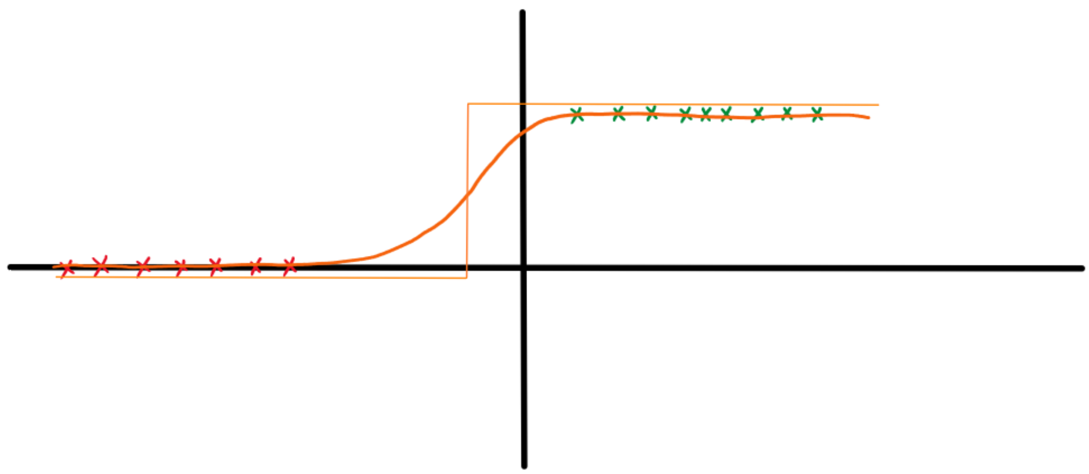
ECE 50024

Homework 3

Parth Sagar Hasabnis phasabni@purdue.edu

Exercise 1:



i. If the two classes of data are linearly separable, then with increasing iterations, the sigmoid function would tend to a step curve.
As $\theta \to \infty$, the angle of w would tend to π/2, and hence of slope of $||w||_2$ would tend closer to $\tan\left(\frac{\pi}{2}\right) = \infty$. Consequently, the transition of the sigmoid hyperplane would tend closer and closer to be parallel to the y-axis, resulting in the y-intercept w_0 to tend to ∞

ii. If we restrict $||w||_2 \leq c_1$ and $w_0 < c2$, then the iterations would stop after some finite number, causing the algorithm to converge. We can also counter non-convergence by:
   a. Stopping the algorithm by a finite number of iterations
   b. Applying a threshold on the maximum permissible error
   c. Adding a regularization term to penalize θ

iii. No, conversely, linear separability promotes convergence for linear-classifiers and convex problems, as in such cases, an analytical solution exists that can derive the optimal θ in a singular step.

Exercise 2:

Logistic regression loss is given by:

$$J(\theta) = \sum_{n=1}^{N} -\left\{ y_n \log h_{\bar{\theta}}(\bar{x}_n) + (1-y_n) \log(1 - h_{\bar{\theta}}(\bar{x}_n)) \right\}$$

$$h_{\bar{\theta}}(\bar{x}) = \frac{1}{1 + e^{-(\bar{w}^T \bar{x} + w_0)}} \qquad \bar{\theta} = \begin{bmatrix} \bar{w} \\ w_0 \end{bmatrix}$$

$$= \sum_{n=1}^{N} -\left\{ y_n \underbrace{\log \frac{h_{\bar{\theta}}(\bar{x}_n)}{1 - h_{\bar{\theta}}(\bar{x}_n)}} + \log(1 - h_{\bar{\theta}}(\bar{x}_n)) \right\}$$

$$= \log \frac{1/(1+\exp(-\theta^T \bar{x}_n))}{1 - 1/(1+\exp(-\theta^T \bar{x}_n))} = \log \frac{1}{1+\exp(-\theta^T \bar{x}_n) - 1} = \bar{\theta}^T \bar{x}_n$$

$$= \sum_{n=1}^{N} -\left\{ y_n \bar{\theta}^T \bar{x}_n + \log(1 - h_{\bar{\theta}}(\bar{x}_n)) \right\}$$

$$= -\sum_{n=1}^{N} (y_n \bar{x}_n)^T \theta - \sum_{n=1}^{N} \underbrace{\log\left(1 - \frac{1}{1+\exp(-\theta^T \bar{x}_n)}\right)}$$

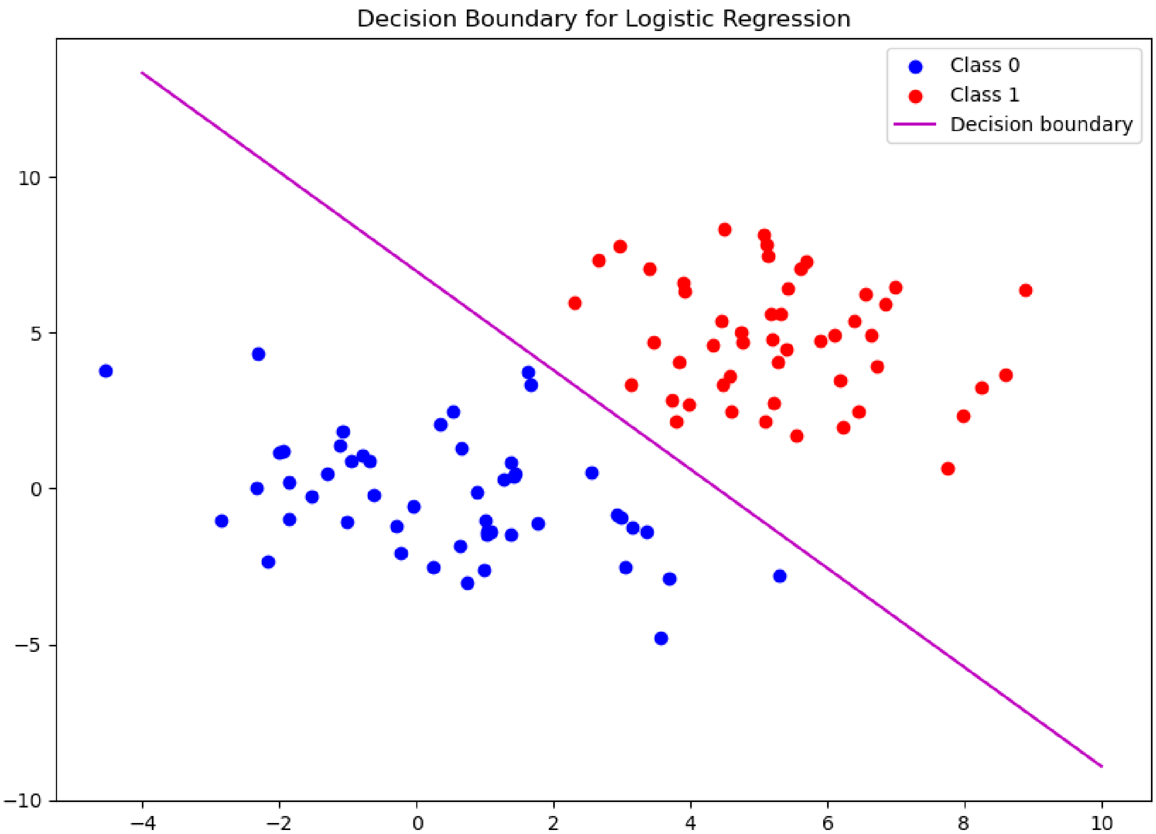
$$= \log \frac{1}{1+\exp(\bar{\theta}^T \bar{x}_n)}$$

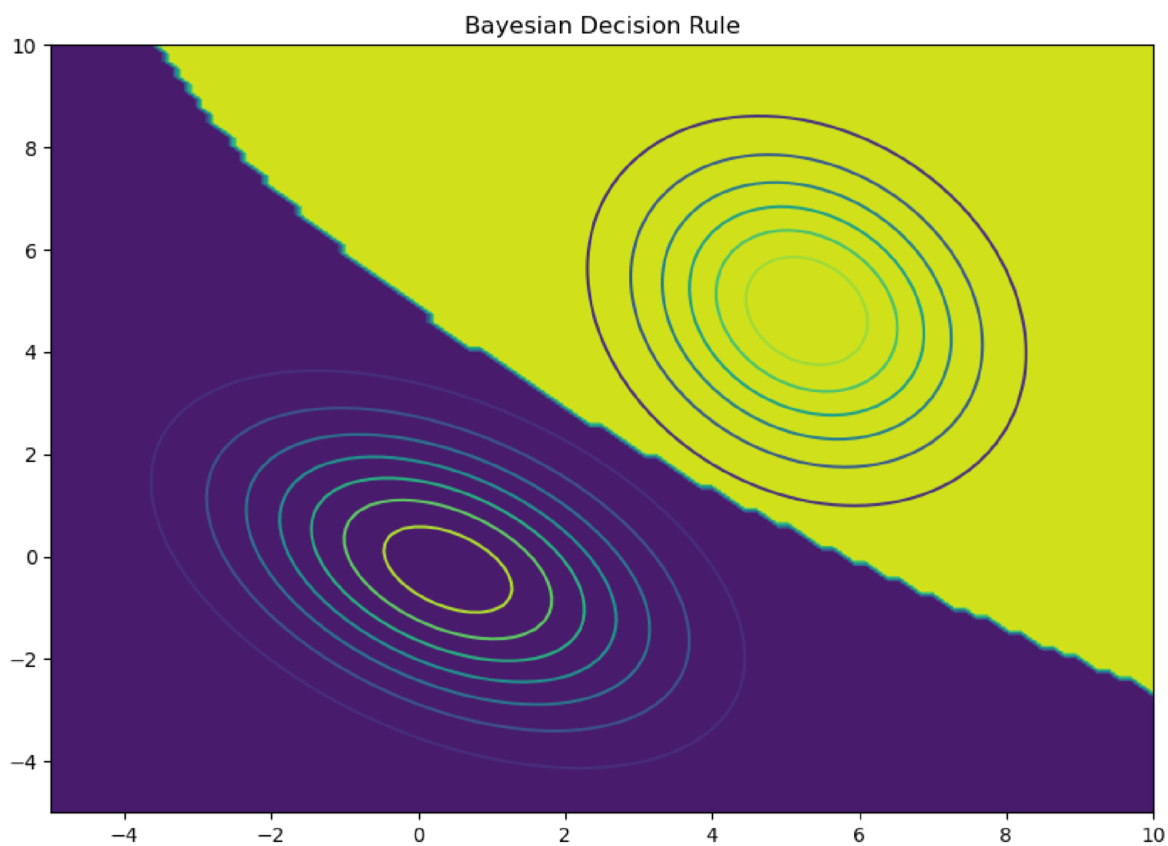
$$= -\log(1 + \exp(\bar{\theta}^T \bar{x}_n))$$

$$= -\left\{ \sum_{n=1}^{N} (y_n \bar{x}_n)^T \theta - \sum_{n=1}^{N} \log(1 + \exp(\bar{\theta}^T \bar{x}_n)) \right\}$$

Theta is given by:

$$\theta = \begin{matrix} [-10.44] \\ [2.38] \\ [1.50] \end{matrix}$$


Decision Boundary for Logistic Regression
Class 0
Class 1
Decision boundary


Bayesian Decision Rule

Exercise 3:

```
K[47:52, 47:52]
```

```
array([[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]])
```

Using kernel trick:

$$\theta^T x = \sum_{n=1}^{N} \alpha_n \langle x_n, x \rangle = \sum_{n=1}^{N} \alpha_n \cdot K(x_n, x)$$

$$J(\theta) = \frac{-1}{N} \left\{ \left(\sum_{n=1}^{N} y_n x_n \right)^T \theta - \sum_{n=1}^{N} \log\left(1 + \exp(\theta^T x_n)\right) \right\} + \lambda \|\theta\|^2$$

$$= \sum_{n=1}^{N} y_n \theta^T x_n = \sum_{n=1}^{N} y_n \sum_{n=1}^{N} \alpha_n \cdot K(x_n, x_n) = \sum_{n=1}^{N} y_n K\alpha = y^T K \alpha$$

$$= \sum_{n=1}^{N} \log\left(e^0 + e^{\theta^T x_n}\right) = \sum_{n=1}^{N} \log\left(e^0 + e^{\theta^T x_n}\right) = [1 \; 1 \; 1 \; 1 \; 1 \; 1] \log \begin{bmatrix} e^0 + e^{\theta^T x_1} \\ e^0 + e^{\theta^T x_2} \\ \vdots \\ e^0 + e^{\theta^T x_n} \end{bmatrix} = 1^T \log\left(e^0 + e^{K\alpha}\right)$$

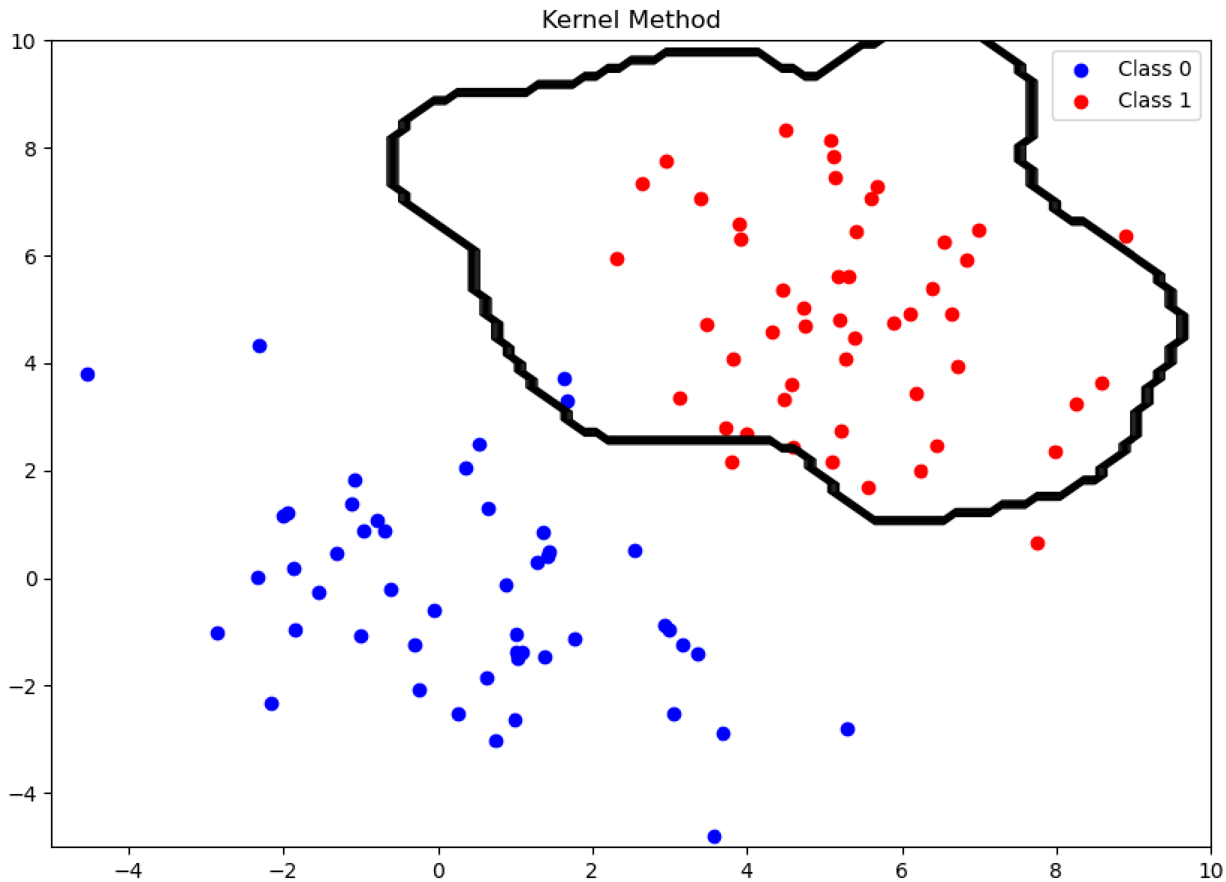
$$= \lambda \theta^T \theta = \lambda \sum_{n=1}^{N} \alpha_n \cdot K(x_n, \theta) = \lambda \sum_{n=1}^{N} \alpha_n \cdot K\left(x_n, \sum_{n=1}^{N} \alpha_n x_n\right) = \lambda \sum_{n=1}^{N} \alpha_n \cdot K(x_n, x_n)\, \alpha = \lambda \alpha^T K \alpha$$

J is expressed as a func^n of α

$$\therefore J(\alpha) = \frac{-1}{N} \left\{ y^T K \alpha - 1^T \log\left(e^0 + e^{K\alpha}\right) \right\} + \lambda \alpha^T K \alpha$$

Alpha is given by:

$$\alpha[0:2] = \begin{array}{c} -0.95 \\ -1.21 \end{array}$$



Kernel Method
Class 0
Class 1
10
8
6
4
2
0
−2
−4
−4
−2
0
2
4
6
8
10

APPENDIX

Conditional Generative Adversarial Networks

Parth Sagar Hasabnis, Master of Science in ECE[1]

Abstract

This document outlines the progress of the project - conditional GANs for the ECE 50024: Machine Learning Course.

Figure 1. Boot

Figure 2. T-shirt

1. Homework 2

Conditional Generative Adversarial Networks (cGANs) addresses a shortcoming of traditional GANS. Traditional GANs generate data by randomly sampling from a latent space and then using a generator network to transform that random input into a new data point. However, the generated data does not follow any given condition, and hence the output can be unpredictable. cGANs, on the other hand, enable the generation of data that is conditioned on a specific input or attribute.

Machine Learning and Deep Learning are the premier technologies in today's world, and both of them run on one single currency - data. Hence, the generation of data has become an important problem, as this data can be used to train autonomous vehicles, smart speakers and virtual assistants, wireless communications channels, and solve many more problems. Using cGANs, the generation of all kinds of data has become possible. We can generate conditioned data that was previously difficult or expensive to collect in the real world. For example – the performance and handling of an autonomous vehicle on a slippery road is an experiment that is difficult and dangerous to conduct in real life, but it is essential for autonomous vehicles to be trained on such conditions. Now we can generate artificial testing data using cGANs, and train vehicles using this artificial data.

There exist multiple implementations of cGANs on the internet, and different people have tried to implement them in their own methods. I shall start playing around with a PyTorch implementation trained on the MNIST dataset [1].

While I have created discriminator-like classifier networks, I am not familiar with the architecture of Generator networks. Hence my next step would be to learn about them and develop a traditional GAN before I move towards creating a cGAN.

2. Homework 3

I had a few issues with installing the Cuda version of PyTorch on my local device, and in the end decided to go with the CPU implementation. I was successful in running the implementation on my device. This implementation of CGANS uses the MNSIT fashion data set to learn the different types of clothing articles from 28x28 pixel images (shown above).

The CGAN was trained for 30 epochs and has resulted in effective generation of new data from the dataset.

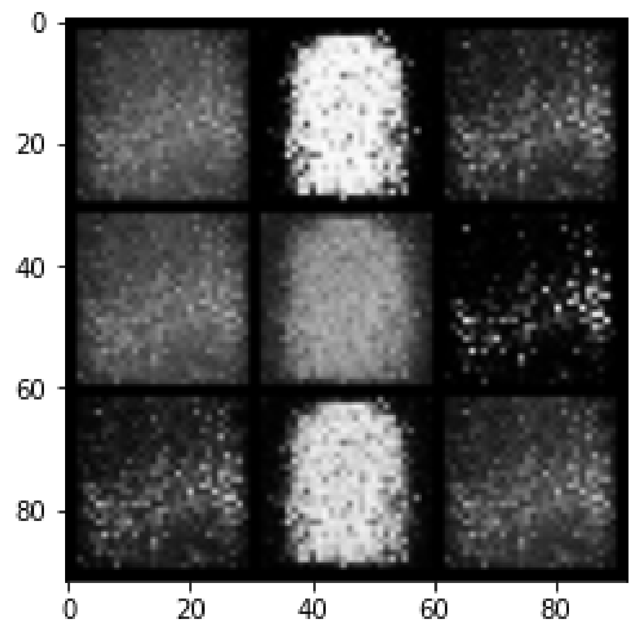


Figure 3. Epoch 1

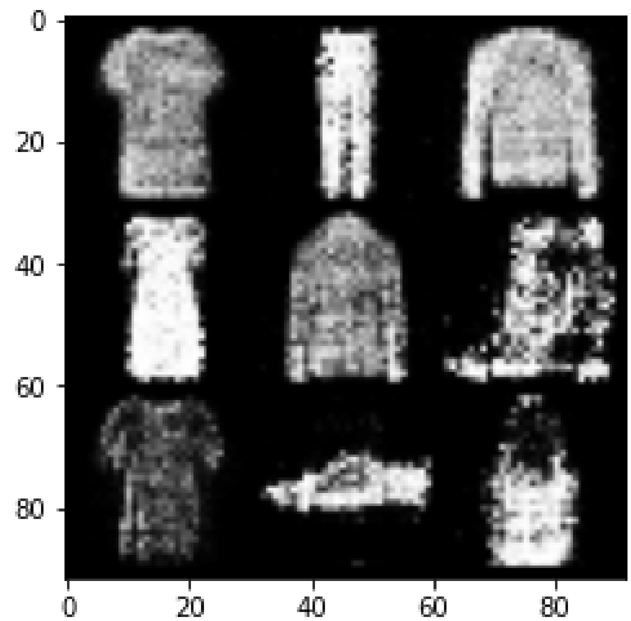


Figure 4. Epoch 30

3. Homework 4

3.1. Assmuptions

There are a few assumptions associated with my implementation of Conditional GANs. They are as follows:

- The FashionMNSIT dataset that I intend to use at the start has all the images in the same orientation, with no occlusions and uniform formatting.

- The dataset contains all images of the same size with no errors in data labelling.

- Each class of the dataset has an equal number of training data vectors so as to not bias one particular label over others.

3.2. Errors Occured

So far, all the errors I occurred during programming were a result of data type and shape mismatch. Matplotlib does not require a ”channel” index while displaying images, while pyTorch requires it. This has been a source of frustration while inputting the data into the neural network and trying to visualise its outputs. But now I have developed a pipeline to handle these data shape mismatches.

3.3. Flowchart

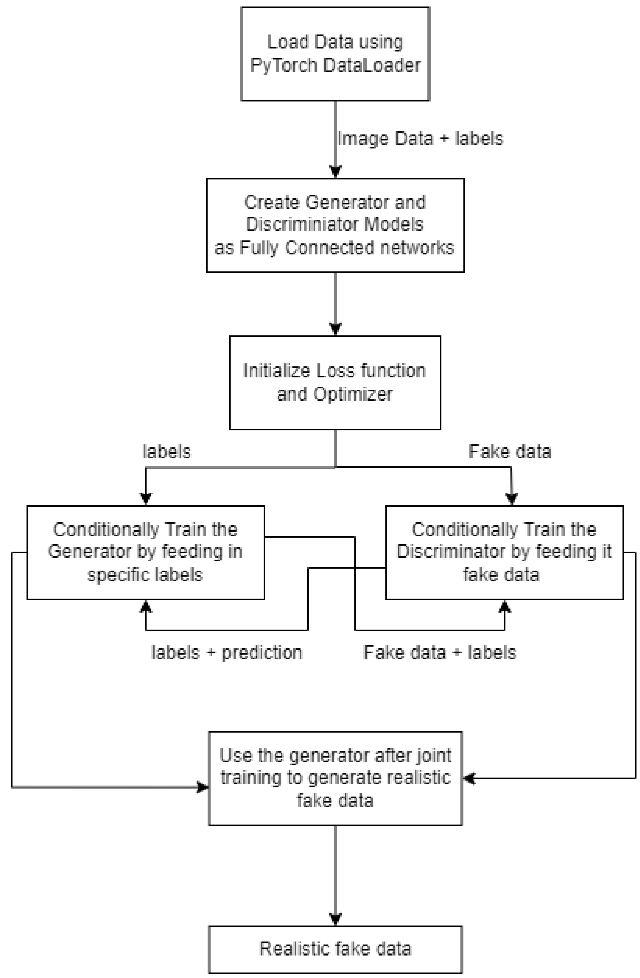


Figure 5. Algorithm Flowchart

4. Acknowledgements

[1] https://github.com/qbxlvnf11/conditional-GAN

Exercise 2

In [ ]:
```
import pandas as pd
import csv
import numpy as np
import cvxpy as cvx

class_0 = pd.read_csv("./data/homework4_class0.txt", delim_whitespace=True,
                  header=None, names=['Data_1','Data_2'])

class_1 = pd.read_csv("./data/homework4_class1.txt", delim_whitespace=True,
               header=None, names=['Data_1','Data_2'])

Data_3 = [1 for i in range(len(class_1))]
class_0['Data_3'] = Data_3
class_1['Data_3'] = Data_3

y_1 = [1 for i in range(len(class_0))]
y_0 = [0 for i in range(len(class_1))]
```

In [ ]:
```
X = pd.concat([class_0, class_1])
y = y_0 + y_1
N = len(y)
y = np.array(y).reshape(N,1)
```

In [ ]:
```
Data_1 = X["Data_1"].values.reshape(N,1)
Data_2 = X["Data_2"].values.reshape(N,1)
Data_3 = X["Data_3"].values.reshape(N,1)
```

In [ ]:
```
X = np.column_stack((Data_3, Data_1, Data_2))
theta = cvx.Variable((3,1))
lambd = 0.0001
#f1 = cvx.sum((y*X)@theta)
#f2 = cvx.sum(cvx.log_sum_exp(np.zeros(N).reshape(100,1), X@theta))

loss = (-cvx.sum(cvx.multiply(y, X @ theta)) + cvx.sum(cvx.log_sum_exp( cvx.hstack([np.z
prob = cvx.Problem(cvx.Minimize(loss))
prob.solve()
```

Out[ ]:
```
0.02354932900520089
```

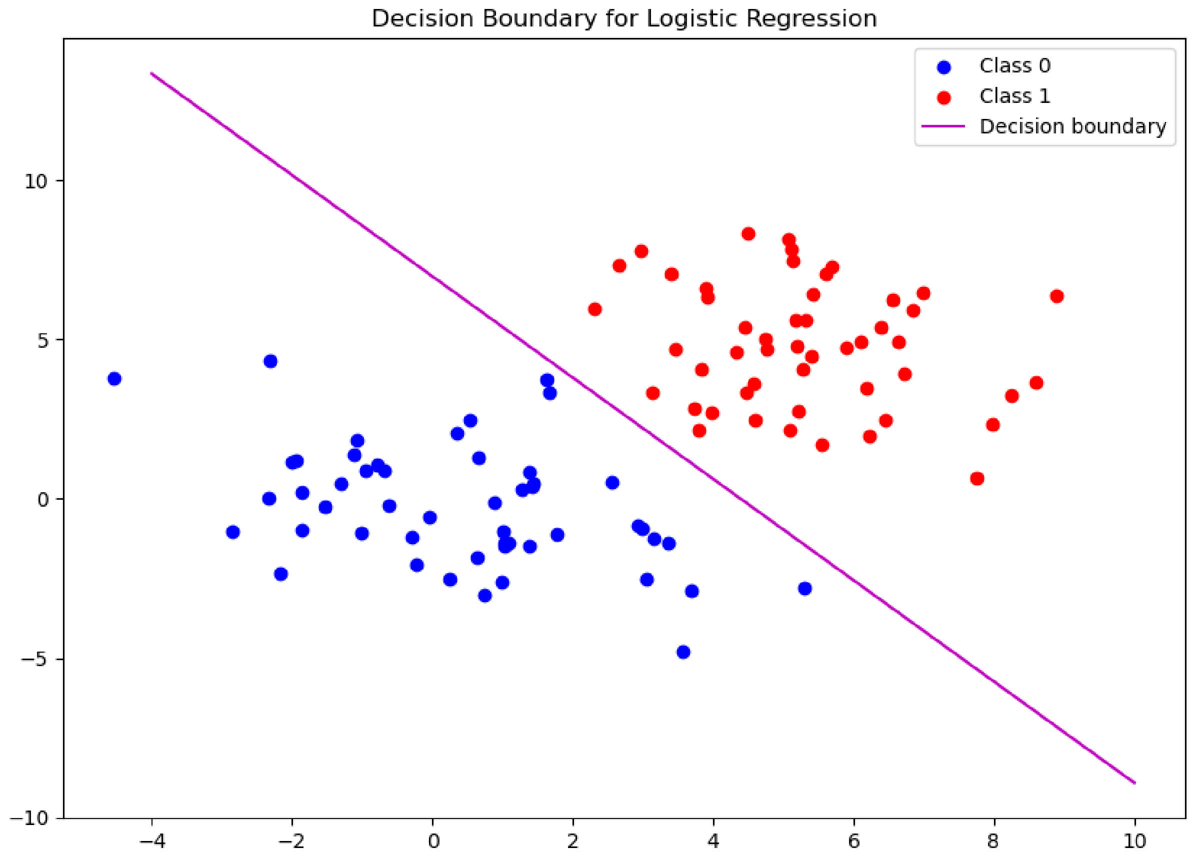
In [ ]:
```
print(theta)
```

```
var415
```

In [ ]:
```
import matplotlib.pyplot as plt

theta = theta.value
c = - theta[0]/theta[2]
m = - theta[1]/theta[2]
x1 = np.linspace(-4,10,100)
x2 = m*x1 + c

plt.figure(figsize = (10,7))
plt.scatter(class_0['Data_1'], class_0['Data_2'], c='b')
```

```
plt.scatter(class_1['Data_1'], class_1['Data_2'], c='r')
plt.plot(x1, x2, c='m')
plt.legend(["Class 0", "Class 1", "Decision boundary"])
plt.title("Decision Boundary for Logistic Regression")
```

Out[ ]: Text(0.5, 1.0, 'Decision Boundary for Logistic Regression')



In [ ]:
```
theta
```

Out[ ]:
```
array([[-10.43645136],
       [  2.37857414],
       [  1.49754409]])
```

Bayesian Decision Rule

In [ ]:
```
X = np.column_stack((Data_1, Data_2))
X_0 = X[0:50, :]
X_1 = X[50:, :]

K1 = class_1.shape[0]
K0 = class_0.shape[0]

pi_0 = K0/(K1+K0)
pi_1 = K1/(K1+K0)

mu_0 = np.mean(X_0, axis=0)
mu_1 = np.mean(X_1, axis=0)

sigma_0 = np.cov(X_0.T)
sigma_1 = np.cov(X_1.T)
```

In [ ]:

```
sig_0_inv = np.linalg.inv(sigma_0)
sig_1_inv = np.linalg.inv(sigma_1)
sig_0_det = np.linalg.det(sigma_0)
sig_1_det = np.linalg.det(sigma_1)
```

In [ ]:

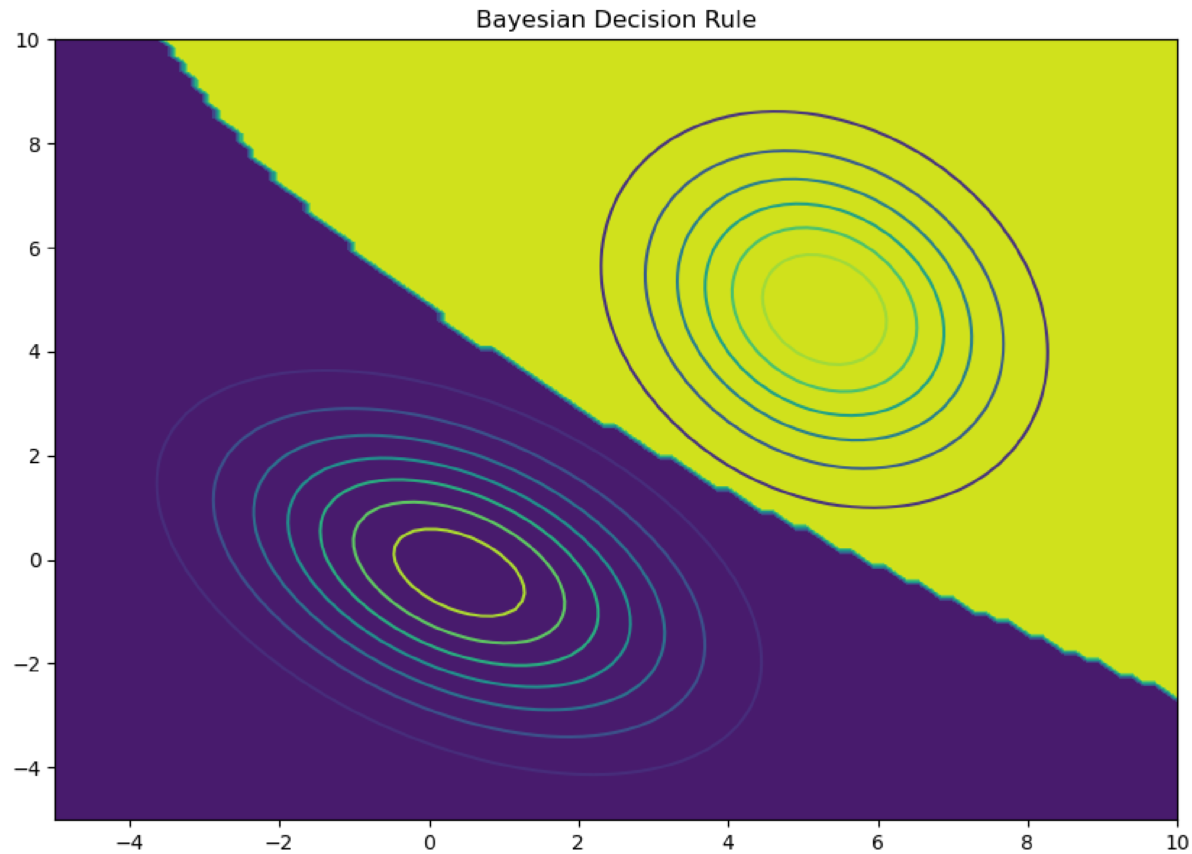
```
N_points = 101
points = np.linspace(-5, 10, N_points)
xx, yy = np.meshgrid(points, points, sparse=True)
xx = np.reshape(xx, max(xx.shape))
yy = np.reshape(yy, max(yy.shape))

mesh = np.zeros((N_points, N_points))

for i in range(N_points):
    for j in range(N_points):
        block = np.array([xx[i], yy[j]])
        LHS = -0.5*np.matmul(np.matmul((block - mu_1).T, sig_1_inv),(block - mu_1)) + np
        RHS = -0.5*np.matmul(np.matmul((block - mu_0).T, sig_0_inv),(block - mu_0)) + np
        mesh[i,j] = 1 if (LHS > RHS) else 0
```

In [ ]:

```
import scipy
gauss1 = scipy.stats.multivariate_normal(mu_1, sigma_1)
gauss0 = scipy.stats.multivariate_normal(mu_0, sigma_0)

XX, YY = np.meshgrid(points, points)
pos  = np.dstack((XX, YY))

Z0 = gauss0.pdf(pos)
Z1 = gauss1.pdf(pos)

fig = plt.figure(figsize=(10,7))
plt.contourf(XX, YY, mesh)
plt.contour(XX, YY, Z0)
plt.contour(XX, YY, Z1)
plt.title("Bayesian Decision Rule")
```

Out[ ]:

```
Text(0.5, 1.0, 'Bayesian Decision Rule')
```



Exercise 3

In [ ]:
```
X = np.column_stack((Data_1, Data_2, Data_3))

h = 1
K = np.zeros((X.shape[0], X.shape[0]))
for m in range(X.shape[0]):
    for n in range(X.shape[0]):
        K[m, n] = np.exp(-np.sum((X[m]-X[n])**2)/h)
```

In [ ]:
```
K[47:52, 47:52]
```

Out[ ]:
```
array([[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]])
```

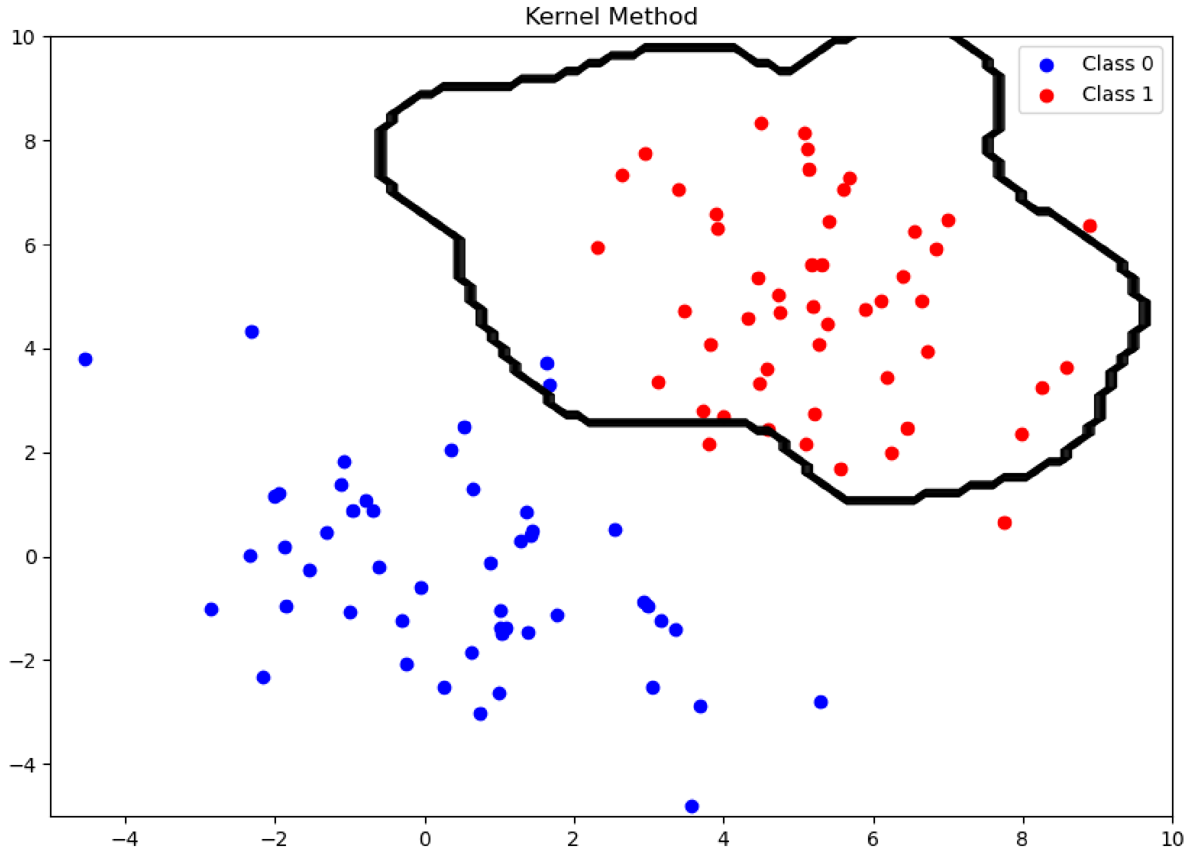
In [ ]:
```
alpha = cvx.Variable((N,1))
lambd = 0.0001

loss = -(cvx.sum(cvx.multiply(y, K @ alpha)) - cvx.sum(cvx.log_sum_exp( cvx.hstack([np.z
prob = cvx.Problem(cvx.Minimize(loss))
prob.solve()
```

Out[ ]: 0.0641699061939224

In [ ]:
```
print(alpha.value[:2])
```

```
[[-0.95245074]
 [-1.21046707]]
```

In [ ]:
```
from numpy.matlib import repmat
N_points = 101
points = np.linspace(-5, 10, N_points)
xx, yy = np.meshgrid(points, points, sparse=True)
xx = np.reshape(xx, max(xx.shape))
yy = np.reshape(yy, max(yy.shape))

mesh = np.zeros((N_points, N_points))
alpha = alpha.value

for i in range(N_points):
    for j in range(N_points):
        block = repmat( np.array([xx[i], yy[j], 1]).reshape((1,3)), N, 1)
        s = block - X
        ks = np.exp(-np.sum(np.square(s)/h, axis=1))
        mesh[i,j] = np.dot(alpha.T, ks).item()

plt.figure(figsize=(10,7))
plt.scatter(class_0['Data_1'], class_0['Data_2'], c='b')
plt.scatter(class_1['Data_1'], class_1['Data_2'], c='r')
plt.contour(xx, yy, mesh>0.5, linewidths=1, colors='k')
plt.legend(['Class 0', 'Class 1'])
plt.title('Kernel Method')
plt.show()
```



In [ ]: