# Homework 1

January 23, 2017

```
In [2]: import urllib
        import scipy.optimize
        import random
        import numpy as np
        import csv
        from sklearn.metrics import mean_squared_error
        from sklearn import svm
        from math import exp
        from math import log
```

## 1 Problem 1

To solve this equation $review/overall \simeq \theta_0 + \theta_1 \times year$ , I use python library numpy.linalg.lstsq
The fitted values are: $\theta_0 = -3.91707489 * 10^{-1}, \theta_1 = 2.14379786 * 10^{-2}$

```
In [3]: def parseData(fname):
          for l in urllib.urlopen(fname):
            yield eval(l)

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

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

```
In [4]: year = np.array([review['review/timeStruct']['year'] for review in winedata])
        y = np.array([review['review/overall'] for review in winedata])
        X = np.vstack([np.ones(len(year)), year]).T
        theta, residuals, rank, s = np.linalg.lstsq(X, y)
        MSE = residuals/len(year)
        print theta, MSE
```

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

## 2 Problem 2

I add a feature which is the square of year since I am considering that the overall review may not be linear with year. $review/overall \simeq \theta_0 + \theta_1 \times year + \theta_2 \times year^2$

The Mean Square Error is 0.49004382 in problem 1, and now it improves to 0.49004374. The improvement is quite small. Actually I think year is not a very good feature to predict the overall review and the review is not very independent on the year.

```
In [12]: X_2 = np.vstack([np.ones(len(year)), year, year**2]).T
         theta_2,residuals_2, rank_2, s_2 = np.linalg.lstsq(X_2, y)
         MSE_2 = residuals_2/len(year)
         print theta_2, MSE_2
```

```
[ -2.32111616e+02   2.13653191e-01  -4.78729979e-05] [ 0.49004374]
```

# 3 Problem 3

The fitted coefficients on training are

[ 2.56420279e+02, 1.35421303e-01, -1.72994866e+00, 1.02651152e-01, 1.09038568e-01, -2.76775146e-01, 6.34332168e-03, 3.85023977e-05, -2.58652809e+02, 1.19540566e+00, 8.33006285e-01, 9.79304353e-02]

The MSE on the training set is 0.602307502903, the MSE on the test set is 0.562457130315

```
In [4]: with open('winequality-white.csv', 'rb') as csvfile:
            reader = csv.reader(csvfile, delimiter=';')
            content = []
            for row in reader:
                content.append(row)
        indexLabel = {}
        for index, label in enumerate(content[0]):
            indexLabel[index+1] = label
        data = map(lambda row: [1]+[float(x) for x in row], content[1:])
        trainingSet = np.array(data[:len(data)/2])
        testSet = np.array(data[len(data)/2:])
```

```
In [5]: theta_wine, residuals_wine, rank_wine, s_wine = np.linalg.lstsq(trainingSet[:,:-1], trainingSet
```

```
In [6]: theta_wine
```

```
Out[6]: array([  2.56420279e+02,   1.35421303e-01,  -1.72994866e+00,
                 1.02651152e-01,   1.09038568e-01,  -2.76775146e-01,
                 6.34332168e-03,   3.85023977e-05,  -2.58652809e+02,
                 1.19540566e+00,   8.33006285e-01,   9.79304353e-02])
```

```
In [7]: print mean_squared_error(np.dot(trainingSet[:,:-1],theta_wine), trainingSet[:,-1])
        print mean_squared_error(np.dot(testSet[:,:-1],theta_wine), testSet[:,-1])
        mse_all = mean_squared_error(np.dot(testSet[:,:-1],theta_wine), testSet[:,-1])
```

```
0.602307502903
0.562457130315
```

# 4 Problem 4

The MSEs on the test set of all 11 ablation experiments are shown under the code below.

Based on the test MSEs, volatile acidity provides the most additional information since (MSE[excluding volatile acidity] - MSE[all feature]) is the maximum one.

The feature “density” provides the least additional information beyond what is present in the 10 other features since (MSE[excluding density] - MSE[including density]) is minimized.

```
In [17]: result = []
         for i in range(1,12):
             train, test = np.delete(trainingSet, i, axis=1), np.delete(testSet, i, axis=1)
             theta_ablation, residuals_ablation, rank_ablation, s_ablation = \
             np.linalg.lstsq(train[:,:-1], train[:,-1])
```

```
            MSE = mean_squared_error(np.dot(test[:,:-1],theta_ablation), test[:,-1])
            print "Remove", indexLabel[i],", the MSE on test is", MSE
            result.append([indexLabel[i],MSE])
        # result.sort(key = lambda x: x[1])
        # for re in result:
        #     if re[1] > mse_all:
        #         print re[0]
```

```
Remove fixed acidity , the MSE on test is 0.559113414376
Remove volatile acidity , the MSE on test is 0.596384850161
Remove citric acid , the MSE on test is 0.562221702812
Remove residual sugar , the MSE on test is 0.553625063967
Remove chlorides , the MSE on test is 0.562629266481
Remove free sulfur dioxide , the MSE on test is 0.55614081793
Remove total sulfur dioxide , the MSE on test is 0.562429005469
Remove density , the MSE on test is 0.544726553466
Remove pH , the MSE on test is 0.559566626382
Remove sulphates , the MSE on test is 0.557346349988
Remove alcohol , the MSE on test is 0.573214743558
```

# 5 Problem 5

Here I use the library sklearn.svm.SVC to train the data. Based on experiments, I found C=0.5 works relatively well.

The accuracy (percentage of correct classifications) of the predictor is 84.8509% on the train data, and 69.90608% on the test data.

```
In [11]: trainingData, trainingLabel = trainingSet[:,:-1], trainingSet[:,-1]
         trainingLabel = map(lambda x: 1.0 if x>5 else 0.0, trainingLabel)
         testData, testLabel = testSet[:,:-1], testSet[:,-1]
         testLabel = map(lambda x: 1 if x>5 else 0, testLabel)
```

```
In [12]: clf = svm.SVC(C=0.5)
         clf.fit(trainingData, trainingLabel)
```

```
Out[12]: SVC(C=0.5, cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0,
           kernel='rbf', max_iter=-1, probability=False, random_state=None,
           shrinking=True, tol=0.001, verbose=False)
```

```
In [13]: correct = filter(lambda x: x[0] == x[1], zip(clf.predict(trainingData),trainingLabel))
         correctRate = len(correct)*1.0/len(trainingLabel)
         print "training:", correctRate
         correct_test = filter(lambda x: x[0] == x[1], zip(clf.predict(testData), testLabel))
         correctRate_test = len(correct_test)*1.0/len(testLabel)
         print "test", correctRate_test
```

```
training: 0.844426296448
test 0.69906084116
```

# 6 Problem 6

The final log-likelihood after convergence is -1383.18949552, and the accuracy of the resulting model on the test set is 0.767251939567

Noted that first, the bias is added in the training data. Second, the final negative log-likelihood given by scipy.optimize.fmin_l_bfgs_b includes the regularization since the function f which is passed into this library includes the regularization.

In [14]:
```
def inner(x,y):
    return sum([x[i]*y[i] for i in range(len(x))])

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

# 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

def fprime(theta, X, y, lam):
    dl = np.array([0.0]*len(theta))
    for i in range(len(X)):
        dl += (np.array(X[i])/(1+exp(np.dot(theta, np.array(X[i])))))
        if not y[i]:
            dl -= X[i]
    dl -= 2*lam*np.array(theta)
    # Negate the return value since we're doing grad ient *ascent*
    return np.array([-x for x in dl])

# Use a library function to run gradient descent (or you can implement yourself!)
result = scipy.optimize.fmin_l_bfgs_b(f, np.zeros(len(trainingData[0])), fprime, args = (train
theta = result[0]
print theta
print "Final log likelihood = ", -result[1]
```

```
[ -2.09893691e+00  -2.59661194e-01  -4.26334221e+00   2.22926668e-01
   1.47520329e-02  -5.41120346e-01   1.59255350e-02  -1.77788493e-03
  -2.09625819e+00  -4.45081694e-01   1.18692510e+00   8.05788118e-01]
Final log likelihood =  -1383.18949552
```

In [15]:
```
probability = sigmoid(np.dot(testData, theta))
predict = map(lambda prob: 1 if prob >= 0.5 else 0, probability)
print "Accuracy =",len(filter(lambda x: x[0] == x[1], zip(predict, testLabel)))*1.0/len(testLa
```

```
Accuracy = 0.767251939567
```

In [ ]: