Ramesh Awasthi

Github: github.com/rameshawasthi

Twitter : https://twitter.com/awastii

The Basic of Logistic Regression

Logistic Regression is one of the most popular algoriths used in the field of data Science.It is a supervised machine learning technique that models the relationship between one or more predictors and the probability of a categorical response.

- It is different from linear Regression in term of the cost function used.
- Logistic function used the Sigmoid Function.

Why and When to use Logistic Regression Model:

Easy to implement and use :

logistic regression models are easy to train, They do not require hyper parameter tunning.

Very efficeint to train:

Logistic Regression models are efficient in that are not computationally expensive.

Does not require that predictors be scaled:

Output is easy to understand:

Unlike some other machine learning algorithms, the predictive value and the coefficients of a logistic regression model are easy to understand and interpret.

Able to handle a reasonable number of categorical features.

Makes strong assumptions about the data.

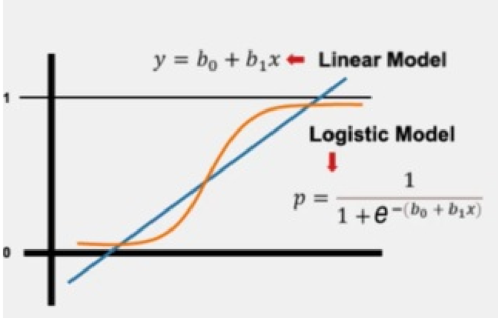
Does not naturally capture complex relationship:

As a result of the assumptions we make in logistic regression, our model may not be able to capture some of the complxe or subtle patterns in the data.

Does not do well with missing or outlier data.

Understanding Regression

Linear Regression outputs continuous numeric values, whereas logistic regression transforms its output to return a probability values which can be used for mapping to two or more classes.



Sigmoid Function

The hypothesis of logistic regression tends it to limit the cost function between 0 and 1. The sigmoidal function is given as :

$$f(x) = \frac{1}{1 + e^{-(x)}}$$

Using linear regression, formula of hypothesis is: $h\theta(x) = \beta_0 + \beta_1 X$

For logistic regression:

$$\sigma(Z) = \sigma(\beta_0 + \beta_1 X)$$
$$Z = \beta_0 + \beta_1 X$$
$$h\theta(x) = sigmoid(Z)$$
$$i.e.,\ h\theta(x) = 1/(1 + e^{-(\beta^0 + \beta^1 X)})$$

Dicision Boundary

The hypothesis of logistic regression can be given as:

$$h\theta(x) = \frac{1}{(1 + e^{-(\beta^0 + \beta^1 X)})}$$

Decision Boundary: A threshold value is decided between 0 and 1, which decides the class a numeric value may correspond to.

Cost Function

The cost function to be minimized in logistic regression can be given as:

$$c(h_\theta(x), y) = \begin{cases} -log(h_\theta) & if\ y{=}1 \\ -log(1{-}h_\theta(x)) & if\ y{=}0 \end{cases}$$

Which can be compressed into a single function:

$$J(\theta) = -\frac{1}{m}\Sigma\left[Y^{(i)}\log\left(h\theta(x(i))\right) + (1 - y^{(i)})\log\left(1 - h\theta(x(i))\right)\right]$$

Types

I) Binary Logistic Regression:

The categorical response has only two possible outcomes.

eg. email Spam or Not

II) Multinomial Logistic Regression:

Three or more categories without ordering.

eg. Predecting which food is preferred more.

III) Ordinal Logistic Regression

Three or more categories with ordering.

eg. Movie or Product rating 1 to 5.

Advantages

1.Makes no assumption about distribution of class in feature space.

2.Quite easier to understand, implement and efficient to train.

3.Can easily formulate multiple regression in consideration.

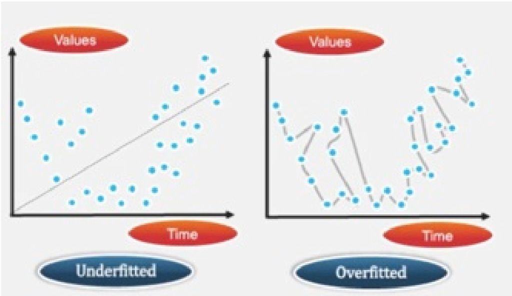
4.It gives direction of association among the dependent and independent variable involved.

5.Gives good accuracy for simple datasets.

Disadvantages

1.Can lead to overfitting if the number of feature is more than observations.

2.It also assumes linearity between independent and dependent variables.

3.It can only be used to predict discrete functions. 

Real Life Application

I) It is often used in classification problem dealing with spam detection in email. A spam may be labelled as '1' and '0' is given to no-spam

II) Credit Card Fraud can also be detected through logistic regression. It uses factors like data of the transaction, amount, place, type of purchase and many more.

II) Tumer Prediction maybe classified into malignant or bening.

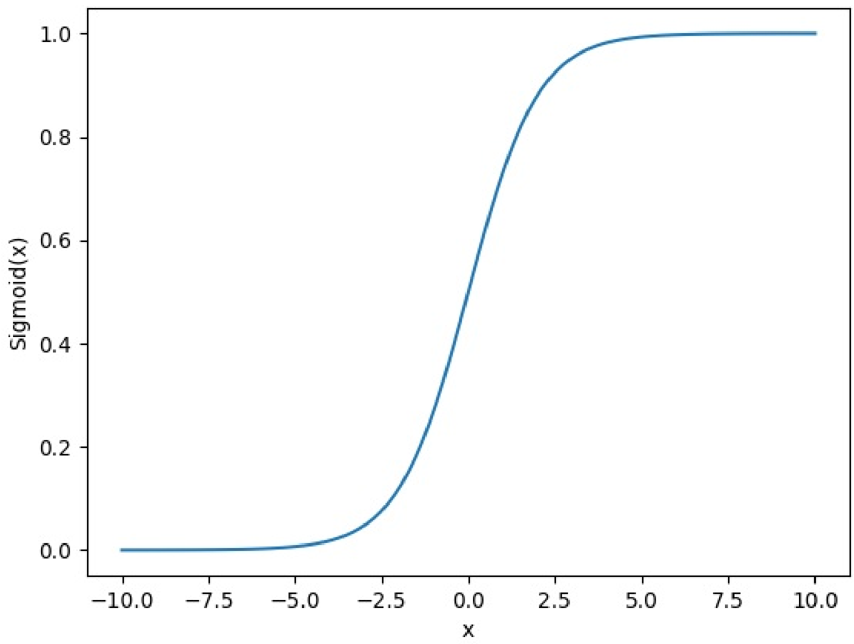
Logistic Regression Model

In [1]:

```
#packages import
import numpy as np
%matplotlib notebook
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import sys
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
```

In [38]:

```
#Display sigmoid Function
x = np.linspace(-10,10,100)
y = 1/(1+np.exp(-x))
plt.plot(x,y)
plt.xlabel("x")
plt.ylabel("Sigmoid(x)")
plt.show()
```



In [26]:

```
#Collection and import dataset, before import data you must be import pandas package.
# https://github.com/rameshawasthi/Data-Science/blob/main/Loan.csv --> dataset or u can use your data set
dataset = pd.read_csv("Loan.csv")
```

In [27]:

```
#check columns of your dataset
dataset.columns
```

Out[27]:

```
Index(['Income', 'Loan Amount', 'Target'], dtype='object')
```

In this dataset we have three columns. The first two (Income and Loan Amount) are the predictor(independent variables), While the last one - Target is the response (or dependent Variable).

we will use dataset to train logistic regression model to predict whether a borrow will Target or not Target on a new loan based on their income and the amount of money they intend to borrow.

In [28]:

```
#check last data of your dataset
dataset.tail()
```

Out[28]:

	Income	Loan Amount	Target
25	15	85	yes
26	18	90	yes
27	16	100	yes
28	22	105	yes
29	14	110	yes

In [29]:

```
#summary statistics of your  dataset
dataset.describe()
```

Out[29]:

	Income	Loan Amount
count	30.000000	30.000000
mean	20.966667	54.233333
std	6.195011	28.231412
min	12.000000	8.000000
25%	16.250000	32.000000
50%	20.500000	54.500000
75%	24.750000	71.750000
max	34.000000	110.000000

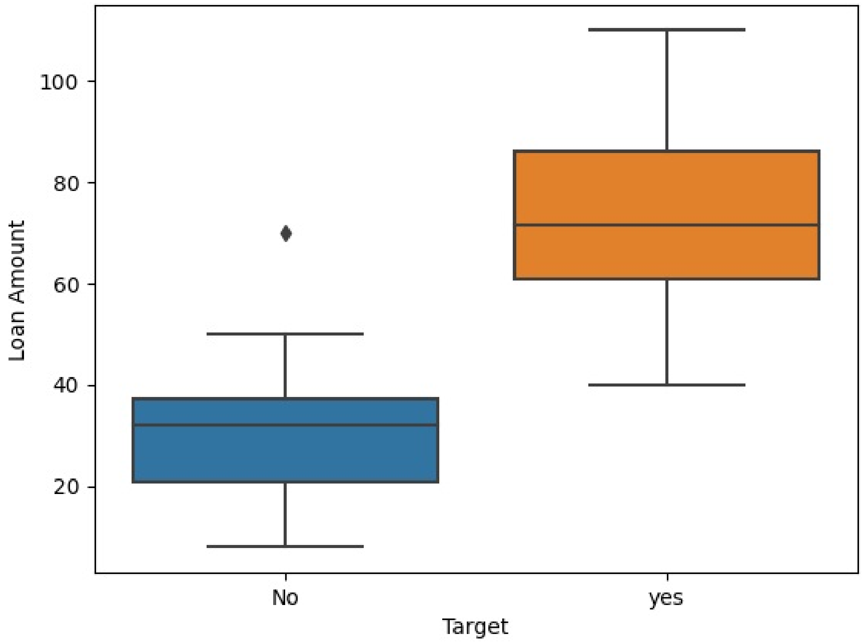
In [30]:

```
#Explore the Data
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 3 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   Income       30 non-null     int64
 1   Loan Amount  30 non-null     int64
 2   Target       30 non-null     object
dtypes: int64(2), object(1)
memory usage: 848.0+ bytes
```

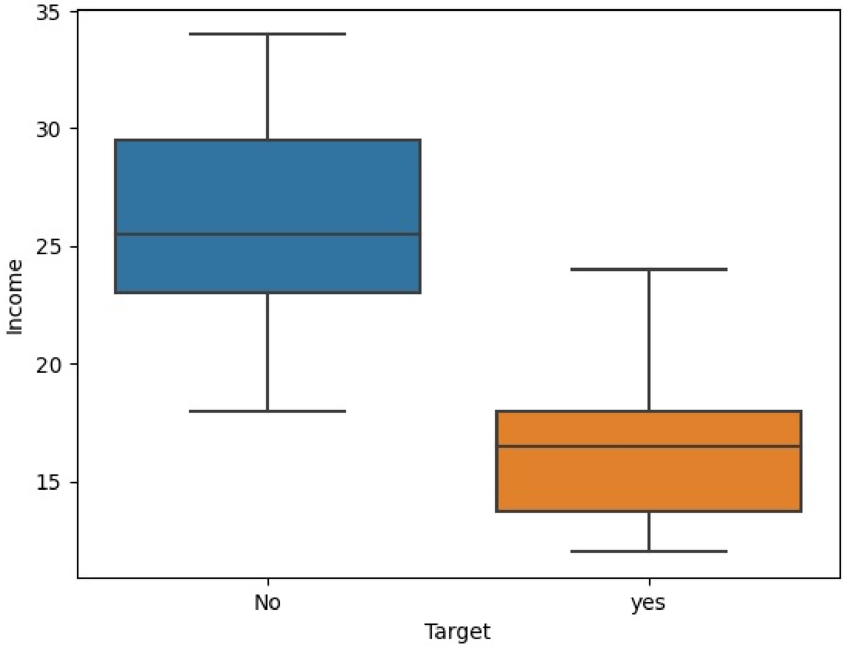
In [34]:

```
#Visualize data, Boxplot
ax = sns.boxplot(data = dataset, x = 'Target', y='Loan Amount')
plt.show()
```



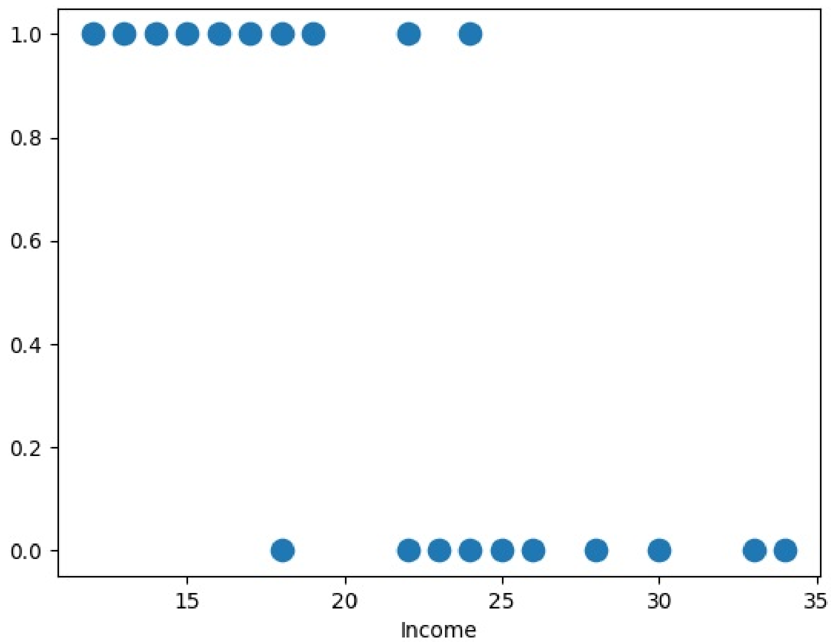
In [35]:

```
ax = sns.boxplot(data = dataset, x = 'Target', y='Income')
```



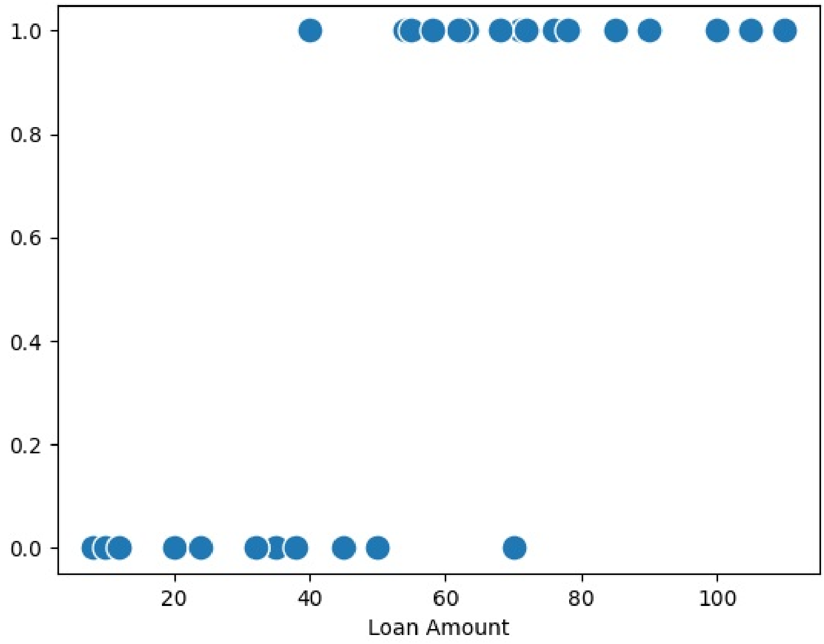
In [36]:

```
ax = sns.scatterplot(x = dataset['Income'], y = np.where(dataset['Target'] == 'No',0 ,1), s=150)
```



In [37]:

```
ax = sns.scatterplot(x = dataset['Loan Amount'], y = np.where(dataset['Target'] == 'No',0 ,1), s=150)
```



Prepare the Data

Primary Objective in this step is to split our data into training and test sets. The training set be used to train the model , while the test will be used to evalutate the model.

In [39]:
```
x = dataset[['Income', 'Loan Amount']]
# X for the independent variables.
```

In [40]:
```
y = dataset['Target']
# y for dependent variable.
```

Using the train_test_split() function, we can split x and y into x_train, x_test, y_train, and y_test.

Within the train_test_split() function, we will set: train_size to .70 to .80 . this mean depend on data size basically we assigned 70% to 80% for training data while rest of 20% to 30% is assigned to the test data.

startify as y which means that we want the data split using a stratified random sampling approach based on the values of y.

random_state to 123 we get the same result every time we do this split.

In [41]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, train_size=0.7, stratify=y, random_state=123)
```

In [42]:
```
#shape of train data
x_train.shape
```

Out[42]:
```
(21, 2)
```

The about result show us that 21 out of the 30 instances in the dataset were assigned to the train set.

In [43]:
```
#shape of test data
x_test.shape
```

Out[43]:
```
(9, 2)
```

The about result show us that 9 out of the 30 instances in the dataset were assigned to the test set.

In [44]:
```
#Train and Evaluate the Model
classifier = LogisticRegression()
#Instantiate a new object called classifier from LogisticRegression class.
```

In [45]:
```
#To train model, we pass the training data(x_train & y_train) to the fit() method of the classifier object.
model = classifier.fit(x_train, y_train)
```

In [46]:
```
#Recall taht there are 9 instances(or  rows) in the test set.
```

```
#to predict labels for the test instances , we pass the independent variable of the test set(x_test) to the pre
model.predict(x_test)
```

Out[46]:
```
array(['yes', 'yes', 'yes', 'yes', 'yes', 'No', 'No', 'No', 'yes'],
      dtype=object)
```

In [47]:
```
#To evaluate how accurate our model is , we pass the test data(x_test and y_test) to score() method of the mode
model.score(x_test, y_test)
```

Out[47]:
```
0.8888888888888888
```

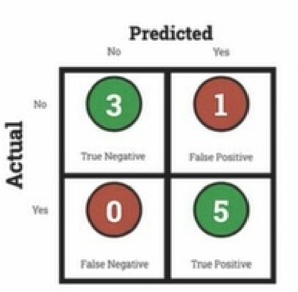
The result tells us, Logistic Regression model is able to correctly predict 8 out of 9(89%) of te labels in the test set.

The accuracy of a model only gives us a one-dimensional perspective of performance.To get a broader perspective, we need to generate a conusion matrix od the model's performance.

In [48]:
```
confusion_matrix(y_test, model.predict(x_test))
# here we pass the dependent variable from the test set(which are actual labels) and the model's predicted labe
```

Out[48]:
```
array([[3, 1],
       [0, 5]])
```

The output is a 2*2 array that shows how many instance the model predicted correctly or incorrectly as either Yes or No. this confusion matric can be define :



The first row of the matrix shows that of the 4 instances that were actually NO, the model predicted 3 of them as NO but 1 of them as Yes. The second row of the matrix shows that of the 5 instances that were actually Yes, the model predicted all 5 correctly as Yes.

Interpret the Model we did built model and evaluated the performance of the model on the test data, we can now interpret the model's output. The model coefficeints.

The relation between the dependent and independent variables in a Logistic Regression model is generally represented as follows:

$$log(\frac{P}{1-P}) = \beta_0 + \beta_1 X_1 + ... + \beta_n X_n$$

In this representation , the left hand side of the equation is know as the logit or log-odds of the probability of an outcome or class P. β0 is the intercept. β1 to βn are coefficients of the independent variables x1 to xn.

In [49]:
```
model.intercept_
#get intercept(β0 ), we use intercept_ attribute for model.
```

Out[49]:
```
array([15.4670632])
```

In [50]:
```
model.coef_
# To get the other model coefficient that is β1, β2 we use coef_ attribute for model
```

Out[50]:
```
array([[-1.0178107 ,  0.14656096]])
```

$$log(\frac{P}{1-P}) = 15.4670632 - 1.0178107 \times \text{Income} + 0.14656096 \times \text{Loan Amount}$$

The model coefficient correspond to the order in which the independent variables are listed in the training data.This means that the above equation is our logistic regression model.

In [56]:
```
coeff_odds = np.round(model.coef_ [0], 2) # round place upto 2 decimal
coeff_odds
```

Out[56]:
```
array([-1.02,  0.15])
```

The above code make coefficeints easier to work with , can convert the coefficients from a two dimensional array to a one-dimensional array and round the values to two decimal places.

round*() is a mathematical function that rounds an array to the given number of decimals. Syntax : numpy.round*(arr, decimals = 0, out = None)

In [57]:
```
pd.DataFrame({'coeff odds': coeff_odds}, index = x.columns)
```

Out[57]:

	coeff odds
Income	-1.02
Loan Amount	0.15

Above we create a Pandas DataFrame using the coefficint values and the columns name from the training data as row indexes.

Above , first coeff tells us that, when all other variables are held constant, a $1 increase in a borrowers income decrease the coeff odds that they will target on their loan by 1.02.

Likewise the second coefficient tells us that a $1 increase in the amount a customer borrows, increase the coeff odds that they will target on their loan by 0.15 when all other variable are held constant.

In [58]:

```
odds = np.round(np.exp(coeff_odds), 2)
pd.DataFrame({'odds': odds}, index = x.columns)
```

Out[58]:

	odds
Income	0.36
Loan Amount	1.16

In above: The coefficients in terms of coeff odds is a bit confusing, A more intuitive approach would be to look at them in terms of odds. above exponentiate the coefficinets so we can interpret them in terms of odds rather than odds.

first coefficent tells us that, for every $1 increase in a borrower's income, the odds taht will target on their loan reduces by 1-.36 =.64 i.e 64% when all other variables are held constant. Earning more money decrease the odds of target.

The second coefficent tells us that, assuming all other variables are held constant, for every $1 increase in the amount borrowed, the odds that a borrower will default on their loan increase by 1.16-1 =.16 i.e 16%. Borrowing more money increase the odds of target.

the second coefficient also saying that for every $1 increase in the amount borrowed , the odds that a borrower will target on their loan increases by a factor of 1.16, assuming all other variables are held constant.