Recommender Sys & Web Mining Hw1

Hao-en Sung (wrangle1005@gmail.com)
Department of Computer Science
University of California, San Diego
San Diego, CA 92092

Abstract

This is the report for CSE 258 Hw1.

1 Regression (week 1):

1.1 Problem 1

The obtained parameter are listed as follows.

$$\theta_{\text{constant}} \approx -3.91707489e+01$$
$$\theta_{\text{year}} \approx 2.14379786e-02$$

I implemented the algorithm as **Task** 1 in Code 1, which is attached in Appendix.

1.2 Problem 2

A proper way is to include features like year^2, year^3, and $\log(\text{year})$. After introducing those addition features, I can sufficiently reduce the MSE from 0.490044 to 0.489612 with parameters listed as follows.

$$\theta_{\text{constant}} \approx 5.80699245e+04$$
$$\theta_{\text{year}} \approx 4.39558190e+00$$
$$\theta_{\text{year}^2} \approx 1.76583362e-02$$
$$\theta_{\text{year}^3} \approx -6.95247398e-03$$
$$\theta_{\log(\text{year})} \approx -8.79879103e+03$$

I implemented the algorithm as **Task** 1 in Code 1, which is attached in Appendix.

1.3 Problem 3

After fitting the model, I achieve around 0.602308 on training data and around 0.562457 on testing data in terms of MSE score.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

The fitted coefficients are listed as follows.

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

I implemented the algorithm as **Task** 3 in Code 1, which is attached in Appendix.

1.4 Problem 4

The MRE scores of eleven experiments performed on testing data are listed as follows.

without 'fixed acidity' : 0.559113
without 'volatile acidity' : 0.592560
without 'citric acid' : 0.559028
without 'residual sugar' : 0.555046
without 'chlorides' : 0.559357
without 'free sulfur dioxide' : 0.550574
without 'total sulfur dioxide' : 0.558942
without 'density' : 0.540389
without 'pH' : 0.559135
without 'sulphates' : 0.555329
without 'alcohol' : 0.588019

Based on the MSE scores from our experiments, it is clear that MSE score drops the most without 'volatile acidity' and the least without 'density'. It is noticeable that most of the feature removals, including 'density', actually improves our model performance on testing data. Thus, we can conclude that 'volatile acidity' is the most important feature; while 'density' is the least important one.

I implemented the algorithm as **Task** 4 in Code 1, which is attached in Appendix.

2 Classification (week 2):

2.1 Problem 5

With parameter $C = 1$, I achieve 0.912209 accuracy on training data and 0.690078 on testing data.

I implemented the algorithm as **Task** 5 in Code 1, which is attached in Appendix.

2.2 Problem 6

To solve this problem, I basically re-implement all the codes for logistic regression. For *fprime*, I mainly rewrite it in my code as

```
nw = np.dot((y-p).transpose(),x) - lb1*np.sign(w) - lb2*2*w.
```
(1)

After the model convergence, I achieve -1414.647324 in terms of log-likelihood and 0.697836 accuracy on testing data.

I implemented the algorithm as **Task** 6 in Code 1, which is attached in Appendix.

Appendix

Code Listing 1: Code for Hw1

```
import numpy as np
import pandas as pd
import urllib2
import scipy.optimize
import random
from sklearn import svm
from sklearn import metrics
from sklearn import preprocessing


## Task 1
def parseData(fname):
  for l in urllib2.urlopen(fname):
    yield eval(l)

# read data
data = list(parseData('file:///Users/hogan/Google Drive/Courses/UCSD/
                                    258/Hw1/dat/beer_50000.json'))

# generate features
ndata = [d for d in data if d.has_key('review/timeStruct')]
X = [[1, d['review/timeStruct']['year']] for d in ndata]
y = [d['review/overall'] for d in ndata]

# fit model
theta,residuals,rank,s = np.linalg.lstsq(X, y)
print 'Coefficients:', theta
print 'MSE:', residuals[0] / len(ndata)


## Task 2

# generate features
ndata = [d for d in data if d.has_key('review/timeStruct')]
X = [[1, d['review/timeStruct']['year'], \
        d['review/timeStruct']['year']^2, \
        d['review/timeStruct']['year']^3, \
        np.log(d['review/timeStruct']['year'])] for d in ndata]
y = [d['review/overall'] for d in ndata]

# fit model
theta,residuals,rank,s = np.linalg.lstsq(X, y)
print 'Coefficients:', theta
print 'MSE:', residuals[0] / len(ndata)


## Task 3
# read data
data = pd.read_csv('../dat/winequality-white.csv', sep=';')
nr, nc = data.shape

# generate features
X = data[['fixed acidity','volatile acidity','citric acid',\
        'residual sugar','chlorides','free sulfur dioxide',\
        'total sulfur dioxide','density','pH','sulphates','alcohol']]
```

```
X['constant'] = np.ones([data.shape[0], 1])
y = data[['quality']]

# split data
tn_X = X[:nr/2]
tn_y = y[:nr/2]
tt_X = X[nr/2:]
tt_y = y[nr/2:]

# fit model
theta,residuals,rank,s = np.linalg.lstsq(tn_X, tn_y)
print 'Coefficients:', theta
print 'MSE for tn:', residuals[0] / (nr/2)

# predict test
tt_p = tt_X.dot(theta)
tt_p.columns = ['predict']

# calculate error
print 'MSE for tt:', \
        np.linalg.norm(tt_p['predict']-tt_y['quality'])**2 / (nr/2)


## Task 4
# enumerate dropped feature
for i in range(11):
    # generate features
    colnames = [list(X)[idx] for idx in range(1,i) + range(i+1,12)]
    nX = X[colnames]

    # split data
    tn_X = nX[:nr/2]
    tn_y = y[:nr/2]
    tt_X = nX[nr/2:]
    tt_y = y[nr/2:]

    # fit model
    theta,residuals,rank,s = np.linalg.lstsq(tn_X, tn_y)
    #print 'Coefficients:', theta
    #print 'MSE for tn:', residuals[0] / (nr/2)

    # predict test
    tt_p = tt_X.dot(theta)
    tt_p.columns = ['predict']

    # calculate error
    tt_e = np.linalg.norm(tt_p['predict']-tt_y['quality'])**2 / (nr/2)
    print 'MSE for tt (w/o ' + list(X)[i] + '):', tt_e


## Task 5
# create labels
y = (y > 5).astype(int)

# split data
tn_X = X[:nr/2]
tn_y = y[:nr/2].iloc[:,0]
tt_X = X[nr/2:]
tt_y = y[nr/2:].iloc[:,0]

# learn
clf = svm.SVC(C=1)
clf.fit(tn_X, tn_y)

# predict
```

```
tn_p = clf.predict(tn_X)
tt_p = clf.predict(tt_X)

# calculate error
tn_a = metrics.accuracy_score(tn_y, tn_p)
print 'ACC for tn', tn_a
tt_a = metrics.accuracy_score(tt_y, tt_p)
print 'ACC for tt', tt_a


## Task 6
eps = 1e-10

# utility functions
def calLOSS(w, x, y, p, lb1, lb2):
    assert(x.shape[0] == y.shape[0])

    val = 0.0
    for i in xrange(x.shape[0]):
        val -= y[i] * np.log(p[i]+eps) + (1-y[i]) * np.log(1-p[i]+eps)

    val += lb1 * np.linalg.norm(w, 1)
    val += lb2 * np.linalg.norm(w, 2)
    return val

def calLL(x, y, p):
    assert(x.shape[0] == y.shape[0])

    val = 0.0
    for i in xrange(x.shape[0]):
        val += y[i] * np.log(p[i]+eps) + (1-y[i]) * np.log(1-p[i]+eps)
    return val

def calACC(w, x, y, p):
    assert(x.shape[0] == y.shape[0] and y.shape[0] == p.shape[0])
    return 1.0 / x.shape[0] * np.sum(y == (p > 0.5))

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

def predict(w, x):
    v = np.dot(x, np.transpose(w))
    return np.apply_along_axis(sigmoid, 0, v)

def LogisticRegression(x, y, lb1=0, lb2=0, \
        MAX_ITER=200, eta=0.001, eta_decay=0.01):
    assert(x.shape[0] == y.shape[0])
    ft_n = x.shape[1]

    w = np.zeros((1, ft_n))
    p = predict(w, x);
    e = calLOSS(w, x, y, p, lb1, lb2)

    for t in xrange(MAX_ITER):
        nw = np.dot((y-p).transpose(),x) - lb1*np.sign(w) - lb2*2*w

        n_eta = eta / (1 + t * eta_decay)
        w += n_eta * nw

        # predict
        p = predict(w, x)

        # calculate err
        err = calLOSS(w, x, y, p, lb1, lb2)
```

```
        # stop condition
        if err < 0.1 * e:
            print 'Stop at #' + str(t) + ' with log-likelihood ' \
                    + str(calLL(x, y, p))
            return w

    print 'Maximum iterations are reached with log-likelihood ' \
            + str(calLL(x, y, p))

    return w

# transform to ndarray
tn_X = tn_X.as_matrix()
tn_y = tn_y.as_matrix().reshape((-1,1))
tt_X = tt_X.as_matrix()
tt_y = tt_y.as_matrix().reshape((-1,1))

## preprocessing
tn_X = preprocessing.scale(tn_X)
tt_X = preprocessing.scale(tt_X)

# learn
w = LogisticRegression(tn_X, tn_y)

# predict
tn_p = predict(w, tn_X)
tt_p = predict(w, tt_X)

# calculate accuracy
tn_a = calACC(w, tn_X, tn_y, tn_p)
print 'ACC for tn', tn_a
tt_a = calACC(w, tt_X, tt_y, tt_p)
print 'ACC for tt', tt_a
```