Nawen Gong

PID: A53275627

In [1]:
```
import numpy as np
import urllib
import scipy.optimize
from sklearn import svm
import random
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [2]:
```
def parseData(fname):
  for l in urllib.request.urlopen(fname):
    yield eval(l)
```

In [3]:
```
data = list(parseData("http://jmcauley.ucsd.edu/cse258/data/beer/beer_50000.json"))
```

In [4]:
```
y = [d['review/taste'] for d in data]
```

In [5]:
```
## What is the distribution of ratings in the dataset (for 'review/taste')? That is, how many 1-star, 2-star, 3-star (etc.) reviews are there?
```

In [6]:
```
unique_stars, counts_starts = np.unique(np.array(y), return_counts=True)
```

In [7]:
```
np.set_printoptions(suppress=True)
print(np.asarray((unique_stars, counts_starts)))
```

```
[[    1.      1.5     2.      2.5     3.      3.5     4.      4.5
5. ]
 [  211.    343.   1099.   1624.   4137.   8797.  16575.  12883.
4331. ]]
```

In [8]:
```
data2 = []

for d in data:
    if 'beer/style' in d and 'beer/ABV' in d and 'review/taste' in d:
        data2.append(d)
```

In [9]:
```
def feature(datum):
  feat = [1]
  if 'Hefeweizen' == datum['beer/style']:
    feat.append(1)
  else:
    feat.append(0)
  feat.append(datum['beer/ABV'])
  return feat
```

In [10]:
```
X1 = [feature(d) for d in data2]
```

In [11]:
```
y = [d['review/taste'] for d in data2]
```

In [12]:
```
theta,residuals,rank,s = np.linalg.lstsq(X1,y,-1)
```

In [13]:
```
print("theta: " + str(theta))
```

```
theta: [ 3.11795084 -0.05637406  0.10877902]
```

Theta0 is the first value in the list 'theta' and it represents the intercept of the linear function. Theta1 is the second value in the list 'theta' and it represents the coefficient between the first feature 'beer is a Hefeweizen' and the predicted value 'review/taste'. Theta2 is the third value in the list 'theta' and it represents the coefficient between the second feature 'beer/ABV' and the predicted value 'review/taste'.

In [14]:
```
print("residuals: " + str(residuals))
```

```
residuals: [22482.91275121]
```

In [15]:
```
print("rank: " + str(rank))
```

```
rank: 3
```

In [16]:
```
print("s: " + str(s))
```

```
s: [1747.0976296    66.41497289   24.54093593]
```

3.Split the data into two equal fractions – the first half for training, the second half for 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)?

In [17]:
```
n = int(0.5*len(data2))
```

In [18]:
```
d_train = data2[:n]
d_target = data2[n:]
```

In [19]:
```
X_train = [feature(d) for d in d_train]
y_train = [d['review/taste'] for d in d_train]
```

In [20]:
```
theta,residuals,rank,s = np.linalg.lstsq(X_train,y_train,-1)
```

In [21]:
```
print("theta: " + str(theta), "residuals: " + str(residuals), "rank
: " + str(rank), "s: " + str(s))
```

```
theta: [ 2.99691466 -0.03573098  0.11672256] residuals: [12099.201
40034] rank: 3 s: [1299.09816954   44.96451837   17.13837497]
```

In [22]:
```
MSE_train = np.mean((residuals/len(d_train)) ** 2)
print("MSE for training set is:" + str(MSE_train))
```

```
MSE for training set is:0.23422507924140004
```

In [23]:
```
X_target = [feature(d) for d in d_target]
y_target = [d['review/taste'] for d in d_target]
```

In [24]:
```
reg = LinearRegression().fit(X_train, y_train)
```

In [25]:
```
y_pred = reg.predict(X_target)
```

In [26]:
```
MSE_target = mean_squared_error(y_target, y_pred)
```

In [27]:
```
print("MSE for atrget set is:" + str(MSE_target))
```

```
MSE for atrget set is:0.4237065211985229
```

4.Using the first half for training and the second half for testing may lead to unexpected results (e.g. the training error could be higher than the test error). Repeat the above experiment by using a random 50% split of the data (i.e., half for training, half for testing, after first shuffling the data). Report the MSE on the train and test set, and suggest one possible reason why the result may be different from the previous experiment (1 mark)

In [28]:
```
data_r = random.sample(data2, int(len(data2)))
```

In [29]:
```
n_train_r = int(0.5*len(data_r))
n_target_r =  int(0.5*len(data_r))
```

In [30]:
```
d_train_r = data_r[:n_train_r]
d_target_r = data_r[n_target_r:]
```

In [31]:
```
X_train_r = [feature(d) for d in d_train_r]
y_train_r = [d['review/taste'] for d in d_train_r]
```

In [32]:
```
X_target_r = [feature(d) for d in d_target_r]
y_target_r = [d['review/taste'] for d in d_target_r]
```

In [33]:
```
theta,residuals,rank,s = np.linalg.lstsq(X_train_r,y_train_r,-1)
```

In [34]:
```
MSE_train_rand = np.mean((residuals/len(d_train_r)) ** 2)
```

In [35]:
```
print("MSE for training set is:" + str(MSE_train_rand))
```

```
MSE for training set is:0.20151436535640052
```

In [36]:
```
reg = LinearRegression().fit(X_train_r, y_train_r)
```

In [37]:
```
y_pred_r = reg.predict(X_target_r)
```

In [38]:
```
MSE_target_rand = mean_squared_error(y_target_r, y_pred_r)
```

In [39]:
```
print("MSE for target set is:" + str(MSE_target_rand))
```

```
MSE for target set is:0.45042666099361456
```

One possible factor that differentiates the MSE between shuffled and non-shuffled data is that, if we don't shuffle data to split it 50-50, the first half using as the training set from the original data might contain patterns, making the predictor biased and weakening its ability to predict appropriately. Shuffling the training data removes any ordering influence from how the data was gathered or prepared.

5.Modify your experiment from Question 4 to use the features review/taste ' θ0 + θ1 × [ABV if beer is a Hefeweizen] + θ2 × [ABV if beer is not a Hefeweizen] e.g. the first beer in the dataset would have feature [1, 5.0, 0] since the beer is a Hefeweizen. Report the training and testing MSE of this method (1 mark).

In [40]:
```
def feature2(datum):
  feat2 = [1]
  if 'Hefeweizen' == datum['beer/style']:
    feat2.append(datum['beer/ABV'])
    feat2.append(0)
  else:
    feat2.append(0)
    feat2.append(datum['beer/ABV'])

  return feat2
```

In [41]:
```
data_r = random.sample(data2, int(len(data2)))
```

In [42]:
```
n_train_r = int(0.5*len(data_r))
n_target_r =  int(0.5*len(data_r))

d_train_r = data_r[:n_train_r]
d_target_r = data_r[n_target_r:]
```

In [43]:
```
X_train_r = [feature2(d) for d in d_train_r]
y_train_r = [d['review/taste'] for d in d_train_r]

X_target_r = [feature2(d) for d in d_target_r]
y_target_r = [d['review/taste'] for d in d_target_r]
```

In [44]:
```
theta,residuals,rank,s = np.linalg.lstsq(X_train_r,y_train_r,-1)
MSE_train_rand = np.mean((residuals/len(d_train_r)) ** 2)

print("MSE for training set is:" + str(MSE_train_rand))
```

```
MSE for training set is:0.20366353344910615
```

In [45]:
```
reg = LinearRegression().fit(X_train_r, y_train_r)
y_pred_r = reg.predict(X_target_r)
MSE_target_rand = mean_squared_error(y_target_r, y_pred_r)
```

In [46]:
```
print("MSE for target set is:" + str(MSE_target_rand))
```

```
MSE for target set is:0.44804653288701185
```

6.The model from Question 5 uses the same two features as the model from Questions 2-4 and has the same dimensionality. Comment on why the two models might perform differently (1 mark).

The predictor used in question 2-4 makes predictions by learning the correlations of 'review/taste' and 'a beer is Hefeweizen' and the correlations of 'review/taste' and each beer's ABV.

The predictor used in question 5 makes predictions by studying the patterns between 'review/taste' and Hefeweizen's ABV and patterns between 'review/taste' and non-Hefeweizen's ABV.

The predictors predicts by learning different patterns that's why the two model performs differently.

1. First, let's train a predictor that estimates whether a beer is a 'Hefeweizen' using five features describing its rating: ['review/taste', 'review/appearance', 'review/aroma', 'review/palate', 'review/overall']. Train your predictor using an SVM classifier (see the code provided in class). Use a random split of the data as we did in Question 4. Use a regularization constant of C = 1000 as in the code stub. What is the accuracy (percentage of correct classifications) of the predictor on the train and test data? (1 mark)

In [47]:
```
data3 = []
for d in data:
    if 'review/taste' in d and 'review/appearance' in d and 'review
/aroma' in d and 'review/palate' in d and 'review/overall' in d:
        data3.append(d)
```

In [48]:
```
data4 = random.sample(data3, int(len(data3)))
```

In [49]:
```
X = [[b['review/taste'], b['review/appearance'], b['review/aroma'],
b['review/palate'], b['review/overall']] for b in data4]
y = ['Hefeweizen' in b['beer/style'] for b in data4]
```

In [50]:
```
int(len(data4) * 0.5)
```

Out[50]:
```
25000
```

In [51]:
```
X_train = X[:25000]
y_train = y[:25000]

X_test = X[25000:]
y_test = y[25000:]
```

In [52]:
```
clf = svm.SVC(C=1000, kernel='linear')
clf.fit(X_train, y_train)
```

Out[52]:
```
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='l
inear',
  max_iter=-1, probability=False, random_state=None, shrinking=Tru
e,
  tol=0.001, verbose=False)
```

In [53]:
```
train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)
```

In [54]:
```
pct_train = np.mean(train_predictions == y_train)
print("the accuracy of the train data is:" + str(pct_train))
```

```
the accuracy of the train data is:0.98672
```

In [55]:
```
pct_test = np.mean(test_predictions == y_test)
print("the accuracy of the test data is:" + str(pct_test))
```

```
the accuracy of the test data is:0.98856
```

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

I will add beer/name as a feature.

In [56]:
```
feature = ['review/taste', 'review/appearance', 'review/aroma', 're
view/palate', 'review/overall', 'beer/name']
```

In [57]:
```
data5 = []

for d in data:
    if 'review/taste' in d and 'review/appearance' in d and 'review
/aroma' in d and 'review/palate' in d and 'review/overall' in d and
'beer/name' in d:
        data5.append(d)
data5 = random.sample(data5, int(len(data5)))
```

In [58]:
```
pattern = ['HEFE', 'HEIFER', 'WEIZEN', 'HEFEWEIZEN', 'WHEAT']
```

In [59]:
```
def feature(datum):
    if any(p in datum['beer/name'].upper() for p in pattern):
        return 1
    else:
        return 0
```

In [60]:
```
X = [[b['review/taste'], b['review/appearance'], b['review/aroma'],
b['review/palate'], b['review/overall'], feature(b)] for b in data5
]
y = ['Hefeweizen' in b['beer/style'] for b in data5]
```

In [61]:
```
X_train = X[:25000]
y_train = y[:25000]

X_test = X[25000:]
y_test = y[25000:]
```

In [62]:
```
clf = svm.SVC(C=1000, kernel='linear')
clf.fit(X_train, y_train)
```

Out[62]:
```
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='l
inear',
  max_iter=-1, probability=False, random_state=None, shrinking=Tru
e,
  tol=0.001, verbose=False)
```

In [63]:
```
train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)
```

In [64]:
```
pct_train = np.mean(train_predictions == y_train)
print("the accuracy of the train data is:" + str(pct_train))
```

```
the accuracy of the train data is:0.99096
```

In [65]:
```
pct_test = np.mean(test_predictions == y_test)
print("the accuracy of the test data is:" + str(pct_test))
```

```
the accuracy of the test data is:0.99036
```