Logistic Regression

- It's a classification algorithm, that is used where the response variable is categorical. The idea of Logistic Regression is to find a relationship between features and probability of particular outcome.
  - E.g. When we have to predict if a student passes or fails in an exam when the number of hours spent studying is given as a feature, the response variable has two values, pass and fail.
- If the probability is more than 50%, it assigns the value in that particular class else if the probability is less than 50%, the value is assigned to the other class. Therefore, we can say that logistic regression acts as a binary classifier.

Working of a Logistic Model

For linear regression, the model is defined by: $y = \beta_0 + \beta_1 x$ - (i)

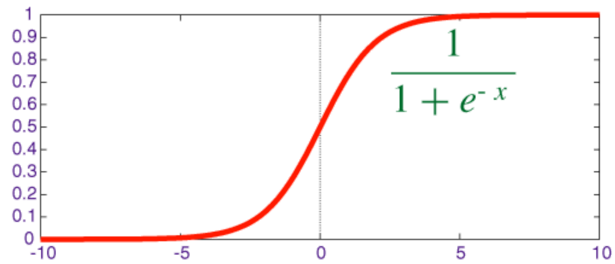
and for logistic regression, we calculate probability, i.e. y is the probability of a given variable x belonging to a certain class. Thus, it is obvious that the value of y should lie between 0 and 1.

But, when we use equation(i) to calculate probability, we would get values less than 0 as well as greater than 1. That doesn't make any sense . So, we need to use such an equation which always gives values between 0 and 1, as we desire while calculating the probability.

So here we Use Sigmoid Function

Sigmoid function

We use the sigmoid function as the underlying function in Logistic regression. Mathematically and graphically, it is shown as:



Why do we use the Sigmoid Function?

1) The sigmoid function's range is bounded between 0 and 1. Thus it's useful in calculating the probability for the Logistic function.

2) It's derivative is easy to calculate than other functions which is useful during gradient descent calculation.

3) It is a simple way of introducing non-linearity to the model.

Now Logistic function On Sigmoid Function

Logistic function

we Know sigmoid function = $1/1+e^{-x}$

when it use in logistic it becomes

$$P(x) = 1/1+e^{-(mx+b)}$$

$$\Rightarrow P(x) = \frac{e^{(mx+b)}}{1+e^{(mx+b)}} \quad \text{--- (1)}$$

let do manupulatn with eqn (1)

$$1 - P(x) = 1 - \frac{e^{h(\theta)}}{1+e^{h(\theta)}} \qquad \because h(\theta) = mx+b$$

$$\Rightarrow 1 - P(x) = \frac{1+e^{h(\theta)} - e^{h(\theta)}}{1+e^{h(\theta)}}$$

$$\Rightarrow 1 - P(x) = 1/1+e^{h(\theta)} = \frac{1}{1+e^{(mx+b)}} \quad \text{--- (2)}$$

Now divide (1) by (2). So we get

$$\frac{P(x)}{1-P(x)} = e^{(mx+b)}$$

→ then take log both side

$$\boxed{\log\left(\frac{P(x)}{1-P(x)}\right) = mx+b}$$

this is the logistic function.

→ by this procedure we can calcute logistic function from sigmoid function.

Logit Function

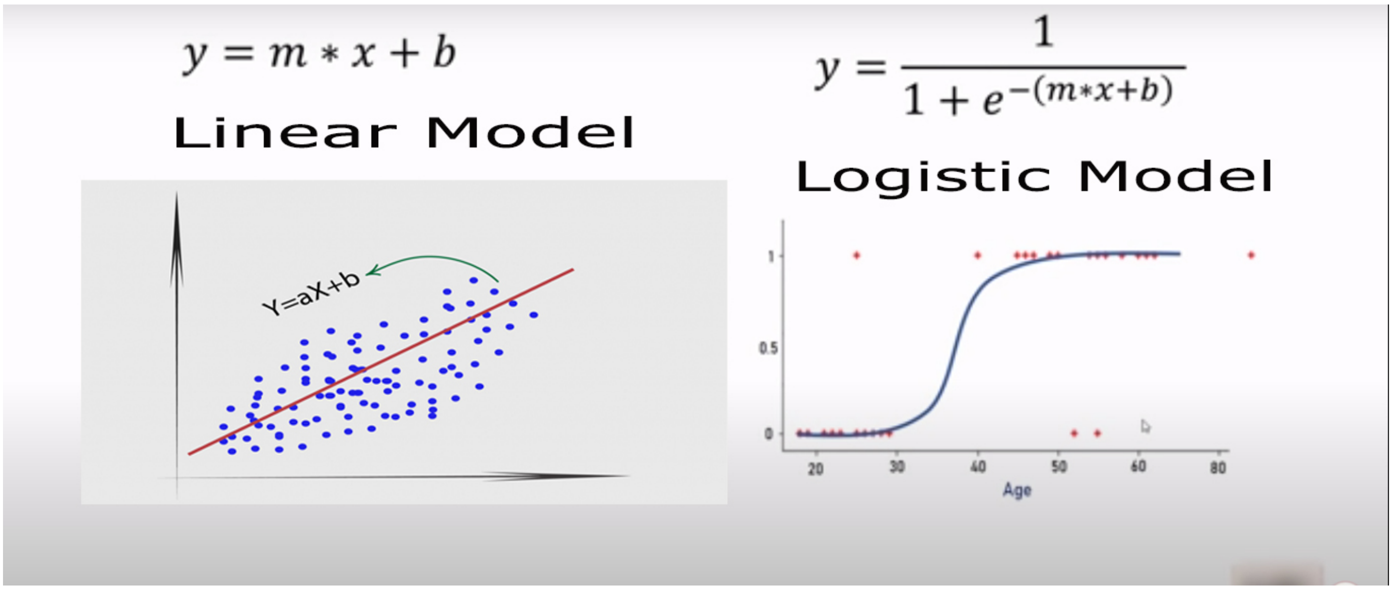
- Logistic regression can be expressed as:

$$\log\left(\frac{p(X)}{1-p(X)}\right) = \beta_0 + \beta_1 X.$$

- where, the left hand side is called the logit or log-odds function, and p(x)/(1-p(x)) is called odds.
- The odds signifies the ratio of probability of success to probability of failure. Therefore, in Logistic Regression, linear combination of inputs are mapped to the log(odds) - the output being equal to 1.
- The cost function for the whole training set is given as :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

Logistic And Linear Model



Pratical Demonstrate Of Logistic Regression

Importing the libraries

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

Importing the dataset

In [2]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
```

In [3]:

```
dataset.head()
```

Out[3]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [4]:

```
X = dataset.drop(['Purchased','User ID','Gender'],axis=1)
y = dataset['Purchased']
```

In [5]:

```
X.shape,y.shape
```

Out[5]:

```
((400, 2), (400,))
```

Splitting the dataset into the Training set and Test set

In [6]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state =
```

Feature Scaling

In [7]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training the Logistic Regression model on the Training set

In [8]:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(C=1.0)
classifier.fit(X_train, y_train)
```

Out[8]:

```
LogisticRegression()
```

Predicting the Test set results

In [9]:

```
y_pred = classifier.predict(X_test)
```

In [10]:

```
calculation = pd.DataFrame(np.c_[y_test,y_pred], columns = ["Original Purchased","Predict P
calculation
```

Out[10]:

	Original Purchased	Predict Purchased
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
95	1	0
96	0	0
97	1	0
98	1	1
99	1	1

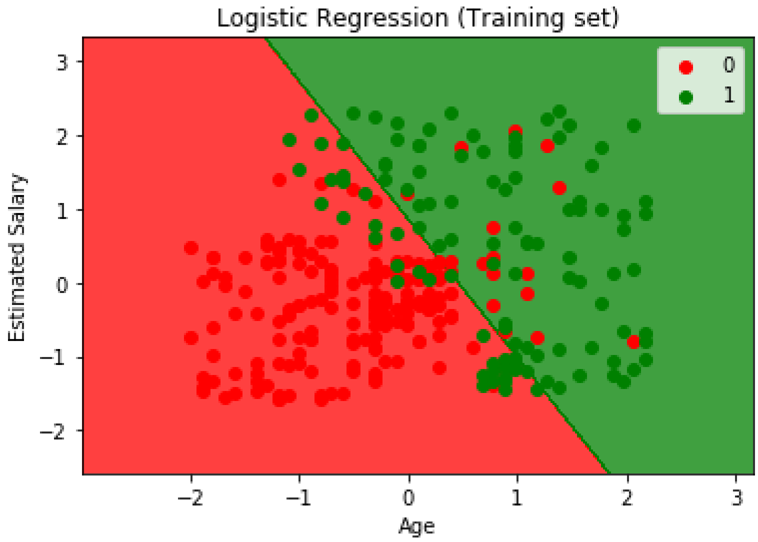
100 rows × 2 columns

Visualising the Training set results

In [11]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.sh
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with 'x' & 'y'.  Please use a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with 'x' & 'y'.  Please use a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
```



Visualising the Test set results

In [12]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1,
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1,
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.sh
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
'c' argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with 'x' & 'y'.  Please use a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with 'x' & 'y'.  Please use a 2-D array with a single row if you really want
to specify the same RGB or RGBA value for all points.
```