In [ ]:
```
from utils import Dataset ,perceptron
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

In [ ]:
```
def plot1(inputs,weights):
    sns.scatterplot(data=inputs,x="X",y="Y",hue='Labels',s=2).set(title='plot with deci
    inputs=np.array(inputs)
    Ya_nn=[]
    ia_nn=[]
    for i in np.linspace(np.amin(inputs[:,:1]),np.amax(inputs[:,:1])):

            slope = -(weights[1])/(weights[2])
            intercept = -(weights[0]/weights[2])
            #y =mx+c, m is slope and c is intercept
            y = (slope*i) + intercept
            ia_nn.append(i)
            Ya_nn.append(y)
    sns.lineplot(x=ia_nn,y=Ya_nn,color='black',markersize=4)
    plt.legend()
    plt.show()
```

In [ ]:
```
def plot2(inputs,weights):
    print('All the points on/above the decision boundary belongs to class 1')
    sns.scatterplot(data=inputs,x="x1",y="x2",hue='out').set(title='plot with decision
    inputs=np.array(inputs)
    Ya_nn=[]
    ia_nn=[]
    for i in np.linspace(np.amin(inputs[:,:1]),np.amax(inputs[:,:1])):
            slope = 0
            intercept =0
            if(weights[2]!=0):
                slope = -(weights[1])/(weights[2])
            if(weights[2]!=0):
                intercept = -(weights[0]/weights[2])
            #y =mx+c, m is slope and c is intercept
            y = (slope*i) + intercept
            ia_nn.append(i)
            Ya_nn.append(y)
    sns.lineplot(x=ia_nn,y=Ya_nn,color='black',markersize=4)
    plt.legend()
    plt.show()
```

Part 1

In [ ]:
```
#without noise 10,000 samples
d=Dataset(10_000)
df=d.get()
df
```

Out[ ]:

	X	Y	Labels

	X	Y	Labels
0	0.898230	-0.439526	0
1	0.723509	2.309685	1
2	-0.822943	-0.568124	0
3	-0.281687	3.959506	1
4	0.281912	0.959440	0
...	...	...	...
9995	0.867456	2.502486	1
9996	0.992129	3.125221	1
9997	-0.242525	0.970145	0
9998	0.135728	2.009254	1
9999	0.860023	2.489745	1

10000 rows × 3 columns

In [ ]:
```
# d=Dataset(10000) with noise added
df1=d.get(True)
df1
```

Out[ ]:

	X	Y	Labels
0	0.561958	2.377508	1
1	-0.524846	0.868321	0
2	0.392301	0.718524	0
3	-0.027525	-1.074236	0
4	-0.127290	4.008247	1
...	...	...	...
9995	-0.457150	2.153987	1
9996	0.682129	3.870551	1
9997	-0.304008	-0.910514	0
9998	0.723689	-0.757239	0
9999	-0.120075	-0.982539	0

10000 rows × 3 columns

In [ ]:
```
df.shape
```

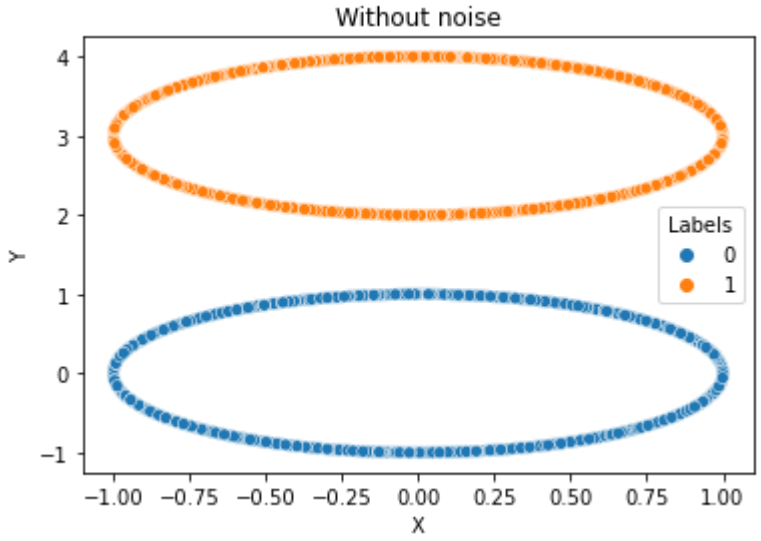
Out[ ]:
```
(10000, 3)
```

In [ ]:
```
df1.shape
```

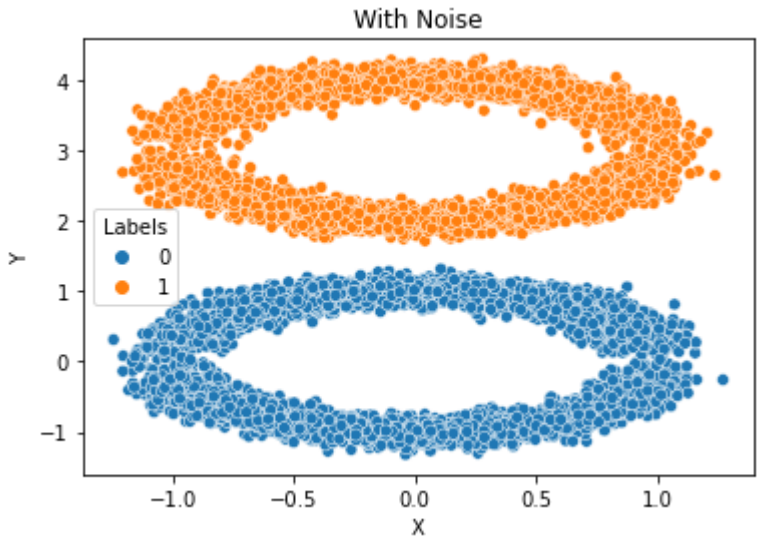
Out[ ]:
```
(10000, 3)
```

Part 2

In [ ]:
```
sns.scatterplot(data=df,x="X",y="Y",hue='Labels').set(title='Without noise')
plt.show()
```



In [ ]:
```
sns.scatterplot(data=df1,x="X",y="Y",hue='Labels').set(title='With Noise')
plt.show()
```



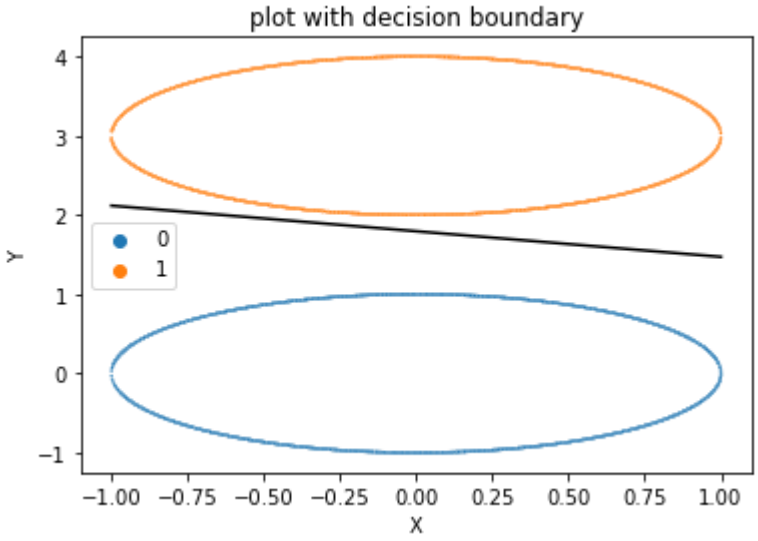
Part 3

In [ ]:
```
p=perceptron()
# trainning the perceptron with learnable bias on without noise data
p.train_weights(np.array(df),5,1)
```

```
>epoch=0, error=6.000 [-4.0, 0.720376711074459, 2.228126731090508]
```

```
>epoch=1, error=0.000 [-4.0, 0.720376711074459, 2.228126731090508]
>epoch=2, error=0.000 [-4.0, 0.720376711074459, 2.228126731090508]
>epoch=3, error=0.000 [-4.0, 0.720376711074459, 2.228126731090508]
>epoch=4, error=0.000 [-4.0, 0.720376711074459, 2.228126731090508]
```

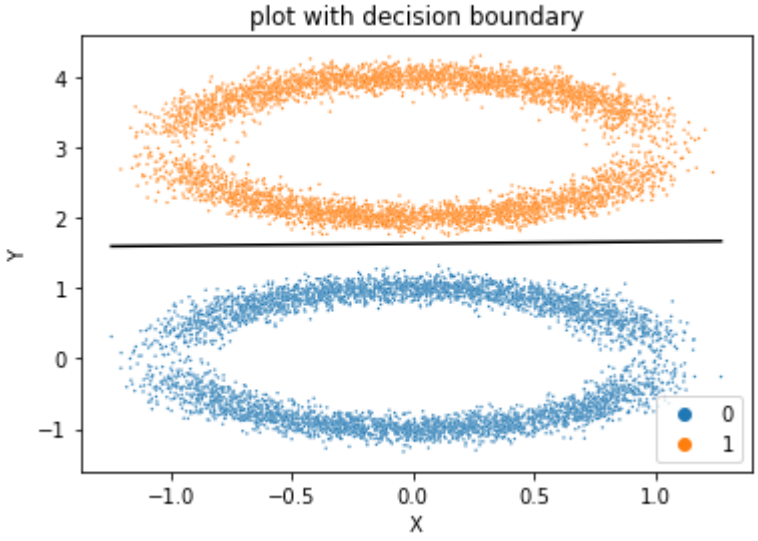
In [ ]:
```
plot1(df,p.weights)
```



In [ ]:
```
p1=perceptron()
# the third parameter is a flag to make bias 0 or learnable
p1.train_weights(np.array(df1),5,1)
```

```
>epoch=0, error=13.000 [-5.0, -0.08963611587720852, 3.0705765156008695]
>epoch=1, error=0.000 [-5.0, -0.08963611587720852, 3.0705765156008695]
>epoch=2, error=0.000 [-5.0, -0.08963611587720852, 3.0705765156008695]
>epoch=3, error=0.000 [-5.0, -0.08963611587720852, 3.0705765156008695]
>epoch=4, error=0.000 [-5.0, -0.08963611587720852, 3.0705765156008695]
```

In [ ]:
```
plot1(df1,p1.weights)
```



The Decision boundary for both the datasets with and without noise exist because the data was linearly separable even after adding noise.

Part 4

In [ ]:
```
#classifier with bias set to constant 0 and data used = without noise data
p3=perceptron()
p3.train_weights(np.array(df),100,0)
```

```
>epoch=0, error=2966.000 [0.0, 0.7298420020772367, 0.357438305509001]
>epoch=1, error=2960.000 [0.0, 0.5832358101376358, 0.35825602366765097]
>epoch=2, error=2933.000 [0.0, 0.2057844099432271, 0.7686138975407416]
>epoch=3, error=2960.000 [0.0, 0.16148262079835995, 0.729842004188166]
>epoch=4, error=2969.000 [0.0, 0.5219670291163601, 0.33676161506953906]
>epoch=5, error=2986.000 [0.0, 0.4717218654541335, 0.704404884716293]
>epoch=6, error=2980.000 [0.0, 1.079825032968965, 0.9073725933009041]
>epoch=7, error=2976.000 [0.0, 0.2209896263871578, 0.7191236710494185]
>epoch=8, error=2969.000 [0.0, 0.43954112055514294, 0.3400822539368865]
>epoch=9, error=2962.000 [0.0, 1.015208100299604, 1.8071932011940732]
>epoch=10, error=3019.000 [0.0, 0.27536465020939405, 0.7640282261868007]
>epoch=11, error=2982.000 [0.0, 1.0515376689089853, 0.8808230634647499]
>epoch=12, error=2973.000 [0.0, 0.5526712300032313, 0.3441118716194903]
>epoch=13, error=2976.000 [0.0, 1.0479557213482988, 1.7845564166478538]
>epoch=14, error=2954.000 [0.0, 0.8895651222267846, 0.6777836946090053]
>epoch=15, error=2981.000 [0.0, 1.0238962330598933, 0.9260682819063845]
>epoch=16, error=2943.000 [0.0, 0.413556559985341, 0.27462716404239196]
>epoch=17, error=2962.000 [0.0, 0.8908658796434203, 0.7124451415946332]
>epoch=18, error=2931.000 [0.0, 0.23115779673824832, 0.740252831825445]
>epoch=19, error=2945.000 [0.0, 0.2235790460408258, 0.7242992172946875]
>epoch=20, error=2986.000 [0.0, 0.5234628793449905, 0.3871186492613027]
>epoch=21, error=2946.000 [0.0, 1.2920249004415536, 0.05901453359430808]
>epoch=22, error=2982.000 [0.0, 1.2204751962876135, 1.8716760371991037]
>epoch=23, error=2990.000 [0.0, 0.22422237420943558, 0.7234297371584014]
>epoch=24, error=2986.000 [0.0, 0.5241062075136003, 0.3862491691250166]
>epoch=25, error=2970.000 [0.0, 0.44639381683861723, 0.678922103955324]
>epoch=26, error=2970.000 [0.0, 0.26706168043483247, 0.7502170105397071]
>epoch=27, error=2977.000 [0.0, 0.21916295331120894, 0.6911493922734425]
>epoch=28, error=2930.000 [0.0, 0.21757398061945143, 0.7685420138586131]
>epoch=29, error=2968.000 [0.0, 1.0142076828134532, 1.8075247419397082]
>epoch=30, error=3019.000 [0.0, 0.2743642327232432, 0.7643597669324357]
>epoch=31, error=2952.000 [0.0, 1.1771151204575394, 1.8575551953991236]
>epoch=32, error=2946.000 [0.0, 0.36000097747622983, 0.29509140122480215]
>epoch=33, error=2964.000 [0.0, 0.9634727157056591, 0.09376357206955566]
>epoch=34, error=2978.000 [0.0, 0.8286671960717851, 0.17091159142503753]
>epoch=35, error=2978.000 [0.0, 1.0807771201850676, 0.9121824437372951]
>epoch=36, error=2969.000 [0.0, 0.41333223013522646, 0.2639591861289309]
>epoch=37, error=2937.000 [0.0, 1.0290007708943756, 1.8365090267355781]
>epoch=38, error=2969.000 [0.0, 0.4234426902239492, 0.2579431579500425]
>epoch=39, error=2936.000 [0.0, 1.0984844252118817, 0.9248185494052407]
>epoch=40, error=2938.000 [0.0, 0.40318690552436887, 1.015195217376354]
>epoch=41, error=2961.000 [0.0, 0.2249056046032074, 0.7175465413194148]
>epoch=42, error=2966.000 [0.0, 0.48820509362789766, 0.3726391945716837]
>epoch=43, error=2983.000 [0.0, 1.070524616034751, 0.14059532518057638]
>epoch=44, error=2992.000 [0.0, 1.1933755799043713, 1.8207417640622467]
>epoch=45, error=2980.000 [0.0, 0.8353787811192361, 0.6998728606801979]
>epoch=46, error=2957.000 [0.0, 0.5709799817256336, 0.36623613088986684]
>epoch=47, error=2934.000 [0.0, 0.7853815210235524, 0.11009130939192546]
>epoch=48, error=2984.000 [0.0, 0.7710810197975861, 0.6625578906055828]
>epoch=49, error=2983.000 [0.0, 1.0559619710752766, 1.7878318840814518]
>epoch=50, error=2985.000 [0.0, 0.38474941336508084, 0.3022871081049666]
>epoch=51, error=2949.000 [0.0, 1.306959398099452, 0.33111040116575696]
>epoch=52, error=2942.000 [0.0, 0.38228576669027503, 0.3017813355892739]
```

```
>epoch=53, error=2971.000 [0.0, 1.013526276108336, 0.01271917072614348]
>epoch=54, error=2973.000 [0.0, 0.6237100943362186, 0.3351875401664276]
>epoch=55, error=2959.000 [0.0, 0.2074676747499653, 0.7705033767694777]
>epoch=56, error=2955.000 [0.0, 0.8055852329601458, 0.6899159179378979]
>epoch=57, error=2949.000 [0.0, 0.45229981165421385, 0.71146885373078]
>epoch=58, error=2977.000 [0.0, 0.14208576312262844, 0.704971490343748]
>epoch=59, error=2965.000 [0.0, 0.4461986841854164, 0.9940558255217292]
>epoch=60, error=2939.000 [0.0, 0.5633765427537001, 0.35474479368297207]
>epoch=61, error=2987.000 [0.0, 0.5119312457203402, 0.394646613895442]
>epoch=62, error=2970.000 [0.0, 0.534745362493044, 0.3779592823018635]
>epoch=63, error=2947.000 [0.0, 1.0273500674541456, 0.9786792570226949]
>epoch=64, error=2951.000 [0.0, 1.0892495592794464, 0.922387617734958]
>epoch=65, error=2960.000 [0.0, 0.2162171657523444, 0.7701594051693813]
>epoch=66, error=2974.000 [0.0, 1.2346106013652909, 1.861068228691483]
>epoch=67, error=2987.000 [0.0, 0.5234085199478451, 0.38316788394450196]
>epoch=68, error=2947.000 [0.0, 0.5312481918168195, 0.3781791364588837]
>epoch=69, error=2957.000 [0.0, 0.18670376451299986, 0.715567940751555]
>epoch=70, error=2991.000 [0.0, 1.0692349887912083, 1.8215128855644127]
>epoch=71, error=2977.000 [0.0, 0.5484383132575446, 0.36547457444020026]
>epoch=72, error=2963.000 [0.0, 0.5307174625607014, 0.37668383309745856]
>epoch=73, error=2929.000 [0.0, 1.0888987756027788, 0.914777160787471]
>epoch=74, error=2959.000 [0.0, 0.4892079182042428, 0.3577475901042183]
>epoch=75, error=2980.000 [0.0, 0.3516770827282829, 0.7137366194037883]
>epoch=76, error=2952.000 [0.0, 0.5167642367549385, 0.32442246994427004]
>epoch=77, error=2995.000 [0.0, 0.3905955395355807, 0.30001945257603435]
>epoch=78, error=2963.000 [0.0, 1.2341992048468862, 0.21152734632781878]
>epoch=79, error=2978.000 [0.0, 1.357478140685172, 0.3222387287233023]
>epoch=80, error=2992.000 [0.0, 0.5982498622677015, 0.36817567160504894]
>epoch=81, error=2968.000 [0.0, 0.8630263116400407, 0.6791626659175686]
>epoch=82, error=2946.000 [0.0, 0.21564619698009468, 0.7665582619873074]
>epoch=83, error=2976.000 [0.0, 0.2254605698586607, 0.722054598538863]
>epoch=84, error=2929.000 [0.0, 0.43899393925054686, 1.0149527097663333]
>epoch=85, error=2961.000 [0.0, 1.094251549779135, 0.9230244344393049]
>epoch=86, error=2949.000 [0.0, -0.2327600096165423, 0.33110459579379425]
>epoch=87, error=2990.000 [0.0, 0.5270926753150253, 0.3836383258418171]
>epoch=88, error=2954.000 [0.0, 0.3945932807696495, 0.30412907566079117]
>epoch=89, error=2955.000 [0.0, 1.6046741234935753, 1.3940772544985056]
>epoch=90, error=2942.000 [0.0, 1.0546074237744407, 1.8206648641931085]
>epoch=91, error=2938.000 [0.0, 1.0717646100135783, 0.9109388556224572]
>epoch=92, error=2995.000 [0.0, 1.037873243028897, 1.8721538680239593]
>epoch=93, error=2976.000 [0.0, 0.45747633718115455, 0.9993320340648483]
>epoch=94, error=2932.000 [0.0, 1.0259644472285208, 1.800666461210835]
>epoch=95, error=2927.000 [0.0, 0.4048775487847811, 0.7628896522114547]
>epoch=96, error=2933.000 [0.0, 0.5754382951548034, 0.3551700715093872]
>epoch=97, error=2969.000 [0.0, 0.5103494190500921, 0.3519204291583412]
>epoch=98, error=2993.000 [0.0, 0.5474508945226755, 0.3843436913677226]
>epoch=99, error=2971.000 [0.0, 0.5483186593984457, 0.3843399300705964]
```

The decision boundary for this model won't exist because when we made the bias 0 we are forcing the decision boundry to pass through the origin and also separate the data , but the point (0,0) is the center for the circle having k=0 (label = 0) which makes the weights oscillate to the error value no matter the number of epochs.which results in a non existent decision boundary which passes through (0,0)

However this is not the case when we allow the bias to be learnable (can update with error) it can shift with the weights accordingly to finally converge to a proper decision boundary.

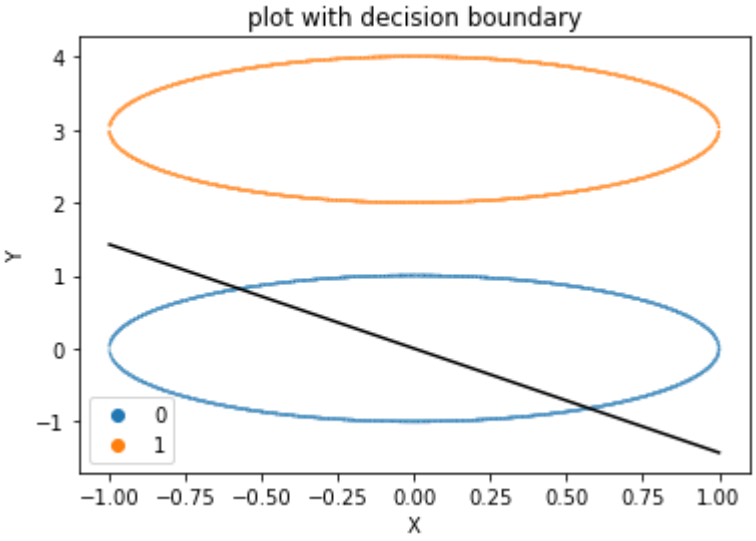
When bias was learnable it took only 2 epochs to find the correct weights but when we fix the bias the error is not reducing even after 100 epochs hence the perceptron will not converge if the bias is set to 0 for the given dataset

The plots can be seen below for both the cases.

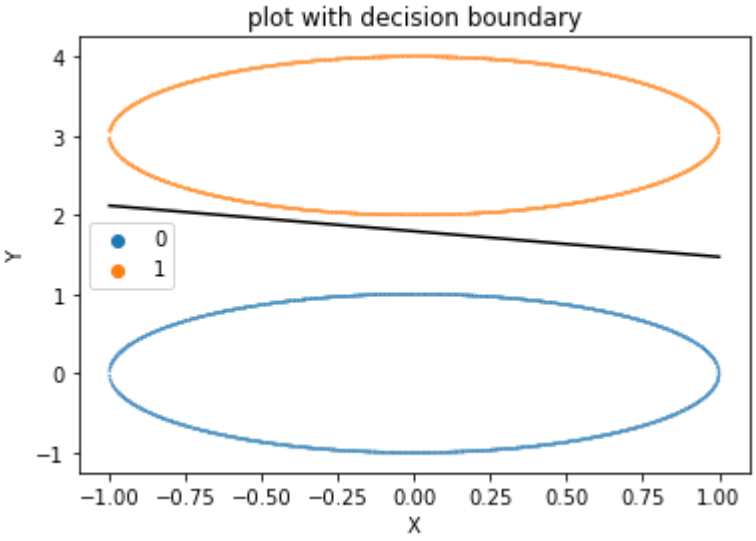
In [ ]:
```
print('Forcing the boundary line to pass through origin')
```

```
plot1(df,p3.weights)
print('Bias can be adjusted with no restriction')
plot1(df,p.weights)
```

```
Forcing the boundary line to pass through origin
```




```
Bias can be adjusted with no restriction
```



Part 5

In [ ]:

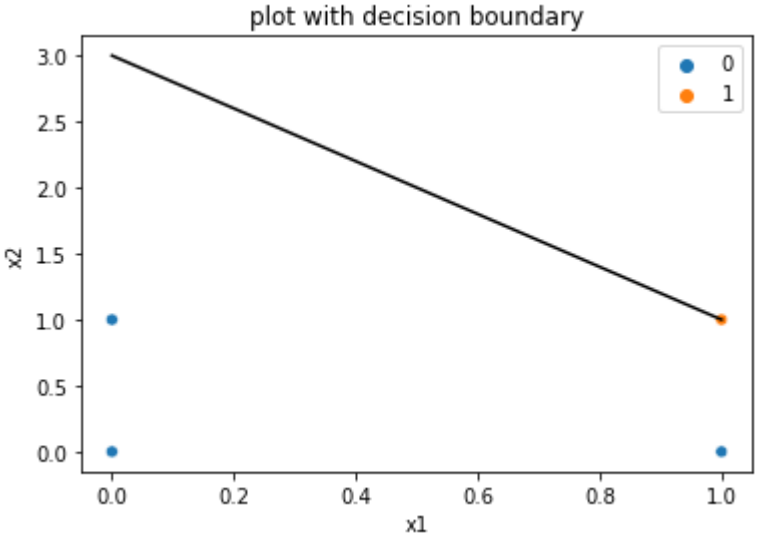
```
xor_d={'x1':[0,0,1,1],'x2':[0,1,0,1],'out':[0,1,1,0]}
and_d={'x1':[0,0,1,1],'x2':[0,1,0,1],'out':[0,0,0,1]}
or_d={'x1':[0,0,1,1],'x2':[0,1,0,1],'out':[0,1,1,1]}
```

In [ ]:

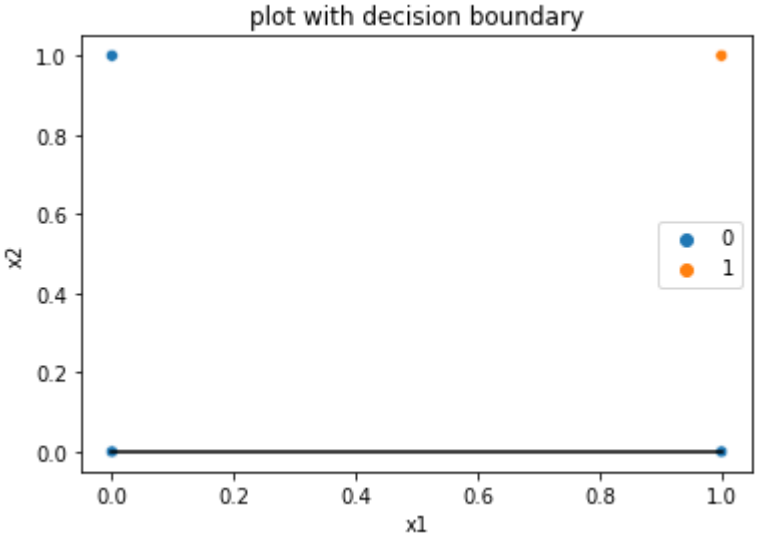
```
and_df=pd.DataFrame(and_d)
print(and_df)
p_and=perceptron()
p_and.train_weights(np.array(and_df),6,1)
print('With learnable bias')
plot2(and_df,p_and.weights)

p_and1=perceptron()
p_and1.train_weights(np.array(and_df),6,0)
```

```
print('With 0 bias')
plot2(and_df,p_and1.weights)
```

```
   x1  x2  out
0   0   0    0
1   0   1    0
2   1   0    0
3   1   1    1
>epoch=0, error=3.000 [-1.0, 2.0, 1.0]
>epoch=1, error=3.000 [-2.0, 2.0, 1.0]
>epoch=2, error=2.000 [-2.0, 2.0, 2.0]
>epoch=3, error=1.000 [-3.0, 2.0, 1.0]
>epoch=4, error=0.000 [-3.0, 2.0, 1.0]
>epoch=5, error=0.000 [-3.0, 2.0, 1.0]
With learnable bias
All the points on/above the decision boundary belongs to class 1
```

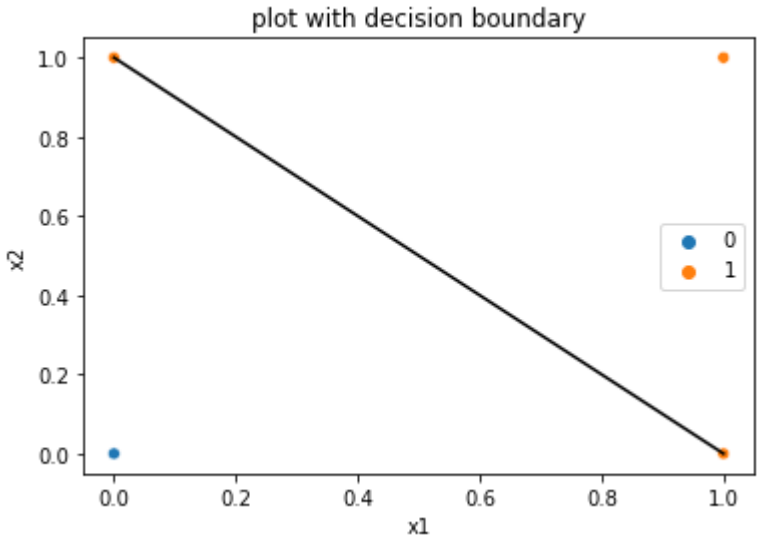



```
>epoch=0, error=3.000 [0.0, 0.0, 0.0]
>epoch=1, error=4.000 [0.0, 0.0, 0.0]
>epoch=2, error=4.000 [0.0, 0.0, 0.0]
>epoch=3, error=4.000 [0.0, 0.0, 0.0]
>epoch=4, error=4.000 [0.0, 0.0, 0.0]
>epoch=5, error=4.000 [0.0, 0.0, 0.0]
With 0 bias
All the points on/above the decision boundary belongs to class 1
```

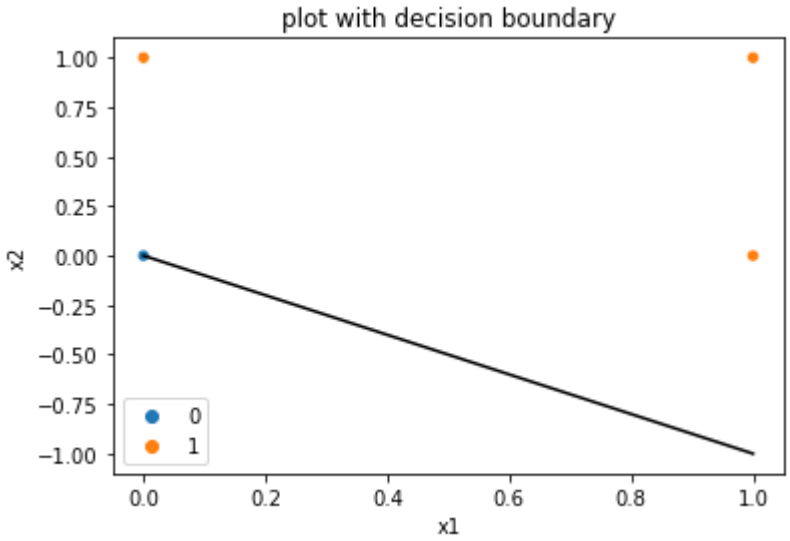


In [ ]:
```
or_df=pd.DataFrame(or_d)
print(or_df)
p_or=perceptron()
#Leanable bias
p_or.train_weights(np.array(or_df),6,1)
plot2(or_df,p_or.weights)
p_or1=perceptron()
p_or1.train_weights(np.array(or_df),6,0)
plot2(or_df,p_or1.weights)
```

```
   x1  x2  out
0   0   0    0
1   0   1    1
2   1   0    1
3   1   1    1
>epoch=0, error=1.000 [-1.0, 1.0, 1.0]
>epoch=1, error=0.000 [-1.0, 1.0, 1.0]
>epoch=2, error=0.000 [-1.0, 1.0, 1.0]
>epoch=3, error=0.000 [-1.0, 1.0, 1.0]
>epoch=4, error=0.000 [-1.0, 1.0, 1.0]
>epoch=5, error=0.000 [-1.0, 1.0, 1.0]
All the points on/above the decision boundary belongs to class 1
```



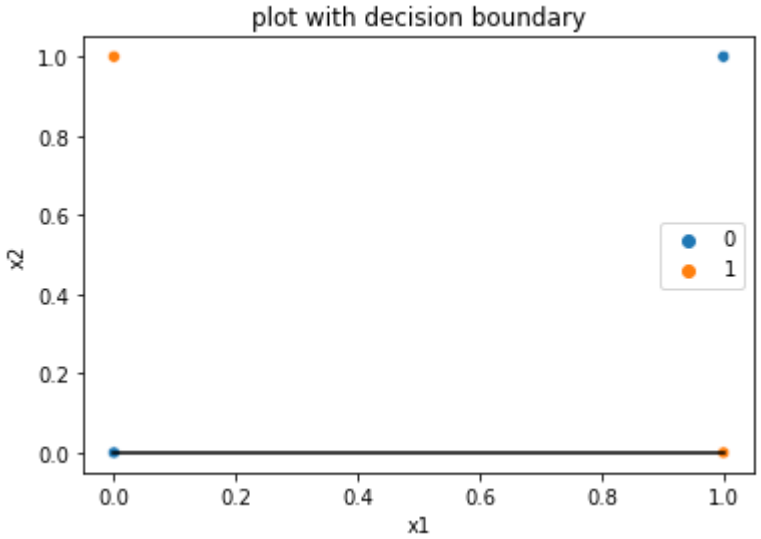

```
>epoch=0, error=1.000 [0.0, 1.0, 1.0]
>epoch=1, error=1.000 [0.0, 1.0, 1.0]
>epoch=2, error=1.000 [0.0, 1.0, 1.0]
>epoch=3, error=1.000 [0.0, 1.0, 1.0]
>epoch=4, error=1.000 [0.0, 1.0, 1.0]
>epoch=5, error=1.000 [0.0, 1.0, 1.0]
All the points on/above the decision boundary belongs to class 1
```



In [ ]:

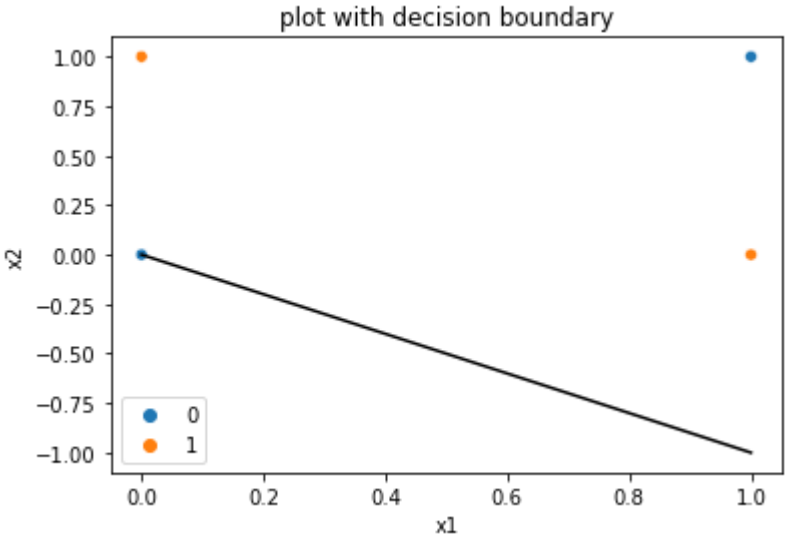
```
xor_df=pd.DataFrame(xor_d)
print(xor_df)
p_xor=perceptron()
p_xor.train_weights(np.array(xor_df),5,1)
plot2(xor_df,p_xor.weights)
p_xor1=perceptron()
p_xor1.train_weights(np.array(xor_df),5,0)
plot2(xor_df,p_xor1.weights)
```

```
   x1  x2  out
0   0   0    0
1   0   1    1
2   1   0    1
3   1   1    0
>epoch=0, error=2.000 [-2.0, 0.0, 0.0]
>epoch=1, error=3.000 [-1.0, 0.0, 0.0]
>epoch=2, error=2.000 [-1.0, -1.0, 0.0]
>epoch=3, error=3.000 [0.0, -1.0, 0.0]
>epoch=4, error=4.000 [0.0, -1.0, 0.0]
All the points on/above the decision boundary belongs to class 1
```




```
>epoch=0, error=2.000 [0.0, 0.0, 0.0]
>epoch=1, error=2.000 [0.0, -1.0, -1.0]
>epoch=2, error=4.000 [0.0, -1.0, -1.0]
```

```
>epoch=3, error=4.000 [0.0, -1.0, -1.0]
>epoch=4, error=4.000 [0.0, -1.0, -1.0]
All the points on/above the decision boundary belongs to class 1
```



Wrong decision boundary for XOR dataset because data can not be separated with one decision boundry

part 6

Given a hyperplane boundary and a point we can classify the point into
class 0 or 1 by putting the point coordinates into the hyperplane equation
by applying the sign(signum) function on the result(let's call it R)

Assumption :: sign function gives 1 when R is >=0 and 0 when R < 0

when the sign function gives 1 the class of the point is 1 else the class of the point is 0.

In [ ]: