1.Successfully installed and studied Jupyter notebook.
2. (10pts) Exercise 1.3 in LFD

a) $x(t)$ is ~~misclaf~~ misclassified by $w(t)$,
i.e. $w^T(t)x(t)$ has opposite sign of $y(t)$.
Therefore, $y(t)w^T(t)x(t) < 0$.

if $y(t) = +1$ then $\text{sign}(w^T(t)x(t)) = -1$
$w^T(t)x(t) < 0$.
Hence, $y(t)w^T(t)x(t) < 0$.

if $y(t) = -1$ then $\text{sign}(w^T(t)x(t)) = +1$.
$w^T(t)x(t) > 0$
Hence, $y(t)w^T(t)x(t) < 0$.

b) Update rule.
$w(t+1) = w(t) + y(t)x(t)$

$x(t)$ is misclassified by $w(t)$ so $y(t)w^T(t)x(t) < 0$.
Multiplying by $y(t)x(t)$ on both sides

$$y(t)w^T(t+1)x(t) = y(t)w^T(t)x(t) + \underbrace{y^2(t)(x(t))^2}_{\text{always greater than } 0}$$

Thus, $y(t)w^T(t+1)x(t) > y(t)w^T(t)(x(t))$.

and also
$w(t+1)$ correctly classifies $x(t)$ so $y(t)w^T(t)x(t) > 0$

c) $y(t)w^T(t)x(t) < 0$.
$y(t)w^T(t+1)x(t) > y(t)w^T(t)x(t)$.

If $x(t)$ is correctly classified then update rule isn't applied.
If $x(t)$ is incorrect classified as -ve so $y(t) = 1$.
Thus, the $x(t) \cdot x(t)$ is positive. Thus, the boundary is moved in right direction.
Update rule increases $y(t)w^T(t)x(t)$ until it passes 0.

3. (10pts) Exercise 1.6 in LFD

a) Reinforcement learning, or supervised learning.

The input space is the set of all books, and the output space is whether a particular person will buy that book , so it can be viewed as supervised learning.If only the title of the book is known then it is not a sufficient information. Input space would contain details about a particular person's book preferences and their liking for a certain genre, and the output space would be accordingly, for example,2 books we should recommend. In this case, our training data is likely of the form:(individual 's buying history and mood; suggested book; buy)Thus, it is a reinforcement learning problem.

b) Reinforcement learning.

The input space would be a tic-tack-toe situation (i.e. for each of the 9 squares, whether it is "o", "x", or "blank"). The output space would be which move should be chosen by the current player. We would reinforce moves that ultimately led the person to win the game.

c) Supervised learning OR unsupervised learning

For supervised learning, the output space would be the set of all movie categories, and our training samples would have the movie as the input value and the category as the output value. As unsupervised learning, we would only know the movie (or its mathematically reduced description).

d) Reinforcement learning.

The input space would be a tune and some parameters describing the style of music desired. The output would be the produced music. We would reinforce on how listeners thought the music was.

e) Reinforcement learning.

The input space is details about the persons financial history. The output space is the maximum credit. We would reinforce on how much money the bank gained or lost on the person.

4. (10pts) Exercise 1.8 in LFD

$$P(\nu \leq 0.1) = P\left(\frac{red}{N} \leq 0.1\right) = P(red \leq 0.1N).$$

$N = 10.$

$$P(red \leq 0.1N) = P(red \leq 1) = P(red = 0) + P(red = 1).$$

$$\mu = 0.9 = P(red) = 1 - P(green)$$

$$P(green) = 0.1$$

$$P(red = 0) = \binom{N}{0} \times P(green)^N$$

$$= \binom{10}{0} \times 0.1^{10}$$

$$= 10^{-10}.$$

$$P(red = 1) = \binom{N}{1} \times P(green)^{N-1} \times P(red)^1$$

$$= \binom{10}{1} \times 0.1^9 \times 0.9^1$$

$$= 9 \times 10^{-9}$$

Hence, $P(\nu \leq 0.1) = 10^{-10} + 9 \times 10^{-9}$

$$= 9.1 \times 10^{-9}.$$

5. (10pts) Exercise 1.9 in LFD

Hoeffding inequality

$P[|v-\mu|>\epsilon] \leq 2e^{-2\epsilon^2 N}$ for any $\epsilon \geq 0$.

$v \leq 0.1$, $\mu = 0.9$

$v - u \leq -0.8$.

$|v-u| \geq 0.8$.

Thus, ϵ = any number less than 0.8.

$P(|v-u| >$ number less than $0.8) \leq \underbrace{2e^{-2\times 0.8^2 \times 10}}_{5.52\times 10^{-6}}$

$P(v \leq 0.1) = 9.1\times 10^{-9}$

$P[|v-\mu|>\epsilon] \leq 5.52\times 10^{-6}$

$P(v \leq 0.1) \leq P(|v-u|>\epsilon)$

6. (10pts) Problem 1.2 in LFD

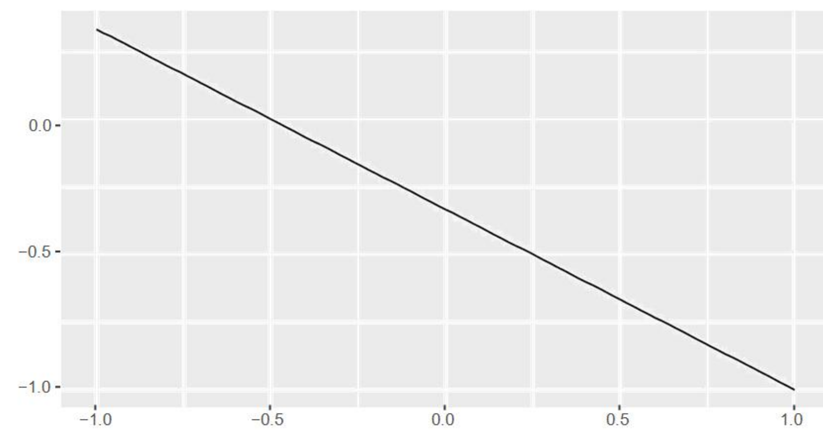
Q.6 1.2 $h(x) = \text{sign}(w^T x)$ with $w = (w_0, w_1, w_2)^T$ and $x = (1, x_1, x_2)^T$.

a) $h(x) = +1\ (-1)$, that implies the $w^T x > 0$ (resp. 0).
So, we may conclude that separation btw these two regions is the line equation $w^T x = 0$

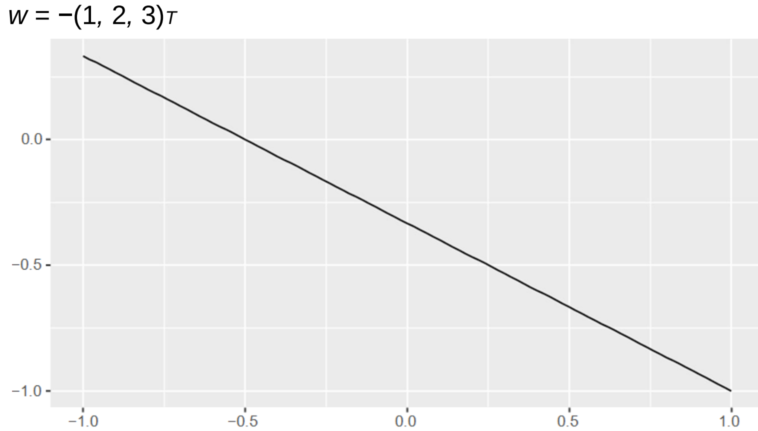
$$w_0 + w_1 x_1 + w_2 x_2 = 0 \longrightarrow \boxed{x_2 = a x_1 + b}$$

where $a = \frac{-w_1}{w_2}$ and $b = \frac{-w_0}{w_2}$.

$w = (1, 2, 3)^T$:



$w = -(1, 2, 3)^T$



The lines are identical in the graph but, the regions where $h(x) = +1$ and $h(x) = -1$ are different and in the first plot the positive region is the one above the line, and in the second plot the positive region is the one below the line.

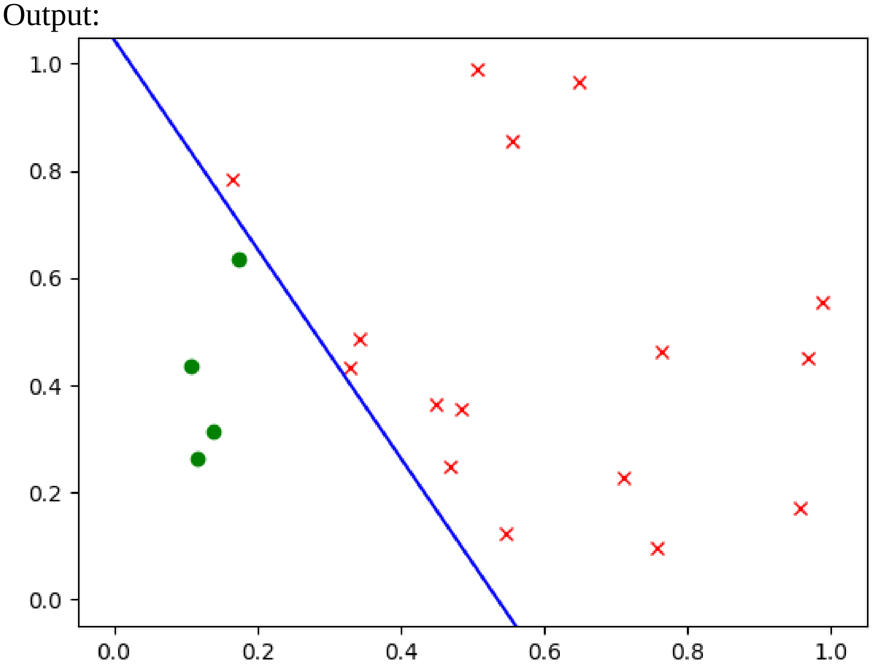
7. (20pts) Problem 1.4 (a – e) in LFD

a.

```
import matplotlib.pyplot as mplt
import numpy as np
def generateRandomLine():
    line = np.random.random([2, 2])
    x1 = line[0, 0]
    y1 = line[0, 1]
    x2 = line[1, 0]
    y2 = line[1, 1]
    k = (y1 - y2) / (x1 - x2)
    b = y1 - k * x1
    # y = kx + b => -b -kx + y = 0 slope equation
    f = np.empty(3)
    f[0] = -b
    f[1] = -k
    f[2] = 1
    return f.reshape((3, 1))

def computeYOfLine(w, x):
    # w0 + w1*x + w2*y = 0
    if w[2, 0] == 0:
        return 0
    return -(w[0, 0] + w[1, 0] * x) / w[2, 0]

dataSetLimit = 20 #m == dataSetLimit

# generate raw data using random and plot them on graph

X = np.random.random([dataSetLimit, 2])
X = np.concatenate((np.ones([dataSetLimit, 1]), X), axis=1)

f = generateRandomLine()

mplt.plot(X[:, 1], X[:, 2], "rx")
# plot the line use x = -1, 2
xs = [-1, 2]
ys = [computeYOfLine(f, x) for x in xs]
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()

# classify data

y = np.matmul(X, f)
# we must use a one dimension array to index X
greatest = (y >= 0).reshape(dataSetLimit) #greaters == greatest
least = X[~greatest] #lesses == least
greatest = X[greatest]

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
```

```
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()

for i in range(dataSetLimit):
    if y[i] < 0:
        y[i] = -1
    else:
        y[i] = 1
print(y)

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
mplt.plot(xs, ys, "b-")
ys = [computeYOfLine(weight, x) for x in xs]
mplt.plot(xs, ys, "r-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()
```

Output:



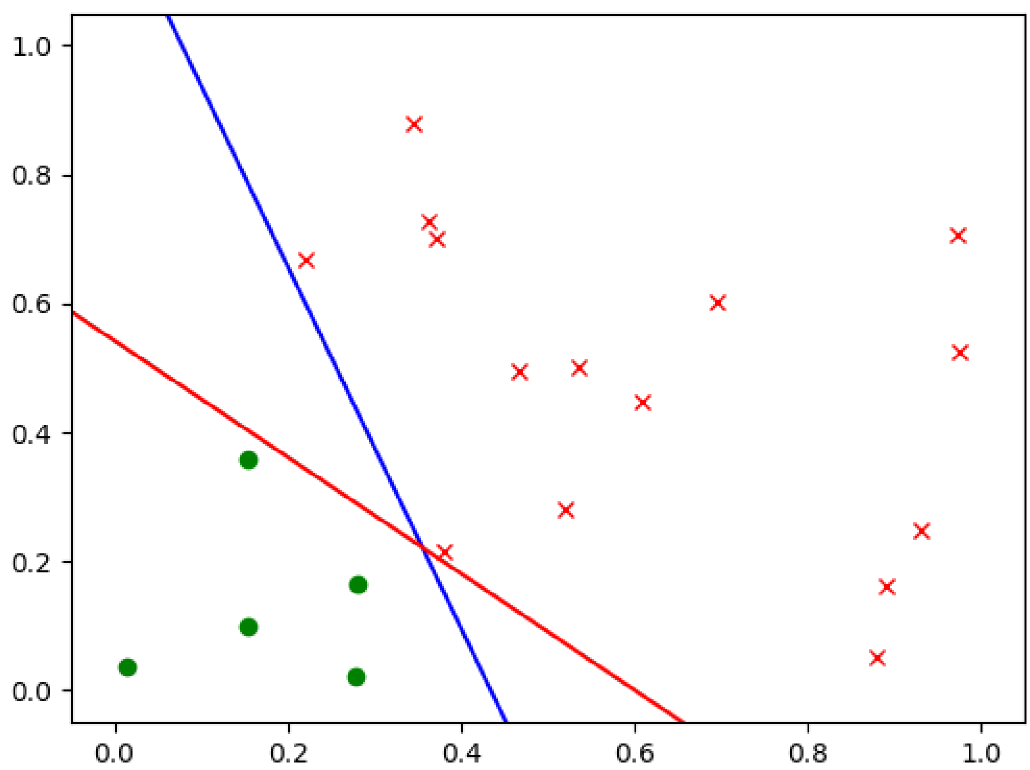
b. Here, the PLA took 2 iterations before converging. We may notice that although *g* is pretty close to *f*, they
are not quite identical.

```
import matplotlib.pyplot as mplt
import numpy as np

def generateRandomLine():
    line = np.random.random([2, 2])
    x1 = line[0, 0]
    y1 = line[0, 1]
    x2 = line[1, 0]
    y2 = line[1, 1]
    k = (y1 - y2) / (x1 - x2)
```

```python
    b = y1 - k * x1
    # y = kx + b => -b -kx + y = 0 slope equation
    f = np.empty(3)
    f[0] = -b
    f[1] = -k
    f[2] = 1
    return f.reshape((3, 1))

def computeYOfLine(w, x):
    # w0 + w1*x + w2*y = 0
    if w[2, 0] == 0:
        return 0
    return -(w[0, 0] + w[1, 0] * x) / w[2, 0]

dataSetLimit = 20 #m == dataSetLimit

# generate raw data using random and plot them on graph

X = np.random.random([dataSetLimit, 2])
X = np.concatenate((np.ones([dataSetLimit, 1]), X), axis=1)

f = generateRandomLine()

mplt.plot(X[:, 1], X[:, 2], "rx")
# plot the line use x = -1, 2
xs = [-1, 2]
ys = [computeYOfLine(f, x) for x in xs]
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()

# classify data

y = np.matmul(X, f)
# we must use a one dimension array to index X
greatest = (y >= 0).reshape(dataSetLimit) #greaters == greatest
least = X[~greatest] #lesses == least
greatest = X[greatest]

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()

for i in range(dataSetLimit):
    if y[i] < 0:
        y[i] = -1
    else:
        y[i] = 1
print(y)
```

```
# run a perceptron learning algorithm

weight = np.empty((3, 1)) #w == weights
while True:
    flag = True
    for i in range(dataSetLimit):
        x = X[i]
        h = 1
        if np.matmul(x, weight) < 0:
            h = -1
        if h != y[i]:
            weight += (y[i] * x).reshape((3, 1))
            flag = False
    if flag:
        break

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
mplt.plot(xs, ys, "b-")
ys = [computeYOfLine(weight, x) for x in xs]
mplt.plot(xs, ys, "r-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()
```

Output:

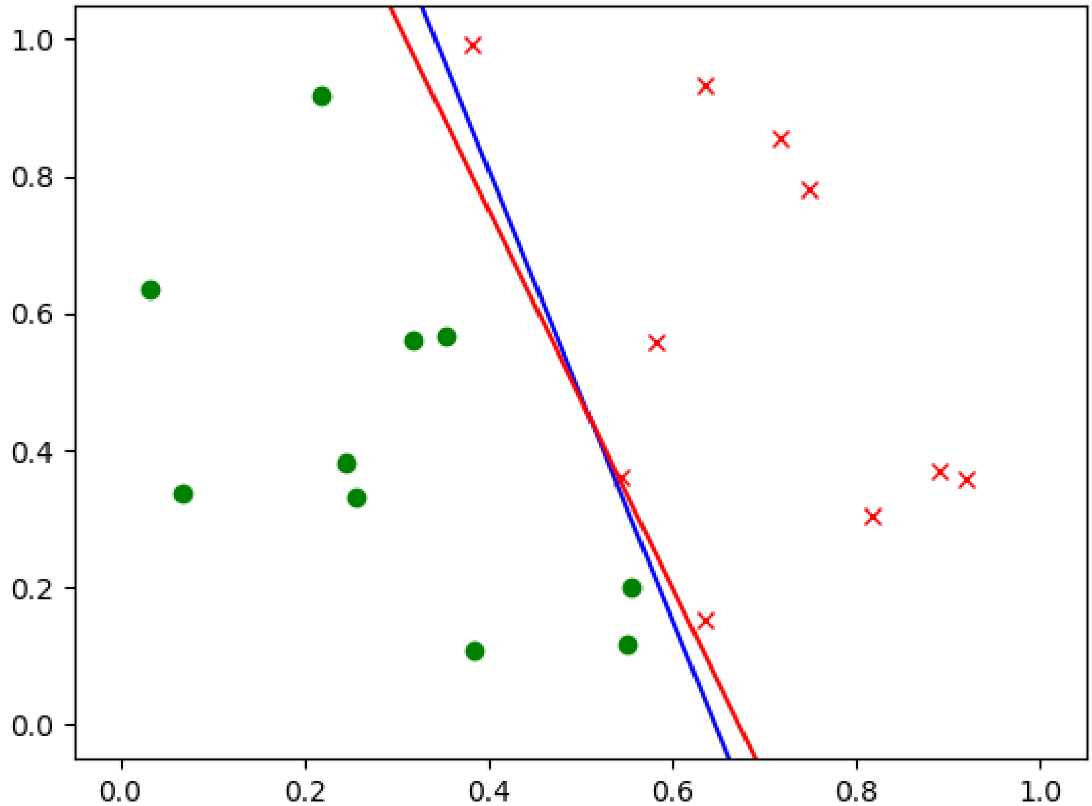


c.

```
import matplotlib.pyplot as mplt
import numpy as np

def generateRandomLine():
    line = np.random.random([2, 2])
    x1 = line[0, 0]
    y1 = line[0, 1]
    x2 = line[1, 0]
    y2 = line[1, 1]
    k = (y1 - y2) / (x1 - x2)
    b = y1 - k * x1
    # y = kx + b => -b -kx + y = 0 slope equation
    f = np.empty(3)
    f[0] = -b
    f[1] = -k
    f[2] = 1
    return f.reshape((3, 1))

def computeYOfLine(w, x):
    # w0 + w1*x + w2*y = 0
    if w[2, 0] == 0:
        return 0
    return -(w[0, 0] + w[1, 0] * x) / w[2, 0]

dataSetLimit = 20 #m == dataSetLimit

# generate raw data using random and plot them on graph

X = np.random.random([dataSetLimit, 2])
X = np.concatenate((np.ones([dataSetLimit, 1]), X), axis=1)

f = generateRandomLine()

mplt.plot(X[:, 1], X[:, 2], "rx")
# plot the line use x = -1, 2
xs = [-1, 2]
ys = [computeYOfLine(f, x) for x in xs]
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()

# classify data

y = np.matmul(X, f)
# we must use a one dimension array to index X
greatest = (y >= 0).reshape(dataSetLimit) #greaters == greatest
least = X[~greatest] #lesses == least
greatest = X[greatest]

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
```

```
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()

for i in range(dataSetLimit):
    if y[i] < 0:
        y[i] = -1
    else:
        y[i] = 1
print(y)

# run a perceptron learning algorithm

weight = np.empty((3, 1)) #w == weights
while True:
    flag = True
    for i in range(dataSetLimit):
        x = X[i]
        h = 1
        if np.matmul(x, weight) < 0:
            h = -1
        if h != y[i]:
            weight += (y[i] * x).reshape((3, 1))
            flag = False
    if flag:
        break

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
mplt.plot(xs, ys, "b-")
ys = [computeYOfLine(weight, x) for x in xs]
mplt.plot(xs, ys, "r-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()
```

Output:



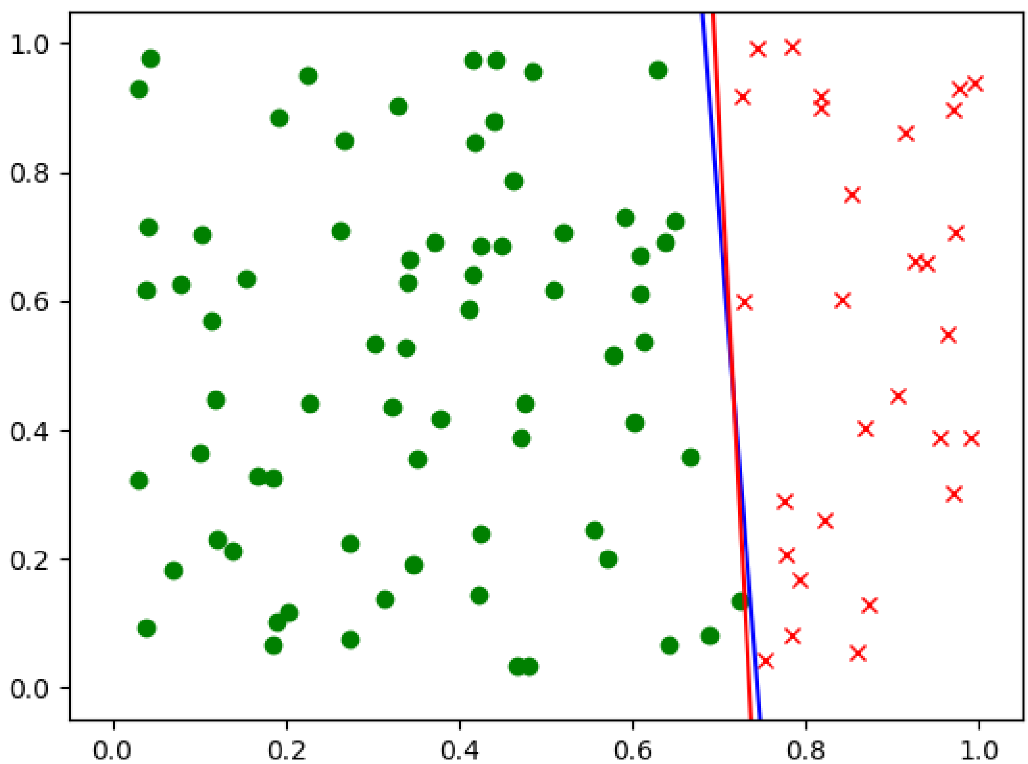
d. Here, the PLA took 17 iterations (which is greater than in (*b*) and (*c*)) before converging. We may
notice that, here *f* and *g* are very close to each other.

```
import matplotlib.pyplot as mplt
import numpy as np

def generateRandomLine():
    line = np.random.random([2, 2])
    x1 = line[0, 0]
    y1 = line[0, 1]
    x2 = line[1, 0]
    y2 = line[1, 1]
    k = (y1 - y2) / (x1 - x2)
    b = y1 - k * x1
    # y = kx + b => -b -kx + y = 0 slope equation
    f = np.empty(3)
    f[0] = -b
    f[1] = -k
    f[2] = 1
    return f.reshape((3, 1))

def computeYOfLine(w, x):
    # w0 + w1*x + w2*y = 0
    if w[2, 0] == 0:
        return 0
    return -(w[0, 0] + w[1, 0] * x) / w[2, 0]
```

```
dataSetLimit = 100 #m == dataSetLimit

# generate raw data using random and plot them on graph

X = np.random.random([dataSetLimit, 2])
X = np.concatenate((np.ones([dataSetLimit, 1]), X), axis=1)

f = generateRandomLine()

mplt.plot(X[:, 1], X[:, 2], "rx")
# plot the line use x = -1, 2
xs = [-1, 2]
ys = [computeYOfLine(f, x) for x in xs]
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()

# classify data

y = np.matmul(X, f)
# we must use a one dimension array to index X
greatest = (y >= 0).reshape(dataSetLimit) #greaters == greatest
least = X[~greatest] #lesses == least
greatest = X[greatest]

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()

for i in range(dataSetLimit):
    if y[i] < 0:
        y[i] = -1
    else:
        y[i] = 1
print(y)

# run a perceptron learning algorithm

weight = np.empty((3, 1)) #w == weights
while True:
    flag = True
    for i in range(dataSetLimit):
        x = X[i]
        h = 1
        if np.matmul(x, weight) < 0:
            h = -1
        if h != y[i]:
            weight += (y[i] * x).reshape((3, 1))
            flag = False
```

```
    if flag:
        break

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
mplt.plot(xs, ys, "b-")
ys = [computeYOfLine(weight, x) for x in xs]
mplt.plot(xs, ys, "r-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()
```

Output:



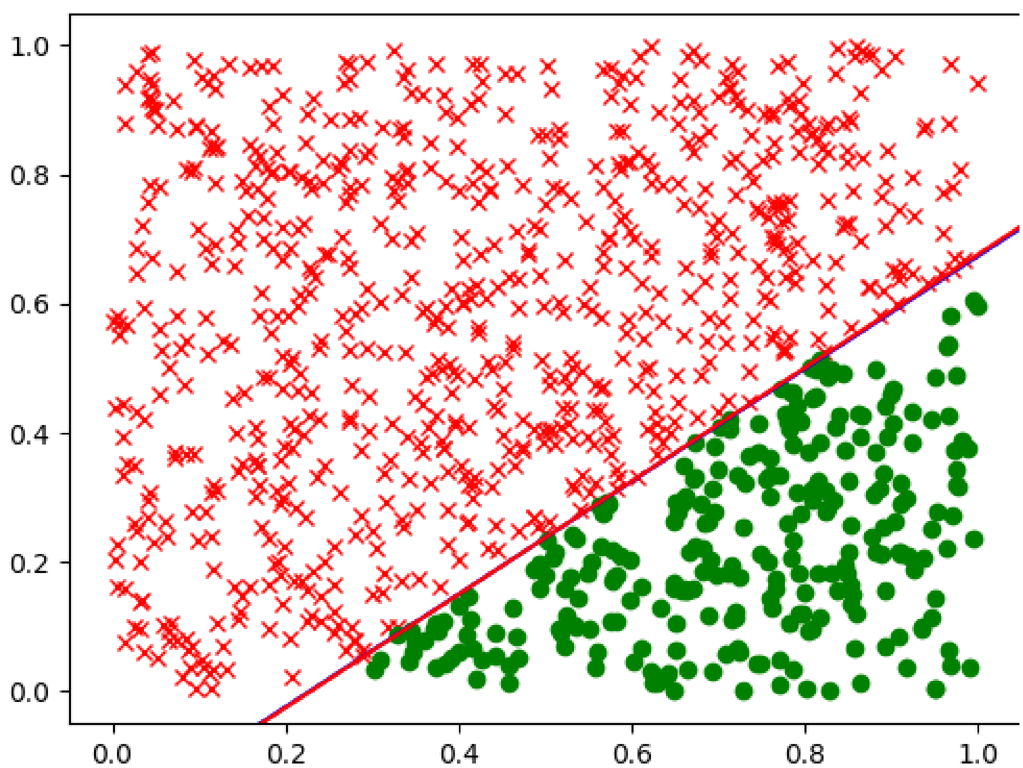
e. In this case, the PLA took 599 iterations (which is greater than in (*b*), (*c*) and (*d*)) before converging. We
may notice that, here *f* and *g* are nearly undistinguishable.

```
import matplotlib.pyplot as mplt
import numpy as np

def generateRandomLine():
    line = np.random.random([2, 2])
    x1 = line[0, 0]
    y1 = line[0, 1]
    x2 = line[1, 0]
    y2 = line[1, 1]
    k = (y1 - y2) / (x1 - x2)
    b = y1 - k * x1
    # y = kx + b => -b -kx + y = 0 slope equation
    f = np.empty(3)
    f[0] = -b
```

```
    f[1] = -k
    f[2] = 1
    return f.reshape((3, 1))

def computeYOfLine(w, x):
    # w0 + w1*x + w2*y = 0
    if w[2, 0] == 0:
        return 0
    return -(w[0, 0] + w[1, 0] * x) / w[2, 0]

dataSetLimit = 1000 #m == dataSetLimit

# generate raw data using random and plot them on graph

X = np.random.random([dataSetLimit, 2])
X = np.concatenate((np.ones([dataSetLimit, 1]), X), axis=1)
f = generateRandomLine()
mplt.plot(X[:, 1], X[:, 2], "rx")
# plot the line use x = -1, 2
xs = [-1, 2]
ys = [computeYOfLine(f, x) for x in xs]
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()

# classify data

y = np.matmul(X, f)
# we must use a one dimension array to index X
greatest = (y >= 0).reshape(dataSetLimit) #greaters == greatest
least = X[~greatest] #lesses == least
greatest = X[greatest]

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()

for i in range(dataSetLimit):
    if y[i] < 0:
        y[i] = -1
    else:
        y[i] = 1
print(y)

# run a perceptron learning algorithm

weight = np.empty((3, 1)) #w == weights
while True:
    flag = True
    for i in range(dataSetLimit):
```

```
            x = X[i]
            h = 1
            if np.matmul(x, weight) < 0:
                h = -1
            if h != y[i]:
                weight += (y[i] * x).reshape((3, 1))
                flag = False
        if flag:
            break

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
mplt.plot(xs, ys, "b-")
ys = [computeYOfLine(weight, x) for x in xs]
mplt.plot(xs, ys, "r-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()
```

Output:



8. (20pts) Problem 1.5 in LFD

a. *Used a value of eta=5 instead of 100 to simplify the values computed and trained 100 and tested 10000 data.*

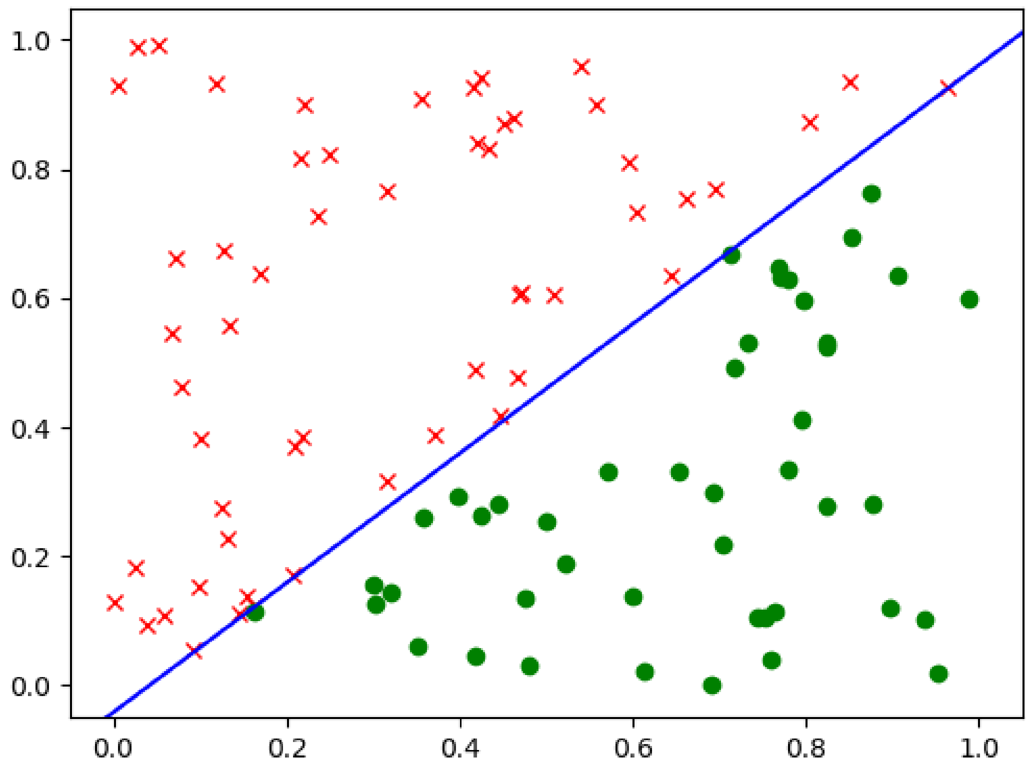
```
import matplotlib.pyplot as mplt
import numpy as np
def generateLine():
    line = np.random.random([2, 2])
    x1 = line[0, 0]
    y1 = line[0, 1]
    x2 = line[1, 0]
    y2 = line[1, 1]
    k = (y1 - y2) / (x1 - x2)
    b = y1 - k * x1
    # y = kx + b => -b -kx + y = 0 slope equation
    f = np.empty(3)
    f[0] = -b
    f[1] = -k
    f[2] = 1
    return f.reshape((3, 1))

def computeYOfLine(w, x):
    # w0 + w1*x + w2*y = 0
    if w[2, 0] == 0:

        return 0
    return -(w[0, 0] + w[1, 0] * x) / w[2, 0]

def myPlot(data,weight,label,style,xs,ys,limit):
    xt=data

    y = np.matmul(data, f)
    # we must use a one dimension array to index xt
    greatest = (y >= 0).reshape(limit) #greaters == greatest
    least = xt[~greatest] #lesses == least
    greatest = xt[greatest]

    mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
    mplt.plot(least[:, 1], least[:, 2], 'go')
    mplt.plot(xs, ys, "b-",label="original",linewidth=2)

    ys = [computeYOfLine(weight, xt) for xt in xs]
    mplt.plot(xs, ys, style,label=label)
    mplt.xlim((-0.05, 1.05))
    mplt.ylim((-0.05, 1.05))
    mplt.legend(loc="upper left")
    mplt.show()

# generate raw data using random and plot them on graph

x_ran_train_limit = 100
y = np.random.random([x_ran_train_limit, 2])
x_ran_train = np.concatenate((np.ones([x_ran_train_limit, 1]), y),
axis=1)
```

```
x_ran_test_limit = 10000
y = np.random.random([x_ran_test_limit, 2])
x_ran_test = np.concatenate((np.ones([x_ran_test_limit, 1]), y), axis=1)

X=x_ran_train
dataSetLimit = x_ran_train_limit

f = generateLine()

mplt.plot(X[:, 1], X[:, 2], "rx")
# plot the line use x = -1, 2
xs = [-1, 2]
ys = [computeYOfLine(f, x) for x in xs]
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()

y = np.matmul(X, f)
# we must use a one dimension array to index X
greatest = (y >= 0).reshape(dataSetLimit) #greaters == greatest
least = X[~greatest] #lesses == least
greatest = X[greatest]

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.show()

for i in range(dataSetLimit):
    if y[i] < 0:
        y[i] = -1
    else:
        y[i] = 1

# run a perceptron learning algorithm
eta = 2
weight = np.empty((3, 1)) #w == weights

while True:
    flag = True
    for i in range(dataSetLimit):
        x = X[i]
        h = 1
        s=np.matmul(x,weight)
        if np.matmul(x, weight) < 0:
            h = -1
        if h != y[i]:
            weight += (eta*(np.add(y[i],-s))* x).reshape((3, 1))
            flag = False
    if flag:
```

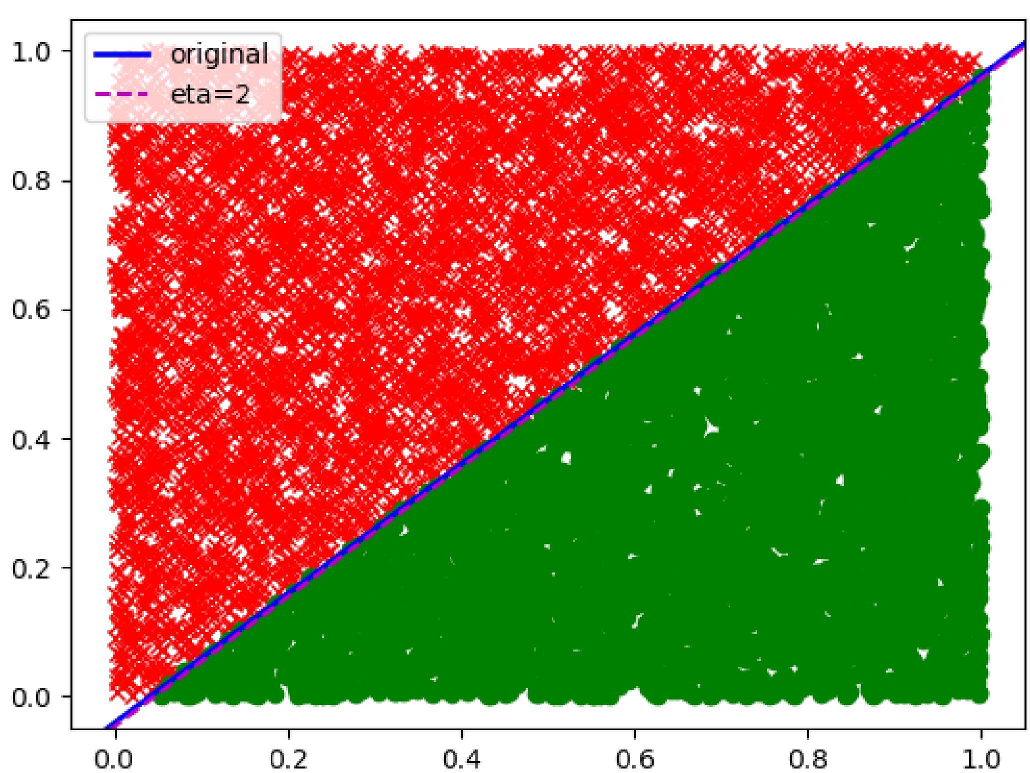
```
        break

# run a perceptron learning algorithm
eta1 = 1
weight1 = np.empty((3, 1)) #w == weights

while True:
    flag = True
    for i in range(dataSetLimit):
        x = X[i]
        h = 1
        s=np.matmul(x,weight1)
        if np.matmul(x, weight1) < 0:
            h = -1
        if h != y[i]:
            weight1 += (eta1*(np.add(y[i],-s))* x).reshape((3, 1))
            flag = False
    if flag:
        break

eta2 = 0.001
weight2 = np.empty((3, 1)) #w == weights

while True:
    flag = True
    for i in range(dataSetLimit):
        x = X[i]
        h = 1
        s=np.matmul(x,weight2)
        if np.matmul(x, weight2) < 0:
            h = -1
        if h != y[i]:
            weight2 += (eta2*(np.add(y[i],-s))* x).reshape((3, 1))
            flag = False
    if flag:
        break

eta3 = 0.0001
weight3 = np.empty((3, 1))

while True:
    flag = True
    for i in range(dataSetLimit):
        x = X[i]
        h = 1
        s=np.matmul(x,weight3)
        if np.matmul(x, weight3) < 0:
            h = -1
        if h != y[i]:
            weight3 += (eta3*(np.add(y[i],-s))* x).reshape((3, 1))
            flag = False
    if flag:
        break
```

```
xt=x_ran_test
limit=x_ran_test_limit
myPlot(data=xt,limit=limit,weight=weight,label="eta=2",style="m--
",xs=xs,ys=ys)
myPlot(data=xt,limit=limit,weight=weight1,label="eta=1",style="y--
",xs=xs,ys=ys)
myPlot(data=xt,limit=limit,weight=weight2,label="eta=0.001",style="k--
",xs=xs,ys=ys)
myPlot(data=xt,limit=limit,weight=weight3,label="eta=0.0001",style="c--
",xs=xs,ys=ys)

xt=x_ran_test
limit=x_ran_test_limit

y = np.matmul(xt, f)
# we must use a one dimension array to index xt
greatest = (y >= 0).reshape(limit) #greaters == greatest
least = xt[~greatest] #lesses == least
greatest = xt[greatest]

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
mplt.plot(xs, ys, "b-",label="original",linewidth=2)

ys = [computeYOfLine(weight, xt) for xt in xs]
mplt.plot(xs, ys, "m--",label="eta=2")

ys = [computeYOfLine(weight1, xt) for xt in xs]
mplt.plot(xs, ys, "y--",label="eta=1")


ys = [computeYOfLine(weight2, xt) for xt in xs]
mplt.plot(xs, ys, "k--",label="eta=0.001")

ys = [computeYOfLine(weight3, xt) for xt in xs]
mplt.plot(xs, ys, "c--",label="eta=0.0001")

mplt.xlim((-0.05, 1.05))
mplt.ylim((-0.05, 1.05))
mplt.legend(loc="upper left")
mplt.show()
```

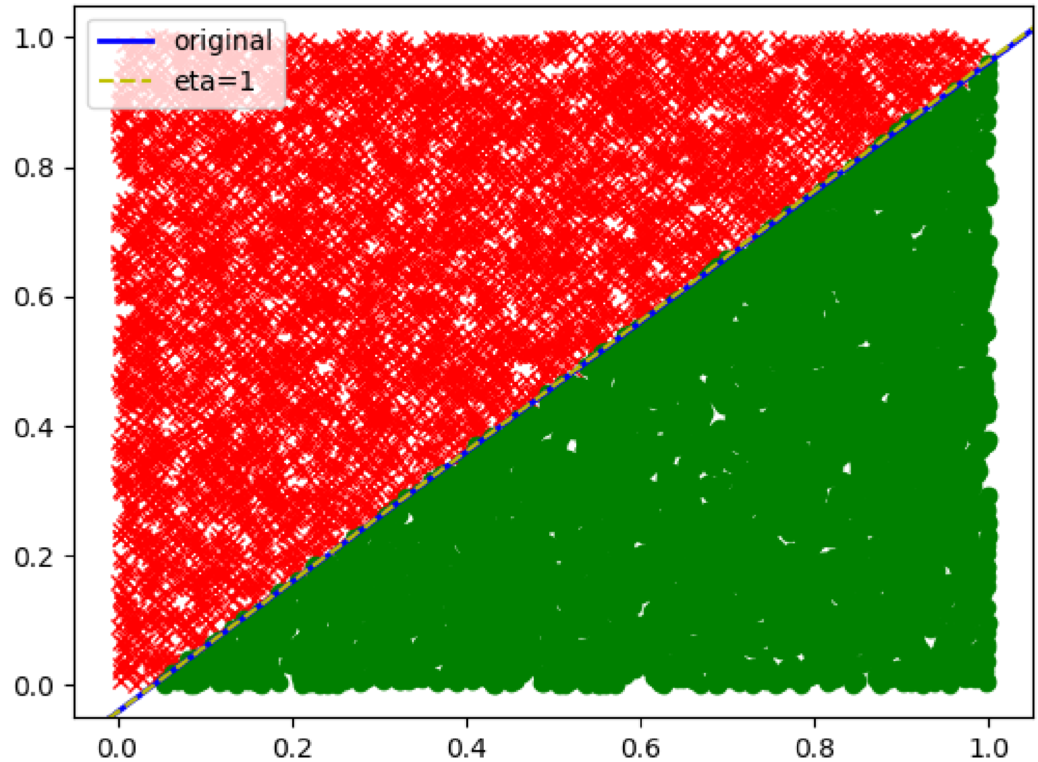
Output:
When training 100 data set:



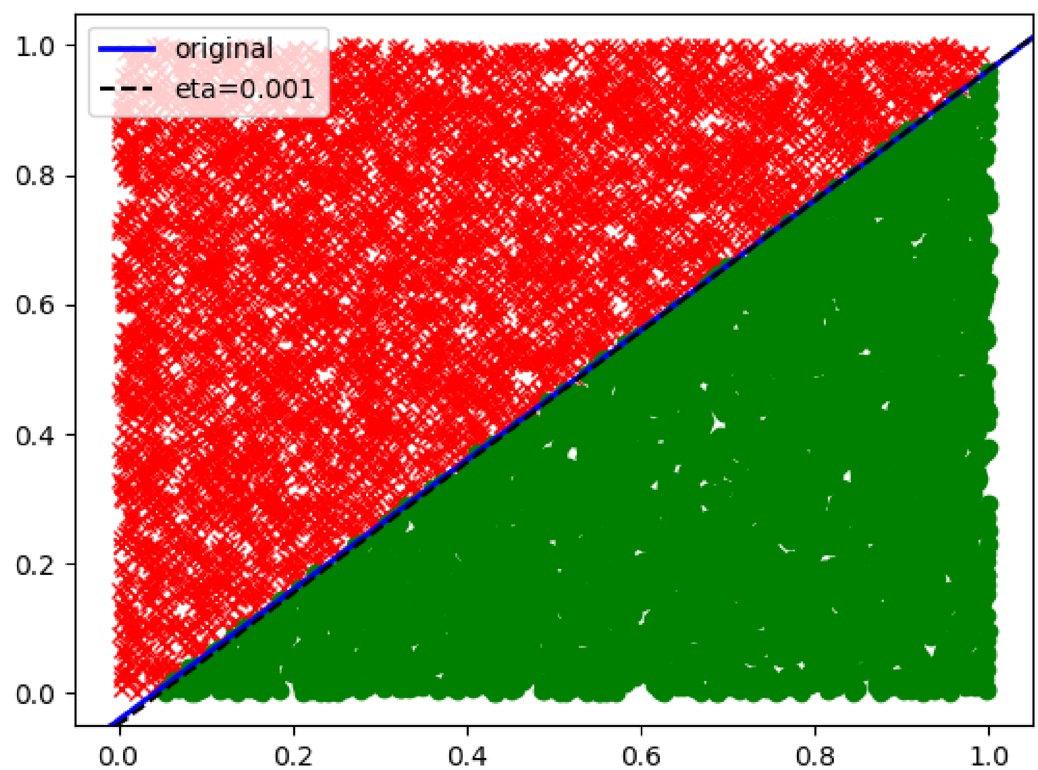
After testing it on 10000 data set with eta= 2 :



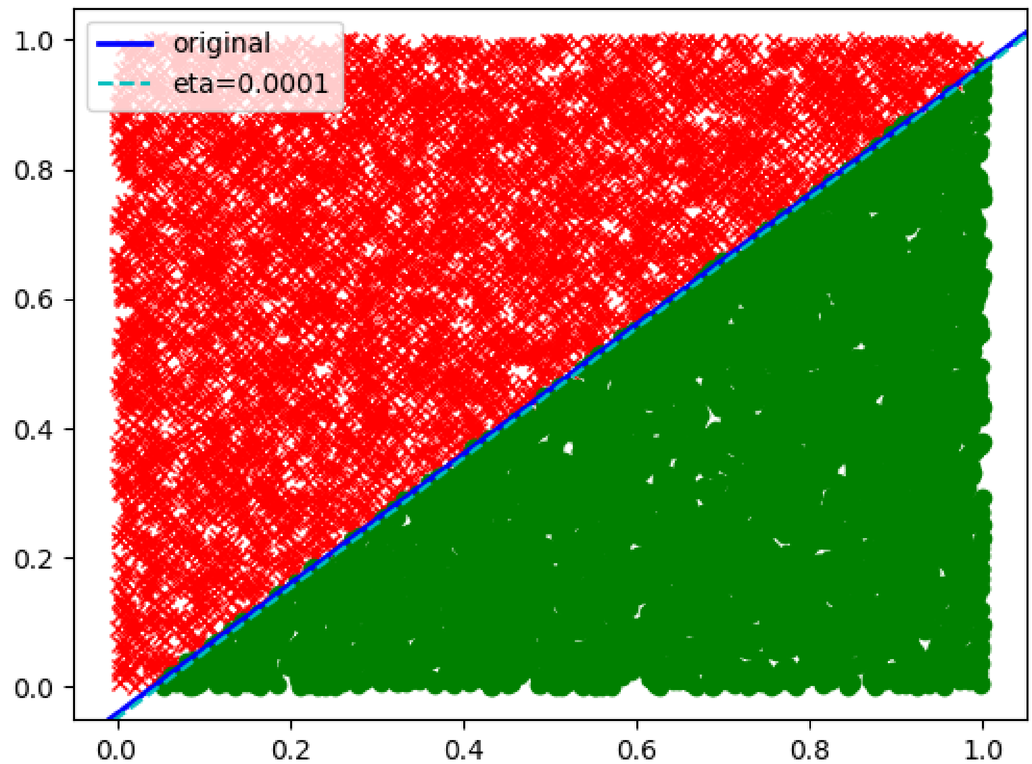
After testing it on 10000 data set with eta= 1:



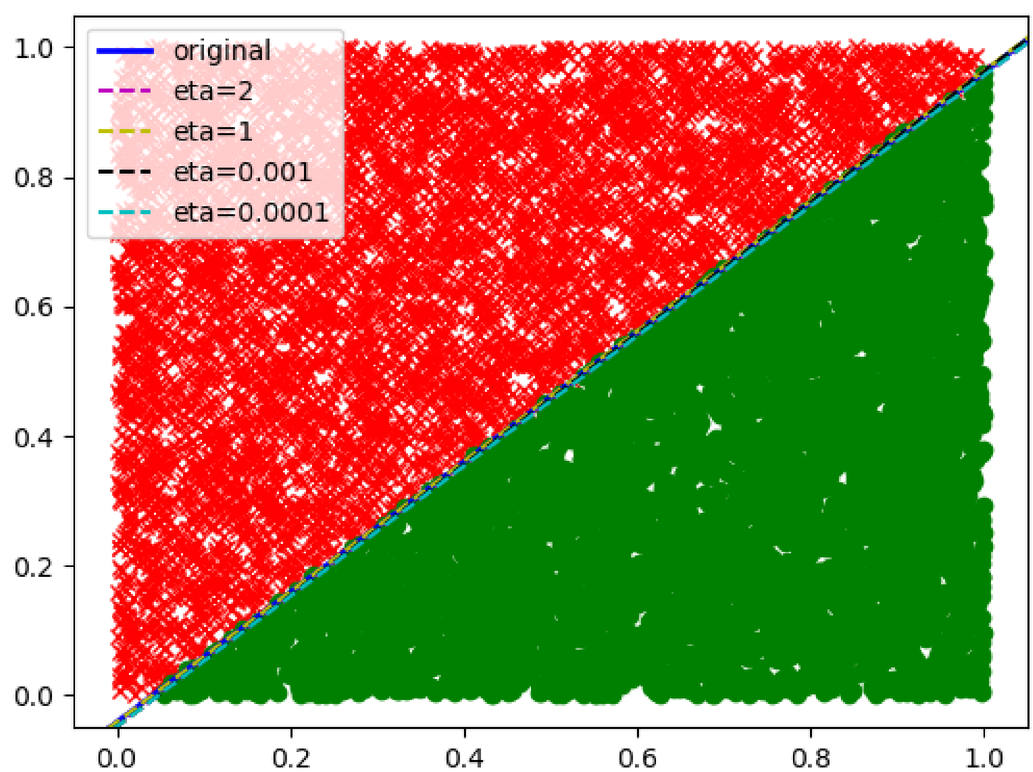
After testing it on 10000 data set with eta= 0.001:



After testing it on 10000 data set with eta= 0.0001:



After testing it on 10000 data set and comparing with all eta:



(b-e)

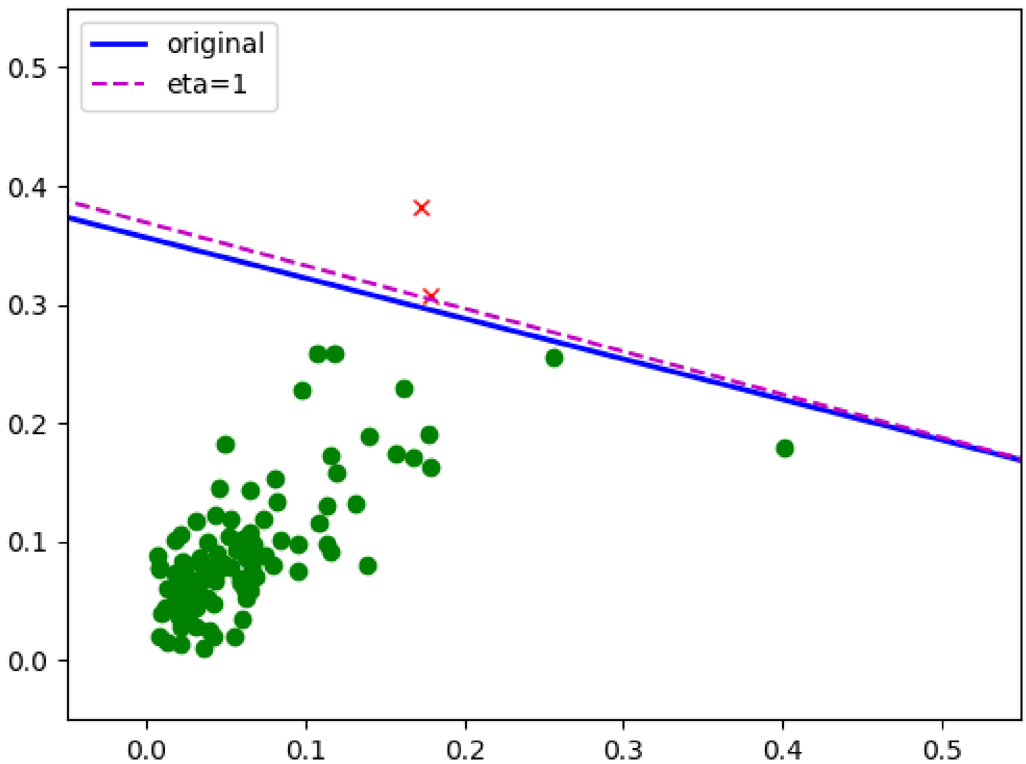
```
import matplotlib.pyplot as mplt
import numpy as np
import traintest as tt

def generateLine():
    line = np.random.random([2, 2])
    x1 = line[0, 0]
    y1 = line[0, 1]
    x2 = line[1, 0]
    y2 = line[1, 1]
    k = (y1 - y2) / (x1 - x2)
    b = y1 - k * x1
    # y = kx + b => -b -kx + y = 0 slope equation
    f = np.empty(3)
    f[0] = -0.35623063
    f[1] = 0.3409135
    f[2] = 1
    return f.reshape((3, 1))

def computeYOfLine(w, x):
    # w0 + w1*x + w2*y = 0
    if w[2, 0] == 0:

        return 0
    return -(w[0, 0] + w[1, 0] * x) / w[2, 0]

def myPlot(data,weight,label,style,xs,ys):
    xt=data
    mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
    mplt.plot(least[:, 1], least[:, 2], 'go')
    mplt.plot(xs, ys, "b-",label="original",linewidth=2)

    ys = [computeYOfLine(weight, xt) for xt in xs]
    mplt.plot(xs, ys, style,label=label)
    mplt.xlim((-0.05, 0.55))
    mplt.ylim((-0.05, 0.55))
    mplt.legend(loc="upper left")
    mplt.show()

dataSetLimit = 100 #m == dataSetLimit

# generate raw data using random and plot them on graph

# X = np.random.random([dataSetLimit, 2])
# X = np.concatenate((np.ones([dataSetLimit, 1]), X), axis=1)
X=tt.X_train

f = generateLine()
```

```
mplt.plot(X[:, 1], X[:, 2], "rx")
# plot the line use x = -1, 2
xs = [-1, 2]
ys = [computeYOfLine(f, x) for x in xs]
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 0.55))
mplt.ylim((-0.05, 0.55))
mplt.show()

# classify data

y = np.matmul(X, f)
# we must use a one dimension array to index X
greatest = (y >= 0).reshape(dataSetLimit) #greaters == greatest
least = X[~greatest] #lesses == least
greatest = X[greatest]

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
mplt.plot(xs, ys, "b-")
mplt.xlim((-0.05, 0.55))
mplt.ylim((-0.05, 0.55))
mplt.show()

for i in range(dataSetLimit):
    if y[i] < 0:
        y[i] = -1
    else:
        y[i] = 1

# run a perceptron learning algorithm
eta = 1
weight = np.empty((3, 1)) #w == weights

while True:
    flag = True
    for i in range(dataSetLimit):
        x = X[i]
        h = 1
        s=np.matmul(x,weight)
        if np.matmul(x, weight) < 0:
            h = -1
        if h != y[i]:
            weight += (eta*(np.add(y[i],-s))* x).reshape((3, 1))
            flag = False
    if flag:
        break

eta2 = 0.001
weight2 = np.empty((3, 1)) #w == weights

while True:
    flag = True
    for i in range(dataSetLimit):
```

```
        x = X[i]
        h = 1
        s=np.matmul(x,weight2)
        if np.matmul(x, weight2) < 0:
            h = -1
        if h != y[i]:
            weight2 += (eta2*(np.add(y[i],-s))* x).reshape((3, 1))
            flag = False
    if flag:
        break

eta3 = 0.0001
weight3 = np.empty((3, 1))

while True:
    flag = True
    for i in range(dataSetLimit):
        x = X[i]
        h = 1
        s=np.matmul(x,weight3)
        if np.matmul(x, weight3) < 0:
            h = -1
        if h != y[i]:
            weight3 += (eta3*(np.add(y[i],-s))* x).reshape((3, 1))
            flag = False
    if flag:
        break

xt=tt.X_test
myPlot(data=xt,weight=weight,label="eta=1",style="m--",xs=xs,ys=ys)
myPlot(data=xt,weight=weight2,label="eta=0.001",style="k--",xs=xs,ys=ys)
myPlot(data=xt,weight=weight3,label="eta=0.0001",style="c--
",xs=xs,ys=ys)

mplt.plot(greatest[:, 1], greatest[:, 2], 'rx')
mplt.plot(least[:, 1], least[:, 2], 'go')
mplt.plot(xs, ys, "b-",label="original",linewidth=2)

ys = [computeYOfLine(weight, xt) for xt in xs]
mplt.plot(xs, ys, "m--",label="eta=1")

ys = [computeYOfLine(weight2, xt) for xt in xs]
mplt.plot(xs, ys, "k--",label="eta=0.001")

ys = [computeYOfLine(weight3, xt) for xt in xs]
mplt.plot(xs, ys, "c--",label="eta=0.0001")

mplt.xlim((-0.05, 0.55))
mplt.ylim((-0.05, 0.55))
mplt.legend(loc="upper left")
mplt.show()
```

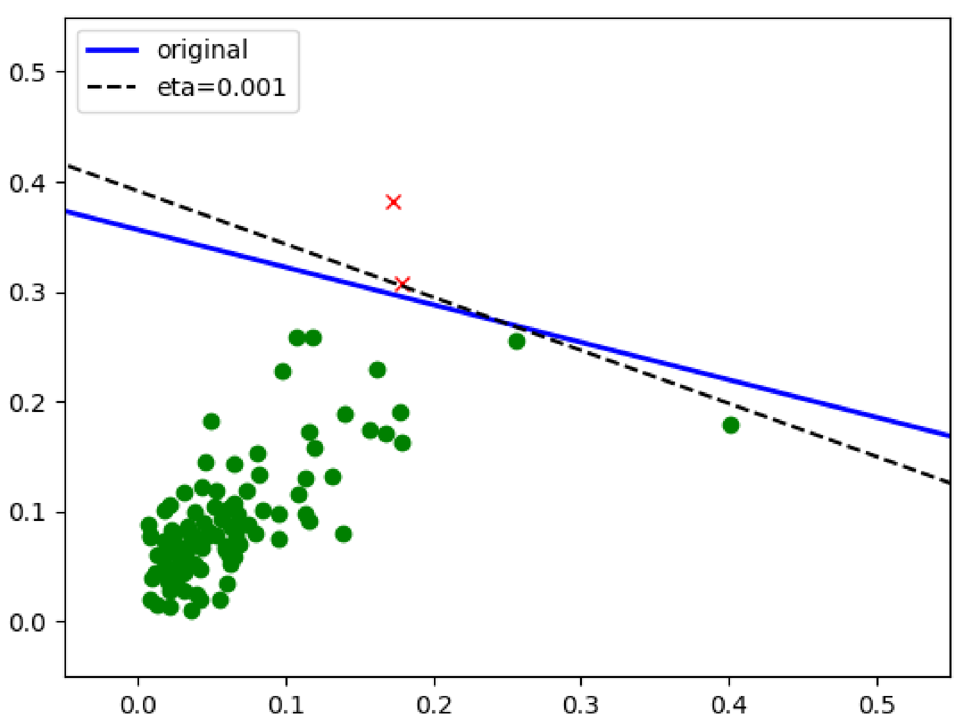
Output:

b.



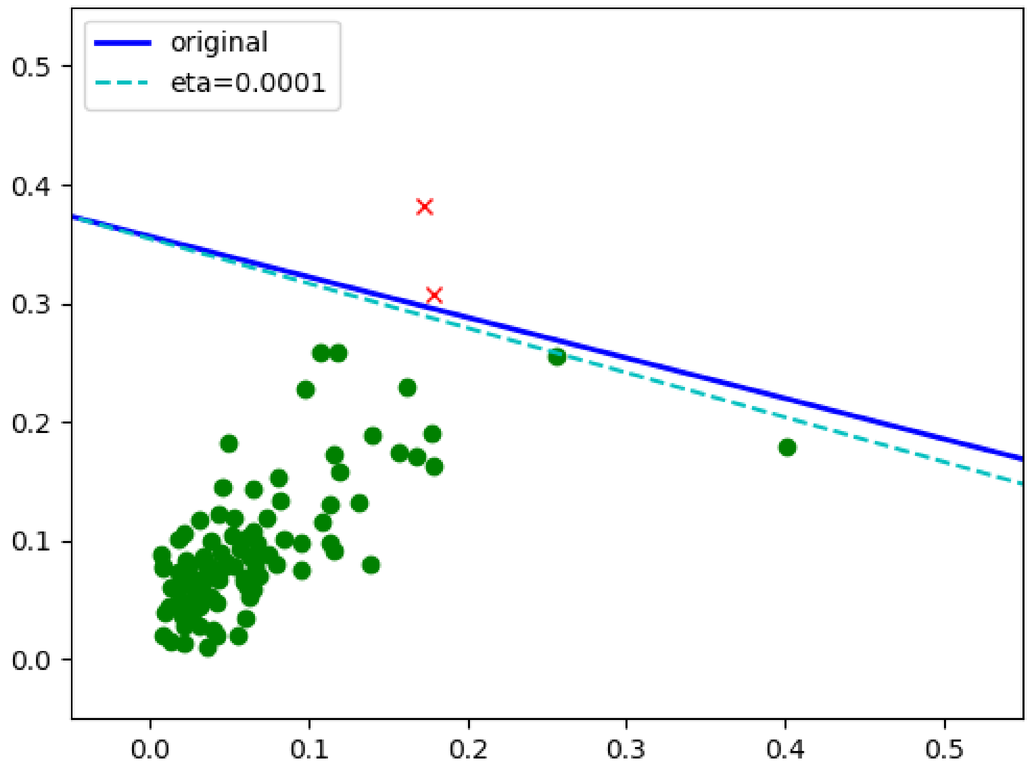
The classification error rate has decreased.

c.



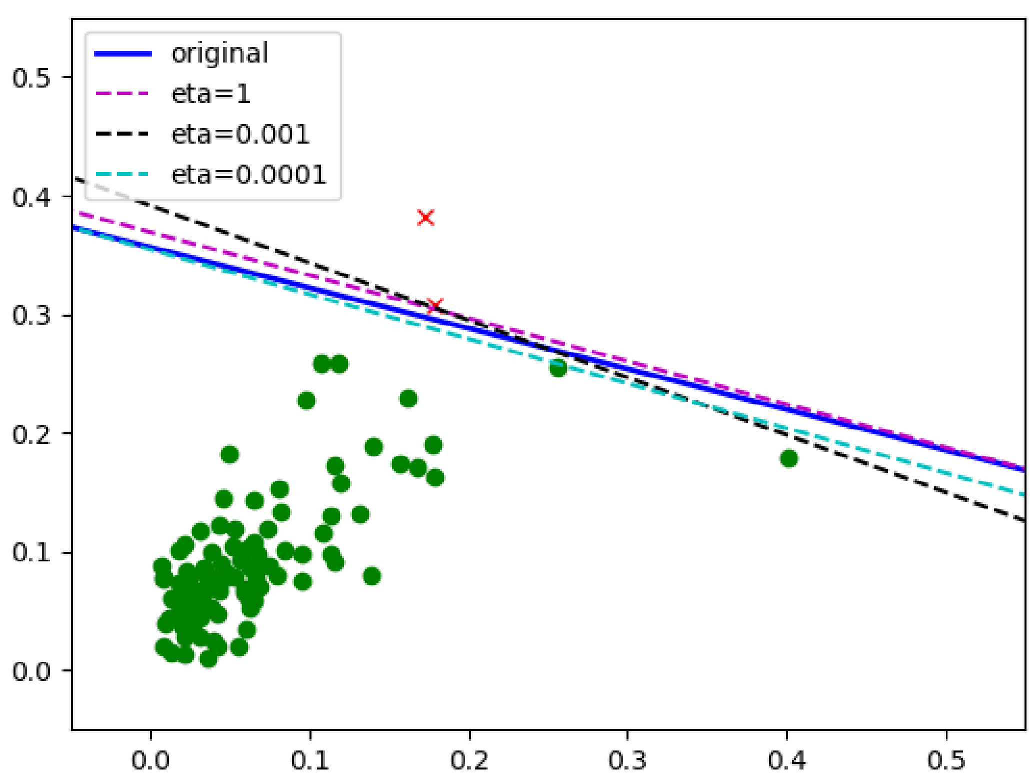
We may see that the classification error rate has now increased.

d.



We may see that the classification error rate has now increased

e.



From the results it is clear that the minimum classification error rate on the test set is actually 1.

9. (10pts) Problem 1.11 in LFD

For supermarket.

$$E_{in}^{s}(h) = \frac{1}{N}\sum_{n=1}^{N} e(h(x_n), f(x_n))$$

$$= \frac{1}{N}\left[\sum_{y_n=1} e(h(x_n),1) + \sum_{y_n=-1} e(h(x_n),-1)\right]$$

$$= \frac{1}{N}\left[\sum_{y_n=1} 10\cdot[[h(x_n)\neq 1]] + \sum_{y_n=-1} [[h(x_n)\neq -1]]\right]$$

For CIA.

$$E_{in}^{c}(h) = \frac{1}{N}\sum_{n=1}^{N} e(h(x_n), f(x_n))$$

$$= \frac{1}{N}\left[\sum_{y_n=1} e(h(x_n),1) + \sum_{y_n=-1} e(h(x_n),-1)\right]$$

$$= \frac{1}{N}\left[\sum_{y_n=1} [[h(x_n)\neq 1]] + \sum_{y_n=-1} 1000\cdot[[h(x_n)\neq -1]]\right]$$

e is defined according to risk matrices.
Pointwise error in CIA would be.

$$e(h(x_n), y_n) = \begin{cases} 1000 & \text{if } h(x_n)=1 \text{ \& } y_n=-1 \\ 1 & \text{if } h(x_n)=-1 \text{ and } y_n=1. \\ 0 & \text{otherwise.} \end{cases}$$