CSE258 Homework 1

Chenyu Huang, University of California San Diego 01/22/2017

1 Regression

Task 1 The fitted value of θ_0 and θ_1 are $[-3.91707489e+01, 2.14379786e-02]$

Task 2 Since the data entries only covers years from 1999 to 2012, We chose to represent the year value as a step function. In this representation, $[1,0,0,0,0,0,0,0,0,0,0,0,0,0]$ represents 1999, while $[0,0,0,0,0,0,0,0,0,0,0,0,0,1]$ represents 2012. With this representation, the relationship between year and review can be written as:

$$review/overall \simeq \sum_{i=0}^{13} \theta_i \times year[i] \tag{1}$$

$$where\ year[i]\ represent\ the\ ith\ digit\ of\ the\ year\ representation \tag{2}$$

The MSE in part 1 of the question is 0.490043819858, while the MSE using the step function representation of year value is 0.48915189521.

Task 3 The fitted coefficients are

$$\begin{bmatrix} 2.56420278e+02 \\ 1.35421303e-01 \\ -1.72994866e+00 \\ 1.02651152e-01 \\ 1.09038568e-01 \\ -2.76775152e-01 \\ 6.34332169e-03 \\ 3.85023935e-05 \\ -2.58652808e+02 \\ 1.19540565e+00 \\ 8.33006284e-01 \\ 9.79304364e-02 \end{bmatrix}$$

The MSE for the training data is 0.602307502903, while the MSE on the testing data is 0.562457127767.

Task 4 (a) The MSE for 11 ablation experiments are

$$\begin{bmatrix} 0.559113415175 \\ 0.596384849311 \\ 0.562221703459 \\ 0.55362506398 \\ 0.562629269948 \\ 0.5561408204 \\ 0.56242900712 \\ 0.544726553466 \\ 0.559566626638 \\ 0.557346348772 \\ 0.573214743821 \end{bmatrix}$$

(b) The larger the MSE with ablation, the more additional information that ablation provides. By the same principle, the smaller the MSE with ablation, the less predictive power that feature will possess. The second feature "volatile acidity" provides the most additional information. While the "density" provides the least additional information.

2 Classification

Task 5 The accuracy of the classifier on the training data is 1.0, while the accuracy on the testing data is 0.668027766435.

Task 6 After convergence, the log likelihood is -1388.69674843, while the accuracy on the testing data is 0.76929358922.

3 Code

Listing 1: task1

```
import numpy #maxtrix operation and linear algebra
import urllib #loading data from web
import scipy.optimize
import random

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

print "Reading data..."
data = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.
    json"))
print "done"

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]

X = numpy.matrix(X)
y = numpy.matrix(y)
thetas = numpy.linalg.inv(X.T * X) * X.T * y.T

predicted = X * thetas
predicted = predicted.T
def mean_squared_error(a, b):
    c = numpy.subtract(a, b)
    c = c ** 2
    return numpy.sum(c) / len(c)

#0.490043819858
print mean_squared_error(predicted.A1, y.A1)

#[ -3.91707489e+01,   2.14379786e-02]
print thetas
```

Listing 2: task2

```
import numpy #maxtrix operation and linear algebra
import urllib #loading data from web
import scipy.optimize
import random
import math

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

print "Reading data..."
data = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.
    json"))
print "done"

def feature(datum):
  feat = []
  f = datum['review/timeStruct']['year']
  for i in range(1999, 2013):
      if f == i:
          feat.append(1)
      else:
          feat.append(0)
  return feat

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

X = numpy.matrix(X)
y = numpy.matrix(y)
thetas = numpy.linalg.inv(X.T * X) * X.T * y.T

predicted = X * thetas
predicted = predicted.T

def mean_squared_error(a, b):
    c = numpy.subtract(a, b)
    c = c ** 2
    return numpy.sum(c) / len(c)

#0.48915189521
print mean_squared_error(predicted.A1, y.A1)

#[4.05154639  0.04845361  0.26663543  0.2247694  -0.29273378 -0.33610958
    -0.29110335 -0.19133134 -0.24438704 -0.20546893 -0.16067206 -0.1301398
    -0.15473656 -0.11142557]
print thetas.T
```

Listing 3: task3

```
import numpy #maxtrix operation and linear algebra
import urllib #loading data from web
import csv
import scipy.optimize
import random

with open('winequality-white.csv', 'r') as f:
    reader = csv.reader(f, delimiter=';')
    wine = []
    for row in reader:
        wine = wine + [row]
f.close()

def dataProcessing(w):
    w = w[1:]
    for i in range(0, len(w)):
        for j in range(0, len(w[0])):
            w[i][j] = float(w[i][j])
    return w

print "Reading and processing data..."
wine = dataProcessing(wine)
train = wine[:len(wine)/2]
test = wine[len(wine)/2:]
print "done"

def feature(datum):
  feat = [1]
  for i in range(0, 11):
    feat.append(datum[i])
  return feat

#compute MSE
def mean_squared_error(a, b):
    c = numpy.subtract(a, b)
    c = c ** 2
    return numpy.sum(c) / len(c)

#load X and y with training data
X = [feature(d) for d in train]
y = [d[11] for d in train]

X = numpy.matrix(X)
y = numpy.matrix(y)
thetas = numpy.linalg.inv(X.T * X) * X.T * y.T

predicted = X * thetas
predicted = predicted.T
```

```
print mean_squared_error(predicted.A1, y.A1)

#update X and y with test data
X = [feature(d) for d in test]
y = [d[11] for d in test]

X = numpy.matrix(X)
y = numpy.matrix(y)

predicted = X * thetas
predicted = predicted.T



# 0.562457127767
print mean_squared_error(predicted.A1, y.A1)

# [[  2.56420278e+02   1.35421303e-01  -1.72994866e+00   1.02651152e-01
#     1.09038568e-01  -2.76775152e-01   6.34332169e-03   3.85023935e-05
#    -2.58652808e+02   1.19540565e+00   8.33006284e-01   9.79304364e-02]]
# print thetas.T
```

Listing 4: task4

```
import numpy #maxtrix operation and linear algebra
import urllib #loading data from web
import csv
import scipy.optimize
import random

with open('winequality-white.csv', 'r') as f:
    reader = csv.reader(f, delimiter=';')
    wine = []
    for row in reader:
        wine = wine + [row]
f.close()

def dataProcessing(w):
    w = w[1:]
    for i in range(0, len(w)):
        for j in range(0, len(w[0])):
            w[i][j] = float(w[i][j])
    return w

print "Reading and processing data..."
wine = dataProcessing(wine)
train = wine[:len(wine)/2]
test = wine[len(wine)/2:]
print "done"

def feature(datum, ablation):
  feat = [1]
  for i in range(0, 11):
      if i == ablation:
          continue
      else:
          feat.append(datum[i])
  return feat


#compute MSE
def mean_squared_error(a, b):
    c = numpy.subtract(a, b)
    c = c ** 2
    return numpy.sum(c) / len(c)

def compute_MSE(i):
    #load X and y with training data
    X = [feature(d, i) for d in train]
    y = [d[11] for d in train]

    X = numpy.matrix(X)
    y = numpy.matrix(y)
```

```
    thetas = numpy.linalg.inv(X.T * X) * X.T * y.T

    #update X and y with test data
    X = [feature(d, i) for d in test]
    y = [d[11] for d in test]

    X = numpy.matrix(X)
    y = numpy.matrix(y)

    predicted = X * thetas
    predicted = predicted.T
    print mean_squared_error(predicted.A1, y.A1)
    return mean_squared_error(predicted.A1, y.A1)

MSE = [compute_MSE(i) for i in range(0, 11)]
```

Listing 5: task5

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

with open('winequality-white.csv', 'r') as f:
    reader = csv.reader(f, delimiter=';')
    wine = []
    for row in reader:
        wine = wine + [row]
f.close()

# Read data from CSV file
def dataProcessing(w):
    w = w[1:]
    for i in range(0, len(w)):
        for j in range(0, len(w[0])):
            w[i][j] = float(w[i][j])
    return w

# Split data into test data and training data
print "Reading and processing data..."
wine = dataProcessing(wine)
train = wine[:len(wine)/2]
test = wine[len(wine)/2:]
print "done"

# Extract the features vector for each wine entry
def feature(datum):
  feat = []
  for i in range(0, 11):
    feat.append(datum[i])
  return feat

# Read the feature vector and the label data from training data
X_train = [feature(d) for d in train]
y_train = [d[11] > 5 for d in train]

X_test = [feature(d) for d in test]
y_test = [d[11] > 5 for d in test]

# print X_train


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

```
clf.fit(X_train, y_train)

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


def count_acc(predict, original):
    count = 0.0
    for i in range(0, len(predict)):
        if predict[i] == original[i]:
            count = count + 1
    return count

# 1.0
# 0.668027766435
print count_acc(train_predictions.tolist(), y_train) / len(y_train)
print count_acc(test_predictions.tolist(), y_test) / len(y_test)
```

Listing 6: task6

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

with open('winequality-white.csv', 'r') as f:
    reader = csv.reader(f, delimiter=';')
    wine = []
    for row in reader:
        wine = wine + [row]
f.close()

# Read data from CSV file
def dataProcessing(w):
    w = w[1:]
    for i in range(0, len(w)):
        for j in range(0, len(w[0])):
            w[i][j] = float(w[i][j])
    return w

# Split data into test data and training data
print "Reading and processing data..."
wine = dataProcessing(wine)
print "done"

def feature(datum):
  feat = []
  for i in range(0, 11):
    feat.append(datum[i])
  return feat

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))

# 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

# 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)
    M =  [x * numpy.exp(-logit) / (1 + numpy.exp(-logit)) for x in X[i]]
    dl = [d + m for (d, m) in zip(dl, M)]
    # dl = dl + X[i] * numpy.exp(-logit) / (1 + numpy.exp(-logit))
    if not y[i]:
        dl = [d - x for (d, x) in zip(dl, X[i])]
        # dl = dl - X[i]
  R = [t * 2 * lam for t in theta]
  dl = [d - r for (d, r) in zip(dl, R)]
  # Negate the return value since we're doing gradient *ascent*
  return numpy.array([-x for x in dl])


X = [feature(d) for d in wine]
# Extract features and labels from the data
y = [d[11] > 5 for d in wine]

y_train = y[:len(y)/2]
y_test = y[len(y)/2:]

X_train = X[:len(X)/2]
X_test = X[len(X)/2:]

# 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
    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
# -1388.69674843

theta = numpy.matrix(theta)
X_test = numpy.matrix(X_test)

y_predict = X_test * theta.T
y_predict = [y > 0.0 for y in y_predict]

def count_acc(predict, original):
    count = 0.0
```

```
    for i in range(0, len(predict)):
        if predict[i] == original[i]:
            count = count + 1
    return count

# print count_acc(y_predict, y_test) / len(y_test)
print "Accuracy = ", count_acc(y_predict, y_test) / len(y_test)
# Compute the accuracy 0.76929358922
```