hw3

March 2, 2023

[ ]:
```python
import math
from pathlib import Path
from typing import Tuple, Callable, List
from IPython.display import display, Math, Image

import cv2
import numba
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cvxpy as cp

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

0.1 Exercise 1:

[ ]:
```python
display(Image(filename="./images/hw3_p1a.png", height=400, width=500))
```

Exercise 1a)

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

$$= tr[(Ax)x^T]$$

$$= x^T A x$$

$* \; tr[ba^T] = a^T b$

$\forall \, a, b \in \mathbb{R}^n$

[ ]:
```python
display(Image(filename="./images/hw3_p1b.png", height=400, width=500))
```

Exercise 1b)

$$P(D|\Sigma) = \prod_{n=1}^{N} \left\{ \underbrace{\frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}}}_{\text{constant w.r.t. } n} \exp\left\{ -\frac{1}{2}(\vec{x}_n - \vec{\mu})^T \Sigma^{-1} (\vec{x}_n - \vec{\mu}) \right\} \right\}$$

$$= \frac{1}{(2\pi)^{Nd/2}} |\Sigma^{-1}|^{N/2} \prod_{n=1}^{N} \exp\left\{ -\frac{1}{2}(\vec{x}_n - \vec{\mu})^T \Sigma^{-1} (\vec{x}_n - \vec{\mu}) \right\}$$

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

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

$$* \quad tr(A+B) = tr(A) + tr(B)$$

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

```
[ ]: display(Image(filename="./images/hw3_p1c.png", height=400, width=500))
```

Exercise 1c)

from part 1b,

$$P(D|\Sigma) = \frac{1}{(2\pi)^{Nd/2}} |\Sigma^{-1}|^{N/2} \exp\left\{-\frac{1}{2} \underbrace{tr\left[\Sigma^{-1} N \tilde{\Sigma}\right]}_{= N\, tr[A]}\right\}$$

$$* \; A = \Sigma^{-1} \tilde{\Sigma} \rightarrow A \tilde{\Sigma}^{-1} = \Sigma^{-1}$$

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

$$* \; \det(AB) = \det(A)\det(B)$$

$$= \frac{1}{(2\pi)^{Nd/2}} |\tilde{\Sigma}^{-1}|^{N/2} |A| \exp\left\{-\frac{N}{2} \sum_{i=1}^{d} \lambda_i\right\}$$

$$\boxed{= \frac{1}{(2\pi)^{Nd/2} |\tilde{\Sigma}|^{N/2}} \left(\prod_{i=1}^{d} \lambda_i\right)^{N/2} \exp\left\{-\frac{N}{2} \sum_{i=1}^{d} \lambda_i\right\}}$$

```
[ ]: display(Image(filename="./images/hw3_p1d.png", height=400, width=500))
```

Exercise 1D)

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

$$= \underset{\lambda_i}{\operatorname{argmin}} -\log\left(\cancel{\text{constant term}} \left(\prod_{i=1}^{d}\lambda_i\right)^{\frac{N}{2}} \exp\left\{-\frac{N}{2}\sum_{i=1}^{d}\lambda_i\right\}\right)$$

$$= \underset{\lambda_i}{\operatorname{argmin}} -\frac{N}{2}\sum_{i=1}^{d} \log(\lambda_i) - -\frac{N}{2}\sum_{i=1}^{d} \lambda_i$$

$$= \frac{\partial}{\partial\lambda_i} -\frac{N}{2}\sum_{i=1}^{d} \log(\lambda_i) + \frac{N}{2}\sum_{i=1}^{d}\lambda_i = 0$$

$$\cancel{\frac{N}{2}\sum_{i=1}^{d}} \frac{\partial}{\partial\lambda_i}\lambda_i = \cancel{\frac{N}{2}\sum_{i=1}^{d}}\frac{\partial}{\partial\lambda_i}\log_e(\lambda_i)$$

$$1 = \frac{1}{\lambda_i} \Longrightarrow \boxed{\lambda_i = 1}$$

0.1.1 Problem 1d:

With $\lambda_i = 1$, we look at the Eigen Decomposition of A:

$$A = QIQ^{-1} = QQ^{-1} = I$$

Since we maximized $p(\mathcal{D}|\Sigma)$ with respect to λ_i, we have:

$$A = \Sigma^{-1}\tilde{\Sigma} = \Sigma_{ML}^{-1}\tilde{\Sigma} = I$$

Thus,

$$\Sigma_{ML} = \tilde{\Sigma}$$

0.1.2 Problem 1f:

An alternative to finding this result, at least numerically would be Gradient Descent.

0.1.3 Problem 1g:

An unbiased estimate is:

[ ]:
```
display(Math(r"\hat{\Sigma}_{unbias} = \frac{1}{N-1} \sum_{n=1}^{N} (\vec{x}_n␣
↪- \hat{\vec{\mu}})(\vec{x}_n - \hat{\vec{\mu}})^T"))
```

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

[ ]:
```
# Load the training data
def data_directory() -> Path:
    return Path().cwd() / "data"

train_cat = np.matrix(np.loadtxt(str(data_directory() / "train_cat.txt"),␣
↪delimiter=","))
train_grass = np.matrix(np.loadtxt(str(data_directory() / "train_grass.txt"),␣
↪delimiter=","))

print(f"Training Cat Shape: {train_cat.shape}")
print(f"Training Grass Shape: {train_grass.shape}")
```

```
Training Cat Shape: (64, 1976)
Training Grass Shape: (64, 9556)
```

0.2 Exercise 2: Bayesian Decision Rule

[ ]:
```
display(Image(filename="./images/hw3_p2a.png", height=400, width=500))
```

Exercise 2A)

$$P_{Y|X}(c_1|\vec{x}) \underset{c_0}{\overset{c_1}{\gtrless}} P_{Y|X}(c_0|\vec{x})$$

· Take log() of both sides

$$\log P_{Y|X}(c_1|\vec{x}) \underset{c_0}{\overset{c_1}{\gtrless}} \log P_{Y|X}(c_0|\vec{x})$$

$$\log P_{X|Y}(\vec{x}|c_1) + \log \overbrace{P_Y(c_1)}^{\pi_1} - \cancel{\log P_X(\vec{x})} \underset{c_0}{\overset{c_1}{\gtrless}}$$

$$\log P_{X|Y}(\vec{x}|c_0) + \log \underbrace{P_Y(c_0)}_{\pi_0} - \cancel{\log P_X(\vec{x})}$$

$$= \cancel{-\frac{d}{2}\log 2\pi} - \frac{1}{2}\log|\Sigma_1| - \frac{1}{2}(\vec{x}-\vec{\mu}_1)^T \Sigma_1^{-1}(\vec{x}-\vec{\mu}_1)$$

$$+ \log \pi_1 \underset{c_0}{\overset{c_1}{\gtrless}}$$

$$\cancel{-\frac{d}{2}\log 2\pi} - \frac{1}{2}\log|\Sigma_0| - \frac{1}{2}(\vec{x}-\vec{\mu}_0)^T \Sigma_0^{-1}(\vec{x}-\vec{\mu}_0) + \log \pi_0$$

$$\boxed{\begin{aligned} = -\tfrac{1}{2}(\vec{x}-\vec{\mu}_1)^T \Sigma_1^{-1}(\vec{x}-\vec{\mu}_1) + \log \pi_1 - \tfrac{1}{2}\log|\Sigma_1| \underset{c_0}{\overset{c_1}{\gtrless}} \\ -\tfrac{1}{2}(\vec{x}-\vec{\mu}_0)^T \Sigma_0^{-1}(\vec{x}-\vec{\mu}_0) + \log \pi_0 - \tfrac{1}{2}\log|\Sigma_0| \end{aligned}}$$

0.2.1 Problem 2b:

[ ]:
```
gN, gM = train_grass.shape
cN, cM = train_cat.shape




# Estimate means
mu0 = train_grass.mean(axis=1)
mu1 = train_cat.mean(axis=1)
# print(train_grass[1, :].sum() / gM)
# print(mu0)
display(Math(rf"\mu_0: {mu0[0:2]}"))
display(Math(rf"\mu_1: {mu1[0:2]}"))
```

```
# Estimate variances
sigma0 = np.cov(train_grass, bias=False)
sigma1 = np.cov(train_cat, bias=False)
display(Math(rf"\Sigma_0:"))
print(sigma0[0:2, 0:2])
# DEBUG:
# i = 1; j = 1
# print(1 / (gM - 1) * ((train_grass[i, :] - mu0[i]) * (train_grass[j, :] -
↪mu0[j]).T))
display(Math(rf"\Sigma_1:"))
print(sigma1[0:2, 0:2])

# Estimate priors
pi0 = gM / (cM + gM)
pi1 = cM / (cM + gM)
display(Math(rf"\pi_0: {pi0}"))
display(Math(rf"\pi_1: {pi1}"))
```

μ_0 : [[0.48249575][0.4864399]]

μ_1 : [[0.44080734][0.43871359]]

Σ_0 :

```
[[0.064484   0.0369168 ]
 [0.0369168  0.06623457]]
```

Σ_1 :

```
[[0.04307832 0.03535405]
 [0.03535405 0.0424875 ]]
```

π_0 : 0.828650711064863

π_1 : 0.171349288935137

0.2.2 Problem 2c:

[ ]:
```
# 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(x: np.array) -> int:
    '''Returns 0/1 depending on classification'''
    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)
```

```python
    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

@numba.jit
def classify(img: np.matrix) -> np.matrix:
    M, N = img.shape
    P = np.zeros(shape=(M-8, N-8)) - 1 # Initialize prediction matrix to -1's
    for i in range(M-8):
        for j in range(N-8):
            block = img[i:i+8, j:j+8].copy()
            x = block.reshape(64, 1)
            P[i, j] = make_decision(x=x)
    return P

Y = plt.imread(str(data_directory() / "cat_grass.jpg")) / 255
P = classify(Y)

fig = plt.figure()
plt.imshow(Y, cmap=plt.cm.gray)
plt.title("Cat in grass (training)")
fig.gca().axes.xaxis.set_visible(False)
fig.gca().axes.yaxis.set_visible(False)
fig = plt.figure()
plt.imshow(P, cmap=plt.cm.gray)
plt.title("Classification")
fig.gca().axes.xaxis.set_visible(False)
fig.gca().axes.yaxis.set_visible(False)
```

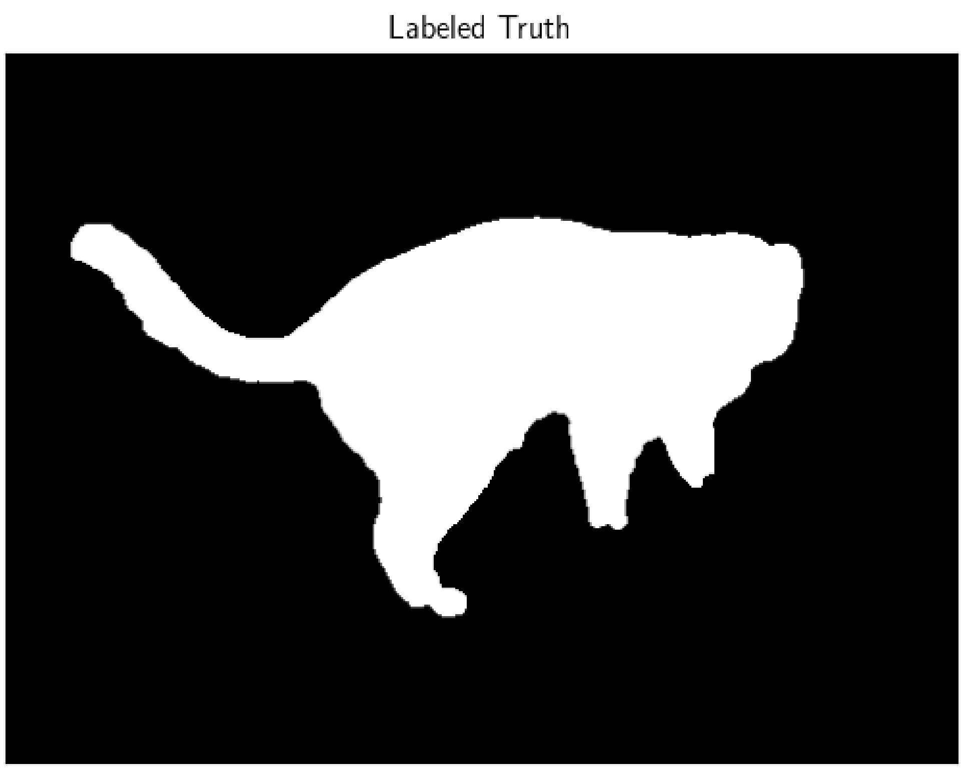
Cat in grass (training)

Classification

0.2.3 Problem 2d:

[ ]:
```
# Calculate the Mean Absolute Error (MAE)
Y_truth = plt.imread(str(data_directory() / "truth.png")) / 255
Y_truth[(Y_truth > 0)] = 1.0 # Truth data should be class one where cat is
M, N = Y_truth.shape
Y_truth = Y_truth[0:M-8, 0:N-8]
n_pixels = Y_truth.size
MAE = np.abs(P - Y_truth).sum() / n_pixels
print(f"MAE: {MAE}")

fig = plt.figure()
plt.imshow(Y_truth, cmap=plt.cm.gray)
plt.title("Labeled Truth")
fig.gca().axes.xaxis.set_visible(False)
fig.gca().axes.yaxis.set_visible(False)
```

```
MAE: 0.09351254956691256
```

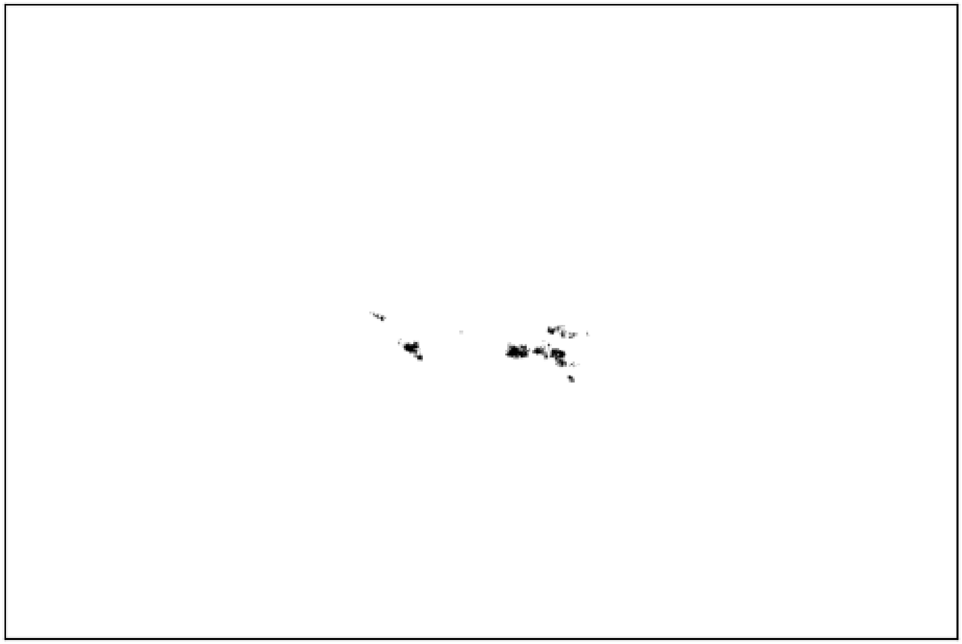


0.2.4 Problem 2e:

```
[ ]: img = cv2.imread(str(data_directory() / "test_wild3.jpeg"))
converted = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
converted: np.matrix = np.asarray(converted) / 255

P2 = classify(converted)
print(P2.max())

fig = plt.figure()
plt.imshow(converted, cmap=plt.cm.gray)
plt.title("Cat in grass (from Google)")
fig.gca().axes.xaxis.set_visible(False)
fig.gca().axes.yaxis.set_visible(False)
fig = plt.figure()
plt.imshow(P2, cmap=plt.cm.gray)
plt.title("Classification")
fig.gca().axes.xaxis.set_visible(False)
fig.gca().axes.yaxis.set_visible(False)
```

```
1.0
```

Cat in grass (from Google)



The model performs very poorly as observed above. This is mainly due to the small amount of training data. We estimated the model parameters based on one example of cat + grass. Considering the number of different cat + grass images available, this is not likely to perform well on an out of sample data.

0.3 Exercise 3: Receiver Operating Curve (ROC)

0.3.1 Problem 3a:

[ ]:
```
display(Image(filename="./images/hw3_p3a.png", height=400, width=500))

tau = pi0 / pi1
display(Math(rf"\tau: {tau}, \ log(\tau): {math.log(tau)}"))
```

Exercise 3A)

$$\frac{P_{X|Y}(\vec{x}\,|\,c_1)}{P_{X|Y}(\vec{x}\,|\,c_0)} \underset{c_0}{\overset{c_1}{\gtrless}} \tau$$

* Consider ratio of posteriors from equation (9)

$$\frac{\dfrac{P_{X|Y}(c_1\,|\,\vec{x})\,P_Y(c_1)}{\cancel{P_X(\vec{x})}}}{\dfrac{P_{X|Y}(c_0\,|\,\vec{x})\,P_Y(c_0)}{\cancel{P_X(\vec{x})}}} \underset{c_0}{\overset{c_1}{\gtrless}} 1$$

$$\frac{P_{X|Y}(c_1\,|\,\vec{x})}{P_{X|Y}(c_0\,|\,\vec{x})} \underset{c_0}{\overset{c_1}{\gtrless}} \underbrace{\frac{P_Y(c_0)}{P_Y(c_1)}}_{\tau}$$

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

$\tau : 4.836032388663967,\ log(\tau) : 1.5760946301378869$

0.3.2 Problem 3b & 3c:

```
[ ]: @numba.jit
     def log_likelihood_ratio(x: np.array) -> int:
         '''Returns 0/1 depending on classification'''
         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)
```

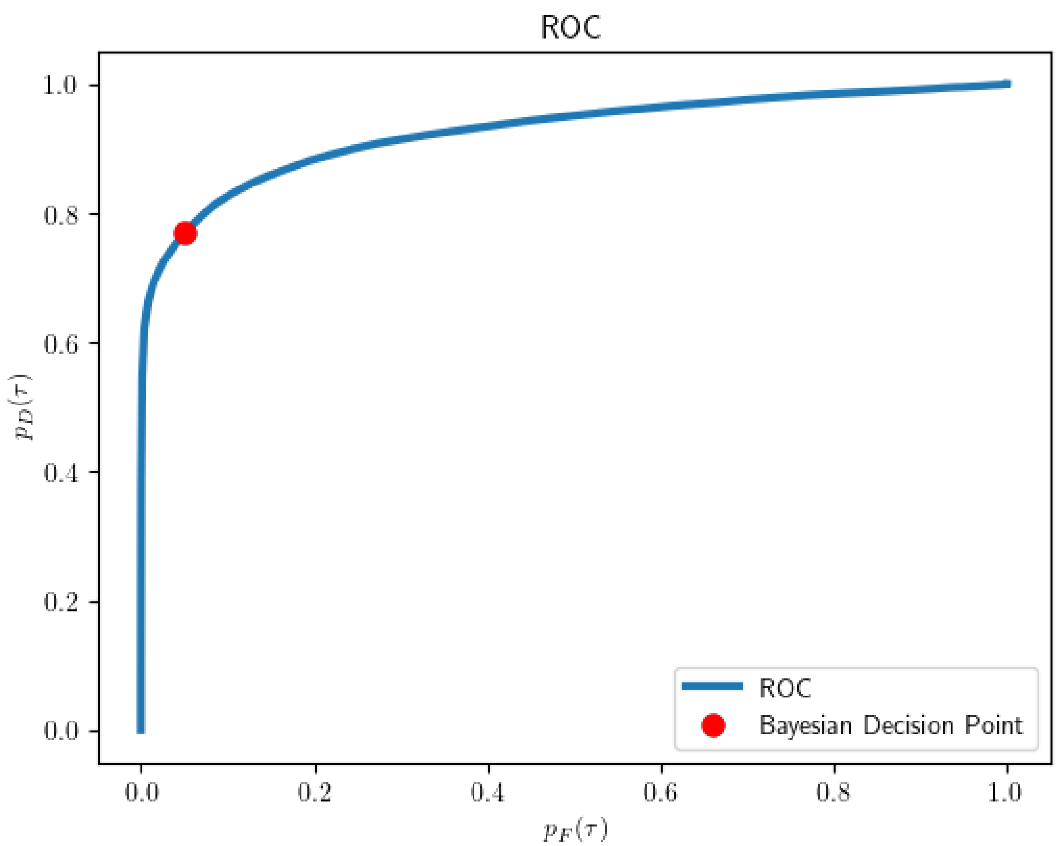
```
    c1 = -1/2 * xmmu1.T @ sigma_1_inv @ xmmu1 + math.log(pi1) - 1/2 * math.
 ↪log(sigma_1_det)
    return c1 - c0

@numba.jit
def count_positives(img: np.matrix, truth: np.matrix, tau: float) -> Tuple[int,␣
 ↪int]:
    M, N = img.shape
    true_positives = 0
    false_positives = 0
    # llr_values = []
    # Loop over the image and determine if this is a true/false positive
    for i in range(M-8):
        for j in range(N-8):
            block = img[i:i+8, j:j+8].copy()
            x = block.reshape(64, 1)
            llr = log_likelihood_ratio(x=x)
            predicted_class = 1 if llr > tau else 0
            # llr_values.append(llr)
            truth_class = truth[i, j]
            if predicted_class > 0:
                if truth_class > 0:
                    true_positives += 1
                else:
                    false_positives += 1
    # print(max(llr_values), min(llr_values))
    return (true_positives, false_positives)

total_positives = (Y_truth > 0).sum()
total_negatives = (Y_truth == 0).sum()
print(f"Total Positives: {total_positives}, Total Negatives: {total_negatives}")

# Loop over different values of tau
count = 100
pds = []
pfs = []
for tau_i in np.linspace(-365, 50, count):
    tp, fp = count_positives(Y, Y_truth, tau = tau_i)
    pds.append(tp / total_positives)
    pfs.append(fp / total_negatives)
```

```
Total Positives: 33135, Total Negatives: 147429
```

[ ]:
```
# Calculate the Bayesian decision point on the ROC
bay_tp, bay_fp = count_positives(Y, Y_truth, tau = tau)
bay_pd = bay_tp / total_positives
bay_pf = bay_fp / total_negatives
```

```
fig = plt.figure()
plt.plot(pfs, pds, linewidth=3)
plt.plot(bay_pf, bay_pd, "ro", markersize=8)
plt.legend(["ROC", "Bayesian Decision Point"])
plt.title("ROC")
plt.xlabel(r"$p_F(\tau)$")
plt.ylabel(r"$p_D(\tau)$")
```

[ ]: Text(0, 0.5, '$p_D(\\tau)$')



0.3.3 Problem 3d:

[ ]:
```
@numba.jit
def count_positives(img: np.matrix, theta_hat: np.matrix, truth: np.matrix, tau:
 ↪ float) -> Tuple[int, int]:
    M, N = img.shape
    true_positives = 0
    false_positives = 0
    ls_values = [] # Least-Square values
```

```python
    # Loop over the image and determine if this is a true/false positive
    for i in range(M-8):
        for j in range(N-8):
            block = img[i:i+8, j:j+8].copy()
            x = block.reshape(64, 1)
            ls_value = theta_hat.T @ x
            predicted_class = 1 if ls_value > tau else 0
            ls_values.append(ls_value)
            truth_class = truth[i, j]
            if predicted_class > 0:
                if truth_class > 0:
                    true_positives += 1
                else:
                    false_positives += 1
    # print(max(ls_values), min(ls_values))
    return (true_positives, false_positives)

# Cat stacked on Grass
X = np.vstack((train_cat.T, train_grass.T))
print(X.shape)

b = np.vstack((
    np.ones((train_cat.shape[1], 1)),
    np.ones((train_grass.shape[1], 1)) * -1
))
print(b.shape)

# Solve linear regression problem using cvxpy
d = 64  # theta dimension
theta_hat = cp.Variable((d, 1))
objective = cp.Minimize(cp.sum_squares(X @ theta_hat - b))
constraints = []
prob = cp.Problem(objective, constraints)

optimal_objective_value = prob.solve()
display(Math(r"\hat{\theta} \text{ using cvxpy (first 10 values):}"))
# print(optimal_objective_value)
theta_hat = theta_hat.value
print(theta_hat[:10])

p, fp = count_positives(Y, theta_hat, Y_truth, tau = 1)

# Loop over different values of tau
count = 100
pds = []
pfs = []
for tau_i in np.linspace(-2, 0, count):
```

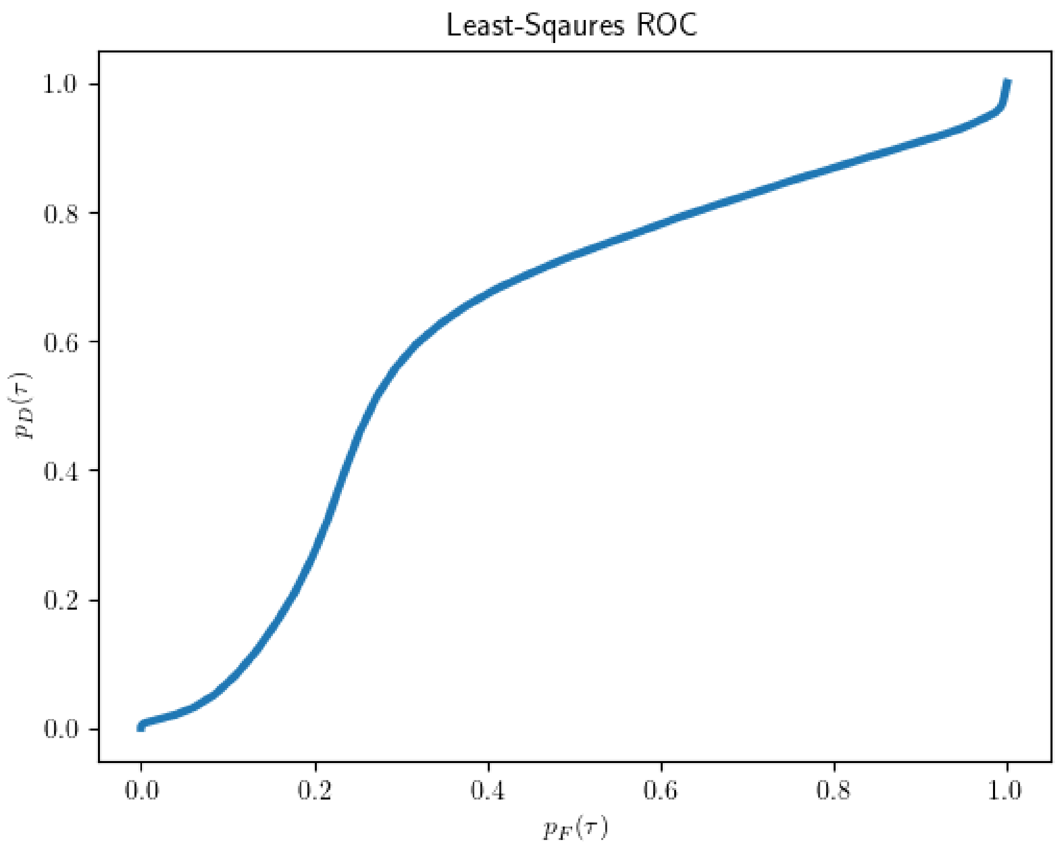
```
    tp, fp = count_positives(Y, theta_hat, Y_truth, tau = tau_i)
    pds.append(tp / total_positives)
    pfs.append(fp / total_negatives)

# Plot the least-squares ROC
fig = plt.figure()
plt.plot(pfs, pds, linewidth=3)
plt.title("Least-Sqaures ROC")
plt.xlabel(r"$p_F(\tau)$")
plt.ylabel(r"$p_D(\tau)$")
```

```
(11532, 64)
(11532, 1)
```

$\hat{\theta}$ using cvxpy (first 10 values):

```
[[-0.02586641]
 [-0.05143753]
 [-0.00874605]
 [-0.00465177]
 [-0.00940259]
 [-0.073091  ]
 [-0.01494949]
 [-0.03753226]
 [-0.00597663]
 [-0.03575012]]
```

[ ]:
```
Text(0, 0.5, '$p_D(\\tau)$')
```


Least-Sqaures ROC
1.0
0.8
0.6
0.4
0.2
0.0
$p_D(\tau)$
0.0
0.2
0.4
0.6
0.8
1.0
$p_F(\tau)$