Homework 1

Wenjun Zhang
A53218995

Tasks - Regression

1. First, train a predictor as below which uses the year ('review/timeStruct'/'year') to predict the overall rating.

$$\text{review / overall} \cong \theta_0 + \theta_1 \times \text{year}$$

The fitted values of θ_0 and θ_1 are -3.91707489e+01 and 2.14379786e-02.

2. A simple regressor above is not very realistic because of the monotonicity. Perhaps we can do better with a polynomial function as below:

$$\text{review / overall} \cong \theta_0 + \theta_1 \times \text{year} + \theta_1 \times \text{year} + \theta_2 \times \text{year}^2 + \theta_3 \times \text{year}^3 \ldots$$

We do with polynomials up to degree 5, which is:

$$\text{review / overall} \cong \theta_0 + \theta_1 \times \text{year} + \theta_1 \times \text{year} + \theta_2 \times \text{year}^2 + \theta_3 \times \text{year}^3 + \theta_4 \times \text{year}^4 + \theta_5 \times \text{year}^5$$

Then we compute the Mean Square Errors of this new representation and the representation from Question 1. The MSE of the representation from Question 1 is 0.49004382, and the MSE of the new representation is 0.49004372, which is slightly smaller than the previous MSE, so the new representation is a bit better compared to the representation from Question 1.

3. Train a regressor below that uses the first 11 feeatures of the csv file to predict the last feature ('quality').

$$\text{quality} \cong \theta_0 + \theta_1 \times \text{'fixed acidity'} + \theta_2 \times \text{'citric acid'} + \ldots + \theta_{11} \times \text{'alcohol'}$$

The fitted coefficients on the training data 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]

And the MSE of the training data and test data are 0.6023075 aand 0.56245713.

4. Implement ablation experiment which removes one feature in order to assess the amount of additional information that feature provides beyond the others.

(a) The MSEs of all 11 ablation experiments are as below:

removed feature	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
MSE	1.21013099	1.1781869	1.20306521	1.14246696	1.20264875	1.18653113	1.20251255	1.15344816	1.21727072	1.17295059	1.20516896

(b) The biggest and the smallest MSE are 1.21727072 and 1.14246696, which correspond to pH and residual sugar. So feature 'pH' provides the most additional information while feature 'residual sugar' provides the least.

5. Using the first 11 features and run an SVM classifier on the data, where the first half are used as traing data and the rest are used for test. When using different C, the accuracy are different. Choosing C=1000, The accuracy of the predictor on the train and test data are 1.0 and 0.668027766435.

6. Using logistic regression, the log-likelihood expression is:

(1)

$$l_\theta(y \mid X) = \sum_i -\log(1+e^{-X_i \bullet \theta}) + \sum_{y_i=0} -X_i \bullet \theta - \lambda \| \theta \|_2^2$$

Thus, the derivative is :

$$\frac{\partial l}{\partial \theta_k} = \sum_i X_{ik}(1-\sigma(X_i \bullet \theta)) + \sum_{y_i=0} -X_{ik} - 2\lambda\theta_k$$

where $\sigma(x) = \dfrac{1}{1+e^{-x}}$.

With the theorem we can complete the code stub for derivative (fprime):

```
## NEGATIVE Derivative of log-likelihooddef fprime(theta, X, y, lam):
  dl = [0.0]*len(theta)
  for k in range(len(theta)):
      sum = 0
      for i in range(len(X)):
        # Fill in code for the derivative
        logit = inner(X[i], theta)
        sum += X[i][k] * (1-sigmoid(logit))
        if not y[i]:
            sum -= X[i][k]
      dl_temp = sum - 2 * lam * theta[k]
      dl[k] = dl_temp
  # Negate the return value since we're doing gradient *ascent*
  return numpy.array([-x for x in dl])
```

(2) the log-likelihood of after convergence is -1399.69403841, and the accuracy (on the te st set) of the resulting model is 0.76929358922.

Code for Q1 & Q2

In [1]:

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

In [2]:

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

In [3]:

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

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

In [4]:

```
### predictor that uses the year to predict the overall rating

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

X1 = [feature(d) for d in data]
y = [d['review/overall'] for d in data]
thetaset1,residuals,rank,s = numpy.linalg.lstsq(X1, y)
print thetaset1
```

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

In [5]:

```
### Regressor with polynomial function using year

def feature(datum):
  feat=[1]
  feat.append(datum['review/timeStruct']['year'])
  feat.append(numpy.power(datum['review/timeStruct']['year'],2))
  feat.append(numpy.power(datum['review/timeStruct']['year'],3))
  feat.append(numpy.power(datum['review/timeStruct']['year'],4))
  feat.append(numpy.power(datum['review/timeStruct']['year'],5))
  return feat
X2 = [feature(d) for d in data]
y = [d['review/overall'] for d in data]
thetaset2,residuals,rank,s = numpy.linalg.lstsq(X2, y)
print thetaset2
```

```
[ -5.24334109e-17  -6.30565241e-14  -6.33694373e-11  -4.23969147e-08
   4.07781528e-11  -9.67413362e-15]
```

In [6]:

```
# Compute the MSE

def msecomp(theta, X, y):
  theta = numpy.matrix(theta).T
  X = numpy.matrix(X)
  y = numpy.matrix(y).T
  diff = X*theta - y
  diffSq = diff.T*diff
  mse = diffSq / len(X)
  return mse
```

In [7]:

```
mse1=msecomp(thetaset1, X1, y)
print(mse1)
mse2=msecomp(thetaset2, X2, y)
print(mse2)
```

```
[[ 0.49004382]]
[[ 0.49004372]]
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

Code for Q3 & Q4

In [1]:

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

In [2]:

```
print "Reading data..."
csvfile = file('winequality-white.csv','rb')
reader = csv.reader(csvfile)
print "done"
```

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

In [3]:

```
## Obatin the data from the file
data = []
for line in reader:
    data.append(line[0].split(';'))
del data[0]
```

In [4]:

```
## Conversion from string to float
for i in range(len(data)):
    for j in range(len(data[i])):
        data[i][j]=float(data[i][j])
```

In [5]:

```
## Obtain the quality
N=12
y=[x[N-1] for x in data]
```

In [6]:

```
## Extract the features data
for features in data:
    del features[N-1]
```

In [7]:

```
def feature(datum):
  ones = numpy.ones((len(data),1))
  feat = numpy.column_stack((ones,data))
  return feat
```

In [8]:

```
X=feature(data)
X_train=X[:len(data)/2]
X_test=X[len(data)/2:]
y_train=y[:len(data)/2]
y_test=y[len(data)/2:]
```

In [9]:

```
theta,residuals,rank,s = numpy.linalg.lstsq(X_train, y_train)
print theta
```

```
[  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 [10]:

```
# Compute the MSE

def msecomp(theta, X, y):
  theta = numpy.matrix(theta).T
  X = numpy.matrix(X)
  y = numpy.matrix(y).T
  diff = X*theta - y
  diffSq = diff.T*diff
  mse = diffSq / len(X)
  return mse
```

In [11]:

```
print msecomp(theta,X_train,y_train)
print msecomp(theta,X_test,y_test)
```

```
[[ 0.6023075]]
[[ 0.56245713]]
```

In [12]:

```
## Ablation Experiment

def ablation(index, Xset):
    XL = []
    XR = []
    for i in range(len(Xset)):
        XL.append(Xset[i][:index])
        XR.append(Xset[i][index+1:])
    ones = numpy.ones((len(Xset), 1))
    Xset_new = numpy.column_stack([XL, XR])
    return Xset_new
```

In [37]:

```
## input different feature index so obtain different training set e.g. index from [1,11]
featureindex=11
X_train_new = ablation(featureindex, X_train)
X_test_new = ablation(featureindex, X_test)
theta, residuals, rank, s = numpy.linalg.lstsq(X_train_new, y_train)
```

Code for Q5 & Q6

In [1]:

```
import numpy
import urllib
import scipy.optimize
import random
from math import exp
from math import log
import csv
from sklearn import svm
```

In [2]:

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

In [3]:

```
print "Reading data..."
csvfile = file('winequality-white.csv','rb')
reader = csv.reader(csvfile)
print "done"
```

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

In [4]:

```
data = []
for line in reader:
    data.append(line[0].split(';'))
del data[0]
```

In [5]:

```
## Conversion from string to float
for i in range(len(data)):
    for j in range(len(data[i])):
        data[i][j]=float(data[i][j])
```

In [ ]:

```

```

In [6]:

```
## Obtain the quality
N=12
y=[x[N-1] for x in data]
```

In [7]:

```
# Obtain the evaluation
y_evaluation=[]
for quality in y:
  if quality > 5:
    y_evaluation.append(1)
  else:
    y_evaluation.append(0)
```

In [8]:

```
## Extract the features data
for features in data:
    del features[N-1]
```

In [9]:

```
X=data
```

In [10]:

```
X_train = X[:len(data)/2]
y_train = y_evaluation[:len(data)/2]
X_test = X[len(data)/2:]
y_test = y_evaluation[len(data)/2:]
```

In [ ]:

```

```

In [11]:

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

Out[11]:

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

In [12]:

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

In [13]:

```
train_accuracy = sum([z[0] == z[1] for z in zip(train_predictions, y_train)]) * 1.0 / len(train_
predictions)
print 'train_accuracy= ', train_accuracy
```

```
train_accuracy=  1.0
```

In [14]:

```
test_accuracy = sum([z[0] == z[1] for z in zip(test_predictions, y_test)]) * 1.0 / len(test_pred
ictions)
print 'test_accuracy= ', test_accuracy
```

test_accuracy= 0.668027766435

In [15]:

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

In [16]:

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

In [17]:

```
# 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]
  print "ll =", loglikelihood
  return -loglikelihood
```

In [18]:

```
## NEGATIVE Derivative of log-likelihood
def fprime(theta, X, y, lam):
  dl = [0.0]*len(theta)
  for k in range(len(theta)):
      sum = 0
      for i in range(len(X)):
        # Fill in code for the derivative
        logit = inner(X[i], theta)
        sum += X[i][k] * (1-sigmoid(logit))
        if not y[i]:
            sum -= X[i][k]
      dl_temp = sum - 2 * lam * theta[k]
      dl[k] = dl_temp
  # Negate the return value since we're doing gradient *ascent*
  return numpy.array([-x for x in dl])
```

In [19]:

```
# Use a library function to run gradient descent (or you can implement yourself!)
theta,l,info = scipy.optimize.fmin_l_bfgs_b(f, [0]*len(X[0]), fprime, args = (X_train, y_train,
1.0))
print "Final log likelihood =", -l
```

```
ll = -1697.51744519
ll = -143194.900113
ll = -8508.69061169
ll = -1662.30572651
ll = -1640.40856092
ll = -1640.03814916
ll = -1639.03949975
ll = -1636.45792249
ll = -1630.77442533
ll = -1620.55739783
ll = -1608.50105879
ll = -1600.16288881
ll = -1597.29785578
ll = -1596.47865693
ll = -1594.70979598
ll = -1590.91731654
ll = -1582.41530859
ll = -1568.4144228
ll = -1572.16480918
ll = -1562.98451058
ll = -1722.1864377
ll = -1553.11941899
ll = -1551.28247863
ll = -1550.93347138
ll = -1550.92435403
ll = -1550.88829208
ll = -1550.75056169
ll = -1550.30379368
ll = -1546.82746913
ll = -1540.24521872
ll = -1532.3436015
ll = -1501.87894509
ll = -1494.96452746
ll = -1493.00529421
ll = -1492.90039231
ll = -1492.8236081
ll = -1492.71329423
ll = -1492.53289951
ll = -1492.37805602
ll = -1492.30775575
ll = -1492.26556835
ll = -1492.02672659
ll = -1490.77384179
ll = -1488.44202781
ll = -1482.2015175
ll = -1470.38581772
ll = -1453.77901942
ll = -1441.71969374
ll = -1438.45018843
ll = -1497.6694022
ll = -1438.1651363
ll = -1437.84903693
ll = -1437.46297497
ll = -1437.37965204
ll = -1437.2770751
ll = -1437.18724176
ll = -1437.13693305
ll = -1437.20154298
ll = -1437.12748193
ll = -1437.01026455
ll = -1436.82956778
```

ll = -1436.12234382
ll = -1467.48393549
ll = -1436.1057686
ll = -1435.48712826
ll = -1494.26836311
ll = -1434.54323247
ll = -1433.12702876
ll = -1444.87525346
ll = -1430.96154754
ll = -1421.59788711
ll = -1404.70361773
ll = -1393.45483117
ll = -1391.46872176
ll = -1391.33178692
ll = -1391.32033674
ll = -1419.5364793
ll = -1391.32032486
ll = -1391.31703012
ll = -1391.31424371
ll = -1391.31292207
ll = -1391.31212881
ll = -1391.31176179
ll = -1391.30953892
ll = -1391.29932174
ll = -1391.28149533
ll = -1391.22467633
ll = -1391.12476616
ll = -1392.86853154
ll = -1391.1152989
ll = -1390.92746727
ll = -1390.71081078
ll = -1390.4788801
ll = -1390.31652346
ll = -1390.11875765
ll = -1389.89408966
ll = -1389.97455642
ll = -1389.56415607
ll = -1389.47158519
ll = -1389.47089753
ll = -1389.47082693
ll = -1389.4707264
ll = -1389.47048544
ll = -1389.4700041
ll = -1389.46755555
ll = -1389.46572061
ll = -1389.5506915
ll = -1389.46547149
ll = -1389.46374688
ll = -1389.46303511
ll = -1389.46247703
ll = -1389.46141693
ll = -1389.4595981
ll = -1389.45748126
ll = -1511.31150097
ll = -1389.45741117
ll = -1389.45631385
ll = -1389.45608794
ll = -1389.45606926
ll = -1389.45606175
ll = -1389.4560267
ll = -1389.45595767

ll = -1389.45582946
ll = -1389.45566643
ll = -1389.45554151
ll = -1389.45548153
ll = -1389.4553918
ll = -1389.45559955
ll = -1389.45534273
ll = -1389.45518108
ll = -1389.45490657
ll = -1389.45481828
ll = -1389.45465738
ll = -1389.45460042
ll = -1389.45454743
ll = -1389.45438856
ll = -1389.45398493
ll = -1389.45337809
ll = -1389.73484372
ll = -1389.45312242
ll = -1389.45254589
ll = -1389.45222415
ll = -1391.12647822
ll = -1389.4520445
ll = -1389.45197024
ll = -1389.45192445
ll = -1389.45682718
ll = -1389.45188034
ll = -1389.45153404
ll = -1389.44975351
ll = -1389.44442088
ll = -1389.44323845
ll = -1389.44204319
ll = -1389.54614497
ll = -1389.44194469
ll = -1389.44056709
ll = -1389.44691568
ll = -1389.44015774
ll = -1389.43751201
ll = -1389.43067122
ll = -1389.42853926
ll = -1389.41592966
ll = -1389.41068806
ll = -1389.40723967
ll = -1389.40543651
ll = -1389.40264678
ll = -1389.40154511
ll = -1389.40107727
ll = -1389.40084335
ll = -1389.40027032
ll = -1389.39940679
ll = -1391.02276802
ll = -1389.39934772
ll = -1389.3981502
ll = -1389.3969698
ll = -1389.39613027
ll = -1389.39529023
ll = -1389.39486145
ll = -1389.3944074
ll = -1389.39215376
ll = -1389.38761389
ll = -1389.37324565
ll = -1389.34229406

ll = -2620.11597732
ll = -1389.34209837
ll = -1389.28805567
ll = -1389.89265836
ll = -1389.281938
ll = -1389.21857467
ll = -1389.18345994
ll = -1389.17261971
ll = -1391.95789189
ll = -1389.16935597
ll = -1389.16216602
ll = -1389.14753171
ll = -1389.13142894
ll = -1389.09315019
ll = -1389.05978986
ll = -1388.98956236
ll = -1388.94098394
ll = -1389.13841257
ll = -1388.91926629
ll = -1388.89386094
ll = -1388.89167182
ll = -1388.88832844
ll = -1388.88736141
ll = -1388.88538001
ll = -1388.88769831
ll = -1388.88403446
ll = -1388.88161422
ll = -1388.87279575
ll = -1388.87015696
ll = -1388.87060657
ll = -1388.86943966
ll = -1388.87023023
ll = -1388.86878811
ll = -1388.86858095
ll = -1388.86710061
ll = -1388.8652977
ll = -1388.86218509
ll = -1388.85775395
ll = -1388.9512525
ll = -1388.85707261
ll = -1388.85139426
ll = -1388.84685376
ll = -1388.84519005
ll = -1388.84460278
ll = -1392.06401741
ll = -1388.84456095
ll = -1388.84411519
ll = -1388.84274347
ll = -1388.83980052
ll = -1388.83545289
ll = -1388.83163881
ll = -1388.83152022
ll = -1388.83087815
ll = -1388.82959796
ll = -1388.82813424
ll = -1388.83650215
ll = -1388.82754333
ll = -1388.82617121
ll = -1405.21819103
ll = -1388.82607567
ll = -1388.82349739

```
ll = -1389.0884086
ll = -1388.82119169
ll = -1388.81694808
ll = -1388.80417909
ll = -1388.80058901
ll = -1388.81787993
ll = -1388.79979554
ll = -1388.79689435
ll = -1388.79477507
ll = -1388.79259795
ll = -1388.79133551
ll = -1388.79522438
ll = -1388.79093038
ll = -1388.78909867
ll = -1388.78296099
ll = -1388.75596013
ll = -1388.74061787
ll = -1388.70546509
ll = -1388.69736541
ll = -1388.8720161
ll = -1388.69730554
ll = -1388.69607359
ll = -1388.69585577
ll = -1388.69570828
ll = -1388.69527348
ll = -1388.69479849
ll = -1388.76384258
ll = -1388.69478015
ll = -1388.6943154
ll = -1388.69413596
ll = -1388.6941008
ll = -1388.69409207
ll = -1388.6940743
ll = -1388.69404143
ll = -1388.69469526
ll = -1388.69403841
Final log likelihood = -1388.69403841
```

In [20]:

```
train_predictions = []
for i in range(len(X)/2):
    if inner(X_train[i], theta) > 0:
        train_predictions.append(1)
    else:
        train_predictions.append(0)
```

In [21]:

```
train_accuracy = sum([z[0] == z[1] for z in zip(train_predictions, y_train)]) * 1.0 / len(train_
predictions)
print 'train_accuracy= ', train_accuracy
```

```
train_accuracy=  0.71335238873
```

In [22]:

```
test_predictions = []
for i in range(len(X)/2):
    if inner(X_test[i], theta) > 0:
        test_predictions.append(1)
    else:
        test_predictions.append(0)
```

In [23]:

```
test_accuracy = sum([z[0] == z[1] for z in zip(test_predictions, y_test)]) * 1.0 / len(test_pred
ictions)
print 'test_accuracy= ', test_accuracy
```

```
test_accuracy=  0.76929358922
```

In [ ]:

```

```