CSE258 Homework 2

Chenyu Huang, University of California San Diego 02/05/2017

1 Classifier evaluation

Task 1 After randomly reshuffle the data, the performance for $\lambda = [0, 0.01, 1.0, 100]$ are as follows:

```
Chenyus-MacBook-Pro:Homework2 chenyu$ python p1.py
Reading data...
done
lambda = 0;     train=0.748774509804; validate=0.757501530925; test=0.738518064911
lambda = 0.01;  train=0.748774509804; validate=0.757501530925; test=0.739742804654
lambda = 1.0;   train=0.729166666667; validate=0.753827311696; test=0.72933251684
lambda = 100.0; train=0.66237745098; validate=0.681567666871; test=0.680342927128
```

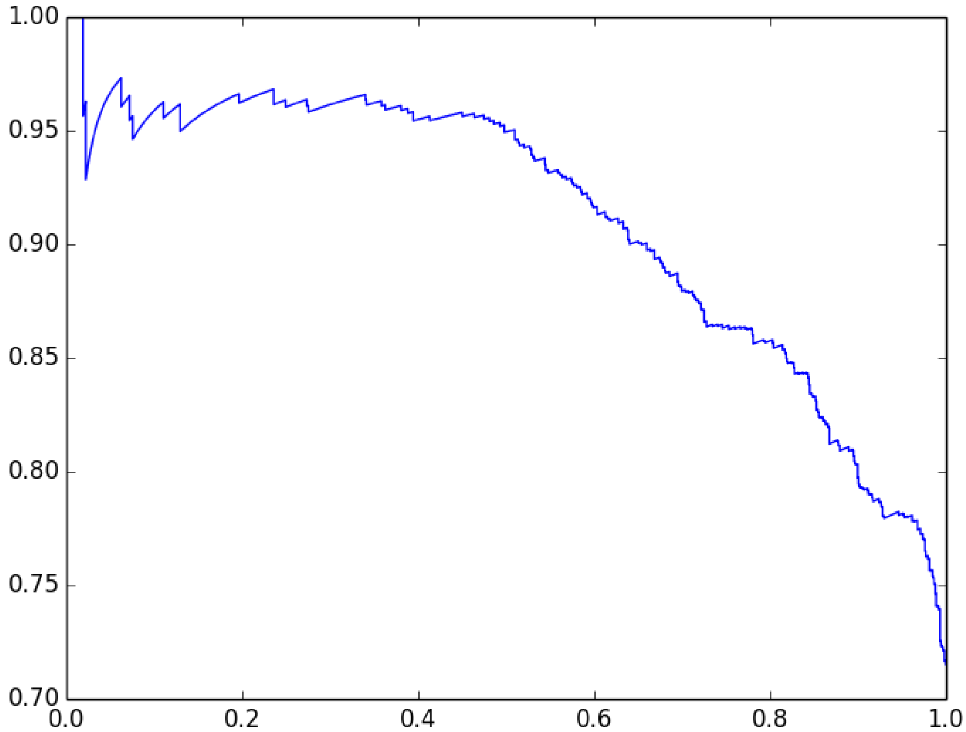
Task 2 The number of true positive, true negative, false positive, false negatives and the balanced error rate are as follows:

```
Chenyus-MacBook-Pro:Homework2 chenyu$ python p2.py
Reading data...
done
true positive: 1129
true negative: 145
false positive: 321
false negative: 38
The balanced error rate is: 0.360701663412
Chenyus-MacBook-Pro:Homework2 chenyu$
```

Task 3 After sorting the prediction according to confidence, the precision and recall values for $k = [10, 500, 1000]$ predictions are:

```
Chenyus-MacBook-Pro:Homework2 chenyu$ python p3.py
Reading data...
done
For top 10 predictions, the precision is 1.0 ,the recall is 0.00856898029135
For top 500 predictions, the precision is 0.956 ,the recall is 0.409597257926
For top 1000 predictions, the precision is 0.864 ,the recall is 0.740359897172
Chenyus-MacBook-Pro:Homework2 chenyu$
```

Task 4 The precision against recall plot for $k \in [1, len(y\_test)]$ is shown as below:


1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.0
0.2
0.4
0.6
0.8
1.0

2 Dimensionality reduction

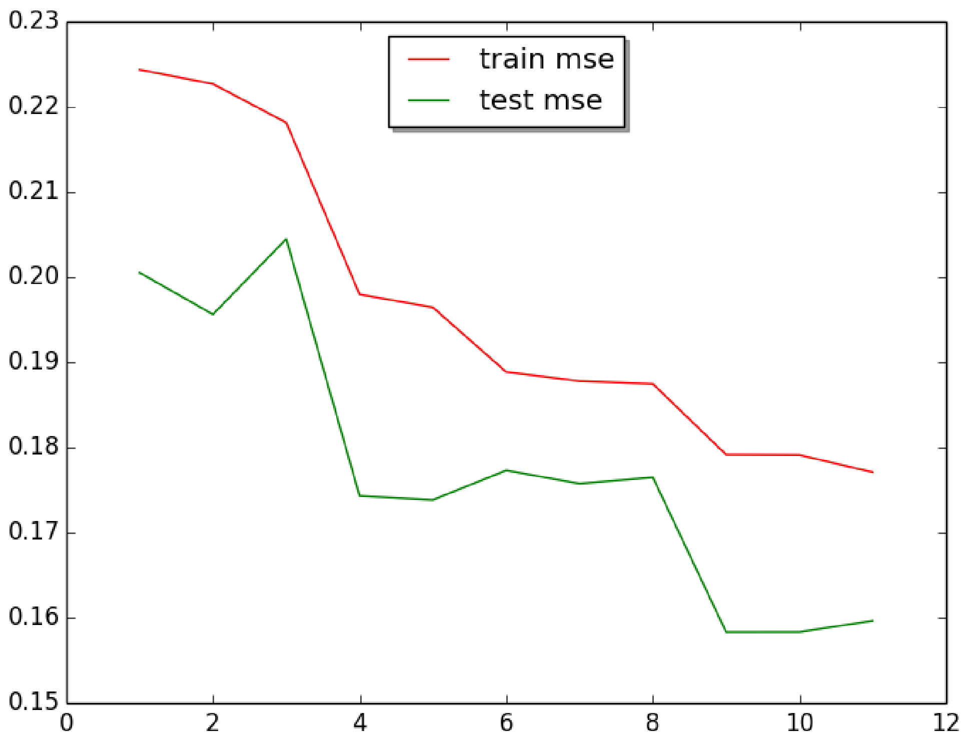
Task 5 The reconstruction error by replacing each point with their mean vector is: 3675818.61688. This is equivalent to having no principle component

Task 6 The transform matrix of all 11 principle components are:

```
Chenyus-MacBook-Pro:Homework2 chenyu$ python p6.py
Reading data...
done
[[ 0.00000000e+00  -3.23636346e-04   1.42201752e-04   3.17030713e-04
   5.36390435e-02   9.30284526e-05   2.54030965e-01   9.65655009e-01
   3.19990241e-05  -2.95831396e-04   3.84043646e-04  -1.00526693e-02]
 [ -0.00000000e+00  -7.57985623e-03  -1.66366340e-03   1.04742899e-03
   5.21677266e-02   4.49425600e-05   9.65020304e-01  -2.56793964e-01
   7.90089050e-06   5.24900596e-04  -1.09699394e-03  -2.89827657e-03]
 [ -0.00000000e+00   1.82124420e-02   2.54680710e-03   3.31838657e-03
   9.93221259e-01  -1.51888372e-04  -6.42297821e-02  -3.91682592e-02
   4.30929482e-04  -6.93199060e-03  -2.85216045e-03  -8.62920933e-02]
 [ -0.00000000e+00   1.56811999e-01   3.28220652e-03   1.66866136e-02
   8.28549640e-02  -6.91822288e-03   1.13029682e-03   5.39110108e-03
  -9.49080503e-04   2.68027305e-03   1.30498102e-03   9.83955205e-01]
 [  0.00000000e+00   9.81360642e-01  -1.45890108e-02   5.92643662e-02
  -3.17546064e-02   5.07483182e-04   8.43759364e-03  -1.77578042e-03
   6.03725221e-04  -9.05011239e-02  -9.35630845e-03  -1.54417839e-01]
 [ -0.00000000e+00   7.76578401e-02  -2.37665885e-01   2.23406619e-02
   5.04113878e-03  -1.43564098e-02  -2.14210997e-04  -2.22913844e-04
   3.36617054e-03   8.77254205e-01   4.08570175e-01  -1.54145486e-02]
 [  0.00000000e+00  -7.36289612e-02  -2.61563804e-01   9.43067566e-01
  -2.14514264e-03   1.19104298e-02  -1.68808905e-03   1.42294158e-04
  -1.17203197e-04  -1.45895558e-01   1.23868963e-01  -2.88797236e-03]
 [ -0.00000000e+00  -1.37617196e-02   2.11129619e-01  -1.16514121e-01
   5.30670319e-04   1.05181628e-02   1.36446528e-03  -8.21179429e-04
   3.09221855e-04  -3.58358431e-01   9.01728510e-01   3.27758247e-03]
 [  0.00000000e+00   1.74575775e-02   9.10890084e-01   3.04081497e-01
  -2.89763923e-03   2.34615054e-02   1.17406025e-03  -3.85957239e-04
   1.23176271e-03   2.68927937e-01  -6.70756658e-02  -1.12101920e-02]
 [  0.00000000e+00   2.31513441e-03  -2.38717789e-02  -1.67445603e-02
   8.92206499e-04   9.99462734e-01  -9.81109101e-05  -3.32812875e-05
   4.14235255e-03   1.18483756e-02  -3.51543098e-03   6.92344110e-03]
 [ -0.00000000e+00   7.48312160e-04   3.08204153e-04   2.55232500e-04
   3.49846801e-04   4.12943179e-03  -6.96565372e-06   4.16951216e-06
  -9.99984215e-01   3.17948604e-03   1.53436134e-03  -1.10029138e-03]]
```

Task 7 Using just 4 PCA dimensions, the reconstruction error is: 1345.4755741

Task 8 The changes on MSE for train and test sets against the number of dimensions used is shown below:


train mse
test mse
0.23
0.22
0.21
0.20
0.19
0.18
0.17
0.16
0.15
0
2
4
6
8
10
12

3 Code

Listing 1: task1

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

random.seed(0)

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

print "Reading data..."
dataFile = open("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]
random.shuffle(lines)
X = [l[:-1] for l in lines]
y = [l[-1] > 5 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 numpy.array([-x for x in dl])

X_train = X[:int(len(X)/3)]
y_train = y[:int(len(y)/3)]
X_validate = X[int(len(X)/3):int(2*len(X)/3)]
y_validate = y[int(len(y)/3):int(2*len(y)/3)]
X_test = X[int(2*len(X)/3):]
y_test = y[int(2*len(X)/3):]

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

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

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

def performance(theta):
  scores_train = [inner(theta,x) for x in X_train]
  scores_validate = [inner(theta,x) for x in X_validate]
  scores_test = [inner(theta,x) for x in X_test]

  predictions_train = [s > 0 for s in scores_train]
  predictions_validate = [s > 0 for s in scores_validate]
  predictions_test = [s > 0 for s in scores_test]

  correct_train = [(a==b) for (a,b) in zip(predictions_train,y_train)]
  correct_validate = [(a==b) for (a,b) in zip(predictions_validate,y_validate
      )]
  correct_test = [(a==b) for (a,b) in zip(predictions_test,y_test)]

  acc_train = sum(correct_train) * 1.0 / len(correct_train)
```

```
    acc_validate = sum(correct_validate) * 1.0 / len(correct_validate)
    acc_test = sum(correct_test) * 1.0 / len(correct_test)
    return acc_train, acc_validate, acc_test

##################################################
# Validation pipeline                            #
##################################################

for lam in [0, 0.01, 1.0, 100.0]:
    theta = train(lam)
    acc_train, acc_validate, acc_test = performance(theta)
    print("lambda = " + str(lam) + ";\ttrain=" + str(acc_train) + "; validate="
        + str(acc_validate) + "; test=" + str(acc_test))
```

Listing 2: task2

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

random.seed(0)

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

print "Reading data..."
dataFile = open("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]
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 numpy.array([-x for x in dl])

X_train = X[:int(len(X)/3)]
y_train = y[:int(len(y)/3)]
X_validate = X[int(len(X)/3):int(2*len(X)/3)]
y_validate = y[int(len(y)/3):int(2*len(y)/3)]
X_test = X[int(2*len(X)/3):]
y_test = y[int(2*len(X)/3):]

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

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

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

def performance(theta):
  scores_train = [inner(theta,x) for x in X_train]
  scores_validate = [inner(theta,x) for x in X_validate]
  scores_test = [inner(theta,x) for x in X_test]

  predictions_train = [s > 0 for s in scores_train]
  predictions_validate = [s > 0 for s in scores_validate]
  predictions_test = [s > 0 for s in scores_test]

  true_positive = [(a==b) and a == True for (a,b) in zip(predictions_test,
      y_test)]
  true_negative = [(a==b) and a == False for (a,b) in zip(predictions_test,
      y_test)]
  false_positive = [(a!=b) and a == True for (a,b) in zip(predictions_test,
      y_test)]
  false_negative = [(a!=b) and a == False for (a,b) in zip(predictions_test,
      y_test)]
  #compute balanced error rate
  ber = 1 - 0.5 * (1.0 * sum(true_positive) / (sum(true_positive) + sum(
```

```
      false_negative)) + 1.0 * sum(true_negative) / (sum(true_negative) + sum
      (false_positive)))

  return ber, true_positive, true_negative, false_positive, false_negative

##################################################
# Validation pipeline                            #
##################################################

lam = 0.01
theta = train(lam)
ber, true_positive, true_negative, false_positive, false_negative=
    performance(theta)
# print("lambda = " + str(lam) + ";\ttrain=" + str(acc_train) + "; validate="
     + str(acc_validate) + "; test=" + str(acc_test))
print sum(true_positive), sum(true_negative), sum(false_positive), sum(
    false_negative)
print ber
```

Listing 3: task3

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

random.seed(0)

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

print "Reading data..."
dataFile = open("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]
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
```

```python
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 numpy.array([-x for x in dl])

X_train = X[:int(len(X)/3)]
y_train = y[:int(len(y)/3)]
X_validate = X[int(len(X)/3):int(2*len(X)/3)]
y_validate = y[int(len(y)/3):int(2*len(y)/3)]
X_test = X[int(2*len(X)/3):]
y_test = y[int(2*len(X)/3):]

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

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

##################################################
# Precision and Recall                           #
##################################################
def precision_and_recall(theta, top):
  scores_test = [inner(theta,x) for x in X_test]
  scores_sort = sorted(scores_test, reverse = True)

  indices = sorted(range(len(scores_test)), key = lambda k: scores_test[k],
      reverse = True)
  y_sort = [y_test[indices[x]] for x in range(len(indices))]

  precision = 1.0 * sum(y_sort[0:top]) / top
  recall = 1.0 * sum(y_sort[0:top]) / sum(y_sort)

  return precision, recall

##################################################
# Validation pipeline                            #
##################################################


lam = 0.01
theta = train(lam)
```

```
for top in [10, 500, 1000]:
    precision, recall = precision_and_recall(theta, top)
    print("For top " + str(top) + " predictions, the precision is " + str(
        precision) + " ,the recall is " + str(recall))
```

Listing 4: task4

```
import numpy
import urllib
import scipy.optimize
import random
import matplotlib.pyplot as plt
from math import exp
from math import log



random.seed(0)

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

print "Reading data..."
dataFile = open("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]
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 numpy.array([-x for x in dl])

X_train = X[:int(len(X)/3)]
y_train = y[:int(len(y)/3)]
X_validate = X[int(len(X)/3):int(2*len(X)/3)]
y_validate = y[int(len(y)/3):int(2*len(y)/3)]
X_test = X[int(2*len(X)/3):]
y_test = y[int(2*len(X)/3):]

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

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

##################################################
# Precision and Recall                           #
##################################################
def precision_and_recall(theta, top):
  scores_test = [inner(theta,x) for x in X_test]
  scores_sort = sorted(scores_test, reverse = True)

  indices = sorted(range(len(scores_test)), key = lambda k: scores_test[k],
      reverse = True)
  y_sort = [y_test[indices[x]] for x in range(len(indices))]

  precision = 1.0 * sum(y_sort[0:top]) / top
  recall = 1.0 * sum(y_sort[0:top]) / sum(y_sort)

  return precision, recall

##################################################
# Validation pipeline                            #
##################################################
```

```
lam = 0.01
theta = train(lam)
precisions = []
recalls = []
for top in range(1, len(y_test)):
    precision, recall = precision_and_recall(theta, top)
    precisions.append(precision)
    recalls.append(recall)
    # print("For top " + str(top) + " predictions, the precision is " + str(
        precision) + " ,the recall is " + str(recall))

plt.plot(recalls, precisions)
plt.show()
```

Listing 5: task5

```
import numpy
import urllib
import scipy.optimize
import random
from math import exp
from math import log
from sklearn.decomposition import PCA
import copy


random.seed(0)

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

print "Reading data..."
dataFile = open("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]
print "done"

X_train = X[:int(len(X)/3)]
X_reduce = copy.deepcopy(X_train)

for i in range(0, len(X_train[0])):
    sum = 0
    for j in range(0, len(X_train)):
        sum = sum + X_train[j][i]
    for j in range(0, len(X_train)):
        X_reduce[j][i] = sum / len(X_train)


def recon_error(X_Orig, X_Compressed):
    error = 0
    for i in range(0, len(X_Orig)):
        for j in range(0, len(X_Orig[0])):
            error = error + (X_Compressed[i][j] - X_Orig[i][j]) * (
                X_Compressed[i][j] - X_Orig[i][j])
    return error

print recon_error(X_train, X_reduce)
```

Listing 6: task6

```
import numpy
import urllib
import scipy.optimize
import random
from sklearn.decomposition import PCA
from collections import defaultdict

### PCA on wine reviews ###

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

print "Reading data..."
dataFile = open("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]
print "done"

print len(y)

X_train = X[:int(len(X)/3)]

pca = PCA(n_components=12)
pca.fit(X_train)
print pca.components_
```

Listing 7: task7

```
import numpy
import urllib
import scipy.optimize
import random
from sklearn.decomposition import PCA
from collections import defaultdict

### PCA on wine reviews ###

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

print "Reading data..."
dataFile = open("winequality-white.csv")
header = dataFile.readline()
fields = ["constant"] + header.strip().replace('"','').split(';')
featureNames = fields[:-1]
labelName = fields[-1]
lines = [[] + [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]
print "done"

X_train = X[:int(len(X)/3)]

pca = PCA(n_components=4)
pca.fit(X_train)
X_new = pca.transform(X_train)

X_restored = pca.inverse_transform(X_new)
X_restored = X_restored.tolist()

def recon_error(X_Orig, X_Compressed):
    error = 0
    for i in range(0, len(X_Orig)):
        for j in range(0, len(X_Orig[0])):
            error = error + (X_Compressed[i][j] - X_Orig[i][j]) * (
                X_Compressed[i][j] - X_Orig[i][j])
    return error

print recon_error(X_train, X_restored)
```

Listing 8: task8

```
import numpy
import urllib
import scipy.optimize
import random
from sklearn.decomposition import PCA
from collections import defaultdict
import matplotlib.pyplot as plt


### PCA on wine reviews ###

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

print "Reading data..."
dataFile = open("winequality-white.csv")
header = dataFile.readline()
fields = ["constant"] + header.strip().replace('"','').split(';')
featureNames = fields[:-1]
labelName = fields[-1]
lines = [[] + [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]
print "done"

X_train = X[:int(len(X)/3)]
y_train = y[:int(len(y)/3)]
X_validate = X[int(len(X)/3):int(2*len(X)/3)]
y_validate = y[int(len(y)/3):int(2*len(y)/3)]
X_test = X[int(2*len(X)/3):]
y_test = y[int(2*len(X)/3):]

train_mse = []
test_mse = []
dimension = []

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

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

for i in range(1, 12):
```

```
    #determine pca
    pca = PCA(n_components=i)
    pca.fit(X_train)

    #apply demensionality reduction
    X_train_reduced = pca.transform(X_train)
    X_test_reduced = pca.transform(X_test)

    #compute theta for regressor
    X = [feature(d, i) for d in X_train_reduced]
    y = [d for d in y_train]
    X = numpy.matrix(X)
    y = numpy.matrix(y)
    thetas = numpy.linalg.inv(X.T * X) * X.T * y.T

    #compute mse on training data
    predicted = X * thetas
    predicted = predicted.T
    dimension.append(i)
    train_mse.append(mean_squared_error(predicted.A1, y.A1))

    #compute mse on test data
    X = [feature(d, i) for d in X_test_reduced]
    y = [d for d in y_test]
    X = numpy.matrix(X)
    y = numpy.matrix(y)
    predicted = X * thetas
    predicted = predicted.T
    test_mse.append(mean_squared_error(predicted.A1, y.A1))

fig, ax = plt.subplots()
ax.plot(dimension, train_mse, 'r', label='train mse')
ax.plot(dimension, test_mse, 'g', label='test mse')
legend = ax.legend(loc='upper center', shadow=True)
plt.show()
```