

LOGISTIC REGRESSION

Classification Model

► Basic Concept

- Logistic regression models the probability of a binary outcome (i.e., an outcome that has two possible values, often denoted as 0 and 1).
- **For example**, it might be used to predict whether an email is spam (1) or not spam (0).
- So it is popular for solving categorical problems.
- It can be used binary(0,1) or multiclass(like rating: 1,2,3,4,5) classification.
- It is a family of linear regression So it is applicable when **data is linearly separable** only.
- Like: **from sklearn.linear_model import LogisticRegression.**
- It is a generalized linear model. Where linear regression give you a continuous number and based on that number we have to generalized whether it is 0 or 1 based on threshold value.

LOGISTIC REGRESSION

- ▶ To solve Logistic Regression problems we can have three approach:
- ▶ **Geometrical Intuition**
- ▶ **Probabilistic approach**
- ▶ **Loss Function**

PROBLEM SOLVING APPROACH

1. Geometrical Intuition

- **Decision Boundary:** In logistic regression, we seek to find the best linear boundary (hyperplane) that separates the classes (e.g., survived vs. not survived) in the feature space. Unlike linear regression, which aims to fit a straight line, logistic regression aims to fit an S-shaped curve (sigmoid function) to the data.
- **Sigmoid Function:** This function maps any real-valued number into a value between 0 and 1, representing the probability of the target class. The logistic function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

where z is the linear combination of input features and weights.

2. Probabilistic Approach

- **Probability Estimation:** Logistic regression estimates the probability that a given input point belongs to a particular class. For a binary classification problem, the output is the probability of the positive class (e.g., survived).
- **Log-Odds:** The log-odds (logit) of the probability of the positive class is modeled as a linear combination of the input features. Mathematically, it can be expressed as:

$$\text{logit}(P) = \log\left(\frac{P}{1-P}\right) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n$$

where P is the probability of the positive class, and β s are the coefficients.

3. Loss Function

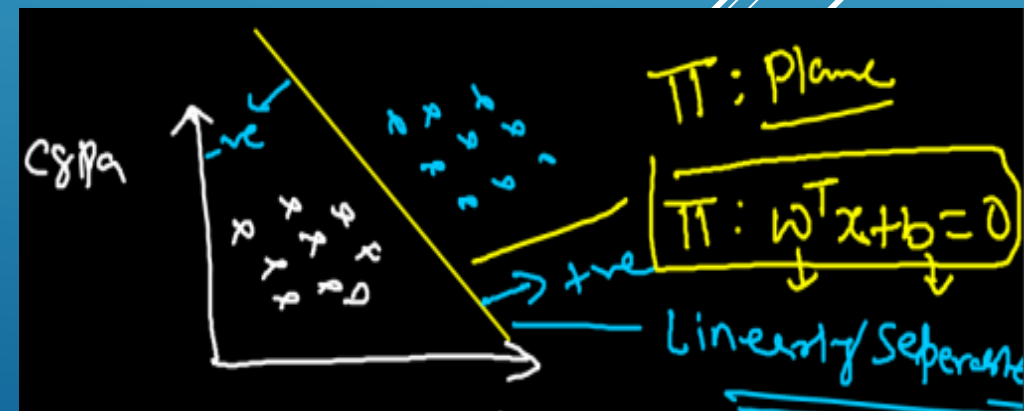
- **Cross-Entropy Loss:** The loss function used in logistic regression is the cross-entropy loss (also known as log loss). This measures the performance of a classification model whose output is a probability value between 0 and 1. The objective is to minimize this loss. For binary classification, it is defined as:

$$\text{Loss} = -\frac{1}{N} \sum_{i=1}^N [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)]$$

where y_i is the actual label, \hat{y}_i is the predicted probability, and N is the number of samples.

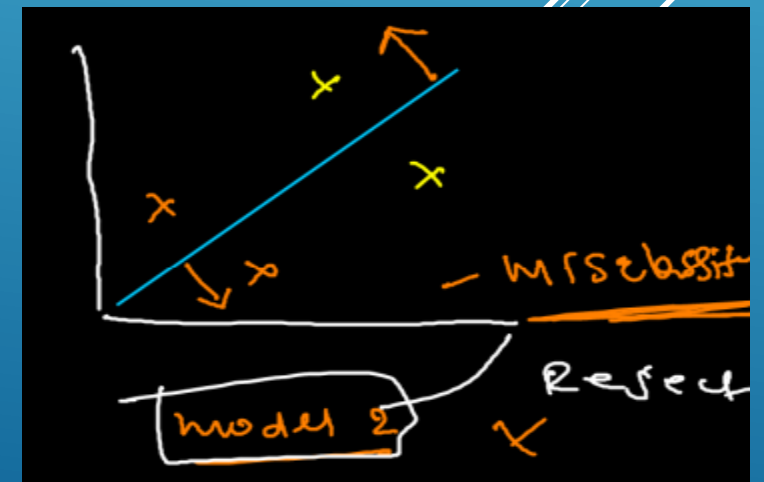
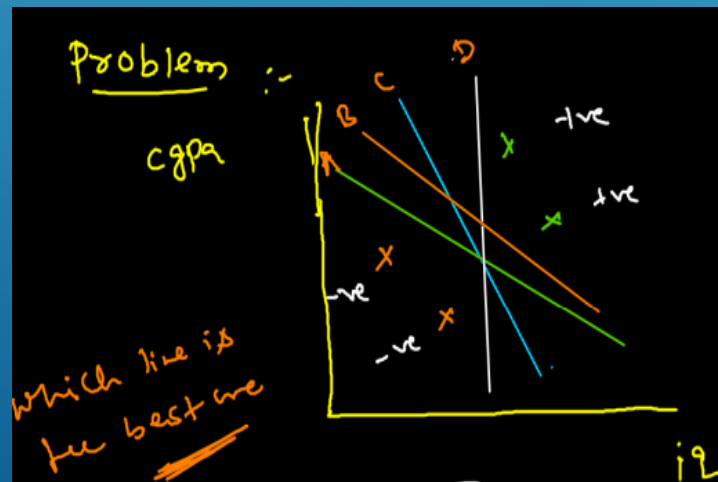
- ▶ $Y = mx + c$ eq of line
- ▶ $f(x) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 \dots \dots \dots + w_nx_n$
- ▶ $[w_0 \ w_1 \ w_2 \ w_3 \ w_4 \dots \dots \dots + w_n] [x_1x_2x_3x_4 \dots \dots \dots x_n]$
- ▶ $f(x) = [w_0 \ w_1 \ w_2 \ w_3 \ w_4 \dots \dots \dots + w_n]^T [x_1x_2x_3x_4 \dots \dots \dots x_n]$
- ▶ $f(x) = w^Tx$
- ▶ $w_0 + w^Tx \rightarrow$ if w_0 is passing through zero than
- ▶ case 1: $w^Tx > 0$ +ve class
- ▶ case 2: $w^Tx < 0$ -ve class
- ▶ If multiply y_i with w^Tx than $y_i * w^Tx$ varies(+1,-1)
- ▶ **Logistic regression aims to find a boundary** that separates the two classes (e.g., positive and negative). This boundary is a **line (in 2D) or a hyperplane (in higher dimensions)** that best **distinguishes between the two classes based on the features.**
- ▶ 2D = Line
- ▶ 3D = Plane
- ▶ nD = Hyperplane

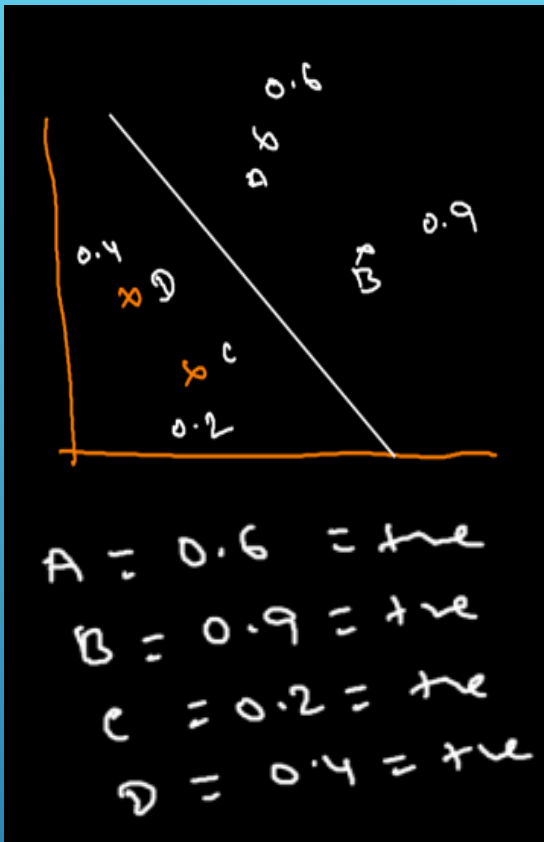
GEOMETRIC INTUITIONS



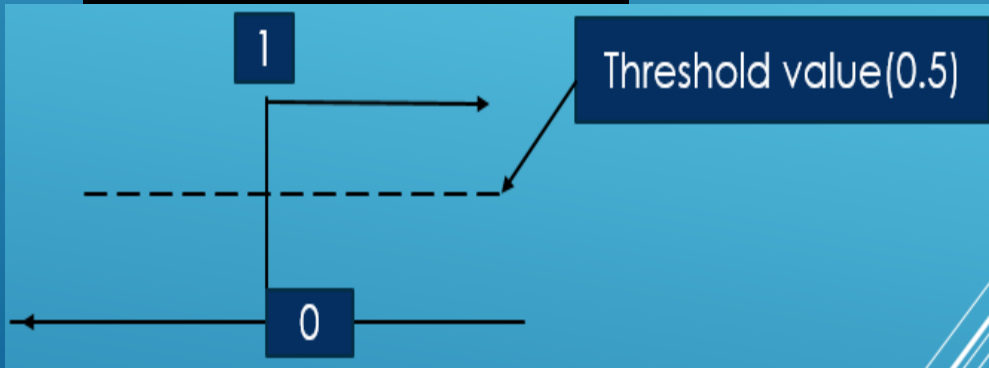
- ▶ Every line have +ve and -ve side. Than how to identify?
- ▶ $w^t x > 0$, $y_i > 0$ and $Y_i * w^t x > 0$ ----> correctly +ve classified
- ▶ $w^t x < 0$, $y_i < 0$ and $Y_i * w^t x < 0$ ----> correctly -ve classified
- ▶ $w^t x > 0$, $y_i < 0$ and $Y_i * w^t x < 0$ ----> Incorrect classified
- ▶ $w^t x < 0$, $y_i > 0$ and $Y_i * w^t x > 0$ ----> Incorrect classified
- ▶ $f(x) = y_i w^t x_i > 0$: correctly classified
- ▶ $f(x) = y_i w^t x_i < 0$: incorrectly classified
- ▶ **But here is the issue:**
- ▶ **How to find: which line is best line to separate linearly as in case of regression which line is best fit line.**
- ▶ **In classification problem we have to make sure that the positive point lands in the positive region also the negative point lands in the negative region.**

GEOMETRIC INTUITION





- To overcome this problem we are using sigmoid function which gives a threshold value that will do +ve or -ve classification.
- Sigmoid is a probability value. So I am saying what percentage of chances there you have to draw sigmoid value.
- If any value is above threshold value is +ve class.
- If any value is below threshold value is -ve class.
- Now which value is best:
- The higher probability value 'either closer to 1 or zero' will lead to best fit line.



x_1	x_2	y_1	y_2	m_1	m_2
-	-	0.4	0.6	1	1
-	-	0.3	0.4	0	0
-	-	0.9	0.7	1	1
-	-	0.4	0.2	0	0

$M1 = M2$
 But $Y1 > Y2$ for 1
 $Y2 > Y1$ for 0

SIGMOID FUNCTION

- ▶ The logistic regression model **calculates the probability** of a data **point belonging to the positive class** based on its **distance from the decision boundary**.
- ▶ **Sigmoid Function:** The logistic (sigmoid) function maps the linear combination of features (the distance from the decision boundary) to a probability between 0 and 1.
- ▶ **Distance Interpretation:** Points closer to the decision boundary have probabilities closer to 0.5, meaning they are less certain of their class. Points further away from the boundary will have probabilities closer to 0 or 1, indicating a higher confidence in their classification.

SIGMOID FUNCTION

► $p = \frac{1}{1+e^{-z}}$ where p = probability

► $z = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 \dots \dots \dots + w_nx_n$

► $p = \frac{1}{1+\frac{1}{e^z}}$ or $p = \frac{e^z}{e^z+1}$

► $p(e^z + 1) = e^z$

► $p + p(e^z) = e^z$

► $p = e^z (1 - p)$

► $\frac{p}{1-p} = e^z = \text{odd}$

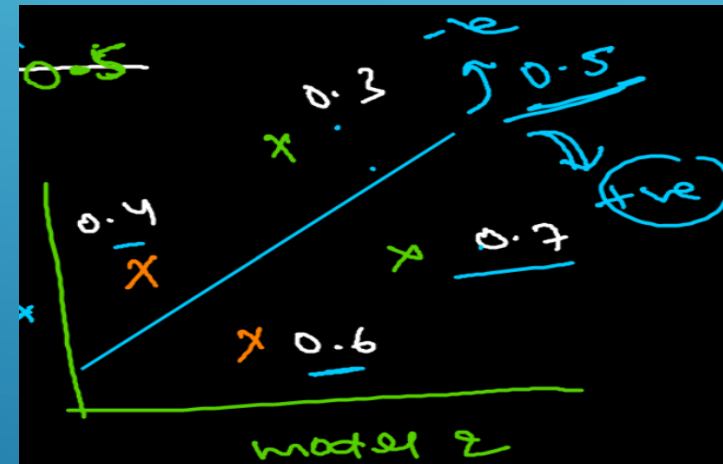
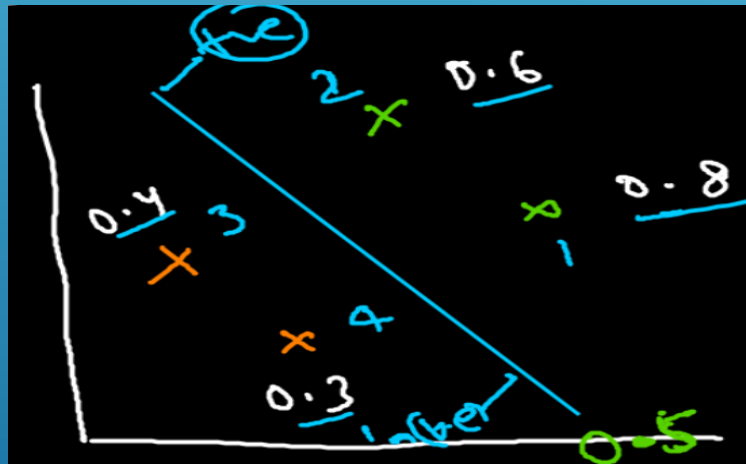
SIGMOID FUNCTION

- ▶ Now p can have only two values either 0 or 1
- ▶ **Case 1: when $p = 0$**
- ▶ $\frac{p}{1-p} = \frac{0}{1-0} = 0 \rightarrow e^z = 0$
- ▶ **Case 2: when $p = 1$**
- ▶ $\frac{p}{1-p} = \frac{1}{1-1} = \infty \rightarrow e^z = \infty$
- ▶ So in logistic regression it shows range is from **zero to infinity**.
- ▶ But in linear regression we have range from $-\infty$ to ∞ .
- ▶ So if there is negative value in logistic regression than to handle that we use log on both side.
- ▶ $\ln \frac{p}{1-p} = \ln e^z$ thus for case 1 : e^z will become $-\infty$.
- ▶ **Log-Odds and Decision Boundary:** The decision boundary can also be understood in terms of log-odds. Logistic regression models the log-odds (logit) of the probability of the positive class. The decision boundary is where the log-odds is zero (or where the probability is 0.5).

SIGMOID FUNCTION

- ▶ So now it can handle any negative or positive value using this log function that why it is also called logit regression as we use log function.
- ▶ And by using sigmoid function we get the probability value between zero and one.
- ▶ $\ln \frac{p}{1-p} = \ln e^z \rightarrow \ln \frac{p}{1-p} = z$
- ▶ Where $z = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 \dots \dots \dots + w_nx_n$
- ▶ $\ln \frac{p}{1-p} = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 \dots \dots \dots + w_nx_n$
- ▶ This is the **Logistic regression formula**:
- ▶ This equation is very similar to linear regression except that y value replaced with $\ln \frac{p}{1-p}$ means dependent variables changed.

- ▶ In linear regression we use OLS method to find the best fit line, Similarly in Logistic_Regression we use maximum Likelihood approach to find the best fit line.
- ▶ Where Likelihood function is the product of probability for the actual class of each observation.
- ▶ Let us have two models with given probability of each class.



MAXIMUM LIKELIHOOD

► For model 1 Likelihood for above the plane(+ve class): $\text{MLE (Model1)} = 0.8*0.6*0.6*0.7 = 0.20$

► For model 2 Likelihood for above the plane: $\text{MLE (Model2)} = 0.7*0.4*0.3*0.6 = 0.05$

► **For MLE:** Likelihood **model1** > Likelihood **model 2**

► So model1 is best.

► **So MLE will help to choose which line is best line.**

- ▶ **Logistic regression** uses the **Log loss**, also known as **logistic loss or binary cross-entropy loss**, is a **performance metric** for **evaluating the predictions** of a logistic regression model. It measures the accuracy of the model by comparing the predicted probabilities to the actual class labels.
- ▶ The **cost function** is defined as:
- ▶
$$\text{Log loss} = -\frac{1}{m} \sum_{i=1}^m [y_i \log(p(x_i)) + (1 - y_i) \log(1 - p(x_i))]$$
- ▶ m = number of training samples
- ▶ Y_i = the actual label of the i th sample
- ▶ $p(x_i)$ the predicted probability for the i th sample.
- ▶ We always look for minimum value.
- ▶ The log loss function **penalizes wrong predictions** more heavily, especially when the model is confident but wrong. A lower log loss value indicates better model performance.

LOG LOSS FUNCTION

$$J(\beta) = -\frac{1}{m} \sum_{i=1}^m [y_i \log(h_{\beta}(x_i)) + (1 - y_i) \log(1 - h_{\beta}(x_i))]$$

Here:

- m is the number of training samples.
- y_i is the actual label of the i -th sample.
- $h_{\beta}(x_i)$ is the predicted probability for the i -th sample.

Suppose we have a dataset with three samples and their corresponding actual labels and predicted probabilities:

Sample	Actual Label (y_i)	Predicted Probability (p_i)
1	1	0.9
2	0	0.2
3	1	0.4

Now, we sum these values and divide by the number of samples ($N = 3$):

$$\text{Log Loss} = -\frac{1}{3} (-0.105 - 0.223 - 0.916) = \frac{1}{3}(1.244) \approx 0.415$$

So, the log loss for this dataset is approximately 0.415.

EXAMPLE

Let's calculate the log loss for this dataset.

For each sample, we compute the term $y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$:

1. For Sample 1:

- $y_1 = 1$
- $p_1 = 0.9$
- $y_1 \log(p_1) + (1 - y_1) \log(1 - p_1) = 1 \cdot \log(0.9) + 0 \cdot \log(0.1) = \log(0.9) \approx -0.105$

2. For Sample 2:

- $y_2 = 0$
- $p_2 = 0.2$
- $y_2 \log(p_2) + (1 - y_2) \log(1 - p_2) = 0 \cdot \log(0.2) + 1 \cdot \log(0.8) = \log(0.8) \approx -0.223$

3. For Sample 3:

- $y_3 = 1$
- $p_3 = 0.4$
- $y_3 \log(p_3) + (1 - y_3) \log(1 - p_3) = 1 \cdot \log(0.4) + 0 \cdot \log(0.6) = \log(0.4) \approx -0.916$

1. **Range:** The log loss value can range from 0 to infinity. A log loss of 0 indicates perfect prediction, where the predicted probability exactly matches the actual label.
2. **Higher Values:** A higher log loss indicates poorer model performance, meaning the predicted probabilities are far from the actual labels.
3. **Penalty for Confidence:** The function heavily penalizes confident but incorrect predictions. For instance, predicting a probability of 0.9 for a class that is actually 0 results in a significant penalty.

Practical Use:

1. **Model Comparison:** When comparing different models, the one with the lower log loss is considered better. This is because it indicates the model's predicted probabilities are closer to the actual outcomes.
2. **Hyperparameter Tuning:** Log loss can guide hyperparameter tuning. Techniques like grid search or random search often use log loss to evaluate different model configurations and select the best one.
3. **Threshold Selection:** For binary classification, the threshold for classifying samples as positive or negative can be adjusted to minimize log loss. Typically, a threshold of 0.5 is used, but this can be fine-tuned based on the log loss value.

INTERPRETING LOG LOSS:

Returning to the example provided:

- **Calculated Log Loss:** The log loss was approximately 0.415.
- **Performance Insight:** This value suggests how well the model's predicted probabilities align with the actual outcomes. Lower values are better, indicating a model that predicts probabilities closer to the actual class labels.

Additional Considerations:

- **Context-Specific Tuning:** The acceptable log loss value may vary depending on the problem domain. For some applications, a log loss of 0.415 might be excellent, while in others, it could be too high.
- **Cross-Validation:** Using cross-validation with log loss helps ensure that the model's performance is consistent across different subsets of the data.

By continually monitoring and aiming to minimize log loss, you can enhance the accuracy and reliability of your logistic regression model's predictions.

EXAMPLE USAGE:

KEY POINTS TO UNDERSTAND LOG LOSS

1. Nature of Log Loss:

- **Logarithmic Scale:** Log loss uses a logarithmic scale, which means it increases steeply as the predicted probability diverges from the actual class label.
- **Penalty for Misconfidence:** It penalizes confident but wrong predictions more than unconfident wrong predictions.

2. Interpreting the Value:

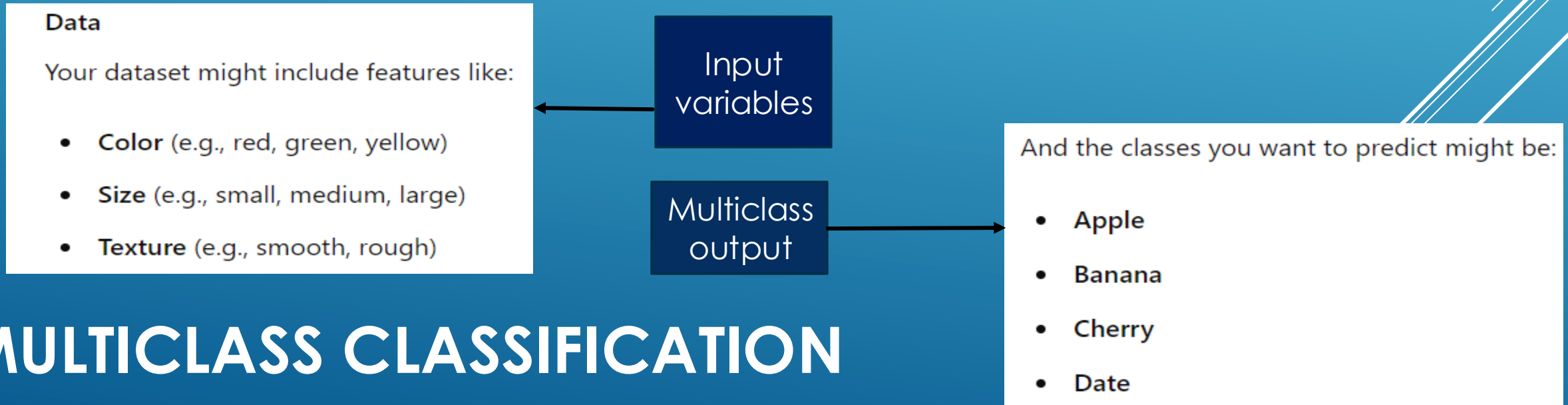
- **0.415 is not 40%:** The value 0.415 doesn't mean your predictions are 41.5% accurate or 58.5% wrong.
- **Performance Indicator:** It indicates the average penalty your model incurs for its predictions. Lower values mean better alignment with actual labels.

3. Comparison with Perfect and Worst Case:

- **Perfect Prediction:** A log loss of 0 means perfect prediction, where predicted probabilities are exactly equal to the actual class labels.
- **Random Guessing:** For binary classification with balanced classes, random guessing usually yields a log loss around 0.693. This is because $-\log(0.5) = 0.693$, which is the penalty for predicting 0.5 probability for either class.

- **Multiclass classification** is a type of classification problem where there are more than two classes or categories to predict. Unlike binary classification, which deals with two classes (e.g., spam vs. not spam), multiclass classification involves predicting one out of three or more possible classes.
- **Example:** Imagine you are building a **machine learning model to classify fruits** based on their **features (e.g., color, size, texture)**. Your dataset contains several types of fruits, and you want to **predict which type each fruit belongs to**.
- **Multiclass Classification Problem:**

In this problem, you have more than two classes (Apple, Banana, Cherry, Date), so it's a multiclass classification problem. The goal of your model is to assign a fruit to one of these classes based on its features.



- **Logistic regression** is originally a **binary classification** algorithm, but it can be extended to handle **multiclass classification** using different strategies. The two most common approaches are:

1. **One-vs-Rest (OvR) or One-vs-All (OvA):**

- **Concept:** For a multiclass problem with k classes, the OvR approach trains k separate binary classifiers. Each classifier is trained to distinguish one class from all the others.
- **How It Works:** For each classifier, logistic regression learns to predict whether an instance belongs to the target class or not. During prediction, each classifier gives a score, and the class with the highest score is chosen as the final prediction.

2. **Softmax Regression (Multinomial Logistic Regression):**

- **Concept:** This is a generalization of logistic regression for multiple classes. Instead of using multiple binary classifiers, it uses a single model that can predict multiple classes simultaneously.
- **How It Works:** The softmax function is used to convert the output scores (logits) into probabilities for each class. The class with the highest probability is chosen as the final prediction.

LOGISTIC REGRESSION AND MULTICLASS CLASSIFICATION

The softmax function converts the raw prediction scores into probabilities that sum to 1. For a given set of scores z_1, z_2, \dots, z_k for k classes, the softmax function is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^k e^{z_j}}$$

Where z_i is the score for class i , and e is the base of the natural logarithm.

By applying the softmax function, the model outputs a probability distribution over all classes, and the class with the highest probability is selected as the predicted class.

SOFTMAX FUNCTION

► Example Scenario

- You have a dataset where each student is described by their CGPA and IQ score, and the target variable is their placement decision. Here's a simplified dataset with example values:

Student	CGPA	IQ	Placement Status
A	9.0	130	Yes
B	8.5	125	No
C	7.0	110	Opted
D	8.0	120	Yes
E	6.5	105	No
F	7.5	115	Opted
G	9.2	135	Yes
H	8.7	128	No
I	7.3	112	Opted

Multiclass Classification

In this scenario, the task is to build a model that can predict whether a student will opt for placement based on their CGPA and IQ score.

EXAMPLE:

STEPS TO HANDLE MULTICLASS CLASSIFICATION

1. Data Preparation:

- **Features:** CGPA and IQ
- **Target Variable:** Placement Decision (with classes: Yes, No, Opted)

2. Choosing a Classification Approach:

- **One-vs-Rest (OvR):**
 - Train three binary classifiers: one for each class (Yes, No, Opted).
 - Each classifier will be trained to distinguish one class from the others.
- **Softmax Regression:**
 - Use a single model that outputs scores for all three classes.
 - The softmax function will convert these scores into probabilities for each class.

3. Training the Model:

- Using the training data, fit the model to learn the relationships between CGPA, IQ, and the placement decision.

4. Making Predictions:

- For a new student with CGPA 7.8 and IQ 125, the model will predict the placement decision based on learned patterns.

Example Prediction

Let's say your trained softmax regression model outputs the following probabilities for a new student:

- **Yes:** 0.7
- **No:** 0.2
- **Opted:** 0.1

Based on these probabilities, the model would predict that the student is most likely to opt for placement (**Yes**), as it has the highest probability.

In the OvR approach, we train three separate binary classifiers:

1. Yes vs. Not-Yes (No or Opted)
2. No vs. Not-No (Yes or Opted)
3. Opted vs. Not-Opted (Yes or No)

ONE-VS-REST (OVR)

1. Yes vs. Not-Yes

- Binary Labels:
 - Yes: 1
 - No, Opted: 0
- Training Data:

CGPA	IQ	Label
9.0	130	1
8.5	125	0
7.0	110	0
8.0	120	1
6.5	105	0
7.5	115	0
9.2	135	1
8.7	128	0
7.3	112	0

- Train Logistic Regression Model to get coefficients β_{Yes} .

2. No vs. Not-No

- Binary Labels:
 - No: 1
 - Yes, Opted: 0
- Training Data:

CGPA	IQ	Label
9.0	130	0
8.5	125	1
7.0	110	0
8.0	120	0
6.5	105	1
7.5	115	0
9.2	135	0
8.7	128	1
7.3	112	0



- Train Logistic Regression Model to get coefficients β_{No} .

3. Opted vs. Not-Opted

- Binary Labels:

- Opted: 1
- Yes, No: 0

- Training Data:

CGPA	IQ	Label
9.0	130	0
8.5	125	0
7.0	110	1
8.0	120	0
6.5	105	0
7.5	115	1
9.2	135	0
8.7	128	0
7.3	112	1

- Train Logistic Regression Model to get coefficients β_{Opted} .

Making Predictions

To predict for a new student with CGPA = 8.2 and IQ = 123:

1. Calculate the probability of being placed (Yes) using β_{Yes} .
2. Calculate the probability of not being placed (No) using β_{No} .
3. Calculate the probability of opting out (Opted) using β_{Opted} .

Choose the class with the highest probability.

1. Calculate the Probability for Each Class:

Assume the logistic regression models have provided us with the following coefficients (for simplicity, let's use hypothetical coefficients):

- $\beta_{Yes} = [\beta_{Yes,0}, \beta_{Yes,1}, \beta_{Yes,2}]$
- $\beta_{No} = [\beta_{No,0}, \beta_{No,1}, \beta_{No,2}]$
- $\beta_{Opted} = [\beta_{Opted,0}, \beta_{Opted,1}, \beta_{Opted,2}]$

Let's assume the coefficients are:

- $\beta_{Yes} = [-1, 0.5, 0.03]$
- $\beta_{No} = [-2, 0.4, 0.04]$
- $\beta_{Opted} = [-1.5, 0.3, 0.02]$

2. Calculate the Linear Scores for the new student with CGPA = 8.2 and IQ = 123:

$$z_{Yes} = \beta_{Yes,0} + \beta_{Yes,1} \cdot 8.2 + \beta_{Yes,2} \cdot 123$$

$$z_{No} = \beta_{No,0} + \beta_{No,1} \cdot 8.2 + \beta_{No,2} \cdot 123$$

$$z_{Opted} = \beta_{Opted,0} + \beta_{Opted,1} \cdot 8.2 + \beta_{Opted,2} \cdot 123$$

Plugging in the values:

$$z_{Yes} = -1 + 0.5 \cdot 8.2 + 0.03 \cdot 123 = -1 + 4.1 + 3.69 = 6.79$$

$$z_{No} = -2 + 0.4 \cdot 8.2 + 0.04 \cdot 123 = -2 + 3.28 + 4.92 = 6.2$$

$$z_{Opted} = -1.5 + 0.3 \cdot 8.2 + 0.02 \cdot 123 = -1.5 + 2.46 + 2.46 = 3.42$$

3. Calculate the Probabilities using the sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}}$:

$$P(Yes) = \sigma(z_{Yes}) = \frac{1}{1 + e^{-6.79}}$$

$$P(No) = \sigma(z_{No}) = \frac{1}{1 + e^{-6.2}}$$

$$P(Opted) = \sigma(z_{Opted}) = \frac{1}{1 + e^{-3.42}}$$

Computing these values:

$$P(Yes) = \frac{1}{1 + e^{-6.79}} \approx 0.9989$$

$$P(No) = \frac{1}{1 + e^{-6.2}} \approx 0.9979$$

$$P(Opted) = \frac{1}{1 + e^{-3.42}} \approx 0.9683$$

4. Choose the Class with the Highest Probability:

- Yes: 0.9989
- No: 0.9979
- Opted: 0.9683

The highest probability is for "Yes," so the predicted placement status is "Yes."

SOFTMAX REGRESSION APPROACH

Plugging in the values:

$$e^{z_{Yes}} \approx e^{6.79} \approx 884.7$$

$$e^{z_{No}} \approx e^{6.2} \approx 492.7$$

$$e^{z_{Opted}} \approx e^{3.42} \approx 30.6$$

1. Calculate the Linear Scores for each class:

Using the same hypothetical coefficients and scores computed earlier:

$$z_{Yes} = 6.79$$

$$z_{No} = 6.2$$

$$z_{Opted} = 3.42$$

2. Compute the Softmax Probabilities:

$$P(Yes) = \frac{e^{z_{Yes}}}{e^{z_{Yes}} + e^{z_{No}} + e^{z_{Opted}}}$$

$$P(No) = \frac{e^{z_{No}}}{e^{z_{Yes}} + e^{z_{No}} + e^{z_{Opted}}}$$

$$P(Opted) = \frac{e^{z_{Opted}}}{e^{z_{Yes}} + e^{z_{No}} + e^{z_{Opted}}}$$

Sum of exponentials = $884.7 + 492.7 + 30.6 = 1408$

$$P(Yes) = \frac{884.7}{1408} \approx 0.628$$

$$P(No) = \frac{492.7}{1408} \approx 0.350$$

$$P(Opted) = \frac{30.6}{1408} \approx 0.022$$

3. Choose the Class with the Highest Probability:

- Yes: 0.628
- No: 0.350
- Opted: 0.022

The highest probability is for "Yes," so the predicted placement status is "Yes."

Conclusion

- Both approaches (OvR and Softmax) predict that the new student with CGPA = 8.2 and IQ = 123 is likely to be placed ("Yes").

- CRITERION TO MEASURE PERFORMANCE ARE:
- ACCURACY
- PRECISION
- RECALL
- F SCORE ETC
- BUT UNDERSTAND THESE TERM FOLLOWING TERM MUST BE UNDERSTAND


KEY EVALUATION METRICS

Definitions

- **True Positive (TP):** The number of positive instances correctly predicted by the model. These are the cases where both the actual and predicted classes are positive.
- **False Negative (FN):** The number of positive instances incorrectly predicted as negative. These are the actual positive cases that the model failed to identify.
- **False Positive (FP):** The number of negative instances incorrectly predicted as positive. These are the cases where the model mistakenly identified negatives as positives.
- **True Negative (TN):** The number of negative instances correctly predicted by the model. These are the cases where both the actual and predicted classes are negative.

The confusion matrix for your training data is:

lua

 Copy code

```
Training_Confusion_Matrix: [[379  60]
                             [ 78 195]]
```

This can be broken down as follows:

- True Positives (TP): 195
- True Negatives (TN): 379
- False Positives (FP): 60
- False Negatives (FN): 78

KEY EVALUATION METRICS

➤ IF DATA IS IMBALANCED
THAN DON'T DEPEND ON
ACCURACY ONLY.

1. **Accuracy:** The ratio of correctly predicted instances to the total instances. It's a simple measure but can be misleading if the data is imbalanced.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

2. **Precision:** The ratio of true positive predictions to the total positive predictions. It indicates how many of the predicted positive instances are actually positive.

$$\text{Precision} = \frac{TP}{TP + FP}$$

3. **Recall (Sensitivity or True Positive Rate):** The ratio of true positive predictions to the actual positive instances. It shows how well the model captures positive instances.

$$\text{Recall} = \frac{TP}{TP + FN}$$

4. **F1 Score:** The harmonic mean of precision and recall. It provides a single metric that balances both precision and recall.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

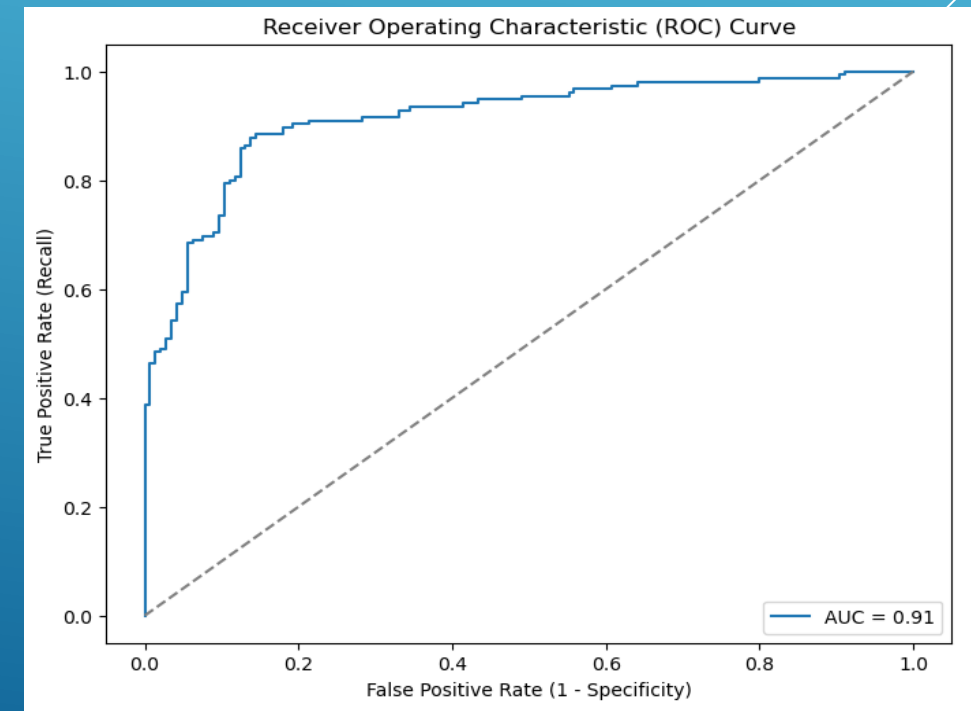
5. **Specificity (True Negative Rate):** The ratio of true negative predictions to the actual negative instances. It indicates how well the model identifies negative instances.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

6. **Area Under the ROC Curve (AUC-ROC):** The ROC curve plots the true positive rate (recall) against the false positive rate (1-specificity). The AUC-ROC score represents the likelihood that the model will rank a randomly chosen positive instance higher than a randomly chosen negative instance.

- ▶ The **Area Under the Receiver Operating Characteristic (AUC-ROC)** curve is a performance metric for **binary classification** problems.
- ▶ It provides a single measure of a model's ability to distinguish between classes.
- ▶ The ROC curve is a graphical representation of the trade-off between the true positive rate (recall) and the false positive rate (1-specificity).
- ▶ The AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes.
- ▶ Ranges from 0 to 1.

- **AUC = 0.5**: Model has no discrimination capability (equivalent to random guessing).
- **$0.5 < \text{AUC} < 0.7$** : Model has a poor discrimination capability.
- **$0.7 < \text{AUC} < 0.8$** : Model has an acceptable discrimination capability.
- **$0.8 < \text{AUC} < 0.9$** : Model has an excellent discrimination capability.
- **$\text{AUC} > 0.9$** : Model has an outstanding discrimination capability.



Confusion Matrix: A table that provides a detailed breakdown of the true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). It helps in understanding the types of errors the model is making.

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

CONFUSION MATRIX

		Predicted condition	
		Positive (PP)	Negative (PN)
Actual condition	Total population = P + N		
	Positive (P)	True positive (TP)	False negative (FN)
	Negative (N)	False positive (FP)	True negative (TN)

Suppose you have a confusion matrix like this:

	Predicted Positive	Predicted Negative
Actual Positive	50	10
Actual Negative	5	35

- True Positives (TP) = 50
- False Negatives (FN) = 10
- False Positives (FP) = 5
- True Negatives (TN) = 35

From this matrix:

- Accuracy: $\frac{50+35}{50+10+5+35} = \frac{85}{100} = 0.85$ or 85%
- Precision: $\frac{50}{50+5} = \frac{50}{55} \approx 0.91$ or 91%
- Recall: $\frac{50}{50+10} = \frac{50}{60} \approx 0.83$ or 83%
- Specificity: $\frac{35}{35+5} = \frac{35}{40} = 0.875$ or 87.5%
- F1 Score: $2 \times \frac{0.91 \times 0.83}{0.91 + 0.83} \approx 0.87$ or 87%

INTERPRETATION EXAMPLE

Precision is particularly important when the cost of false positives is high. A high precision model ensures that when the model predicts a positive class, it is very likely to be correct. Situations where precision is critical include:

1. **Spam Detection:** You want to minimize the number of legitimate emails classified as spam. High precision ensures that most emails marked as spam are truly spam, reducing the chances of important emails being lost.
2. **Fraud Detection:** In financial transactions, a false positive (flagging a legitimate transaction as fraud) can inconvenience customers and damage the business's reputation. High precision ensures that flagged transactions are more likely to be actual fraud.
3. **Medical Diagnostics for Non-critical Conditions:** When diagnosing conditions that are not life-threatening but could cause unnecessary anxiety or treatments (e.g., predicting allergies), high precision ensures that those identified as positive are very likely to have the condition.

WHEN TO PRIORITIZE PRECISION

Recall is particularly important when the cost of false negatives is high. A high recall model ensures that most actual positive cases are identified. Situations where recall is critical include:

1. **Medical Diagnostics for Critical Conditions:** In life-threatening conditions like cancer, it's crucial to identify as many true cases as possible. Missing a positive case (false negative) can have severe consequences. High recall ensures that most actual cases are detected.
2. **Security and Safety Systems:** In systems like smoke detectors or intrusion detection systems, it's important to catch all possible threats or incidents. Missing a real threat (false negative) can lead to serious harm or breaches.
3. **Disease Outbreak Detection:** In epidemiology, identifying all possible cases of a contagious disease is crucial to prevent an outbreak. High recall ensures that most cases are caught and contained.

WHEN TO PRIORITIZE RECALL

In many situations, there is a trade-off between precision and recall. Balancing the two can be achieved using the F1 Score, which is the harmonic mean of precision and recall. This is useful when you need a balance between precision and recall, and neither false positives nor false negatives can be disproportionately high.

Example:

1. **Information Retrieval Systems:** Search engines need to balance precision and recall to provide relevant results without missing important information. The F1 score helps in optimizing the balance.
2. **Recommendation Systems:** When recommending products or content, it's important to suggest relevant items (precision) while also ensuring that users don't miss out on potentially interesting items (recall).

BALANCING PRECISION AND RECALL

Summary

- **Prioritize Precision:** When false positives are more costly or problematic (e.g., spam detection, fraud detection).
- **Prioritize Recall:** When false negatives are more costly or problematic (e.g., critical medical diagnostics, security systems).
- **Balance Precision and Recall:** When both types of errors need to be minimized and a balance is crucial (e.g., information retrieval, recommendation systems).

Choosing the right metric depends on the specific application and the relative costs of false positives and false negatives.

► Problem Statement: Predicting Passenger Survival on the Titanic

- **Objective:** To develop a logistic regression model to predict whether a passenger survived the Titanic disaster based on various features available in the dataset.

Goal: To identify the key factors that significantly influence survival, and to create a model that can accurately predict the survival outcome of new passengers based on these features.

Approach:

1. **Data Cleaning:** Handle missing values, encode categorical variables, and normalize or scale numerical features if necessary.
2. **Feature Selection:** Evaluate the importance of each feature in predicting survival using statistical tests or feature selection techniques.
3. **Model Training:** Build a logistic regression model using the selected features.
4. **Model Evaluation:** Assess the model's performance using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

Variables:

1. **Survived** (Target Variable): Binary indicator (0 = No, 1 = Yes) indicating whether the passenger survived or not.
2. **Pclass:** Passenger class (1 = 1st, 2 = 2nd, 3 = 3rd).
3. **Name:** Name of the passenger.
4. **Sex:** Gender of the passenger (male or female).
5. **Age:** Age of the passenger.
6. **SibSp:** Number of siblings or spouses aboard the Titanic.
7. **Parch:** Number of parents or children aboard the Titanic.
8. **Ticket:** Ticket number.
9. **Fare:** Fare paid by the passenger.
10. **Cabin:** Cabin number.
11. **Embarked:** Port of Embarkation (C = Cherbourg, Q = Queenstown, S = Southampton).

CASE STUDY:

▼ Logistic Regression : Binary Classification

```
[1]: 1 # Titanic data set
      2 import os
      3 import warnings
      4 warnings.filterwarnings('ignore')
      5
      6 import numpy as np
      7 import pandas as pd
      8 import matplotlib.pyplot as plt
      9 import seaborn as sns
```



```
[2]: 1 df = pd.read_csv(r"D:\desktop\ML\DATA\titanic_train.csv")
```

```
[3]: 1 df.head()
```

```
[3]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

[4]: 1 df.info()

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 12 columns):  
#   Column      Non-Null Count  Dtype  
---  -  
0   PassengerId  891 non-null    int64  
1   Survived     891 non-null    int64  
2   Pclass       891 non-null    int64  
3   Name         891 non-null    object  
4   Sex          891 non-null    object  
5   Age          714 non-null    float64  
6   SibSp        891 non-null    int64  
7   Parch        891 non-null    int64  
8   Ticket       891 non-null    object  
9   Fare         891 non-null    float64  
10  Cabin        204 non-null    object  
11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)  
memory usage: 83.7+ KB
```

```
[5]: 1 df.isnull().sum()/len(df)
```

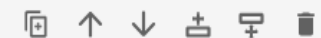
```
[5]: PassengerId    0.000000
     Survived      0.000000
     Pclass       0.000000
     Name         0.000000
     Sex          0.000000
     Age         0.198653
     SibSp        0.000000
     Parch        0.000000
     Ticket       0.000000
     Fare         0.000000
     Cabin        0.771044
     Embarked     0.002245
     dtype: float64
```

```
[6]: 1 df.columns
```

```
[6]: Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
         'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
        dtype='object')
```

- Since cabin have 77% data , we can drop it along with 'PassengerId','Name','Ticket', 'Fare' as they are not relevant for survival

```
[7]: 1 df = df.drop(['PassengerId', 'Name', 'Ticket', 'Fare', 'Cabin'],axis = 1)
     2
```



```
[8]: 1 df.dtypes
```

```
[8]: Survived    int64  
     Pclass     int64  
     Sex       object  
     Age       float64  
     SibSp     int64  
     Parch     int64  
     Embarked  object  
     dtype: object
```

```
[9]: 1 df.head()
```

```
[9]:
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Embarked
0	0	3	male	22.0	1	0	S
1	1	1	female	38.0	1	0	C
2	1	3	female	26.0	0	0	S
3	1	1	female	35.0	1	0	S
4	0	3	male	35.0	0	0	S

```
[11]: 1 df.isnull().sum()/len(df)*100
```

```
[11]: Survived      0.000000  
Pclass         0.000000  
Sex            0.000000  
Age           19.865320  
SibSp          0.000000  
Parch          0.000000  
Embarked       0.224467  
dtype: float64
```

- Age and Embarked has missing value: where Age is numerical data and Embarked is character type data.

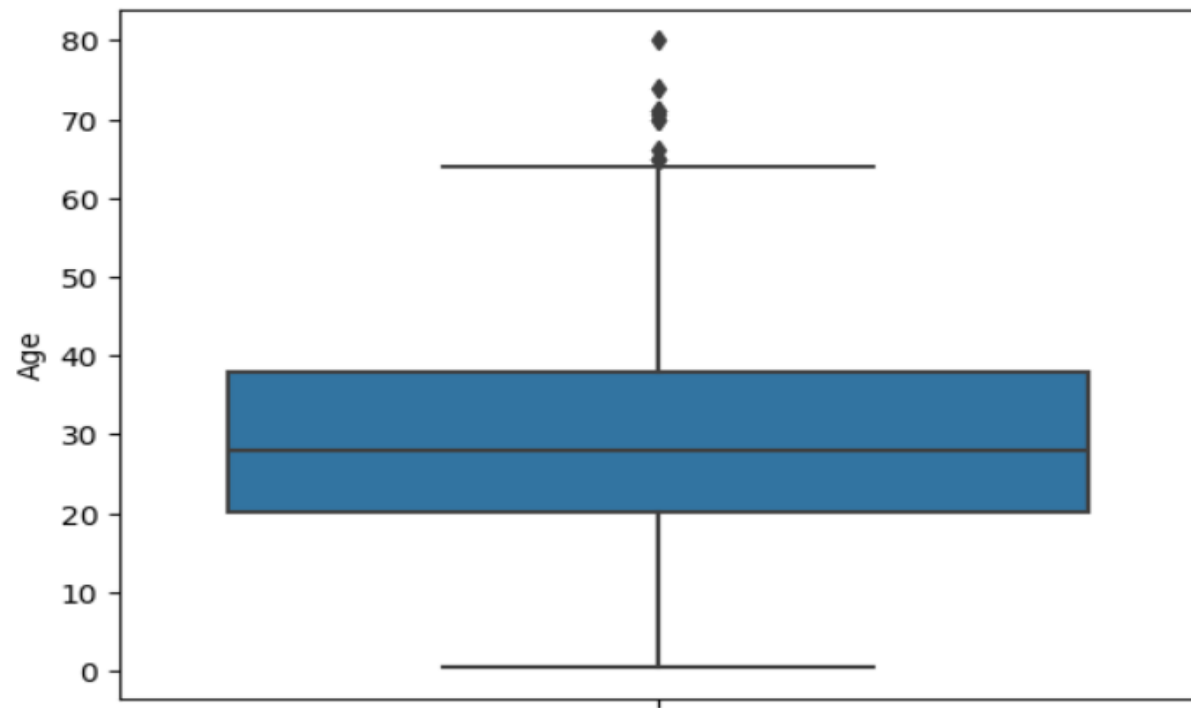

```
[13]: 1 df['Age'].describe()
```

```
[13]: count    714.000000  
      mean     29.699118  
      std     14.526497  
      min      0.420000  
      25%     20.125000  
      50%     28.000000  
      75%     38.000000  
      max     80.000000  
      Name: Age, dtype: float64
```

- ▼ • In case of Age Max = 80 (means old age people are possible, also Min = 0.42(means infant also possible)
- So We are assuming there is no outlier even the plot may show. ¶
- therefore for missing handling missing value we can go with median or mean.

```
[14]: 1 sns.boxplot(y = 'Age', data = df)
```

```
[14]: <Axes: ylabel='Age'>
```



```
[15]: 1 df['Age'] = df['Age'].fillna(df['Age'].mean())
```

```
[16]: 1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 7 columns):  
#   Column      Non-Null Count  Dtype  
---  -  
0   Survived    891 non-null   int64  
1   Pclass      891 non-null   int64  
2   Sex         891 non-null   object  
3   Age         891 non-null   float64  
4   SibSp       891 non-null   int64  
5   Parch       891 non-null   int64  
6   Embarked    889 non-null   object  
dtypes: float64(1), int64(4), object(2)  
memory usage: 48.9+ KB
```

```
[17]: 1 df[['Embarked']].value_counts()
```

```
[17]: Embarked  
S          644  
C          168  
Q           77  
Name: count, dtype: int64
```

```
[18]: 1 df['Embarked'] = df['Embarked'].fillna('S')
```

```
[19]: 1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 891 entries, 0 to 890  
Data columns (total 7 columns):  
#   Column      Non-Null Count  Dtype  
---  ---  
0   Survived    891 non-null    int64  
1   Pclass      891 non-null    int64  
2   Sex         891 non-null    object  
3   Age         891 non-null    float64  
4   SibSp       891 non-null    int64  
5   Parch       891 non-null    int64  
6   Embarked    891 non-null    object  
dtypes: float64(1), int64(4), object(2)  
memory usage: 48.9+ KB
```

- **Encoding: variables have data type object like 'Sex' and 'Embarked'**

- **One-Hot Encoding**

- What it does: Converts categorical variables into a series of binary columns (e.g., "male" -> [1, 0], "female" -> [0, 1]).
- When to use: This is typically the better approach for most machine learning algorithms, especially linear models and neural networks, as it avoids implying any ordinal relationship.
- Example algorithms: Linear Regression, Logistic Regression, Neural Networks, SVMs.
- To use One Hot Encoding on the Sex column in your DataFrame, you can use the OneHotEncoder from sklearn.preprocessing
- or the get_dummies function from pandas.

- **Label Encoding**

- What it does: Converts categorical values to numeric labels (e.g., "male" -> 0, "female" -> 1).
- When to use: This is suitable for algorithms that can handle ordinal relationships between categories. However, it's generally not recommended for categorical variables without a meaningful order, as it can introduce unintended ordinal relationships.
- Example algorithms: Tree-based methods like Decision Trees, Random Forests, Gradient Boosting Machines can handle label encoded variables well.
-

```
[20]: 1 df['Sex'].value_counts()
```

```
[20]: Sex
male      577
female    314
Name: count, dtype: int64
```

- We can do label encoding as there only two possibility. However OHE is best practice when there no ordinal issue.

```
[21]: 1 # Label encoder
      2 # df['Sex'] = df['Sex'].astype('category')
      3 # df['Sex'] = df['Sex'].cat.codes
```

```
from sklearn.preprocessing import OneHotEncoder
```

- **Initialize the OneHotEncoder**

```
encoder = OneHotEncoder()
```

- **Fit and transform the data**

```
one_hot_encoded = encoder.fit_transform(df[['Sex']]).toarray()
```

- **Create a DataFrame with the one hot encoded columns**

```
one_hot_df = pd.DataFrame(one_hot_encoded, columns=encoder.get_feature_names_out(['Sex']))
```

- **Concatenate the original DataFrame with the one hot encoded DataFrame**

```
df = pd.concat([df, one_hot_df], axis=1)
```

```
df.head()
```

```
[22]: 1 df = pd.get_dummies(df, columns = ['Sex'])  
      2 df.head()
```

```
[22]:
```

	Survived	Pclass	Age	SibSp	Parch	Embarked	Sex_female	Sex_male
0	0	3	22.0	1	0	S	False	True
1	1	1	38.0	1	0	C	True	False
2	1	3	26.0	0	0	S	True	False
3	1	1	35.0	1	0	S	True	False
4	0	3	35.0	0	0	S	False	True

```
[23]: 1 df = pd.get_dummies(df, columns = ['Embarked'])  
      2 df.head()
```

```
[23]:
```

	Survived	Pclass	Age	SibSp	Parch	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S
0	0	3	22.0	1	0	False	True	False	False	True
1	1	1	38.0	1	0	True	False	True	False	False
2	1	3	26.0	0	0	True	False	False	False	True
3	1	1	35.0	1	0	True	False	False	False	True
4	0	3	35.0	0	0	False	True	False	False	True


```
[24]: 1 df['Pclass'].value_counts()
```

```
[24]: Pclass
3     491
1     216
2     184
Name: count, dtype: int64
```

```
[25]: 1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 10 columns):
#   Column          Non-Null Count  Dtype
---  -
0   Survived        891 non-null    int64
1   Pclass          891 non-null    int64
2   Age            891 non-null    float64
3   SibSp          891 non-null    int64
4   Parch          891 non-null    int64
5   Sex_female      891 non-null    bool
6   Sex_male        891 non-null    bool
7   Embarked_C      891 non-null    bool
8   Embarked_Q      891 non-null    bool
9   Embarked_S      891 non-null    bool
dtypes: bool(5), float64(1), int64(4)
memory usage: 39.3 KB
```

```
[26]: 1 df = pd.get_dummies(df, columns = ['Pclass'])
      2
      3 df.head()
```

```
[26]:
```

	Survived	Age	SibSp	Parch	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S	Pclass_1	Pclass_2	Pclass_3
0	0	22.0	1	0	False	True	False	False	True	False	False	True
1	1	38.0	1	0	True	False	True	False	False	True	False	False
2	1	26.0	0	0	True	False	False	False	True	False	False	True
3	1	35.0	1	0	True	False	False	False	True	True	False	False
4	0	35.0	0	0	False	True	False	False	True	False	False	True

```
[27]: 1 df = df.astype(int)
```

```
[28]: 1 df.head()
```

```
[28]:
```

	Survived	Age	SibSp	Parch	Sex_female	Sex_male	Embarked_C	Embarked_Q	Embarked_S	Pclass_1	Pclass_2	Pclass_3
0	0	22	1	0	0	1	0	0	1	0	0	1
1	1	38	1	0	1	0	1	0	0	1	0	0
2	1	26	0	0	1	0	0	0	1	0	0	1
3	1	35	1	0	1	0	0	0	1	1	0	0
4	0	35	0	0	0	1	0	0	1	0	0	1

- dummy variables: their presence or absence does not affect the dependent variable so we can drop it.
- 'Sex_male', 'Embarked_S', 'Pclass_3'

```
[29]: 1  
2 df = df.drop(['Sex_male', 'Embarked_S', 'Pclass_3'], axis = 1)
```



```
[30]: 1 df.head()
```

```
[30]:
```

	Survived	Age	SibSp	Parch	Sex_female	Embarked_C	Embarked_Q	Pclass_1	Pclass_2
0	0	22	1	0	0	0	0	0	0
1	1	38	1	0	1	1	0	1	0
2	1	26	0	0	1	0	0	0	0
3	1	35	1	0	1	0	0	1	0
4	0	35	0	0	0	0	0	0	0

- In classification you no need to treat outliers
- In classification problem you to check whether data is balanced or not.
- If not balanced than balance it ¶
- balancing is done only for dependent variable.

```
[31]: 1 df['Survived'].value_counts()
```

```
[31]: Survived  
0     549  
1     342  
Name: count, dtype: int64
```

- Since in this case twice of minority class is greater than majority class , we say data is balanced.

• Split the data in to dependent and independent variables ¶

```
[32]: 1 x = df.iloc[:,1:]  
      2 y = df.iloc[:,0]
```

```
[33]: 1 x.head()
```

```
[33]:
```

	Age	SibSp	Parch	Sex_female	Embarked_C	Embarked_Q	Pclass_1	Pclass_2
0	22	1	0	0	0	0	0	0
1	38	1	0	1	1	0	1	0
2	26	0	0	1	0	0	0	0
3	35	1	0	1	0	0	1	0
4	35	0	0	0	0	0	0	0

```
[34]: 1 y.head()
```

```
[34]: 0    0  
      1    1  
      2    1  
      3    1  
      4    0  
      Name: Survived, dtype: int32
```

- **Using df.loc**

- Assuming 'Column0' is the first column, and you want all rows and all columns except the first one

- `x = df.loc[:, df.columns[1:]]`

- Assuming 'Column0' is the first column and you want all rows of this column

- `y = df.loc[:, df.columns[0]]`

- `y = df.loc[:, 'Survived']`

- The main difference between loc and iloc in pandas lies in how they select data:

- **loc:** Selects data by labels (i.e., using row and column names).

- **iloc:** Selects data by integer position (i.e., using row and column indices).

- Split the data into train and test

```
[35]: 1 from sklearn.model_selection import train_test_split
```

```
[36]: 1 x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state= 42,stratify = y)
```

Building Model

```
[40]: 1 from sklearn.linear_model import LogisticRegression
```

```
[44]: 1 logit = LogisticRegression()  
      2 logit.fit(x_train,y_train)
```

```
[44]: ▾ LogisticRegression  
      LogisticRegression()
```

- Prediction

```
[46]: 1 y_pred_train = logit.predict(x_train)  
      2 y_pred_test = logit.predict(x_test)
```

• Model Evaluation

```
[47]: 1 from sklearn.metrics import accuracy_score
```

```
[49]: 1 print('Training_Accuracy_Score:',accuracy_score(y_train,y_pred_train))
```

Training_Accuracy_Score: 0.8061797752808989

```
[51]: 1 print('Test_Accuracy_Score:',accuracy_score(y_test,y_pred_test))
```

Test_Accuracy_Score: 0.7988826815642458

```
[52]: 1 from sklearn.metrics import classification_report,confusion_matrix
```

```
[53]: 1 print('Training_Classification_Report:',classification_report(y_train,y_pred_train))
```

Training_Classification_Report:		precision	recall	f1-score	support
0	0.83	0.86	0.85	439	
1	0.76	0.71	0.74	273	
accuracy			0.81	712	
macro avg		0.80	0.79	0.79	712
weighted avg		0.80	0.81	0.80	712


```
[54]: 1 print('Test_Classification_Report:',classification_report(y_test,y_pred_test))
```

```
Test_Classification_Report:              precision    recall  f1-score   support

     0       0.80      0.89      0.84       110
     1       0.79      0.65      0.71        69

 accuracy          0.80       179
 macro avg       0.80      0.77      0.78       179
weighted avg       0.80      0.80      0.79       179
```

```
[55]: 1 print('Training_Confusion_Matrix:',confusion_matrix(y_train,y_pred_train))
```

```
Training_Confusion_Matrix: [[379  60]
 [ 78 195]]
```

```
•[61]: 1 print('Testing_Confusion_Matrix:',confusion_matrix(y_train,y_pred_train))
```

```
Testing_Confusion_Matrix: [[98 12]
 [24 45]]
```

1		Predicted Negative	Predicted Positive
2	Actual Negative	TN	FP
3	Actual Positive	FN	TP
4			

Metrics Derived from the Confusion Matrix

1. Accuracy:

- **Definition:** The ratio of correctly predicted samples (both positive and negative) to the total number of samples.
- **Formula:**

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

- **Interpretation:** Indicates the overall effectiveness of the model.

2. Precision:

- **Definition:** The ratio of correctly predicted positive samples to the total predicted positive samples.
- **Formula:**

$$\text{Precision} = \frac{TP}{TP + FP}$$

- **Interpretation:** Measures the accuracy of the positive predictions (how many of the predicted positives are actually positive).

3. Recall (Sensitivity or True Positive Rate):

- **Definition:** The ratio of correctly predicted positive samples to all the actual positive samples.
- **Formula:**

$$\text{Recall} = \frac{TP}{TP + FN}$$

- **Interpretation:** Measures the ability of the model to identify positive samples correctly.

4. F1-Score:

- **Definition:** The harmonic mean of precision and recall, providing a single metric that balances both concerns.
- **Formula:**

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

- **Interpretation:** Useful when you need to balance precision and recall, especially in cases of imbalanced classes.

5. Support:

- **Definition:** The number of actual occurrences of the class in the dataset.
- **Interpretation:** Indicates how many samples there are of each class.

6. Macro Average

Macro average calculates the metric independently for each class and then takes the average (hence treating all classes equally). It does not take class imbalance into account.

$$\text{Macro Avg Precision} = \frac{\text{Precision}_0 + \text{Precision}_1}{2}$$

$$\text{Macro Avg Recall} = \frac{\text{Recall}_0 + \text{Recall}_1}{2}$$

$$\text{Macro Avg F1-Score} = \frac{\text{F1-Score}_0 + \text{F1-Score}_1}{2}$$

7. Weighted Average

Weighted average calculates the metric for each class and then takes the average, weighted by the number of true instances for each class. This accounts for class imbalance.

$$\text{Weighted Avg Precision} = \frac{(\text{Precision}_0 \times \text{Support}_0) + (\text{Precision}_1 \times \text{Support}_1)}{\text{Support}_0 + \text{Support}_1}$$

$$\text{Weighted Avg Recall} = \frac{(\text{Recall}_0 \times \text{Support}_0) + (\text{Recall}_1 \times \text{Support}_1)}{\text{Support}_0 + \text{Support}_1}$$

$$\text{Weighted Avg F1-Score} = \frac{(\text{F1-Score}_0 \times \text{Support}_0) + (\text{F1-Score}_1 \times \text{Support}_1)}{\text{Support}_0 + \text{Support}_1}$$

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Where:

- **TN (True Negative):** Actual class is negative and predicted class is negative.
- **FP (False Positive):** Actual class is negative but predicted class is positive.
- **FN (False Negative):** Actual class is positive but predicted class is negative.
- **TP (True Positive):** Actual class is positive and predicted class is positive.

In your confusion matrix:

```
lua
[[379  60]
 [ 78 195]]
```

[Copy code](#)

This can be broken down as follows:

- True Positives (TP): 195
- True Negatives (TN): 379
- False Positives (FP): 60
- False Negatives (FN): 78

For Class 1:

- **Precision:** The ratio of correctly predicted positive observations to the total predicted positives. It is calculated as:

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Precision} = \frac{195}{195 + 60} = \frac{195}{255} \approx 0.76$$

- **Recall:** The ratio of correctly predicted positive observations to all the observations in the actual class. It is calculated as:

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Recall} = \frac{195}{195 + 78} = \frac{195}{273} \approx 0.71$$

For Class 0:

- **Precision:** The ratio of correctly predicted negative observations to the total predicted negatives. It is calculated as:

$$\text{Precision} = \frac{TN}{TN + FN}$$

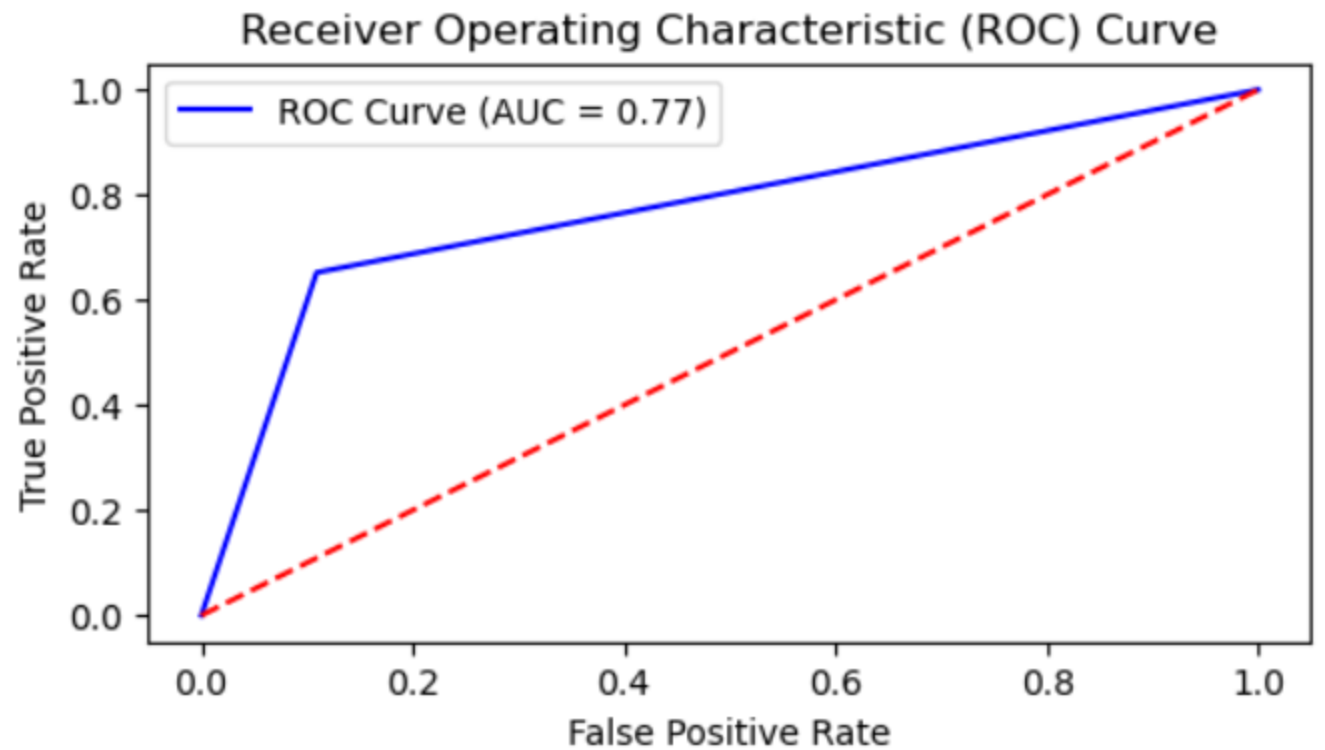
$$\text{Precision} = \frac{379}{379 + 78} = \frac{379}{457} \approx 0.83$$

- **Recall:** The ratio of correctly predicted negative observations to all the observations in the actual class. It is calculated as:

$$\text{Recall} = \frac{TN}{TN + FP}$$

$$\text{Recall} = \frac{379}{379 + 60} = \frac{379}{439} \approx 0.86$$

```
1 from sklearn.metrics import roc_curve, roc_auc_score, RocCurveDisplay
2 fpr, tpr, thresholds = roc_curve(y_test, y_pred_test)
3 roc_auc = roc_auc_score(y_test, y_pred_test)
4
5 # Plot the ROC curve
6 plt.figure(figsize=(6, 3))
7 plt.plot(fpr, tpr, color='blue', label=f'ROC Curve (AUC = {roc_auc:.2f})')
8 plt.plot([0, 1], [0, 1], color='red', linestyle='--')
9 plt.xlabel('False Positive Rate')
10 plt.ylabel('True Positive Rate')
11 plt.title('Receiver Operating Characteristic (ROC) Curve')
12 plt.legend()
13 plt.show()
```



- ▶ The Iris dataset is a classic dataset used in pattern recognition and machine learning.
- ▶ It contains measurements of iris flowers from three different species: Iris setosa, Iris versicolor, and Iris virginica.
- ▶ The dataset is small and simple, making it ideal for beginners in data science and machine learning.

• Description of the Iris Dataset

-
- Features:
- Sepal Length (in cm)
- Sepal Width (in cm)
- Petal Length (in cm)
- Petal Width (in cm)
- Target:
- Species: The type of iris flower, which can be one of the following:
- Iris setosa
- Iris versicolor
- Iris virginica
- https://en.wikipedia.org/wiki/Iris_flower_data_set

CASE STUDY: MULTICLASS CLASSIFICATION(OVR/OVA)

[1]:

```
import os
import warnings
warnings.filterwarnings('ignore')

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

[2]:

```
from sklearn.datasets import load_iris  
# iris = load_iris()  
# iris
```

[3]:

```
df = sns.load_dataset('iris')  
df.head()
```

[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

label encoding of dependent variable

[4]:

```
from sklearn.preprocessing import LabelEncoder  
encoder = LabelEncoder()  
df['species'] = encoder.fit_transform(df['species'])
```

[5]:

```
df.head()
```

[5]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

•[6]:

```
df['species'].value_counts()  
# Since dependent variable is not binary so it is multiclass classification problem
```

[6]:

```
species  
0      50  
1      50  
2      50  
Name: count, dtype: int64
```

[7]:

```
df.info() # this helps to know about any missing value and data type
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 150 entries, 0 to 149  
Data columns (total 5 columns):  
#   Column          Non-Null Count  Dtype  
---  ---  
0   sepal_length    150 non-null   float64  
1   sepal_width     150 non-null   float64  
2   petal_length    150 non-null   float64  
3   petal_width     150 non-null   float64  
4   species         150 non-null   int32  
dtypes: float64(4), int32(1)  
memory usage: 5.4 KB
```

```
[8]: df.describe()
```

```
[8]:
```

	sepal_length	sepal_width	petal_length	petal_width	species
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

```
•[9]: # Since there is not much difference between min and max  
      # so no outlier treatment is required
```

Split the data into dependent and independent variable

•[10]:

```
x = df.iloc[:,0:-1]  
y = df[['species']]
```

[11]:

```
x.head()
```

[11]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

[12]:

```
y.head()
```

[12]:

	species
0	0
1	0

Split the data into train and test

[14]:

```
from sklearn.model_selection import train_test_split
```

[15]:

```
#x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.25,random_state = 43)
```

[16]:

```
#x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.25,random_state = 25)
```

[17]:

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.25,random_state = 100)
```

Building logistic regression model for multiclass classification

[18]:

```
from sklearn.linear_model import LogisticRegression
```

[19]:

```
logit_ovr = LogisticRegression(multi_class='ovr')  
logit_ovr.fit(x_train,y_train)
```

[19]:

```
▼      LogisticRegression  
LogisticRegression(multi_class='ovr')
```


[20]:

```
# Prediction
y_predict_train = logit_ovr.predict(x_train)
y_predict_test = logit_ovr.predict(x_test)
```

[21]:

```
from sklearn.metrics import accuracy_score
print('Training_Accuracy_Score:', accuracy_score(y_train, y_predict_train))
print('*****'*5)
print('Test_Accuracy_Score:', accuracy_score(y_test, y_predict_test))
```

Training_Accuracy_Score: 0.9464285714285714

Test_Accuracy_Score: 0.9473684210526315

[22]:

```
from sklearn.metrics import classification_report, confusion_matrix
print('\nTraining_Classification_Report:\n', classification_report(y_train, y_predict_train))
print('*****'*5)
print('\nTesting_Classification_Report:\n', classification_report(y_test, y_predict_test))
```

Training_Classification_Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	36
1	0.95	0.90	0.92	40
2	0.89	0.94	0.92	36
accuracy			0.95	112
macro avg	0.95	0.95	0.95	112
weighted avg	0.95	0.95	0.95	112

Testing_Classification_Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	14
1	0.90	0.90	0.90	10
2	0.93	0.93	0.93	14
accuracy			0.95	38
macro avg	0.94	0.94	0.94	38
weighted avg	0.95	0.95	0.95	38

[24]:

```
print('\nTraining_Confusion_Matrix:\n',confusion_matrix(y_train,y_predict_train))
print('*****'*5)
print('\nTesting_Confusion_Matrix:\n',confusion_matrix(y_test,y_predict_test))
```

Training_Confusion_Matrix:

```
[[36  0  0]
 [ 0 36  4]
 [ 0  2 34]]
```

Testing_Confusion_Matrix:

```
[[14  0  0]
 [ 0  9  1]
 [ 0  1 13]]
```

The matrix is 3x3, which means there are three classes in the classification problem (e.g., setosa, versicolor, and virginica for the Iris dataset). ↑

Rows represent the actual classes (true labels).and Columns represent the predicted classes.

Row 0: Actual class is setosa.

Column 0: 36 samples correctly predicted as setosa. Column 1: 0 samples incorrectly predicted as versicolor (false negatives). Column 2: 0 samples incorrectly predicted as virginica (false negatives).

Row 1: Actual class is versicolor.

Column 0: 0 samples incorrectly predicted as setosa (false positives). Column 1: 36 samples correctly predicted as versicolor. Column 2: 4 samples incorrectly predicted as virginica (false negatives).

Row 2: Actual class is virginica.

Column 0: 0 samples incorrectly predicted as setosa (false positives). Column 1: 2 samples incorrectly predicted as versicolor (false positives). Column 2: 34 samples correctly predicted as virginica.

True Positives (Diagonal Elements):

setosa: 36 correct versicolor: 36 correct virginica: 34 correct

Misclassifications:

versicolor misclassified as virginica: 4 times

virginica misclassified as versicolor: 2 times

Softmax is a mathematical function that converts a vector of values into a probability distribution. The elements of the output vector lie in the range (0, 1), and their sum is 1. This function is often used in the final layer of a neural network-based classifier to produce a probability distribution over classes.

Mathematical Definition

For a vector $\mathbf{z} = [z_1, z_2, \dots, z_K]$, the softmax function is defined as:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

where:

- e is the base of natural logarithms.
- z_i is the i -th element of the input vector \mathbf{z} .
- K is the number of classes.

WHAT IS SOFTMAX?

1. **Input:** In a classification task, the output layer of a neural network typically has one neuron per class. Each neuron outputs a raw score (also called logits) for its corresponding class.
2. **Softmax Transformation:** These raw scores are passed through the softmax function to produce probabilities. Each score is exponentiated and then normalized by the sum of all exponentiated scores.
3. **Output:** The output of the softmax function is a probability distribution. Each value represents the probability that the input belongs to the corresponding class.

HOW SOFTMAX WORKS IN MACHINE LEARNING

Consider a neural network trained to classify images of animals into three categories: cats, dogs, and rabbits. The network's final layer produces three raw scores (logits) for a given input image.

1. **Raw Scores (Logits):** Suppose the network outputs the following scores for a particular image:

$$\mathbf{z} = [2.0, 1.0, 0.1].$$

2. **Apply Softmax:**

$$\sigma(\mathbf{z})_1 = \frac{e^{2.0}}{e^{2.0} + e^{1.0} + e^{0.1}} = \frac{e^{2.0}}{e^{2.0} + e^{1.0} + e^{0.1}}$$

$$\sigma(\mathbf{z})_2 = \frac{e^{1.0}}{e^{2.0} + e^{1.0} + e^{0.1}} = \frac{e^{1.0}}{e^{2.0} + e^{1.0} + e^{0.1}}$$

$$\sigma(\mathbf{z})_3 = \frac{e^{0.1}}{e^{2.0} + e^{1.0} + e^{0.1}} = \frac{e^{0.1}}{e^{2.0} + e^{1.0} + e^{0.1}}$$

EXAMPLE: MULTI-CLASS CLASSIFICATION

Calculating these:

$$e^{2.0} \approx 7.389, \quad e^{1.0} \approx 2.718, \quad e^{0.1} \approx 1.105$$

$$\sigma(\mathbf{z})_1 \approx \frac{7.389}{7.389 + 2.718 + 1.105} = \frac{7.389}{11.212} \approx 0.659$$

$$\sigma(\mathbf{z})_2 \approx \frac{2.718}{7.389 + 2.718 + 1.105} = \frac{2.718}{11.212} \approx 0.242$$

$$\sigma(\mathbf{z})_3 \approx \frac{1.105}{7.389 + 2.718 + 1.105} = \frac{1.105}{11.212} \approx 0.099$$

3. **Output Probabilities:** The softmax function converts the raw scores into probabilities: $[0.659, 0.242, 0.099]$. This means there is a 65.9% chance the image is a cat, a 24.2% chance it is a dog, and a 9.9% chance it is a rabbit.

1. Simplicity and Interpretability:

- Logistic regression is relatively simple to understand and implement.
- The model provides coefficients for each feature, which can be interpreted to understand the relationship between the features and the outcome. This makes it easier to explain to non-technical stakeholders.

2. Probabilistic Interpretation:

- Logistic regression outputs probabilities, providing a measure of confidence in the predictions. This probabilistic interpretation is useful in many applications, such as medical diagnostics and risk assessment.

3. Efficiency:

- Logistic regression is computationally efficient, both in terms of training and prediction. It works well with small to moderately large datasets.

4. No Need for Feature Scaling:

- Although feature scaling (e.g., standardization) can improve the performance of logistic regression, it is not strictly necessary for the algorithm to function properly.

5. Handles Binary and Multiclass Problems:

- While logistic regression is primarily used for binary classification, extensions like multinomial logistic regression can handle multiclass classification problems.

ADVANTAGE OF LOGISTIC REGRESSION

6. Less Prone to Overfitting:

- Logistic regression tends to be less prone to overfitting compared to more complex models, especially when regularization techniques (e.g., L1 or L2 regularization) are applied.

7. Regularization:

- Regularization techniques (L1 and L2) can be easily incorporated into logistic regression to prevent overfitting and manage multicollinearity among features.

8. Baseline Model:

- Logistic regression serves as a good baseline model in many classification tasks. It is often used as a starting point before moving to more complex models.

9. Wide Applicability:

- Logistic regression can be applied to a wide range of problems, including medical research, finance, marketing, and social sciences, where binary outcomes are common.

10. Robust to Missing Data:

- Logistic regression can be reasonably robust to missing data, especially if proper data imputation techniques are used.

11. Assumptions:

- Logistic regression makes fewer assumptions about the distributions of the features compared to some other models like linear discriminant analysis (LDA).

Example Use Cases

- **Medical Diagnostics:** Predicting whether a patient has a particular disease (yes/no) based on diagnostic features.
- **Marketing:** Determining whether a customer will respond to a marketing campaign (yes/no).
- **Finance:** Predicting whether a loan applicant will default (yes/no).

Benefits of Softmax

- **Interpretable Output:** Provides a clear and interpretable probability distribution over classes.
- **Normalization:** Ensures that the outputs sum to 1, making them valid probabilities.
- **Differentiable:** The softmax function is differentiable, allowing it to be used in gradient-based optimization methods for training neural networks.

Applications

- **Multi-Class Classification:** Commonly used in the final layer of neural networks for tasks where there are more than two classes.
- **Attention Mechanisms:** Used in attention mechanisms in sequence models, such as transformers, to weigh the importance of different input elements.

Conclusion

The softmax function is a crucial component in machine learning for converting raw model outputs into probabilities. It enables models to make probabilistic predictions that are interpretable and useful for decision-making in multi-class classification tasks.

1. Methodology:

- **OvR (One-vs-Rest):**
 - **Approach:** Trains one binary classifier per class. Each classifier is trained to distinguish between one class and all other classes combined.
 - **Prediction:** When making predictions, each binary classifier gives a score for its class. The class with the highest score is chosen as the final prediction.
- **Softmax:**
 - **Approach:** Uses a single classifier to model all classes simultaneously. It applies the softmax function to compute the probabilities of each class.
 - **Prediction:** The softmax function outputs a probability distribution over all classes. The class with the highest probability is chosen as the final prediction.

DIFFERENCE BETWEEN OVR AND SOFTMAX

2. Model Complexity:

- **OvR:**
 - Requires training k binary classifiers for k classes, where each classifier is trained independently.
 - Often used with models that are inherently binary, like logistic regression, when extended to multiclass problems.
- **Softmax:**
 - Requires training a single classifier that directly handles all k classes.
 - Generally results in a more compact model compared to OvR, as there's only one model to train and manage.

3. Performance:

- OvR:
 - Can sometimes lead to less optimal performance if the binary classifiers do not cooperate well. The performance can also be influenced by class imbalance since each classifier deals with the imbalanced class data separately.
- Softmax:
 - Often better suited for problems where classes are not mutually exclusive or when you need to capture the relationship between classes. Softmax tends to be more robust in handling class imbalances and interactions between classes.

4. Probabilities:

- OvR:
 - Each classifier gives a score, but these scores are not probabilistic. They are typically transformed into probabilities by applying some method (e.g., Platt scaling).
- Softmax:
 - Directly provides class probabilities, which are normalized to sum to 1. This makes it easier to interpret the output as a probability distribution.

5. Applicability:

- OvR:
 - Can be used with binary classifiers extended to multiclass tasks, such as binary SVMs.
- Softmax:
 - Typically used with models specifically designed for multiclass classification, like multinomial logistic regression or neural networks.