CSE258 HW1

January 22, 2017

1 CSE258 HW1

In [3]:
```
import numpy as np
import urllib
import scipy.optimize
import random
import csv
from sklearn.metrics import mean_squared_error
from sklearn import svm
from math import exp
from math import log
```

In [4]:
```
def parseData(fname):
    for l in urllib.urlopen(fname):
        yield eval(l)

print "Reading data..."
data = list(parseData("http://jmcauley.ucsd.edu/cse255/data/beer/beer_50000
print "done"
```

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

1.0.1 Regression Q1

We first train a predictor with linear regression as below:
$'review/overall' = \theta_0 + \theta_1 *' year'$
The fitted values are $\theta_0 = -3.917e+01, \theta_1 = 2.1438e-02$

In [5]:
```
x1 = [[1,i['review/timeStruct']['year']] for i in data]

y = [i['review/overall'] for i in data]
```

In [6]:
```
result1 = np.linalg.lstsq(x1,y)
print result1[0]
MSE1 = result1[1]/50000
print MSE1
```

```
[ -3.91707489e+01   2.14379786e-02]
[ 0.49004382]
```

1.0.2 Regression Q2

Then we use a second-order polynomial predictor instead of linear one:

$'review/overall' = \theta_0 + \theta_1 * year + \theta_2 * year^2$

The MSE in Q1 is 0.49004382. While here we have MSE as 0.49003734, which is only little better. Actually, 'review/overall' can depends little on one-dimension feature, 'year', since we see that the beers in the same 'year' may varies a lot in 'review/overall'. So our improvement is little.

```
In [7]: x2= []
        for feat in x1:
            newFeat = feat[:]
            newFeat.append(feat[1]**2)
            x2.append(newFeat)
```

```
In [8]: result2 = np.linalg.lstsq(x2,y)
        print result2[0]
        MSE2 = result2[1]/50000
        print MSE2
```

```
[ -2.32112075e+02   2.13653648e-01  -4.78731119e-05]
[ 0.49004374]
```

1.0.3 Regression Q3

The fitted coefficients $\theta = [\theta_0, \theta_1, ...]$ is:

[2.56420279e+02 1.35421303e-01 -1.72994866e+00 1.02651152e-01 1.09038568e-01 -2.76775146e-01 6.34332169e-03 3.85023977e-05 -2.58652809e+02 1.19540566e+00 8.33006285e-01 9.79304353e-02]

And train data MSE is 0.6023075

The MSE on the test data is 0.56245713031281874

```
In [47]: with open('winequality-white.csv', 'rb') as f:
             reader = csv.reader(f, delimiter=';')
             wine = []
             for row in reader:
                 wine.append(row)
         for i in range(1,len(wine)):
             for j in range(0,len(wine[i])):
                 wine[i][j] = float(wine[i][j])
         feature = wine[0]
         wine = wine[1:]
         train = np.array(wine[0:len(wine)/2])
         test = np.array(wine[len(wine)/2:len(wine)])
```

```
In [48]: train_x = np.concatenate((np.ones((len(train),1)), train[:,0:-1]), axis=1)
         train_y = train[:,-1]
```

```
In [49]: result3 = np.linalg.lstsq(train_x,train_y)
         MSE3 = result3[1]/len(train)
         print result3[0]
         print MSE3
```

```
[  2.56420279e+02   1.35421303e-01  -1.72994866e+00   1.02651152e-01
   1.09038568e-01  -2.76775146e-01   6.34332169e-03   3.85023977e-05
  -2.58652809e+02   1.19540566e+00   8.33006285e-01   9.79304353e-02]
[ 0.6023075]
```

```
In [50]: test_x = np.concatenate((np.ones((len(test),1)), test[:,0:-1]), axis=1)
         test_y = test[:,-1]

In [51]: MSE_full = mean_squared_error(np.dot(test_x, result3[0]), test_y)
         print MSE_full
```

```
0.562457130313
```

1.0.4 Regression Q4

The MSEs of all 11 ablation experiments are calculated below:

The features with least and most information should be the ones have the smallest and largest increment based on MSE with full features(0.562457130313).

Based on the test MSEs, we see 'density' provides the least additional information with smallest increment of -0.0177305768467

While 'volatile acidity' provides the most information with largest increment of 0.0339277198487

```
In [52]: MSE_abl = []
         for i in range(1,12):
             # get ablation feature data
             train_abl = np.delete(train_x, i, axis=1)
             test_abl = np.delete(test_x, i, axis=1)
             # training
             fit = np.linalg.lstsq(train_abl,train_y)
             # mse
             MSE_abl.append(mean_squared_error(np.dot(test_abl, fit[0]), test_y))
         for i in range(0,len(MSE_abl)):
             print "The MSE without the feature", feature[i], "is", MSE_abl[i]
             print "The difference with original MSE is", MSE_abl[i] - MSE_full
             print
```

```
The MSE without the feature fixed acidity is 0.559113414376
The difference with original MSE is -0.00334371593669

The MSE without the feature volatile acidity is 0.596384850162
The difference with original MSE is 0.0339277198487
```

```
The MSE without the feature citric acid is 0.562221702812
The difference with original MSE is -0.000235427501259

The MSE without the feature residual sugar is 0.553625063967
The difference with original MSE is -0.00883206634537

The MSE without the feature chlorides is 0.562629266481
The difference with original MSE is 0.000172136168481

The MSE without the feature free sulfur dioxide is 0.55614081793
The difference with original MSE is -0.00631631238286

The MSE without the feature total sulfur dioxide is 0.562429005469
The difference with original MSE is -2.81248436144e-05

The MSE without the feature density is 0.544726553466
The difference with original MSE is -0.0177305768467

The MSE without the feature pH is 0.559566626382
The difference with original MSE is -0.00289050393082

The MSE without the feature sulphates is 0.557346349988
The difference with original MSE is -0.00511078032493

The MSE without the feature alcohol is 0.573214743558
The difference with original MSE is 0.0107576132454
```

1.0.5 Classification Q5

Under C=0.8, I have accuracy on train and test data as 89.91% and 69.86%.

```
In [21]: train_lab = map(lambda x : 1 if x>5 else 0, train_y)
         test_lab = map(lambda x : 1 if x>5 else 0, test_y)
         clf = svm.SVC(C=0.8)
         clf.fit(train_x, train_lab)

         # prediction with classifier
         train_pred = clf.predict(train_x)
         test_pred = clf.predict(test_x)

In [22]: train_pair = np.vstack((np.array(train_lab), train_pred))
         test_pair = np.vstack((np.array(test_lab), test_pred))
         train_correct = filter(lambda x : x[0] == x[1], train_pair.T)
         test_correct = filter(lambda x : x[0] == x[1], test_pair.T)
```

```
In [23]: train_accuracy = len(train_correct) * 1. /len(train_x)
         test_accuracy = len(test_correct) * 1. /len(test_x)
         print train_accuracy, test_accuracy
```

```
0.899142507146 0.698652511229
```

1.0.6 Classification Q6

The log-likelihood after convergence is -1383.18, and the accuract of the resulting model is 76.68%

```
In [28]: 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 [53]: # NEGATIVE Log-likelihood
         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]
             return -loglikelihood
```

```
In [54]: # NEGATIVE Derivative of log-likelihood
         def fprime(theta, X, y, lam):
             dl = [0.0]*len(theta)
             for i in range(len(X)):
                 xi = sigmoid(inner(X[i], theta))
             # Fill in code for the derivative
                 for j in range(len(dl)):
                     dl[j] += X[i][j] * (1.0 - xi)
                     if not y[i]:
                         dl[j] -= X[i][j]
             dl -= lam * 2.0 * theta
             # Negate the return value since we're doing gradient *ascent*
             return np.array([-x for x in dl])
```

```
In [55]: # If we wanted to split with a validation set:
         #X_valid = X[len(X)/2:3*len(X)/4]
         #X_test = X[3*len(X)/4:]

         # Use a library function to run gradient descent (or you can implement you
         theta,l,info = scipy.optimize.fmin_l_bfgs_b(f, [0]*len(train_x[0]), fprime
```

```
        print "Final log likelihood =", -l
        # predict the test data
        test_pred = map(lambda x: 0 if x<0.5 else 1, [sigmoid(inner(X, theta)) for
        test_pair = np.vstack((np.array(test_lab), test_pred))
        test_correct = filter(lambda x: x[0] == x[1], test_pair.T)
        print "Accuracy = ", len(test_correct) * 1.0 / len(test_pred)
```

```
Final log likelihood = -1383.18543063
Accuracy =  0.766843609637
```

In [ ]: