot(theta_5e)
MSE_5e = np.mean(np.square(y_prediction_5e - y_5e))
print("Training set MSE is {}".format(MSE_5e))

X_test_5e = np.matrix(feature_q5(test_data))
y_test_5e = [d['rating'] for d in test_data]

theta_test_5e, residuals_test_5e, rank_test_5e, s_test_5e = np.linalg.lstsq(X_test_5e, y_test_5e, rcond = -1)
print("Test set theta is {}".format(theta_test_5e))
y_prediction_5e = X_test_5e.dot(theta_test_5e)
MSE_test_5e = np.mean(np.square(y_prediction_5e - y_5e))
print("Test set MSE is {}".format(MSE_test_5e))
```

```
Training set theta is [   3.57186562    4.92491771  -27.37364523   79.14320299 -111.86833959
   55.66379667]
Training set MSE is 1.550380631825618
Test set theta is [   3.72695202   -1.07083423   32.77971948 -151.17875316  231.51295213
 -110.71999797]
Test set MSE is 1.5713404476493256
```

TO BE COMPLETED: PROOF FOR QUESTION 6

PROOF

$$MSE = \frac{1}{N} * \sum_{i=1}^{i=N} (y_i - (\theta \cdot X))^2$$

In this scenario, $\theta \cdot X = \theta_0$ (Trivial predictor as already provided in the question).

$$MSE = \frac{1}{N} * \sum_{i=1}^{i=N} (y_i - \theta_0)^2$$

$$\frac{\partial(MSE)}{\partial\theta} = \frac{1}{N} * -2 * \sum_{i=1}^{i=N} (y_i - \theta_0) * N$$

As MSE is a convex function in θ, we achieve the minimum/maximum value at $\frac{\partial(MSE)}{\partial\theta} = 0$

By further solving the equation, we get

$$\sum_{i=1}^{i=N} (y_i) - N * \theta_0 = 0$$

This implies,

$$\sum_{i=1}^{i=N} (y_i) = N * \theta_0$$

and thus,

$$\theta_0 = \frac{1}{N} * \sum_{i=1}^{i=N} (y_i)$$

The RHS in the above equation is the average value of the label y. This value of θ_0 could lead to either maximum value or minimum value. Let's compute the second order derivative to prove that it's the minimum value

$$\frac{\partial^2(MSE)}{\partial\theta^2} = -2 * -1 * N = 2 * N$$

As the second order derivative is a positive number, we can conclude that $\theta_0 = \frac{1}{N} * \sum_{i=1}^{i=N} y_i$ is the point of minima. This value of θ_0 results in minimizing MSE and is equal to the average value of the label y.

CLASSIFICATION TASKS

In [29]:
```
## Loading the Beer reviews Dataset
data = list(parseDataFromFile("C:\\Users\\ramasarma\\Documents\\UCSD\\Fall 2020\\CSE 258\\Homework1\\beer_50000.json'
```

The following analysis (manual) has been done to find one of the features to reduce the Balanced Error Rate for the logistic regression based classifier

In [30]:
```
def feature(datum):
    feat = [[1, len(d['review/text'])] for d in datum if 'user/gender' in d]
    return feat
```

In [31]:
```
X = feature(data)
Y = [d['user/gender'] for d in data if 'user/gender' in d]
## Encoding the dataset properly (0 - Male and 1 - Female)
count_males = 0
count_females = 0
for i in range(len(Y)):
    if Y[i] == 'Male':
        Y[i] = 0
        count_males += 1
    else:
        Y[i] = 1
        count_females += 1

print("The total number of males and females are {} and {}".format(count_males, count_females))
# X_train = X[:len(X) // 2]
# Y_train = Y[:len(Y) // 2]
```

```
# X_test = X[len(X) // 2 : ]
# Y_test = Y[len(Y) // 2 : ]
model = linear_model.LogisticRegression(C=1.0)
model.fit(X, Y)
```

```
The total number of males and females are 20095 and 308
```

Out[31]:
```
LogisticRegression()
```

In [32]:
```
train_predictions = model.predict(X)
test_predictions = model.predict(X)
# accuracy of logistic regression based model
sum(test_predictions == Y) / len(Y)
print(len(Y))
```

```
20403
```

In the above example, we saw that out of 20403 samples where there's a gender based review available, only 308 of them are from female reviewers. The remaining 20095 are from male reviewers. There's an imbalance within the dataset itself. Because of this imbalance, we are seeing that just predicting the gender to be Male results in a highly accurate predictor. But, the balanced error rate is 0.5, which tells us that we haven't built a classifier that would filter out false negatives.

Question 7 - Training a vanilla logistic regression based classifier

In [33]:
```
Y_false = np.array([0 for b in Y]) # What is the accuracy if we simply predict "Male" everytime?
sum(Y_false == Y) / len(Y)
```

Out[33]:
```
0.9849041807577317
```

In [34]:
```
def compute_confusion_matrix(Y_actual, Y_predict):
    TP = 0
    TN = 0
    FP = 0
    FN = 0
    for i in range(len(Y_predict)):
        if(Y_actual[i] == 1 and Y_predict[i] == 1):
            TP += 1
        elif(Y_actual[i] == 0 and Y_predict[i] == 0):
```

```
            TN += 1
        elif(Y_actual[i] == 0 and Y_predict[i] == 1):
            FP += 1
        elif(Y_actual[i] == 1 and Y_predict[i] == 0):
            FN += 1

    return (TP, FP, TN, FN)
```

In [35]:

```
TP, FP, TN, FN = compute_confusion_matrix(Y, test_predictions)
print("True Positive = {}".format(TP))
print("True Negative = {}".format(TN))
print("False Positive = {}".format(FP))
print("False Negative = {}".format(FN))
```

```
True Positive = 0
True Negative = 20095
False Positive = 0
False Negative = 308
```

$$TruePositiveRate = \frac{TP}{TP + FN}$$

$$TrueNegativeRate = \frac{TN}{TN + FP}$$

$$FalsePositiveRate = \frac{FP}{FP + TN} = 1 - TNR$$

$$FalseNegativeRate = \frac{FN}{FN + TP} = 1 - TPR$$

In [36]:

```
TPR = TP / (TP + FN)
TNR = TN / (TN + FP)
FPR = FP / (FP + TN)
FNR = FN / (FN + TP)
BER = 0.5 * (FPR + FNR)
print("The values of the TPR, TNR, FPR, FNR, BER are {}, {}, {}, {}, {}".format(TPR, TNR, FPR, FNR, BER))
```

```
The values of the TPR, TNR, FPR, FNR, BER are 0.0, 1.0, 0.0, 1.0, 0.5
```

Question 8 - Training a balanced classifier using LogisticRegression

In [37]:
```
# X_train = X[:len(X) // 2]
# Y_train = Y[:len(Y) // 2]

# X_test = X[len(X) // 2 : ]
# Y_test = Y[len(Y) // 2 : ]
model = linear_model.LogisticRegression(C=1.0,class_weight='balanced')
model.fit(X, Y)
```

Out[37]:
```
LogisticRegression(class_weight='balanced')
```

In [38]:
```
train_predictions = model.predict(X)
test_predictions = model.predict(X)
```

In [39]:
```
# accuracy of logistic regression based model
sum(test_predictions == Y) / len(Y)
```

Out[39]:
```
0.4225849139832378
```

In [40]:
```
Y_false = np.array([0 for b in Y]) # What is the accuracy if we simply predict "Male" everytime?
sum(Y_false == Y) / len(Y)
```

Out[40]:
```
0.9849041807577317
```

In [41]:
```
TP, FP, TN, FN = compute_confusion_matrix(Y, test_predictions)
print("True Positive = {}".format(TP))
print("True Negative = {}".format(TN))
print("False Positive = {}".format(FP))
print("False Negative = {}".format(FN))
TPR = TP / (TP + FN)
TNR = TN / (TN + FP)
FPR = FP / (FP + TN)
FNR = FN / (FN + TP)
BER = 0.5 * (FPR + FNR)
print("The values of the TPR, TNR, FPR, FNR, BER are {}, {}, {}, {}, {}".format(TPR, TNR, FPR, FNR, BER))
```

```
True Positive = 199
True Negative = 8423
False Positive = 11672
```

```
False Negative = 109
The values of the TPR, TNR, FPR, FNR, BER are 0.6461038961038961, 0.4191589947748196, 0.5808410052251803, 0.3538961
038961039, 0.4673685545606421
```

Question 9 - Training a better classifier. The feature of the overall review rating has been added to improve the quality of results for this classifier. There are three more features that have been added namely overall review, taste, and ABV of the beer. This has reduced the BER by almost 4.2%

In [42]:

```
# Aim - Generate a classifier with lesser BER than the Logistic regression classifier
data = list(parseDataFromFile("C:\\Users\\ramasarma\\Documents\\UCSD\\Fall 2020\\CSE 258\\Homework1\\beer_50000.json'
def feature(datum):
    feat = [[1,len(d['review/text']), d['review/overall'], d['review/taste'], d['beer/ABV']] for d in datum if 'user/
    return feat
```

In [43]:

```
X = feature(data)
Y = [d['user/gender'] for d in data if 'user/gender' in d]
## Encoding the dataset properly (0 - Male and 1 - Female)
for i in range(len(Y)):
    if Y[i] == 'Male':
        Y[i] = 0
    else:
        Y[i] = 1

# We split the training and the test set
# X_train = X[:len(X) // 2]
# Y_train = Y[:len(Y) // 2]

# X_test = X[len(X) // 2 : ]
# Y_test = Y[len(Y) // 2 : ]
model = linear_model.LogisticRegression(C=1.0,class_weight='balanced')
model.fit(X, Y)
train_predictions = model.predict(X)
test_predictions = model.predict(X)
sum(test_predictions == Y) / len(Y)
TP, FP, TN, FN = compute_confusion_matrix(Y, test_predictions)
print("True Positive = {}".format(TP))
print("True Negative = {}".format(TN))
print("False Positive = {}".format(FP))
print("False Negative = {}".format(FN))
TPR = TP / (TP + FN)
TNR = TN / (TN + FP)
```

```python
FPR = FP / (FP + TN)
FNR = FN / (FN + TP)
BER = 0.5 * (FPR + FNR)
print("The values of the TPR, TNR, FPR, FNR, BER are {}, {}, {}, {}, {}".format(TPR, TNR, FPR, FNR, BER))
```

```
True Positive = 189
True Negative = 9876
False Positive = 10219
False Negative = 119
The values of the TPR, TNR, FPR, FNR, BER are 0.6136363636363636, 0.49146553869121673, 0.5085344613087833, 0.38636363
636363635, 0.447449048836209847
```

In [ ]: