Recommender Sys & Web Mining Hw2

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

Abstract

This is a report for CSE 258 Hw1. Notice that the codes for every problem are attached as Code 1 in the Appendix.

Tasks (Classifier evaluation)

Problem 1

The model performance on data without random shuffle is shown as follows. It can be found that the valid set performance is not properly aligned to test set performance. It may be caused by the non-uniform distributed data set.

lambda	train	valid	test
0	0.732843137255	0.720759338641	0.778322106552
0.01	0.732230392157	0.721984078383	0.780159216167
1.0	0.726715686275	0.704225352113	0.766074709124
100.0	0.658700980392	0.630128597673	0.696876913656

Table 1: Performance on Non-shuffled Data

On the other hand, once the data set are sorted, one can have the following results, which are more reasonable.

lambda	train	valid	test
0	0.751225490196	0.757501530925	0.738518064911
0.01	0.748774509804	0.757501530925	0.738518064911
1.0	0.729166666667	0.753827311696	0.72933251684
100.0	0.66237745098	0.681567666871	0.680342927128

Table 2: Performance on Shuffled Data

Problem 2

For test set, there are 946 true positives, 287 false positives, 260 true negatives, and 140 false negatives; while the *Balanced Error Rate* is around 0.326797.

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

Problem 3

The precision and recall scores of top 10, 500, and 500 confident label for test set are recorded as follows.

Top #	Precision	Recall
10	1.000000	0.009208
500	0.902000	0.415285
1000	0.813000	0.748619

Table 3: Most Confident Label

Problem 4

The precision-recall figure for test set is shown as follows.

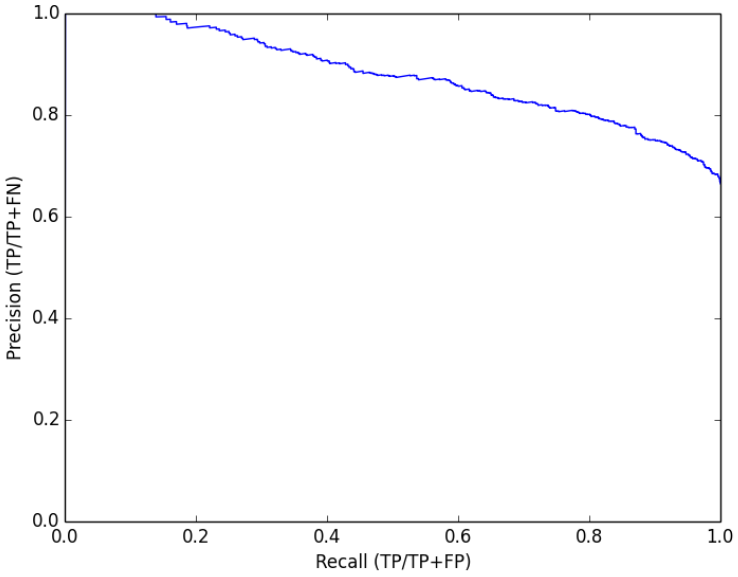

Figure 1: Precision and Recall

Tasks (Dimensionality reduction)

Problem 5

The calculated reconstruction error between X and X_{mean} is 3554264.46181.

Problem 6

The PCA transform matrix $M \in \mathbb{R}^{12\times 5}$ is shown as follows.

Problem 7

The calculated reconstruction error between X and reconstructed X for using five PCA components is 261.040523.

feature	n0	n1	n2	n3	n4
f0	0.000000	0.000000	0.000000	0.000000	0.000000
f1	0.001544	-0.009163	0.012900	-0.147658	0.984965
f2	0.000169	-0.001545	0.000929	0.015452	-0.003978
f3	0.000339	0.000140	0.001258	-0.005005	0.041692
f4	0.047328	0.014943	0.995192	0.084200	-0.000808
f5	0.000098	-0.000072	0.000078	-0.006573	-0.001498
f6	0.261877	0.964685	-0.026393	-0.006381	0.007875
f7	0.963858	-0.262737	-0.042789	0.010614	-0.001753
f8	0.000036	-0.000018	0.000447	-0.001152	0.000328
f9	0.000003	-0.000042	-0.007017	0.017027	-0.075506
f10	0.000341	-0.000361	-0.002142	0.002601	-0.003538
f11	-0.012504	0.006455	-0.082723	0.985063	0.149361

Table 4: PCA Transform Matrix

Problem 8

I enumerate models with different numbers of PCA components. As shown in the following figure, the more PCA components are used, the lower the mean squared error (MSE).

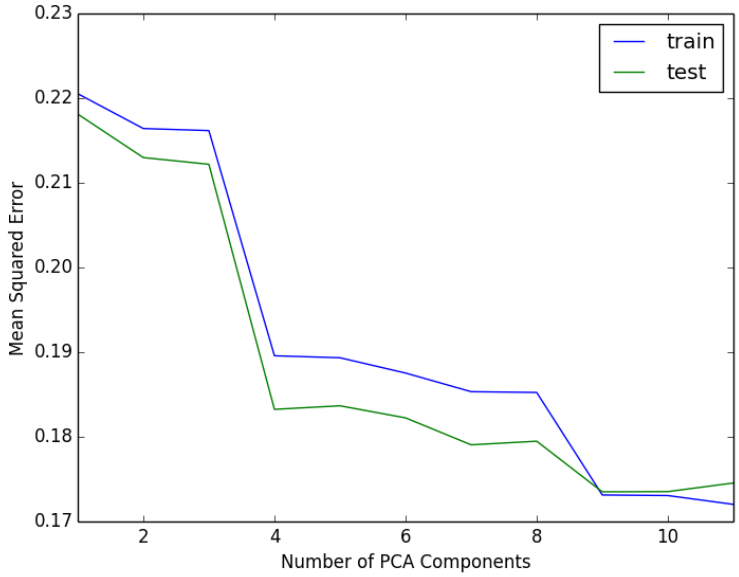

Figure 2: MSE with increasing number of PCA

Appendix

Code Listing 1: Code for Hw2

```
import numpy as np
import urllib
import scipy.optimize
import random
from math import exp
from math import log
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression

random.seed(0)

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

print "Reading data..."
dataFile = open("../dat/winequality-white.csv")
header = dataFile.readline()
fields = ["constant"] + header.strip().replace('"','').split(';')
featureNames = fields[:-1]
labelName = fields[-1]
lines = [[1.0] + [float(x) for x in l.split(';')] for l in dataFile]
X = [l[:-1] for l in lines]
y = [l[-1] > 5 for l in lines]
r = [l[-1] for l in lines]
print "done"

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))

##################################################
# Logistic regression by gradient ascent         #
##################################################

# 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]
  # for debugging
  # print "ll =", loglikelihood
  return -loglikelihood

# NEGATIVE Derivative of log-likelihood
def fprime(theta, X, y, lam):
  dl = [0]*len(theta)
  for i in range(len(X)):
    logit = inner(X[i], theta)
    for k in range(len(theta)):
      dl[k] += X[i][k] * (1 - sigmoid(logit))
      if not y[i]:
        dl[k] -= X[i][k]
```

```
  for k in range(len(theta)):
    dl[k] -= lam*2*theta[k]
  return np.array([-x for x in dl])


##################################################
# Train                                          #
##################################################

def train(lam):
  theta,_,_ = scipy.optimize.fmin_l_bfgs_b(f, [0]*len(X[0]),
          fprime, pgtol = 10, args = (X_tn, y_tn, lam))
  return theta

##################################################
# Predict                                        #
##################################################

def score(theta):
  s_tn = [inner(theta,x) for x in X_tn]
  s_vl = [inner(theta,x) for x in X_vl]
  s_tt = [inner(theta,x) for x in X_tt]
  return s_tn, s_vl, s_tt

def predict(theta):
  s_tn, s_vl, s_tt = score(theta)

  p_tn = [s > 0 for s in s_tn]
  p_vl = [s > 0 for s in s_vl]
  p_tt = [s > 0 for s in s_tt]
  return p_tn, p_vl, p_tt

def calACC(theta):
  p_tn, p_vl, p_tt = predict(theta)

  correct_tn = [(a==b) for (a,b) in zip(p_tn,y_tn)]
  correct_vl = [(a==b) for (a,b) in zip(p_vl,y_vl)]
  correct_tt = [(a==b) for (a,b) in zip(p_tt,y_tt)]

  acc_tn = sum(correct_tn) * 1.0 / len(correct_tn)
  acc_vl = sum(correct_vl) * 1.0 / len(correct_vl)
  acc_tt = sum(correct_tt) * 1.0 / len(correct_tt)
  return acc_tn, acc_vl, acc_tt

def calPERF(y, s):
  ys =  sorted(zip(y,s), key=lambda x: x[1], reverse=True)
  y, _ = zip(*ys)

  TP, TPLst = 0, []
  FP, FPLst = 0, []
  TN, TNLst = 0, []
  FN, FNLst = 0, []
  precisionLst = []
  recallLst = []

  # Default Threshold
  for i in range(len(y)):
    if y[i] == 1:
      FN += 1
    else:
      TN += 1
  TPLst.append(TP)
  FPLst.append(FP)
  TNLst.append(TN)
  FNLst.append(FN)
```

```
  precisionLst.append(0)
  recallLst.append(0)

  # Move Threshold Downward
  for i in range(len(y)):
    if y[i] == 1:
      TP += 1
      FN -= 1
    if y[i] == 0:
      TN -= 1
      FP += 1
    TPLst.append(TP)
    FPLst.append(FP)
    TNLst.append(TN)
    FNLst.append(FN)
    precisionLst.append(1.0*TP/(TP+FP))
    recallLst.append(1.0*TP/(TP+FN))

  return TPLst, FPLst, TNLst, FNLst, precisionLst, recallLst

##################################################
# Classifier Evaluation                          #
##################################################

# Shuffle Data
data = zip(X, y, r)
random.shuffle(data)
X, y, r = zip(*data)

# Partition Data
X_tn = X[:int(len(X)/3)]
y_tn = y[:int(len(y)/3)]
r_tn = r[:int(len(y)/3)]
X_vl = X[int(len(X)/3):int(2*len(X)/3)]
y_vl = y[int(len(y)/3):int(2*len(y)/3)]
r_vl = r[int(len(y)/3):int(2*len(y)/3)]
X_tt = X[int(2*len(X)/3):]
y_tt = y[int(2*len(X)/3):]
r_tt = r[int(2*len(X)/3):]

## Problem 1
for lam in [0, 0.01, 1.0, 100.0]:
  theta = train(lam)

  acc_tn, acc_vl, acc_tt = calACC(theta)
  print("lambda = " + str(lam) + ";\ttrain=" + str(acc_tn)
          + "; valid=" + str(acc_vl) + "; test=" + str(acc_tt))

## Problem 2
lam = 0.01
theta = train(lam)
_, _, s_tt = score(theta)
_, _, p_tt = predict(theta)

TPLst, FPLst, TNLst, FNLst, _, _ = calPERF(y_tt, s_tt)
num = sum([1 if v > 0 else 0 for v in s_tt])
TP, FP, TN, FN = TPLst[num], FPLst[num], TNLst[num], FNLst[num]
BER = 0.5 * (1.0*FP/(TN+FP) + 1.0*FN/(FN+TP))
print("Test Performance:")
print("TP=" + str(TP) + "; TN=" + str(TN) + "; FP=" + str(FP)
        + "; FN=" + str(FN) + "; BER=" + str(BER))

## Problem 3
for n in [10, 500, 1000]:
  _, _, _, _, precisionLst, recallLst = calPERF(y_tt, s_tt)
```

```
  print("With " + str(n) + " confident label:\tPrecision="
          + str(precisionLst[n]) + "; Recall=" + str(recallLst[n]))

## Problem 4
_, _, _, _, precisionLst, recallLst = calPERF(y_tt, s_tt)
plt.figure()
plt.plot(recallLst, precisionLst)
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.savefig('../res/precision_recall.png')

##################################################
# Dimensionality reduction                       #
##################################################

## Problem 5
Xmean = np.mean(X, axis=0)
err = np.sum((X_tn - Xmean) ** 2)
print("Reconstuction Error for Mean: " + str(err))

## Problem 6
np.set_printoptions(suppress=True, precision=6)
pca = PCA()
pca.fit(X)
print("PCA Tranform Matrix:")
print pca.components_[:5,:].T

## Problem 7
tX = (X - pca.mean_).dot(pca.components_[:5,:].T)
iX = tX.dot(pca.components_[:5,:]) + pca.mean_
err = np.sum((X - iX) ** 2)
print("Reconstuction Error for PCA: " + str(err))

## Problem 8
tnMSE = []
ttMSE = []
for i in range(1, 12):
  tX_tn = (X_tn - pca.mean_).dot(pca.components_[:i,:].T)
  tX_tt = (X_tt - pca.mean_).dot(pca.components_[:i,:].T)
  nX_tn = np.concatenate([np.ones((tX_tn.shape[0],1)), tX_tn], axis=1)
  nX_tt = np.concatenate([np.ones((tX_tt.shape[0],1)), tX_tt], axis=1)

  regr = LinearRegression()
  regr.fit(nX_tn, y_tn)
  tnMSE.append(np.mean((regr.predict(nX_tn) - y_tn) ** 2))
  ttMSE.append(np.mean((regr.predict(nX_tt) - y_tt) ** 2))

plt.figure()
plt.plot(range(1,12), tnMSE, label='train')
plt.plot(range(1,12), ttMSE, label='test')
plt.xlim([1,11])
plt.xlabel('Number of PCA Components')
plt.ylabel('Mean Squared Error')
plt.legend(loc=1)
plt.savefig('../res/mse.png')
```