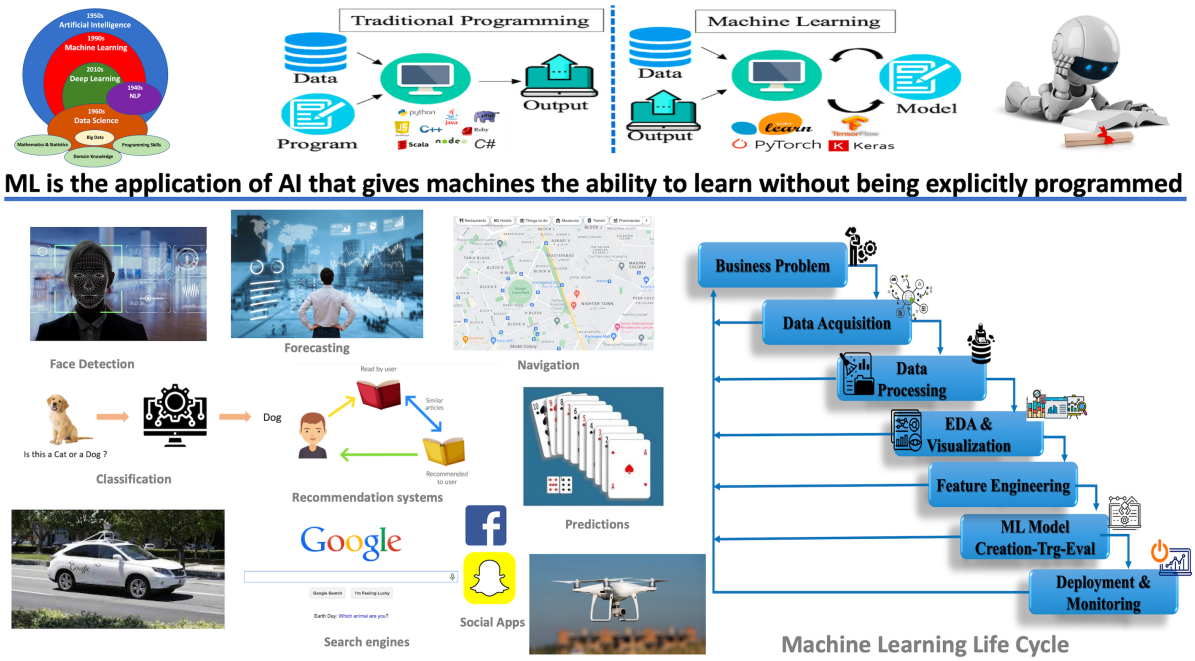
Department of Data Science

Course: Tools and Techniques for Data Science

Instructor: Muhammad Arif Butt, Ph.D.

Lecture 6.21 (Logistic Regression: Part-II)

Open in Colab
(https://colab.research.google.com/github/arifpucit/data-science/blob/master/Section-4-Mathematics-for-Data-Science/Lec-4.1(Descriptive-Statistics).ipynb)

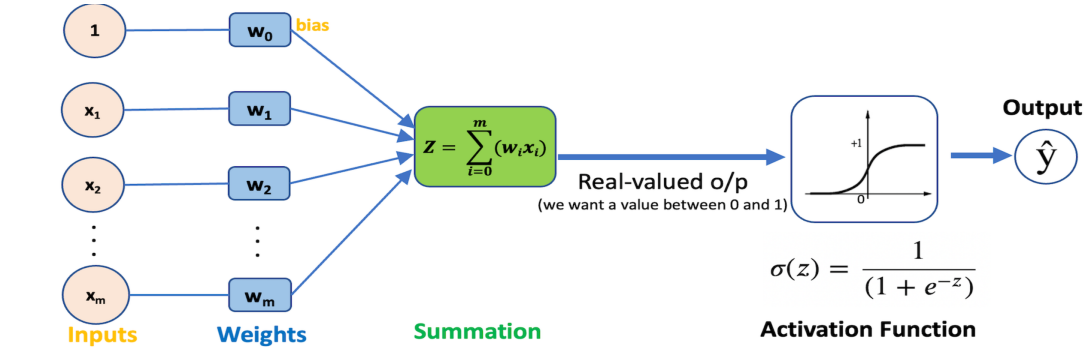


Learning agenda of this notebook

- Logistic Regression using Single Layer Perceptron (Recap)
- Loss/Cost Function for Logistic Regression
  - How to Choose a best model out of many?
  - Output of Sigmoid Function is Probability
  - Choosing Best Model using Maximum Liklihood Estimate
  - Choosing Best Model using Binary Cross Entropy
  - Loss/Error Function for Logistic Regression (Cross-Entropy Loss)
- Gradient Descent for Logistic Regression

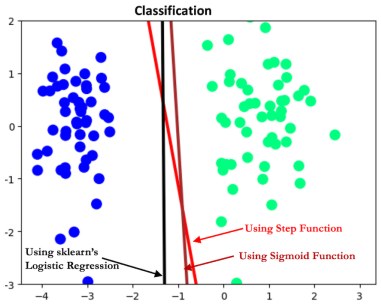
- Gradient Descent for Logistic Regression
  - Cross-Entropy Loss Function
  - Minimizing Cross-Entropy Loss Function using Gradient Descent Algorithm
  - Derivative of Sigmoid Function
  - Derivative of Cross-Entropy-Loss or Log-Loss Function
- Python Code for Logistic Regression using Gradient Descent
  - **Sample Code:**
- Logistic Regression using Scikit-Learn Model
  - **Sample Code:**
- A Comparison between the two

1. Logistic Regression using Single Layer Perceptron (Recap)



$$Z = \sum_{i=0}^{m} (w_i x_i)$$

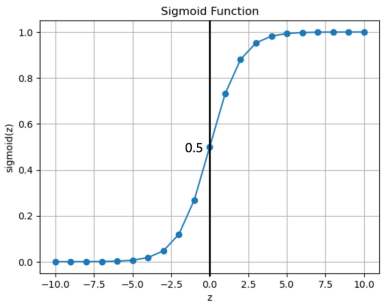
$$\sigma(z) = \frac{1}{(1 + e^{-z})}$$



Algorithm:

1. Initialize weights with random values
2. epochs = 100, $\eta = 0.01$
3. for i in epochs:
4. Pick random $x_i \in P \cup N$
5. $\vec{w}_{new} = \vec{w}_{old} + \eta \times (y_i - \hat{y}_i)\vec{x}_i$

z	Sigmoid(z)
-10	0.000045
-8	0.000335
-6	0.002473
-4	0.017986
-2	0.119203
0	0.500000
2	0.880797
4	0.982014
6	0.997527
8	0.999665
10	0.999955



Test your Concept 1:

- Consider a perceptron with two `inputs` (x1 and x2), `weights` (w1 and w2), and bias (w0). Calculate the weighted sum z and then apply the `step`, and `sigmoid` activation function to calculate $\hat{y}$ for the following dataset:
  - x1 = 0, x2 = 1, w1 = 0.5, w2 = -0.5, w0 = 0.2
  - x1 = 1, x2 = 0, w1 = -0.4, w2 = 0.6, w0 = -0.1

Test your Concept 2:

- Consider a following simplified dataset having four observations with two input features and a binary ground label.

x1	x2	y
9	7	0
4	3	1
5	2	1

x1	x2	y
2	4	1

- Since the ground label of 1^{st} observation/datapoint is zero:
$$\sigma(z) < 0.5 \implies z < 0 \implies w_0 + 9w_1 + 7w_2 < 0$$
- Since the ground label of 2^{nd} observation/datapoint is one:
$$\sigma(z) \geq 0.5 \implies z \geq 0 \implies w_0 + 4w_1 + 3w_2 \geq 0$$
- Since the ground label of 3^{rd} observation/datapoint is one:
$$\sigma(z) \geq 0.5 \implies z \geq 0 \implies w_0 + 5w_1 + 2w_2 \geq 0$$
- Since the ground label of 4^{th} observation/datapoint is one:
$$\sigma(z) \geq 0.5 \implies z \geq 0 \implies w_0 + 2w_1 + 4w_2 \geq 0$$

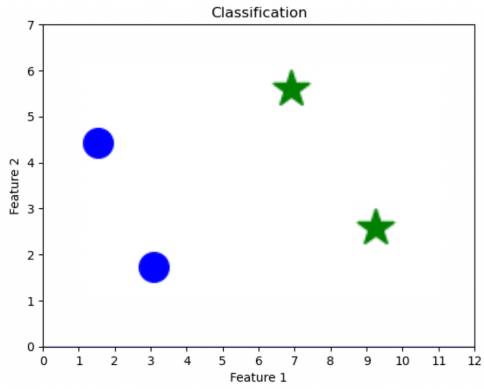
Why to Keep the threshold value to 0.5?

- Changing the threshold can also affect the overall performance of the model. For example, if the threshold is set `too low`, the model may classify too many inputs as belonging to the `positive class`, leading to a high `False Positive` rate. On the other hand, if the threshold is set `too high`, the model may classify too many inputs as belonging to the `negative class`, leading to a high `False Negative` rate.
- In general, the optimal value of the threshold will depend on the specific problem we are trying to solve and the trade-off we are willing to make between false positives and false negatives. It is generally a good idea to experiment with different threshold values and evaluate the performance of the model in order to find the best value for our specific problem.

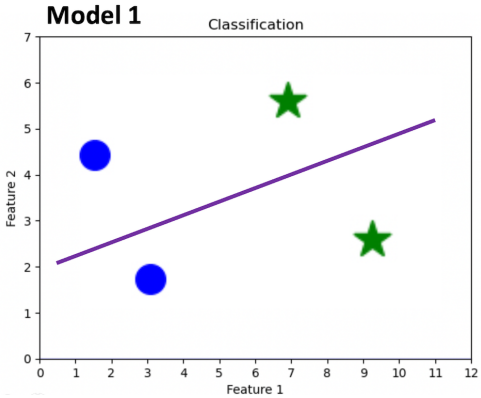
- Question: `How to choose the optimal threshold for a given model?`
- The most common method for classification threshold selection is plotting the `ROC curve`.

2. Loss/Cost Function for Logistic Regression

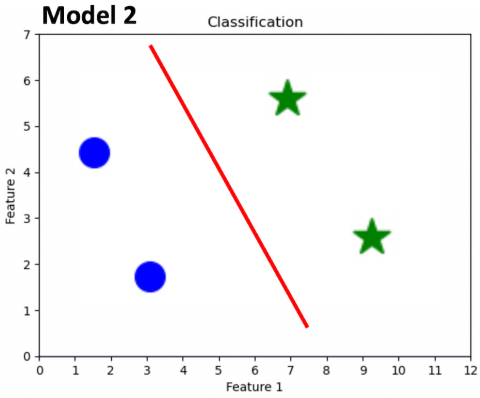
a. Choose the best model



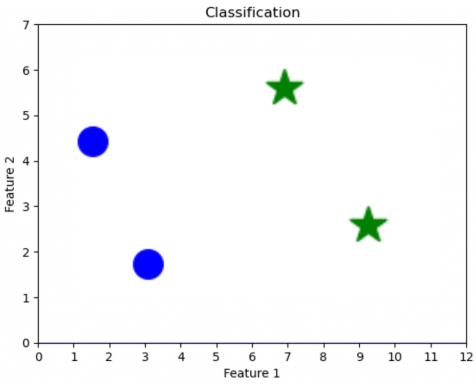
Model 1



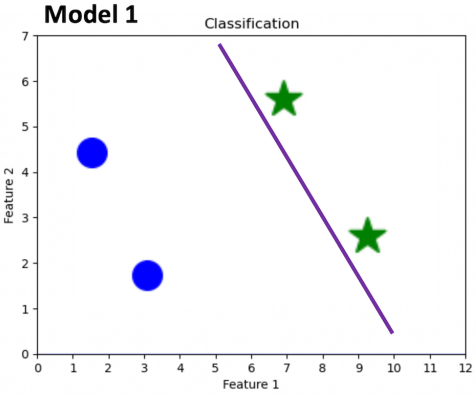
Model 2



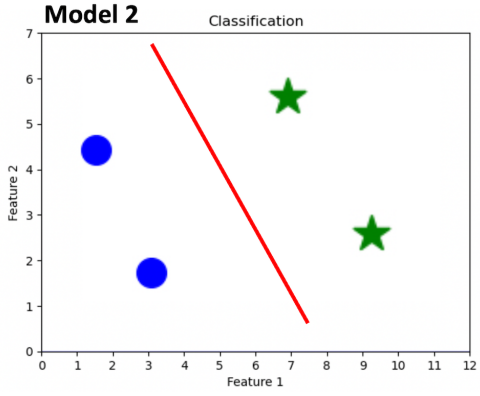
b. Choose the best model



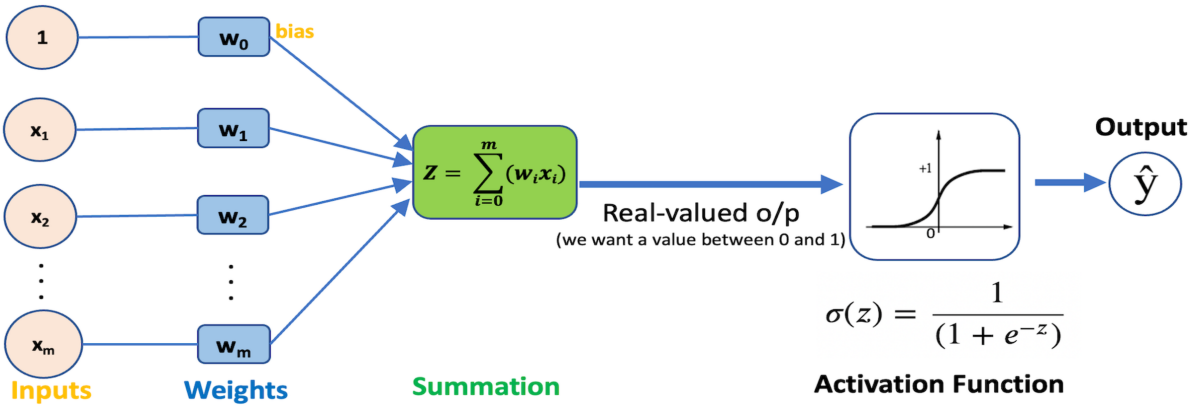
Model 1

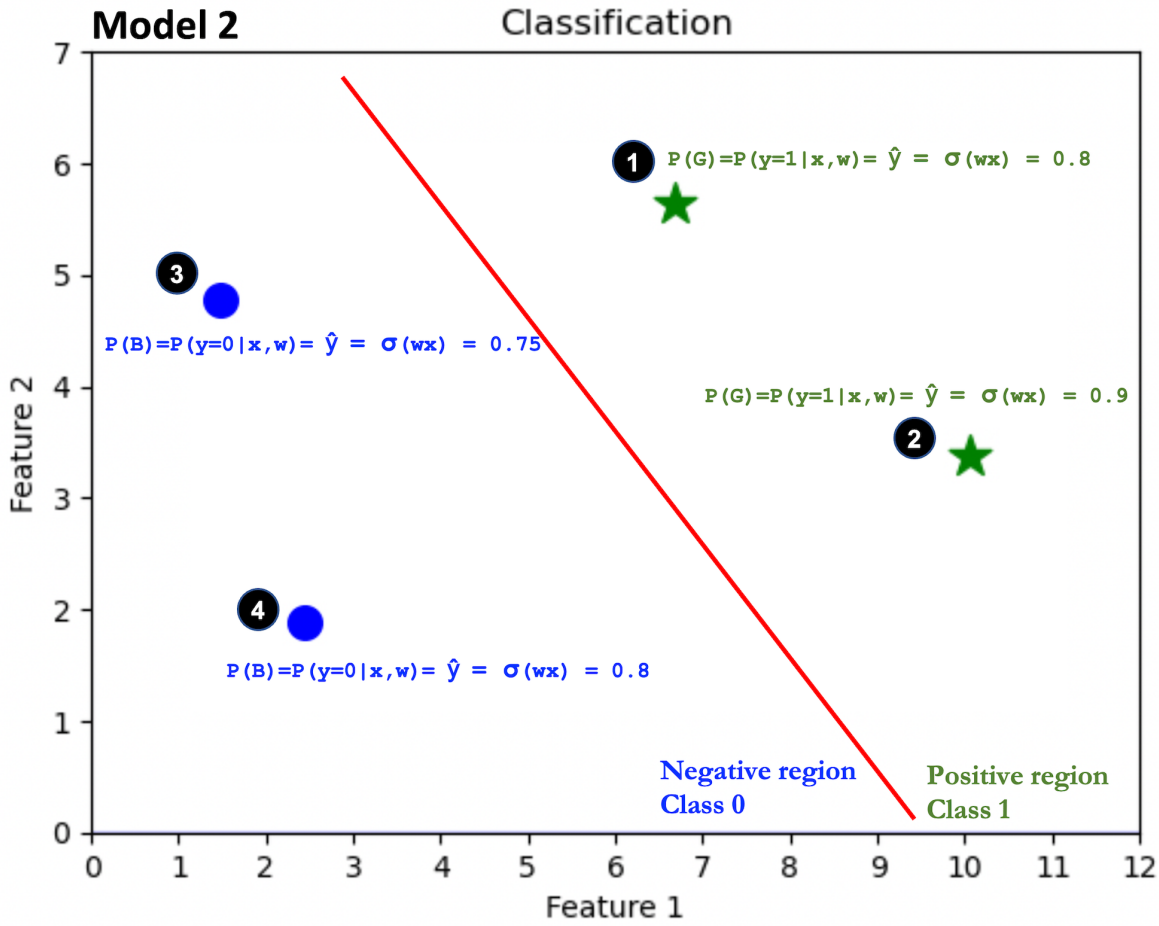


Model 2



c. Output of Sigmoid Function is Probability

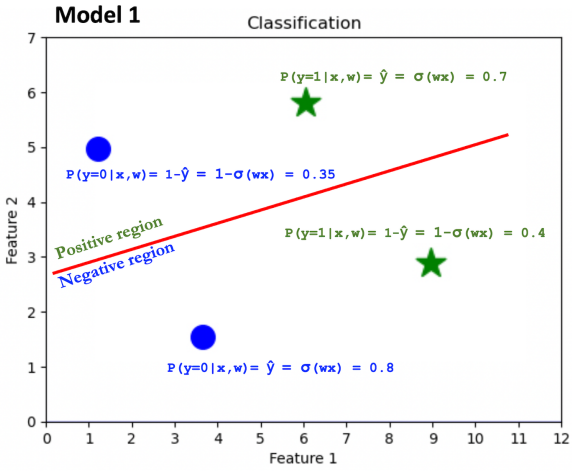


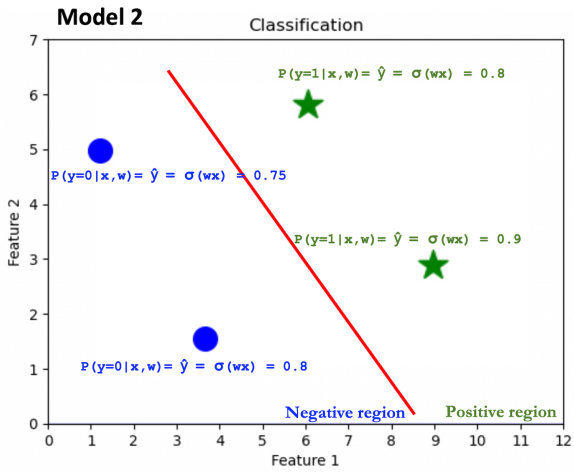


- There is 80% chance of data point 1 of belonging to Positive Class, and 20% chance of belonging to Negative Class
- There is 90% chance of data point 2 of belonging to Positive Class, and 10% chance of belonging to Negative Class
- There is 75% chance of data point 3 of belonging to Negative Class, and 25% chance of belonging to Positive Class
- There is 80% chance of data point 4 of belonging to Negative Class, and 20% chance of belonging to Positive Class

d. Choosing Best Model using Maximum Liklihood Estimate

Maximum Liklihood Estimation is a technique used for estimating the parameters of a given probability distribution, using some observed data.





Note: If a point is misclassified, consider **1-ŷ** and not **ŷ**

$$\text{MLE} = \prod_{i=1}^{n} P(y_i|x,w) = \prod_{i=1}^{n} \sigma(wx) = \prod_{i=1}^{n} \hat{y}_i$$

$\text{MLE} = 0.35 \times 0.8 \times 0.7 \times 0.4 = 0.0784$

$\text{MLE} = 0.75 \times 0.8 \times 0.8 \times 0.9 = 0.432$

- Model having higher MLE is better (We maximize in case of maximum liklihood)

Limitation: We need to compute product of very small numbers, and for a large dataset there will be thousands of datapoints in which case we have to take the product of thousands of very small numbers, which will be a very very small value. Moreover, comparison of two such very small numbers is pointless.

Solution: Convert this product into summation

e. Choosing Best Model using Binary Cross Entropy

The summation of negative logs of maximum liklihood is called cross entropy.

Maximizing likelihood is equivalent to minimizing cross entropy.

Binary Cross Entropy for Model 1:

$$\log(\text{MLE}) = \log(0.35) + \log(0.8) + \log(0.7) + \log(0.4) = (-1.049) + (-$$

- Since the log of any number between 0 and 1 is negative, so to make the result positive, we take summation of negative logarithms

$$-\log(\text{MLE}) = -\log(0.35) - \log(0.8) - \log(0.7) - \log(0.4) = 2.5459$$

Binary Cross Entropy for Model 2:

$$\log(\text{MLE}) = \log(0.75) + \log(0.8) + \log(0.8) + \log(0.9) = (-0.287$$

- Since the log of any number between 0 and 1 is negative, so to make the result positive, we take summation of negative logarithms

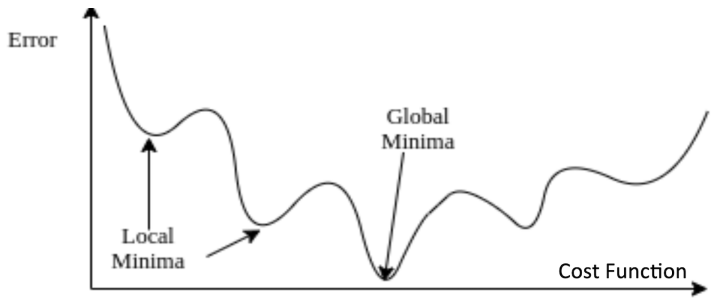
$$-\log(\text{MLE}) = -\log(0.75) - \log(0.8) - \log(0.7) - \log(0.7) = 0.839$$

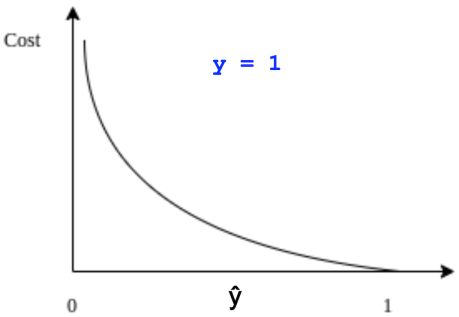
> - Model having lower Cross Entropy is better (We minimize in case of cross entropy)

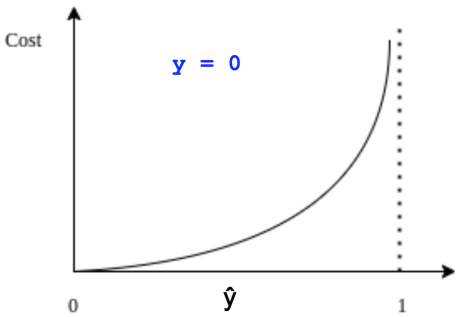
> - In maximum liklihood we select the model having largest value, i.e, we maximize.
> - In cross entropy we select the model having smallest value, i.e, we minimize.

f. Loss/Error Function for Logistic Regression (Cross-Entropy Loss)

- The cost function







summarizes how well the model is performing, i.e., how close the model's predictions are to the actual outputs.

- In linear regression, we use mean squared error (MSE) as the cost function.

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - (\beta_0 + \sum_{i=1}^{m}\beta_i x_i))^2$$

- In logistic regression, since $\hat{y} = \sigma(wx))$, so using MSE as loss function for logistic regression might give a wavy, non-convex graph; containing many local optima. Thus, finding an optimal solution with the gradient descent method is not possible.
- Therefore, we use a logarithmic function to represent the cost of logistic regression. It is guaranteed to be convex for all input values, containing only one minimum, allowing us to run the gradient descent algorithm.
- When dealing with a binary classification problem, the logarithmic cost of error depends on the value of y. We can define the cost for two cases separately:

$$\text{Loss}(\hat{y}, y) = \begin{cases} -log(\hat{y}), & \text{if} \quad y = 1 \\ -log(1-\hat{y}), & \text{if} \quad y = 0 \end{cases}$$

$$\text{Loss}(\hat{y}, y) = -y\log\hat{y} - (1-y)\log(1-\hat{y})$$

For `n` observations, we can calculate the cost as:

$$\text{J(w)} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i\log\hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right]$$

- When the observation belong to class 1, the second term of this formula becomes zero, else first becomes zero.

> - **Now we need to find out the values of weights for which this function gives us the minimum error**

3. Gradient Descent for Logistic Regression

> Gradient Descent is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function.

a. Cross-Entropy Loss Function:

$$\text{J(w)} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i\log\hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right.$$

$$\text{J(w)} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i\log\sigma(z) + (1-y_i)\log(1-\right.$$

$$J(w) = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\sigma(\sum(wx)) + (1-y_i)\log(1-\sigma(\sum(wx))\right]$$

b. Minimizing Cross-Entropy Loss Function using Gradient Descent Algorithm

$$Repeat:$$

$$w_j \leftarrow w_j - \eta\frac{\partial}{\partial w_j}J(w)$$

- Where,

$$\frac{\partial}{\partial w}J(w) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)x$$

- Plugging this derivative of Cross-Entropy Loss function into the gradient descent function gives us the following:

1. initialize random values of $\vec{w}$
2. epochs = 100
3. $\eta = 0.01$
4. for i in epochs:
5. $\quad \vec{w}_{old} \leftarrow \vec{w}_{old} - \eta \times \frac{1}{n} \times \text{np.dot}[(\hat{y} - y), x]$

- Surprisingly, the update rule is the same as the one derived by using the sum of the squared errors in linear regression.
  - For linear regression: $\hat{y} = \sum(wx)$
  - For logistic regression: $\hat{y} = \sigma(z) = \sigma(\sum(wx))$

c. Derivative of Sigmoid Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

$$\frac{d}{dz}\sigma(z) = \frac{d}{dz}\left[\frac{1}{1+e^{-z}}\right]$$

$$= \frac{d}{dz}\left(1+\mathrm{e}^{-z}\right)^{-1} \qquad \cdots (i)$$

$$= -(1+e^{-z})^{-2}\frac{d}{dz}\left(1+\mathrm{e}^{-z}\right) \qquad \cdots (ii)$$

$$= -(1+e^{-z})^{-2}[\frac{d}{dz}(1) + \frac{d}{dz}\left(\mathrm{e}^{-z}\right)] \qquad \cdots (iii)$$

$$= -(1+e^{-z})^{-2}[0 + e^{-z}\frac{d}{dz}(-z)] \qquad \cdots (iv)$$

$$= -(1+e^{-z})^{-2}(-e^{-z})$$

$$= \frac{e^{-z}}{(1+e^{-z})^2} \qquad \cdots (v)$$

$$= \frac{1}{1+e^{-z}} \cdot \frac{e^{-z}}{1+e^{-z}} \qquad \cdots (vi)$$

$$= \frac{1}{1+e^{-z}} \cdot \frac{(1+e^{-z})-1}{1+e^{-z}} \qquad \cdots (vii)$$

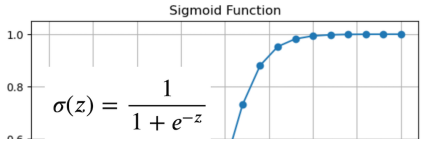
$$= \frac{1}{1+e^{-z}} \cdot \left(\frac{1+e^{-z}}{1+e^{-z}} - \frac{1}{1+e^{-z}}\right)$$

$$= \frac{1}{1+e^{-z}} \cdot \left(1 - \frac{1}{1+e^{-z}}\right) \qquad \cdots (viii)$$

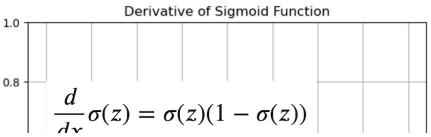
$$= \sigma(z) \cdot (1-\sigma(z))$$

$$\frac{d}{dz}\sigma(z) \quad = \quad \sigma(z) \cdot (1-\sigma(z))$$

x	Sigmoid(x)
-10	0.000045
-8	0.000335
-6	0.002473
-4	0.017986



x	sigmoid(x)'
-10	0.000045
-8	0.000335
-6	0.002467
-4	0.017663



d. Derivative of Cross-Entropy-Loss or Log-Loss Function

$$J = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right]$$

$$\frac{\partial J}{\partial w} = \frac{\partial}{\partial w}\left[-\frac{1}{n}\sum_{i=1}^{n}\left[y_i \log\hat{y}_i + (1-y_i)\log(1-\hat{y}_i)\right]\right]$$

$$\frac{\partial J}{\partial w} = -\frac{1}{n}\sum_{i=1}^{n}\left[\frac{\partial}{\partial w}(y_i \log\hat{y}_i) + \frac{\partial}{\partial w}(1-y_i)\log(1-\hat{y}_i)\right]$$

$= \frac{\partial}{\partial w}(y\log\hat{y})$
$= y\frac{\partial}{\partial w}(\log\hat{y})$

We know that $\frac{d}{dx}logx = \frac{1}{x}\frac{d}{dx}(x)$, therefore
$= y \cdot \frac{1}{\hat{y}} \cdot \frac{\partial}{\partial w}\hat{y}$

We know that $\hat{y} = \sigma(wx)$, therefore
$= y \cdot \frac{1}{\hat{y}} \cdot \frac{\partial}{\partial w}\left[\sigma(wx)\right]$

We know that $\sigma(x)' = \sigma(x)\cdot(1-\sigma(x))$, therefore
$= y \cdot \frac{1}{\hat{y}} \cdot \sigma(wx)\cdot(1-\sigma(wx)) \cdot \frac{\partial}{\partial w}(wx)$

$= y \cdot \frac{1}{\hat{y}} \cdot \sigma(wx)\cdot(1-\sigma(wx)) \cdot x$
$= y \cdot \frac{1}{\hat{y}} \cdot \hat{y}\cdot(1-\hat{y}) \cdot x$
$= y\cdot(1-\hat{y})\cdot x$
$= y\cdot(1-\hat{y})\cdot x$

$= \frac{\partial}{\partial w}(1-y)\log(1-\hat{y})$
$= (1-y)\cdot\frac{\partial}{\partial w}\log(1-\hat{y})$

We know that $\frac{d}{dx}logx = \frac{1}{x}\frac{d}{dx}(x)$, therefore

$= (1-y)\cdot\frac{1}{1-\hat{y}} \cdot \frac{\partial}{\partial w}(1-\hat{y})$
$= (1-y)\cdot\frac{1}{1-\hat{y}} \cdot \frac{\partial}{\partial w}(-\hat{y})$
$= -(1-y)\cdot\frac{1}{1-\hat{y}} \cdot \frac{\partial}{\partial w}(\hat{y})$

We know that $\hat{y} = \sigma(wx)$, therefore
$= -(1-y)\cdot\frac{1}{1-\hat{y}} \cdot \frac{\partial}{\partial w}\sigma(wx)$

We know that $\sigma(x)' = \sigma(x)\cdot(1-\sigma(x))$, therefore
$= -(1-y)\cdot\frac{1}{1-\hat{y}} \cdot \sigma(wx)\cdot(1-\sigma(wx)) \cdot \frac{\partial}{\partial w}(wx)$
$= -(1-y)\cdot\frac{1}{1-\hat{y}} \cdot \sigma(wx)\cdot(1-\sigma(wx)) \cdot x$
$= -(1-y)\cdot\frac{1}{1-\hat{y}} \cdot \hat{y}\cdot(1-\hat{y}) \cdot x$
$= -\hat{y}\cdot(1-y)\cdot x$

$$\frac{\partial J}{\partial w} = -\frac{1}{n}\sum_{i=1}^{n}\left[\frac{\partial}{\partial w}(y_i \log\hat{y}_i) + \frac{\partial}{\partial w}(1-y_i)\log(1-\hat{y}_i)\right]$$

$$\frac{\partial J}{\partial w} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i(1-\hat{y}_i)x - \hat{y}_i(1-y_i)x\right] \quad \cdots (ii)$$

$$\frac{\partial J}{\partial w} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i(1-\hat{y}_i) - \hat{y}_i(1-y_i)\right]x$$

$$\frac{\partial J}{\partial w} = -\frac{1}{n}\sum_{i=1}^{n}\left[y_i - y_i\hat{y}_i - \hat{y}_i + y_i\hat{y}_i\right]x$$

$$\frac{\partial J}{\partial w} = -\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)x$$

$$\frac{\partial J}{\partial w} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)x$$

$$\frac{\partial J}{\partial w} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)x$$

4. Python Code for Logistic Regression using Gradient Descent

Generate Dataset

In [1]:
```
from sklearn.datasets import make_classification
import numpy as np
X, y = make_classification(n_samples=100,
                           n_features=2,
                           n_classes=2,
                           n_clusters_per_class=1,
                           class_sep=20,
                           random_state=41,
                           n_informative=1,
                           n_redundant=0,
                           hypercube=False)
X.shape, y.shape
```

Out[1]:
```
((100, 2), (100,))
```

In [2]:
```
X[0:5]
```

Out[2]:
```
array([[ 0.51123145, -0.11697552],
       [ 0.06316371, -0.73115232],
       [-0.0425064 , -0.7081059 ],
       [-3.2891569 , -2.01199214],
       [ 0.1111445 ,  1.63493163]])
```

In [3]:
```
y
```

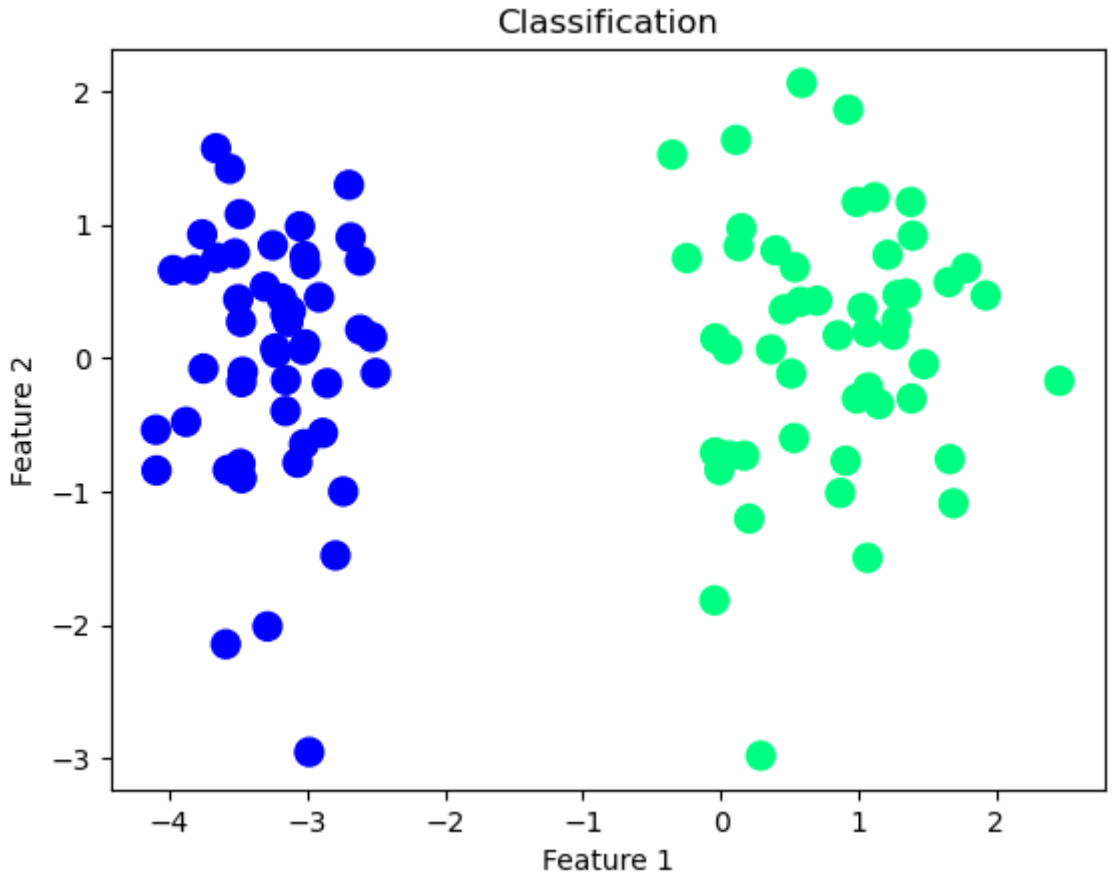
Out[3]:
```
array([1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0,
0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0,
1,
       0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1,
1,
       0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0,
0,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1])
```

Visualize Dataset

In [4]:
```
import matplotlib.pyplot as plt

plt.scatter(X[:,0],X[:,1], c=y, cmap='winter',  s=100)

plt.title("Classification")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.show();
```



Training Algorithm using Gradient Descent

$$\vec{w}_{new} = \vec{w}_{old} - \eta \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i$$

In [5]:
```
def gradient_descent(X,y):
    X = np.insert(X, 0, 1, axis=1)
    weights = np.ones(X.shape[1])
    lr = 0.1
    epochs = 10000
    for i in range(epochs):
        y_hat = sigmoid(np.dot(X, weights))
        weights = weights - lr * (1/X.shape[0]) *  np.dot( (y_hat - y),
    return weights[0],weights[1:]
```

```
def perceptron(X,y):
    X = np.insert(X, 0, 1, axis=1)
    weights = np.ones(X.shape[1])
    lr = 0.1
    epochs = 10000
    for i in range(epochs):
        j = np.random.randint(0,100)
        y_hat = sigmoid(np.dot(X[j], weights) )
        weights = weights + lr*(y[j]-y_hat) * X[j]
    return weights[0],weights[1:]
```

In [6]:
```
def sigmoid(z):
    return 1/(1 + np.exp(-z))
```

In [7]:
```
bias , weights = gradient_descent(X, y)
print("w0: ", bias)
print("w1 and w2: ", weights)
```

```
w0:  4.929952752929981
w1 and w2:  [4.21457057 0.16023185]
```

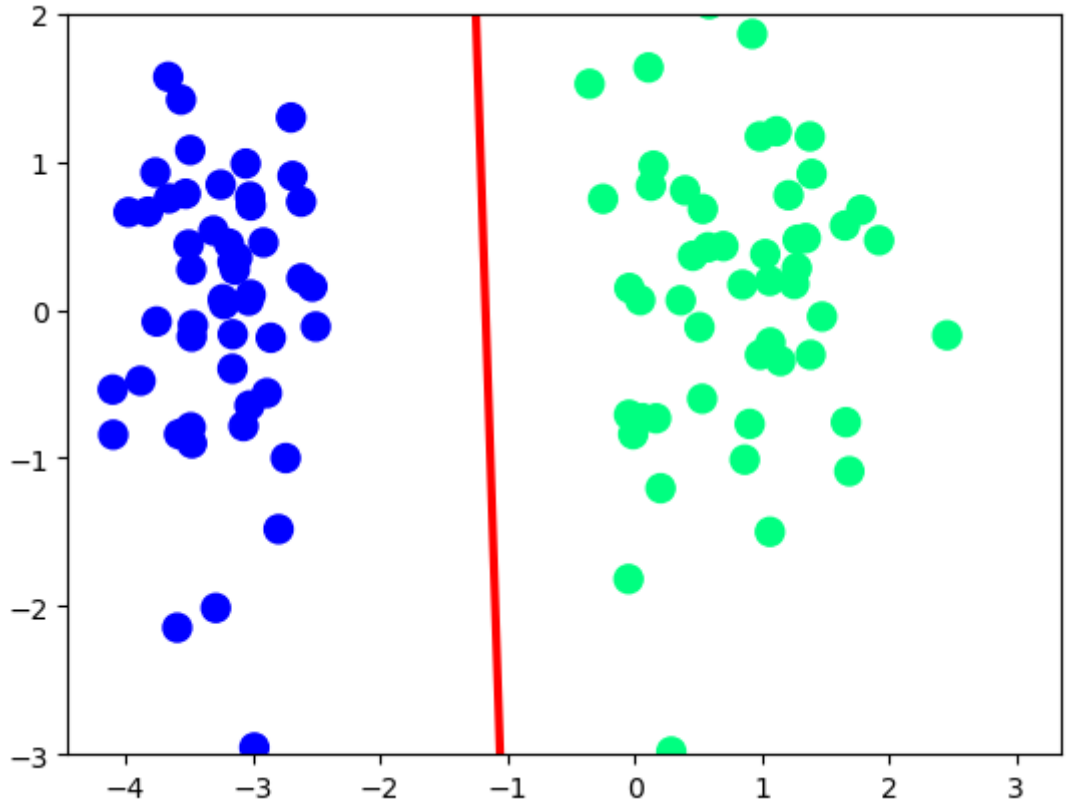
Visualize Result

- $Ax + By + C = 0$
- Slope $= -\frac{A}{B}$
- y-intercept $= -\frac{C}{B}$

In [8]:
```
m = -(weights[0]/weights[1])
b = -(bias/weights[1])

x_input = np.linspace(-3,3,100)
y_input = m*x_input + b

plt.plot(x_input, y_input, color='red', linewidth=3)
plt.scatter(X[:,0],X[:,1], c=y, cmap='winter',  s=100)
plt.ylim(-3,2)
plt.show();
```



Comparing Results with Scikit-Learn's Logistic Regression Model

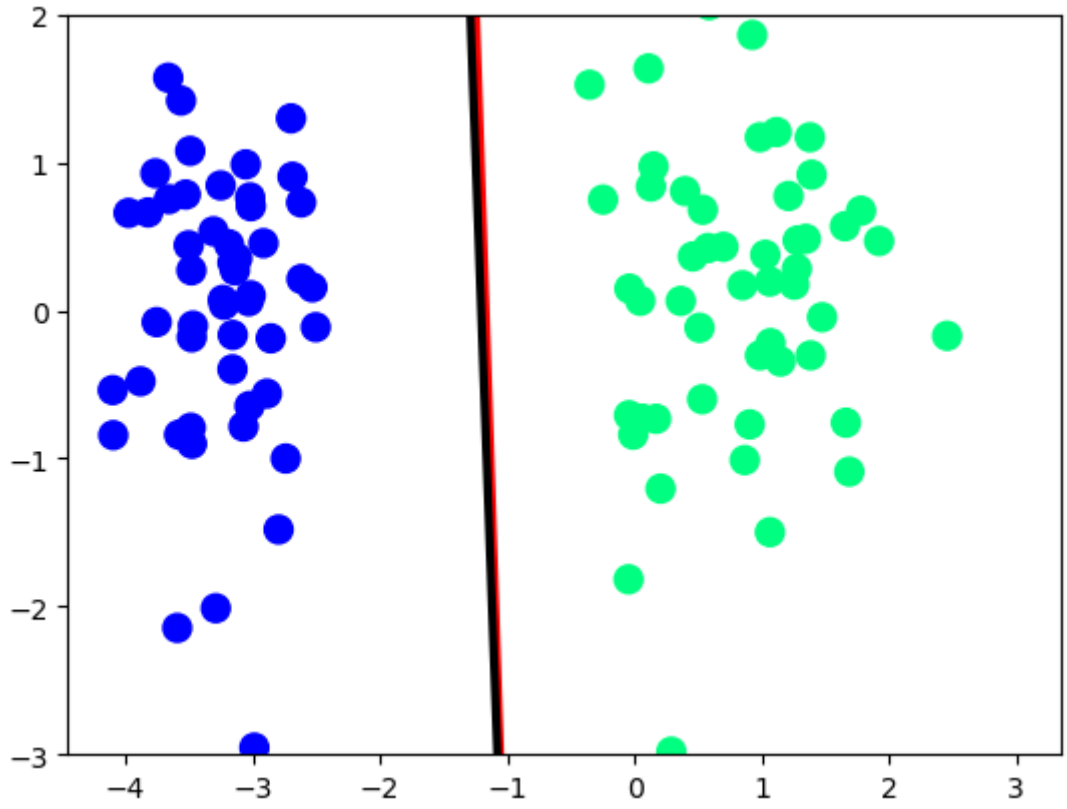
In [9]:
```
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings("ignore")

model = LogisticRegression(penalty='none',solver='sag')
model.fit(X, y)

m = -(model.coef_[0][0]/model.coef_[0][1])
b = -(model.intercept_/model.coef_[0][1])

x_input1 = np.linspace(-3,3,100)
y_input1 = m*x_input + b

plt.plot(x_input,  y_input,  color='red',   linewidth=3)
plt.plot(x_input1, y_input1, color='black', linewidth=3)
plt.scatter(X[:,0], X[:,1], c=y, cmap='winter', s=100)
plt.ylim(-3,2)
plt.show();
```

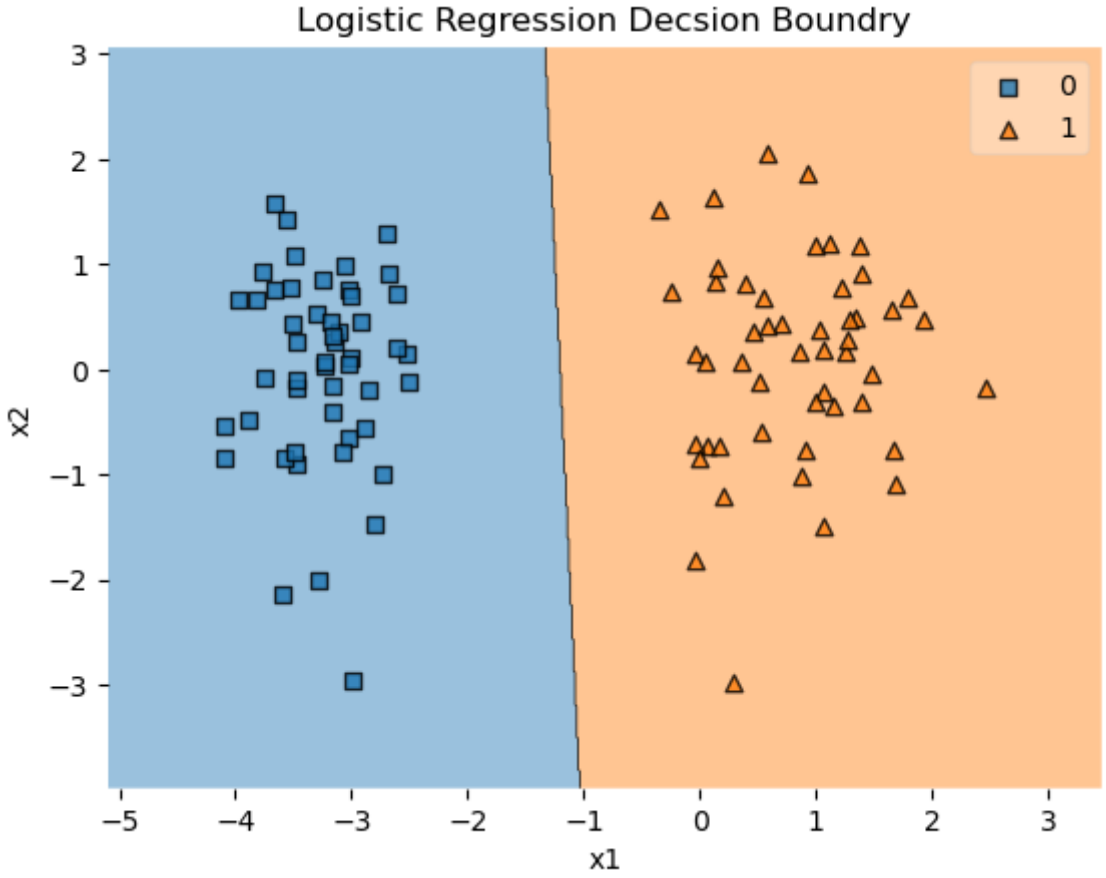


Visualize Decision Boundry using mlxtend Library:

In [ ]:
```
# https://rasbt.github.io/mlxtend/
#import sys
#!{sys.executable} -m pip install mlxtend --upgrade --no-deps
```

In [10]:
```
from mlxtend.plotting import plot_decision_regions

plot_decision_regions(X=X, y=y, clf=model)
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Logistic Regression Decsion Boundry')
plt.show();
```



Plot Sigmoid:

In [ ]:
```
# plot sigmoid curve using a example
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

x = np.arange(-10,11)
sigmoid = 1/(1+np.exp(-x))
plt.plot(x,sigmoid, marker='o')
plt.title("Sigmoid Function")
plt.xlabel("z")
plt.ylabel("sigmoid(z)")
plt.grid(True)
plt.show()
df = pd.DataFrame(data = {"z":x, "Sigmoid(z)":sigmoid})
df[0:21:2]
```

Plot Derivative of Sigmoid:

In [ ]:

```
deri = (1/(1+np.exp(-x)))*(1- 1/(1+np.exp(-x)))
plt.title("Derivative of Sigmoid Function")
plt.plot(x,deri, marker='o')
plt.xlabel("x")
plt.ylabel("sigmoid(x)'")
plt.grid(True)
plt.ylim(0,1)
plt.show()
```