Homework 1 **Xiaowen Mao**
CSE 258 PID: A53220159
Spring 2017 x9mao@eng.ucsd.edu

1. $review/overall \approx \theta_0 + \theta_1 \times year$

$\theta_0 = -39.170748935477874$.

$\theta_1 = 0.021437978563939183$.

$MSE = 0.49004381985815582$

The source code is as showed in Figure 1.

```
In [11]: def feature(datum):
           feat = [1]
           feat.append(datum['review/timeStruct']['year'])
           return feat

         X = [feature(d) for d in data]
         y = [d['review/overall'] for d in data]
         theta,residuals,rank,s = numpy.linalg.lstsq(X, y)

In [12]: theta

Out[12]: array([ -3.91707489e+01,   2.14379786e-02])

In [13]: def MSE(X, y, theta):
             theta = numpy.matrix(theta)
             X = numpy.matrix(X)
             y = numpy.matrix(y)
             e = y.T - X*theta.T
             mse = e.T * e
             mse = numpy.array(mse.flatten().tolist()[0])
             mse = mse / len(X)
             return mse[0]

In [14]: MSE(X,y,theta)

Out[14]: 0.49004381985815582
```

Figure 1: Code for Task 1

2. Better Representation For Year

Learning from data, the year is in the range of 1999 to 2012. We can represent it with a list with size of 14, shown as Figure 2.

```
year = {1999:[1,0,0,0,0,0,0,0,0,0,0,0,0,0],
        2000:[0,1,0,0,0,0,0,0,0,0,0,0,0,0],
        2001:[0,0,1,0,0,0,0,0,0,0,0,0,0,0],
        2002:[0,0,0,1,0,0,0,0,0,0,0,0,0,0],
        2003:[0,0,0,0,1,0,0,0,0,0,0,0,0,0],
        2004:[0,0,0,0,0,1,0,0,0,0,0,0,0,0],
        2005:[0,0,0,0,0,0,1,0,0,0,0,0,0,0],
        2006:[0,0,0,0,0,0,0,1,0,0,0,0,0,0],
        2007:[0,0,0,0,0,0,0,0,1,0,0,0,0,0],
        2008:[0,0,0,0,0,0,0,0,0,1,0,0,0,0],
        2009:[0,0,0,0,0,0,0,0,0,0,1,0,0,0],
        2010:[0,0,0,0,0,0,0,0,0,0,0,1,0,0],
        2011:[0,0,0,0,0,0,0,0,0,0,0,0,1,0],
        2012:[0,0,0,0,0,0,0,0,0,0,0,0,0,1]}
```

Figure 2: Year Representation

The equation for it in terms of θ is:

$$review/overall \approx \theta_0 + \theta_1 \times [if\ 1999] + \theta_2 \times [if\ 2000] + ... + \theta_{14} \times [if\ 2012]$$

Then we compute the MSE:

$MSE_1 = 0.49004381985815582$

$MSE_2 = 0.48915867194268153$

We can see MSE is decreasing with the second representation.

The source code is attached in Figure 3.

```
In [3]: def MSE(X, y, theta):
            theta = numpy.matrix(theta)
            X = numpy.matrix(X)
            y = numpy.matrix(y)
            e = y.T - X*theta.T
            mse = e.T * e
            mse = numpy.array(mse.flatten().tolist()[0])
            mse = mse / len(X)
            return mse[0]

In [4]: def feature(datum):
          feat = [1] + year.get(datum['review/timeStruct']['year'])
          return feat

        X = [feature(d) for d in data]
        y = [d['review/overall'] for d in data]
        theta,residuals,rank,s = numpy.linalg.lstsq(X, y)

In [5]: MSE(X,y,theta)

Out[5]: 0.48915867194268153
```

Figure 3: Code for Task 2

3. $quality = \theta_0 + \theta_1 \times fixed\ acidity + \theta_2 \times volatile\ acidity + ... + \theta_{11} \times alcohol$

The fitted coefficients on the training data is

$$\begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \\ \theta_5 \\ \theta_6 \\ \theta_7 \\ \theta_8 \\ \theta_9 \\ \theta_{10} \\ \theta_{11} \end{pmatrix} = \begin{pmatrix} 2.56420279e+02 \\ 1.35421303e-01 \\ -1.72994866e+00 \\ 1.02651152e-01 \\ 1.09038568e-01 \\ -2.76775146e-01 \\ 6.34332168e-03 \\ 3.85023977e-05 \\ -2.58652809e+02 \\ 1.19540566e+00 \\ 8.33006285e-01 \\ 9.79304353e-02 \end{pmatrix}$$

Then we compute the MSE on the train and test data:

$MSE_{train} = 0.602307502903$

$MSE_{test} = 0.562457130315$

In [10]:
```
def MSE(X, y, theta):
    theta = numpy.matrix(theta)
    X = numpy.matrix(X)
    y = numpy.matrix(y)
    e = y.T - X*theta.T
    mse = e.T * e
    mse = numpy.array(mse.flatten().tolist()[0])
    mse = mse / len(X)
    return mse[0]
```

In [11]:
```
print "MSE on training: ", MSE(X_train, y_train, theta)
```

```
MSE on training:  0.602307502903
```

In [12]:
```
X_test = [feature(d) for d in testData]
y_test = [float(d[11]) for d in testData]
print "MSE on testing: ", MSE(X_test, y_test, theta)
```

```
MSE on testing:  0.562457130315
```

Figure 4: MSE on train & test data

The code is attached in Figure 5.

```
In [1]: import numpy
        import urllib
        import scipy.optimize
        import random
        import csv

        print "Reading data..."
        with open('winequality-white.csv', 'rb') as csvfile:
            reader = csv.reader(csvfile, delimiter=';')
            data = []
            for row in reader:
                data.append(row)
        print "done"

        Reading data...
        done

In [2]: title = data[0]
        half = (len(data)-1)/2
        trainData = data[1:half+1]
        testData = data[half+1:]

        def feature(datum):
          feat = [1] + [float(datum[i]) for i in range(11)]
          return feat

In [3]: print "training"
        X_train = [feature(d) for d in trainData]
        y_train = [float(d[11]) for d in trainData]
        theta,residuals,rank,s = numpy.linalg.lstsq(X_train, y_train)
        print "done"

        training
        done

In [4]: theta

Out[4]: 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])
```

Figure 5: Code for Task 3

4. Ablation Experiment

a. MSEs (on the test set) of all 11

Remove	MSE
fixed acidity	0.559113414376
volatile acidity	0.596384850161
citric acid	0.562221702811
residual sugar	0.553625063967
chlorides	0.562629266481
free sulfur dioxide	0.55614081793
total sulfur dioxide	0.562429005469
density	0.544726553466
pH	0.559566626382
sulphates	0.557346349988
alcohol	0.573214743558

The code is attached in Figure 6.

```
In [64]: def ablation(datum, remove):
             feat = [1] + [float(datum[i]) for i in range(remove) + range(remove+1,11)]
             return feat
```

```
In [84]: def ablationMSE(i):
             X_train_a = [ablation(d,i) for d in trainData]
             y_train_a = [float(d[11]) for d in trainData]
             theta_a,residuals,rank,s = numpy.linalg.lstsq(X_train_a, y_train_a)

             X_test_a = [ablation(d,i) for d in testData]
             y_test_a = [float(d[11]) for d in testData]
             print "MSE: " + str(MSE(X_test_a,y_test_a,theta_a)) + " ablation on " + title[i]
             return MSE(X_test_a,y_test_a,theta_a)
```

```
In [85]: abl = []
         for i in range(11):
             abl.append(ablationMSE(i))

         MSE: 0.559113414376 ablation on fixed acidity
         MSE: 0.596384850161 ablation on volatile acidity
         MSE: 0.562221702811 ablation on citric acid
         MSE: 0.553625063967 ablation on residual sugar
         MSE: 0.562629266481 ablation on chlorides
         MSE: 0.55614081793 ablation on free sulfur dioxide
         MSE: 0.562429005469 ablation on total sulfur dioxide
         MSE: 0.544726553466 ablation on density
         MSE: 0.559566626382 ablation on pH
         MSE: 0.557346349988 ablation on sulphates
         MSE: 0.573214743558 ablation on alcohol
```

Figure 6: Code for Task 4

b. The Most and Least Additional Information

By removing "volatile acidity", we get the highest MSE. So, "volatile acidity" provides the most additional information.

By removing "density", we get the lowest MSE. So, "density" provides the least additional information.

5. SVM Classifier

The accuracy (percentage of correct classifications) of the predictor on the train and test data:

$$accuracy_{\ train} = 0.899142507146$$

$$accuracy_{\ test} = 0.698652511229$$

The code is attached in Figure 8.

```
In [240]: def featureX(datum):
              feat = [1] + [float(datum[i]) for i in range(11)]
              return feat

          def featureY(datum):
              if float(datum[11]) <= 5:
                  return 0
              else:
                  return 1

          X_train = [featureX(d) for d in trainData]
          y_train = [featureY(d) for d in trainData]

          X_test = [featureX(d) for d in testData]
          y_test = [featureY(d) for d in testData]

In [241]: def accuracy(predict, truth):
              truth = numpy.array(truth)
              e = sum(abs(predict-truth))
              return  1-float(e) / len(predict)

In [242]: clf = svm.SVC(C=0.8)
          clf.fit(X_train, y_train)

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

In [243]: print "on training: ", accuracy(train_predictions, y_train)

          on training:  0.899142507146

In [244]: print "on testing: ", accuracy(test_predictions, y_test)

          on testing:  0.698652511229
```

Figure 7: Code for Task 5

6. Logistic Regression

a. derivative ($fprime$)

```
# NEGATIVE Derivative of log-likelihood
def fprime(theta, X, y, lam):
    dl = [0.0]*len(theta)
    for i in range(len(X)):
        # Fill in code for the derivative
        logit = inner(X[i], theta)
        for j in range(len(theta)):
            dl[j] += (y[i] - sigmoid(logit)) * X[i][j]
    for j in range(len(theta)):
        dl[j] -=  2 * lam * theta[j]
    # Negate the return value since we're doing gradient *ascent*
    return numpy.array([-x for x in dl])
```

Figure 8: Code stub for $fprime$

b. after convergence After convergence, we have with $\lambda = 1.0$:

$$log - likelihood = -1383.18364755$$

$$accuracy_{\ test} = 0.766843609637$$

The code is attached in Figure 9.

```
In [73]: def accuracy(theta, X, y):
             correct = 0
             for i in range(len(y)):
                 if sigmoid(inner(X[i], theta)) > 0.5:
                     predict = 1
                 else:
                     predict = 0
                 if predict == y[i]:
                     correct += 1
             return  float(correct) / len(y)

In [77]: theta,l,info = scipy.optimize.fmin_l_bfgs_b(f, [0]*len(X_train[0]), fprime, args = (X_train, y_train, 1.0))
         print "Final log likelihood =", -l

         Final log likelihood = -1383.18364755

In [78]: ac = accuracy(theta, X_test, y_test)
         print "Accuracy = ", ac

         Accuracy =  0.766843609637
```

Figure 9: Code for task 6