In [1]:

```
import numpy
import urllib
import scipy.optimize
import random

def parseData(fname):
  for l in urllib.urlopen(fname):
    yield eval(l)

print "Reading data..."
data = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.json
print "done"
```

```
Reading data...
done
```

In [6]:

```
# 1. 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? You may
# write out the values or include a simple plot (1 mark).
def dis_rate(datam):
    res = {1.0:0, 1.5:0, 2.0:0, 2.5:0, 3.0:0, 3.5:0, 4.0:0, 4.5:0, 5.0:0}
    for d in data:
      if d.has_key('review/taste'):
        res[d['review/taste']] = res[d['review/taste']] + 1
    return res
res = dis_rate(data)
res
```

Out[6]:

```
{1.0: 211,
 1.5: 343,
 2.0: 1099,
 2.5: 1624,
 3.0: 4137,
 3.5: 8797,
 4.0: 16575,
 4.5: 12883,
 5.0: 4331}
```

In [7]:

```
# 2. Train a simple predictor to predict a beer's 'taste' score using two
# features: review/taste ≃ θ0 + θ1 × [beer is a Hefeweizen] + θ2 × beer/ABV
# Report the values of θ0, θ1, and θ2. Briefly describe your interpretation
# of these values, i.e., what do θ0, θ1, and θ2 represent (1 mark)?
def feature(datam):
    feat = [1]
    if datam['beer/style'] == "Hefeweizen":
        feat.append(1)
    else:
        feat.append(0)
    feat.append(datam['beer/ABV'])
    return feat
X = [feature(d) for d in data]
Y = [d['review/taste'] for d in data]
theta,residuals,rank,s = numpy.linalg.lstsq(X, Y, rcond = -1)
theta
# The euqation can be represented in :
# review/taste = [θ0, θ1, θ2].[1, isHefeweizen, beer/ABV]
# if beer is Hefeweizen, θ1 = 1 else θ1 = 0
```

Out[7]:

```
array([ 3.11795084, -0.05637406,  0.10877902])
```

In [8]:

```
# 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)?
length = len(data)
train = data[:length/2]
test = data[length/2:length]
X = [feature(d) for d in train]
Y = [d['review/taste'] for d in train]
theta,residuals,rank,s = numpy.linalg.lstsq(X, Y, rcond = -1)
print theta
# MSE for train
def MSE(data, theta):
    res = 0
    for d in data:
        f = feature(d)
        res = res + numpy.square(d['review/taste'] -
                                 (theta[0] + theta[1]*f[1] + theta[2]*f[2]))
    res = res / len(data)
    return res
print MSE(train, theta)
print MSE(test, theta)
```

```
[ 2.99691466 -0.03573098  0.11672256]
0.48396805601335435
0.4237065211985192
```

In [24]:

```
# 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).
rand_data = numpy.copy(data)
mse_train = 0
mse_test = 0
for i in range(100):
    numpy.random.shuffle(rand_data)
    train = rand_data[:length/2]
    test = rand_data[length/2:length]
    X = [feature(d) for d in train]
    Y = [d['review/taste'] for d in train]
    theta,residuals,rank,s = numpy.linalg.lstsq(X, Y, rcond = -1)
    mse_train = mse_train + MSE(train, theta)
    mse_test = mse_test + MSE(test, theta)
print mse_train / 100
print mse_test / 100
# the mse for test set is closer to mse for train set than previous, this happe
# because the shuffled data are more irregular and less predictable, so we got
# mse in test set
```

```
0.4497756590511239
0.4496039833467828
```

In [25]:

```
# 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).
rand_data = numpy.copy(data)
mse_train = 0
mse_test = 0
length = len(rand_data)
def feature(datam):
    feat = [1]
    if datam['beer/style'] == "Hefeweizen":
        feat.append(datam['beer/ABV'])
        feat.append(0)
    else:
        feat.append(0)
        feat.append(datam['beer/ABV'])
    return feat
for i in range(100):
    numpy.random.shuffle(rand_data)
    train = rand_data[:length/2]
    test = rand_data[length/2:length]
    X = [feature(d) for d in train]
    Y = [d['review/taste'] for d in train]
    theta,residuals,rank,s = numpy.linalg.lstsq(X, Y, rcond = -1)
    mse_train = mse_train + MSE(train, theta)
    mse_test = mse_test + MSE(test, theta)
print mse_train / 100
print mse_test / 100
```

```
0.4489805618904388
0.4504005822900977
```

In [ ]:

```
# 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).

# Answer: Although they both have ABV and style, they are just in
# different forms(eg: when beer/style = Hefeweizen, feature in Q5 is
# [1,ABV,0], while feature in Q4 is [1,1,ABV]).
```

In [12]:

```
# 7. 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 tes

from sklearn import svm
from sklearn.metrics import accuracy_score

rand_data = numpy.copy(data)
length = len(rand_data)
X = [[d['review/taste'],d['review/appearance'],d['review/aroma'],
      d['review/palate'],d['review/overall'],] for d in rand_data]
Y = ["Hefeweizen" in d['beer/style'] for d in rand_data]
X_train = X[:length/2]
Y_train = Y[:length/2]
X_test = X[length/2:]
Y_test = Y[length/2:]
# svm modle
clf = svm.SVC(C=1000, kernel='linear')
clf.fit(X_train, Y_train)
train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)

# accuracy
accuracy_train = accuracy_score(train_predictions, Y_train)
accuracy_test = accuracy_score(test_predictions, Y_test)
print accuracy_train
print accuracy_test
```

```
0.98792
0.98736
```

In [15]:

```
# 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).
import numpy
import urllib
import scipy.optimize
import random
from sklearn import svm
from sklearn.metrics import accuracy_score

# I use review/taste, beer/ABV and  "Hefeweizen" in'review/text' to
# form the feature
rand_data = numpy.copy(data)
length = len(rand_data)
X = [[d['review/taste'],d['beer/ABV'], "Hefeweizen" in d['review/text']]
     for d in rand_data]
Y = ["Hefeweizen" in d['beer/style'] for d in rand_data]
X_train = X[:length/2]
Y_train = Y[:length/2]
X_test = X[length/2:]
Y_test = Y[length/2:]
# svm modle
clf = svm.SVC(C=1000, kernel='linear')
clf.fit(X_train, Y_train)
train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)

# accuracy
accuracy_train = accuracy_score(train_predictions, Y_train)
accuracy_test = accuracy_score(test_predictions, Y_test)
print accuracy_train
print accuracy_test
```

```
0.98936
0.98768
```

In [ ]:

```

```