# hw4

March 23, 2023

[ ]:
```
import math
from pathlib import Path
from IPython.display import display, Math, Image

import numba
import numpy as np
import matplotlib.pyplot as plt
import cvxpy as cvx

plt.rcParams['text.usetex'] = True

def data_directory() -> Path:
    return Path().cwd() / "data"
```

## 1 Exercise 1:

[ ]:
```
display(Image(filename="./images/hw4_p1i.png", height=400, width=500))
```

## Exercise 1i :

- If the two classes are linearly seperable then we know a solution exists. Let this solution be $\theta^*$.
- If $\theta^*$ is a solution, then we know that $c\theta^*$, $c \in \mathbb{R}$, $c>0$ is a solution as well.

Now, we inspect the loss function at $c\theta^*$:

$$\lim_{c\to\infty} -\sum_{\text{Class1}} \log\left(\frac{1}{1+\exp(-c\theta^{*T}x_n)}\right)$$

$$= -\sum_{\text{Class1}} \log(1) = 0$$

$$\lim_{c\to\infty} -\sum_{\text{Class 0}} \log\left(1 - \frac{1}{1+\exp(-c\underbrace{\theta^{*T}x_n}_{<0 \text{ for class } 0})}\right)$$

$$= -\sum_{\text{Class 0}} \log(1-0) = 0$$

Thus, we see that $J(\theta) \to 0$ as $c\theta^* \to \infty$

### 1.0.1 Problem 1ii:

If we restrict as given in the problem statement, then we will avoid the non-convergence issue. Another way to prevent non-convergence is to add a penalty term like we did with the ridge regression technique.

### 1.0.2 Problem 1iii:

The cross-entropy loss is the first time we have encountered non-convergence for linear classifiers.

# 2 Exercise 2:

[ ]:
```
display(Image(filename="./images/hw4_p2a.png", height=400, width=500))
```

Exercise 2a)

Starting with the original loss:

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

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

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

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

$$\log(1-h_{\vec{\theta}}(\vec{x}_n)) = \log\left(1-\frac{1}{1+\exp(-\vec{\theta}^T\vec{x})}\right)$$

$$= \log\frac{\exp(-\vec{\theta}^T\vec{x})}{1+\exp(-\vec{\theta}^T\vec{x})}\cdot\frac{e^{\vec{\theta}^T\vec{x}}}{e^{\vec{\theta}^T\vec{x}}} = -\log\frac{1}{1+e^{\vec{\theta}^T\vec{x}}}$$

$$= -\log(1+e^{\vec{\theta}^T\vec{x}})$$

Combining these two simplifications:

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

$$\boxed{= -\left\{ \underbrace{\left(\sum_{n=1}^{N} y_n x_n\right)^T\vec{\theta}} - \sum_{n=1}^{N}\log(1+e^{\vec{\theta}^T\vec{x}_n})\right\}}$$

We can swap the order
of T here b/c this is just
a scalar value.

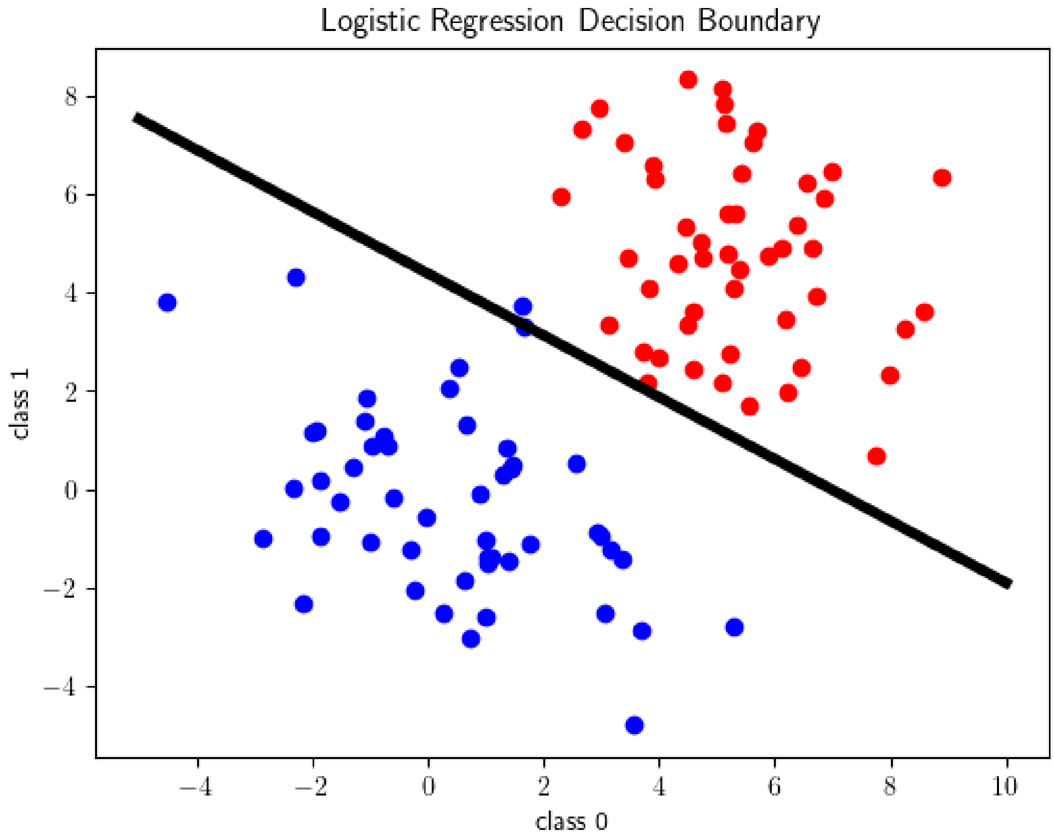
### 2.0.1 Problem 2b:

[ ]:
```
class0 = np.loadtxt(data_directory() / "homework4_class0.txt")
class1 = np.loadtxt(data_directory() / "homework4_class1.txt")
A = np.vstack([class0, class1])
X = np.hstack([A, np.ones((A.shape[0], 1))])
y = np.vstack([np.zeros([class0.shape[0], 1]), np.ones([class1.shape[0], 1])])
y = y.reshape((100, 1))
N = X.shape[0]
lambd = 0.0001

# print(f"Class0 Shape: {class0.shape}")
# print(f"Class1 Shape: {class1.shape}")
# print(f"X Shape: {X.shape}")
# print(f"y Shape: {y.shape}")

# Use CVXPY to minimize regularized logistic loss function
theta = cvx.Variable((3, 1))
loss = -cvx.sum(cvx.multiply(y, X @ theta)) \
    + cvx.sum(cvx.log_sum_exp(cvx.hstack([np.zeros((N, 1)), X @ theta]),␣
↪axis=1))
reg = cvx.sum_squares(theta)
prob = cvx.Problem(cvx.Minimize(loss / N + lambd * reg))

prob.solve()
w = theta.value
display(Math(r"\bf{\hat{\vec{\theta}}}:"))
print(w)

fig = plt.figure()
# Scatter plot the data
plt.scatter(class0[:, 0], class0[:, 1], color="b")
plt.scatter(class1[:, 0], class1[:, 1], color="r")
# Plot the decision boundary
x_axis = np.linspace(-5, 10, 100)
g_w = -w[2] / w[0] - (w[1] / w[0]) * x_axis
plt.plot(x_axis, g_w, "k", linewidth=4)
plt.title("Logistic Regression Decision Boundary")
plt.xlabel("class 0")
plt.ylabel("class 1")
```

^ :

```
[[  2.37857446]
 [  1.49754408]
 [-10.43645355]]
```

[ ]:
```
Text(0, 0.5, 'class 1')
```




### 2.0.2 Problem 2d:

```
[ ]: # Estimate means
mu0 = class0.mean(axis=0).reshape((2, 1))
mu1 = class1.mean(axis=0).reshape((2, 1))
display(Math(rf"\mu_0: {mu0}"))
display(Math(rf"\mu_1: {mu1}"))

# Estimate variances
sigma0 = np.cov(class0.T, bias=False)
sigma1 = np.cov(class1.T, bias=False)
display(Math(rf"\Sigma_0:"))
print(sigma0)
display(Math(rf"\Sigma_1:"))
print(sigma1)

# Estimate priors
N0 = class0.shape[0]
N1 = class1.shape[0]
# PI1 and PI2 are the same!
```

```
pi0 = N0 / (N0 + N1)
pi1 = N1 / (N0 + N1)
display(Math(rf"\pi_0: {pi0}"))
display(Math(rf"\pi_1: {pi1}"))

# Calculate sigma inverses for decision rule
sigma_0_inv = np.linalg.inv(sigma0)
sigma_1_inv = np.linalg.inv(sigma1)
sigma_0_det = np.linalg.det(sigma0)
sigma_1_det = np.linalg.det(sigma1)

@numba.jit
def make_decision(x0, x1) -> int:
    '''Returns 0/1 depending on classification'''
    x = np.array([x0, x1]).reshape((2, 1))
    xmmu0 = (x - mu0)
    xmmu1 = (x - mu1)
    c0 = -1/2 * xmmu0.T @ sigma_0_inv @ xmmu0 + math.log(pi0) - 1/2 * math.
 ↪log(sigma_0_det)
    c1 = -1/2 * xmmu1.T @ sigma_1_inv @ xmmu1 + math.log(pi1) - 1/2 * math.
 ↪log(sigma_1_det)
    return 1 if c1 > c0 else 0

x_axis = np.linspace(-5, 10, 100)
y_axis = np.linspace(-5, 10, 100)
XA, YA = np.meshgrid(x_axis, y_axis)
vfunc = np.vectorize(make_decision)
Z = vfunc(XA, YA)

fig = plt.figure()
plt.scatter(class0[:, 0], class0[:, 1], color="b")
plt.scatter(class1[:, 0], class1[:, 1], color="r")
plt.contour(XA, YA, Z > 0.5)
plt.title("Contour Plot of Bayesian Decision Rule")
plt.xlabel("class 0")
plt.ylabel("class 1")
```

$\mu_0 : [[0.403986][-0.249424]]$

$\mu_1 : [[5.284972][4.805084]]$

$\Sigma_0 :$

```
[[ 3.99504907 -1.7000222 ]
 [-1.7000222   3.69066941]]
```

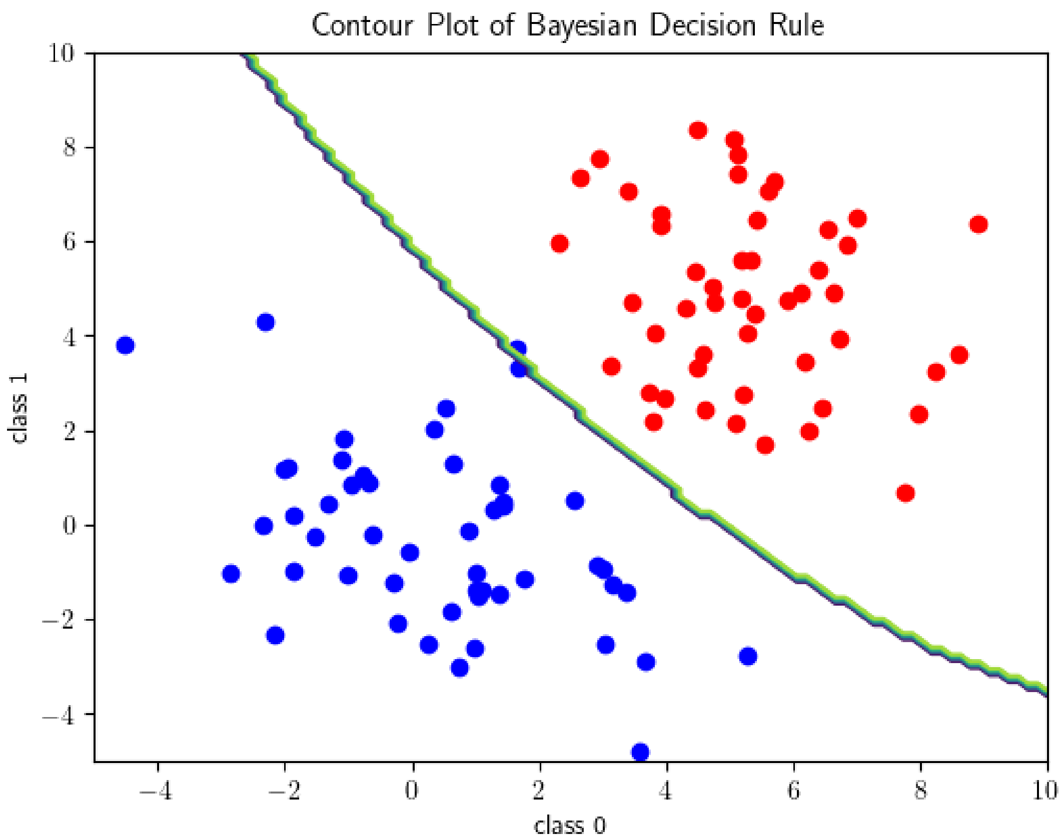
$\Sigma_1 :$

```
[[ 2.2889408  -0.63728394]
 [-0.63728394  3.73282879]]
```

$\pi_0 : 0.5$

$\pi_1 : 0.5$

```
[ ]: Text(0, 0.5, 'class 1')
```




# 3 Exercise 3:

### 3.0.1 Problem 3a:

```
[ ]: # kernel function
h = 1
@numba.jit
def kernel(x1, x2):
  return np.exp(-np.sum((x1 - x2)**2) / h)

# Construct kernel matrix K
K  = np.zeros((N,N))
for i in range(N):
  for j in range(N):
    K[i,j] = kernel(X[i], X[j])
```

```
display(Math(r"\bf{K[47:52, 47:52]}:"))
print(K[47:52, 47:52])
```

$\mathbf{K[47:52, 47:52]:}$

```
[[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]]
```

[ ]:

```
display(Image(filename="./images/hw4_p3b.png", height=400, width=500))
```

Exercise 3b:

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

$$+ \lambda \|\vec{\theta}\|^2$$

$$\left(\sum_{n=1}^{N} y_n \vec{x}_n\right)^T \left(\sum_{n=1}^{N} \alpha_n \vec{x}_n\right)$$

$$y_i x_i^T \alpha_j x_j = y_i x_i^T x_j \alpha_j$$

$$= \vec{y}^T K \vec{\alpha} \quad (1)$$

$$\|\vec{\theta}\|^2 = \left(\sum_{n=1}^{N} \alpha_n \vec{x}_n\right)^T \left(\sum_{n=1}^{N} \alpha_n \vec{x}_n\right)$$

$$\alpha_i x_i^T \alpha_j x_j$$

$$= \vec{\alpha}^T K \vec{\alpha} \quad (2)$$

for $\sum \log(\ldots)$, if we let log expression be vectorized, we can get:

$$\vec{\mathbb{1}}^T \log\left(\vec{1} + e^{K\vec{\alpha}}\right) \quad (3)$$

$$\left(\sum_{n=1}^{N} \alpha_n \vec{x}_n\right)^T \vec{x}_n = \vec{\alpha}^T K \quad \text{b/c } K \text{ is symmetric}$$

$$= K\vec{\alpha}$$

Combining 1, 2, + 3:

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

### 3.0.2 Problem 3c:

[ ]:
```
# Use CVXPY to minimize regularized logistic loss function + kernel trick
alpha_cvx = cvx.Variable((N, 1))
#  cvx.sum(cvx.multiply(y, K @ alpha_cvx))
loss = y.T @ K @ alpha_cvx \
    - cvx.sum(cvx.log_sum_exp(cvx.hstack([np.zeros((N, 1)), K @ alpha_cvx]),␣
 ↪axis=1))
reg = cvx.quad_form(alpha_cvx, K)
prob = cvx.Problem(cvx.Minimize((-1/N) * loss + lambd * reg))

prob.solve()
alpha = alpha_cvx.value
display(Math(r"\vec{\bf{\alpha}}:"))
print(alpha[0:2])
```

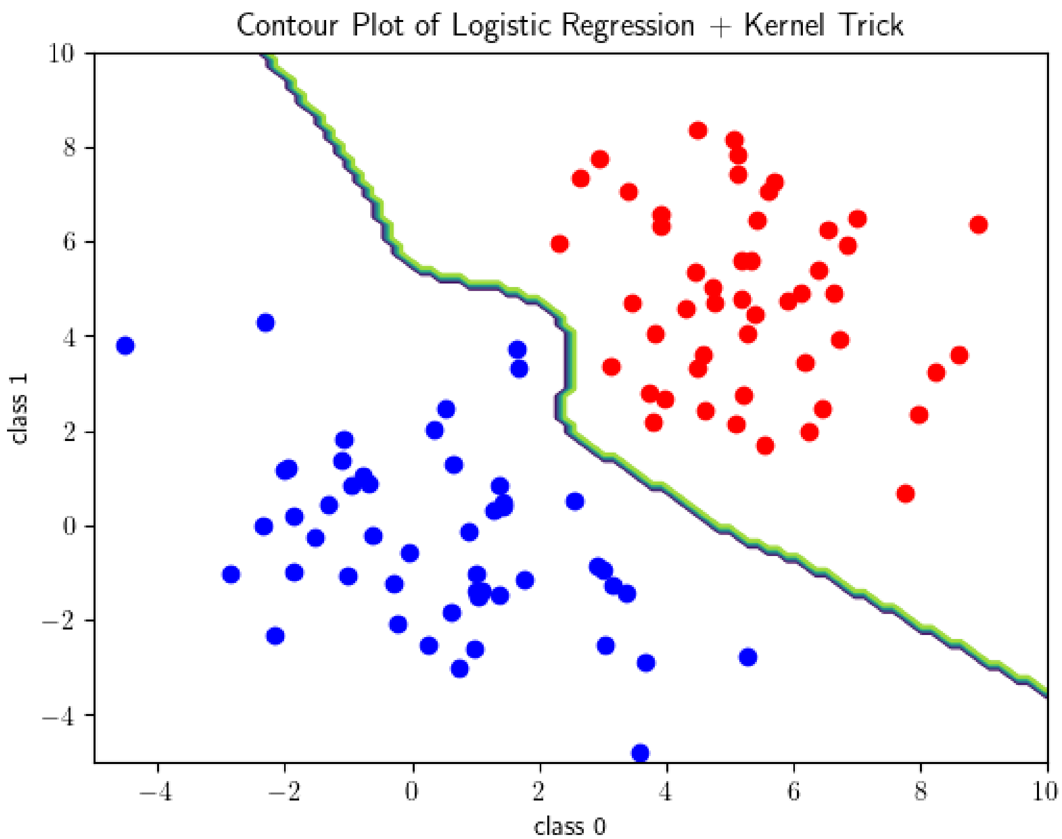
$\vec{\bf{\alpha}}$:

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

### 3.0.3 Problem 3d:

[ ]:
```
@numba.jit
def make_decision_kernel(x0, x1) -> int:
    '''Returns 0/1 depending on classification'''
    x = np.array([x0, x1, 1]).reshape((3, 1))
    sum = 0.0
    for i in range(N):
        data = X[i].reshape((3, 1))
        sum += alpha[i][0] * kernel(data, x)
    return 1 / (1 + np.exp(-sum))

xset = np.linspace(-5, 10, 100)
yset = np.linspace(-5, 10, 100)
XA, YA = np.meshgrid(xset, yset)
vfunc = np.vectorize(make_decision_kernel)
Z = vfunc(XA, YA)

fig = plt.figure()
plt.scatter(class0[:, 0], class0[:, 1], color="b")
plt.scatter(class1[:, 0], class1[:, 1], color="r")
plt.contour(XA, YA, Z > 0.5)
plt.title("Contour Plot of Logistic Regression + Kernel Trick")
plt.xlabel("class 0")
plt.ylabel("class 1")
```

[ ]:
```
Text(0, 0.5, 'class 1')
```


Contour Plot of Logistic Regression + Kernel Trick
class 1
class 0
10
8
6
4
2
0
−2
−4
−4
−2
0
2
4
6
8
10