In [67]:

```
import numpy as np
import pandas as pd
import urllib.request
import scipy.optimize
from sklearn.utils import shuffle
from collections import Counter
import matplotlib.pyplot as plt
```

Week 1

problem 1

In [68]:

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

In [69]:

```
print ("Reading data...")
data = list(parseData("http://jmcauley.ucsd.edu/cse258/data/beer/beer_50000.json"))
print ("done")
```

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

In [70]:

```
def feature(datum):
    feat = [1]
    return feat
```

In [71]:

```
X = [feature(d) for d in data]
y = [d['review/taste'] for d in data]
```

In [72]:

```
starCounter = Counter(y)
print(Counter(y))
```

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

problem 2

In [73]:

```
data2 = [d for d in data if 'beer/style' in d]
```

In [74]:

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

In [75]:

```
X1 = [feature(d) for d in data2]
y = [d['review/taste'] for d in data2]
```

In [76]:

```
theta,residuals,rank,s = np.linalg.lstsq(X1, y, rcond = None)
```

In [77]:

```
print(theta)
```

```
[ 3.11795084 -0.05637406  0.10877902]
```

#

theta 0: the base of taste review without the influence of the beer style and ABV

# theta 1: the contribute level of Hefeweizen to the taste review

# theta 2: the contribute level of ABV to the taste review

problem 3

In [78]:

```
X_train = X1[: len(X1)//2]
y_train = y[: len(y)//2]
```

In [79]:

```
theta1,residuals1,rank,s = np.linalg.lstsq(X_train, y_train, rcond = None)
```

In [80]:

```
print("MSE = ", residuals1 /len(y)*2)
```

```
MSE =  [0.48396806]
```

In [81]:

```
X_test = X1[len(X1)//2:]
y_test = y[len(y)//2:]
```

In [82]:

```
theta = np.matrix(theta1).T
X = np.matrix(X_test)
y = np.matrix(y_test).T
diff = X * theta - y
```

In [83]:

```
print("Test MSE = ", diff.T * diff / len(y) *2)
```

```
Test MSE =  [[0.84741304]]
```

problem 4

In [109]:

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

In [110]:

```
X_shuffle, y_shuffle = shuffle(X1, y)
```

In [111]:

```
X_train = X_shuffle[: 25000]
y_train = y_shuffle[: 25000]
```

In [112]:

```
theta1,residuals1,rank,s = np.linalg.lstsq(X_train, y_train, rcond = None)
```

In [113]:

```
print("Train MES = ", residuals1 /25000)
```

```
Train MES =  [0.44623072]
```

In [89]:

```
print("theta =", theta1)
```

```
theta = [ 3.1410697  -0.04954687  0.10545044]
```

In [90]:

```
X_test = X1[len(X1)//2:]
y_test = y[len(y)//2:]
```

In [91]:

```
theta = numpy.matrix(theta1).T
X = numpy.matrix(X_test)
y = numpy.matrix(y_test).T
diff = X * theta - y
```

In [92]:

```
print("Test MSE = ", diff.T * diff / 25000)
```

```
Test MSE =  [[0.41167149]]
```

Possible reason

The difference in the first half of the data and the second half is large, but the differences between each element in the halves are some.

By shuffling, we are less likely to converge to a solution lying in the global minimum for the whole training set (higher bias), but more likely to find a solution that generalizes better (lower variance)

problem 5

In [46]:

```
data2 = [d for d in data if 'beer/style' in d]
```

In [47]:

```
def feature(datum):
    feat = [1]
    if datum['beer/style'] == "Hefeweizen":
        feat.append(datum['beer/ABV'])
        feat.append(0)
    elif datum['beer/style'] != "Hefeweizen":
        feat.append(0)
        feat.append(datum['beer/ABV'])
    return feat
```

In [72]:

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

In [73]:

```
y = [d['review/taste'] for d in data2]
X_shuffle, y_shuffle = shuffle(X1, y)
X_train = X_shuffle[: 25000]
y_train = y_shuffle[: 25000]
```

In [74]:

```
theta1,residuals1,rank,s = np.linalg.lstsq(X_train, y_train, rcond = None)
```

In [75]:

```
print("Train MSE = ", residuals1 /25000)
```

```
Train MSE =  [0.44759196]
```

In [76]:

```
print("theta = ", theta1)
```

```
theta =  [3.1072901  0.0966647  0.11068058]
```

In [77]:

```
X_test = X1[len(X1)//2:]
y_test = y[len(y)//2:]
```

In [78]:

```
theta = np.matrix(theta1).T
X = np.matrix(X_test)
y = np.matrix(y_test).T
diff = X * theta - y
```

In [79]:

```
print("Test MSE = ", diff.T * diff / 25000)
```

```
Test MSE =  [[0.41127901]]
```

problem 6

The feature in question 4 and 5 are different. The first and second feature in question 4 are "if beer is a Hefeweizen" and ABV, respectively, which means X[2] is whether 0 or 1, and X[3] is beer/ABV value. However, in question 5, X[2] and X[3] are ABV when beer is a Hefeweizen, and ABV when beer is not a Hefeweizen. According to Naive Bayes rule, ABV and Hefeweizen are conditionally independent given the label. Hefeweizen just provides the condition for ABV.

Week 2

Problem 7

In [44]:

```
import numpy
import urllib.request
import scipy.optimize
from sklearn.utils import shuffle
from math import log, exp
from sklearn import svm # Support Vector Machines
```

In [45]:

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

In [46]:

```
print ("Reading data...")
data = list(parseData("http://jmcauley.ucsd.edu/cse258/data/beer/beer_50000.json"))
print ("done")
```

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

In [47]:

```
def feature(datum):
    feat = []
    feat.append(datum['review/taste'])
    feat.append(datum['review/appearance'])
    feat.append(datum['review/aroma'])
    feat.append(datum['review/palate'])
    feat.append(datum['review/overall'])
    return feat
```

In [48]:

```
X = [feature(d) for d in data]
y = ['Hefeweizen' in d['beer/style'] for d in data]
X, y = shuffle(X, y)
```

In [49]:

```
X_train = X[: len(X)//2]
y_train = y[: len(y)//2]

X_test = X[len(X)//2:]
y_test = y[len(y)//2:]
```

In [50]:

```
# Create a support vector classifier object, with regularization parameter C = 1000
clf = svm.SVC(C=1000, kernel='linear')
clf = svm.SVC(C = 1000, kernel = 'linear')
clf.fit(X_train, y_train)
```

Out[50]:

```
SVC(C=1000, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [51]:

```
train_predictions = clf.predict(X_train)
test_predictions = clf.predict(X_test)
```

In [52]:

```
correct_train = train_predictions == y_train
correct_test = test_predictions == y_test
```

In [53]:

```
trainAcc = sum(correct_train) / len(correct_train)
print("Training accuracy = ", trainAcc)
```

```
Training accuracy =  0.9884
```

In [54]:

```
testAcc = sum(correct_test) / len(correct_test)
print("Testing accuracy = ", testAcc)
```

```
Testing accuracy =  0.98688
```

In [55]:

```
sum(y_test)
```

Out[55]:

```
328
```

In [56]:

```
accFalse = (50000 - sum(y_test)) / 50000
accFalse
```

Out[56]:

```
0.99344
```

Problem 8

In [57]:

```
def inner(x,y):
    return sum([x[i]*y[i] for i in range(len(x))])

def sigmoid(x):
    return 1.0 / (1 + exp(-x))
```

In [58]:

```
def f(theta, X, y, lam):
    loglikelihood = 0
    for i in range(len(X)):
        logit = inner(X[i], theta)
        loglikelihood -= log(1 + exp(-logit))
        if not y[i]:
            loglikelihood -= logit
    for k in range(len(theta)):
        loglikelihood -= lam * theta[k]*theta[k]
    # for debugging
    # print "ll =", loglikelihood
    return -loglikelihood
```

In [59]:

```
def fprime(theta, X, y, lam):
    dl = [0]*len(theta)
    for i in range(len(X)):
        logit = inner(X[i], theta)
        for k in range(len(theta)):
            dl[k] += X[i][k] * (1 - sigmoid(logit))
            if not y[i]:
                dl[k] -= X[i][k]
    for k in range(len(theta)):
        dl[k] -= lam*2*theta[k]
    return np.array([-x for x in dl])
```

In [60]:

```
X_train = X[:int(len(X)/3)]
y_train = y[:int(len(y)/3)]
X_validate = X[int(len(X)/3):int(2*len(X)/3)]
y_validate = y[int(len(y)/3):int(2*len(y)/3)]
X_test = X[int(2*len(X)/3):]
y_test = y[int(2*len(X)/3):]
```

In [61]:

```
def train(lam):
    theta,_,_ = scipy.optimize.fmin_l_bfgs_b(f, [0]*len(X[0]), fprime, pgtol = 10, args = (X_train, 
y_train, lam))
    return theta
```

In [62]:

```
def performance(theta):
    scores_train = [inner(theta,x) for x in X_train]
    scores_validate = [inner(theta,x) for x in X_validate]
    scores_test = [inner(theta,x) for x in X_test]

    predictions_train = [s > 0 for s in scores_train]
    predictions_validate = [s > 0 for s in scores_validate]
    predictions_test = [s > 0 for s in scores_test]

    correct_train = [(a==b) for (a,b) in zip(predictions_train,y_train)]
    correct_validate = [(a==b) for (a,b) in zip(predictions_validate,y_validate)]
    correct_test = [(a==b) for (a,b) in zip(predictions_test,y_test)]

    acc_train = sum(correct_train) * 1.0 / len(correct_train)
    acc_validate = sum(correct_validate) * 1.0 / len(correct_validate)
    acc_test = sum(correct_test) * 1.0 / len(correct_test)
    return acc_train, acc_validate, acc_test
```

In [66]:

```
for lam in [0, 0.01, 1.0, 100.0]:
    theta = train(lam)
    acc_train, acc_validate, acc_test = performance(theta)
    print("lambda = " + str(lam) + ";\ttrain=" + str(acc_train) + "; validate=" + str(acc_validate)
+ "; test=" + str(acc_test))
```

```
lambda = 0; train=0.9887795511820473; validate=0.9875202495950081; test=0.9866202675946482
lambda = 0.01; train=0.9887795511820473; validate=0.9875202495950081; test=0.9866202675946482
lambda = 1.0; train=0.9887795511820473; validate=0.9875202495950081; test=0.9866202675946482
lambda = 100.0; train=0.9887795511820473; validate=0.9875202495950081; test=0.9866202675946482
lambda = 100.0; train=0.9887795511820473; validate=0.9875202495950081; test=0.9866202675946482
```