Homework 2

Theofanis Nitsos - p3352325

Excercise 1

$$y = 3x_1^2 + 4x_2^2 + 5x_3^2 + 7x_1x_2 + x_1x_3 + 4x_2x_3 - 2x_1 - 3x_2 - 5x_3 + \eta =$$
$$= 3\phi_1(x_1) + 4\phi_2(x_2) + 5\phi_3(x_3) + 7\phi_4(x_1,x_2) + \phi_5(x_1,x_3) + 4\phi_6(x_2,x_3) - 2x_1 - 3x$$
$$= \begin{bmatrix} 3 & 4 & 5 & 7 & 1 & 1 & 4 & -2 & -3 & -5 \end{bmatrix} \begin{bmatrix} \phi_1(x_1) \\ \phi_2(x_2) \\ \phi_3(x_3) \\ \phi_4(x_1,x_2) \\ \phi_5(x_1,x_3) \\ \phi_6(x_2,x_3) \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} + \eta$$

The original space dimension is 3 while the transformed is 9

Excercise 2

$$x_1^2 + 3x_2^2 + 6x_3^2 + x_1x_2 + x_2x_3 = \phi_1(x_1) + 3\phi_2(x_2) + 6\phi_3(x_3) + \phi_4(x_1,x_2) + \phi_5(x_2,x$$
$$= \begin{bmatrix} 1 & 3 & 6 & 1 & 1 \end{bmatrix} \begin{bmatrix} \phi_1(x_1) \\ \phi_2(x_2) \\ \phi_3(x_3) \\ \phi_4(x_1,x_2) \\ \phi_5(x_2,x_3) \end{bmatrix}$$

The original space dimension is 3 while the transformed is 5

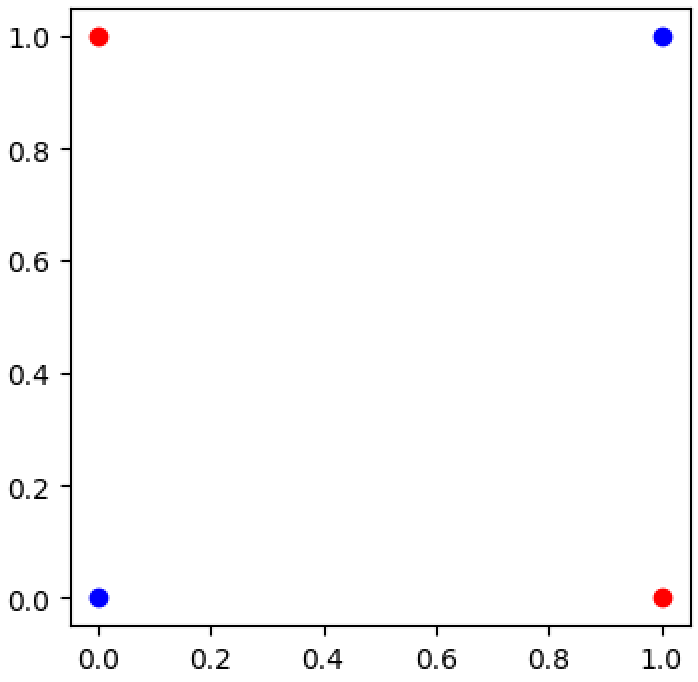
Excercise 3

(a)

In [1]:
```python
import matplotlib.pyplot as plt
import numpy as np

blue_points = [(0, 0), (1, 1)]
x_blue, y_blue = zip(*blue_points)
plt.figure(figsize = (4,4))
plt.scatter(x_blue, y_blue, color='blue', label='Blue Points')
```

```
# Points in red
red_points = [(0, 1), (1, 0)]
x_red, y_red = zip(*red_points)
plt.scatter(x_red, y_red, color='red', label='Red Points')
```

```
Out[1]:  <matplotlib.collections.PathCollection at 0x11f893750>
```



Lets asssume there exists a hyperplane $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$ that can separate the data For the blue points (0,0) & (1,1) it stands:
$\theta_0 > 0$ (1) and
$\theta_0 + \theta_1 + \theta_2 > 0$ (2)

For the red points (0,1) & (1,0) it stands:
$\theta_0 + \theta_2 < 0$ (3) and
$\theta_0 + \theta_1 < 0$ (4)

-(1)+(3) $\Rightarrow \theta_2 < 0$ (5)
-(2)+(4) $\Rightarrow -\theta_2 < 0 \Rightarrow \theta_2 > 0$ (6)

(5), (6) are contradictory, therefore the points are not linearly separable.

(b)

The transformation proposed is the following:

$$\phi(\boldsymbol{x}) = \begin{bmatrix} \phi_1(\boldsymbol{x}) \\ \phi_2(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} exp(-\frac{(\boldsymbol{x}-\boldsymbol{c}_1)^2}{2\sigma^2}) \\ exp(-\frac{(\boldsymbol{x}-\boldsymbol{c}_2)^2}{2\sigma^2}) \end{bmatrix}$$ where

$\boldsymbol{x} = [x_1 \quad x_2]^T$

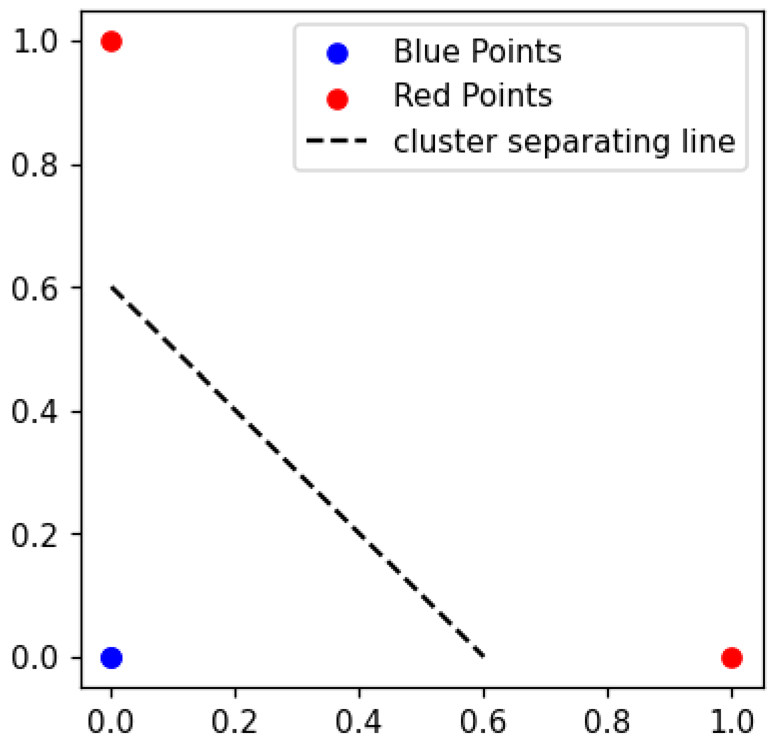
$\boldsymbol{c}_1 = [0 \quad 1]^T$
$\boldsymbol{c}_1 = [1 \quad 0]^T \; \sigma = 0.01$

The classification problem becomes linearly separable (as seen in the figure below) using the line: $\phi_1(\boldsymbol{x}) + \phi_2(\boldsymbol{x}) < (>) 0.6$. There are more lines that can separate the 2 clusters. This is just one possibility.

In [2]:
```
c_1 = np.array([1,0])
c_2 = np.array([0,1])
sigma = 0.01**2

def normal_function(x):
    """This function squares the given number."""
    result_x1 = np.exp(-np.sum((np.array(x) - c_1) ** 2)/(2*sigma)) ** 2
    result_x2 = np.exp(-np.sum((np.array(x) - c_2) ** 2)/(2*sigma)) ** 2
    return result_x1, result_x2

p_1_b = normal_function(blue_points[0])
p_2_b = normal_function(blue_points[1])

p_3_r = normal_function(red_points[0])
p_4_r = normal_function(red_points[1])

print('blue',p_1_b, p_2_b)
print('red',p_3_r, p_4_r)
```

```
blue (0.0, 0.0) (0.0, 0.0)
red (0.0, 1.0) (1.0, 0.0)
```

In [3]:
```
plt.figure(figsize = (4,4), dpi=110)
plt.scatter([p_1_b[0],p_2_b[0]],[p_1_b[1],p_2_b[1]], color='blue', label='Bl
plt.scatter([p_3_r[0],p_4_r[0]],[p_3_r[1],p_4_r[1]], color='red', label='Rec
plt.plot([0, 0.6], [0.6, 0],color='black', linestyle='--', label='cluster se
_=plt.legend()
```



Excercise 4

Class 1 (blue):

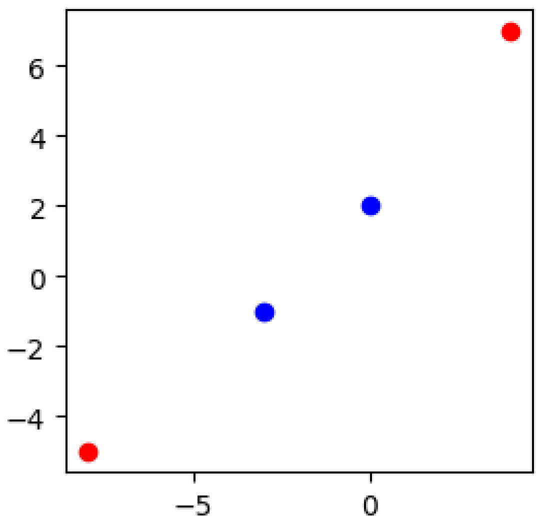
$(x_1, x_2) = (-3, -1)$
$(x_1, x_2) = (0, 2)$

Class 2 (red):

$(x_1, x_2) = (-8, -5)$
$(x_1, x_2) = (4, 7)$

In [4]:
```
blue_points = np.array([[-3,-1],[0,2]])
red_points = np.array([[-8,-5],[4,7]])

plt.figure(figsize=(3,3))
plt.scatter(blue_points[:,0], blue_points[:,1], color='blue', label = 'Blue
plt.scatter(red_points[:,0], red_points[:,1], color='red', label = 'Red Poir
```

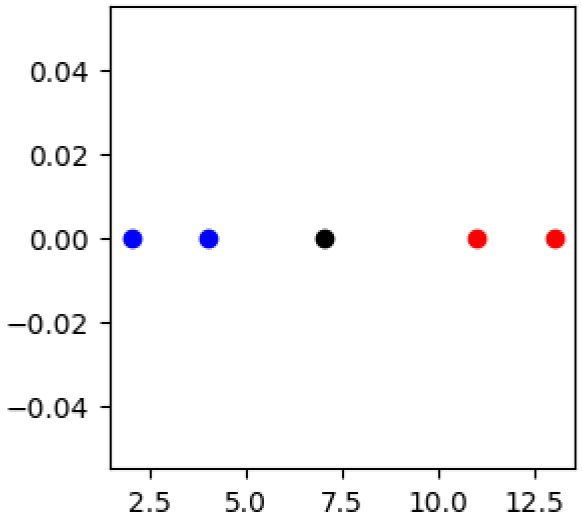
Out[4]:
```
<matplotlib.collections.PathCollection at 0x11f9bf0d0>
```



One possible solution to mapping these points to a new one-dimensional space where the problems becomes linearly separable is using the transformation:
$\phi(\boldsymbol{x}) = |x_1 + x_2|$
The result of this transformation can be seen in the figure below.
The points can be distinguished with the equation below:
$\phi(\boldsymbol{x}) < (>) 7$ That is only one of the equations tha can linearly separate this classification problem.

```
In [5]: x = np.array([0,0]) # we define an arbitrary value as x because we only need
        plt.figure(figsize=(3,3))
        plt.scatter(abs(np.sum(blue_points, axis=1)),x, color='blue', label = 'Blue
        plt.scatter(abs(np.sum(red_points, axis=1)),x, color='red', label = 'Blue Po
        plt.scatter(7,0,color = 'black')
```

```
Out[5]: <matplotlib.collections.PathCollection at 0x11fa353d0>
```



Although the points are depicted in a 2 dimensional figure the y axis is not required. The black dot is the point separating the two classes.

Excercise 5

(a)

Class +1 (blue):

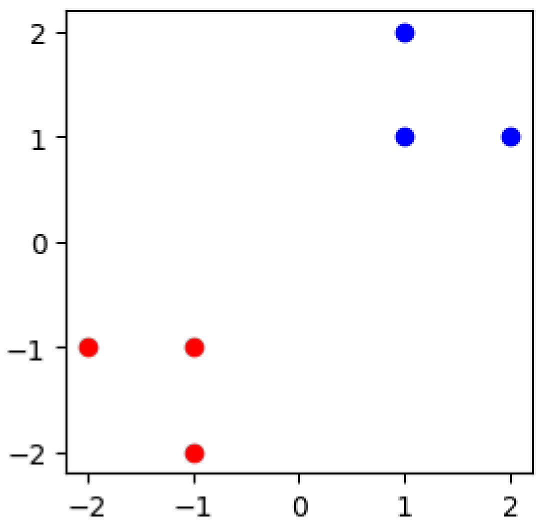
$(x_1, x_2) = (1, 1)$
$(x_1, x_2) = (1, 2)$
$(x_1, x_2) = (2, 1)$

Class -1 (red):

$(x_1, x_2) = (-1, -1)$
$(x_1, x_2) = (-1, -2)$
$(x_1, x_2) = (-2, -1)$

```
In [6]: blue_points = np.array([[1,1],[1,2],[2,1]])
        red_points = np.array([[-1,-1],[-1,-2],[-2,-1]])

        plt.figure(figsize=(3,3))
        plt.scatter(blue_points[:,0], blue_points[:,1], color='blue', label = 'Blue
        plt.scatter(red_points[:,0], red_points[:,1], color='red', label = 'Red Poir
```

```
Out[6]: <matplotlib.collections.PathCollection at 0x11faad3d0>
```



We will apply linear regression to find the hyperplane described by the equation below:

$$0 = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

$(X^T X)\boldsymbol{\theta} = X^T \boldsymbol{y}$, where

$$X = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 1 & 2 & 1 \\ 1 & -1 & -1 \\ 1 & -1 & -2 \\ 1 & -2 & -1 \end{bmatrix}, \boldsymbol{y} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \\ -1 \\ -1 \end{bmatrix}, \boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix},$$

$$\begin{bmatrix} 6 & 0 & 0 \\ 0 & 12 & 10 \\ 0 & 10 & 12 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 8 \\ 8 \end{bmatrix}$$ Solving the system of equations we get:

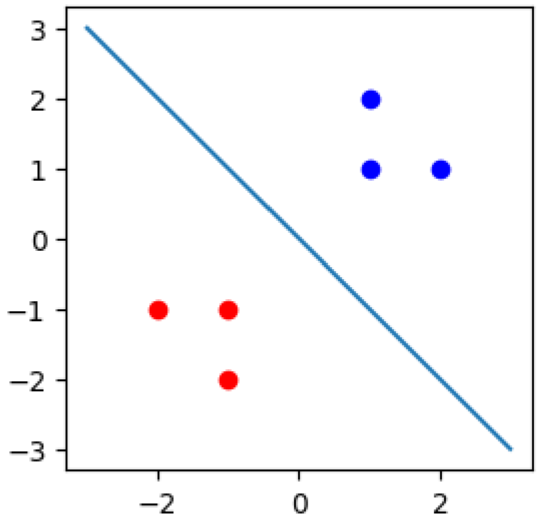
$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.3636 \\ 0.3636 \end{bmatrix}$$

Thus the equation of the line that separates the data is:
$0 = 0.3636x_1 + 0.3636x_2 \Rightarrow x_2 = -x_1$

(b)

In [7]:
```
x_values = np.linspace(-3,3,10)
y_values = - x_values
plt.figure(figsize = (3,3))
plt.scatter(blue_points[:,0], blue_points[:,1], color='blue', label = 'Blue
plt.scatter(red_points[:,0], red_points[:,1], color='red', label = 'Red Poir
plt.plot(x_values, y_values)
```

Out[7]:
```
[<matplotlib.lines.Line2D at 0x11fad2f90>]
```



(c)

No that is not the unique line that separates the data from the two classes.

In the above case we used the least squares criterio to minimise the sum of squared error. In the general case there are other ways to minimize the sum of squared error meaning other hyperplanes could potentially be calculated. In this particular case though the minimisation problem can be solved analytically providing a single solution; a single solution to minimising the sum of squared error.

Excercise 6

(a)

```
In [11]: # Generate data
         mean = [0,0]
         cov = [[1,0],[0,1]]

         X = np.random.multivariate_normal(mean,cov,200)

         #Define the the model
         def f(X,theta):
             return theta[0] + theta[1]*X[:,0] + theta[2]*X[:,1]+ theta[3]*X[:,0]*X[:

         theta = [3,2,1,1]
         y = f(X,theta)
         y = y + np.random.normal(0,np.sqrt(0.05),len(X))
```

(b)

Assuming we only know X and y.
X is defined as $X = [1 \quad \boldsymbol{x}_i], \boldsymbol{x}_i = [x_{i1} \quad x_{i2}]^T$
y is defined as $y = [y_i]$
In order to calculate $\boldsymbol{\theta}$ we will use the equation $\boldsymbol{\theta} = [\theta_0 \quad \theta_1 \quad \theta_2]^T = (X^T X)^{-1} X^T \boldsymbol{y}$

```
In [12]: X = np.c_[np.ones(X.shape[0]), X]
         X_T = np.transpose(X)
         theta = np.dot(np.linalg.inv(np.dot(X_T,X)), np.dot(X_T,y))
         print(theta)
```

```
[3.22045362 1.89879277 1.00843592]
```

Thus minimising the sum of error squares the equation is:
$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

$\hat{y}_i = \theta_0 + \theta_1 x_1 + \theta_2 x_2$

Just a note: everytime this code is executed it will produce a different X matrix, ultimately leading to different parameters $\boldsymbol{\theta}$. For this reason the equation is written with θ in parametric form and not actual values

(c)

```
In [224…: y_estimate = np.dot(X, np.array(theta))

          MSE = 1/200 * sum((y - y_estimate)**2)

          print('MSE is : ',MSE)
```

```
MSE is :  0.8087357746137284
```

(d), (e)

By applying the suggested transformation $\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \phi_3(x) \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix}$ the equations become

$X' = [1 \quad x_{i1} \quad x_{i2} \quad x_{i1} \cdot x_{i2}]$, $\boldsymbol{\theta}' = [\theta_0 \quad \theta_1 \quad \theta_2 \quad \theta_3]^T$

$((X')^T X')\boldsymbol{\theta}' = (X')^T \boldsymbol{y}$

In [13]:
```
X_4 = np.c_[X, X[:, 1] * X[:, 2]]
X_4T = np.transpose(X_4)
theta_4 = np.dot(np.linalg.inv(np.dot(X_4T,X_4)), np.dot(X_4T,y))
print(theta_4)
```

```
[3.02640141 2.00316883 0.98806349 1.01110373]
```

(f)

Repeating the same calculations of (c) for the new model

In [14]:
```
y_4estimate = np.dot(X_4, np.array(theta_4))

MSE = 1/200 * sum((y - y_4estimate)**2)

print('MSE is : ',MSE)
```

```
MSE is :  0.051248619122522154
```

(g)

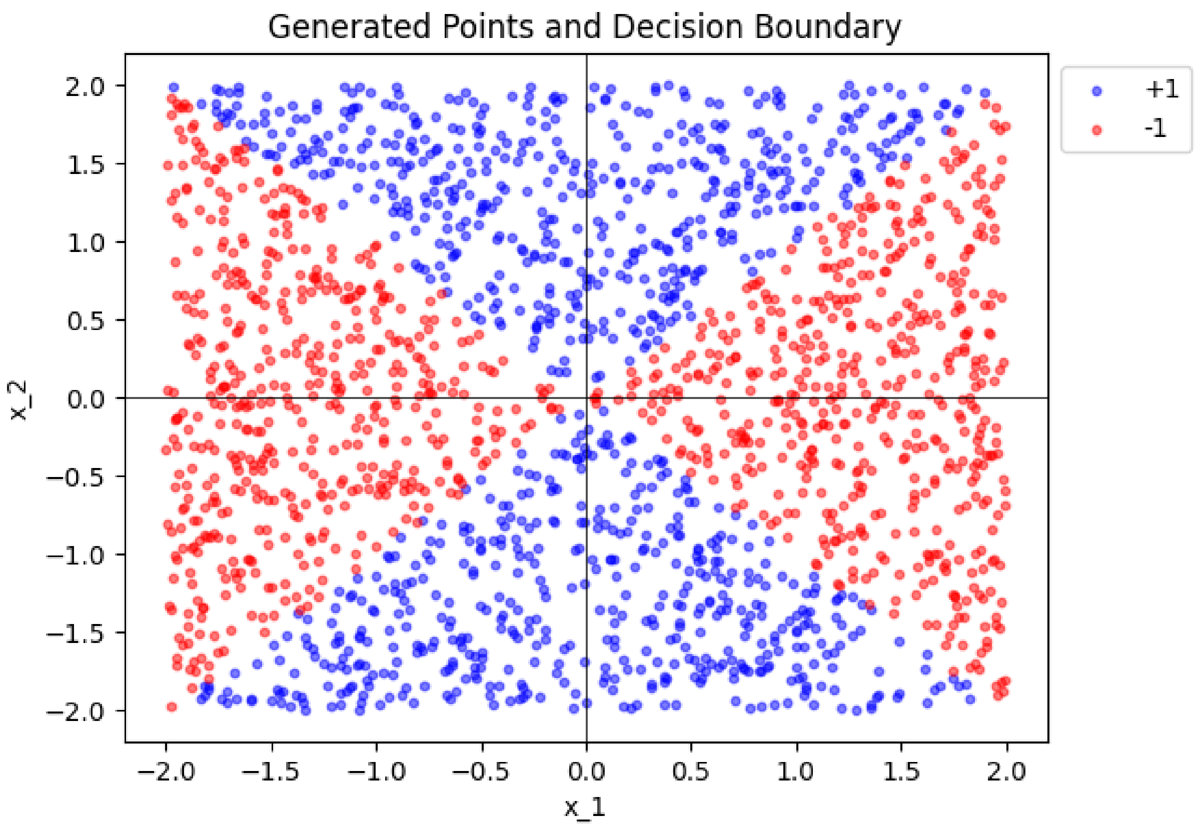
The error in (c) is significantly larger than the one in (f) which is to be expected since the model we used to calculate the corresponding y was significantly different than the one used to generate the data.

The error in (f) is very small since we use the actual model that initially generated the data. It is worth noting that the error is not 0 despite using the "correct" model because of the parameter η added to the model to generate the data. This white noise parameter creates small deviations that are not accounted for in our model causing slightly different parameters $\boldsymbol{\theta}$ and MSE greater than 0.

Excercise 7

(a)

In [15]:
```
# Set the random seed for reproducibility
np.random.seed(42)

# Generate 2000 points in the square area using a uniform distribution
X = np.random.uniform(low=-2, high=2, size=(2000, 2))

# Define the curve x_2^2 - x_1^2 = 0
def decision_boundary(x):
    return x[:, 1]**2 - x[:, 0]**2

# Classify points based on the curve
y = np.where(decision_boundary(X) > 0, 1, -1)

# Plot the points and decision boundary
plt.scatter(X[y == 1, 0], X[y == 1, 1], color='blue', label='+1', s= 10, alp
plt.scatter(X[y == -1, 0], X[y == -1, 1], color='red', label='-1', s= 10, al
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.title('Generated Points and Decision Boundary')
plt.xlabel('x_1')
plt.ylabel('x_2')
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.show()
```



(b), (c)

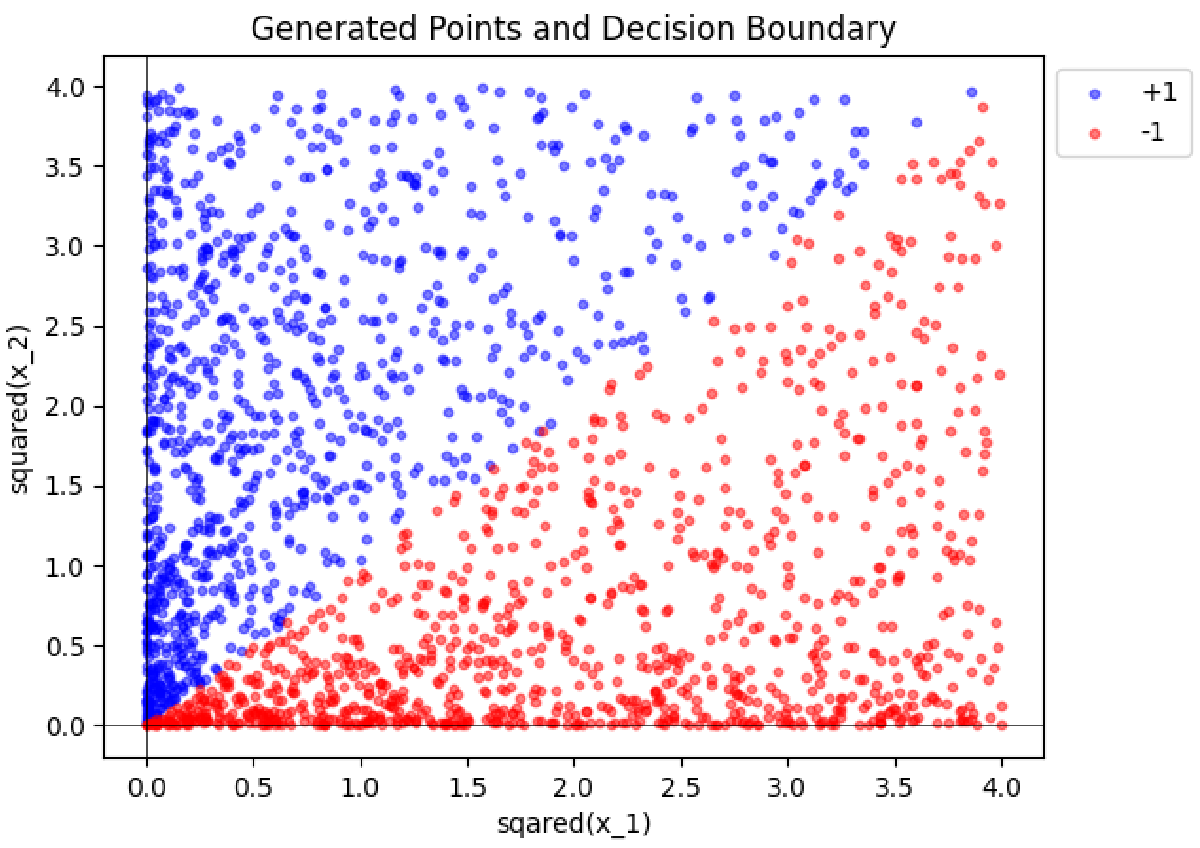
We assume that our only input is :

$X = [\, x_{i1} \quad x_{i2} \,]$ and $\boldsymbol{y} = [y_i]$

we create a new table $X' = [\, x_{i1}^2 \quad x_{i2}^2 \,]$

In [16]:
```
X_transform = np.c_[X[:, 0]**2, X[:, 1]**2]

plt.scatter(X_transform[y == 1, 0], X_transform[y == 1, 1], color='blue', la
plt.scatter(X_transform[y == -1, 0], X_transform[y == -1, 1], color='red', l
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.title('Generated Points and Decision Boundary')
```

```python
plt.xlabel('sqared(x_1)')
plt.ylabel('squared(x_2)')
plt.legend(loc='upper left', bbox_to_anchor=(1, 1))
plt.show()
```



The resulting plot shows that the points can be easily separated from the line equation $x_1' = x_2'$.
This is reasonable since we used the equation $x_2^2 - x_1^2 = 0 \Rightarrow x_2^2 = x_1^2$ to assign the points to different classes. In the figure above the straight line separating the 2 different classes is $x_1' = x_2'$. Since $x_1' = x_1^2$ and $x_2' = x_2^2$ we have essentially the same equation.

The different shape of the figure is due to the transformation from (x_1, x_2) to $(x_1', x_2') = (x_1^2, x_2^2)$ Also there is a bigger concentration of points closer to the axes because of the exponentiation.

(d)

Similar to the problems before we will calculate the $\boldsymbol{\theta}$ using the Least Squares criterion to minimise the sum of errors:
$y = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$ or in matrix form $y = X\boldsymbol{\theta}$
where $X = [1 \quad x_{i1}^2 \quad x_{i2}^2]$, $\boldsymbol{\theta} = [\theta_0 \quad \theta_1 \quad \theta_2]^T$

Using the least squares criterion we will solve the equation:
$(X^T X)\boldsymbol{\theta} = X^T \boldsymbol{y}$

```
In [17]: X_2 = np.c_[np.ones(X_transform.shape[0]), X_transform]
         X_2T = np.transpose(X_2)
         theta_2 = np.dot(np.linalg.inv(np.dot(X_2T,X_2)), np.dot(X_2T,y))
         print(theta_2)
```

```
[ 0.01330263 -0.47417596  0.47039024]
```

The calculated parameters are :

$$\boldsymbol{\theta} = \begin{bmatrix} 0.013 \\ -0.474 \\ 0.47 \end{bmatrix}$$

```
In [ ]:
```