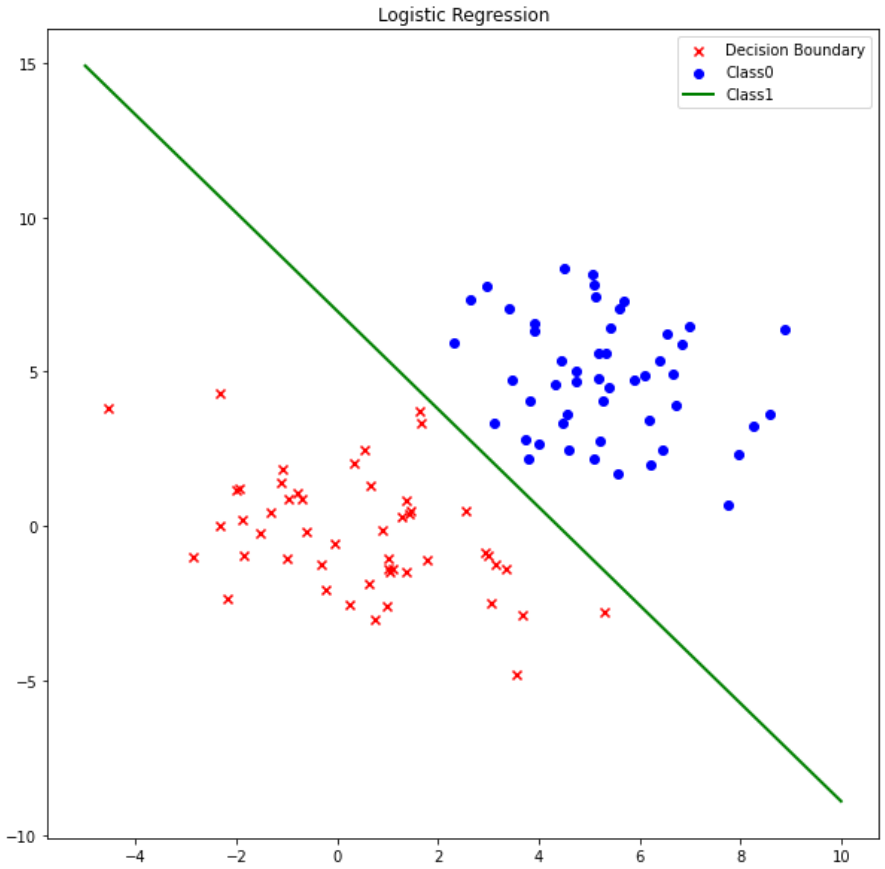
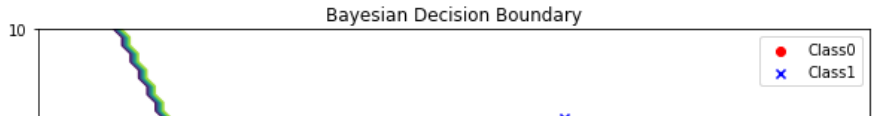
```
#HW4_2

import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cvx
from numpy.matlib import repmat

# (a) Logistic Regression loss


X0_org = np.matrix(np.loadtxt('homework4_class0.txt', converters={0: convt, 1: convt}))
X0 = np.hstack((X0_org,np.ones((X0_org.shape[0],1))))
Y0 = np.zeros((X0.shape[0],1))

X1_org = np.matrix(np.loadtxt('homework4_class1.txt', converters={0: convt, 1: convt}))
X1 = np.hstack((X1_org,np.ones((X1_org.shape[0],1))))
Y1 = np.ones((X1.shape[0],1))

x0_0 = np.zeros(50)
x0_1 = np.zeros(50)
x1_0 = np.zeros(50)
x1_1 = np.zeros(50)

for i in range(50):
    x0_0[i] = X0[i,0]
    x0_1[i] = X0[i,1]
    x1_0[i] = X1[i,0]
    x1_1[i] = X1[i,1]

x = np.vstack((X0, X1))
y = np.vstack((Y0,Y1))

#(b) Implementing regularization and minimizing loss function

lambd  = 0.0001
N = X0.shape[0] + X1.shape[0]

theta       = cvx.Variable((3,1))
first_term  = cvx.sum(cvx.multiply(y, x @ theta))
second_term = cvx.sum(cvx.log_sum_exp( cvx.hstack([np.zeros((N,1)), x @ theta]), axis=1 ) )
loss        = second_term - first_term
regul_term  = cvx.sum_squares(theta)
prob        = cvx.Problem(cvx.Minimize(loss/N + lambd*regul_term))
prob.solve()
w = theta.value

# (c) Scatter plot and decision boundary.

x_boundary  = np.linspace(-5,10,100)
y_boundary  = -(w[0]*x_boundary + w[2])/w[1]
plt.figure(figsize=(10,10))
```

```
plt.scatter(x0_0, x0_1, c='r', marker='x')
plt.scatter(x1_0, x1_1, c='b', marker='o')
plt.plot(x_boundary, y_boundary, 'g', linewidth=2.0)
plt.legend(['Decision Boundary','Class0', 'Class1'])
plt.title('Logistic Regression')
plt.show()
```




```
# (d) Bayesian

mu_0 = np.mean(X0_org, axis=0)
sigma_0 = np.cov(X0_org.T)
sigma0_inverse = np.linalg.inv(sigma_0)
dete0 = np.linalg.det(sigma_0)

mu_1 = np.mean(X1_org, axis=0)
sigma_1 = np.cov(X1_org.T)
sigma1_inverse = np.linalg.inv(sigma_1)
dete1 = np.linalg.det(sigma_1)
```

```
xb = np.linspace(-5,10,100)
yb = np.linspace(-5,10,100)
Bayes_pred = np.zeros((100, 100))

for i, x1 in enumerate(xb):
    for j, y1 in enumerate(yb):
        value = np.column_stack([x1, y1])
        value1 = value - mu_1
        value0 = value - mu_0

        LHS = -0.5*np.matmul(np.matmul(value1, sigma1_inverse), value1.T) -0.5*np.log(dete1)
        RHS = -0.5*np.matmul(np.matmul(value0, sigma0_inverse), value0.T) -0.5*np.log(dete0)

        if (LHS - RHS) > 0:
          Bayes_pred[i,j] = 1.0

plt.figure(figsize=(10,10))
plt.scatter(x0_0, x0_1, c='r', marker='o')
plt.scatter(x1_0, x1_1, c='b', marker='x')
plt.contour(xb,yb,Bayes_pred)
plt.legend(['Class0', 'Class1'])
plt.title('Bayesian Decision Boundary')
plt.show()
```




```
# HW4_4

# (a) Constructing Kernel

Kern = np.zeros((N, N))
h=1
for i in range(N):
    for j in range(N):
       Kern[i,j] = np.exp(-np.sum(np.square(x[i,:]-x[j,:]))/h)

print(Kern[47:52, 47:52])

lambda_2 = 0.0001

alpha = cvx.Variable((100,1))
loss_kernl = -cvx.sum(cvx.multiply(y, Kern@alpha))\
          +cvx.sum(cvx.log_sum_exp( cvx.hstack([np.zeros((N,1)), Kern@alpha]), axis=1 ) )
reg_kernl = cvx.quad_form(alpha, Kern)
prob_kernl = cvx.Problem(cvx.Minimize(loss_kernl/N + lambda_2*reg_kernl))
prob_kernl.solve()

reg_alpha = alpha.value
# (c) First two elements of the regression coefficients
print(reg_alpha[:2])

xs = np.linspace(-5,10,100)
ys = np.linspace(-5,10,100)
outp = np.zeros((100,100))
for i in range(100):
  for j in range(100):
    data = repmat( np.array([xs[i], ys[j], 1]).reshape((1,3)), N, 1)
    s = data - x
    ks = np.exp(-np.sum(np.square(s)/h, axis=1))
    outp[i,j] = np.dot(reg_alpha.T, ks).item()


 [[1.00000000e+00 5.05310080e-25 6.06536602e-20 4.65474122e-29
   4.06890793e-17]
  [5.05310080e-25 1.00000000e+00 3.95931666e-13 2.69357110e-33
   5.38775392e-12]
  [6.06536602e-20 3.95931666e-13 1.00000000e+00 2.30352619e-65
   3.78419625e-34]
  [4.65474122e-29 2.69357110e-33 2.30352619e-65 1.00000000e+00
   2.16278503e-06]
  [4.06890793e-17 5.38775392e-12 3.78419625e-34 2.16278503e-06
   1.00000000e+00]]
 [[-0.95245074]
  [-1.21046707]]
```

```
# (d) Scatter plot and decision boundary
plt.figure(figsize=(10,10))
plt.scatter(x0_0, x0_1, c='g', marker='o')
plt.scatter(x1_0, x1_1, c='r', marker='X')
plt.contour(xs, ys, outp>0.5, linewidths=1, colors='k')
plt.legend(['Class0', 'Class1'])
plt.title('Kernel Method')
plt.show()
```