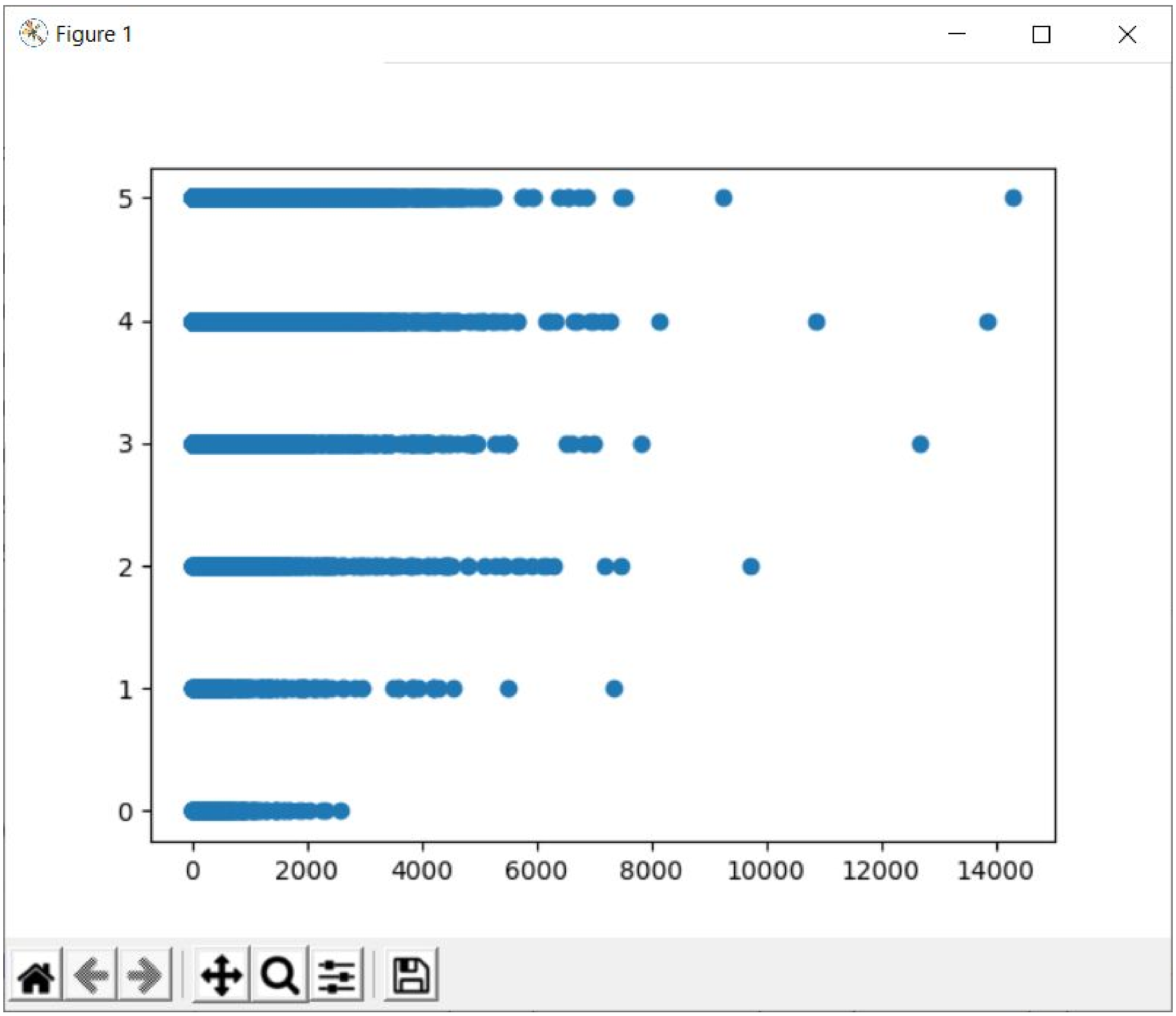
Regression

1)

```
# Number of 0, 1, ..., 5 star ratings
zero_star_ratings = len([rating for rating in y if rating == 0])  # 326
one_star_ratings = len([rating for rating in y if rating == 1])  # 286
two_star_ratings = len([rating for rating in y if rating == 2])  # 778
three_star_ratings = len([rating for rating in y if rating == 3])  # 2113
four_star_ratings = len([rating for rating in y if rating == 4])  # 3265
five_star_ratings = len([rating for rating in y if rating == 5])  # 3232
```



2)

```
def _text_length_feature(datum):
    """Return a vector of just one feature,
    The length of the review text in characters
    """
    feat = [1, len(datum['review_text'])]
    return feat


# Train a predictor to estimate rating from review length
x = [_text_length_feature(d) for d in data]  # Vector of 1 and review
length
y = [d['rating'] for d in data]  # Ratings
theta, residuals, rank, s = np.linalg.lstsq(x, y)  # Solve matrix equation

theta  # [3.68568136e+00  6.87371675e-05] Positive, longer reviews lead to
higher ratings?


def _new_regression_eq(theta, review_length):
    """Represents the equation rating = θ_0 + θ_1 * (length of review)
    """
    return theta[0] + theta[1] * review_length


# Calculate MSE
sum = 0
for i in range(len(data)):
    sum += (_new_regression_eq
            (theta,
             len(data[i]['review_text'])) - y[i]) ** 2
mse = sum / len(data)  # 1.5522086622355356
```

Resulting MSE: **1.5522086622355356**

3)

```
def _text_length_and_comments_feature(datum):
    """Return a vector of
    the length of the review text in characters
    and the number of comments
    """
    feat = [1, len(datum['review_text']), datum['n_comments']]
    return feat


# Train a predictor to estimate rating from review length
x = [_text_length_and_comments_feature(d)
     for d in data]  # Vector of 1, length, and number of comments
theta, residuals, rank, s = np.linalg.lstsq(x, y)  # Solve matrix equation


theta  # [3.68916737e+00  7.58407490e-05  -3.27928935e-02], coefficient
θ_1 is different because there is a third parameter when performing matrix
multiplication with its own value


def _new_regression_eq(theta, review_length, num_comments):  # Re-define
    """Represents the equation rating = θ_0 + θ_1 * (length of review) +
θ_2 * (number of comments)
    """
    return theta[0] + theta[1] * review_length + theta[2] * num_comments


# Calculate MSE
sum = 0
for i in range(len(data)):
    sum += (_new_regression_eq
            (theta,
             len(data[i]['review_text']), data[i]['n_comments']) - y[i])
** 2
mse = sum / len(data)  # 1.549835169277462, slightly better
```

Resulting MSE: **1.549835169277462**

4)

```
# Find maximum review length
reviews = [len(d['review_text']) for d in data]
max_length = np.max(reviews)   # amax


def _text_length_poly_feature(datum, degree):
    """Return a vector of just one feature,
    The length of the review text in characters, with polynomials up to 5
    """
    length = len(datum['review_text']) / max_length
    feat = [length ** (i + 1) for i in range(degree)]
    feat.insert(0, 1)
    return feat


# Train a predictor to estimate rating from review length
x = [_text_length_poly_feature(d, 5)
     for d in data]  # Vector of 1, review length with degree 5
y = [d['rating'] for d in data]  # Ratings
theta, residuals, rank, s = np.linalg.lstsq(x, y)  # Solve matrix equation

theta
# Degree 3: [3.63659658  2.8884065  -8.48042966  6.12504475]
# 4: [3.64736873  2.20419719  -1.80763945  -11.6451833  12.21844408]
# 5: [  3.6441158  2.47396326  -5.65441081  5.55309592  -15.94637484
14.68100179]


def _new_regression_eq(theta, review_length):
    """Represents the equation rating = θ_0 + (θ_1 * (length of review)^1)
+ ... + (θ_n * (length of review)^n)
    """
    review_length /= max_length
    val = theta[0]
    for i in range(len(theta) - 1):
        val += theta[i + 1] * review_length ** (i + 1)
    return val
```

```
# Calculate MSE
sum = 0
for i in range(len(data)):
    sum += (_new_regression_eq
            (theta,
             len(data[i]['review_text'])) - y[i]) ** 2
mse = sum / len(data)
# Degree 3: 1.5497985323805497
# 4: 1.549629132452472
# 5: 1.5496142023298662
```

Resulting MSE (in order by polynomial degree):

3: **1.5497985323805497**
4: **1.549629132452472**
5: **1.5496142023298662**

5)

```
# Split the data
np.random.shuffle(data)  # Randomize order of data
training_data = data[:len(data) // 2]  # First half
testing_data = data[len(data) // 2:]  # Second half

# Train a predictor to estimate rating from review length
x = [_text_length_poly_feature(d, 5)
     for d in training_data]  # Vector of 1, review length with degree 5
y = [d['rating'] for d in training_data]  # Ratings
theta, residuals, rank, s = np.linalg.lstsq(x, y)  # Solve matrix equation

# Degree 3: [3.65398022  2.85740482  -8.6349972   7.03106877]
# 4: [3.65538388  2.70313039  -4.77004286  -8.0607224  11.03884403]
# 5: [3.66726222  2.44067015  -12.01072447  41.08045065  -67.57729633
35.98020285]

# Calculate MSE
sum_training = 0
sum_testing = 0
y_testing = [d['rating'] for d in testing_data]
for i in range(len(training_data)):
    sum_training += (_new_regression_eq
                     (theta,
                      len(training_data[i]['review_text'])) - y[i]) ** 2
    sum_testing += (_new_regression_eq
                    (theta,
                     len(testing_data[i]['review_text'])) - y[i]) ** 2
mse_training = sum_training / len(training_data)
mse_testing = sum_testing / len(testing_data)
# Training and testing error (varies due to random datasets):
# Note that for these results, the dataset was re-randomized
# Degree 3: 1.5516115542413098 (training), 1.5659240213323633 (testing)
# 4: 1.5604723081811585 (training), 1.5792144817011526 (testing)
# 5: 1.5200201662189214 (training), 1.5308161745889985 (testing)
```

Resulting MSE (in order by polynomial degree, and training/testing data randomized for each):

3: **1.5516115542413098 (training), 1.5659240213323633 (testing)**
4: **1.5604723081811585 (training), 1.5792144817011526 (testing)**
5: **1.5200201662189214 (training), 1.5308161745889985 (testing)**

7)

```
def _text_length_feature(datum):
    """Return a vector of just one feature,
    The length of the review text in characters
    """
    feat = [1, len(datum['review/text'])]
    return feat


x = [_text_length_feature(d) for d in data]  # Vector of 1 and review
length
y = [d['user/gender'] for d in data]  # Genders
model = linear_model.LogisticRegression()
model.fit(x, y)
predictions = model.predict(x)


def evaluate_classifier(predictions, y):
    # Find True Positive, True Negative, False Positive, False Negative,
and Balanced Error Rates
    true_positive = 0  # Correctly guessed as female
    true_negative = 0  # Correctly guessed as male
    false_positive = 0  # Incorrectly guessed as female
    false_negative = 0  # Correctly guessed as male

    for i in range(len(y)):
        if (predictions[i] == y[i] == 'Female'):
            true_positive += 1
        elif (predictions[i] == y[i] == 'Male'):
            true_negative += 1
        elif (predictions[i] != y[i] == 'Female'):
            false_positive += 1
        elif (predictions[i] != y[i] == 'Male'):
            false_negative += 1

    true_positive_rate = true_positive / len(y)
    true_negative_rate = true_negative / len(y)
    false_positive_rate = false_positive / len(y)
    false_negative_rate = false_negative / len(y)
```

8)

```
    balanced_error_rate = (false_positive_rate + false_negative_rate) / 2

    return (true_positive_rate, true_negative_rate, false_positive_rate,
false_negative_rate, balanced_error_rate)


evaluate_classifier(predictions, y)
# In order of true positive rate, true negative rate, false positive rate,
false negative rate, balanced error rate
# (0.0, 0.9849041807577317, 0.015095819242268294, 0.0,
0.007547909621134147)
```

TP: **0.0**
TN: **0.9849041807577317**
FP: **0.015095819242268294**
FN: **0.0**
BER: **0.007547909621134147**

8)

```
# Retrain using balanced
model = linear_model.LogisticRegression(class_weight='balanced')
model.fit(x, y)
predictions = model.predict(x)

evaluate_classifier(predictions, y)
# Note: Our classifier got worse. Our BER is higher because we are
classifying a lot of men as women,
# Because most who took the survey were men
# In order of true positive rate, true negative rate, false positive rate,
false negative rate, balanced error rate
# (0.00975346762730971, 0.41283144635592806, 0.0053423516149585844,
0.5720727344018036, 0.2887075430083811)
```

TP: **0.00975346762730971**
TN: **0.41283144635592806**
FP: **0.0053423516149585844**
FN: **0.5720727344018036**
BER: **0.2887075430083811**

9)

```
def _more_features_feature(datum):
    """Return a vector of multiple features that may help
    classify a female better
    """
    feat = [1, len(datum['review/text']), float(datum['beer/beerId']),
            float(datum['beer/brewerId']), datum['review/overall'],
            datum['review/palate'], datum['review/taste'],
            datum['review/appearance'], datum['review/aroma']]
    return feat


x = [_more_features_feature(d) for d in data]
# Convert all to float
# Retrain using balanced and more features
model = linear_model.LogisticRegression(class_weight='balanced')
model.fit(x, y)
predictions = model.predict(x)

print(evaluate_classifier(predictions, y))
# Note: Performed worse for true positive
# In order of true positive rate, true negative rate, false positive rate,
false negative rate, balanced error rate
# (0.0032348184090574914, 0.7855707493995981, 0.011861000833210802,
0.1993334313581336, 0.10559721609567221)
```

TP: **0.0032348184090574914**
TN: **0.7855707493995981**
FP: **0.011861000833210802**
FN: **0.1993334313581336**
BER: **0.10559721609567221**