```
import os
import sys
import math
import numpy as np
import random
from numpy.linalg import inv
from random import shuffle
import matplotlib.pyplot as plt

alpha = 0.1

def read_file(filename):
    result = []
    with open(filename, 'r') as f:
        for line in f:
            line = line.strip('\n')
            line = line.split(',')
            result.append(line)
    result = np.array(result)
    result = result.astype(float)
    return result

def read_file_r(filename):
    result = []
    with open(filename, 'r') as f:
        for line in f:
            line = line.strip('\n')
            result.append(line)
    return result

##################################################################################################
# generative
##################################################################################################
# compute u1(class 1), u2(class 0)
def compute_mean(data_file, label_file, tr_index):
    N1 = 0
    N2 = 0
    sum1 = 0
    sum2 = 0
    for i in tr_index:
        if label_file[i] == '1':
            sum1 += data_file[i]
            N1 += 1
        else:
            sum2 += data_file[i]
            N2 += 1
    u1 = sum1 / float(N1)
    u2 = sum2 / float(N2)
    return u1, u2, N1, N2
```

```
# compute s1 (class 1), s2(class 0)
def compute_s(data, data_label, tr_index, u1, u2):
    s1 = 0
    s2 = 0
    for i in tr_index:
        if data_label[i] == '1':
            dif = data[i] - u1
            s1 += dif[:,None]*dif
        else:
            dif = data[i] - u2
            s2 += dif[:,None]*dif
    return s1, s2

# compute w0 and w
def compute_w_ge(s, u1, u2, N1, N2):
    s_inverse = inv(s)
    prod1 = np.dot(np.dot(u1, s_inverse),u1)
    prod2 = np.dot(np.dot(u2, s_inverse), u2)
    w0 = (-1) * prod1 / 2.0 + prod2 / 2.0 + math.log(float(N1)/N2)
    w = np.dot(s_inverse, u1 - u2)
    return w0, w

# compute error rate
def compute_error_ge(data, data_label, index_test, w, w0):
    error = 0
    for te_i in index_test:
        a = np.dot(w, data[te_i]) + w0
        if a >= 0:
            if data_label[te_i] != '1':
                    error += 1
        else:
            if data_label[te_i] != '0':
                error += 1
    return error / float(len(index_test))

##############################################################################
# discrimitive
###############################################################################
def compute_y (w0, phi):
    a = np.dot(w0, phi)
    return 1 / (1 + np.exp(-a))

# compute R
def compute_R(y):
    r = map(lambda x: x * (1 - x), y)
    return np.diagflat(r)

# compute w
def compute_w(phi, phi_tranpose, R, y, t, w0):
    global alpha
    N = len(phi[0])
    p1 = inv(np.dot(alpha, np.identity(N)) + np.dot(np.dot(phi_tranpose, R), phi))
    p2 = np.dot(phi_tranpose, (y-t)) + np.dot(alpha, w0)
    w1 = w0 - np.dot(p1, p2)
    return w1

def compute_w_sN(data, data_label, tr_index):
    global alpha
    phi = []
    t = []
    for i in tr_index:
        phi.append(data[i])
        t.append(data_label[i])

    t = np.array(t).astype(float)
    d = len(phi[0])
    N = len(phi)
    # add feature in the last row
    phi_p = np.ones((N, d+1))
    phi_p[:,:-1] = phi
    phi = phi_p
    w0 = [0] * (d + 1)

    phi_tranpose = np.transpose(phi)
    y = compute_y(w0, phi_tranpose)
    R = compute_R(y)

    # compute w1
    w1 = compute_w(phi, phi_tranpose, R, y, t, w0)
    #sum1 = sum(np.square(np.subtract(w1, w0))) / sum(np.square(w0))
    sum1 = 1
    n = 1
    while n < 100 and sum1 >= 10 ** (-3):
        w0 = w1
        y = compute_y(w0, phi_tranpose)
        R = compute_R(y)
        w1 = compute_w(phi, phi_tranpose, R, y, t, w0)
        sum1 = sum(np.square(np.subtract(w1, w0))) / sum(np.square(w0))
        n += 1

    y = compute_y(w1, phi_tranpose)
    R = compute_R(y)
    s = np.dot(np.dot(phi_tranpose, R), phi)
    s0 = inv(np.identity(d + 1) / alpha)
    sN = inv(s0 + s)
    return w1, sN

# compute error
def compute_error(data, data_label, index_test, w_map, sN):
    error = 0
    for te_i in index_test:
        temp = data[te_i].tolist()
        temp.append(1)
        d = np.array(temp)
        ua = np.dot(w_map, d)
        sig_s = np.dot(np.dot(d, sN), d)
        a = ua / math.sqrt(1 + np.pi * sig_s / 8)
        if a >= 0:
            if data_label[te_i] != '1':
                error += 1
        else:
            if data_label[te_i] != '0':
                error += 1
    return error / float(len(index_test))
################################################################################

def main():
    data_file = ['A.csv', 'B.csv', 'usps.csv']
    label_file = ['labels-A.csv', 'labels-B.csv', 'labels-usps.csv']

    file_number = len(data_file)
    # x, store length of train file for plot use
    x = []
    # three file error list (matrix)
    ge_error_file = []
    dis_error_file = []

    for i in range(file_number):
        # store single file error (list)
        ge_error_list = []
        dis_error_list = []

        data = read_file(data_file[i])
        data_label = read_file_r(label_file[i])
        N = len(data)
        x.append(N-int(N/3))

        index_N = range(N)
        # run 30 times
        for t in range(30):
            # set up 1/3 data set index
            te_N = int(N/3)
            index_test = np.random.choice(N, te_N, replace = False)

            # remain 2/3 train set
            index_train = [v for j, v in enumerate(index_N) if j not in index_test]
            tr_N = len(index_train)
            # Record performance
            splits = np.arange(0.05, 1.05, 0.05)
            ge_error_rate = []
            dis_error_rate = []

            for s in splits:
                # train set index
                tr_index = np.random.choice(index_train, int(s*tr_N), replace = False)

                tr_label = []



    ################################################################
                # generative

                # u1: mean of class 1, u2: mean of class 0
                u1, u2, N1, N2 = compute_mean(data, data_label, tr_index)
                s1, s2 = compute_s(data, data_label, tr_index, u1, u2)
                s1 = s1 / float(N1)
                s2 = s2 / float(N2)
                # compute S
                Sig = (N1 / float(N)) * s1 +  (N2 / float(N)) * s2
                #d = len(Sig)
                #Sig = Sig + 10**(-9)*np.identity(d)
                w0, w = compute_w_ge(Sig, u1, u2, N1, N2)
                # test file
                ge_error = compute_error_ge(data, data_label, index_test, w, w0)
                ge_error_rate.append(ge_error)

    #################################################################
                # discrimitive
                w_map, sN = compute_w_sN(data, data_label, tr_index)

                # test file
                error = compute_error(data, data_label, index_test, w_map, sN)
                dis_error_rate.append(error)

            ge_error_list.append(ge_error_rate)
            dis_error_list.append(dis_error_rate)
        ge_error_file.append(ge_error_list)
        dis_error_file.append(dis_error_list)


    ge_mean =[]
    ge_std = []
    dis_mean = []
    dis_std  = []
    for k in range(3):
        ge_mean.append(np.mean(ge_error_file[k], axis = 0))
        ge_std.append(np.std(ge_error_file[k], axis = 0))
        dis_mean.append(np.mean(dis_error_file[k], axis = 0))
        dis_std.append(np.std(dis_error_file[k], axis = 0))
    print "start to plot:"
    splits = np.arange(0.05, 1.05, 0.05)
    plt.figure(1)
    x1 = x[0] * splits
    plt.errorbar(x1, ge_mean[0], ge_std[0],  marker = '^')
    plt.errorbar(x1, dis_mean[0], dis_std[0],  marker = '*')
    plt.title('A data set')
    plt.xlabel('train size')
    plt.ylabel('error')
    plt.legend(['generative', 'discriminative'], loc = 0)
    plt.savefig("A.png")
    plt.clf()


    plt.figure(2)
    x2 = x[1] * splits
    plt.errorbar(x2, ge_mean[1], ge_std[1], marker = '^')
    plt.errorbar(x2, dis_mean[1], dis_std[1], marker = '*')
    plt.title('B data set')
    plt.xlabel('train size')
    plt.ylabel('error')
    plt.legend(['generative', 'discriminative'], loc = 0)
    plt.savefig("B.png")
    plt.clf()


    plt.figure(3)
    x3 = x[2] * splits
    plt.errorbar(x3, ge_mean[2], ge_std[2],  marker = '^')
    plt.errorbar(x3, dis_mean[2], dis_std[2],  marker = '*')
    plt.title('USPS data set')
    plt.xlabel('train size')
    plt.ylabel('error')
    plt.legend(['generative', 'discriminative'], loc = 0)
    plt.savefig("USPS.png")
    plt.clf()
main()
```

Task 2:

```
import os
import sys
import math
import numpy as np
import random
from numpy.linalg import inv
from random import shuffle
import datetime
import matplotlib.pyplot as plt
```

```
alpha = 0.1
eta = 10**(-3)

def read_file(filename):
    result = []
    with open(filename, 'r') as f:
        for line in f:
            line = line.strip('\n')
            line = line.split(',')
            result.append(line)
    result = np.array(result)
    result = result.astype(float)
    return result

def read_file_r(filename):
    result = []
    with open(filename, 'r') as f:
        for line in f:
            line = line.strip('\n')
            result.append(line)
    return result

# compute y
def compute_y (w0, phi):
    a = np.dot(w0, phi)
    return 1 / (1 + np.exp(-a))

# compute R
def compute_R(y):
    r = map(lambda x: x * (1 - x), y)
    return np.diagflat(r)

# compute w
def compute_w(tr, tr_r, w0):
    global alpha
    global eta
    #w0 = [0] * len(tr[0])
    t = np.array(tr_r).astype(float)
    y = compute_y(w0, np.transpose(tr))
    tr_transpose = np.transpose(tr)
    p1 = np.dot(tr_transpose, (y - t)) + np.dot(alpha, w0)
    w1 = w0 - np.dot(eta, p1)
    return w1

# compute sN
def compute_sN(w, tr):
    y = compute_y(w, np.transpose(tr))
    R = compute_R(y)
    l = len(y)
    s = 0
    global alpha
    for i in range(l):
        p1 = np.dot(R[i][i], tr[i])
        p2 = p1[:,None]*tr[i]
        s += p2
    s0 = inv(np.identity(len(tr[0])) / alpha)
    sN = inv(s0 + s)
    return sN

# compute error
def compute_error(w, tr, te_N, data, data_label):
    sN = compute_sN(w, tr)
    error = 0
    for i in range(te_N):
        ua = np.dot(w, data[i])
        sig_s = np.dot(np.dot(data[i], sN), data[i])
        a = ua / (math.sqrt(1 + np.pi * sig_s / 8))
        if a >= 0:
            if data_label[i] == '0':
                error += 1
        else:
            if data_label[i] == '1':
                error += 1
    return error /float(te_N)

def main():
    data_file = ['A.csv', 'usps.csv']
    label_file = ['labels-A.csv', 'labels-usps.csv']
    file_number = len(data_file)
    global eta
    global alpha
    file_time = []
    file_error =[]
    w_file = []
    for i in range(file_number):
        data = read_file(data_file[i])
        data_label = read_file_r(label_file[i])
        N = len(data)
        d = len(data[0])
        temp = np.ones((N, d+1))
        temp[:,:-1] = data
        data = temp
        te_N = int(N/3)
        tr = data[te_N:]
        tr_r = data_label[te_N:]
        error_list = []
        # run 3 times
        time_list = []
        w_list =[]
        for t in range(3):
            run_time = []
            w0 = [0] * len(tr[0])
            # clock time start
            a = datetime.datetime.now()
            w = compute_w(tr, tr_r, w0)
            b = datetime.datetime.now()
            run_time.append((b-a).total_seconds())
            if t == 0:
                w_list.append(w)
                error_rate = compute_error(w, tr, te_N, data, data_label)
                error_list.append(error_rate)
            sum1 = 1
            n = 1
            while n < 6000 and sum1 >= eta:
                w0 = w
                a = datetime.datetime.now()
                w = compute_w(tr, tr_r, w0)
                b = datetime.datetime.now()
                run_time.append(run_time[-1] + (b-a).total_seconds())
                if t == 0:
                    w_list.append(w)
                    error_rate = compute_error(w, tr, te_N, data, data_label)
                    error_list.append(error_rate)
                sum1 = sum(np.square(np.subtract(w, w0))) / sum(np.square(w0))
                n += 1
            time_list.append(run_time)
        w_file.append(w_list)
        time_list = np.mean(time_list, axis = 0)
        file_time.append(time_list)
        file_error.append(error_list)


    """
    print "Gradient:"
    for k in range(2):
        print data_file[k]
        print file_time[k]
        print file_error[k]
        #print w_file[k]

    a_newton_time = [ 0.01537367,  0.02855167,  0.04108567,  0.05452367,  0.068228  ]
    a_newton_error = [0.05855855855855856, 0.04804804804804805, 0.046546546546546545,
0.046546546546546545, 0.046546546546546545]
    plt.figure(1)
    x = file_time[0]
    y = file_error[0]
    plt.plot(x, y)
    plt.plot(a_newton_time, a_newton_error)
    plt.xlabel('time line')
    plt.ylabel('error rate')
```

```
	plt.title('gradient A data')
	plt.gcf().autofmt_xdate()
	plt.savefig('task2_A.png')
	plt.clf()


	plt.figure(2)
	usps_newton_time = [ 0.02759267,  0.05277067 , 0.078221 ,   0.10310333,  0.12794833 ,
0.15363867,
	                                                                        0.17856  ,   0.204489 ,   0.22854267]
	usps_newton_error = [0.04093567251461988, 0.042884990253411304, 0.03313840155945419,
0.037037037037037035, 0.03898635477582846, 0.037037037037037035, 0.03313840155945419,
0.03313840155945419, 0.03508771929824561]
	x = file_time[1]
	y = file_error[1]
	plt.plot(x, y)
	plt.plot(usps_newton_time, usps_newton_error)
	plt.xlabel('time line')
	plt.ylabel('error rate')
	plt.title('gradient USPS data')
	plt.gcf().autofmt_xdate()
	plt.savefig('task2_usps.png')
	plt.clf()
	"""
main()

# ###################
# Newton method
import os
import sys
import math
import numpy as np
import random
from numpy.linalg import inv
from collections import OrderedDict
from random import shuffle
import datetime
import matplotlib.pyplot as plt

alpha = 0.1

def read_file(filename):
	result = []
	with open(filename, 'r') as f:
		for line in f:
			line = line.strip('\n')
			line = line.split(',')
			result.append(line)
	result = np.array(result)
	result = result.astype(float)
```

```
    return result

def read_file_r(filename):
    result = []
    with open(filename, 'r') as f:
        for line in f:
            line = line.strip('\n')
            result.append(line)
    return result

# compute y
def compute_y (w0, phi):
    a = np.dot(w0, phi)
    return 1.0 / (1 + np.exp(-a))

# compute R
def compute_R(y):
    r = map(lambda x: x * (1 - x), y)
    return np.diagflat(r)

# compute w
def compute_w(tr, tr_r, w0):
    global alpha
    t = np.array(tr_r).astype(float)
    y = compute_y(w0, np.transpose(tr))
    R = compute_R(y)
    tr_transpose = np.transpose(tr)
    p1 = inv(np.dot(alpha, np.identity(len(tr[0]))) + np.dot(np.dot(tr_transpose, R), tr))
    p2 = np.dot(tr_transpose, (y-t)) + np.dot(alpha, w0)
    w1 = w0 - np.dot(p1, p2)

    return w1

# compute sN
def compute_sN(w, tr):
    y = compute_y(w, np.transpose(tr))
    R = compute_R(y)
    l = len(y)
    s = 0
    global alpha
    s = np.dot(np.dot(np.transpose(tr), R), tr)
    s0 = inv(np.identity(len(tr[0])) / alpha)
    sN = inv(s0 + s)
    return sN

# compute error
def compute_error(w, tr, te_N, data, data_label):
    # compute sN
    sN = compute_sN(w, tr)
    error = 0
    for i in range(te_N):
        ua = np.dot(w, data[i])
        sig_s = np.dot(np.dot(data[i], sN), data[i])
        a = ua / float(math.sqrt(1 + np.pi * sig_s / 8))
        if a >= 0:
            if data_label[i] == '0':
                error += 1
        else:
            if data_label[i] == '1':
                error += 1

    return error /float(te_N)


def main():
    data_file = ['A.csv', 'usps.csv']
    label_file = ['labels-A.csv', 'labels-usps.csv']

    file_number = len(data_file)
    global alpha
    file_time = []
    file_error =[]
    w_file = []
    for i in range(file_number):
        error_list = []
        data = read_file(data_file[i])
        N = len(data)
        d = len(data[0])
        print N
        print d
        temp = np.ones((N, d+1))
        temp[:,:-1] = data
        data = temp
        data_label = read_file_r(label_file[i])
        N = len(data)
        te_N = int(N/3)
        tr = data[te_N:]
        tr_r = data_label[te_N:]
        time_list = []
        w_list = []
        # run 3 times
        for t in range(3):
            run_time = []
            w0 = [0] * len(tr[0])
            # clock time start
            a = datetime.datetime.now()
            w = compute_w(tr, tr_r, w0)
            b = datetime.datetime.now()
            run_time.append((b-a).total_seconds())
            if t == 2:
                w_list.append(w)
                error_rate = compute_error(w, tr, te_N, data, data_label)
                error_list.append(error_rate)

            sum1 = 1
            n = 1
            while n <= 100 and sum1 >= 10 ** (-3):
                w0 = w
                a = datetime.datetime.now()
                w = compute_w(tr, tr_r, w0)
                b = datetime.datetime.now()
                run_time.append(run_time[-1] + (b-a).total_seconds())
                if t == 2:
                    w_list.append(w)
                    error_rate = compute_error(w, tr, te_N, data, data_label)
                    error_list.append(error_rate)
                sum1 = sum(np.square(np.subtract(w, w0))) / sum(np.square(w0))
                n += 1
            time_list.append(run_time)
        #print w_list[-1]
        w_file.append(w_list)
        time_list = np.mean(time_list, axis = 0)
        file_time.append(time_list)
        file_error.append(error_list)


    print "Newton"
    for k in range(2):
        print data_file[k]
        print file_time[k]
        print file_error[k]
        #print w_file[k]

    plt.figure(1)
    #x = [datetime.datetime.now() + datetime.timedelta(seconds = i) for i in file_time[0]]
    x = file_time[0]
    y = file_error[0]
    plt.plot(x, y)
    plt.xlabel('time line')
    plt.ylabel('error rate')
    plt.title('Newton A data')
    plt.gcf().autofmt_xdate()
    plt.savefig('newton_A.png')
    plt.clf()

    plt.figure(2)
    #x = [datetime.datetime.now() + datetime.timedelta(seconds = i) for i in file_time[1]]
    x = file_time[1]
    y = file_error[1]
```

```
        plt.plot(x, y)
        plt.xlabel('time line')
        plt.ylabel('error rate')
        plt.title('Newton USPS data')
        plt.gcf().autofmt_xdate()
        plt.savefig('newton_USPS.png')
        plt.clf()

main()
```