Classification Algorithms - Part 1 | Conceptual Understanding and Python Implementation

Relevel
by Unacademy

Assignment Discussion- Previous Class

Assignment Question

Predict Car Selling price by trying to achieve maximum R2 Adjusted score.

Instructions –

- Do a 75/25 split
- Random seed = 100

Important links for this class

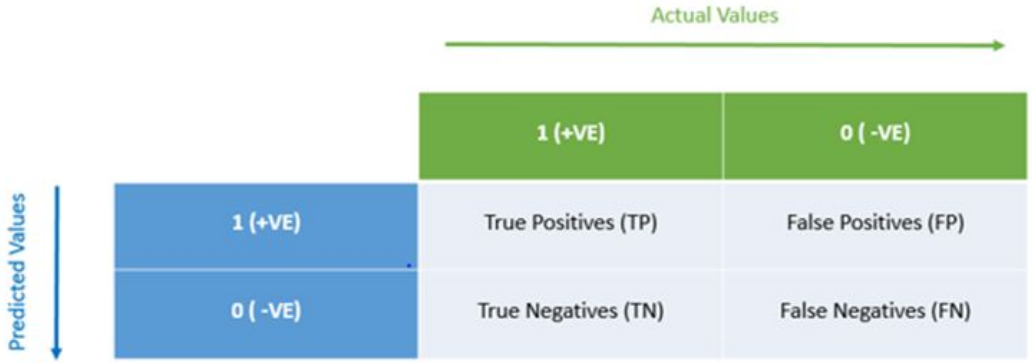
Dataset - https://drive.google.com/file/d/1lWF2v9erkOxpGp3Z0-iJmXyeocjK-C_C/view?usp=share_link
Jupyter Notebook - -
https://drive.google.com/file/d/1wx6cPq0X2XYWn8dUlLC2dHqq1QjYndBt/view?usp=share_link

Precision , Recall & F-Score

Before we dive into Classification algorithms, it is important that we discuss some more performance metrics for classification.

1. Recall
2. Precision
3. F-Beta Score

It is also important to know about these because most of the Python “scoring” functions would return these as output, and it is always beneficial to understand thoroughly what we see as output.
We have already discussed the concept of Confusion Matrix earlier, but for sake of continuity, let’s look at the Confusion matrix for a binary classification below –



Precision , Recall & F-Score

1. Recall / True Positive Rate :

For a binary classification, it is a score which tells us the ratio of correct +ve class predictions with respect to all +ve class predictions. Recall is also known as True Positive Rate. We'll look at the mathematical formula for better understanding -

$$Recall\ (TPR) = \frac{TP}{(TP + TN)}$$

For example , for a confusion matrix as below –

5	4
3	1

TPR = 5/8
=> 0.625
As a thumb rule, recall can be calculated by dividing the total correct +ve class predictions by the sum of entire column.

Precision , Recall & F-Score

Why Recall is used?

Recall is used to control Type-II Error. Thus, a model with higher recall is expected to have lower Type-II error because the True Positives have been maximized while we also have False negatives in denominator. Hence a better Recall score would definitely reduce Type-II error.

For example –
In medical situations, controlling Type-2 errors is of more importance than Type-1 errors. Such as in a Cancer prediction model, it is okay to predict a healthy patient as a cancer patient (Type – 1 error), but it would be too dangerous to treat a Cancer patient as a healthy patient (Type- 2 error). In latter case, the patient might lose his life due to this error, but in former case there is no such devastating danger of loss.

Improving Recall for such model would make sure that no cancer patient is predicted incorrectly to be a healthy patient, which is what is needed in such scenario.

Precision , Recall & F-Score

Recall for multi-class prediction models –

In multiclass models, there is no such concept as +ve class or –ve class.
For example, let's assume a model which predicts a fruit name out of Orange, Apple and Banana. In such case, Recall will be calculated separately for each class.

An example of same has been discussed below.

Let's assume the confusion matrix of above model as :

		Actual		
		Apple	**Orange**	**Banana**
Predicted	**Apple**	3	4	2
	Orange	1	8	6
	Banana	4	5	2

Precision , Recall & F-Score

We need to go by definition of Recall which is -
*“Ratio of **correct +ve class predictions** with respect to **all +ve class predictions**.”*

The only difference is that here +ve class would be replaced by each class label iteratively.
Thus,
Recall for Apple = 3/(3+1+4) => 0.375

As a thumb rule, it is the correct predicted value divided by the sum of entire column for that class.
Recall for Orange = 4/(8+4+5) => 0.235
Recall for Banana = 2/(2+6+2) => 0.2

Recall is maximum for Apple, which means model is best for Apple class in terms of Type 2 error control.

Precision , Recall & F-Score

2. Precision :

For a binary classification, It is the ratio of correct +ve class predictions with respect to Total +ve class predictions.
Mathematically,

$$Precision = \frac{TP}{(TP + FP)}$$

For a confusion matrix as below –

5	4
3	1

Precision = 5/(5+4) => 5/9 => 0.56
As a thumb rule, precision is Correct Positive class predictions divided by the sum of entire row.

Precision , Recall & F-Score

Why precision is used -

In contrast to TPR, precision is used to control Type 1 errors. Better the precision score, lower the Type 1 error in model.

A practical example –

Spam detection model.

In such models, it is okay to miss marking some of the actual Spams as spam. The worst that will happen is that the user will read a spam mail.
But one can not afford missing an important mail because it was incorrectly tagged as spam. This can cause severe impacts. For example a person can miss the meeting invite to a very important meeting due to such error.

Precision , Recall & F-Score

Precision for multiclass models –

Let's take the same confusion matrix from Recall example –

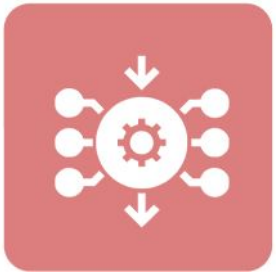
		Actual: Apple	Actual: Orange	Actual: Banana
Predicted	Apple	3	4	2
	Orange	1	8	6
	Banana	4	5	2

Going by the Precision definition in this case we can calculate the same way as we did in case of Recall.
The definition says – " *ratio of* ***correct +ve class predictions*** *with respect to* ***Total +ve class predictions***. "

The only difference is that here +ve class will be replaced by Class labels as there is no concept of +ve /-ve class in multiclass models.
Precision for Apple : 3/(3+4+2) => 3/9 => 0.33
We can see that as a thumb rule, precision is Correct predictions divided by the sum of entire row.
Precision for Orange : 8/(1+8+6) = > 0.53
Precision for Banana : 2/(4+5+2) => 0.18

Precision , Recall & F-Score

3. F-Beta Score :



Now that we have understood Precision and Recall, it is easy to get the idea behind F-Beta.

Generally, there might be cases where both Type-1 and Type-2 errors are of huge impact and it is important to control both. Or there may be cases where although both are important, but comparatively controlling Type-1 is of more importance than Type-2 and vice versa.

Precision and Recall only help us in controlling either Type-1 or Type-2 error, but not both at the same time.

This is achieved by the use of F-Beta Score.

The mathematical formula for F-Beta is given as :

$$F_{Beta} = (1 + \beta^2) \frac{Precision * Recall}{(\beta^2 * Precision) + Recall}$$

Precision , Recall & F-Score

There are 2 important things to note here –

1. F Beta uses both Precision and Recall in its formula, and thus it is able to take into account both
2. There is a variable

β which has following significance :

At
β = 1,

the formula becomes :

$$F_{Beta} = \frac{2 * Precision * Recall}{Precision + Recall}$$

This is nothing but the harmonic mean of 2 metrics. Which basically means that both are given equal significance.
This formula at β = 1 is also known as **F1 Score**.

Precision , Recall & F-Score

At $\beta = 0.5$,

The coefficient with Precision in denominator is squared, and is less than 1, thus Precision gets more significance overall.

Thus, $\beta < 1$, Type-I errors are controlled better than Type-2 errors. Lower the value for β, more prominent the role of
Precision is.

On the other hand, at $\beta > 1$, Type-2 errors are controlled better than Type-1 errors. Higher the value for β, more
prominent the role of Recall becomes.

In Python, all of these performance metrics are available as scoring parameters in GridSearchCV function. We'll be
implementing the same later today.

Logistic Regression

Logistic Regression, contrary to its name, is a Classification algorithm. It is mainly used for Binary Classification, although it is also capable of multi class prediction as well.

It has “Regression” in its name because the backend mathematics is similar to Linear Regression as we are going to see in a few minutes. It also belongs to Regression class algorithms in ML, but the use case is classification prediction due to its properties as we’ll be discussing next.

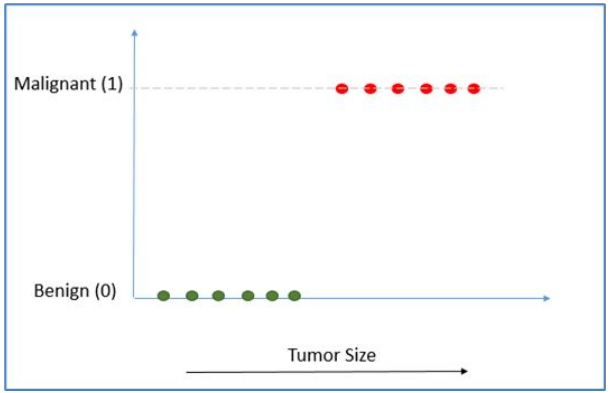
Before we discuss this algorithms, we should address the following question –

Why Linear regression cannot be used for Classification and why Logistic Regression is needed instead?

Let’s assume a Binary Classification problem. Let’s take our Cancer classification example, where we need to Predict whether a Cancer tumor is Malignant or Benign (target) , according to the tumor size (Independent Variable).

Logistic Regression

Let's plot this on a chart as below –



Since there are only 2 Target labels, these are represented by 0s and 1s on Y Axis. Thus, all the independent datapoints will take either of the 2 values 0 or 1 only. This is very different from a regression problem where datapoints are scattered across X and Y axis and a linear regression line is fitted.

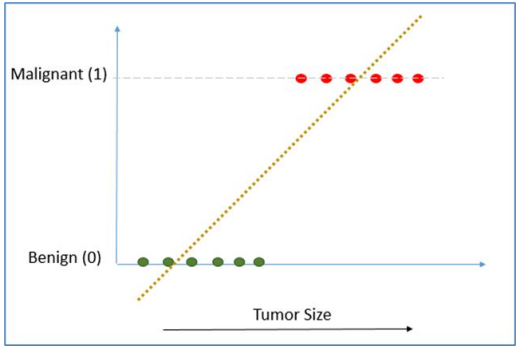
Problem :

A linear regression line can also be fitted here, but it because the data points are organized in a really different manner, such a fitted line would be very unstable and sensitive to new data points and outliers. This is one of the reasons Linear Regression cannot be used for Binary Classification.

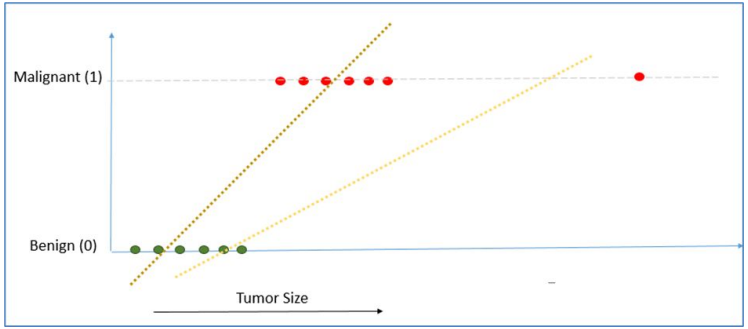
Logistic Regression

Let's try to understand the same through visuals -

If a regression line is fitted to the same data as above, it may look like this –



Now let's say a new data point which is also an outlier, is added, and the linear regression line is again plotted, it will look as below –



Logistic Regression

It can be seen that with just 1 outlier, the line has shifted so much. This new line will be predicting significantly
different predictions than the first line, just because of addition of 1 outlier. Such sensitive and unstable models
cannot be accepted.

Also, it can be observed, that Linear Regression line will have the prediction range from -∞ to +∞, but our
requirement is to predict in the range of 0 to 1 only. This is another reason Linear Regression is not suitable for
such cases.

Just to summarise, Linear regression cannot be used for Classification because –

- It will be extremely sensitive to new data points and outliers if used
- The range is not between desirable bandwidth of 0 to 1.

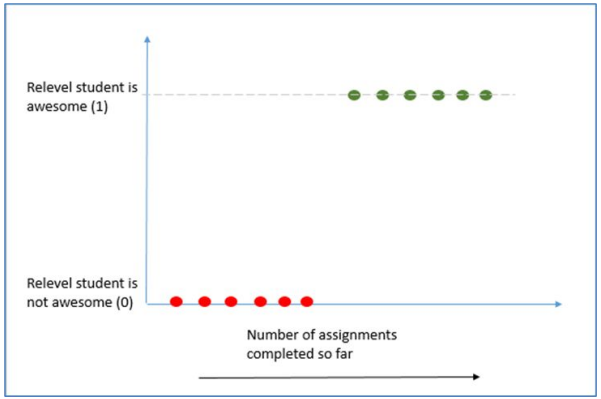
In order to overcome these problems, Logistic Regression is used.

Logistic Regression | Intro

Let's get a high level understanding of Logistic regression with some pointers, and we will dive into technical part later –

- Used for Binary Classification - Logistic Regression is used for Binary Classification (most of the times). For example, its job is usually to predict whether something is True or False, Big or Small, Dead or Alive etc. To make things easier we usually assign 0 and 1 labels to these 2 Target labels. Thus, in more simpler terms, Logistic regression predicts something to belong to either 0 or 1.

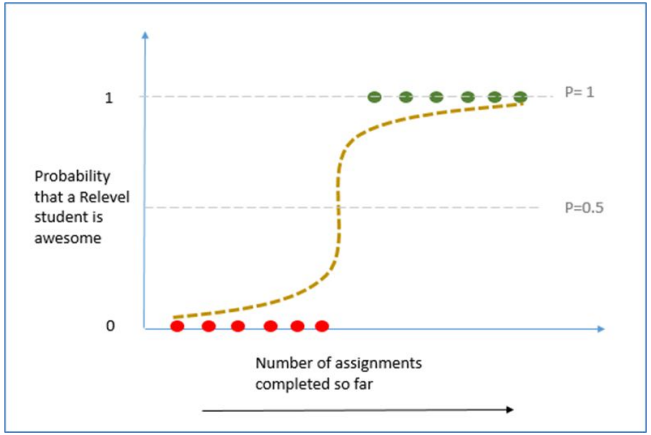
This is contrary to Linear Regression which can predict over a range of values.



Logistic Regression | Intro

- Fits and S Shaped Curve - Instead of fitting a straight line to the data, Logistic Regression fits and S shaped Curve. The range of this Curve is 0 to 1 on y Axis, and Y axis represents Probability. Probability 0 belongs to –ve class and 1 belongs to +ve class.

We'll get into high level details of how Logistic Regression fits this S curve in a few minutes.



- Just like Linear Regression, Logistic Regression can also work with both Continuous and Categorical features.

Logistic Regression

Differences between Linear Regression and Logistic Regression –

- Linear Regression uses Residuals and Least Squared method. It also calculates R2 for comparison.

Logistic regression does not and cannot use Residuals or Least squared method. It uses a concept called *Maximum Likelihood*, which we'll discuss in this session. An R2 value cannot be derived for Logistic regression.
(Although theoretically in advance mathematics , statisticians do calculate R2 for Logistic regression sometimes, but there are multiple methods and there is little consensus among statisticians on which method should be accepted as standard.)

- The range for Logistic Regression predictor is 0 to 1 , while for Linear regression is can be any real number.

- Logistic Regression is used for Classification and Linear regression is used for regression models.

Logistic Regression

Logit Function, Odds , log(odds) & Probability–

Logistics regression uses the concept of Maximum Likelihood to fit an S Curve to the data. The process behind logistic regression also includes a log(odds) transformation using Logit function. In order to understand the what, why and how of that transformation and Maximum likelihood, it is important that we understand about odds and log(odds) and Logit function first.

Let's take an example that India wins 2 out of 5 cricket matches. In this scenario the odds of India winning would be 2:3 or 2/3.

Let's look at the formula for Odds , log(odds) and Probability ; and apply that to Cricket match example-

Odds = Count of Something Happening / Count of something not happening
Won Matches / Lost Matches = > 2/3 => 0.67

Probability = Count of Something happening/ Count of total things happening
Won matches / Total Matches => 2/5 => 0.4

Log(Odds) of winning = log(0.67) => -0.5778

Thus, we can see that Odds and Probability are different. Log(Odds) is literally log of odds and nothing else.

Logistic Regression

Relationship Between Odds and Probability :

Odds can be also be rewritten as :

$$Odds\ (Winning) = \frac{P\ (Winning)}{P\ (Losing)}$$

(For proof, one can substitute the probability formula in above equation, the denominators will cancel out and we'll be left with original Odds formula discussed above.)

This can be rearranged as –

$$Odds\ (Winning) = \frac{p\ (Winning)}{1 - p\ (Winning)}$$

$$\textbf{General Notation => Odds} = \frac{p}{1 - p}$$

Logistic Regression

Taking log both sides ;

⇨ $\text{Log(Odds)} = \text{Log}\left(\frac{p}{1-p}\right)$

This is also known as the Logit function which forms the basis for Logistic Regression.

How, that we'll discuss soon.

Logistic Regression

Why do we need Log(Odds) and what is the practical significance ?

In order to understand the significance of log(odds), we need to look at the same Cricket example with multiple cases of Win-Lose ratio in parallel –

Scenario – Chances of India winning are low (Lost Matches > Won Matches)	
Cases	Odds Winning
Case 1 : 1 Win and 4 Losses	1/4 ⇨ 0.25
Case 2 : 1 Win and 8 Losses	1/4 ⇨ 0.25
Case 3 : 1 Win and 16 Losses	1/16 ⇨ 0.063

Inference -> Worse the Indian team Performs, Odds of winning tend to go to 0. The range of Odds in this scenario where India have more chance of losing lie between **0 to 1.**

Logistic Regression

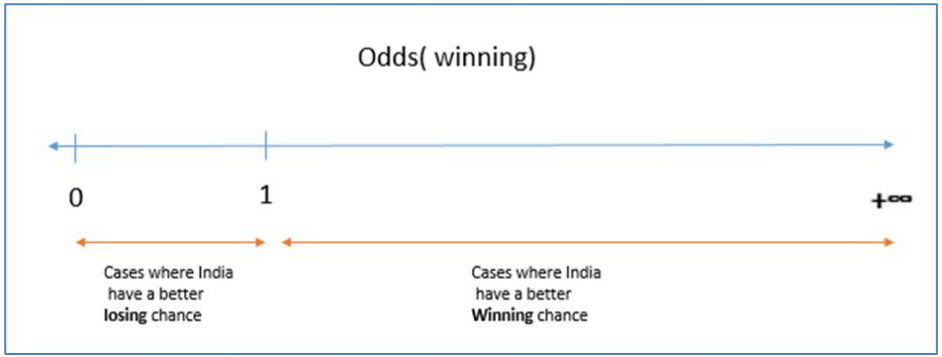
Now let's look at the scenario where India have better chances of winning (India have won more matches than lost matches)

Scenario – Chances of India winning are High (Lost Matches < Won Matches)	
Cases	Odds Winning
Case 1 : 4 Win and 3 Losses	4/3 ⇨ 1.33
Case 2 : 8 Win and 3 Losses	8/3 ⇨ 2.7
Case 3 : 32 Win and 3 Losses	32/3 ⇨ 10.7

Inference -> Inference -> Better the Indian team Performs, Odds of winning tend to go to +∞. The range of Odds in this scenario where India have more chance of winning lie between **1 to +∞**

Logistic Regression

If we plot these inferences on a number line, it will be as below –



Finally, the whole point of this exercise is to show that “Odds” is an asymmetrical metric. For negative cases it has range 0 to 1 while for Positive cases the range is 1 to +Infinity.

Due to this asymmetry, Odds as a metric is rarely used directly in Statistics or ML applications. Rather, Log(Odds) is used in maximum cases which transforms Odds to become Symmetrical in nature.

Logistic Regression

This can be shown with the help of following contrasting cases example –

Case 1: 1 Win and 6 Losses	Case 2: 6 Win and 1 Losses
Odds = 1/6 => 0.167 Log(Odds) = log(0.167) => **-2.582**	Odds = 6/1 => 6 Log(Odds) = log(6) => **+2.582**

Inference -> While the Odds are asymmetrical in nature for above opposite cases, the log(odds) are symmetrical with same magnitude but opposite sign.

Conclusion : Taking log(odds) makes everything symmetrical, and hence used in place of Odds.

Logistic Regression

How does Logistic Regression work

Step 1 :

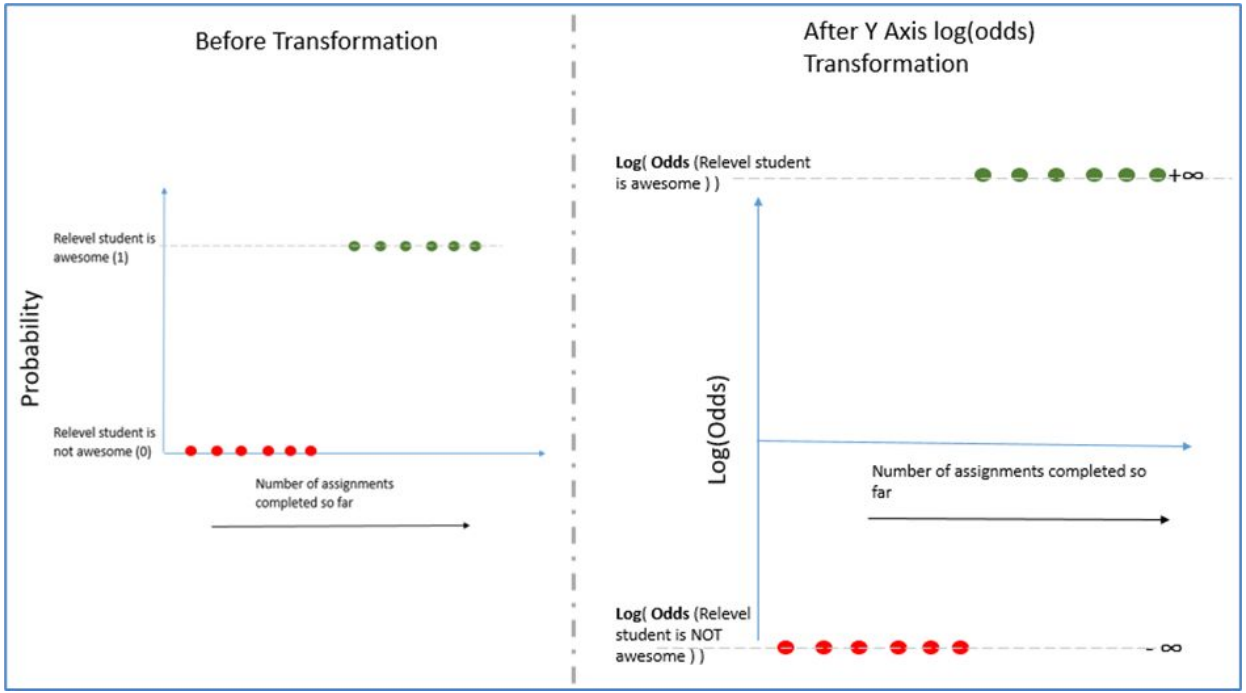
Logistic Regression uses the log(odds) on y-Axis. This means that the Probability on Y axis is transformed to log(odds) using Logit function discussed above.

Reason for Transformation : The reason for using log(odds) is because we know it makes things symmetrical and maths easier. By this transformation we can fit a straight line which can be transformed back to S curve with the help of Logit function. We cannot fit an S curve directly, and this is why Y Axis is transformed to log(odds).

Effect on datapoints : Subsequently, this would mean that now data points which earlier had 1 and 0 probabilities in earlier chart would have values as +∞ an - ∞ respectively.

Logistic Regression

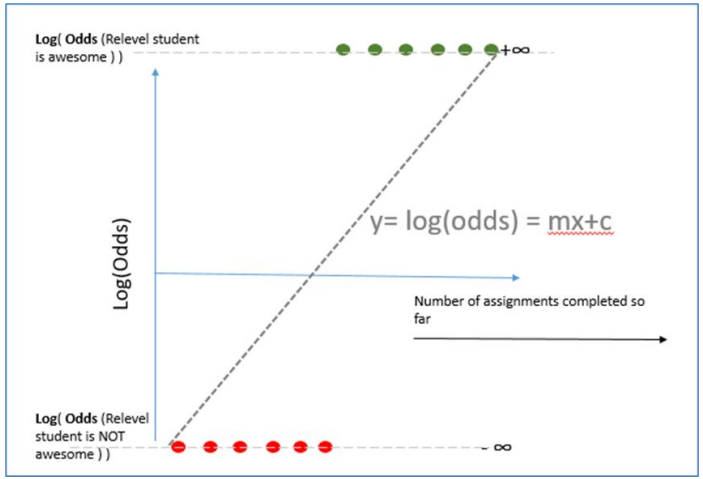
This transformation is shown in visuals below for better grasp :



Logistic Regression

Step 2 :

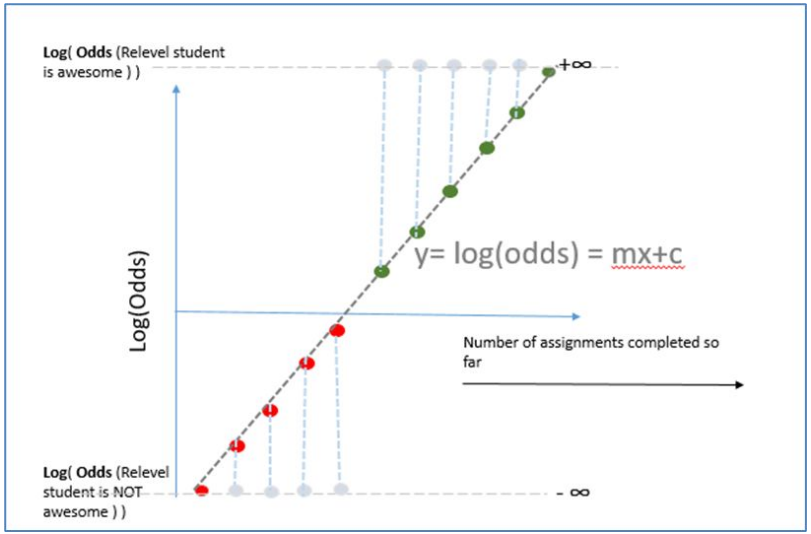
An arbitrary line is drawn on the transformed axis, with the objective as to find the best fit line.



(Later on, the best fit is chosen as per Maximum Likelihood calculation (Explained later) and Gradient Descent process.)

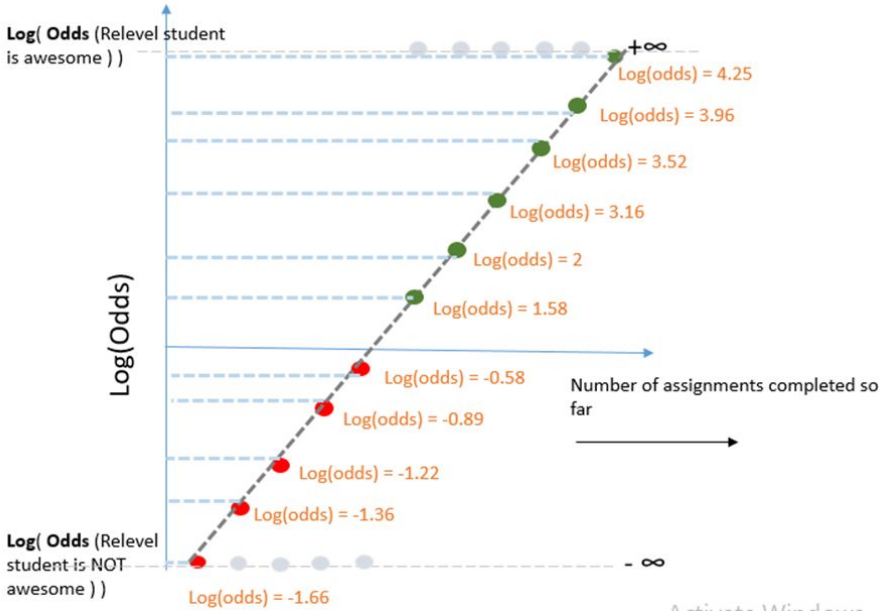
Logistic Regression

Since the datapoints are at extreme ends on this axis (+∞ an - ∞) , they are projected on to the line as shown in the visual below.



Logistic Regression

After projection, each data point will have a corresponding log(odds) value.



Logistic Regression

Step 3 :

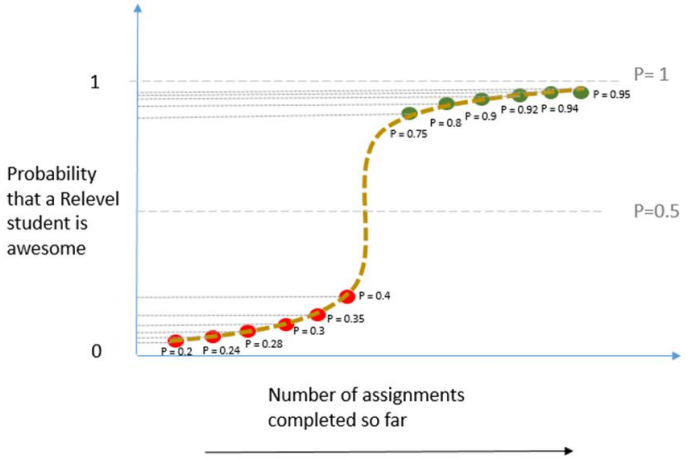
For each datapoint, log(odds) can be converted in Probabilities (P) by using Sigmoid function –

$$P = \frac{e^{log(odds)}}{1 + e^{log(odds)}}$$

(Sigmoid function is nothing but Logit function rearranged. It is the inverse of Logit function.)

This formula is just a reordering of earlier Logit function that we discussed, let's skip derivation to save time.

Geometrically, this function is transforming back out log(odds) axis to Probabilities. Our Straight line is convert into an S-Curve with the help of this function.



Logistic Regression

Step 4 :

Calculate Likelihood of this particular Log(odds) line or S Curve.
Likelihood of the line or S Curve is given by multiplication of all the Probabilities for all the datapoints as following –

Likelihood =
0.95 * 0.94 * 0.92 * 0.9 * 0.8 * 0.75 * (1-0.4) * (1-0.35) * (1-0.3) * (1-0.28) * (1-0.24) * (1-0.2)
=> 0.0530119
Please observe that for negative class, (1-P) is used in likelihood formula as for those point we have to calculate Probability of those being 0.

Note:
Although for the sake of simplicity we have used Multiplication of Probabilities to calculate likelihood, but mostly statisticians prefer multiplication of log(P) instead. This is because use of log(P) makes the calculations simpler. A lot of computer based algorithm functions also use the same to save computation.
But, the result would be same in any case and the choice of log or not won't impact the final outcome at all.

Logistic Regression

In case of log(P) based likelihood, the calculation would be as follows –

Likelihood =

=> log(0.95) * log(0.94) * log(0.92) * log(0.9) * log(0.8) * log(0.75) * log(1-0.4) * log(1-0.35) * log(1-0.3) * log(1-0.28) * log (1-0.24) * log (1-0.2)

=> Log(0.95 + 0.94 + 0.92 + 0.9 + 0.8 + 0.75 + (1-0.4) + (1-0.35) + (1-0.3) + (1-0.28) + (1-0.24) + (1-0.2))
=> 3.2464

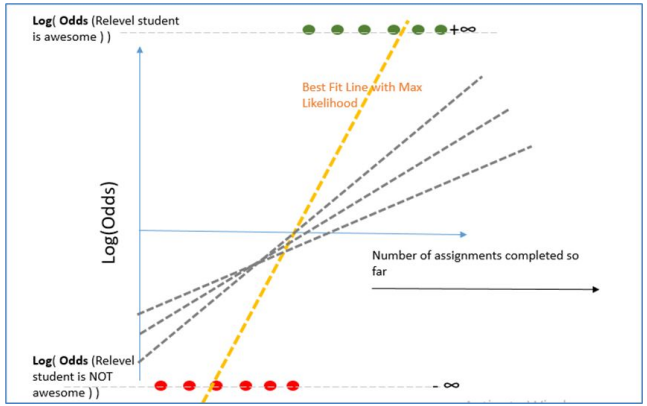
Logistic Regression

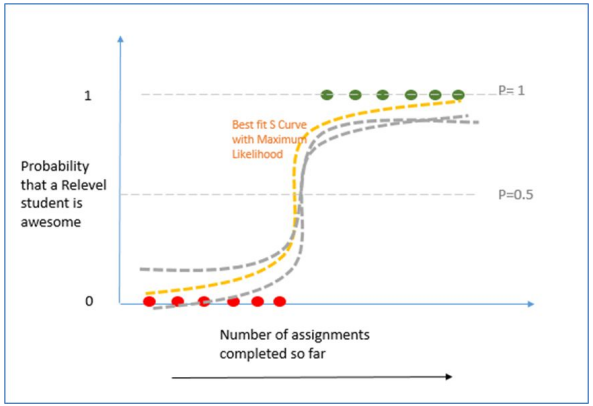
Step 5 :

Now the initial straight line with y axis as log(odds)) will be rotated as per Gradient Descent process discussed earlier. A new straight line will be obtained, again projections would be taken and maximum likelihood would be calculated for the new S Curve obtained.

This iterative process will happen until the algorithm will reach minima on Gradient Descent, and that minima will be the configuration at which we'll get maximum likelihood.

At that configuration, the model will have its best-fit line on log(odds) axis ; and corresponding S-Curve on original axis with y axis as Probabilities.





Logistic Regression

As can be seen geometrically in above charts, there will be multiple iterations as per Gradient Descent method, and the curve/line with Maximum Likelihood will be finally selected.

Note : By default the threshold P value is considered to be 0.5. It means that all the data points having P>0.5 will be classified as 1 and otherwise 0. But this can be adjusted as per the requirement to be slightly higher or lower as well.

Advantages and Disadvantages :

Advantages:

1. Logistic Regression is very efficient in predicting target as compared to alternatives.
2. Logistic regression is less prone to Overfitting unlike Decision trees and KNN (alternatives)
3. Logistic Regression is scalable, ie. It works well with large datasets.
4. Logistic Regression doesn't necessitates feature scaling.

Disadvantages :

1. It assumes that there is a linear relation between a feature and target variable. In real world datasets, that is not usually the case, hence applicability of logistic regression goes for a toss.

ROC Curve and AUC

ROC stands for **Receiver Operating Characteristic Curve.**

What is it used for ?

So far, we have taken a default value of P = 0.5 as threshold value in our discussion. But this value can in fact by modified as per a specific use case. Some problems might require a higher threshold value such as P = 0.7 as some might require a lower value such as P = 0.4
It basically ties back to the same question – What is more important for you ? Reducing Type 1 errors or Type 2 errors? Previously discussed Spam and Cancer examples would apply here again.
ROC Curve can help in the same by choosing the right threshold P value.

Examples –

- For a SPAM classifier, FPR need to be really low, even if it comes at a cost of reduced accuracy. We don't want someone to miss an important mail just because the classifier was too strict. It is okay to allow some SPAM messages as a cost.
- Again for a Disease classifier, TPR must be high, even if it comes at a cost of wrongly classifying fir people as sick.

The decision for optimal threshold can be taken by ROC curve.

AUC is simply the area under the ROC curve, the higher, the better. The maximum value for AUC can be 1.

ROC Curve and AUC

What is it exactly and how it is plotted ?

It is essentially a plot of set of values of (TPR,FPR) , obtained by varying values of threshold P.

$$TPR = \frac{TP}{(TP + TN)}$$

$$FPR = \frac{FP}{(FP + FN)}$$

In order to understand it better, let's assume we have following estimated probabilities for a test set :

Actual Class Label	Estimated Probability
0	0.41
0	0.43
0	0.46
0	0.44
0	0.49
0	0.54
0	0.59
1	0.61
1	0.66
1	0.71

ROC Curve and AUC

Now , **if the threshold P = 0.5** , the output of the model will be as following –

Actual Class Label	Estimated Probability	P (threshold) = 0.5
0	0.41	0
0	0.43	0
0	0.46	0
0	0.44	0
0	0.49	0
0	0.54	1
0	0.59	1
1	0.61	1
1	0.66	1
1	0.71	1

As per the predictions , respective Accuracy, TPR and FPR is –

TPR	0.29
FPR	0.375
Accuracy	0.2

ROC Curve and AUC

Now let's change default P = 0.55 (A bit higher) ; we get following results -

Actual Class Label	Estimated Probability	P (threshold) = 0.5	P (threshold) = 0.55
0	0.41	0	0
0	0.43	0	0
0	0.46	0	0
0	0.44	0	0
0	0.49	0	0
0	0.54	1	0
0	0.59	1	1
1	0.61	1	1
1	0.66	1	1
1	0.71	1	1

TPR	0.14
FPR	0.33
Accuracy	0.1

ROC Curve and AUC

Now again, one last time, let's reduce P=value to 0.45 this time ->

Actual Class Label	Estimated Probability	P (threshold) = 0.5	P (threshold) = 0.55	P (threshold) = 0.45
0	0.41	0	0	0
0	0.43	0	0	0
0	0.46	0	0	1
0	0.44	0	0	0
0	0.49	0	0	1
0	0.54	1	0	1
0	0.59	1	1	1
1	0.61	1	1	1
1	0.66	1	1	1
1	0.71	1	1	1

TPR	0.57
FPR	0.5
Accuracy	0.4

ROC Curve and AUC

Summarising all the results for different P values –

	P (threshold) = 0.5	P (threshold) = 0.55	P (threshold) = 0.45
TPR	0.57	0.29	0.14
FPR	0.50	0.38	0.33
Accuracy	0.40	0.20	0.10

Conclusion –

P =0.5 :

Highest TPR (That's goof), High FPR (That's bad) , Best Accuracy

P =0.55 :

Medium TPR (That's ok), Medium FPR (That's ok) , Medium Accuracy

P =0.45 :

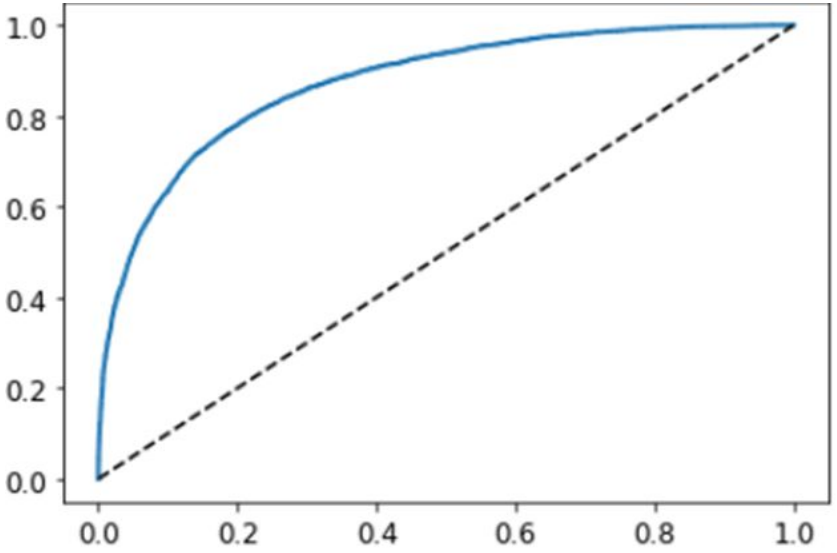
Lowest TPR (That's bad), Lowest FPR (That's good) , Lowest Accuracy

Now as per the use case, the choice of Threshold P value can be made.

ROC Curve and AUC

Plotting ROC Curve –

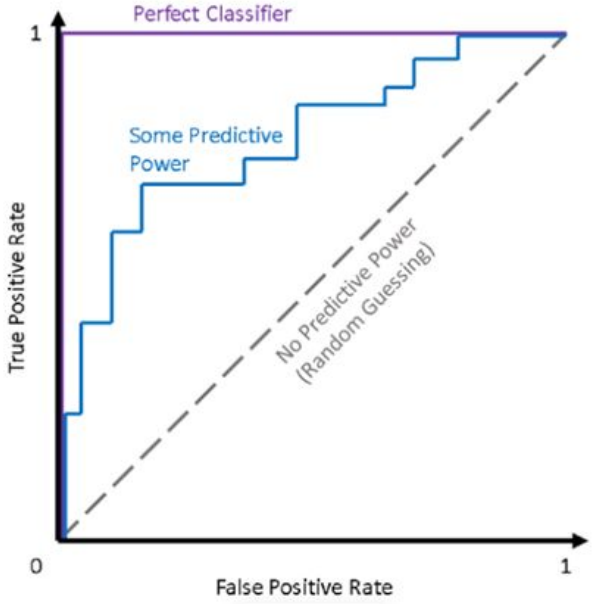
ROC Curve is nothing but a plot of highlighted TPR and FPR values for all P values ranging from 0 to 1. It looks something like this when plotted using Python –



ROC Curve and AUC

Characteristics of ROC – AUC :

- Better the AUC (Area under ROC) , better a model is. Ideal or maximum value is 1.
- The more an ROC is tilted towards upper left corner of the plot, better a model is.



Logistic Regression implementation in Python

We'll be using Voice sample data of about 3000 males and females. The Target variable would be Male/Female , as the objective is to predict whether a person is Male or Female based on their voice characteristics.

Let's first load and Process the data before we train our Logistic Regression model as below –

Loading and Preparing Voice data for Model training –

Reading CSV data file

The file contains characteristics data about 3168 voice samples for Males and Females. There are 20 Features and 1 Target variable Target variable is binary - Male/Female

```
voice = pd.read_csv("D:\\Relevel\\Week 14_Day3\\voice.csv")
voice.head()
```

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	...	centroid	meanfun	minfun	maxfun	meandc
0	0.059781	0.064241	0.032027	0.015071	0.090193	0.075122	12.863462	274.402906	0.893369	0.491918	...	0.059781	0.084279	0.015702	0.275862	0.0078
1	0.066009	0.067310	0.040229	0.019414	0.092666	0.073252	22.423285	634.613855	0.892193	0.513724	...	0.066009	0.107937	0.015826	0.250000	0.0090
2	0.077316	0.083829	0.036718	0.008701	0.131908	0.123207	30.757155	1024.927705	0.846389	0.478905	...	0.077316	0.098706	0.015656	0.271186	0.0079
3	0.151228	0.072111	0.158011	0.096582	0.207955	0.111374	1.232831	4.177296	0.963322	0.727232	...	0.151228	0.088965	0.017798	0.250000	0.2014
4	0.135120	0.079146	0.124656	0.078720	0.206045	0.127325	1.101174	4.333713	0.971955	0.783568	...	0.135120	0.106398	0.016931	0.266667	0.7128

5 rows × 21 columns

Logistic Regression implementation in Python

Objective ¶

The Objective of this exercise is to train a model which predicts whether a voice sample belongs to a Male or a Female

Exploring data

```
voice.shape
```

```
(3168, 21)
```

```
voice.describe()
```

	meanfreq	sd	median	Q25	Q75	IQR	skew	kurt	sp.ent	sfm	mode	ce
count	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.000000	3168.0
mean	0.180907	0.057126	0.185621	0.140456	0.224765	0.084309	3.140168	36.568461	0.895127	0.408216	0.165282	0.1
std	0.029918	0.016652	0.036360	0.048680	0.023639	0.042783	4.240529	134.928661	0.044980	0.177521	0.077203	0.0
min	0.039363	0.018363	0.010975	0.000229	0.042946	0.014558	0.141735	2.068455	0.738651	0.036876	0.000000	0.0
25%	0.163662	0.041954	0.169593	0.111087	0.208747	0.042560	1.649569	5.669547	0.861811	0.258041	0.118016	0.1
50%	0.184838	0.059155	0.190032	0.140286	0.225684	0.094280	2.197101	8.318463	0.901767	0.396335	0.186599	0.1
75%	0.199146	0.067020	0.210618	0.175939	0.243660	0.114175	2.931694	13.648905	0.928713	0.533676	0.221104	0.1
max	0.251124	0.115273	0.261224	0.247347	0.273469	0.252225	34.725453	1309.612887	0.981997	0.842936	0.280000	0.2

Logistic Regression implementation in Python

Checking Null values and data types

```
voice.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3168 entries, 0 to 3167
Data columns (total 21 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   meanfreq  3168 non-null   float64
 1   sd        3168 non-null   float64
 2   median    3168 non-null   float64
 3   Q25       3168 non-null   float64
 4   Q75       3168 non-null   float64
 5   IQR       3168 non-null   float64
 6   skew      3168 non-null   float64
 7   kurt      3168 non-null   float64
 8   sp.ent    3168 non-null   float64
 9   sfm       3168 non-null   float64
 10  mode      3168 non-null   float64
 11  centroid  3168 non-null   float64
 12  meanfun   3168 non-null   float64
 13  minfun    3168 non-null   float64
 14  maxfun    3168 non-null   float64
 15  meandom   3168 non-null   float64
 16  mindom    3168 non-null   float64
 17  maxdom    3168 non-null   float64
 18  dfrange   3168 non-null   float64
 19  modindx   3168 non-null   float64
 20  label     3168 non-null   object
dtypes: float64(20), object(1)
memory usage: 519.9+ KB
```

Conclusion - There are no null values

Logistic Regression implementation in Python

Replacing Target Variable values with 0 and 1 because this binary forms is compatible with most algorithms

```
voice.label = [1 if each == "female" else 0 for each in voice.label]
```

```
voice.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3168 entries, 0 to 3167
Data columns (total 21 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   meanfreq  3168 non-null   float64
 1   sd        3168 non-null   float64
 2   median    3168 non-null   float64
 3   Q25       3168 non-null   float64
 4   Q75       3168 non-null   float64
 5   IQR       3168 non-null   float64
 6   skew      3168 non-null   float64
 7   kurt      3168 non-null   float64
 8   sp.ent    3168 non-null   float64
 9   sfm       3168 non-null   float64
 10  mode      3168 non-null   float64
 11  centroid  3168 non-null   float64
 12  meanfun   3168 non-null   float64
 13  minfun    3168 non-null   float64
 14  maxfun    3168 non-null   float64
 15  meandom   3168 non-null   float64
 16  mindom    3168 non-null   float64
 17  maxdom    3168 non-null   float64
 18  dfrange   3168 non-null   float64
 19  modindx   3168 non-null   float64
 20  label     3168 non-null   int64
dtypes: float64(20), int64(1)
memory usage: 519.9 KB
```

Observation - The Target variable datatype has changed to int64

Logistic Regression implementation in Python

Preparing Test and Train data

Segregating Target variable and features into x and y

In [94]:
```
#We should have x and y values for test-train datas.
y = voice.label.values
x = voice.drop(["label"],axis=1)
```

Performing Train test split finally with 80/20 ratio.

In [95]:
```
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x_normalised,y,test_size=0.2,random_state = 100)
#test_size=0.2 means %20 test datas, %80 train datas
```

Our data is ready for training now.

Logistic Regression implementation in Python

Logistic Regression model training and Performance measurement –

1. Logistic Regression

Training our Logistic Regresion model using LogisticRegression() function of sklearn library.

For students to explore - GridSearchCV is used for Hyperparameter tuning and Cross Validation.

Also in logistic regression function , C-Value has following significance --

C -Value : Strength of regularisation. A high value of C gives more weight to training data. A lower C value means that the data might not represent real world data and less importance is given to data. The ideal value is usually found as per an iterative process or via GridSearchCV function

In [96]:
```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(C=100, random_state=0)
log_reg = log_reg.fit(x_train,y_train)
log_reg
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[96]:
```
LogisticRegression(C=100, random_state=0)
```

Logistic Regression implementation in Python

```
In [97]: y_pred_test = log_reg.predict(x_test)
         y_pred_test
```

```
Out[97]: array([0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1,
                1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0,
                1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
                1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1,
                1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1,
                1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1,
                1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0,
                1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0,
                0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1,
                0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1,
                0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1,
                0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
                1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1,
                0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0,
                1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0,
                0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0,
                0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1,
                0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0,
                1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1,
```

Logistic Regression implementation in Python

P Values associated with each prediction can also be checked using predict_proba function

In [98]:
```
# Probability values given by Logistic regression algorythm

log_reg.predict_proba(x_test)[:,0]
```

Out[98]:
```
array([9.96181259e-01, 9.85574169e-01, 9.94907880e-01, 1.26343139e-04,
       1.24233757e-03, 1.85442402e-05, 2.79274832e-02, 2.03090459e-02,
       9.81738401e-01, 9.57129014e-01, 9.95632141e-01, 1.43056862e-04,
       8.31410299e-01, 7.04206379e-03, 1.00691302e-01, 4.99837339e-02,
       6.81131823e-04, 9.97203528e-01, 9.97377075e-01, 8.38780357e-05,
       9.99018832e-01, 3.21971470e-07, 1.19663101e-03, 1.63671206e-03,
       8.09851007e-01, 9.94487345e-01, 9.99546036e-01, 9.94019281e-01,
       2.42151664e-05, 9.95989192e-01, 8.88114487e-01, 7.68860811e-04,
       8.49965417e-04, 3.10299971e-05, 8.87694012e-04, 6.39041212e-02,
       7.88198899e-04, 8.11857285e-01, 2.72210010e-01, 9.96396049e-01,
       1.09452536e-04, 9.99498773e-01, 9.99095730e-01, 8.25848242e-01,
       2.50542984e-05, 1.39359410e-02, 9.20939180e-01, 1.10696961e-01,
       2.43691341e-07, 9.98102381e-01, 9.91765406e-01, 1.16996036e-03,
       9.96220963e-01, 3.72963422e-04, 7.59727631e-02, 9.92051170e-01,
       7.60948377e-03, 9.95919641e-01, 8.77813882e-01, 9.92227752e-01,
       9.68649213e-01, 6.41481176e-04, 9.38304239e-01, 1.75400575e-01,
       1.10099788e-01, 1.34960069e-04, 2.18692469e-02, 9.98175715e-01,
       7.90482569e-02, 9.99924032e-01, 3.04387280e-02, 1.70875514e-02,
       9.86930047e-01, 9.99428547e-01, 6.46762119e-01, 9.85879133e-01,
```

Logistic Regression implementation in Python

Model Predictions and Performance Evaluation

sklearn.metrics library of sklearn provides a lot of useful performance metrics accuracy_score is one of them to calculate accuracy

1. Accuracy

Accuracy for Test Set

In [99]:
```
from sklearn.metrics import accuracy_score

print('Model accuracy score ( Test set ): {0:0.4f}'. format(accuracy_score(y_test, y_pred_test)))
```

```
Model accuracy score ( Test set ): 0.9637
```

Predictions & Accuracy for Test Set

In [100]:
```
y_pred_train = log_reg.predict(x_train)

print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

```
Training-set accuracy score: 0.9755
```

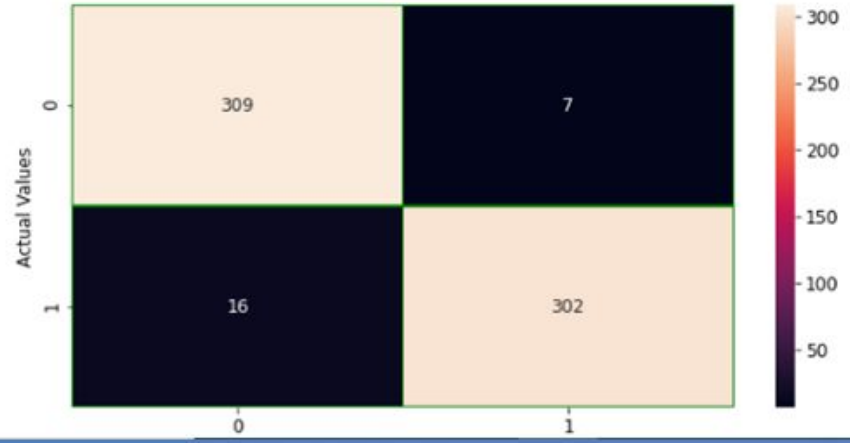
Since both train and test accuracies are comparable and close to each other, we can conclude that model is not overfitted.

Logistic Regression implementation in Python

2. Confusion Matrix

The confusion matrix can be plotted as per below code -

In [101]:
```
conf_mat = confusion_matrix(y_test,y_pred_test)                    # Getting confusuion matrix

#Visualization Confusion Matrix
f, ax = plt.subplots(figsize=(10,5))
sns.heatmap(conf_mat,annot=True,linewidths=0.5,linecolor="green",fmt=".0f",ax=ax)
plt.xlabel("Predicted Values")
plt.ylabel("Actual Values")
plt.show()
```



Logistic Regression implementation in Python

3. Precision, Recall an F1 Score

classification_report is a very useful function which provides all the metrics in one go

In [102]:
```
from sklearn.metrics import classification_report


print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96       316
           1       0.98      0.95      0.96       318

    accuracy                           0.96       634
   macro avg       0.96      0.96      0.96       634
weighted avg       0.96      0.96      0.96       634
```

Logistic Regression implementation in Python

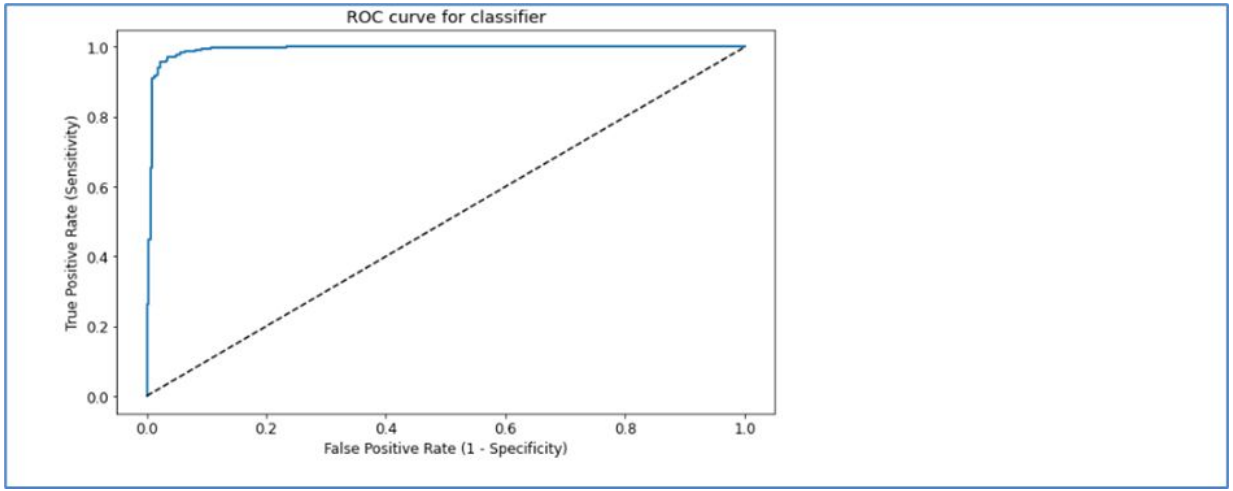
3. ROC AUC

Again, using sklearn.metrics's roc_curve function, the required data to plot ROC and calculate AUC can be obtained

In [103]:
```
# Calculating data required to plot ROC

from sklearn.metrics import roc_curve


y_pred1 = log_reg.predict_proba(x_test)[:,1]
y_pred1 = y_pred1.reshape(-1,1)

fpr, tpr, thresholds = roc_curve(y_test, y_pred1, pos_label = 1)
```

In [104]:
```
### Plotting ROC

plt.figure(figsize=(10,6))

plt.plot(fpr, tpr, linewidth=2)

plt.plot([0,1], [0,1], 'k--' )

plt.rcParams['font.size'] = 12

plt.title('ROC curve for classifier')

plt.xlabel('False Positive Rate (1 - Specificity)')

plt.ylabel('True Positive Rate (Sensitivity)')

plt.show()
```

Logistic Regression implementation in Python



Area under the ROC

[105]:
```
from sklearn.metrics import roc_auc_score

ROC_AUC = roc_auc_score(y_test, y_pred1)

print('ROC AUC : {:.4f}'.format(ROC_AUC))
```

```
ROC AUC : 0.9918
```

Since AUC is very close to 1, it can be concluded that model can almost perfectly classify postive & negative class.

THANK YOU