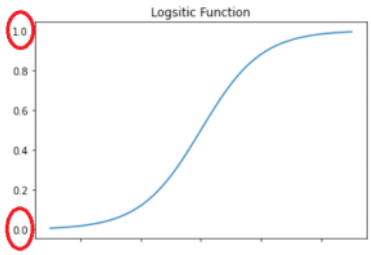
Introduction to Logistic Regression

07 October 2022 10:01

- Logistic regression is a classification algorithm that **predicts the probability of an outcome** that can only have **two values** (i.e. a dichotomy).
- A logistic regression produces a **logistic curve**, which is limited to values between **0 and 1.**



- As the X-value increases , Y gets more probability moving towards 1.0
- Negative value of X, Y tends to be lower probability.

- Logistic regression models the probability that each input belongs to a particular category.
- Logistic regression is an excellent tool to know for classification problems, which are problems where the output value that we wish to predict only takes on only a small number of discrete values.
- It can be used for both the Binary and multiple classification problems.

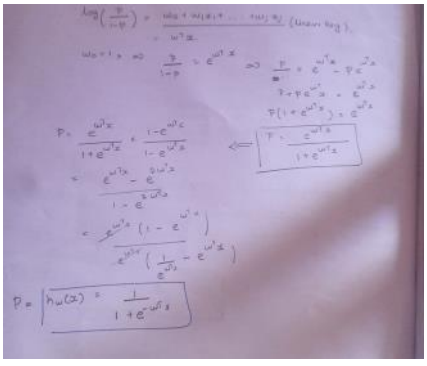
<u>Let we focus on the Binary classification problem, where the output can be only two distinct values.</u>

$$\log\left(\frac{p}{1-p}\right) = w_0 + w_1 x_1, \ldots, w_j x_j$$
$$= w^T x$$

where: This is called "**Logit Function**"

- w0 is the <u>intercept term</u>, and w1 to wj represents the parameters for all the other features (a total of j features).
- By convention of we can assume that w0=1 , so that we can re-write the whole thing using the matrix notation wTx .
- P = Event Happened.
- (1-P) = Event not Happened.

The "**Logit Function**" Can be rearranged as "**Logistic Function**":



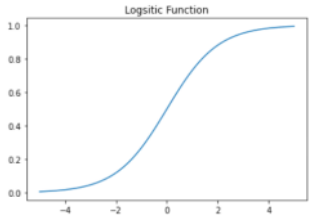
(For Understanding the Maths Behind)

$$p = \frac{e^{w^T x}}{1 + e^{w^T x}}$$

- By Plot using the Logistic Function. (or) most common form

$$h_w(x) = \frac{1}{1 + e^{-w^T x}}$$

```
x_values = np.linspace(-5, 5
y_values = [1 / (1 + np.exp(
plt.plot(x_values, y_values)
plt.title('Logsitic Function')
plt.show()
```



- Value of X can be from 0 to infinity
- For any value of X, we get **"Sigmoid Function"**
- When utilizing logistic regression, we are trying to lea e the probability of correctly classifying our glasses

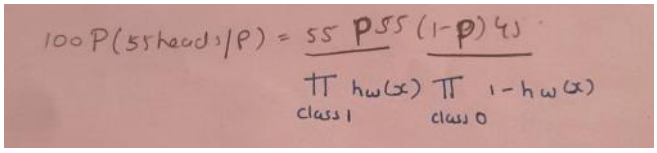
1	Logistic Regression is a _____ regression technique that is used to model data having a _____ outcome.

Your Answer	Linear, binary
Correct Answer	Non-linear , binary

Defining the cost function for Logistic Regression

08 October 2022 13:17

Maximum Likelihood Estimation(MLE):

- MLE is the method of estimating the parameter of assumed probability distribution, given some observed data.
- This is achieved by maximizing the likelihood function so that, under the assumed statistical model, the observed data is most probable.
- MLE for parameter(p) is the value of (p) that max the likelihood Probability P(data(p))
    - MLE is the value of p for which data is most likely.
    - e.g.
    - 

Coming to Logistic Regression,

- When utilizing logistic regression, we are trying to learn the w values in order to maximize the probability of correctly classifying our classes.

Let's say someone did give us some w values of the logistic regression model, how would we determine if they were good values or not?

- What we would hope is that for the household of class 1, the probability values are close to 1 and for the household of class 0 the probability is close to 0

But we don't care about getting the correct probability for just one observation, we want to correctly classify all our observations.

- If we assume our data are independent and identically distributed (think of it as all of them are treated equally),
- we can just take the product of all our individually calculated probabilities and that becomes the objective function we want to maximize.

By Maths,

$$\prod_{class1} h_w(x) \prod_{class0} 1 - h_w(x)$$

- The $\prod$ symbol means take the product of the hw(x) for the observations that are classified as that class.
- You will notice that for observations that are labelled as class 0, we are taking 1 minus the logistic function.
- That is because we are **trying to find a value to maximize**, and since observations that are labelled as class 0 should have a probability close to zero, 1 minus the probability should be close to 1.
- This procedure is also known as the **maximum likelihood estimation**.

Next we will re-write the original cost function as:

$$\ell(w) = \sum_{i=1}^{N} y_i log(h_w(x_i)) + (1 - y_i) log(1 - h_w(x_i))$$

where:

- We define **yi to be 1** when the ith observation is labeled **class 1** and **yi = 0** when labeled as **class 0**,
- The N simply represents the total number of the data.
- then we only compute **hw(xi)** for observations that are labeled **class 1** and
- **1—hw(xi)** for observations that are labeled **class 0**, which is still the same idea as the original function.
- Next we'll **transform** the original hw(xi) by **taking the log**.
- As we'll later see this logarithm transformation <u>will make our cost function more convenient to work with</u>,

Why Logarithm?

- logarithm is a monotonically increasing function, the logarithm of a function achieves its maximum value at the same points as the function itself.
- When we take the log, our product across all data points, it becomes a sum.

Often times you'll also see the notation above be simplified in the form of a maximum likelihood estimator:

$$\ell(w) = \sum_{i=1}^{N} log\big(P(y_i \mid x_i, w)\big)$$

- The equation above simply denotes the idea that , w represents the parameters we would like to estimate the parameters w by maximizing conditional probability of yi given xi .

<u>Now by definition of probability in the logistic regression model:</u>

$$h_w(x_i) = \frac{1}{1+e^{-w^T x_i}} \text{ and } 1 - h_w(x_i) = \frac{e^{-w^T x_i}}{1+e^{-w^T x_i}}.$$

By substituting these expressions into our ℓ(w) equation and simplifying it further we can obtain a simpler expression,

$$
\begin{aligned}
\ell(w) &= \sum_{i=1}^{N} y_i log(h_w(x_i)) + (1-y_i)log(1-h_w(x_i)) \\
&= \sum_{i=1}^{N} y_i log(\frac{1}{1+e^{-w^T x_i}}) + (1-y_i)log(\frac{e^{-w^T x_i}}{1+e^{-w^T x_i}}) \\
&= \sum_{i=1}^{N} -y_i log(1+e^{-w^T x_i}) + (1-y_i)(-w^T x_i - log(1+e^{-w^T x_i})) \\
&= \sum_{i=1}^{N} (y_i - 1)(w^T x_i) - log(1+e^{-w^T x_i})
\end{aligned}
$$

Gradient Descent on Logistic Cost function

08 October 2022 13:18

Previously,

l(w) =

$$= \sum_{i=1}^{N} (y_i - 1)(w^T x_i) - log(1 + e^{-w^T x_i})$$

- Now that we obtain the formula to assess our algorithm,
- we'll dive into the meat of the algorithm, which is to derive the gradient for the formula (the derivative of the formula with respect to each coefficient)
- And it turns out the derivative of log likelihood with respect to a single coefficient wj is as follows (the form is the same for all coefficients):

$$\frac{\partial \ell(w)}{\partial w_j} = \sum_{i=1}^{N} (x_{ij}) \left(y_i - \frac{1}{1 + e^{-w^T x_i}} \right)$$

To compute it, you simply need the following two terms:

- $\left(y_i - \frac{1}{1+e^{-w^T x_i}} \right)$ is the vector containing the difference between the predicted probability and the original label.
- x_{ij} is the vector containing the j_{th} feature's value.

<u>Updating model parameters using Gradient Descent:</u>

- Recall our task is **to find the optimal value** for each individual weight to lower the cost.
- This requires taking the **partial derivative of the cost/error function** with respect to a single weight, and then running gradient descent for each individual weight to update them.
- Thus, for any individual weight wj , we'll compute the following:

$$w_j^{(t+1)} = w_j^{(t)} + \alpha * \sum_{s=i}^{i+B} \frac{\partial \ell_s(w)}{\partial w_j}$$

where:

- α denotes the learning rate or so called step size, in other places you'll see it denoted as η .
- w(t)j denotes the weight of the jth feature at iteration t .
- And we'll do this iteratively for each weight, many times, until the whole network's cost function is minimized.

Implementation of Logistic Regression in Python

08 October 2022 13:19

Labelled Data	**Unlabelled Data**
• Data in which given with Yes/No • Or with 0/1	• Data without labelling • After analysing we may get the label E.g. - We categorise the data like, - P1 if the customer inactive for 3 months - P2 if the customer inactive for 6 months - P3 if the customer inactive for 12 days The way this P1, P2, P3 are created is called **Buckets**

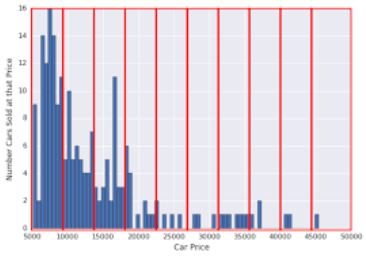
Creation of Buckets:

- As we discussed from the above table
- Which categorise the data.

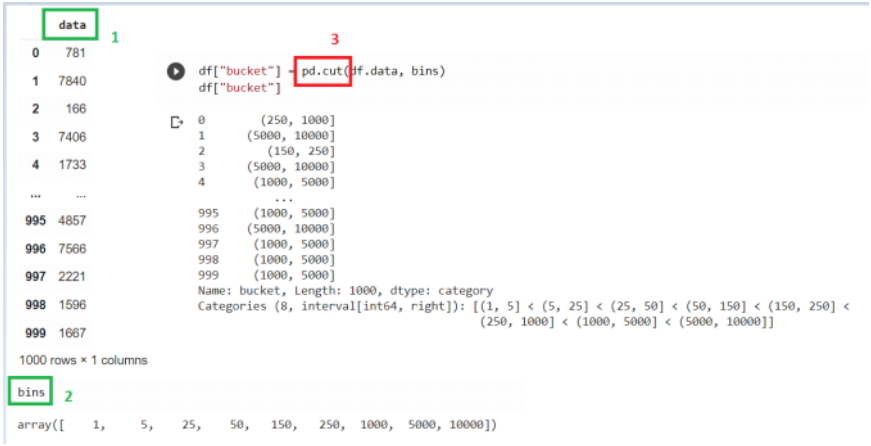
Definition:

- Bucketing is a process in machine learning which is used to convert a feature into multiple binary features called buckets or bins, and this is typically based on value range.

E.g. The Number of cars that sold in that price, ranging from 5000-10000, 10000-15000, 20000-25000 etc.,



- Buckets/Bin can be created using the python function called **"CUT"**
- E.g.



Imbalanced Dataset:

- Oftentimes in practical machine learning problems there will be significant differences in the rarity of different classes of data being predicted.
- For example, when detecting cancer we can expect to have datasets with large numbers of false outcomes, and a relatively smaller number of true outcomes.
- The overall performance of any model trained on such data will be constrained by its ability to predict rare points.
- In problems where these rare points are only equally important or perhaps less important than non-rare points,
- this constraint may only become significant in the later "tuning" stages of building the model.
- But in problems where the rare points are important, or even the point of the classifier (as in a cancer example), dealing with their scarcity is a first-order concern for the mode builder.

Several different techniques exist in the practice for dealing with imbalanced dataset. The most naive class of techniques is **sampling**: changing the data

presented to the model by **under sampling** common classes, **oversampling** (duplicating) rare classes, **or both**.

Synthetic Minority Oversampling Technique- SMOTE

A problem with imbalanced classification is that there are too few examples of the minority class for a model to effectively learn the decision boundary.

- One way to solve by oversampling
- achieved by simply duplicating examples from the minority class in the training dataset prior to fitting a model
- This can balance the class distribution but does not provide any additional information to the model.

SMOTE:

- SMOTE works by selecting examples that are close in the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line.

- Specifically, a random example from the minority class is first chosen. Then k of the nearest neighbors for that example are found (typically k=5). A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space.

Telco Customer Churn data:

What is Churn Prediction?

Churn prediction is analytical studies on the possibility of a customer abandoning a product or service. The goal is to understand and take steps to change it before the costumer gives up the product or service.

About Data

- customerID : Customer ID
- gender : Whether the customer is a male or a female
- SeniorCitizen : Whether the customer is a senior citizen or not (1, 0)
- Partner : Whether the customer has a partner or not (Yes, No)
- Dependents : Whether the customer has dependents or not (Yes, No)
- tenure : Number of months the customer has stayed with the company
- PhoneService : Whether the customer has a phone service or not (Yes, No)
- MultipleLines : Whether the customer has multiple lines or not (Yes, No, No phone service)
- InternetService : Customer's internet service provider (DSL, Fiber optic, No)
- OnlineSecurity : Whether the customer has online security or not (Yes, No, No internet service)
- OnlineBackup : Whether the customer has online backup or not (Yes, No, No internet service)

- DeviceProtection : Whether the customer has device protection or not (Yes, No, No internet service)
- TechSupport : Whether the customer has tech support or not (Yes, No, No internet service)
- StreamingTV : Whether the customer has streaming TV or not (Yes, No, No internet service)
- StreamingMovies : Whether the customer has streaming movies or not (Yes, No, No internet service)
- Contract : The contract term of the customer (Month-to-month, One year, Two year)
- PaperlessBilling : Whether the customer has paperless billing or not (Yes, No)
- PaymentMethod : The customer's payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))
- MonthlyCharges : The amount charged to the customer monthly
- TotalCharges : The total amount charged to the customer
- Churn : Whether the customer churned or not (Yes or No)

```
[ ] from google.colab import drive
    drive.mount('/content/drive')

    Mounted at /content/drive
```

Dataset initialization

```
[ ] data = pd.read_csv("/content/drive/MyDrive/Almabetter/Module 04 ML/WA_Fn-UseC_-Telco-Customer-Churn.csv")
    print("Dataset size")
    print("Rows {} Columns {}".format(data.shape[0], data.shape[1]))

    Dataset size
    Rows 7043 Columns 21
```

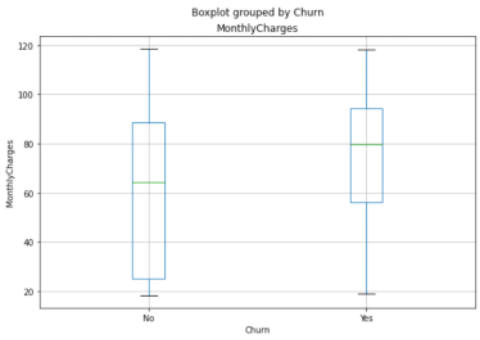
```
print("Columns and data types")
pd.DataFrame(data.dtypes).rename(columns = {0:'dtype'})
```

	dtype
customerID	object
gender	object
SeniorCitizen	int64
Partner	object
Dependents	object
tenure	int64
PhoneService	object
MultipleLines	object
InternetService	object
OnlineSecurity	object
OnlineBackup	object
DeviceProtection	object
TechSupport	object
StreamingTV	object
StreamingMovies	object
Contract	object
PaperlessBilling	object

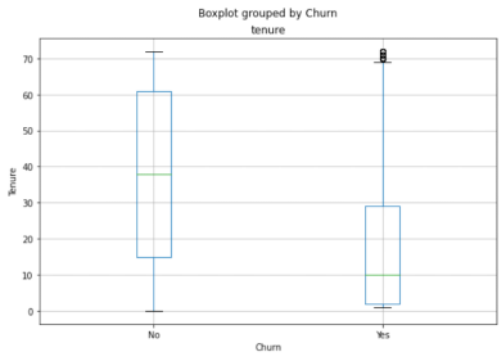
That's a lot of columns, to simplify our experiment we will only use 2 features tenure and MonthlyCharges and the target would be Churn of course. Let us do a simple EDA and visualization on our features and target.

1. EDA: Independent variables

```
fig = plt.figure(figsize=(9, 6))
ax = fig.gca()
df.boxplot(column = 'MonthlyCharges', by = 'Churn', ax = ax)
ax.set_ylabel("MonthlyCharges")
plt.show()
```




```
fig = plt.figure(figsize=(9, 6))
ax = fig.gca()
df.boxplot(column = 'tenure', by = 'Churn', ax = ax)
ax.set_ylabel("Tenure")
plt.show()
```



- As we Plotting the Monthly charges and Tenure with the churn data
- And finding number of data's in each variable against Yes or no in churn

Insights from our simple EDA:

- We can see a difference between our target classes on tenure as you can see in the second boxplot, which is good because our model (Logistic Regression) may use this to separate the two classes.
- There is only a slight difference between our target classes on monthly charges as shown in the first boxplot.

2. Converting the Churn feature:

- Here, the churn feature is in "string" datatype.
- We convert this string into "integer" as "0" and "1"
- Rename the column as "Class"

```
df['class'] = df['Churn'].apply(lambda x : 1 if x == "Yes" else 0)
```

3. Creating "X" and "y" :

```
# features will be saved as X and our target will be saved as y
X = df[['tenure','MonthlyCharges']].copy()
y = df['class'].copy()
```

4. Splitting the Data into Train and Test:

```
[17] from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split( X,y , test_size = 0.2, random_state = 0)
     print(X_train.shape)
     print(X_test.shape)

     (5634, 2)
     (1409, 2)
```

5. Fitting logistic regression on train data

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(fit_intercept=True, max_iter=10000)
clf.fit(X_train, y_train)
```

6. Evaluating the performance of the trained model

<u>Predicted Probability:</u>

```
[ ] # Get the predicted probabilities
    train_preds = clf.predict_proba(X_train)
    test_preds = clf.predict_proba(X_test)
```

What we get here is,

```
test_preds
```

```
array([[0.7145149 , 0.2854851 ],
       [0.78522641, 0.21477359],
       [0.53064776, 0.46935224],
       ...,
       [0.77288679, 0.22711321],
       [0.71618111, 0.28381889],
       [0.57740038, 0.42259962]])
```

The predict_proba() method:

- In the context of classification tasks, some sklearn estimators also implement the predict_proba method that **returns the class probabilities for each data point**.
- The method accepts a single argument that corresponds to the data over which the probabilities will be computed and returns an array of lists containing the class probabilities for the input data points.

The predict() method:

- All supervised estimators in scikit-learn implement the predict() method that can be executed on a trained model in order **to predict the actual label (or class) over a new set of data.**
- The method accepts a single argument that corresponds to the data over which the predictions will be made and it returns an array containing the predicted label for each data point.
- Default value of Probability is 0.5, when < 0.5 gives "0" and >0.5 gives "1"

```
[ ] # Get the predicted classes
    train_class_preds = clf.predict(X_train)
    test_class_preds = clf.predict(X_test)
```

```
[ ] train_class_preds
```

```
array([0, 0, 0, ..., 0, 1, 0])
```

The Accuracy Score:

- Accuracy is a scoring system in binary classification (i.e., determining if an answer or returned information is correct or not),
- sometimes used as an alternative to F-score, and it's calculated as: (True Positives + True Negatives) / (True Positives + True Negatives + False Positives + False Negatives).

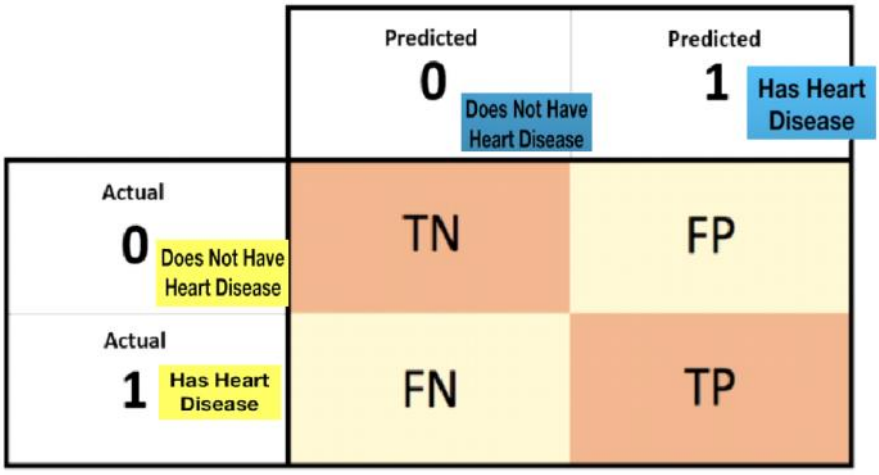
```
[ ] from sklearn.metrics import accuracy_score, confusion_matrix
    import seaborn as sns
```

```
[ ] # Get the accuracy scores
    train_accuracy = accuracy_score(train_class_preds,y_train)
    test_accuracy = accuracy_score(test_class_preds,y_test)

    print("The accuracy on train data is ", train_accuracy)
    print("The accuracy on test data is ", test_accuracy)
```

```
The accuracy on train data is  0.7857649982250621
The accuracy on test data is  0.7735982966643009
```

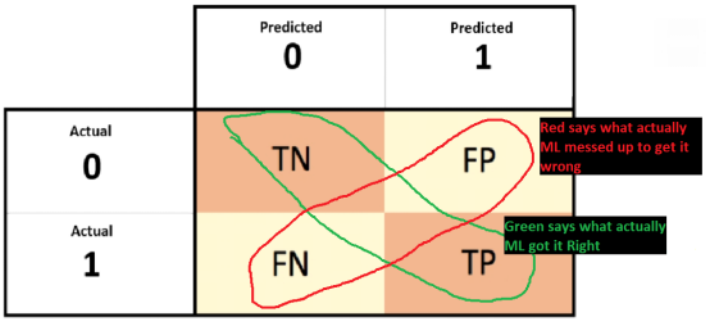
7. Confusion Matrix

- Confusion Matrix is a useful machine learning method which allows you to measure Recall, Precision, Accuracy, and AUC-ROC curve.



Example for Confusion matrix we can say,

- **True positive** - if the patient is "Actual heart disease" and ML "Predicted correctly he has heart disease"
- **False Negative** - if the patient is "Actual heart disease" and ML "Predicted wrongly as he doesn't have heart disease"
- **True Negative** - if the patient is "Actually does not have any heart disease" and ML "Predicted correctly that he doesn't have heart disease"
- **False Positive** - if the Patient is "Actually does not have any heart disease" and ML " Predicted Wrongly that he had heart disease"



- Size of the confusion matrix depends on how many we want to predict
- In this case we predicting for Binary (either have heart disease or not have heart disease)
- If we increase it to 3, we get the matrix tables in shape 3*3 and so on.

When did confusion matrix very helpful?

- When we have the dataset that can be trained using various algorithm such as Logistic, K nearest, Random forest.
- We can do the confusion matric for the each of the algorithm and calculate the more sophisticated matric such as sensitivity, specificity, ROC and AUC.
- By this metrics, find the best algorithm that can be used for the given dataset.

Note: Decision to consider which Evaluation metric to be taken (like precision or recall) is depends on the individual or the organisation.

Let we get the confusion matrix for both Training and Testing:

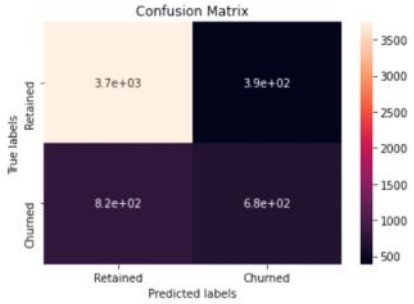
For Train

```
labels = ['Retained', 'Churned']
cm = confusion_matrix(y_train, train_class_preds)
print(cm)

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax) #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(labels)
ax.yaxis.set_ticklabels(labels)
```

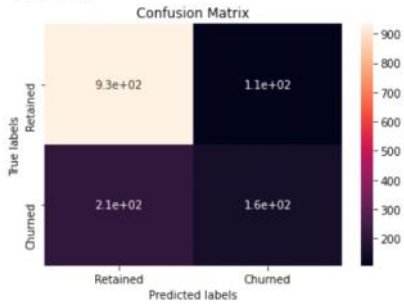
For Test

```
labels = ['Retained', 'Churned']
cm = confusion_matrix(y_test, test_class_preds)
print(cm)

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels')
ax.set_ylabel('True labels')
ax.set_title('Confusion Matrix')
ax.xaxis.set_ticklabels(labels)
ax.yaxis.set_ticklabels(labels)
```

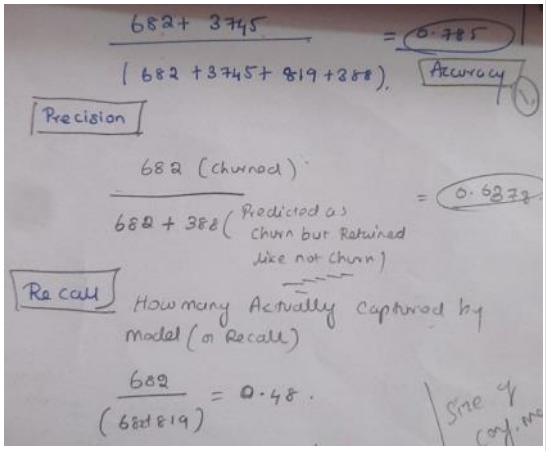
```
[[3745  388]
 [ 819  682]]
```




```
[[934 107]
 [212 156]]
```



Calculating the Metrics from Confusion matrix:



- Precision for a label is defined as the number of true positives divided by the number of predicted positives.
- Recall for a label is defined as the number of true positives divided by the total number of actual positives.

8. Implementing Cross-validated Logistic Regression

```
[ ] from sklearn.linear_model import LogisticRegressionCV
    from sklearn.model_selection import cross_validate
```

```
[ ] logistic = LogisticRegression()
```

```
scoring = ['accuracy']
scores = cross_validate(logistic,X_train, y_train, scoring = scoring, cv = 5, return_train_score=True,return_estimator=True,verbose = 10)
```

- Note, Since we have classified the CV = 5, hence we got the 5 folded Values for the Training accuracy, test accuracy, estimator.
- Estimator is nothing but the unknown Maximum likelihood parameters.

```
[ ] scores['train_accuracy']
```

```
array([0.78500111, 0.78677613, 0.78788551, 0.78877302, 0.78127773])
```

```
[ ] scores['test_accuracy']
```

```
array([0.78881988, 0.79148181, 0.77107365, 0.77462289, 0.80639432])
```

```
scores['estimator']
```

```
(LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False),
 LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False),
 LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
```

```
                    warm_start=False),
 LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
```


```
 LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False),
 LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                    intercept_scaling=1, l1_ratio=None, max_iter=100,
                    multi_class='auto', n_jobs=None, penalty='l2',
                    random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                    warm_start=False))
```


```
[ ] for model in scores['estimator']:
        print(model.coef_)
```

```
[[-0.05617762  0.03293792]]
[[-0.05562275  0.03215852]]
[[-0.05820295  0.03454813]]
[[-0.05711808  0.03362381]]
[[-0.05530045  0.03257423]]
```

Evaluation metrics for classification models

08 October 2022 13:20

Accuracy :

- Accuracy will require two inputs
  (i) actual class labels
  (ii) predicted class labels.
- To get the class labels from probabilities(these probabilities will be probabilities of getting a HIT), you can take a threshold of 0.5. Any probability above 0.5 will be labeled as class 1 and anything less than 0.5 will be labeled as class 0

Precision :

- Precision for a label is defined as the number of **true positives** divided by the number of **predicted positives**.
- Report precision in **percentages**.

Recall :

- Recall for a label is defined as the number of **true positives** divided by the total number of **actual positives**.
- Report recall in **percentages**.

F1-Score :

- This is defined as the harmonic mean of precision and recall.
- **2 x [(Precision x Recall) / (Precision + Recall)].**

Why is F1 score better than accuracy?

- Remember that the F1 score is balancing precision and recall on the positive class while accuracy looks at correctly classified observations both positive and negative.

Log Loss :

- Log-loss is indicative of how close the prediction probability is to the corresponding actual/true value
- This is defined as

$$LogLoss = \sum_{i=1}^{M} [y^i log(P^i) + (1 - y^i) log(1 - P^i)]$$

- Here M refers to the number of observations and yi = 1 or 0 depending upon the label for the ith observation and Pi is the probability of class 1 or probability of getting a HIT.

AUC-ROC :

- The Receiver Operator Characteristic (ROC) curve is an evaluation metric for **binary classification problems**.
- It is a **probability curve** that plots the **True Positive Rate against False Positive Rate** at various threshold values and essentially **separates the**

'signal' from the 'noise'.

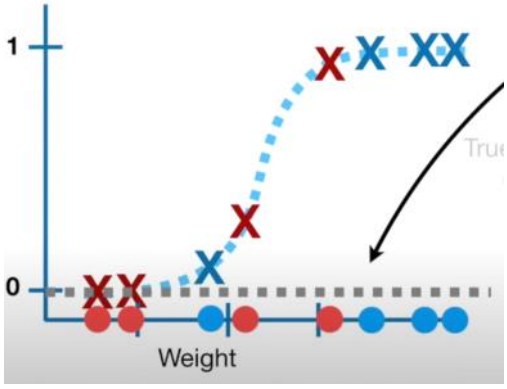
- The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

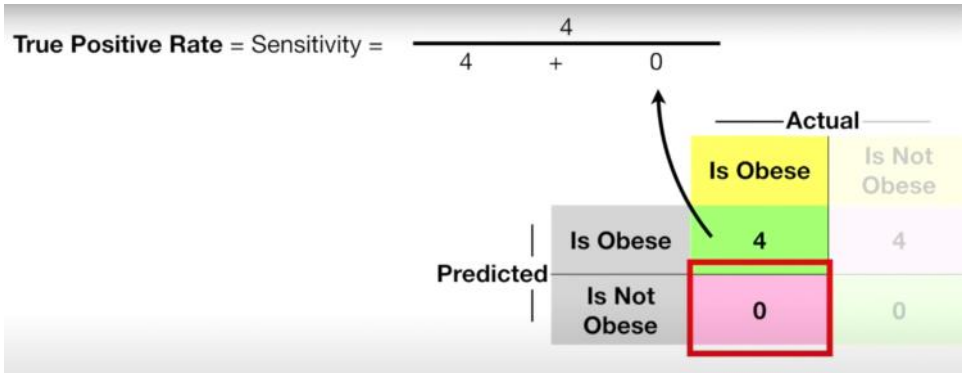
ROC: "Makes it easier to identify the best threshold(Generally we does with 0.5) for making the decision"

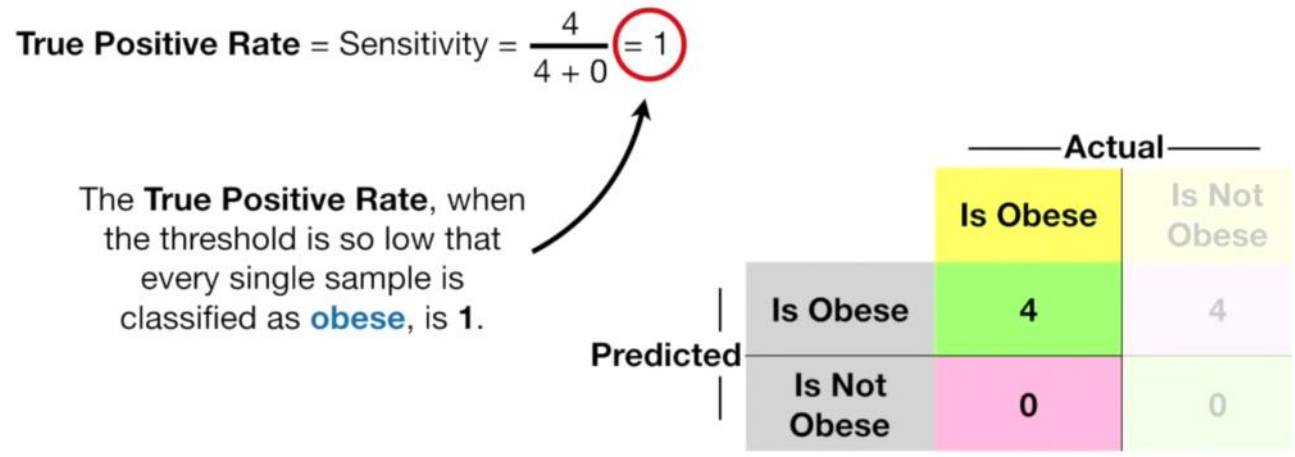
- What ROC does is, instead of overwhelmed with the confusion matrixes, ROC graph simplifies the all the information.
- $$\text{True Positive Rate} = \text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$
- False Negative Rate = 1 - Sensitivity
- True Positive rate tells us what proportion of the samples correctly classified.
- False Negative Rate tells us what proportion of the samples that are wrongly classified as correctly classified.

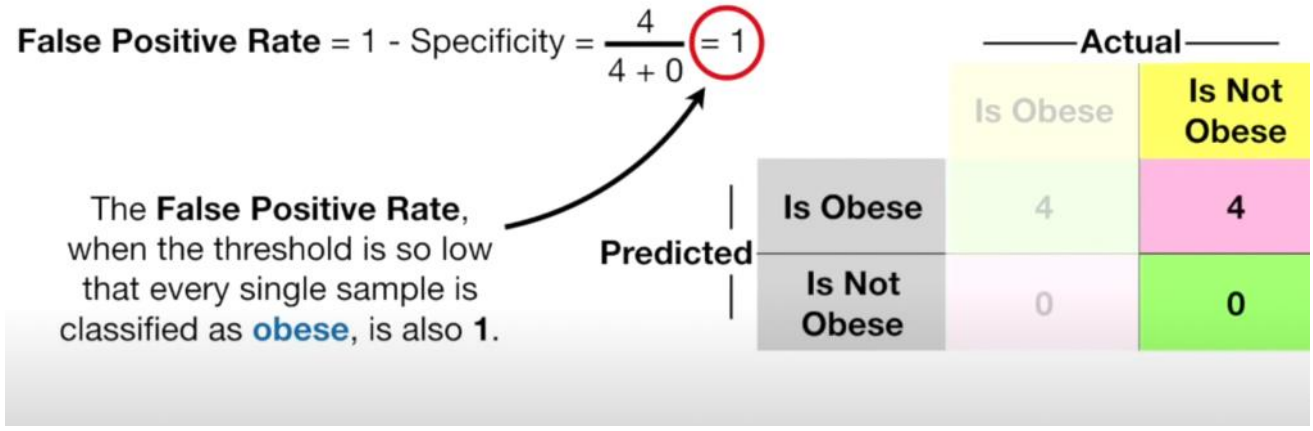
Working of ROC: (E.g. Provided weight is Obese or Not Obese)

1. Let we place the minimum threshold that classifies all the samples as obese.

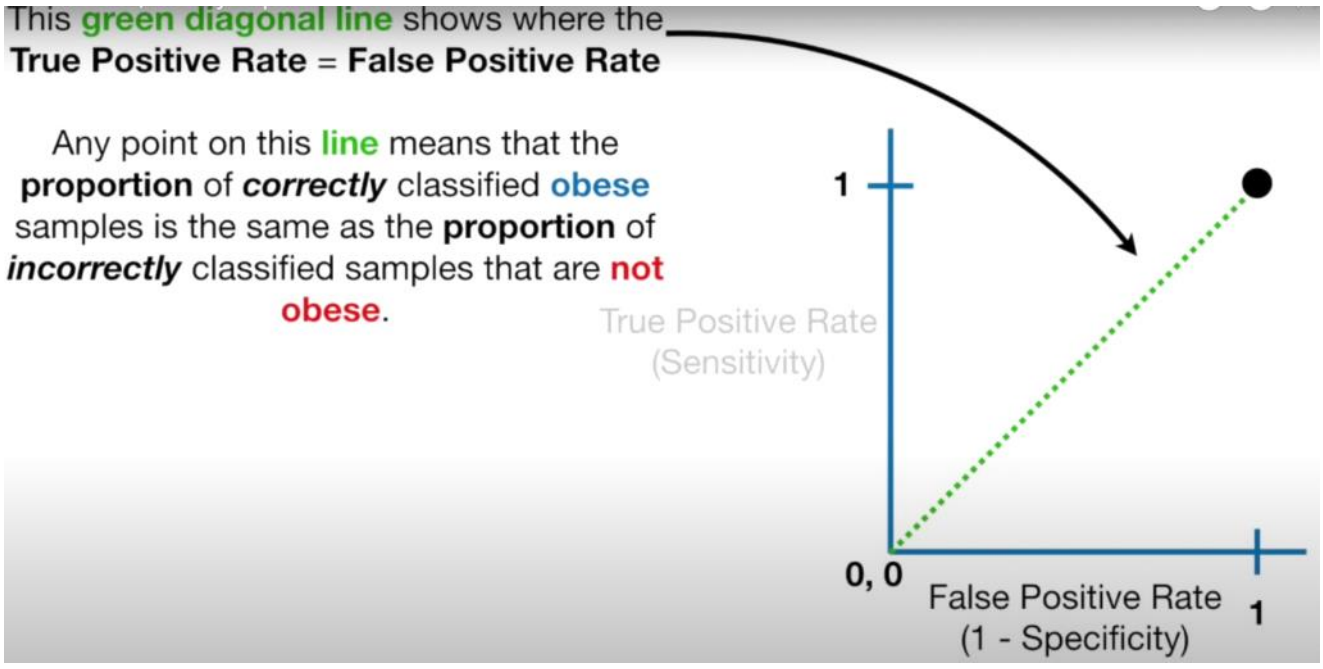




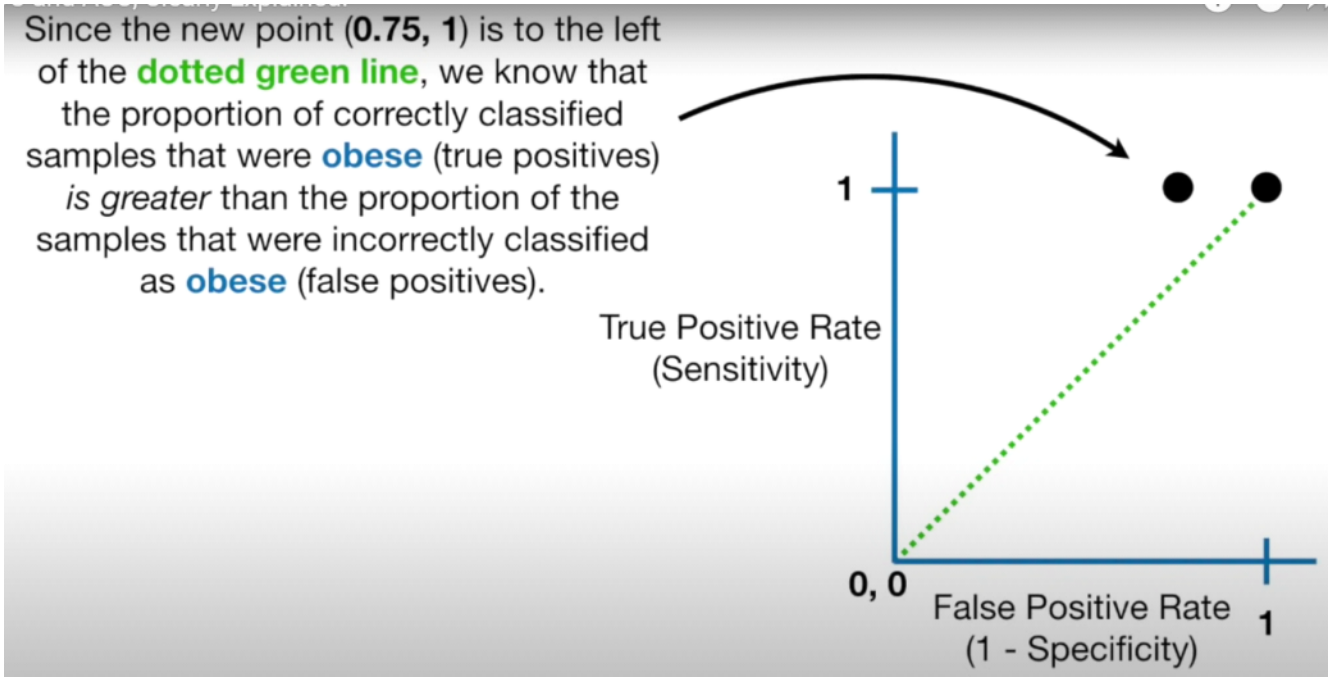




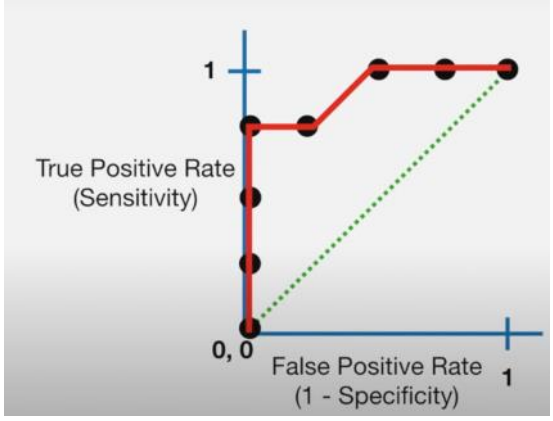
- The False Positive Rate = 1, shows that every single sample that are <u>not</u> obese is also calculated as Obese
- The True Positive Rate = 1, shows that every single sample that are obese is calculated as Obese



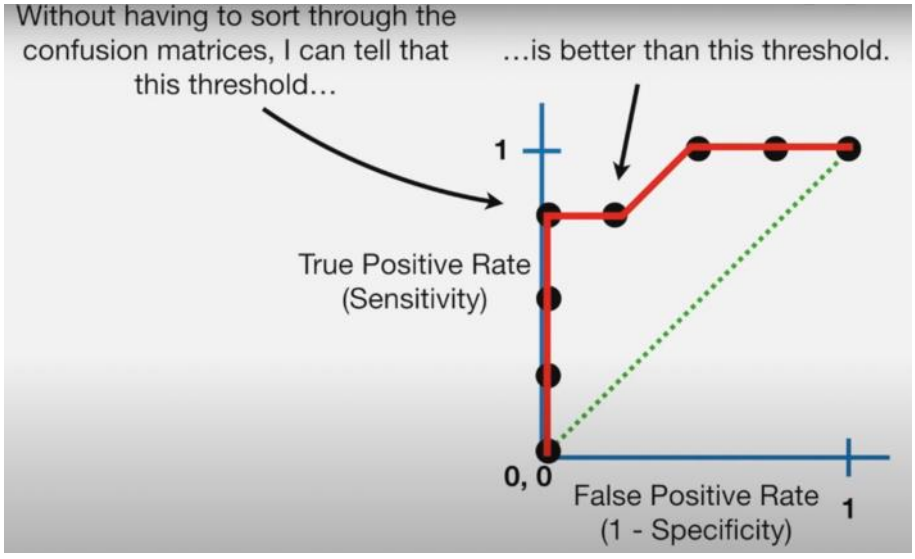
- As we change the thresold,



- After series of thresold values we get various points
- After connecting those lines, we get the ROC graph



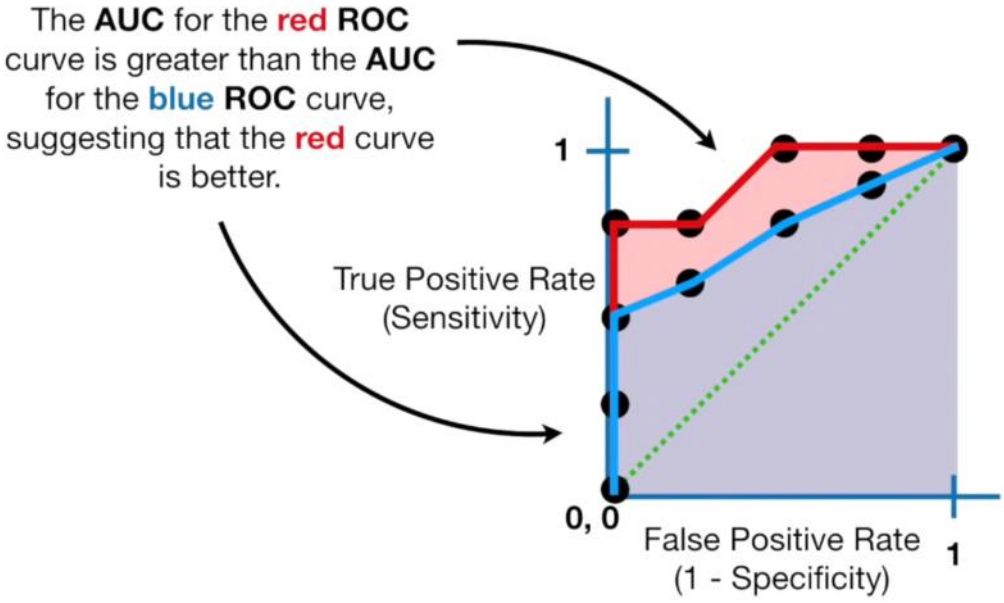
- ROC graph just summarizes all the confusion matrices that each thresold produces



- Depending on how many false positive iam willing to accept, thresold changes.

AUC:

- Let Red be the Logistic regression
- Blue be the random forest



- The AUC of Red > Blue
- Which means, we can use the Logistic regression for this dataset than getting better metrics than Random forest.

Sometimes False Positive Rate will be replaced by the Precision: why?

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} =$$ **Precision** is the proportion of positive results that were correctly classified

- If there are very small numbers of the non-obese with respect to the Obese in dataset
- Then, precision will be more useful metrics, as it not include the True negatives in its calculation.
- E.g.
  - If we study the cases of rare disease spread
  - Number of people that are affected will be less than that of number of people not affected
  - Here precision will be more helpful.

Note

09 October 2022 15:08

2 Standardisation of features is required before training a logistic regression.

Your Answer	False
Correct Answer	False

- As Standardisation of Features not required for the Logistic functions
- Logistic regressions and tree-based algorithms such as decision trees, random forests and gradient boosting are not sensitive to the magnitude of variables. So standardization is not needed before fitting these kinds of models.

Consider a following model for logistic regression: P (y =1|x, w)= g(w0 + w1x)

where g(z) is the logistic function.

In the above equation the P (y =1|x; w) , viewed as a function of x, that we can get by changing the parameters w.

10) What would be the range of p in such case?

A) (0, inf)

B) (-inf, 0)

C) (0, 1)

D) (-inf, inf)

Solution: C

For values of x in the range of real number from $-\infty$ to $+\infty$ Logistic function will give the output between (0,1)

7 Which of the following option is true?

Your Answer	Linear Regression errors values has to be normally distributed but in case of Logistic Regression it is not the case

What is errors value in linear regression and how it is normally distributed?