Assignment - 3 Submission

Name: GOVIND AJITH KUMAR
VID: 11669948 8

(2) Exercise 3.4

$$y = w^{*T} x + \epsilon$$

ϵ = noise term with zero mean and σ^2 variance

to prove:

$$E_D[E_{in}(w_{lin})] = \sigma^2\left(1 - \frac{d+1}{N}\right)$$

(a) to prove that in sample estimate of y is given by $\hat{y} = Xw^* + H\epsilon$

Ans: We know that $y = w^{*T} x + \epsilon$ — (1)

We also know that $\hat{y} = Hy$ — (2)

Putting equation (1) in (2):

$$\hat{y} = H(w^{*T} x + E)$$

$$= Hw^{*T} x + HE$$

But from equation 3.6 in LFD book:

$$H = X(X^T X)^{-1} X^T$$

$$\Rightarrow \hat{y} = X(X^T X)^{-1} X^T w^{*T} x + HE$$

expanding....

$$\hat{y} = (X X^{-1} (X^T)^{-1} X^T) w^{*T} X + H\epsilon$$

$$\hat{y} = (I (X^T)^{-1} X^T) w^{*T} X + H\epsilon$$

$$\hat{y} = (I)(I)\, w^{*T} X + H\epsilon$$

hence: $\boxed{y = X w^* + H\epsilon}$

Reference: Class Notes

(b) to prove: $\hat{y} - y$ can be expressed by a matrix times ϵ.

Ans: $\hat{y} = X w^* + H\epsilon$

but $y = w^{*T} x + \epsilon$

$$\hat{y} - y = H\epsilon - \epsilon$$

$$= (H - I)\epsilon$$

$\Rightarrow$ the matrix is $H - I$.

Here: $H = X(X^TX)^{-1}X^T$

(c) Ans:

We know: $E_{in}(w_{lin}) = \frac{1}{N} E^TE$

But $E = \hat{y} - y$

$\Rightarrow E_{in}(w_{lin}) = \frac{1}{N}(\hat{y}-y)^T(\hat{y}-y)$

$= \frac{1}{N}(\epsilon^T(H-I)^T(H-I)\epsilon)$

But $(H-I)^T(H-I) = I-H$

↳ from exercise 3.2

$\therefore$

$E_{in}(w_{lin}) = \frac{1}{N}(\epsilon^T(H-I)^T(H-I)\epsilon)$

$= \frac{1}{N}\epsilon^T(I-H)\epsilon$

$$= \frac{1}{N}\epsilon^T I \epsilon - \frac{1}{N}\epsilon^T H \epsilon$$

$$= \frac{1}{N}\epsilon^T \epsilon - \frac{1}{N}\epsilon^T H \epsilon$$

(d) to prove that:

$$E_D[E_{in}(w_{lin})] = \sigma^2\left(1 - \frac{d+1}{N}\right)$$

Ans:

$$E_D[E_{in}(w_{lin})] = \frac{1}{N} E_D[\epsilon^T (I-H)\epsilon]$$

$$= \frac{1}{N}\left(E_D[\epsilon^T \epsilon] - E_D[\epsilon^T H \epsilon]\right)$$

Here:

$$\frac{-1}{N} E_D(\epsilon^T H \epsilon)$$

$$= \frac{-1}{N}\left\{ \sum_{i=1}^{N} H_{ii}\epsilon_i^2 + \sum_{i,j\in\{1,N\}} H \epsilon_i \epsilon_j \right\}$$

Here:

$$-\frac{1}{N} E_D\left[\sum_{i=1}^{N} H_{ii}\epsilon_i^2\right] = -\text{trace}(H)\sigma^2$$
$$= -(d+1)\sigma^2$$

This is because:

$$E_D(\epsilon^T H \epsilon) = E_D\left[\sum_{i=1}^{N} H_{ii}\epsilon_i^2 + \sum_{i\neq j}^{N} H_{ij}\epsilon_i\epsilon_j\right]$$
$$= \text{trace}(H)\sigma^2 + 0 = \underline{(d+1)\sigma^2}$$

Now,

$$E_D\left[E_{in}(w_{lin})\right] = N\sigma^2 + \left(-\frac{(d+1)\sigma^2}{N}\right)$$

This is because:

$$E_D[\epsilon^T\epsilon] = E_D[\epsilon_1^2 + \epsilon_2^2 \cdots \epsilon_N^2]$$
$$= E_D\epsilon_1^2 + E_D\epsilon_2^2 \cdots\cdots E_D\epsilon_N^2$$
$$= N\sigma^2$$

$$= \sigma^2\left(1 - \frac{d+1}{N}\right)$$

(e) to prove:

$$E_{D,\epsilon'}\left[E_{test}(w_{lin})\right] = \sigma^2\left(1 + \frac{d+1}{N}\right)$$

Ans: We know that training dataset is represented by

$$y = Xw^* + \epsilon$$

for test dataset: $y' = Xw^* + \epsilon'$

$$E_{test}(w_{lin}) = \|\hat{y} - y\| \quad \text{---- (1)}$$

But we know that:

$$\hat{y} - y = H\epsilon - I\epsilon' \quad \text{---- (2)}$$

Combining (1) and (2)

$$\|\hat{y} - y\| = (H\epsilon - I\epsilon')^T (H\epsilon - I\epsilon')$$

$$= (\epsilon^T H^T - \epsilon'^T)(H\epsilon - \epsilon')$$

$$= \epsilon^T H \epsilon - \epsilon^T H \epsilon' - \epsilon'^T H \epsilon + \epsilon'^T \epsilon' \quad \text{---- (3)}$$

Now,

$$E_D[E_{test}(w_{lin})] = E_D[\|\hat{y} - y\|]$$

But we know:

$$E_D(\epsilon^T H \epsilon) = \sigma^2\left(\frac{d+1}{N}\right) \quad \text{---- (4)}$$

We also know that:

$$E_D(\epsilon'^T \epsilon') = \sigma^2 \quad \text{——— (5)}$$

Additionally, the terms 2 and 3 in equation (4) will be equated to 0. This is because ϵ and ϵ' are independent.

Hence, $E_D(\epsilon^T H \epsilon') = 0$ — (6)

$$E_D(\epsilon'^T H \epsilon) = 0 \quad \text{——— (7)}$$

Hence by adding (4), (5), (6) & (7) we get:

$$E_{D,\epsilon'}\left[E_{test}(w_{lin})\right] = \sigma^2 + 0 + 0 + \sigma^2\frac{(d+1)}{N}$$

$$= \sigma^2\left(1 + \frac{d+1}{N}\right)$$

Hence, proved.

Reference: LFD Textbook

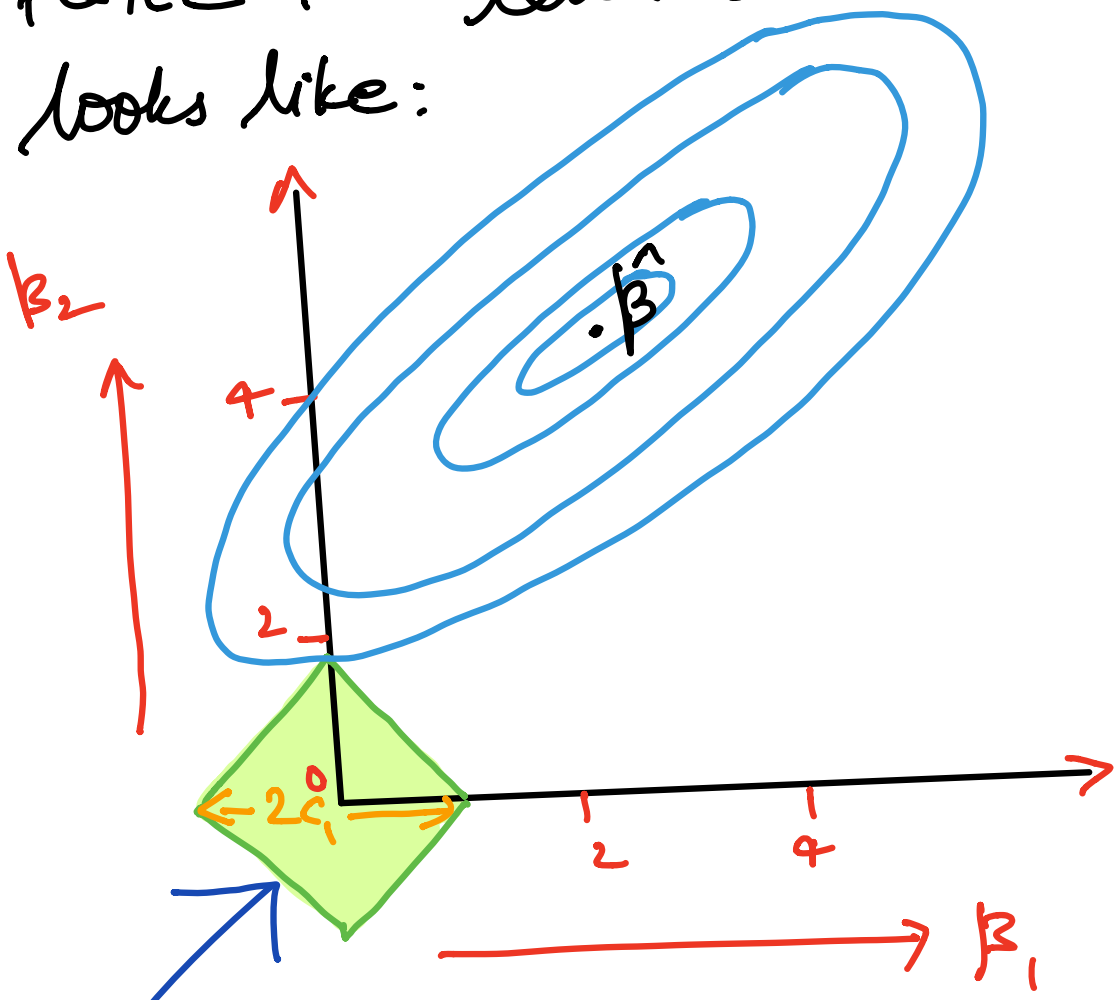
(8) Problem 4.19

(a) We know that Ridge constraint looks like: $\beta_1^2 + \beta_2^2 = 1$

But for lasso regression the constraint looks like:

$$|\beta_1| + |\beta_2| = 1$$

Hence the level sets (contour plots) looks like:



$$|\beta_1| + |\beta_2| = c_1$$

Reference: Class Notes

(b) Lasso regression is built on the extension of regularized linear regression. In case of lasso regression the loss function can be of the form

$$L = \sum (\hat{y}_i - y_i)^2 + \lambda \sum |\beta|$$

In ridge regression the β coefficients were reduced, but the values were never driven down to zero. In effect this means that the irrelevant features were minimized, hence their impact was minimized, but it never got rid of the irrelevant features.

But, in case of Lasso it sets the value of β to zero if they were not relevant. This was a huge advantage because it simplified the model, which proves to be a huge advantage.

(7) Problem 4.8

Reference: LFD Textbook

given that:

$$w(t+1) \leftarrow w(t) - \eta \nabla E_{aug}(w(t)) \quad \text{---(1)}$$

when we compute ∇E_{aug} as per eq (4.6) in LFD book we get:

$$\nabla E_{aug}(w) = \nabla E_{in}(w) + 2\lambda w. \quad \text{---(2)}$$

Using equation (2) in equation (1) we get:

$$w(t+1) \leftarrow w(t) - \eta \nabla E_{aug}(w(t))$$

$$= w(t) - \eta \left(\nabla E_{in}(w)\right) + 2\lambda w)$$

$$= w(t) - 2w(t)\eta\lambda - \eta \nabla E_{in}(w(t))$$

$$= (1 - 2\eta\lambda)(w(t)) - \eta \nabla E_{in}(w(t))$$

Hence, we can conclude that:

$$w(t+1) \leftarrow (1-2\eta\lambda)\, w(t) - \eta \nabla E_{in}(w(t))$$

---

(5) **Problem 3.17**

$$E(u,v) = e^u + e^{2v} + u^2 - 3uv + 4v^2 - 3u - 5v$$

(a)

$$E(u+\Delta u, v+\Delta v)$$
$$= E(u,v) + \nabla E(u,v)^T \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}$$
$$+ O\left(\|(\Delta u, \Delta v)\|^2\right) \rightarrow \text{by definition}$$

We are asked to approximate $E(u+\Delta u, v+\Delta v)$ by $\hat{E}_1(\Delta u, \Delta v)$, where $\hat{E}_1$ is the first order Taylor's expansion of E around $(u,v) = (0,0)$.

so,

$$\hat{E}_1(\Delta u, \Delta v) \simeq E(0,0) + \nabla E(0,0)^T \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}$$

$$= \left(e^0 + e^0 + e^0 + 0 - 3(0) + 4(0) \quad -3(0) - 5(0)\right)$$
$$+ \left(e^u + e^{uv} v + 2u - 3v - 3,\ 2e^{2v} + e^{uv} u - 3u + 8v - 5\right)\Big|_{(0,0)} \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}$$

Substituting $u=0$ and $v=0$ we get

$$\hat{E}_1(\Delta u, \Delta v) = -2\Delta u - 3\Delta v + 3$$

Comparing with $a_u \Delta u + a_v \Delta v + a$

$$\Rightarrow a_u = -2$$
$$a_v = -3$$
$$a = 3$$

(b) Differentiating to minimize $\hat{E}_1(\Delta u, \Delta v)$

$$= E(0,0) + \nabla E(0,0)^T \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}$$

expanding we get:

$$\hat{E}_1(\Delta u, \Delta v) \geq E(0,0) - \|\nabla E(0,0)\| \cdot \|\Delta u, \Delta v\|.$$

To calculate the optimal $(\Delta u, \Delta v)$

$$= - \frac{\nabla E(0,0)}{\|\nabla E(0,0)\|} \|(\Delta u, \Delta v)\|$$

Now, from problem (a) we know that

$$\nabla E(0,0) = \begin{pmatrix} -2 \\ -3 \end{pmatrix}$$

$$\| \nabla E(0,0) \| = \sqrt{(-2)^2 + (-3)^2}$$

$$= \sqrt{4+9} = \sqrt{13}$$

$$\| \Delta u, \Delta v \| = 0.5 = \frac{1}{2}$$

Hence, multiplying them all we get:

the optimal $(\Delta u, \Delta v)$ as

$$-\left(\frac{1}{2} \times \frac{1}{\sqrt{13}}\right)\begin{pmatrix} -2 \\ -3 \end{pmatrix}$$

Taking the negative sign in:

$$\frac{1}{2\sqrt{13}}\begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

(c) From (a) we continue to expand $E(u+\Delta u, v+\Delta v)$ to the second order after which we get:

$$E(u,v) + \nabla E(u,v)^T \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}$$

$$+ \frac{1}{2}(\Delta u, \Delta v) \nabla^2 E(u,v) \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix} + O\left(\|(\Delta u, \Delta v)\|^2\right)$$

Now, $[u=0, v=0]$

Hence,

$$\hat{E}_2(\Delta u, \Delta v) =$$

$$E(0,0) + \nabla E(0,0)^T \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix} + \frac{1}{2}(\Delta u, \Delta v) \nabla^2 E(0,0) \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}$$

Here:

$E(0,0) = 3 \rightarrow$ from (a)

$$\nabla E(0,0)^T \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix} = (-2, -3) \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix} \rightarrow \text{from (a)}$$

Expanding the third term we get,

$$\frac{1}{2}(\Delta u, \Delta v) \begin{bmatrix} e^u + e^{uv} v^2 + 2 & e^{uv}(uv+1) - 3 \\ e^{uv}(uv+1) - 3 & 4e^{2v} + e^{uv} u^2 + 8 \end{bmatrix} \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}$$

$$= \frac{1}{2}(\Delta u, \Delta v) \begin{bmatrix} e^0 + (e^0(0)) + 2 & e^0(0+1) - 3 \\ e^0(0+1) - 3 & 4e^0 + e^0(0) + 8 \end{bmatrix} \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}$$

$$= \frac{1}{2}(\Delta u, \Delta v) \begin{bmatrix} 1+2 & 1-3 \\ 1-3 & 4+8 \end{bmatrix} \begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}$$

$$= \frac{1}{2}(\Delta u, \Delta v)\begin{bmatrix} 3 & -2 \\ -2 & 12 \end{bmatrix}\begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix}$$

Adding all 3 terms we get:

$$\frac{3}{2}(\Delta u)^2 + 6(\Delta v)^2 - 2\Delta u \Delta v - 2\Delta u - 3\Delta v + 3$$

Comparing with

$$b_{uu}(\Delta u)^2 + b_{vv}(\Delta v)^2 + b_{uv}(\Delta u)(\Delta v) + b_u(\Delta u) + b_v(\Delta v) + b$$

Hence:

$$b_{uu} = -\frac{3}{2} = -1.5$$

$$b_{vv} = -2$$

$$b_u = -2$$

$$b_v = -3$$

$$b = 3$$

(d) We have to minimize $\hat{E}_2$ over $(\Delta u, \Delta v)$

We know that

$$\nabla \hat{E}_2(0,0) = \nabla E(0,0) + \nabla^2 E(0,0)\begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix} = 0$$

However, we assertion that:

$$\begin{pmatrix} \Delta u \\ \Delta v \end{pmatrix} = -\left(\nabla^2 E(u,v)\right)^{-1} \nabla E(u,v)$$

Now from problem (c) we know that

$$\nabla^2 E(0,0) = \begin{pmatrix} 3 & -2 \\ -2 & 12 \end{pmatrix} \quad \text{---} \quad (1)$$

Using the fact that $\nabla^2 E(u,v)\big|_{(0,0)}$ is positive definite (as mentioned in the question) implies that:

$$\det\left(\nabla^2 E(u,v)\right| \neq 0$$

from equation (1) we know that $\nabla^2 E(u,v)$ $= 36 - 4 = 32 \neq 0$. Hence an inverse exists.

Checking the invertability over any (u,v) we know:

$$(u,v)\left(\nabla^2 E(u,v)\right)(u,v)^T \geq 0$$

$$\Rightarrow (u,v)\begin{pmatrix} 3 & -2 \\ -2 & 12 \end{pmatrix}\begin{pmatrix} u \\ v \end{pmatrix} \geq 0$$

Hence, we have proved that the matrix is semi-definite, thus we can prove that the optimal column vector

$$\begin{bmatrix} \Delta u^* \\ \Delta v^* \end{bmatrix} = -\left(\nabla^2 E(u,v)\right)^{-1} \nabla E(u,v)$$

since $u=0, v=0$

$$\begin{pmatrix} \Delta u^* \\ \Delta v^* \end{pmatrix} = -\left(\nabla^2 E(0,0)\right)^{-1} \nabla E(0,0)$$

We have already shown that:

$$\nabla^2 E(0,0) = 32$$

From earlier calculations of $\nabla E(0,0)$ we know that $\nabla E(0,0) = \begin{pmatrix} -(24+6) \\ -4-9 \end{pmatrix} = \begin{pmatrix} -30 \\ -13 \end{pmatrix}$

$$\therefore \begin{pmatrix} \Delta u^* \\ \Delta v^* \end{pmatrix} = -\left(32\right)^{-1} \begin{pmatrix} -30 \\ -13 \end{pmatrix}$$

$$= \frac{1}{32} \begin{pmatrix} 30 \\ 13 \end{pmatrix}$$

(e) —

4.4 (a) We have to normalize f

We know that: $$f(x) = \sum_{q=0}^{q_f} a_q L_q(x)$$

Here: $L_q(x)$ is a Legendre polynomial.

To normalize we have to calculate

$E_{q,x}(f^2)$ ⟶ from Exercise 4.2

$$E_{q,x}(f^2) = E_x[E_{q|x}[f^2|x]]$$

$$= \sum_{q=0}^{q_f} E_x\left[L_q^2(x)\right] \quad -(1)$$

Now, going into the definition of $L_q(x)$ ⇒

$$E_x[L_q^2(x)] = \frac{1}{2}\int_{-1}^{1} L_q^2(x)\,dx$$

$$= \frac{1}{2q+1}$$

But, from equation (1)

$$E_{a,n}[f^2] = \sum_{q=0}^{Q_f} \frac{1}{2q+1}$$

But rescaling to $E_{a,n}[f^2] = 1$

Hence by comparing them we have a factor of $\frac{1}{\sum_{q=0}^{Q_f} \frac{1}{(2q+1)}}$.

but, this is for f^2. Hence to normalize f, we have a factor of

$$\frac{1}{\sqrt{\sum_{q} \frac{1}{2q+1}}}.$$

(c) We can be given various learning scenarios that all have different degree of target function (Q_f). They may all have different coefficients. They may all have different sizes (N). Also, as said in the question, they may have different

σ. Hence to accomodate all of these variations over various experiments we take the average. In this question we are using g_2 and g_{10}. So, these can be done using $\mathcal{H}_2$ & $\mathcal{H}_{10}$.

Reference: LFD Textbook

(c)

From the formula:

Reference: LFD Textbook

$$E_{out}(h(x)) = E_{x,y}\left[(h(x)-y(x))^2\right]$$

Hence doing the same for g_{10}, we get:

$$E_{out}(g_{10}) = E_{x,y}\left[(g_{10}(x)-y(x))^2\right]$$

Now, $y(x) = f(x)+\sigma\epsilon$ ← from formula!

$$E_{out}(g_{10}) = E_{x,y}\left[(g_{10}(x)-[f(x)+\sigma\epsilon])^2\right]$$

$$= E_{x,y}\left[\left(g_{10}(x) - f(x) - \sigma\epsilon\right)^2\right]$$

(b) Taking the hint and going into chapter 3, we find that going into Z space will help us find $g_2(x)$.

We know: $x \in \mathcal{X}$.

From chapter 3 we also know:

$$z = \phi_2(x) \text{ which belongs}$$

in the Z_2 space.

Now we know: $\tilde{g}_2 = \tilde{w}^T z$ (from formula ①)

Hence:

$$g_2(x) = g_2(z)$$

$$\Rightarrow g_2(x) = g_2(\phi_2(x))$$

$$\Rightarrow g_2(x) = \tilde{w}^T \phi_2(x) \rightarrow \text{similar to } ①$$

Just like this we can calculate $g_{10}(x)$

Hence, $z = \phi_{10}(x)$ and $z \in \mathcal{Z}_{10}$

and $\tilde{g}_{10} = \tilde{w}^T z$ where $z \in \mathcal{Z}_{10}$

Hence : $g_{10}(x) = g_{10}(z)$

$\Rightarrow g_{10}(x) = g_{10}(\phi_{10}(x))$

$\Rightarrow \underline{g_{10}(x) = \tilde{w}^T \phi_{10}(x)}$

Reference: Page 101, Chapter 3, LFD

(d) –

(f) –

Problem_3_2

November 8, 2020

1 Problem 3.2 Learning From Data

[1]:
```
from matplotlib import pyplot as plt
import numpy as np
import random
from random import seed
```

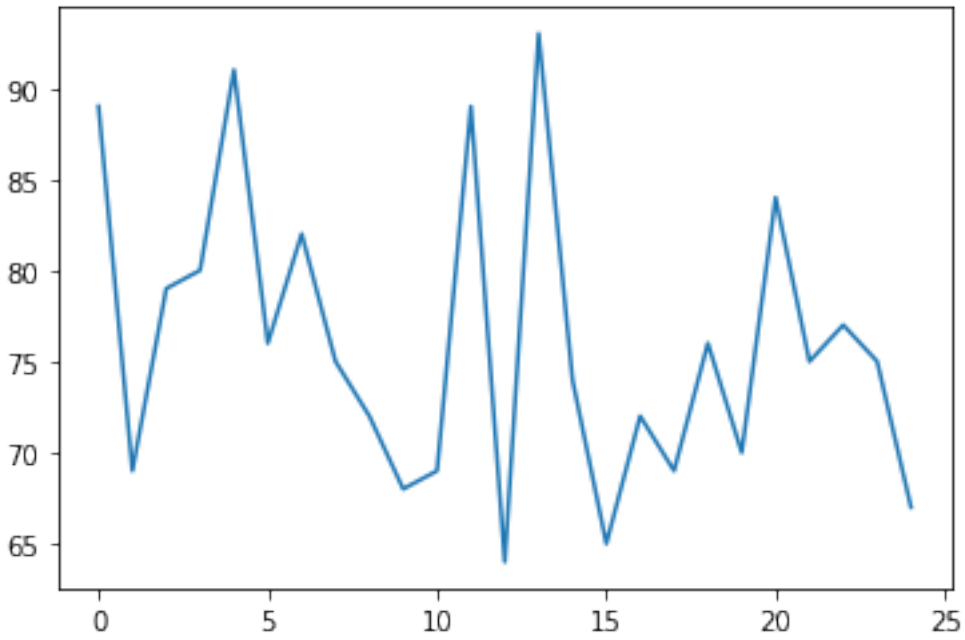
[2]:
```
thk = 5
rad = 10
s = np.linspace(0.2, 5, 25)
iter_list = []
```

[3]:
```
for sep in s:
    xs_red = []
    ys_red = []

    for x_coord in np.arange(-(rad + thk), rad + thk, 0.6):
        for y_coord in np.arange(0, rad + thk, 0.6):
            if rad ** 2 <= (x_coord - 0) ** 2 + (y_coord - 0) ** 2 <= (rad +␣
↪thk) ** 2:
                xs_red.append(x_coord)
                ys_red.append(y_coord)

    xs_blue = []
    ys_blue = []

    for x_coord in np.arange(-(thk / 2), (thk / 2 + (2 * rad) + thk), 0.6):
        for y_coord in np.arange(-sep, -(rad + +sep + thk), -0.6):
            if rad ** 2 <= (x_coord - ((thk / 2) + rad)) ** 2 + (y_coord -␣
↪(-sep)) ** 2 <= (rad + thk) ** 2:
                xs_blue.append(x_coord)
                ys_blue.append(y_coord)

    """
    A function for prediction of Y
    """
    def Y_predict(x_vector, w):
```

```
        x_new = [1]
        for i in x_vector:
            x_new.append(i)
        x_new = np.array((x_new))
        res = (np.dot(x_new, w))
        if res > 0:
            Y = 1
            return Y
        elif res < 0:
            Y = -1
            return Y
        elif res == 0:
            Y = 0
            return Y


    count = 0
    """
    The main training function for the data, with the
    Attributes
    ----------

    X - The data set
    iterations - the number of times the weights are iterated
    eta - the learning rate
    """

    def train(X, iterations, eta):
        global count
        global w
        global all_combined_targets
        for y_idx in range(len(X)):
            ran_num = random.randint(0, len(X) - 1)
            x_train = X[ran_num]
            y_t = Y_predict(x_train, w)
            misrepresented_list = []
            for i, j in enumerate(all_combined_targets):
                if j != y_t:
                    misrepresented_list.append(i)
            if len(misrepresented_list) == 0:
                print('Full accuracy achieved')
                break
            random_selection = random.randint(0, len(misrepresented_list) - 1)
            random_index = misrepresented_list[random_selection]
            x_selected = X[random_index]
            y_selected = all_combined_targets[random_index]
            x_with1 = [1]
```

```
            for i in x_selected:
                x_with1.append(i)
            x_with1 = np.array((x_with1))
            s_t = np.matmul(w, x_with1)
            if (y_selected * s_t) <= 1:
                w = w + (eta * (y_selected - s_t) * x_with1)
                count += 1
            if (count == iterations):
                break


    xs_red = np.array(xs_red)
    ys_red = np.array(ys_red)
    xs_blue = np.array(xs_blue)
    ys_blue = np.array(ys_blue)
    points_1 = []
    res1 = []
    for i in range(len(xs_red)):
        points_1.append([xs_red[i], ys_red[i]])
        res1.append(-1)
    points_1 = np.array(points_1)

    points_2 = []
    res2 = []
    for i in range(len(xs_blue)):
        points_2.append([xs_blue[i], ys_blue[i]])
        res2.append(1)
    points_2 = np.array(points_2)
    all_input = np.concatenate((points_1, points_2))  # creating a combined␣
↪dataset
    all_d = np.concatenate((res2, res1))

    # Visualizing the linearly separable dataset
    length_dataset = len(xs_red)
    d1 = -1 * (np.ones(int(length_dataset / 2)))
    d2 = np.ones(int(length_dataset / 2))
    all_combined_targets = np.concatenate((d2, d1))

    # initializing all parameters
    count = 0
    w0, w1, w2 = 0, 0, 0
    w = np.array((w0, w1, w2))
    weight = 0
    iterations = 100
    eta = 0.01
    # calling the function
    train(all_input, iterations, eta)
```

```
    iter_list.append(count)

plt.plot(iter_list)
plt.show()
```



Problem_3_1

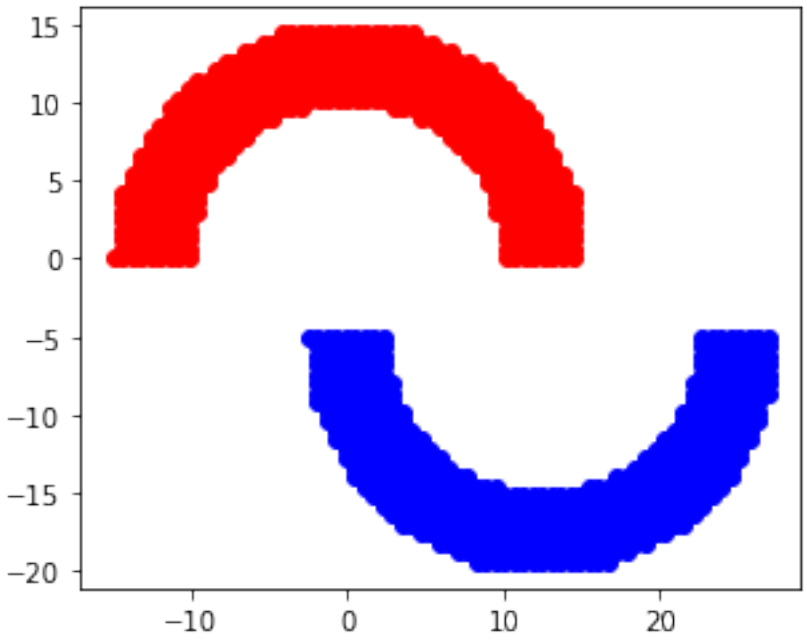
November 8, 2020

1 Problem 3.1 Learning From Data

[1]:
```
from matplotlib import pyplot as plt
import numpy as np
import random
from random import seed
```

[2]:
```
thk = 5
sep = 5
rad = 10

xs_red = []
ys_red = []

fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_aspect('equal')

for x_coord in np.arange(-(rad+thk),rad+thk,0.6):
    for y_coord in np.arange(0 ,rad+thk,0.6):
        if rad**2 <= (x_coord - 0)**2 + (y_coord - 0)**2 <= (rad+thk)**2:
            xs_red.append(x_coord)
            ys_red.append(y_coord)


xs_blue = []
ys_blue = []

for x_coord in np.arange(-(thk/2),(thk/2 + (2*rad) + thk),0.6):
    for y_coord in np.arange(-sep ,-(rad++sep+thk),-0.6):
        if rad**2 <= (x_coord - ((thk/2) + rad))**2 + (y_coord - (-sep))**2 <=␣
 ↪(rad+thk)**2:
            xs_blue.append(x_coord)
            ys_blue.append(y_coord)


plt.scatter(xs_red, ys_red,color = 'red')
```

```
plt.scatter(xs_blue, ys_blue,color = 'blue')
plt.show()
```



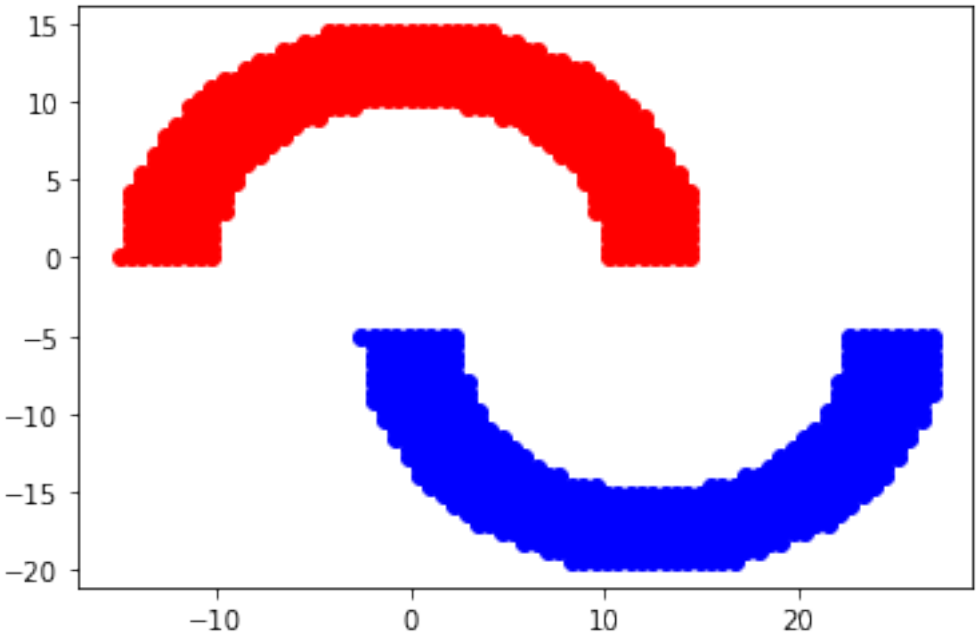
[3]:
```
"""
A function for prediction of Y
"""
def Y_predict(x_vector,w):
    x_new = [1]
    for i in x_vector:
        x_new.append(i)
    x_new = np.array((x_new))
    res = (np.dot(x_new,w))
    if res > 0:
        Y = 1
        return Y
    elif res < 0:
        Y = -1
        return Y
    elif res ==0:
        Y = 0
        return Y

"""
The main training function for the data, with the
Attributes
```

```
----------

X - The data set
iterations - the number of times the weights are iterated
eta - the learning rate
"""
def train(X,iterations,eta):
    global count
    global w
    global all_combined_targets
    for y_idx in range (len(X)):
        ran_num = random.randint(0,len(X)-1)
        x_train = X[ran_num]
        y_t = Y_predict(x_train,w)
        misrepresented_list = []
        for i,j in enumerate(all_combined_targets):
            if j!=y_t:
                misrepresented_list.append(i)
        if len(misrepresented_list)==0:
            print('Full accuracy achieved')
            break
        random_selection = random.randint(0,len(misrepresented_list)-1)
        random_index = misrepresented_list[random_selection]
        x_selected = X[random_index]
        y_selected = all_combined_targets[random_index]
        x_with1 = [1]
        for i in x_selected:
            x_with1.append(i)
        x_with1 = np.array((x_with1))
        s_t = np.matmul(w,x_with1)
        if (y_selected*s_t)<=1:
            w = w+(eta*(y_selected-s_t)*x_with1)
        if (count==iterations):
            print('maximum iterations reached in the training block')
            break
            count+=1
```

[4]:
```
xs_red = np.array(xs_red)
ys_red = np.array(ys_red)
xs_blue = np.array(xs_blue)
ys_blue = np.array(ys_blue)
points_1 = []
res1 = []
for i in range(len(xs_red)):
    points_1.append([xs_red[i],ys_red[i]])
    res1.append(-1)
points_1 = np.array(points_1)
```

```
points_2 = []
res2 = []
for i in range(len(xs_blue)):
    points_2.append([xs_blue[i],ys_blue[i]])
    res2.append(1)
points_2 = np.array(points_2)
all_input = np.concatenate((points_1, points_2)) #creating a combined dataset
all_d = np.concatenate((res2,res1))
```

[5]:
```
#Visualizing the linearly separable dataset
plt.scatter(xs_red, ys_red, color='red')
plt.scatter(xs_blue,ys_blue, color='blue')
length_dataset = len(xs_red)
d1 = -1 * (np.ones(int(length_dataset/2)))
d2 = np.ones(int(length_dataset/2))
all_combined_targets = np.concatenate((d2,d1))
```

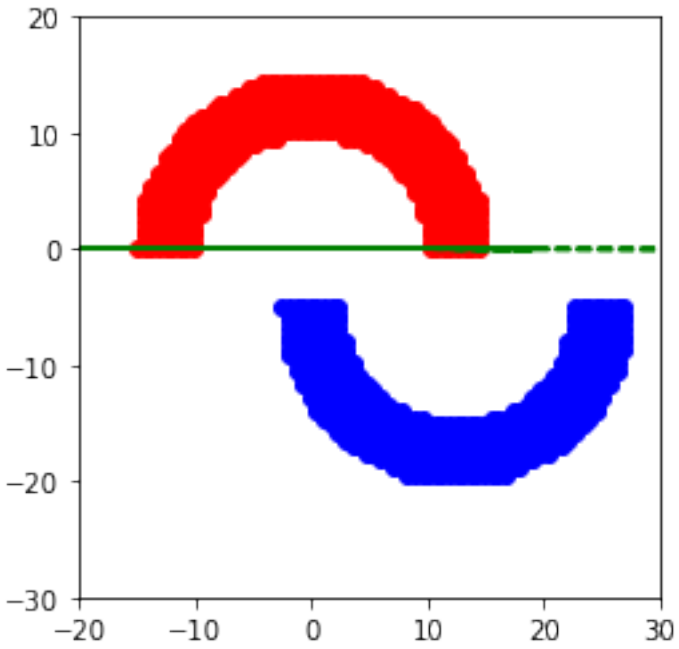


[6]:
```
#initializing all parameters
count = 0
# w0 = random.randint(1,4)
# w1 = random.randint(1,4)
# w2 = random.randint(1,4)
w0,w1,w2 = 0,0,0
w = np.array((w0,w1,w2))
```

```
weight= 0
iterations = 100
eta = 0.01
#calling the function
train(all_input,iterations,eta)
```

[7]:
```
#Visualizing the linearly separable dataset
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_aspect('equal')

plt.scatter(xs_red, ys_red, color='red')
plt.scatter(xs_blue,ys_blue, color='blue')
m = -(w[1]/w[2])
c = -(w[0]/w[2])
plt.plot( m*all_input + c,all_input ,'g--')
plt.xlim([-20, 30])
plt.ylim([-30, 20])
```

[7]: (-30.0, 20.0)



[8]:
```
from sklearn import linear_model
```

[9]:
```
reg = linear_model.LinearRegression()
reg.fit(all_input,all_d)
```

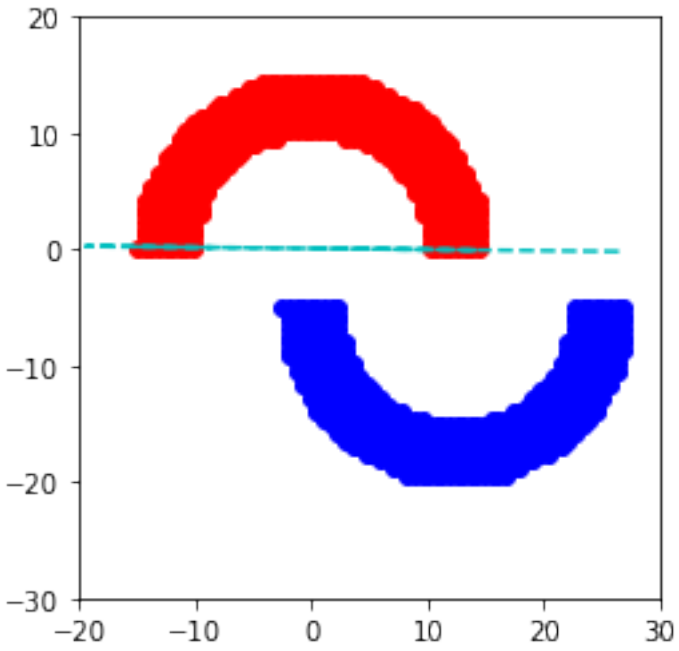
[9]:
```
LinearRegression()
```

[10]:
```
reg.coef_
```

[10]:
```
array([-0.00967062,  0.07796808])
```

[11]:
```
#Visualizing the linearly separable dataset
fig = plt.figure()
ax = fig.add_subplot(111)
ax.set_aspect('equal')

plt.scatter(xs_red, ys_red, color='red')
plt.scatter(xs_blue,ys_blue, color='blue')
m = reg.coef_[0]
c = reg.coef_[1]
plt.plot( all_input, m*all_input + c ,'c--')
plt.xlim([-20, 30])
plt.ylim([-30, 20])
```

[11]:
```
(-30.0, 20.0)
```



PocketAlgorithm_Exercise_3_2

November 8, 2020

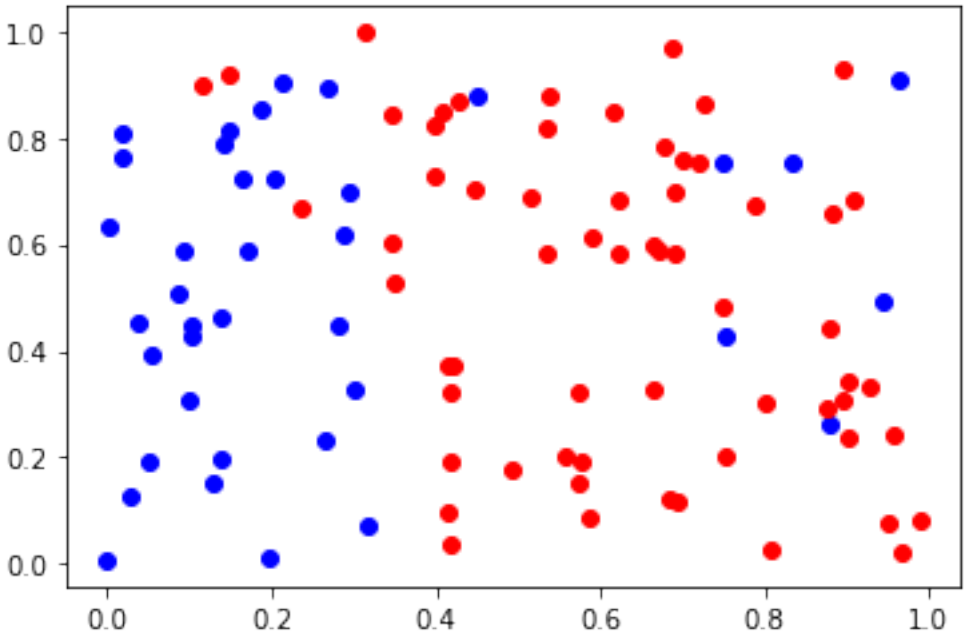
1 Exercise 3.2 Learning from data

First the problem is solved as per the question, then its solved using weights from Linear regression

[1]:
```
from matplotlib import pyplot as plt
import numpy as np
import random
from random import seed
np.random.seed(1)
```

[2]:
```
#creating a linearly in-sepearable datasset
def generate_samples(n_train=100, n_test=1000):

    X_train = np.zeros((n_train, 2), dtype=np.float)
    X_test = np.zeros((n_test, 2), dtype=np.int)
    y_train = np.zeros((n_train, ), dtype=np.float)
    y_test = np.zeros((n_test, ), dtype=np.int)

    X_train[:, 0] = np.random.rand(n_train)
    X_test[:, 0] = np.random.rand(n_test)

    X_train[:, 1] = np.random.rand(n_train)
    X_test[:, 1] = np.random.rand(n_test)

    while True:

        weights_for_line = np.random.rand(3, 1)

        b = weights_for_line[0] / (0.5 - 0.2)
        w_1 = weights_for_line[1] / 0.9 - 0.3
        w_2 = weights_for_line[2] / 0.5 - 0.1

        for i in range(n_train):
            y_train[i] = np.sign(w_1 * X_train[i, 0] + w_2 * X_train[i, 1] + b)
        if np.abs((np.count_nonzero(y_train == 1) + 1) /
                          (np.count_nonzero(y_train == -1) + 1) - 0.5) < 0.1:
            break
    # flip the labels of the 10% dataset
```

```
    flip_y = np.random.choice(n_train, n_train // 10, replace=True)
    y_train[flip_y] = -y_train[flip_y]

    for i in range(n_test):
        y_test[i] = np.sign(w_1 * X_test[i, 0] + w_2 * X_test[i, 1] + b)
    flip_y = np.random.choice(n_test, n_test // 10, replace=True)
    y_test[flip_y] = -y_test[flip_y]

    return X_train, y_train, X_test, y_test

X_train, y_train, X_test, y_test = generate_samples(100)
X = X_train
err_train_now = []
err_train_hat = []
err_test_now = []
err_test_hat = []
train_err_now = 1
train_err_min = 1
pos_x = []
pos_y = []
neg_x = []
neg_y = []
for i in range(X.shape[0]):
    if y_train[i] == 1:
        pos_x.append(X[i, 0])
        pos_y.append(X[i, 1])
    else:
        neg_x.append(X[i, 0])
        neg_y.append(X[i, 1])

plt.scatter(pos_x, pos_y, c='blue')
plt.scatter(neg_x, neg_y, c='red')
plt.show()
```



[3]:
```
learningRate = 0.01
Y = y_train
oneVector = np.ones((X_train.shape[0], 1))
X_train = np.concatenate((oneVector, X_train), axis=1)
oneVector_test = np.ones((X_test.shape[0], 1))
X_test= np.concatenate((oneVector_test, X_test), axis=1)
plotData = []
weights = np.random.rand(3, 1)
w_hat = weights
misClassifications = 1
minMisclassifications = 10000
iteration = 0
```

[4]:
```
print(weights.shape)
```

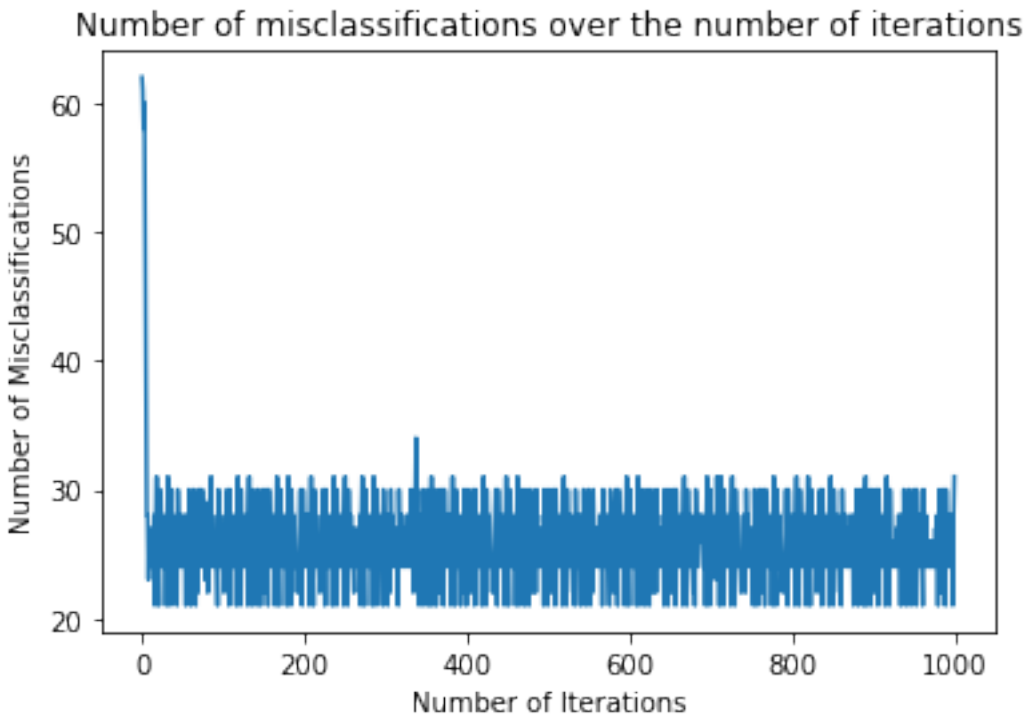
```
(3, 1)
```

[5]:
```
def evaluate_error(w, X, y):
    n = X.shape[0]
    pred = np.matmul(X, w)
    pred = np.sign(pred) - (pred == 0)
    pred = pred.reshape(-1)
    return np.count_nonzero(pred == y) / n
```

[6]:
```
while (misClassifications != 0 and (iteration<1000)):
    iteration += 1
```

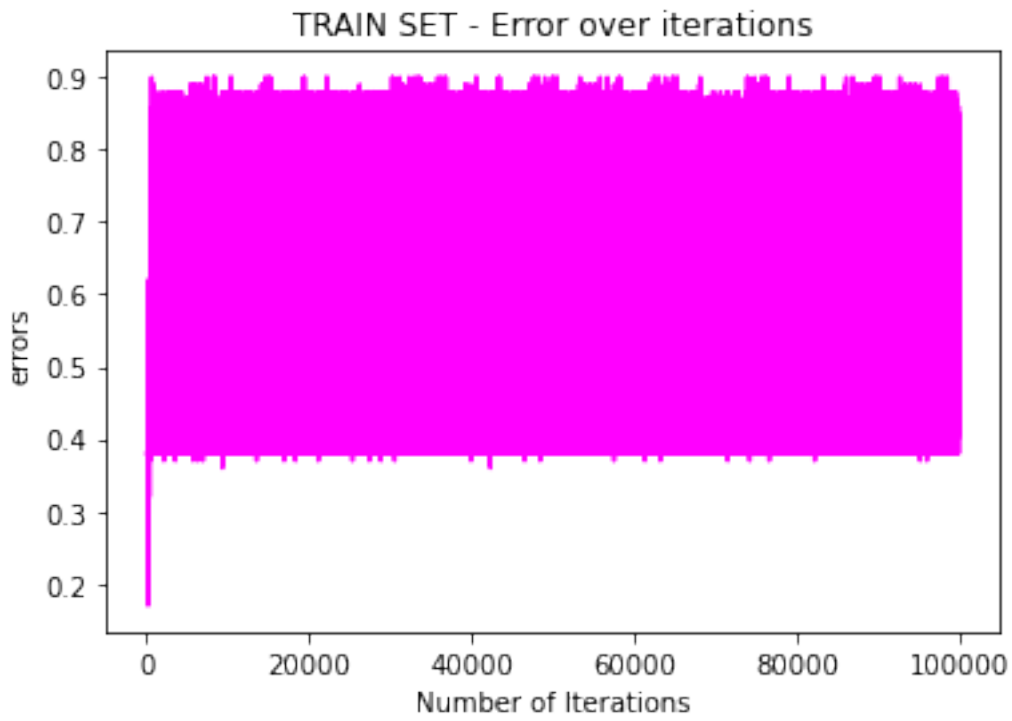
```
    misClassifications = 0
    for i in range(0, len(X_train)):
        currentX = X_train[i].reshape(-1, X_train.shape[1])
        currentY = Y[i]
        wTx = np.dot(currentX, weights)[0][0]
        if currentY == 1 and wTx < 0:
            misClassifications += 1
            weights = weights + learningRate * np.transpose(currentX)
        elif currentY == -1 and wTx > 0:
            misClassifications += 1
            weights = weights - learningRate * np.transpose(currentX)

        train_err_now = evaluate_error(weights, X_train, y_train)
        err_train_now.append(train_err_now)

        if train_err_now < train_err_min :
            train_err_min = train_err_now
            err_train_hat.append(train_err_min)
            w_hat = weights

    test_err_hat = evaluate_error(w_hat,X_test,y_test)
    test_err_now = evaluate_error(weights,X_test,y_test)

    err_test_hat.append(test_err_hat)
    err_test_now.append(test_err_now)

    plotData.append(misClassifications)
    if misClassifications<minMisclassifications:
        minMisclassifications = misClassifications
print(weights.transpose())
print ("Best Case Accuracy of Pocket Learning Algorithm is: ",(((X_train.
 ↪shape[0]-minMisclassifications)/X_train.shape[0])*100),"%")
plt.title('Number of misclassifications over the number of iterations')
plt.plot(np.arange(0,iteration),plotData)
plt.xlabel("Number of Iterations")
plt.ylabel("Number of Misclassifications")
plt.show()
```

```
[[ 0.00747124 -0.02781082  0.00501601]]
Best Case Accuracy of Pocket Learning Algorithm is:  79.0 %
```



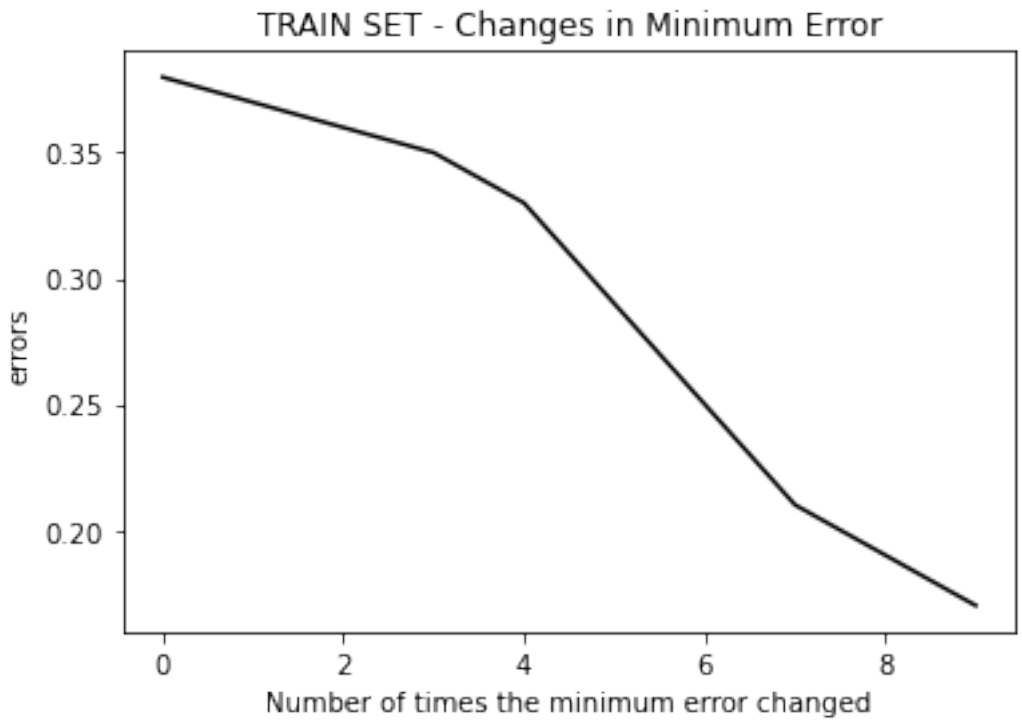
[7]:
```
plt.figure()
plt.title('TRAIN SET - Error over iterations')
plt.xlabel("Number of Iterations")
plt.ylabel("errors")
plt.plot(err_train_now,color = 'magenta')
plt.show
```

[7]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



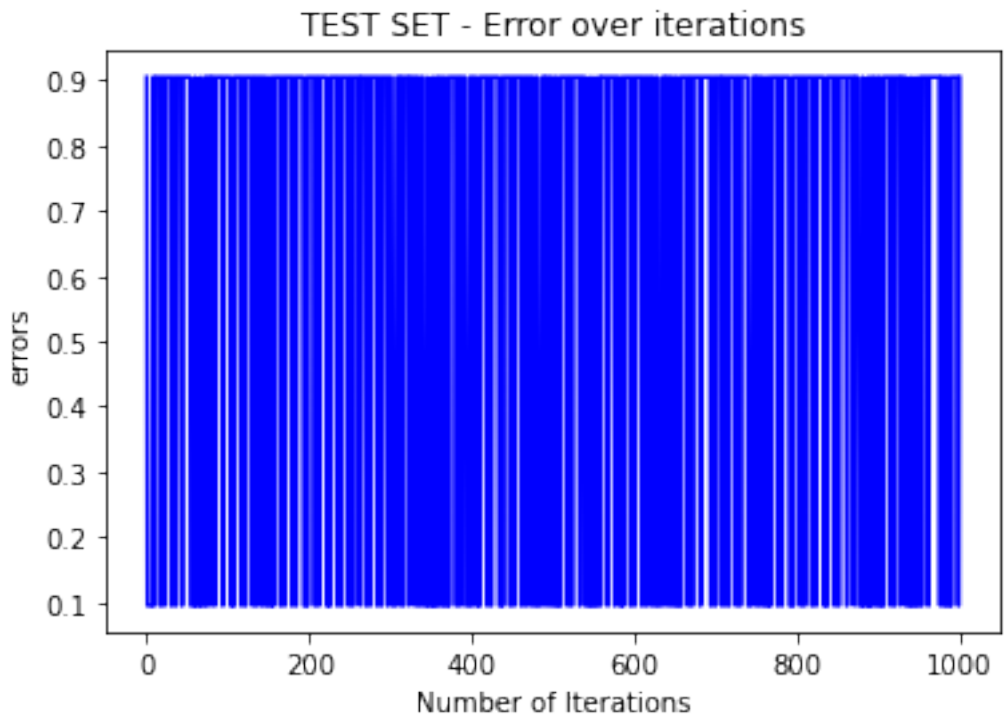
[8]:
```
plt.figure()
plt.title('TRAIN SET - Changes in Minimum Error')
plt.xlabel("Number of times the minimum error changed")
plt.ylabel("errors")
plt.plot(err_train_hat,color='black')
plt.show
```

[8]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[9]:
```
plt.figure()
plt.title('TEST SET - Error over iterations')
plt.xlabel("Number of Iterations")
plt.ylabel("errors")
plt.plot(err_test_now,color = 'blue')
plt.show
```

[9]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[10]:
```
plt.figure()
plt.title('TEST SET - Changes in Minimum Error')
plt.xlabel("Number of times the minimum error changed")
plt.ylabel("errors")
plt.plot(err_test_hat,color='red')
plt.show
```

[10]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```

2 Using weights from Linear Regression

[11]:
```
%reset
```

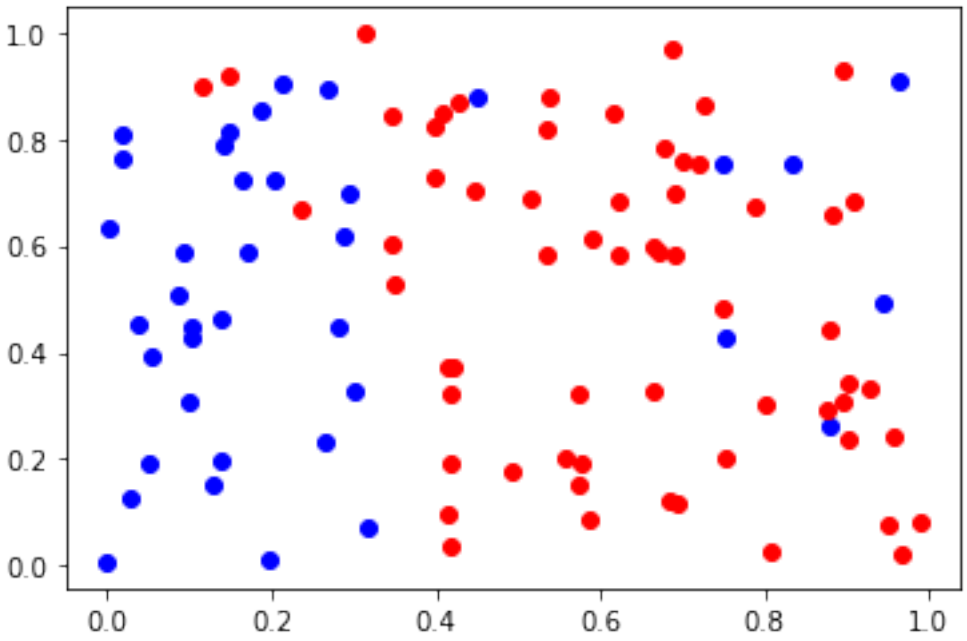
```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```

[12]:
```
from sklearn import linear_model
from matplotlib import pyplot as plt
import numpy as np
import random
from random import seed
np.random.seed(1)
```

[13]:
```
#creating a linearly in-sepearable datasset
def evaluate_error(w, X, y):
    n = X.shape[0]
    pred = np.matmul(X, w)
    pred = np.sign(pred) - (pred == 0)
    pred = pred.reshape(-1)
    return np.count_nonzero(pred == y) / n

def generate_samples(n_train=100, n_test=1000):
```

```
    X_train = np.zeros((n_train, 2), dtype=np.float)
    X_test = np.zeros((n_test, 2), dtype=np.int)
    y_train = np.zeros((n_train, ), dtype=np.float)
    y_test = np.zeros((n_test, ), dtype=np.int)

    X_train[:, 0] = np.random.rand(n_train)
    X_test[:, 0] = np.random.rand(n_test)

    X_train[:, 1] = np.random.rand(n_train)
    X_test[:, 1] = np.random.rand(n_test)

    while True:

        weights_for_line = np.random.rand(3, 1)

        b = weights_for_line[0] / (0.5 - 0.2)
        w_1 = weights_for_line[1] / 0.9 - 0.3
        w_2 = weights_for_line[2] / 0.5 - 0.1

        for i in range(n_train):
            y_train[i] = np.sign(w_1 * X_train[i, 0] + w_2 * X_train[i, 1] + b)
        if np.abs((np.count_nonzero(y_train == 1) + 1) /
                        (np.count_nonzero(y_train == -1) + 1) - 0.5) < 0.1:
            break
    # flip the labels of the 10% dataset
    flip_y = np.random.choice(n_train, n_train // 10, replace=True)
    y_train[flip_y] = -y_train[flip_y]

    for i in range(n_test):
        y_test[i] = np.sign(w_1 * X_test[i, 0] + w_2 * X_test[i, 1] + b)
    flip_y = np.random.choice(n_test, n_test // 10, replace=True)
    y_test[flip_y] = -y_test[flip_y]

    return X_train, y_train, X_test, y_test

X_train, y_train, X_test, y_test = generate_samples(100)
X = X_train
err_train_now = []
err_train_hat = []
err_test_now = []
err_test_hat = []
train_err_now = 1
train_err_min = 1
pos_x = []
pos_y = []
neg_x = []
neg_y = []
```

```
for i in range(X.shape[0]):
    if y_train[i] == 1:
        pos_x.append(X[i, 0])
        pos_y.append(X[i, 1])
    else:
        neg_x.append(X[i, 0])
        neg_y.append(X[i, 1])

plt.scatter(pos_x, pos_y, c='blue')
plt.scatter(neg_x, neg_y, c='red')
plt.show()
```



[14]:
```
learningRate = 0.01
Y = y_train
oneVector = np.ones((X_train.shape[0], 1))
X_train = np.concatenate((oneVector, X_train), axis=1)
oneVector_test = np.ones((X_test.shape[0], 1))
X_test= np.concatenate((oneVector_test, X_test), axis=1)
plotData = []
misClassifications = 1
minMisclassifications = 10000
iteration = 0
```

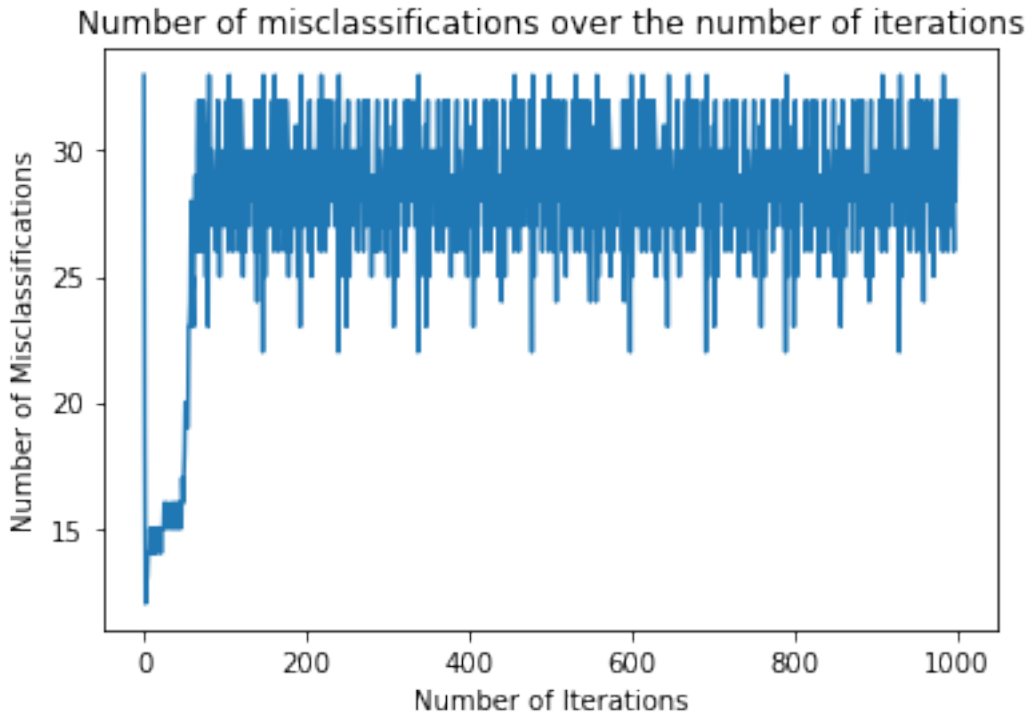
[15]:
```
reg = linear_model.LinearRegression()
reg.fit(X_train,y_train)
```

```
weights_linear_regression = reg.coef_
l = []
for i in weights_linear_regression:
    l.append(i)
weights = np.array(l)
weights = weights.reshape(3,1)
w_hat = weights
```

[16]:
```
err_train_now = []
err_train_hat = []
err_test_now = []
err_test_hat = []
train_err_now = 1
train_err_min = 1
while (misClassifications != 0 and (iteration<1000)):
    iteration += 1
    misClassifications = 0
    for i in range(0, len(X_train)):
        currentX = X_train[i].reshape(-1, X_train.shape[1])
        currentY = Y[i]
        wTx = np.dot(currentX, weights)[0][0]
        if currentY == 1 and wTx < 0:
            misClassifications += 1
            weights = weights + learningRate * np.transpose(currentX)
        elif currentY == -1 and wTx > 0:
            misClassifications += 1
            weights = weights - learningRate * np.transpose(currentX)

        train_err_now = evaluate_error(weights, X_train, y_train)
        err_train_now.append(train_err_now)

        if train_err_now < train_err_min :
            train_err_min = train_err_now
            err_train_hat.append(train_err_min)
            w_hat = weights

    test_err_hat = evaluate_error(w_hat,X_test,y_test)
    test_err_now = evaluate_error(weights,X_test,y_test)

    err_test_hat.append(test_err_hat)
    err_test_now.append(test_err_now)

    plotData.append(misClassifications)
    if misClassifications<minMisclassifications:
        minMisclassifications = misClassifications
print(weights.transpose())
```

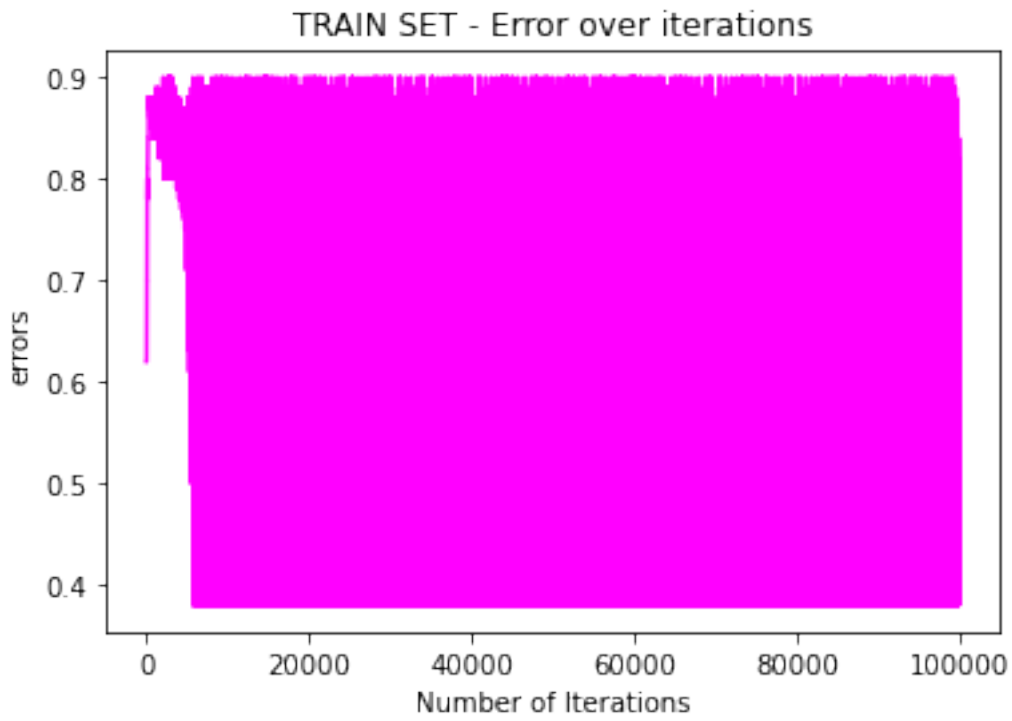
```
print ("Best Case Accuracy of Pocket Learning Algorithm is: ",(((X_train.
 ↪shape[0]-minMisclassifications)/X_train.shape[0])*100),"%")
plt.title('Number of misclassifications over the number of iterations')
plt.plot(np.arange(0,iteration),plotData)
plt.xlabel("Number of Iterations")
plt.ylabel("Number of Misclassifications")
plt.show()
```

```
[[-8.67361738e-17 -2.86485107e-02  1.12287992e-03]]
Best Case Accuracy of Pocket Learning Algorithm is:  88.0 %
```



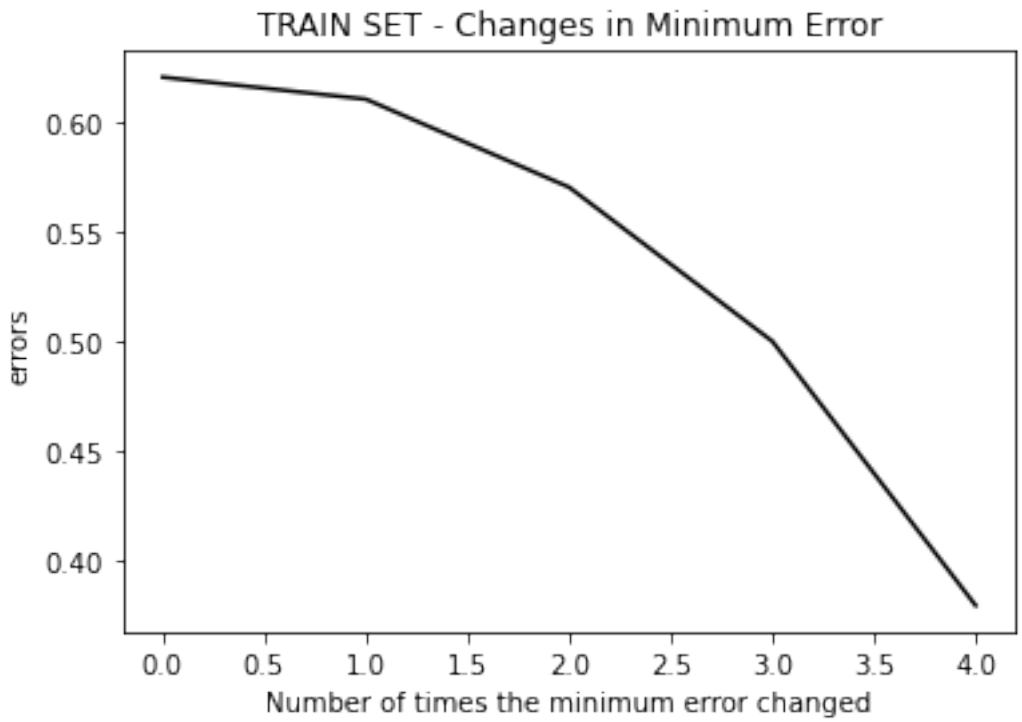
[17]:
```
plt.figure()
plt.title('TRAIN SET - Error over iterations')
plt.xlabel("Number of Iterations")
plt.ylabel("errors")
plt.plot(err_train_now,color = 'magenta')
plt.show
```

[17]: <function matplotlib.pyplot.show(*args, **kw)>



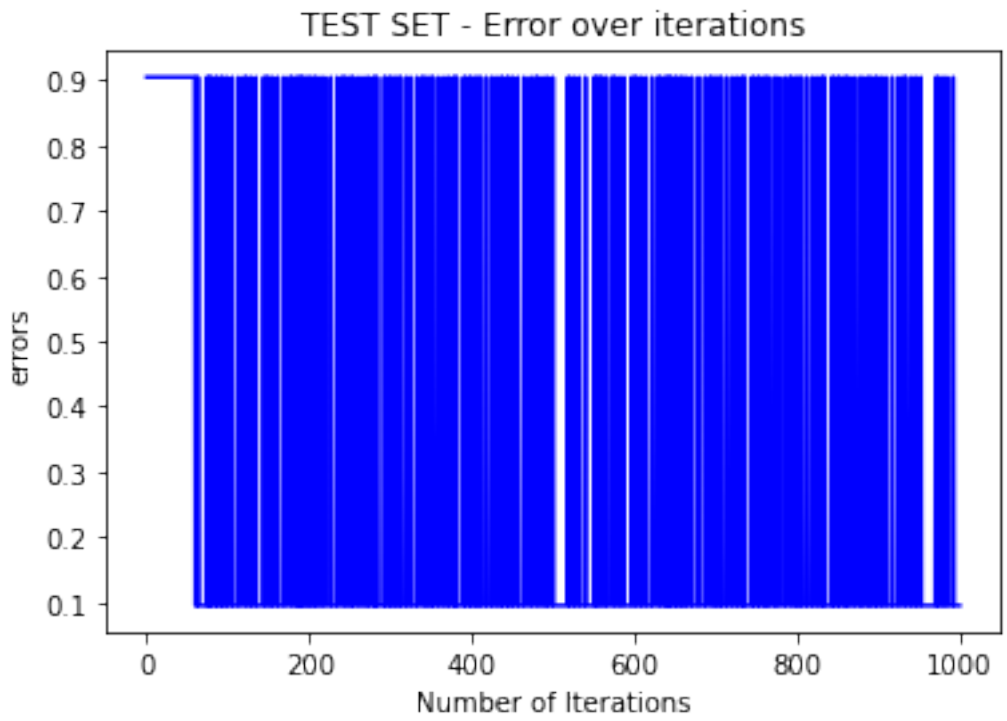
[18]:
```
plt.figure()
plt.title('TRAIN SET - Changes in Minimum Error')
plt.xlabel("Number of times the minimum error changed")
plt.ylabel("errors")
plt.plot(err_train_hat,color='black')
plt.show
```

[18]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[19]:
```
plt.figure()
plt.title('TEST SET - Error over iterations')
plt.xlabel("Number of Iterations")
plt.ylabel("errors")
plt.plot(err_test_now,color = 'blue')
plt.show
```

[19]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[20]:
```
plt.figure()
plt.title('TEST SET - Changes in Minimum Error')
plt.xlabel("Number of times the minimum error changed")
plt.ylabel("errors")
plt.plot(err_test_hat,color='red')
plt.show
```

[20]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```

3 Using weights from Logistic Regression

[21]:
```
%reset
```

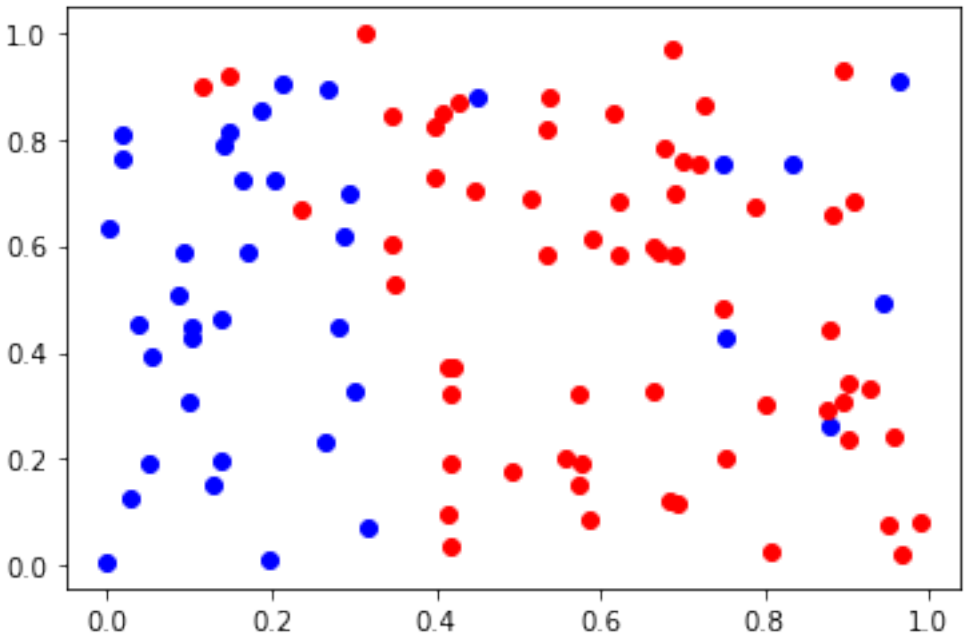
```
Once deleted, variables cannot be recovered. Proceed (y/[n])? y
```

[22]:
```
from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot as plt
import numpy as np
import random
from random import seed
np.random.seed(1)
```

[23]:
```
#creating a linearly in-sepearable datasset
def evaluate_error(w, X, y):
    n = X.shape[0]
    pred = np.matmul(X, w)
    pred = np.sign(pred) - (pred == 0)
    pred = pred.reshape(-1)
    return np.count_nonzero(pred == y) / n

def generate_samples(n_train=100, n_test=1000):
```

```
    X_train = np.zeros((n_train, 2), dtype=np.float)
    X_test = np.zeros((n_test, 2), dtype=np.int)
    y_train = np.zeros((n_train, ), dtype=np.float)
    y_test = np.zeros((n_test, ), dtype=np.int)

    X_train[:, 0] = np.random.rand(n_train)
    X_test[:, 0] = np.random.rand(n_test)

    X_train[:, 1] = np.random.rand(n_train)
    X_test[:, 1] = np.random.rand(n_test)

    while True:

        weights_for_line = np.random.rand(3, 1)

        b = weights_for_line[0] / (0.5 - 0.2)
        w_1 = weights_for_line[1] / 0.9 - 0.3
        w_2 = weights_for_line[2] / 0.5 - 0.1

        for i in range(n_train):
            y_train[i] = np.sign(w_1 * X_train[i, 0] + w_2 * X_train[i, 1] + b)
        if np.abs((np.count_nonzero(y_train == 1) + 1) /
                        (np.count_nonzero(y_train == -1) + 1) - 0.5) < 0.1:
            break
    # flip the labels of the 10% dataset
    flip_y = np.random.choice(n_train, n_train // 10, replace=True)
    y_train[flip_y] = -y_train[flip_y]

    for i in range(n_test):
        y_test[i] = np.sign(w_1 * X_test[i, 0] + w_2 * X_test[i, 1] + b)
    flip_y = np.random.choice(n_test, n_test // 10, replace=True)
    y_test[flip_y] = -y_test[flip_y]

    return X_train, y_train, X_test, y_test

X_train, y_train, X_test, y_test = generate_samples(100)
X = X_train
err_train_now = []
err_train_hat = []
err_test_now = []
err_test_hat = []
train_err_now = 1
train_err_min = 1
pos_x = []
pos_y = []
neg_x = []
neg_y = []
```

```
for i in range(X.shape[0]):
    if y_train[i] == 1:
        pos_x.append(X[i, 0])
        pos_y.append(X[i, 1])
    else:
        neg_x.append(X[i, 0])
        neg_y.append(X[i, 1])

plt.scatter(pos_x, pos_y, c='blue')
plt.scatter(neg_x, neg_y, c='red')
plt.show()
```



[24]:
```
learningRate = 0.01
Y = y_train
oneVector = np.ones((X_train.shape[0], 1))
X_train = np.concatenate((oneVector, X_train), axis=1)
oneVector_test = np.ones((X_test.shape[0], 1))
X_test= np.concatenate((oneVector_test, X_test), axis=1)
plotData = []
misClassifications = 1
minMisclassifications = 10000
iteration = 0
```

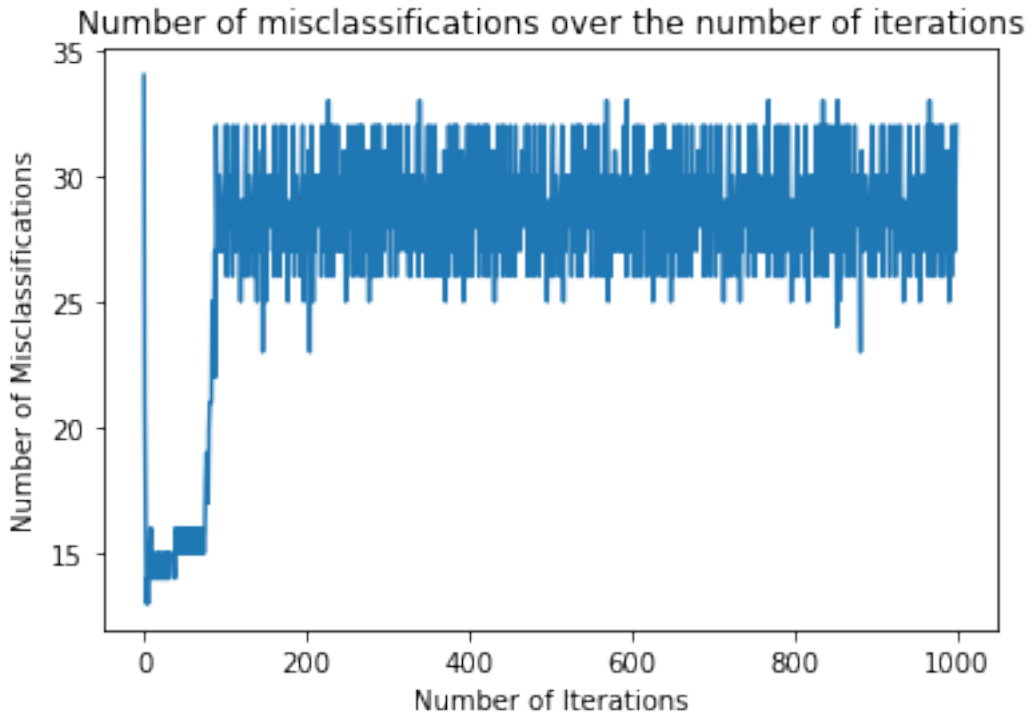
[25]:
```
logisticRegr = LogisticRegression()
logisticRegr.fit(X_train, y_train)
```

```
logisticRegr.fit(X_train, y_train)
weights_logistic_regression = logisticRegr.coef_
l = []
for i in weights_logistic_regression:
    l.append(i)
weights = np.array(l)
weights = weights.reshape(3,1)
w_hat = weights
```

[26]:
```
err_train_now = []
err_train_hat = []
err_test_now = []
err_test_hat = []
train_err_now = 1
train_err_min = 1
while (misClassifications != 0 and (iteration<1000)):
    iteration += 1
    misClassifications = 0
    for i in range(0, len(X_train)):
        currentX = X_train[i].reshape(-1, X_train.shape[1])
        currentY = Y[i]
        wTx = np.dot(currentX, weights)[0][0]
        if currentY == 1 and wTx < 0:
            misClassifications += 1
            weights = weights + learningRate * np.transpose(currentX)
        elif currentY == -1 and wTx > 0:
            misClassifications += 1
            weights = weights - learningRate * np.transpose(currentX)

        train_err_now = evaluate_error(weights, X_train, y_train)
        err_train_now.append(train_err_now)

        if train_err_now < train_err_min :
            train_err_min = train_err_now
            err_train_hat.append(train_err_min)
            w_hat = weights

    test_err_hat = evaluate_error(w_hat,X_test,y_test)
    test_err_now = evaluate_error(weights,X_test,y_test)

    err_test_hat.append(test_err_hat)
    err_test_now.append(test_err_now)

    plotData.append(misClassifications)
    if misClassifications<minMisclassifications:
        minMisclassifications = misClassifications
print(weights.transpose())
```

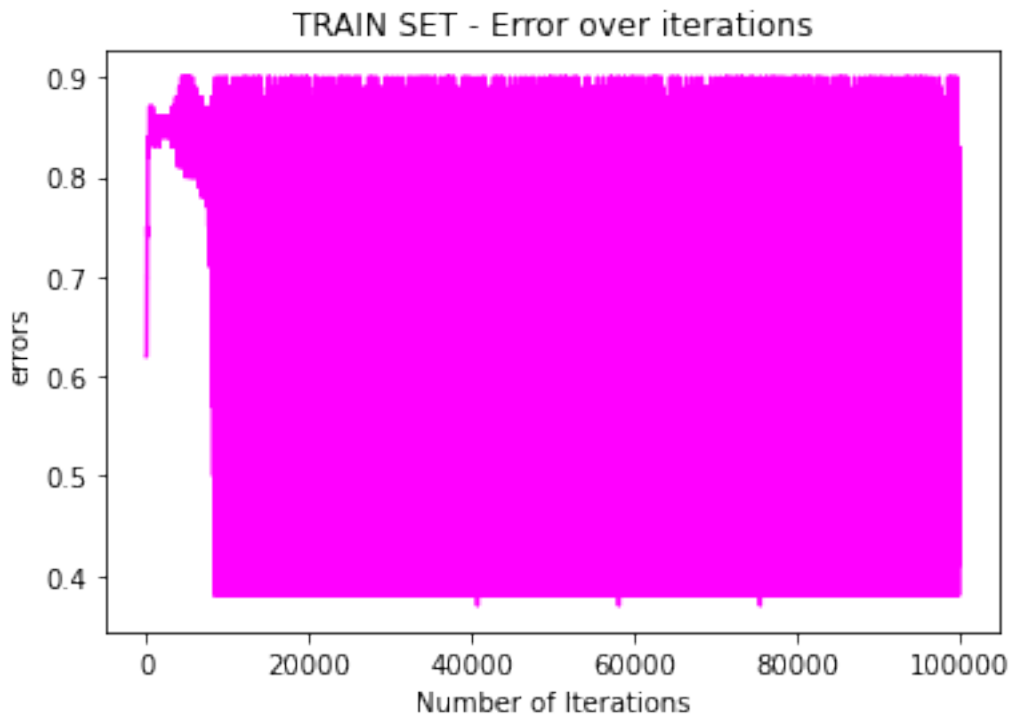
```
print ("Best Case Accuracy of Pocket Learning Algorithm is: ",(((X_train.
 ↪shape[0]-minMisclassifications)/X_train.shape[0])*100),"%")
plt.title('Number of misclassifications over the number of iterations')
plt.plot(np.arange(0,iteration),plotData)
plt.xlabel("Number of Iterations")
plt.ylabel("Number of Misclassifications")
plt.show()
```

```
[[ 3.00085250e-05 -2.89055681e-02  3.49017994e-03]]
Best Case Accuracy of Pocket Learning Algorithm is:  87.0 %
```



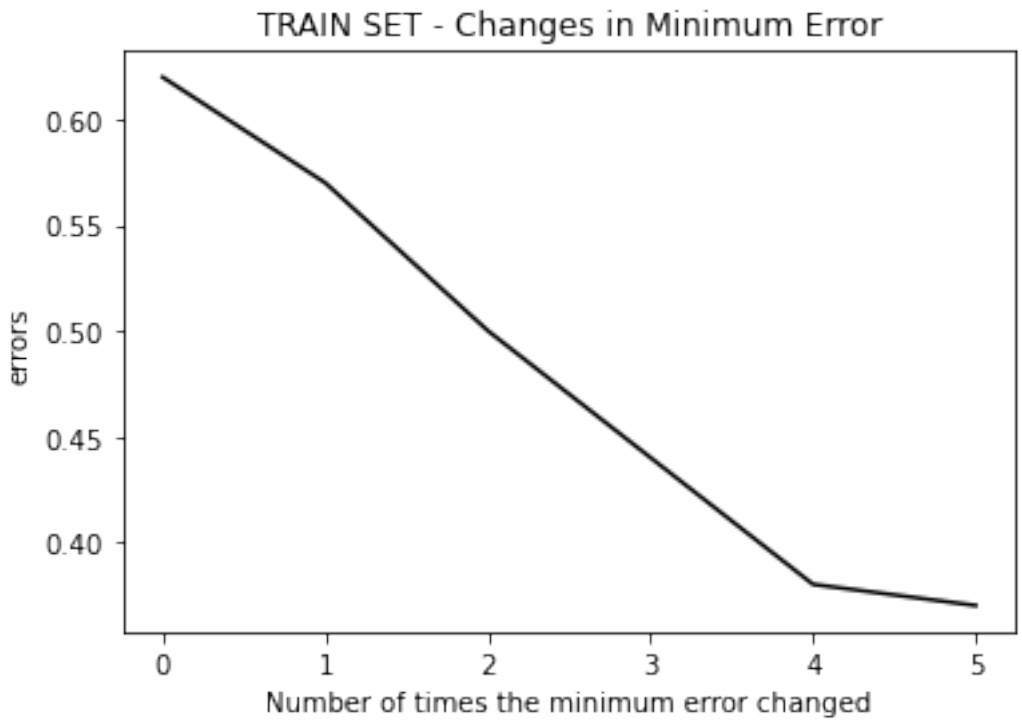
[27]:
```
plt.figure()
plt.title('TRAIN SET - Error over iterations')
plt.xlabel("Number of Iterations")
plt.ylabel("errors")
plt.plot(err_train_now,color = 'magenta')
plt.show
```

[27]: <function matplotlib.pyplot.show(*args, **kw)>



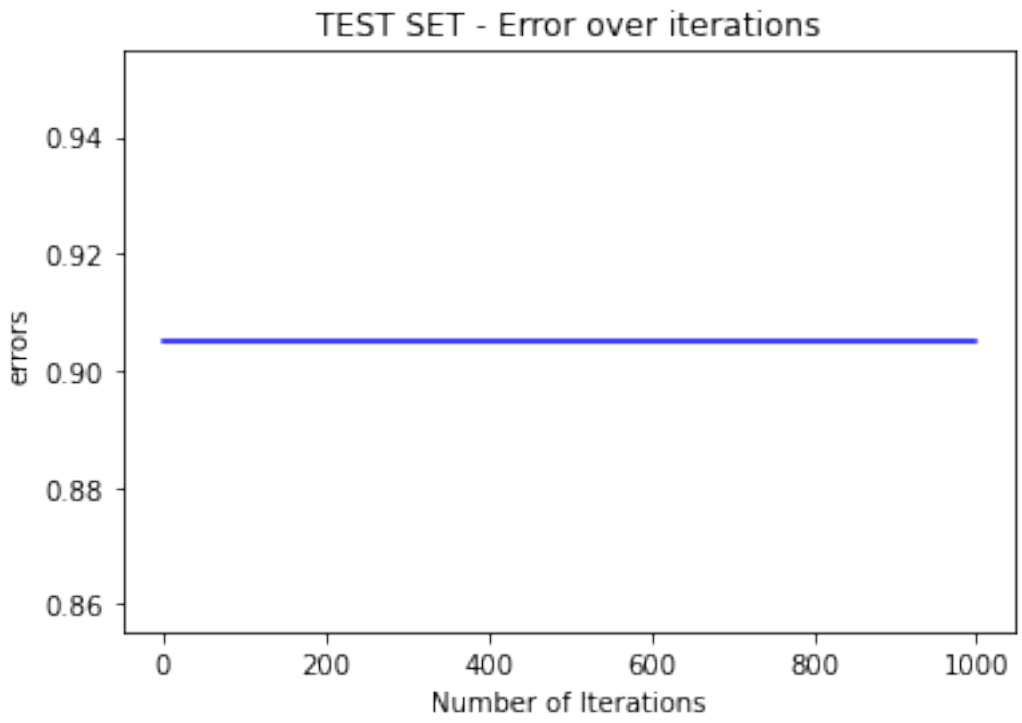
[28]:
```
plt.figure()
plt.title('TRAIN SET - Changes in Minimum Error')
plt.xlabel("Number of times the minimum error changed")
plt.ylabel("errors")
plt.plot(err_train_hat,color='black')
plt.show
```

[28]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[29]:
```
plt.figure()
plt.title('TEST SET - Error over iterations')
plt.xlabel("Number of Iterations")
plt.ylabel("errors")
plt.plot(err_test_now,color = 'blue')
plt.show
```

[29]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```



[30]:
```
plt.figure()
plt.title('TEST SET - Changes in Minimum Error')
plt.xlabel("Number of times the minimum error changed")
plt.ylabel("errors")
plt.plot(err_test_hat,color='red')
plt.show
```

[30]:
```
<function matplotlib.pyplot.show(*args, **kw)>
```


TEST SET - Changes in Minimum Error
errors
0.94
0.92
0.90
0.88
0.86
0
200
400
600
800
1000
Number of times the minimum error changed