ECE 50024

Homework 3

Parth Sagar Hasabnis phasabni@purdue.edu

Exercise 1:

a) $x \in \mathbb{R}^{d\times 1} \quad \therefore x^T \in \mathbb{R}^{1\times d}$

$A \in \mathbb{R}^{d\times d}$

$$\underset{1\times d}{x^T}\ \underset{d\times d}{A}\ \underset{d\times 1}{x} \in \mathbb{R}^{1\times 1}$$

$\therefore x^T A x$ is a scalar

$tr[x^T A x] = x^T A x$, Trace of a scalar is itself

$tr[x^T A x] = tr[A x x^T]$, $tr[AB] = tr[BA]$

$\therefore x^T A x = tr[A x x^T]$

b) The likelihood func^ is given by

$$p(\mathcal{D}|\Sigma) = \prod_{i=1}^{N} \left(\frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left\{ \frac{-1}{2} (x_i-\mu)^T \Sigma^{-1} (x_i-\mu) \right\} \right)$$

$$= \frac{1}{(2\pi)^{Nd/2}|\Sigma|^{-N/2}} \exp\left\{ \sum_{i=1}^{n} \frac{-1}{2} (x_i-\mu)^T \Sigma^{-1} (x_i-\mu) \right\}$$

$$= \frac{1}{(2\pi)^{Nd/2}} |\Sigma^{-1}|^{N/2} \exp\left\{ \frac{-1}{2} \sum_{i=1}^{n} tr\left[\Sigma^{-1} (x_i-\mu)(x_i-\mu)^T \right] \right\} \qquad x^T A x = tr[A x x^T]$$

$$= \frac{1}{(2\pi)^{Nd/2}} |\Sigma^{-1}|^{N/2} \exp\left\{ \frac{-1}{2} tr\left[\Sigma^{-1} \sum_{i=1}^{n} (x_i-\mu)(x_i-\mu)^T \right] \right\} \qquad tr(A+B) = tr(A) + tr(B)$$

c) $p(D|\Sigma) = \frac{1}{(2\pi)^{Nd/2}} \times \frac{1}{|\Sigma|^{N/2}} \exp\left\{ \frac{-1}{2} tr\left[\Sigma^{-1} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T \right] \right\}$

$= \frac{1}{(2\pi)^{Nd/2}} \times \frac{1}{|\Sigma|^{N/2}} \exp\left\{ \frac{-1}{2} tr\left[\Sigma^{-1} N \tilde{\Sigma} \right] \right\}$ ... $\tilde{\Sigma} = \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu)(x_n - \mu)^T$

$= \frac{1}{(2\pi)^{Nd/2}} \times \frac{1}{|\Sigma|^{N/2}} \exp\left\{ -\frac{N}{2} tr[A] \right\}$ $\quad tr(NA) = N\, tr(A), \quad A = \Sigma^{-1}\tilde{\Sigma}$

$A = \Sigma^{-1}\tilde{\Sigma}$

$\Sigma A = \tilde{\Sigma}$

$\Sigma = \tilde{\Sigma} A^{-1}$

$|\Sigma| = |\tilde{\Sigma}|\,|A^{-1}|$

$|A^{-1}| = |A|^{-1}$

$|A| = \prod_{i=1}^{d} \lambda_i$ $\quad \lambda_1, \ldots, \lambda_d$ are the eigenvalues of A

$\therefore p(D|\Sigma) = \frac{1}{(2\pi)^{Nd/2}} \frac{1}{|\tilde{\Sigma}|^{N/2}} \left(\prod_{i=1}^{d} \lambda_i \right)^{N/2} \exp\left\{ -\frac{N}{2} \sum_{i=1}^{d} \lambda_i \right\}$ $\quad tr[A] = \sum_{i=1}^{d} \lambda_i$

d) $\lambda_i = \underset{\lambda_i}{\operatorname{argmax}}\ p(D|\Sigma)$

$= \underset{\lambda_i}{\operatorname{argmin}}\ -\log p(D|\Sigma)$

$-\log p(D|\Sigma) = \frac{Nd}{2} \log(2\pi) + \frac{N}{2} \log|\tilde{\Sigma}| - \frac{N}{2} \sum_{i=1}^{d} \log \lambda_i + \frac{N}{2} \sum_{i=1}^{d} \lambda_i$

$\frac{d(-\log p(D|\Sigma))}{d\lambda_i} = -\frac{N}{2} \times \frac{1}{\lambda_i} + \frac{N}{2} = 0$

$\frac{N}{2} = \frac{N}{2\lambda_i}$

$\therefore \lambda_i = 1$

$\therefore \lambda = [\lambda_1, \lambda_2, \ldots, \lambda_d]^T_{1\times d} = [1 \quad 1 \quad 1 \ldots . \quad 1]^T_{1\times d}$

e) Consider the eigen decomposition of A

$$A = Q \Lambda Q^{-1}$$

$$\Lambda = \begin{bmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \lambda_i \end{bmatrix} = \begin{bmatrix} 1 & & 0 \\ & \ddots & \\ 0 & & 1 \end{bmatrix} = I$$

$$\therefore A = Q Q^{-1}$$

But $A = \Sigma^{-1} \tilde{\Sigma}$

$$\Sigma^{-1} \tilde{\Sigma} = Q Q^{-1}$$

This implies that $\Sigma^{-1} = \tilde{\Sigma}^{-1}$

$$\therefore \Sigma = \tilde{\Sigma}$$

This is the value of Σ that maximizes $p(O|\Sigma)$

$$\therefore \tilde{\Sigma} = \hat{\Sigma}_{ML} = \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu)(x_n - \mu)^T$$

f) An alternative way of finding $\hat{\Sigma}_{ML}$ would be:

$$\hat{\Sigma}_{ML} = \underset{\Sigma}{\operatorname{argmax}} \; p(O|\Sigma)$$

$$= \underset{\Sigma}{\operatorname{argmin}} \; -\log(p(O|\Sigma))$$

We can take the gradient of $-\log(p(O|\Sigma))$ and equate it to 0. Under first order optimality conditions and if $-\log(p(O|\Sigma))$ is convex, we can get $\hat{\Sigma}_{ML}$

g) The unbiased estimate of Σ is given by:

$$\hat{\Sigma}_{unbias} = \frac{1}{N-1} \sum_{n=1}^{N} (x_n - \hat{\mu})(x_n - \hat{\mu})^T$$

Exercise 2:

a)

$$P_{Y|X}(C_1|x) = \frac{P_{X|Y}(x|C_1)\,P(C_1)}{P_X(x)} \qquad P_{Y|X}(C_0|x) = \frac{P_{X|Y}(x|C_0)\,P(C_0)}{P_X(x)}$$

$$P_X(x) = P_{X|Y}(x|C_1) + P_{X|Y}(x|C_0)$$

$$P_{Y|X}(C_1|x) \underset{C_0}{\overset{C_1}{\gtrless}} P_{Y|X}(C_0|x)$$

$$\frac{P_{X|Y}(x|C_1)\,P(C_1)}{P_X(x)} \underset{C_0}{\overset{C_1}{\gtrless}} \frac{P_{X|Y}(x|C_0)\,P(C_0)}{P_X(x)}$$

$$\pi_1 \frac{1}{(2\pi)^{d/2}|\Sigma_1|^{1/2}} \exp\left\{\frac{-1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\right\} \underset{C_0}{\overset{C_1}{\gtrless}} \pi_0 \frac{1}{(2\pi)^{d/2}|\Sigma_0|^{1/2}} \exp\left\{\frac{-1}{2}(x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0)\right\}$$

Taking log on both sides

$$\frac{-1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) + \log\pi_1 - \frac{1}{2}\log|\Sigma_1| \underset{C_0}{\overset{C_1}{\gtrless}} \frac{-1}{2}(x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0) + \log\pi_0 - \frac{1}{2}\log|\Sigma_0|$$

b)

- The first few values in μ_0 are: [[0.48249575] [0.4864399]]
- The first few values in μ_1 are: [[0.44080734] [0.43871359]]
- The first few values in Σ_0 are:

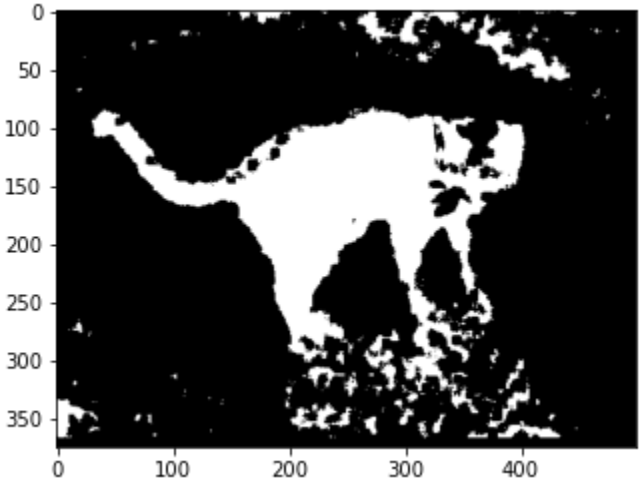
  [[0.064484 0.0369168]

  [0.0369168 0.06623457]]
- The first few values in Σ_1 are:

  [[0.04307832 0.03535405]

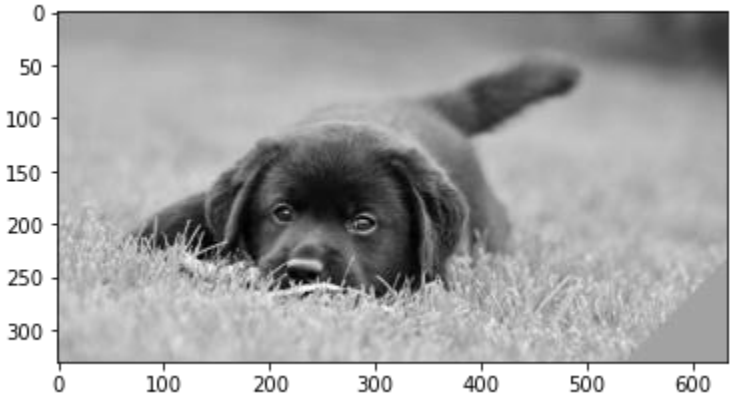
  [0.03535405 0.0424875]]
- π_0 = 0.83, π_1 = 0.17

c) Predicted Binary Mask

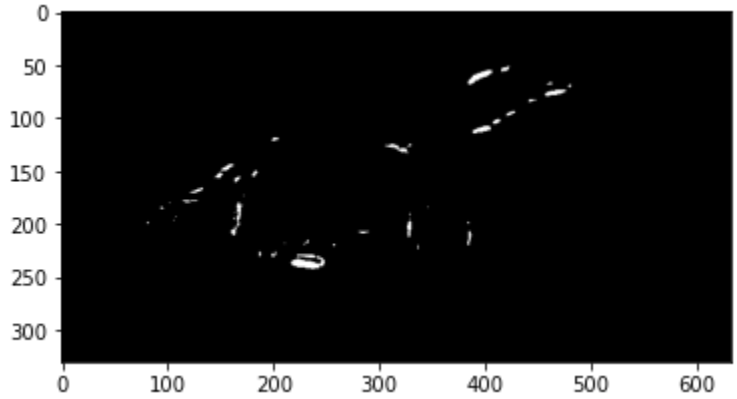


d) Mean Absolute Error = 0.087642

e) Test image and mask



Test image



Test image mask

The testing image does not perform well with the given Bayesian decision rule. This could be because the Class 1 pixels in the test image do not follow a similar distribution to the training pixels, and hence as misclassified.

Exercise 3:

a)

$$\frac{p_{X|Y}(x|C_1)}{p_{X|Y}(x|C_0)} \underset{C_0}{\overset{C_1}{\gtrless}} \tau$$

$$\frac{p_{X|Y}(x|C_1)}{p_{X|Y}(x|C_0)} = \frac{(2\pi)^{d/2}}{(2\pi)^{d/2}} \frac{|\Sigma_0|^{1/2}}{|\Sigma_1|^{1/2}} \frac{\exp\left\{\frac{-1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1)\right\}}{\exp\left\{\frac{-1}{2}(x-\mu_0)^T \Sigma_0^{-1}(x-\mu_0)\right\}}$$

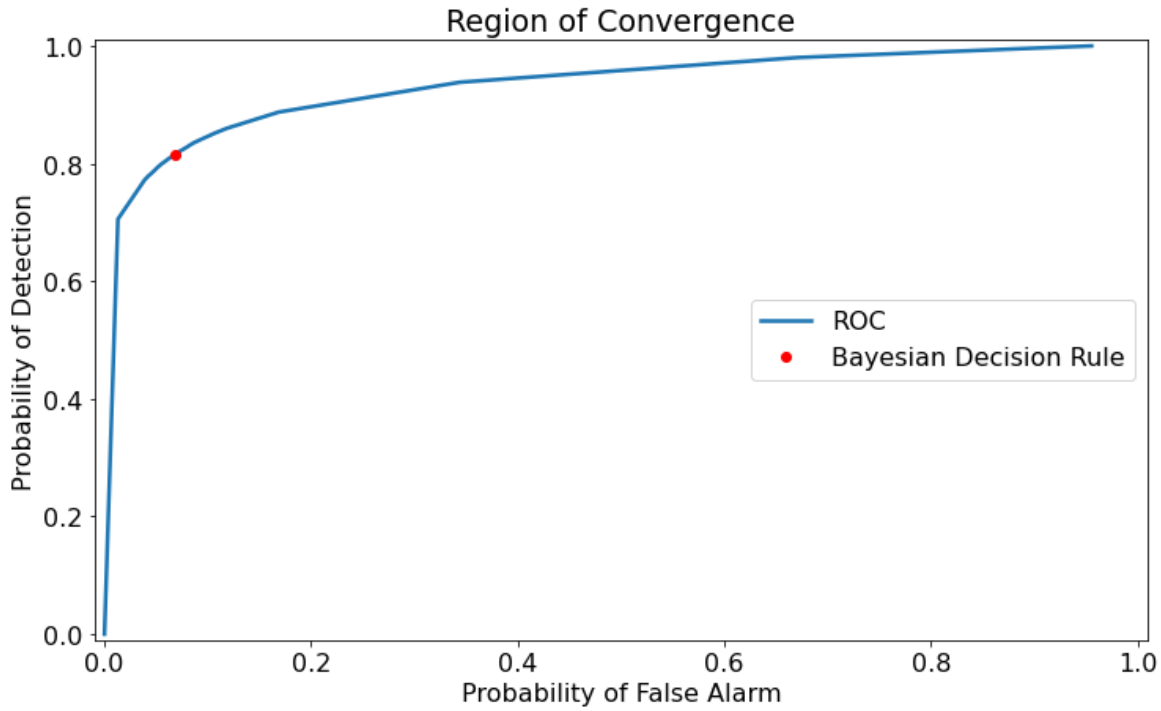
$$= \left(\frac{|\Sigma_0|}{|\Sigma_1|}\right)^{1/2} \exp\left\{\frac{1}{2}\left[(x-\mu_0)^T \Sigma_0^{-1}(x-\mu_0) - (x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1)\right]\right\} = \tau$$

But, $\frac{(2\pi)^{d/2}}{(2\pi)^{d/2}} \frac{|\Sigma_0|^{1/2}}{|\Sigma_1|^{1/2}} \frac{\exp\left\{\frac{-1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1)\right\}}{\exp\left\{\frac{-1}{2}(x-\mu_0)^T \Sigma_0^{-1}(x-\mu_0)\right\}} = \frac{\pi_0}{\pi_1}$

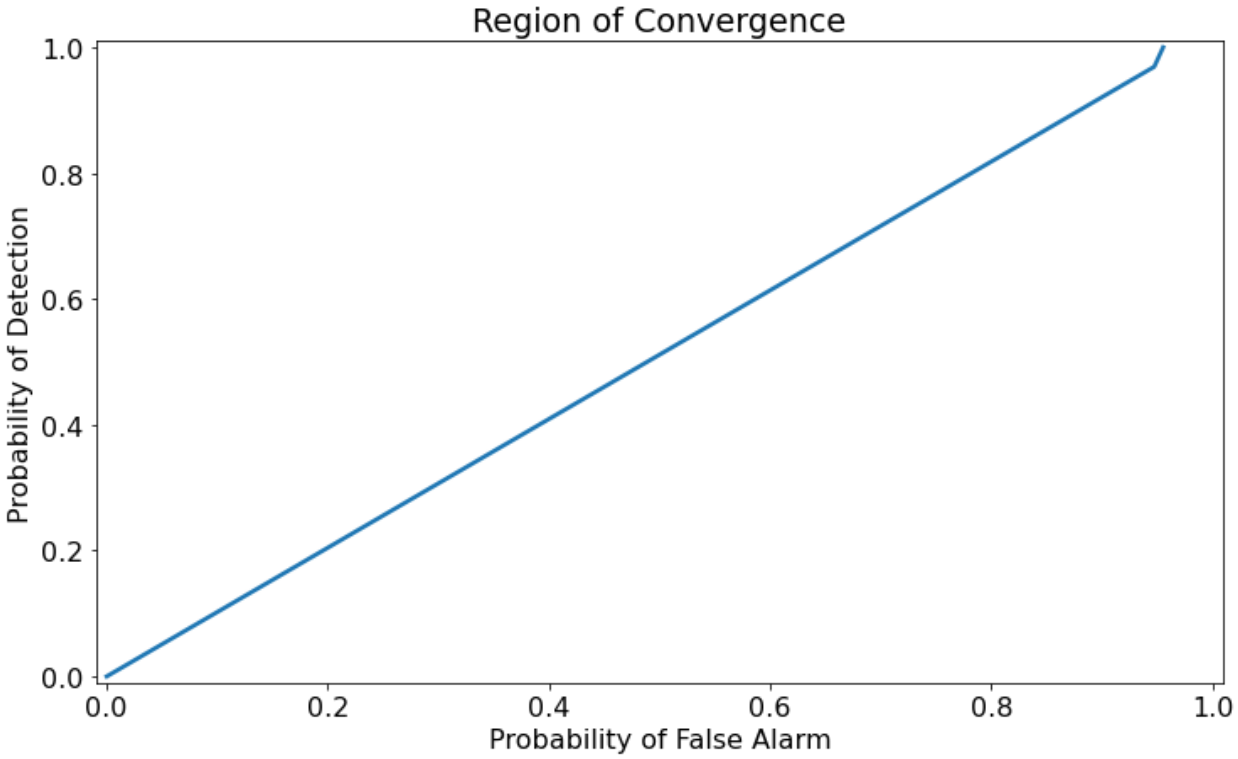
(From Exercise 2)

$$\therefore \boxed{\tau = \frac{\pi_0}{\pi_1}}$$

c)



d)


Region of Convergence
Probability of Detection
Probability of False Alarm
0.0
0.2
0.4
0.6
0.8
1.0

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.

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.

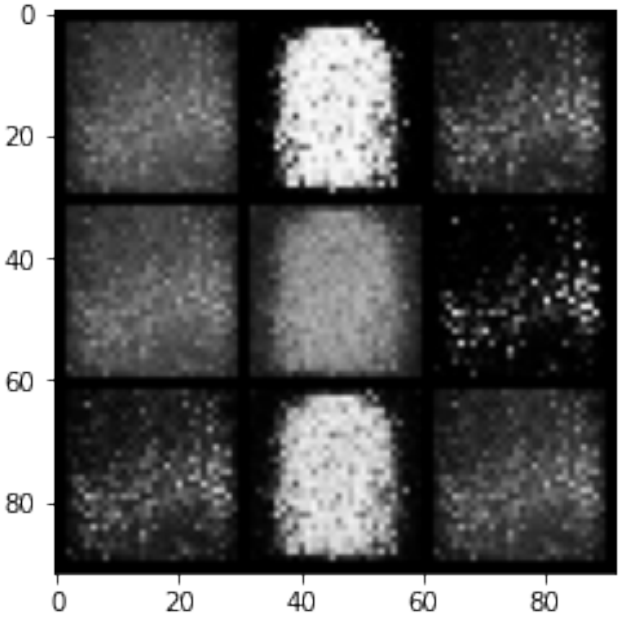
Figure 1. Boot

Figure 2. T-shirt

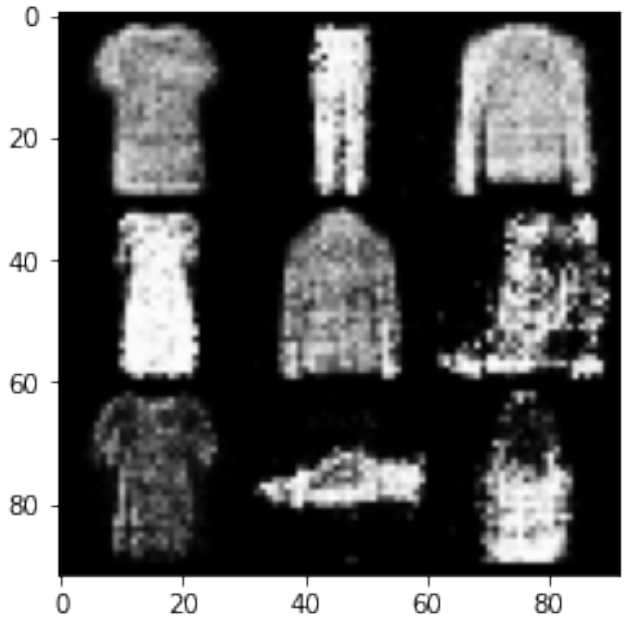
2. Homework 3

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.



Epoch 1



Epoch 30

3. Acknowledgements

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

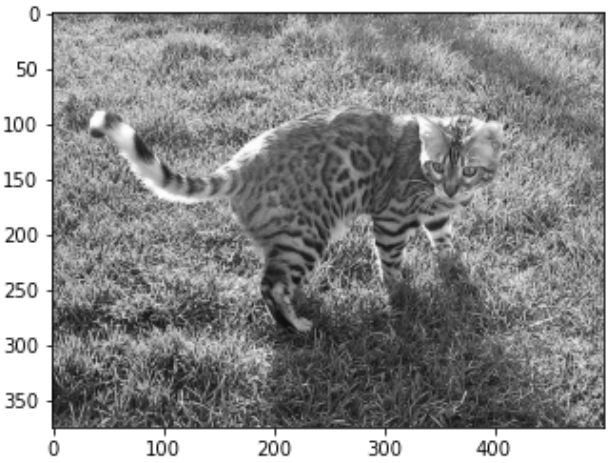
Exercise 2

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import csv
```

In [ ]:
```
path = "C:\\Users\\Parth\\OneDrive - purdue.edu\\Spring 2023\\ECE ML\\Homework\\HW3\\hom

train_cat = np.matrix(np.loadtxt(path + 'train_cat.txt', delimiter = ','))
train_grass = np.matrix(np.loadtxt(path + 'train_grass.txt', delimiter = ','))
```

In [ ]:
```
Y = plt.imread(path + 'cat_grass.jpg') / 255
plt.imshow(Y,cmap='gray')
print(f"Dimensions of the image are {Y.shape}")
```

```
Dimensions of the image are (375, 500)
```



In [ ]:
```
train_cat = []

with open(path + 'train_cat.txt') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')

    for data in csv_reader:
      train_cat.append(data)

train_cat = np.array(train_cat, dtype='float64')
train_cat = train_cat
K1 = train_cat.shape[1]
print(f"K1 = {K1}")
```

```
K1 = 1976
```

In [ ]:
```
train_grass = []

with open(path + 'train_grass.txt') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter=',')

    for data in csv_reader:
      train_grass.append(data)

train_grass = np.array(train_grass, dtype='float64')
```

```
K0 = train_grass.shape[1]
print(f"K0 = {K0}")
```

```
K0 = 9556
```

In [ ]:
```
pi_1 = K1/(K1 + K0)
pi_0 = 1 - pi_1

print(f"pi_0 = {pi_0:.2f}, pi_1 = {pi_1:.2f}")
```

```
pi_0 = 0.83, pi_1 = 0.17
```

In [ ]:
```
mu_0 = np.sum(train_grass, axis=1)/train_grass.shape[1]
mu_1 = np.sum(train_cat, axis=1)/train_cat.shape[1]
mu_0 = mu_0.reshape(64,1)
mu_1 = mu_1.reshape(64,1)


print(f"The first few values in mu_0 are: {mu_0[:2]}")
print(f"The first few values in mu_1 are: {mu_1[:2]}")
```

```
The first few values in mu_0 are: [[0.48249575]
 [0.4864399 ]]
The first few values in mu_1 are: [[0.44080734]
 [0.43871359]]
```

In [ ]:
```
sigma_0 = np.zeros((64, 64))
for i in train_grass.T:
  j = i.reshape((64,1))
  temp = np.reshape(j - mu_0, (64,1))
  sigma_0 += np.matmul(temp, temp.T)
sigma_0 = sigma_0/(train_grass.shape[1] - 1)

sigma_1 = np.zeros((64, 64))
for i in train_cat.T:
  j = i.reshape((64,1))
  temp = np.reshape(j - mu_1, (64,1))
  sigma_1 += np.matmul(temp, temp.T)
sigma_1 = sigma_1/(train_cat.shape[1] - 1)
```

In [ ]:
```
print(f"The first few values in sigma_0 are: \n{sigma_0[:2,:2]}")
print(f"The first few values in sigma_1 are: \n{sigma_1[:2,:2]}")
```

```
The first few values in sigma_0 are:
[[0.064484   0.0369168 ]
 [0.0369168  0.06623457]]
The first few values in sigma_1 are:
[[0.04307832 0.03535405]
 [0.03535405 0.0424875 ]]
```

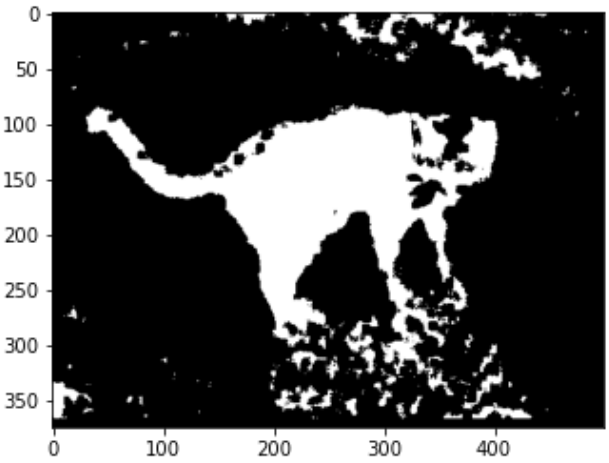
In [ ]:
```
sig_0_inv = np.linalg.inv(sigma_0)
sig_1_inv = np.linalg.inv(sigma_1)
sig_0_det = np.linalg.det(sigma_0)
sig_1_det = np.linalg.det(sigma_1)
```

In [ ]:
```
truth = plt.imread(path + 'truth.png')
truth = np.round(np.array(truth, dtype='float64'))
Beta = 0
Alpha = 0
```

```
Tot_P = np.count_nonzero(truth == 1)
Tot_N = np.count_nonzero(truth == 0)

M,N = Y.shape
mask = np.zeros(Y.shape)
for i in range(0,M-8):
  for j in range(0,N-8):
    block = Y[i:i+8, j:j+8] # This is a 8x8 block
    block = block.flatten()
    block = block.reshape(64,1)
    LHS = -0.5*np.matmul(np.matmul((block - mu_1).T, sig_1_inv),(block - mu_1)) + np.log
    RHS = -0.5*np.matmul(np.matmul((block - mu_0).T, sig_0_inv),(block - mu_0)) + np.log
    mask[i,j] = 1 if (LHS > RHS) else 0

    # TP, FP
    if (mask[i,j] == 1 and truth[i,j] == 1):
      Beta +=1

    if (mask[i,j] == 1 and truth[i,j] == 0):
      Alpha +=1

Beta = Beta/Tot_P
Alpha = Alpha/Tot_N
```

In [ ]:
```
plt.imshow(mask, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x20233098cf8>
```

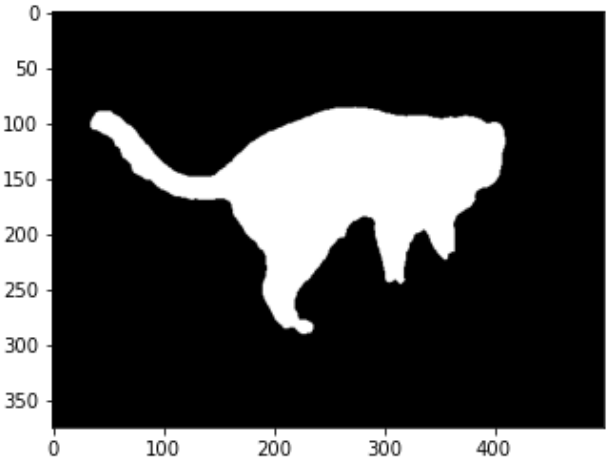


In [ ]:
```
M,N = mask.shape
MAE = np.sum(abs(truth[0:M-8, 0:N-8]-mask[0:M-8, 0:N-8]))/(M*N)
MAE
```

Out[ ]:
```
0.08764266666666666
```

In [ ]:
```
plt.imshow(truth, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x286512edc50>
```

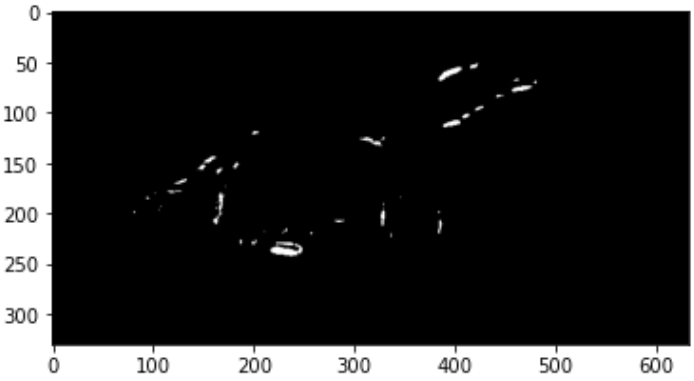


Test image

In [ ]:
```
from cv2 import cvtColor, COLOR_BGR2GRAY, imread

test = imread(path + 'doge3.jpg')
test = np.round(np.array(test, dtype='float32'))
test = cvtColor(test, COLOR_BGR2GRAY)

M,N = test.shape
mask = np.zeros(test.shape)
for i in range(0,M-8):
  for j in range(0,N-8):
    block = test[i:i+8, j:j+8] # This is a 8x8 block
    block = block.flatten()
    block = block.reshape(64,1)
    LHS = -0.5*np.matmul(np.matmul((block - mu_1).T, sig_1_inv),(block - mu_1)) + np.log
    RHS = -0.5*np.matmul(np.matmul((block - mu_0).T, sig_0_inv),(block - mu_0)) + np.log
    mask[i,j] = 1 if (LHS > RHS) else 0
```

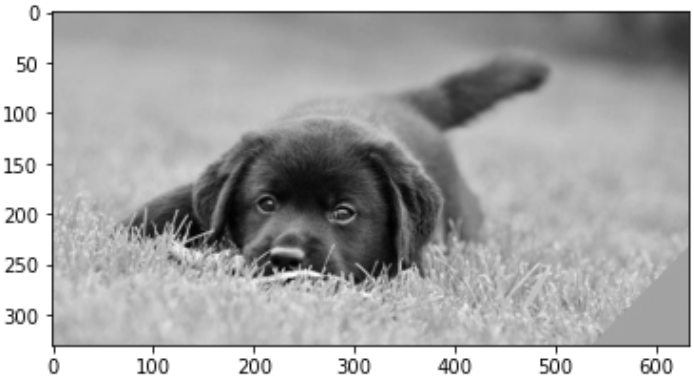
In [ ]:
```
plt.imshow(mask, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x202337de1d0>
```



In [ ]:
```
plt.imshow(test, cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x2023383ec88>
```



Exercise 3

Likelihood Ratio Test

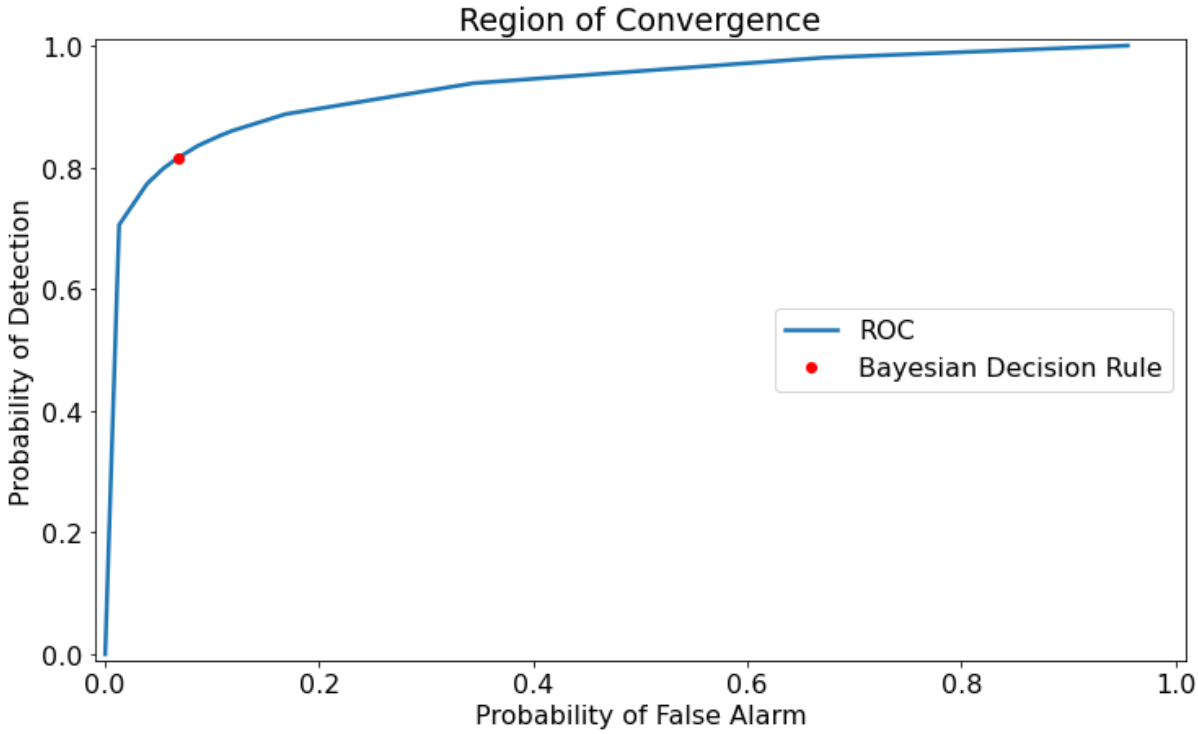
In [ ]:
```
tau_array = np.log(np.logspace(-1000,1000, base=np.e, num=20))
tau_array = np.array([-1000, -500,  -100, -50, -20,  -10,    -9,    -8,    -7,    -6,
           -3,    -2,    -1,     0,     1,     2,     3,     4,     5,
            6,     7,     8,     9, 10, 20, 50, 100, 500,  1000])
Beta_array = np.zeros(len(tau_array))
Alpha_array = np.zeros(len(tau_array))

M,N = Y.shape
mask = np.zeros(Y.shape)
for tau in range(len(tau_array)):
  for i in range(0,M-8):
    for j in range(0,N-8):
      block = Y[i:i+8, j:j+8] # This is a 8x8 block
      block = block.flatten()
      block = block.reshape(64,1)
      LHS = -0.5*np.matmul(np.matmul((block - mu_1).T, sig_1_inv),(block - mu_1)) - 0.5*
      RHS = -0.5*np.matmul(np.matmul((block - mu_0).T, sig_0_inv),(block - mu_0)) - 0.5*
      mask[i,j] = 1 if (LHS - RHS>= tau_array[tau]) else 0

      # TP, FP
      if (mask[i,j] == 1 and truth[i,j] == 1):
        Beta_array[tau] +=1

      if (mask[i,j] == 1 and truth[i,j] == 0):
        Alpha_array[tau] +=1

Beta_array = Beta_array/Tot_P
Alpha_array = Alpha_array/Tot_N
```

```
c:\Users\Parth\miniconda3\envs\TF\lib\site-packages\numpy\core\function_base.py:265: Run
timeWarning: overflow encountered in power
  return _nx.power(base, y)
c:\Users\Parth\miniconda3\envs\TF\lib\site-packages\ipykernel_launcher.py:1: RuntimeWarn
ing: divide by zero encountered in log
  """Entry point for launching an IPython kernel.
```

In [ ]:
```
import matplotlib
matplotlib.rcParams.update({'font.size': 16})
plt.figure(figsize=(12,7))
```

```
plt.plot(Alpha_array, Beta_array, linewidth=2.5)
plt.plot(Alpha, Beta, 'o',c='r')
plt.title("Region of Convergence")
plt.xlabel("Probability of False Alarm")
plt.ylabel("Probability of Detection")
plt.xlim(-0.01,1.01)
plt.ylim(-0.01,1.01)
plt.legend(["ROC","Bayesian Decision Rule"], loc=5)
```

Out[ ]: <matplotlib.legend.Legend at 0x28656ee04e0>



Linear Regression

In [ ]:
```
X1 = train_cat.T
X0 = train_grass.T
A = np.vstack((X1,X0))

y1 = [1 for i in range(X1.shape[0])]
y0 = [-1 for i in range(X0.shape[0])]
y = np.array(y1 + y0)
b = y.reshape((y.shape[0],1))

t1 = np.linalg.inv(np.matmul(A.T,A))
t2 = np.matmul(A.T, b)
theta = np.matmul(t1, t2)
```

In [ ]:
```
tau_array = np.array([-1000, -500,  -100, -50, -20,  -10,    -9,    -8,    -7,    -6,
           -3,    -2,    -1,     0,     1,     2,     3,     4,     5,
            6,     7,     8,     9, 10, 20, 50, 100, 1000])
Beta_array = np.zeros(len(tau_array))
Alpha_array = np.zeros(len(tau_array))

M,N = Y.shape
mask = np.zeros(Y.shape)
```

```
for tau in range(len(tau_array)):
  for i in range(0,M-8):
    for j in range(0,N-8):
        block = Y[i:i+8, j:j+8] # This is a 8x8 block
        block = block.flatten()
        block = block.reshape(64,1)
        test = np.matmul(theta.T, block)
        mask[i,j] = 1 if (test >= tau_array[tau]) else 0

        # TP, FP
        if (mask[i,j] == 1 and truth[i,j] == 1):
            Beta_array[tau] +=1

        if (mask[i,j] == 1 and truth[i,j] == 0):
            Alpha_array[tau] +=1

Beta_array = Beta_array/Tot_P
Alpha_array = Alpha_array/Tot_N
```

In [ ]:
```
import matplotlib
matplotlib.rcParams.update({'font.size': 16})
plt.figure(figsize=(12,7))
plt.plot(Alpha_array, Beta_array, linewidth=2.5)
plt.title("Region of Convergence")
plt.xlabel("Probability of False Alarm")
plt.ylabel("Probability of Detection")
plt.xlim(-0.01,1.01)
plt.ylim(-0.01,1.01)
```

Out[ ]: (-0.01, 1.01)