CS561 HW09

November 4, 2023

1 Problem 1

(a and b) Suppose we define a quadratic discriminant classifier as such: we'd like to find a condition such that the distribution

$$x|Y=0 \sim \mathcal{N}(\mu_0, \Sigma_0)$$

and

$$x|Y=1 \sim \mathcal{N}(\mu_1, \Sigma_1)$$

where our classifier outputs, in consideration of the priors,

$$\frac{p(Y=0|x)}{p(Y=1|x)} = \frac{p(x|Y=0)p(Y=0)}{p(x|Y=1)p(Y=1)} \geq 1$$

as class 0 and 1 otherwise. In other words

$$\frac{p(Y=0)\frac{1}{(2\pi)^{1/2}|\Sigma_0|^{1/2}}\exp\left(-\frac{1}{2}(x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0)\right)}{p(Y=1)\frac{1}{(2\pi)^{1/2}|\Sigma_1|^{1/2}}\exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)\right)} \geq 1$$

taking the logarithm on both sides, we obtain

$$\log(p(Y=0))-\frac{1}{2}\log(|\Sigma_0|)-\frac{1}{2}(x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0) \geq \log(p(Y=1))-\frac{1}{2}\log(|\Sigma_1|)-\frac{1}{2}(x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1)$$

we can simplify the expression as follows:

$$x^T(\Sigma_1^{-1}-\Sigma_0^{-1})x+(\mu_0^T(\Sigma_0^{-1}+(\Sigma_0^{-1})^T)-\mu_1^T(\Sigma_1^{-1}+(\Sigma_1^{-1})^T))\,x \geq \log(|\Sigma_0|)-\log(|\Sigma_1|)+\mu_0^T\Sigma_0^{-1}\mu_0-\mu_1^T\Sigma_1^{-1}\mu_1+\log(p(Y$$

therefore, the threshold for $y=0$ can be written in the form

$$x^TBx + W^Tx \geq c$$

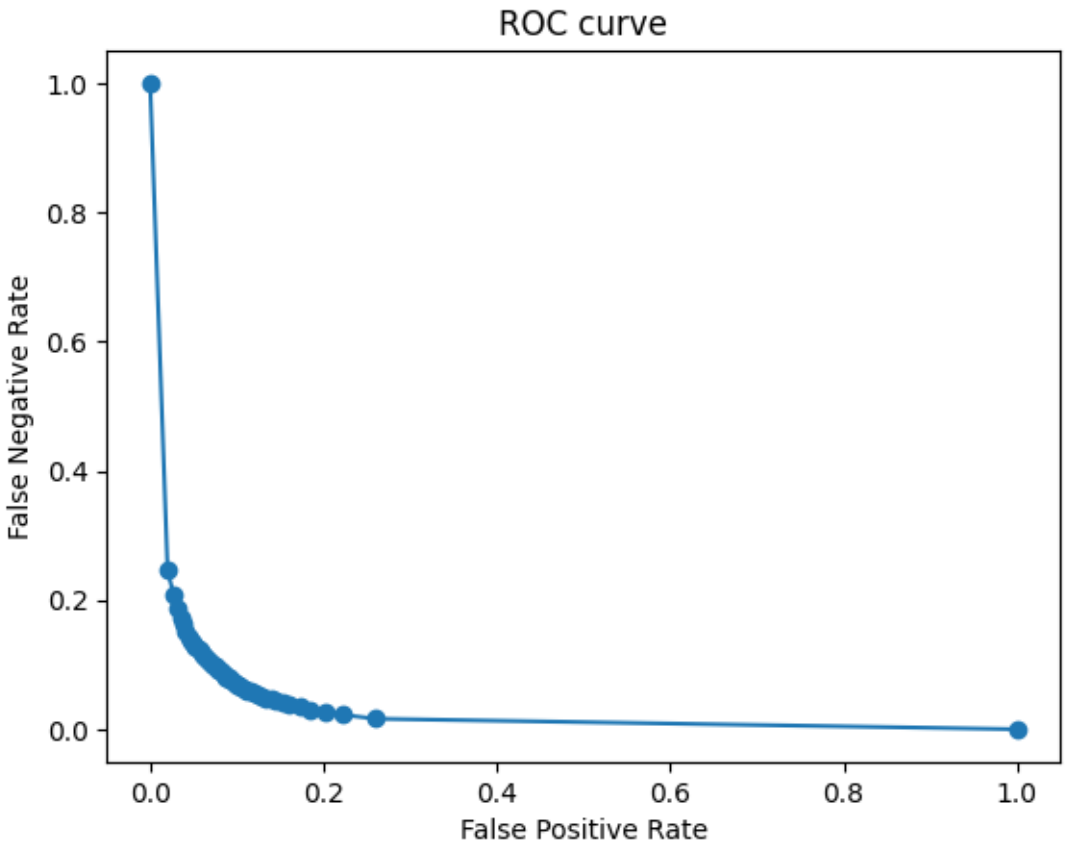
where

$$\begin{aligned} B &= \Sigma_1^{-1}-\Sigma_0^{-1} \\ W^T &= \mu_0^T(\Sigma_0^{-1}+(\Sigma_0^{-1})^T)-\mu_1^T(\Sigma_1^{-1}+(\Sigma_1^{-1})^T) \\ c &= \log(|\Sigma_0|)-\log(|\Sigma_1|)+\mu_0^T\Sigma_0^{-1}\mu_0-\mu_1^T\Sigma_1^{-1}\mu_1+\log(p(Y=1))-\log(p(Y=0)) \end{aligned}$$

[3]:
```
#1c
import numpy as np
import matplotlib.pyplot as plt
```

```
def classifier(x, p0=0.5):
    if p0==0:
        return 1
    if p0==1:
        return 0
    p1 = 1-p0
    mu0 = np.array([1,0])
    mu1 = np.array([0,2])
    sigma0 = np.array([[8,3],[3,2]])
    sigma1 = np.array([[1,0.1],[0.1,1]])

    B = np.linalg.inv(sigma1)-np.linalg.inv(sigma0)
    WT = mu0.T@(np.linalg.inv(sigma0)+np.linalg.inv(sigma0).T)-mu1.T@(np.linalg.
 ↪inv(sigma1)+np.linalg.inv(sigma1).T)
    c = np.log(np.linalg.det(sigma0))-np.log(np.linalg.det(sigma1))+mu0.T@np.
 ↪linalg.inv(sigma0)@mu0-mu1.T@np.linalg.inv(sigma1)@mu1+np.log(p1)-np.log(p0)
    if x.T@B@x+WT@x>=c:
        return 0
    else:
        return 1

def roc_entries(true, pred):
    tn = np.sum(np.logical_and(pred==0,true==0))
    tp = np.sum(np.logical_and(pred==1,true==1))
    fn = np.sum(np.logical_and(pred==0,true==1))
    fp = np.sum(np.logical_and(pred==1,true==0))
    fpr = fp / (fp+tn) # P(pred=1| true==0)
    fnr = fn / (fn+tp) # P(pred=0| true==1)
    return fpr, fnr

n = 10000
FPRs = []
FNRs = []
class_0 = np.random.multivariate_normal(np.array([1,0]), np.
 ↪array([[8,3],[3,2]]), n)
class_1 = np.random.multivariate_normal(np.array([0,2]), np.array([[1,0.1],[0.
 ↪1,1]]), n)
for p in [i/50 for i in range(0,51)]:
    pred = np.array([classifier(v, p0=p) for v in class_0] + [classifier(v, ␣
 ↪p0=p) for v in class_1])
    true = np.array([0]*n+[1]*n)
    fpr, fnr = roc_entries(true, pred)
    FPRs.append(fpr)
    FNRs.append(fnr)

plt.plot(FPRs, FNRs)
```

```
plt.scatter(FPRs, FNRs)
plt.title("ROC curve")
plt.xlabel("False Positive Rate")
plt.ylabel("False Negative Rate")
plt.show()
```



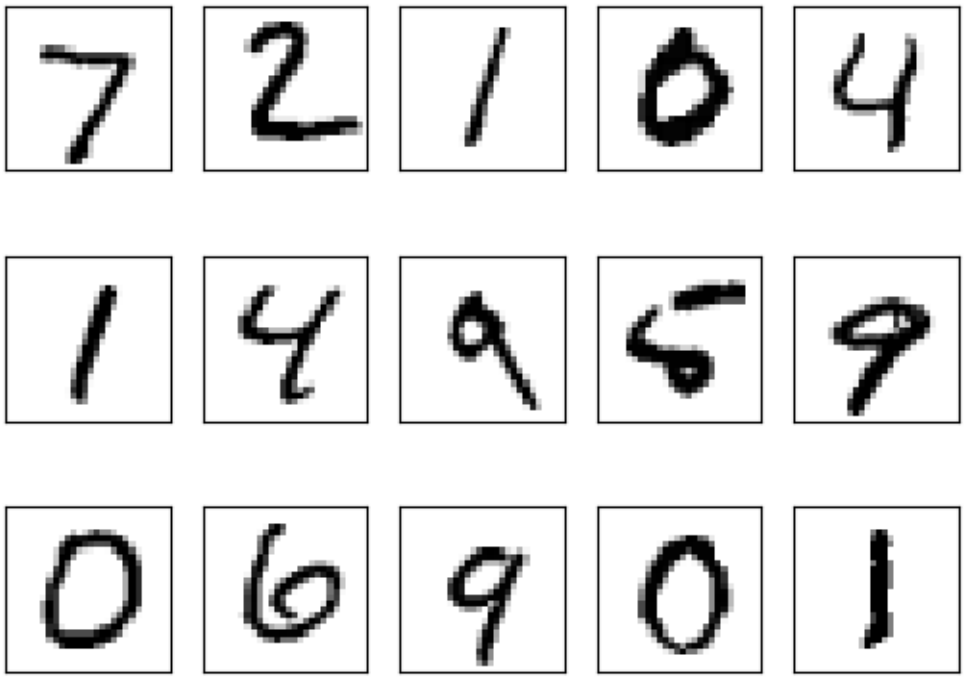
2 Problem 2

[2]:
```
import numpy as np
import warnings
warnings.filterwarnings("ignore")
import tensorflow as tf
import matplotlib.pyplot as plt
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
## function to plot images in grid
def show_images(images, rows, cols):
    for i in range(rows * cols):
        plt.subplot(rows, cols, i + 1)
        plt.imshow(images[i], cmap=plt.cm.gray_r)
```

```
        plt.xticks(())
        plt.yticks(())
    plt.show()
## Uncomment below to see a few images
print('A few example images:')
show_images(x_test, 3, 5)
```

```
2023-11-02 11:58:47.658233: I tensorflow/core/util/port.cc:110] oneDNN custom
operations are on. You may see slightly different numerical results due to
floating-point round-off errors from different computation orders. To turn them
off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2023-11-02 11:58:47.702002: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2023-11-02 11:58:48.114347: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not
find cuda drivers on your machine, GPU will not be used.
2023-11-02 11:58:48.117355: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations,
rebuild TensorFlow with the appropriate compiler flags.
2023-11-02 11:58:49.947927: W
tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warning: Could not
find TensorRT
```

```
A few example images:
```



(a) Since the MNIST dataset is comprised of 784 pixels per image, we can treat them as random vectors from $\mathbb{R}^{784}$. The means are vectors of shape 784×1 while the covarances are matrices of shape 784×784.

[3]:
```
#2b
means = np.array([np.mean(x_train[y_train==i].reshape(-1,784), axis=0) for i in␣
 ↪range(10)])
covs = np.array([np.cov(x_train[y_train==i].reshape(-1,784).T) for i in␣
 ↪range(10)])
```

[4]:
```
#2c
show_images(means.reshape(-1,28,28), 1, 10)
```

(d) The covariance, $\hat{\Sigma}$, is not always invertible. However, we know that $\hat{\Sigma}$ is a symmetric matrix described as

$$\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T$$

for any vector y, consider

$$y^T\hat{\Sigma}y = \frac{1}{n}\sum_{i=1}^{n} y^T(x_i - \mu)(x_i - \mu)^T y = \frac{1}{n}(y^T(x_i - \mu))^2 \geq 0$$

therefore, $\hat{\Sigma}$ is semi-positive definite, which also means all its eigenvalues are nonnegative. Considering $\lambda > 0$, we can see that the eigenvalues of $\hat{\Sigma} + \lambda I$ are all λ greater than a corresponding eigenvalues of $\hat{\Sigma}$. Since $\hat{\Sigma} + \lambda I$ have all positive eigenvalues, its determinant cannot be zero and therefore is invertible.

[5]:
```
#2e
def log_likelihood(x, mu, cov_inv, logabsdet):
    x = x.reshape(mu.shape) #cast to same shape
    shift = x-mu
    log_likelihood = - 0.5*logabsdet - 0.5*(shift.T@cov_inv@shift)
    return log_likelihood
```

[6]:
```
#2f
def predict(tests, lambd=1):
    output = []
    for y in range(10):
        regularizer = lambd*np.identity(784)
        mu = means[y]
        cov_reg = covs[y]+regularizer
        cov_inv = np.linalg.inv(cov_reg)
```

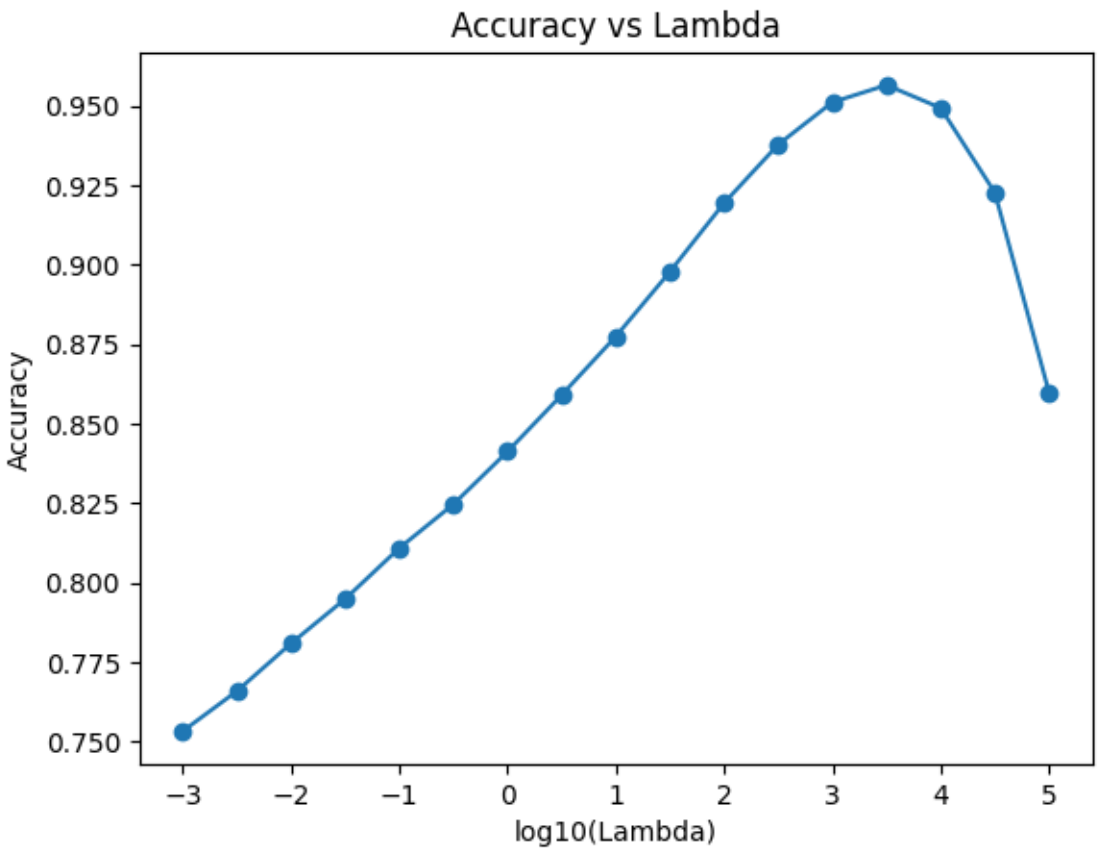
```
        _, logabsdet = np.linalg.slogdet(cov_reg)
        output.append([log_likelihood(x, mu, cov_inv, logabsdet) for x in tests␣
↪])
    return np.argmax(np.array(output), axis=0)
```

[7]:
```
y_test_pred = predict(x_test, lambd=1)
print(f"Error rate = {np.sum(y_test_pred!=y_test)/len(y_test)}")
print(f"Accuracy = {np.sum(y_test_pred==y_test)/len(y_test)}")
```

```
Error rate = 0.1588
Accuracy = 0.8412
```

[8]:
```
#2g
errors = []
log_lambdas = np.array(range(-6,11))/2
for i in log_lambdas:
    y_test_pred = predict(x_test, lambd=10**i)
    error = np.sum(y_test_pred!=y_test)/len(y_test)
    errors.append(error)
```

[9]:
```
plt.plot(log_lambdas, 1-np.array(errors))
plt.scatter(log_lambdas, 1-np.array(errors))
plt.title("Accuracy vs Lambda")
plt.xlabel("log10(Lambda)")
plt.ylabel("Accuracy")
plt.show()
```



[10]:
```
# 2h and 2i
print(f"Best lambda = 10^{log_lambdas[np.argmin(errors)]}")
print(f"Best error rate = {round(errors[np.argmin(errors)],4)}")
print(f"Best accuracy = {round(1-errors[np.argmin(errors)],4)}")
```

```
Best lambda = 10^3.5
Best error rate = 0.0435
Best accuracy = 0.9565
```