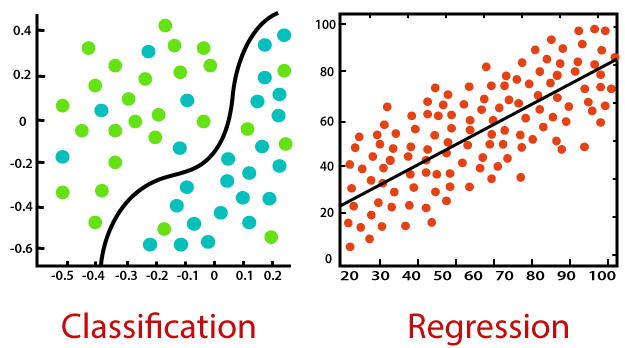
Chapter 3 : part 1

**Logistic Regression**

**3.1.1 REGRESSION vs CLASSIFICATION**

*Regression* and *Classification* algorithms are *Supervised* *Learning* algorithms. Both the algorithms are used for prediction in Machine learning and work with the labeled datasets. But the difference between both is how they are used for different machine learning problems.

* The main difference between them is: *Regression* *algorithms* are used to *predict* the *continuous* *values* such as price, salary, age, etc. and *Classification* *algorithms* are used to *predict/Classify* the *discrete* *values* such as Male or Female, True or False, Spam or Not Spam, etc.



* Classification: Classification is a process of *finding* a *function* which helps in *dividing* *the dataset* into *classes* based on different parameters. In Classification, a computer program is trained on the training dataset and based on that training, it categorizes the data into different classes.
* The task of the classification algorithm is to ***find the mapping function*** to map the ***input(x)*** to the ***discrete output(y)***.
* Example: The best example to understand the *Classification* problem is *Email Spam Detection*. The model is trained on the basis of millions of emails on different parameters, and whenever it receives a new email, it identifies whether the email is spam or not. If the email is spam, then it is moved to the Spam folder.

|  |  |
| --- | --- |
| * Types of ML Classification Algorithms: Classification Algorithms can be further divided into the following types: | 1. Logistic Regression 2. K-Nearest Neighbors 3. Support Vector Machines (SVM) 4. Kernel SVM 5. Naïve Bayes 6. Decision Tree Classification 7. Random Forest Classification |

* Regression: Regression is a process of *finding the* ***correlations*** *between* ***dependent*** *and* ***independent*** *variables*. It helps in predicting the continuous variables such as prediction of Market Trends, prediction of House prices, etc.
* The task of the Regression algorithm is to find the ***mapping function*** to map the ***input variable(x)*** to the continuous output ***variable(y)***.
* Example: Suppose we want to do weather forecasting, so for this, we will use the *Regression* *algorithm*. In weather prediction, the model is trained on the *past* *data*, and once the training is completed, it can easily predict the weather for *future* *days*.

|  |  |
| --- | --- |
| * Types of Regression Algorithm: | 1. Simple Linear Regression 2. Multiple Linear Regression 3. Polynomial Regression 4. Support Vector Regression 5. Decision Tree Regression 6. Random Forest Regression |

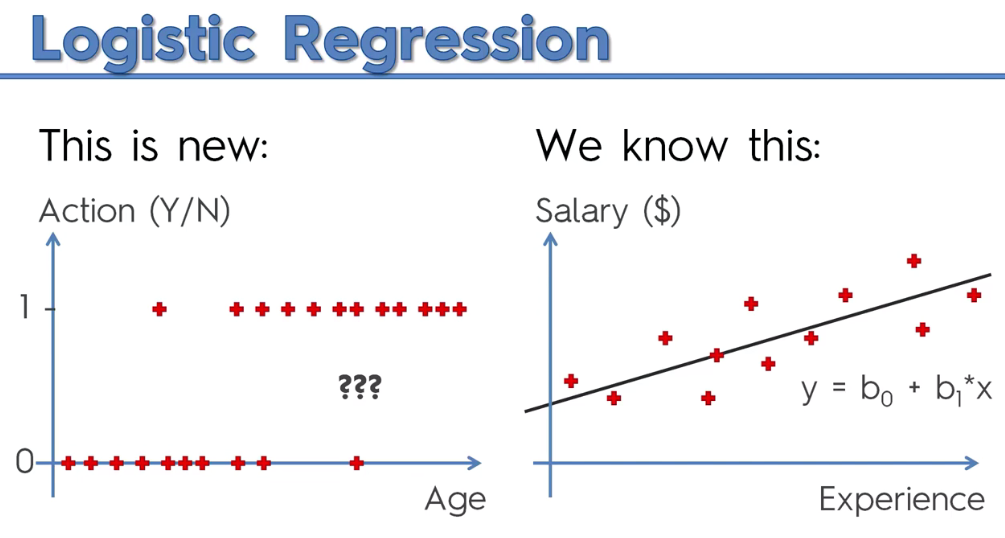
* Difference between Regression and Classification:

|  |  |
| --- | --- |
| Regression Algorithm | Classification Algorithm |
| 1. In Regression, the *output* *variable* must be of *continuous* nature or real value. | 1. In Classification, the *output* *variable* must be a *discrete* *value*. |
| 1. The task of the regression algorithm is to *map* the *input* value (x) with the *continuous* *output* variable(y). | 1. The task of the classification algorithm is to *map* the *input* value(x) with the *discrete* *output* variable(y). |
| 1. Regression Algorithms are used with continuous data. | 1. Classification Algorithms are used with discrete data. |
| 1. In Regression, we try to find the best fit line, which can predict the output more accurately. | 1. In Classification, we try to find the decision boundary, which can divide the dataset into different classes. |
| 1. *Regression* *algorithms* can be used to solve the regression problems such as *Weather* *Prediction*, *House* *price* prediction, etc. | 1. *Classification* *Algorithms* can be used to solve classification problems such as Identification of *spam* *emails*, *Speech* *Recognition*, *Identification* of *cancer* *cells*, etc. |
| 1. The regression Algorithm can be further divided into *Linear* and *Non-linear* Regression. | 1. The Classification algorithms can be divided into *Binary* Classifier and *Multi-class* Classifier. |

* Does regression mean prediction?

Using *regression* to make *predictions* doesn't necessarily involve *predicting* the *future*. Instead, you *predict the mean* of the *dependent* *variable* given specific values of the *independent* *variable(s)*.

**3.1.2 Introduction to Classification problem**



* We already know how to deal with data like in the Right side (with regression). But how do we deal with the data in the Left side?
* In the Left side data, lets assume the problem as: Action of buying a product, from the left side data we can see there is actually a correlation, which is "older people usually takes action to – Yes and younger people actually doesn’t take actions".
* We can see that the observations on the bottom they're a bit more to the left and observations on the top are a bit more to the right implying that, ***"probably older people are more likely to take action based on the shopping offer"***.

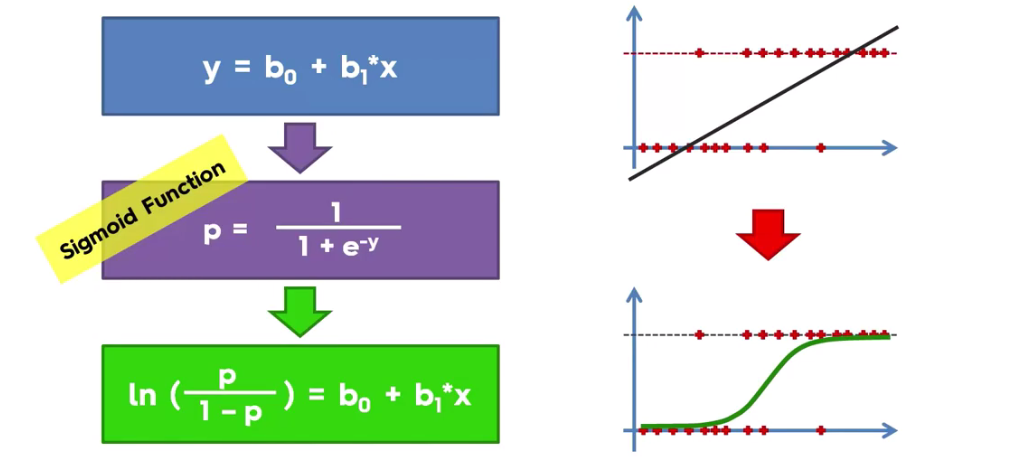
|  |  |  |
| --- | --- | --- |
| * Use Linear Regression first: We try our existing method in our toolkit which is the *linear* *regression*, it doesn't look like the best approach and not the best method to solve this problem. * Let's draw a horizontal line . Now, we want to predict for that person of certain age we want to predict whether they will take up the offer or not. * To do that, we will predict the *probability* *rules* that will state a probability or a likelihood of that person taking up that offer. | |  |
| * Now, in our dataset ***taking action*** is indicated by ***0***, ***1*** (the red dots/observations are either 0 or 1). We also know that probabilities are from ***0*** to ***1***. So basically we could fit in probabilities between 0 and 1. * Lets consider, the line is crossing at age 55. And at age 35. * Here, the linear regression line, at least the part that's in the middle between 0 and 1. So those people between 35 and 55 there is a probability of them taking up this offer. Their probability is increasing as we move to the right. (more older people the probability is increasing). | |  |
| * So the part of the linear regression in the middle kind of predicting the fact but lines at the top at the bottom makes no sense. A probably can never be less than zero. It can never be above 1. (following figure on Left side). * We could interpret as is that people above 55 age they are very likely take the offer (high probability, say 100%). Anybody below 35 on the other side on the left they're definitely not taking it (probability 0%). Then the Graph becomes as in the right side as below: | | |
|  | In this model, we would still be able to make some sort of predictions and assumptions. | |

**3.1.3 Logistic Regression & Sigmoid Function**

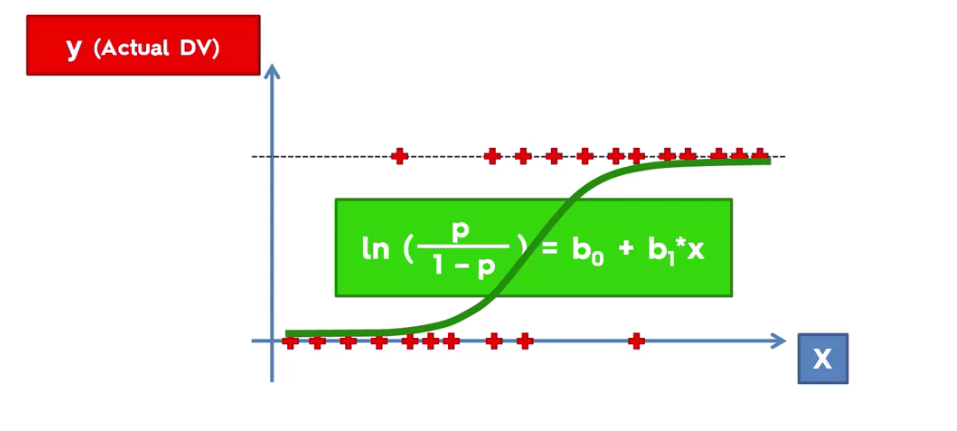
In our first attempt, where we used the Simple Linear Regression, we used the equation.

But in our last attempt we ended up with S-shaped model. For this reason we use the Sigmoid –Function

Combining Simple Linear equation and Sigmoid –Function, replacing y, we get:

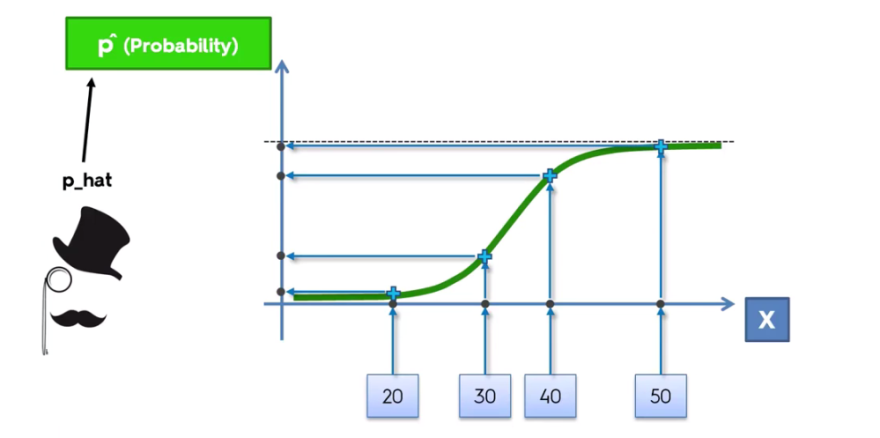


* So basically your linear regression will start to look like following. And is the formula for logistic regression

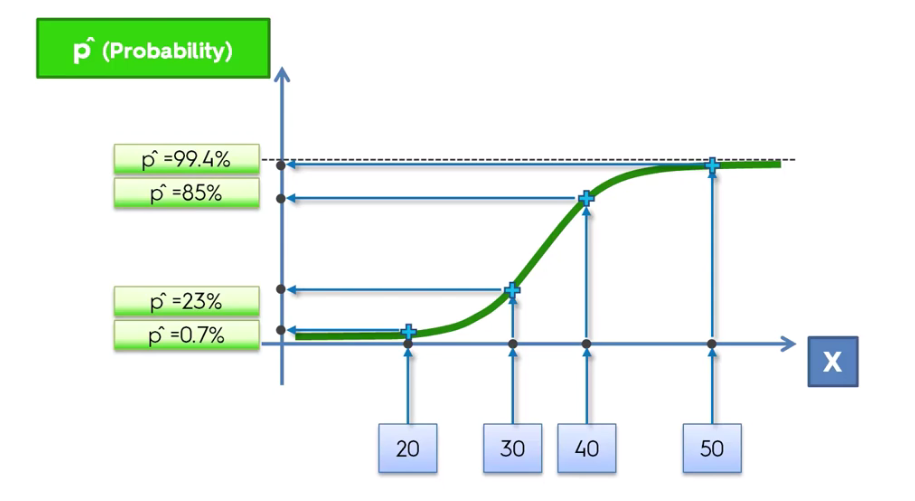


* This line for the logistic regression is the same as a slope for a linear regression. So basically this line is the best fitting line that can fit these data which is obtained from that formula. Basically we doing exactly the same thing as a for linear regression but it just looks different.
* What can we do if this logistic regression?

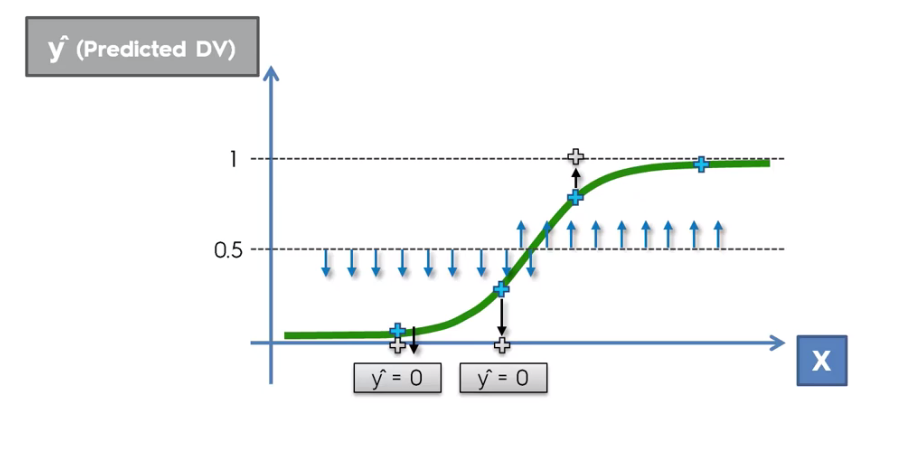
We can use it to predict probabilities instead of predicting for sure that something will or will not happen. Actually we predict probability. The probability here is called (p hat) (anything you see in the ^ in the section means that it's something we're predicting.



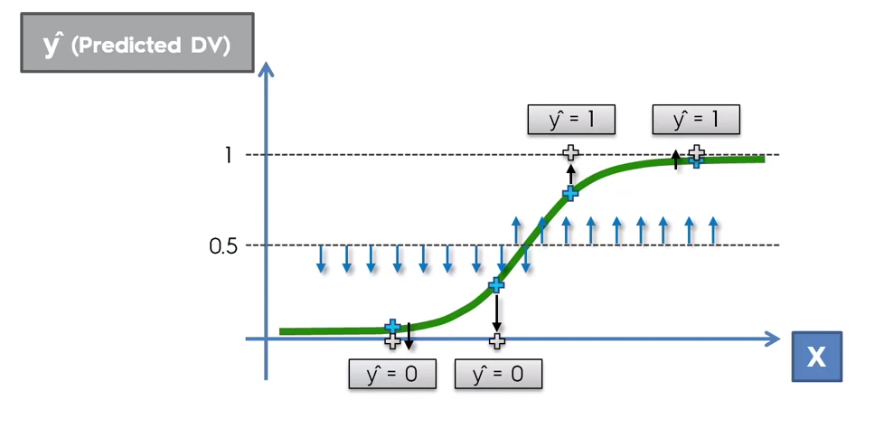
* How to predict: Let's take some random values 20, 30, 40, 50 as ages for the independent variable X. We first project those values to the s-shaped curve and then from the curve we again project to the y-axis (or axis ). The projected points on the axis are the corresponding probabilities.



* The person who's 20 years old the probability of taking up this offer is very low say 0.7%, 30 years has 23%, 40 years has 85%, and 50 years has 99.4%.
* So that's the first thing that you can get out of a logistic regression. We're going to be using it very actively when we're talking about building some Demographic Segmentation because you use this probability as a score. So it's actually even better than just having a one or a zero.
* How to Classify: Anyway you might want to say: ***Well I don't want probability. I want a prediction as well. I want a prediction for the y value***.



* We can only get a prediction . So as it has suggests is a predictive value for the dependent variable. How do you get ?
* Well the approach is very arbitrary. You have to select a line. In this case we're going to check 50%. You can select it anywhere but ***50%*** is just because it's in the middle and you have symmetry.
* Anything below this line (and down part of the s-curve) will be projected downwards onto the zero line. Similarly for the upper portion, which is projected on the .
* i.e. if your predicted probability of ***taking up this offer*** is less than ***50%***, then you just fall to , i.e. you are not gong to take the offer. So we are deviding up to "yes" or "no" (0 or 1), (instead of continuous [0, 1] interval)



**3.1.4 Logistic Regression: Implementation in Python**

|  |  |
| --- | --- |
| We will see, how *logistic* *regression* managed just to *separate* some *categories* and predict a *binary* *outcome*. To understand the implementation of *Logistic Regression* in *Python*, we will use the below example:   * Example: There is a dataset given which contains the information of various users obtained from the *social* *networking* *sites*. There is a car making company that has recently launched a new SUV car. So the company ***wanted to check how many users from the dataset, wants to purchase the car***. * For this problem, we will build a Machine Learning model using the *Logistic regression algorithm*. The dataset is shown in the beside image. In this problem, we will predict the ***purchased*** variable (*Dependent* *Variable*) by using ***age*** and ***salary*** (*Independent* *variables*). | Logistic Regression in Machine Learning |

* Steps in Logistic Regression: To implement the Logistic Regression using Python, we will use the same steps as we have done in previous topics of Regression. Below are the steps:

1. *Data* *Pre-processing* step
2. *Fitting* Logistic Regression to the Training set
3. *Predicting* the test result
4. Test *accuracy* of the *result*(Creation of *Confusion* *matrix*)
5. *Visualizing* the test set *result*.

* Scaling & Splitting: Scaling Before Splitting and Scaling after Splitting (Splitted matrices will be different because Standard-Deviation will be different)

|  |  |
| --- | --- |
| Scaling after Splitting | Scaling Before Splitting |
| #*# Data Split : No need for this example*  **from** sklearn**.**model\_selection **import** train\_test\_split  #*0.25 test\_size means "1/4"th of the total observation*  X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size= 0.25, random\_state = 0)  #*Feature-Scaling*  **from** sklearn**.**preprocessing **import** StandardScaler  #*sc\_x = StandardScaler()*  #*X\_scaled = sc\_x.fit\_transform(X)*  st\_x= **StandardScaler**()  X\_train= st\_x**.fit\_transform**(X\_train)  X\_test= st\_x**.transform**(X\_test) | #*Feature-Scaling*  **from** sklearn**.**preprocessing **import** StandardScaler  #*y need not to be scaled: categorical variable*  sc\_x = **StandardScaler**()  X\_scaled = sc\_x**.fit\_transform**(X)    #*Data Split*  **from** sklearn**.**model\_selection **import** train\_test\_split  #*0.25 test\_size means "1/4"th of the total observation*  X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X\_scaled, y, test\_size= 0.25, random\_state = 0) |

* Fit , Predict: & Confusion matrix

#*Fit dataset to Logistic regression*

**from** sklearn**.**linear\_model **import** LogisticRegression #*import class*

#*instead of "regressor" we now use "classifier"*

classifer = **LogisticRegression**(random\_state= 0) #*create object*

classifer**.fit**(X\_train, y\_train) #*fit the dataset*

#*Predict*

y\_pred = classifer**.predict**(X\_test)

|  |  |
| --- | --- |
| * Why Linear? It's because the logistic regression is a linear classifier, means that we're in two dimensions and two categories of users are going to be separated by a straight line. * Confusion Matrix: Evaluating the Model-Performance using Confusion Matrix This confusion matrix is going to contain the correct and incorrect predictions that our model made on the set. |  |

* Test Accuracy of the result: We can find the accuracy of the predicted result by interpreting the confusion matrix. By above output, we can interpret that ***65+24= 89 (Correct Output)*** and ***8+3= 11(Incorrect Output)***.
* Visualizing the training set result: Finally, we will visualize the training set result. To visualize the result, we will use ***ListedColormap*** class of ***matplotlib*** library. Below is the code for it:

#*Visualising the Training set results*

**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, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, classifer**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             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**(('yellow', 'blue'))(i), label = j)

pLt**.title**('Logistic Regression (Training set)')

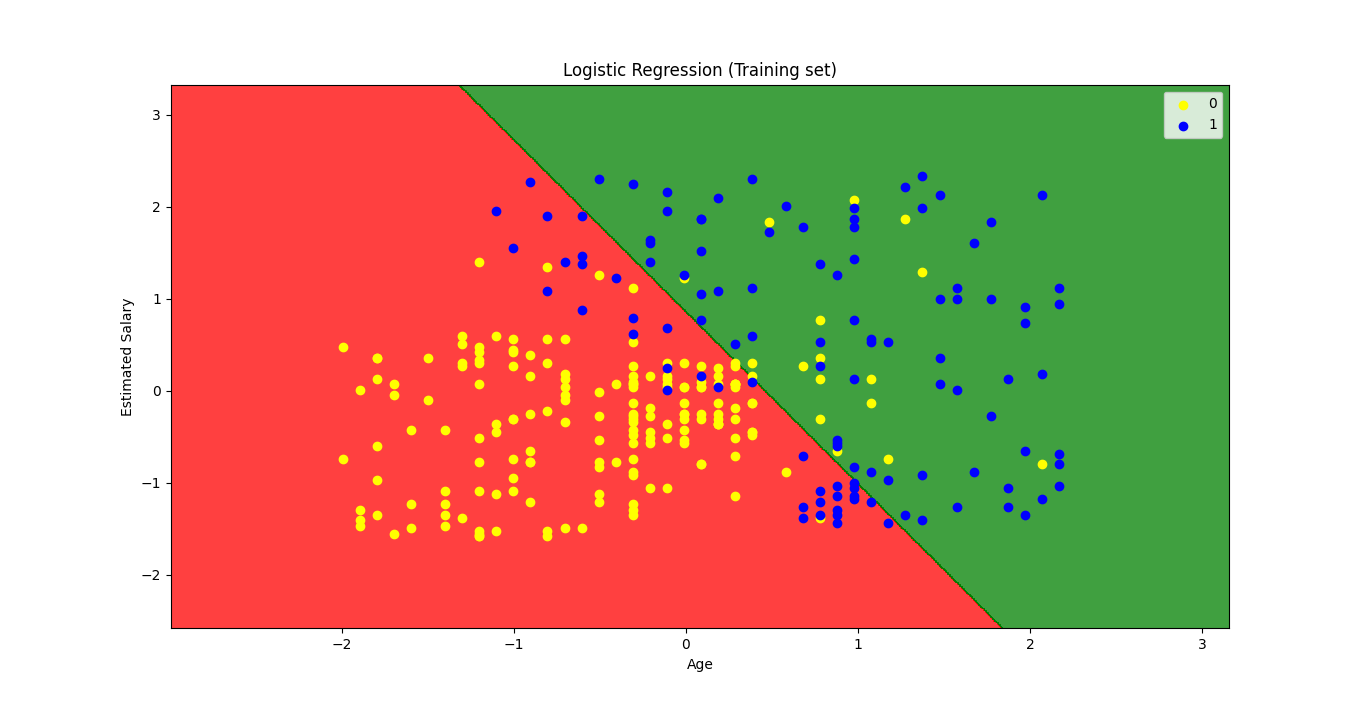
pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()

* In the above code, we have imported the ***ListedColormap*** class of ***Matplotlib*** library to create the colormap for visualizing the result.
* We have created two new variables ***x\_set*** and ***y\_set*** to replace ***x\_train*** and ***y\_train***.
* After that, we have used the ***np.meshgrid*** command to create a Rectangular Grid, which has a range of ***-1(minimum)*** to ***1 (maximum)***. The pixel points we have taken are of ***0.01*** resolution.
* To create a Filled Contour, we have used ***pLT.contourf*** command, it will create regions of provided colors (yellow and blue). In this function, we have passed the ***classifer.predict*** to show the predicted data points predicted by the classifier.
* Output: By executing the above code, we will get the below output:



* The graph can be explained in the below points:
* In the above graph, we can see that there are some **Blue points** within the *green* *region* and **Yellow points** within the *red* *region*.
* All these data points are the observation points from the training set, which shows the result for purchased variables.
* This graph is made by using two independent variables i.e., ***Age on the x-axis***and ***Estimated salary on the y-axis***.
* The ***Yellow point observations*** are for which purchased (dependent variable) is probably ***0***, i.e., users who did not purchase the SUV car.
* The ***Blue point observations*** are for which ***purchased*** (dependent variable) is ***probably 1*** means user who purchased the SUV car.
* We can also estimate from the graph that the users who are *younger* with *low* *salary*, did not purchase the car, whereas *older* *users* with high estimated *salary* purchased the *car*.
* But there are some *Yellow* *points* in the *green* *region* (Not Buying the car) and some *Blue points* in the *red* *region*(buying the car). So we can say that younger users with a high estimated salary purchased the car, whereas an older user with a low estimated salary did not purchase the car.
* The goal of the classifier: We have successfully visualized the training set result for the logistic regression, and our goal for this classification is to divide the users who *purchased the SUV* car and who *did not purchase the car*. So from the output graph, we can clearly see the two regions (Red and Green) with the observation points. The *Red region is for those users who didn't buy the car*, and *Green Region is for those users who purchased the car*.
* Linear Classifier: As we can see from the graph, the classifier is a *Straight* *line* or *linear* in nature as we have used the *Linear model* for *Logistic Regression*. In further topics, we will learn for *non-linear Classifiers*.
* Visualizing the test set result: Our model is well trained using the training dataset. Now, we will visualize the result for new observations (***Test set***). The code for the test set will remain same as above except that here we will use ***x\_test*** and ***y\_test*** instead of ***x\_train*** and ***y\_train***. Below is the code for it:

#*Visualising the Test set results*

**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, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, classifer**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             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**(('yellow', 'blue'))(i), label = j)

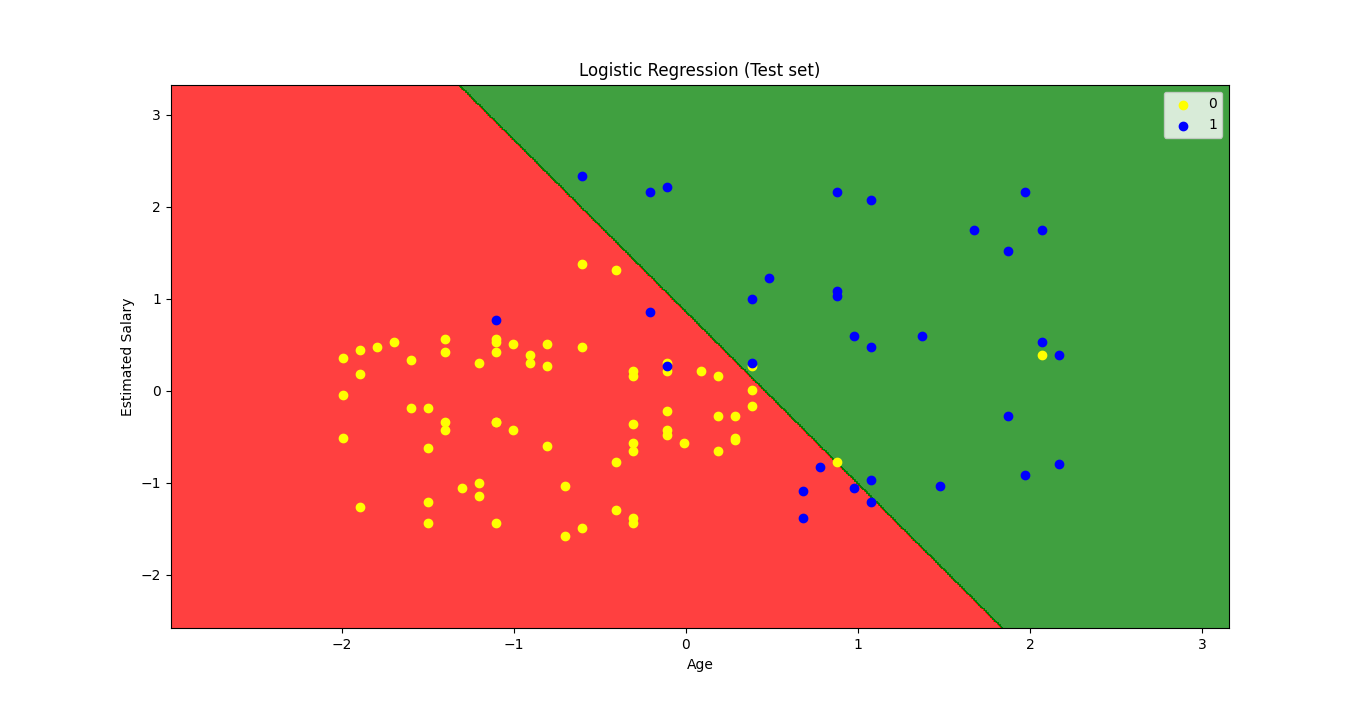
pLt**.title**('Logistic Regression (Test set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()



Full Code (Practiced)

#*Library*

**import** pandas **as** pd

**import** matplotlib**.**pyplot **as** pLt

**import** numpy **as** np

#*Data Extract*

dataSet = pd**.read\_csv**("Social\_Network\_Ads.csv")

X = dataSet**.**iloc[:, [2,3]]**.**values

y = dataSet**.**iloc[:, 4]**.**values

#*Data Split*

**from** sklearn**.**model\_selection **import** train\_test\_split

#*0.25 test\_size means "1/4"th of the total observation*

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size= 0.25, random\_state = 0)

#*Feature-Scaling*

**from** sklearn**.**preprocessing **import** StandardScaler

#*y need not to be scaled: categorical variable*

#*sc\_x = StandardScaler()*

#*X\_scaled = sc\_x.fit\_transform(X)*

st\_x= **StandardScaler**()

X\_train= st\_x**.fit\_transform**(X\_train)

X\_test= st\_x**.transform**(X\_test)

"""

# Feature-Scaling

from sklearn.preprocessing import StandardScaler

#  y need not to be scaled: categorical variable

sc\_x = StandardScaler()

X\_scaled = sc\_x.fit\_transform(X)

# Data Split

from sklearn.model\_selection import train\_test\_split

# 0.25 test\_size means "1/4"th of the total observation

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size= 0.25, random\_state = 0)

"""

#*Fit dataset to Logistic regression*

**from** sklearn**.**linear\_model **import** LogisticRegression #*import class*

#*instead of "regressor" we now use "classifier"*

classifer = **LogisticRegression**(random\_state= 0) #*create object*

classifer**.fit**(X\_train, y\_train) #*fit the dataset*

#*Predict*

y\_prd = classifer**.predict**(X\_test)

#*Making the confusion matrix use the function "confusion\_matrix"*

#*Class in capital letters, functions are small letters*

**from** sklearn**.**metrics **import** confusion\_matrix

cm = **confusion\_matrix**(y\_true= y\_test, y\_pred= y\_prd)

#*parameters of cm: y\_true: Real values, y\_pred: Predicted value*

#*Visualising the Training set results*

**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, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, classifer**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             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**(('yellow', 'blue'))(i), label = j)

pLt**.title**('Logistic Regression (Training set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()

#*Visualising the Test set results*

**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, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, classifer**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             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**(('yellow', 'blue'))(i), label = j)

pLt**.title**('Logistic Regression (Test set)')

pLt**.xlabel**('Age')

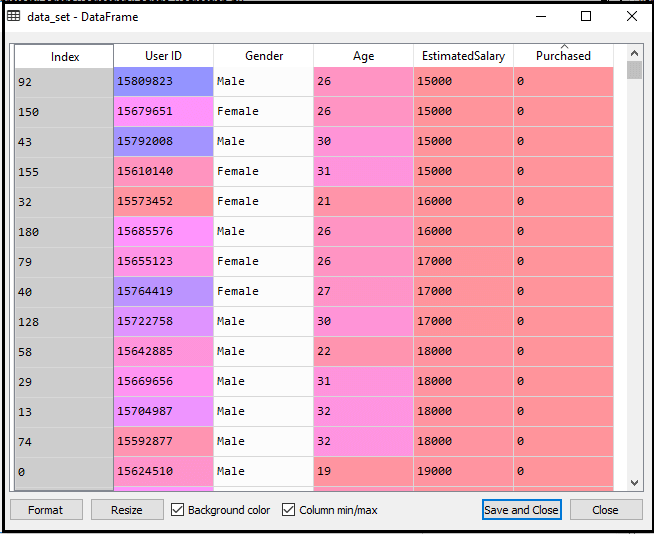
pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()

#*python prctc\_lgstc.py*

* We will get the dataset as the output. Consider the given image:



New TEMPLATE for Classification Problem

#*Library*

**import** pandas **as** pd

**import** matplotlib**.**pyplot **as** pLt

**import** numpy **as** np

#*Data Extract*

dataSet = pd**.read\_csv**("Social\_Network\_Ads.csv")

X = dataSet**.**iloc[:, [2,3]]**.**values

y = dataSet**.**iloc[:, 4]**.**values

        #*Feature-Scaling after Data Split*

#*Data Split*

**from** sklearn**.**model\_selection **import** train\_test\_split

#*0.25 test\_size means "1/4"th of the total observation*

X\_train, X\_test, y\_train, y\_test = **train\_test\_split**(X, y, test\_size= 0.25, random\_state = 0)

#*Feature-Scaling*

**from** sklearn**.**preprocessing **import** StandardScaler

#*y need not to be scaled: categorical variable*

#*sc\_x = StandardScaler()*

#*X\_scaled = sc\_x.fit\_transform(X)*

st\_x= **StandardScaler**()

X\_train= st\_x**.fit\_transform**(X\_train)

X\_test= st\_x**.transform**(X\_test)

        #*Feature-Scaling before Data Split*

"""

# Feature-Scaling

from sklearn.preprocessing import StandardScaler

#  y need not to be scaled: categorical variable

sc\_x = StandardScaler()

X\_scaled = sc\_x.fit\_transform(X)

# Data Split

from sklearn.model\_selection import train\_test\_split

# 0.25 test\_size means "1/4"th of the total observation

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size= 0.25, random\_state = 0)

"""

#*Fit train set to Model classifier*

**from** sklearn**.**\_ **import**

clsFier =

clsFier**.fit**(X\_train, y\_train) #*fit the dataset*

#*Predict*

y\_prd = clsFier**.predict**(X\_test)

#*Making the confusion matrix use the function "confusion\_matrix"*

#*Class in capital letters, functions are small letters*

**from** sklearn**.**metrics **import** confusion\_matrix

cm = **confusion\_matrix**(y\_true= y\_test, y\_pred= y\_prd)

#*parameters of cm: y\_true: Real values, y\_pred: Predicted value*

#*Visualizing the trainig set resultl*

#*Visualising the Training set results*

**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, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, clsFier**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             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**(('yellow', 'blue'))(i), label = j)

pLt**.title**('Logistic Regression (Training set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()

#*Visualising the Test set results*

**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, step = 0.01),

                     np**.arange**(start = X\_set[:, 1]**.min**() - 1, stop = X\_set[:, 1]**.max**() + 1, step = 0.01))

pLt**.contourf**(X1, X2, clsFier**.predict**(np**.array**([X1**.ravel**(), X2**.ravel**()])**.**T)**.reshape**(X1**.**shape),

             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**(('yellow', 'blue'))(i), label = j)

pLt**.title**('Logistic Regression (Test set)')

pLt**.xlabel**('Age')

pLt**.ylabel**('Estimated Salary')

pLt**.legend**()

pLt**.show**()