MIS4311
Machine Learning Applications

Spring 2025

Lecture #4

Supervised Learning

Classification:

- The classification technique or model attempts to get some conclusion from observed values.
- In classification problem, we have the categorized output such as “Black” or “white” or “Teaching” and “Non-Teaching”.
- While building the classification model, we need to have training dataset that contains data points and the corresponding labels.

  For example, if we want to check whether the image is of a car or not.
- For checking this, we will build a training dataset having the two classes related to “car” and “no car”.
- Then we need to train the model by using the training samples.

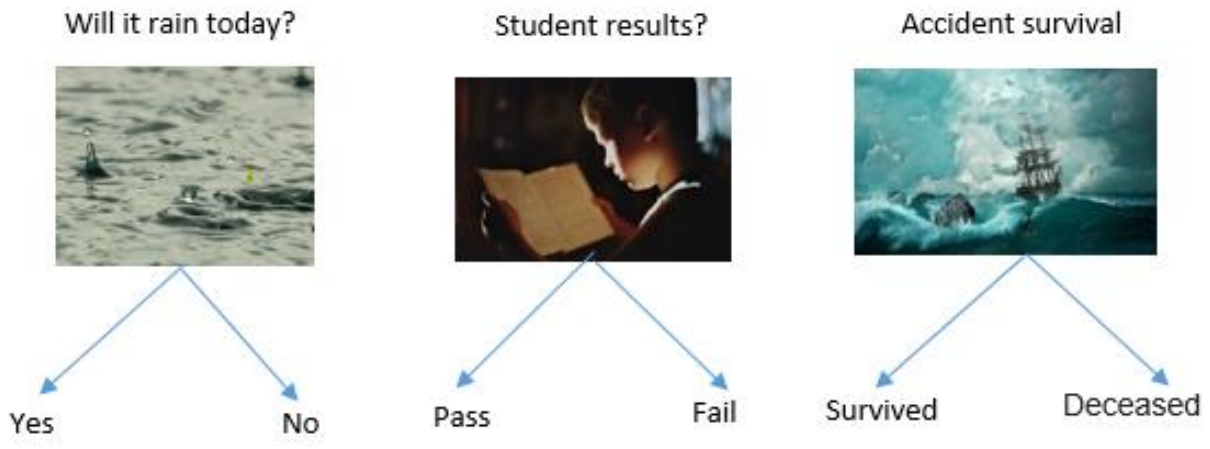
Logistic Regression

Logistic regression is a statistical method that is used for building machine learning models **when we have a binary output (dependent) variable**.

It is used to describe data and the relationship between **one dependent variable and one or more independent variables.**

For example: given the parameters, will the student pass or fail? Will it rain or not? etc.

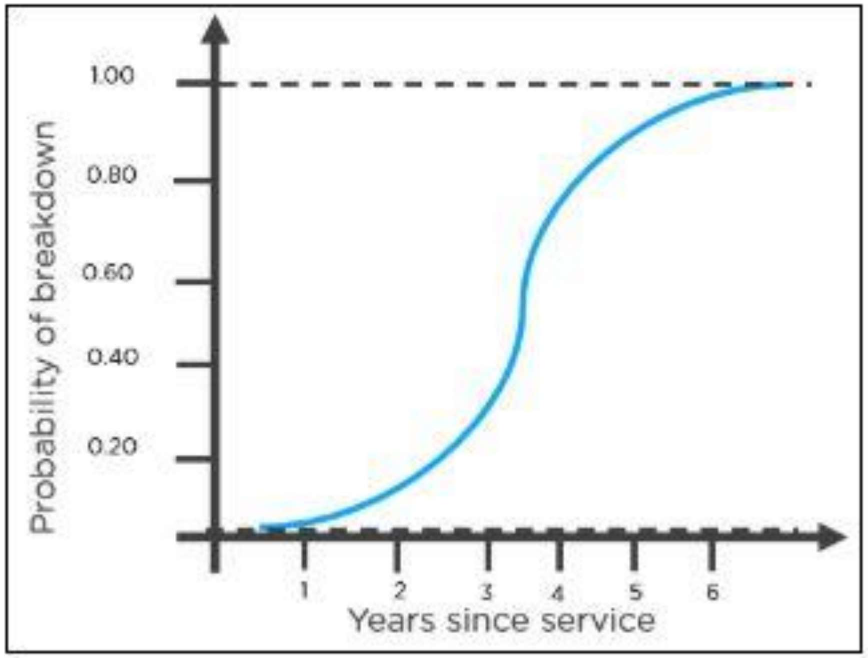
We may have continuous or categorical independent variables



Logistic Regression

The name "logistic regression" is **derived from the concept of the logistic function.**

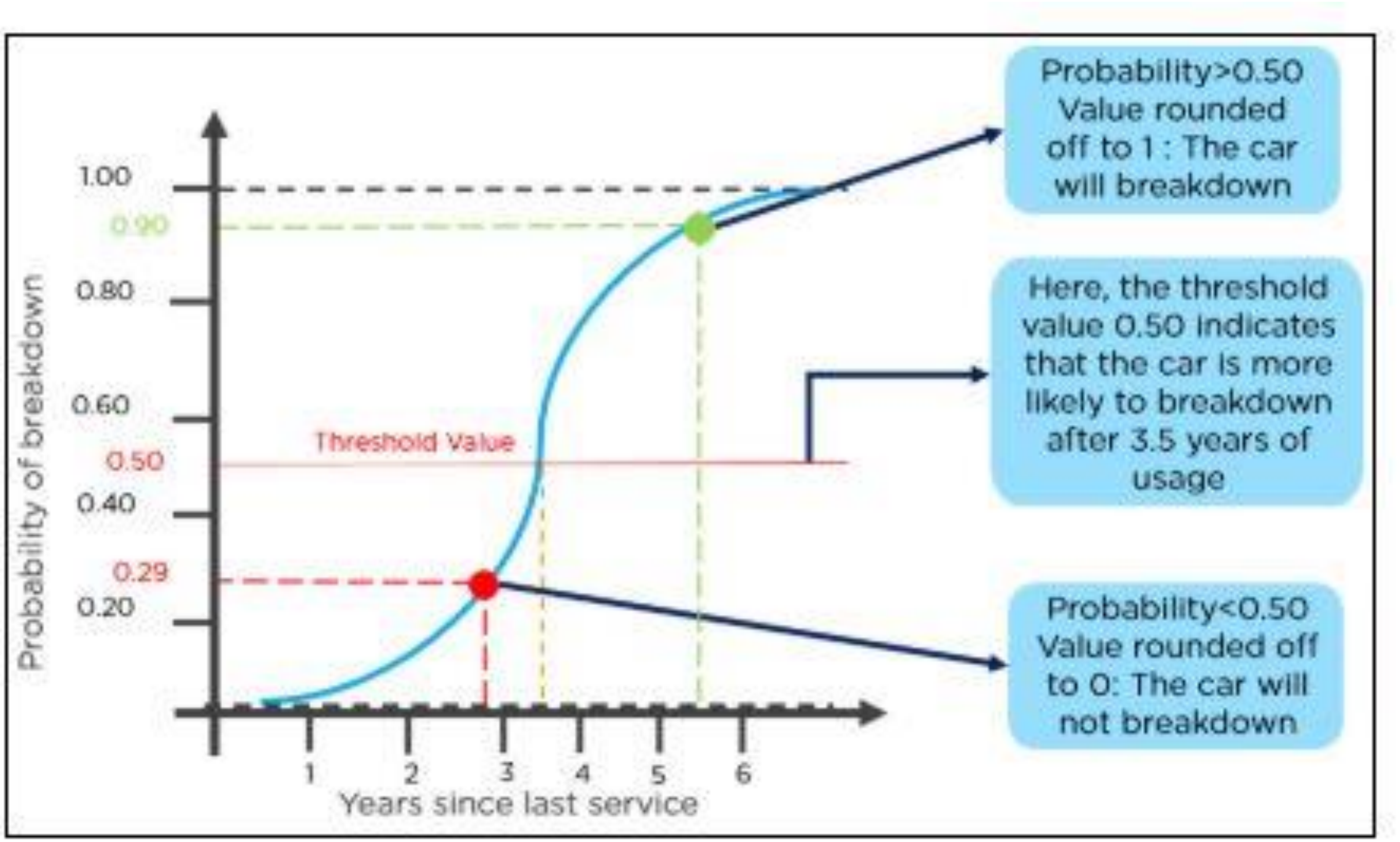
The logistic function is also known **as the sigmoid function**. The value of this **logistic function lies between zero and one**.



An example of a logistic function we can use to find the probability of a vehicle breaking down, depending on how many years it has been since it was serviced last.

Logistic Regression

Use the results in the graph to decide whether the vehicle will break down or not, according to the **threshold** value.



Logistic Regression

Advantages of the LR:

•Logistic regression performs better when **the data is linearly separable**

•It does not require too many computational resources

•It is easy to implement and train a model using logistic regression

Assumptions of the LR:

•Independent variables exhibit **a linear relationship with the log of output variables.**

• Input variables should be independent of each other.

•Output variable should be binary.

Logistic Regression Algorithm

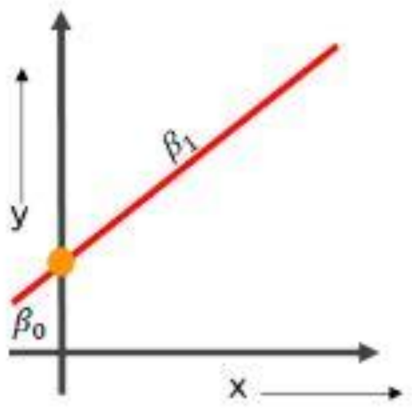
To understand logistic regression, let's go over the odds of success.

Odds (θ) = Probability of an event happening / Probability of an event not happening

$$\theta = p / (1 - p)$$

The values of odds range from zero to ∞ and the values of probability lies between zero and one.

Consider the equation of a straight line: $y = \beta_0 + \beta_1 * x$



β_0 is the y-intercept

β_1 is the slope of the line

Logistic Regression Algorithm

To predict the odds of success, we use the following formula:

$$\log\left(\frac{p(x)}{1-P(x)}\right) = \beta_0 + \beta_1 x$$

Exponentiating both the sides, we have:

$$e^{ln}\left(\frac{p(x)}{1-p(x)}\right) = e^{\beta_0+\beta_1 x}$$

$$\left(\frac{p(x)}{1-p(x)}\right) = e^{\beta_0+\beta_1 x}$$

Thus $p(x) = \dfrac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}}$ that is equivalent to $p(x) = \dfrac{1}{1+e^{-(\beta_0+\beta_1 x)}}$

which is the sigmoid function

Linear Regression vs Logistic Regression

Linear Regression	Logistic Regression
Used to solve regression problems	Used to solve classification problems
The response variables are continuous in nature	The response variable is categorical in nature
It helps estimate the dependent variable when there is a change in the independent variable	It helps to calculate the possibility of a particular event taking place
It is a straight line	It is an S-curve (S = Sigmoid)

Naive Bayes' Algorithm

There are three main types of **Naïve Bayes** algorithms:

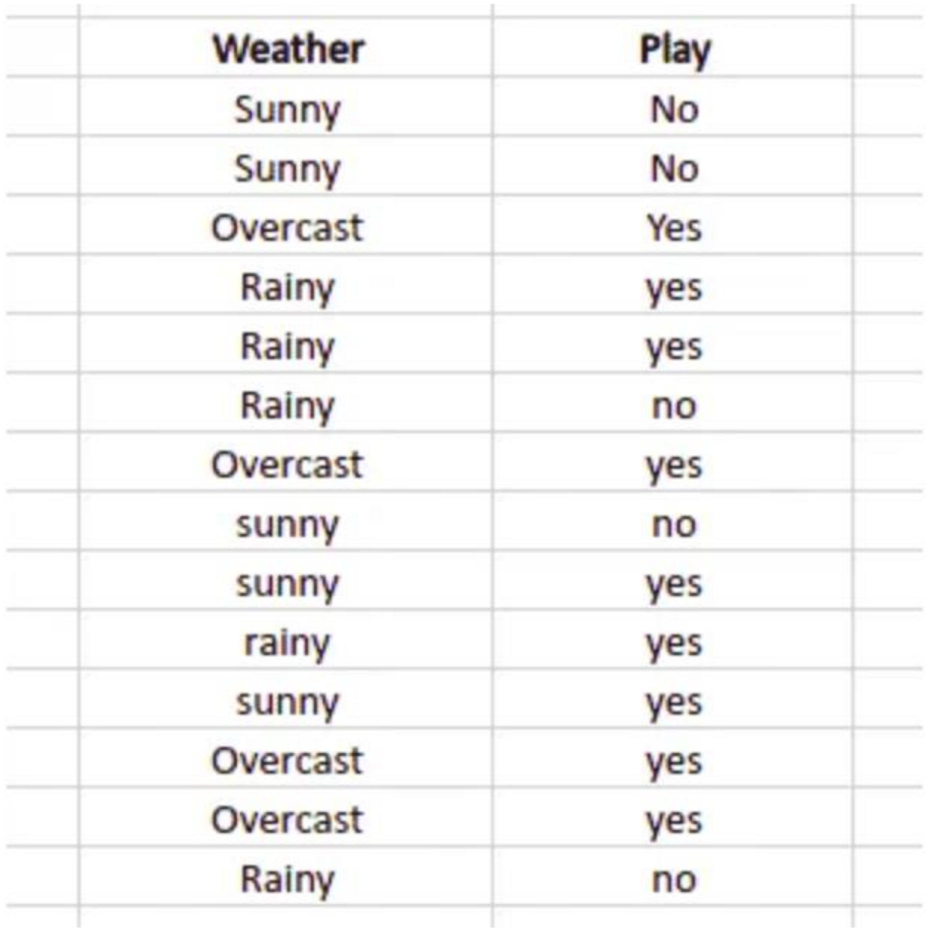
Gaussian Naïve Bayes is used for **continuous data,** It assumes that the features are normally distributed. It calculates the mean and variance of each feature for each class, and uses these values to calculate the probability of a class given a set of features.

Multinomial Naïve Bayes is used for **discrete data**, It assumes that the features are counts of occurrences of events. It calculates the probability of each feature for each class, and uses these values to calculate the probability of a class given a set of features.

Bernoulli Naïve Bayes is used for **binary data,** It assumes that the features are binary variables, indicating the presence or absence of a feature. It calculates the probability of each feature for each class, and uses these values to calculate the probability of a class given a set of features.

1st Example

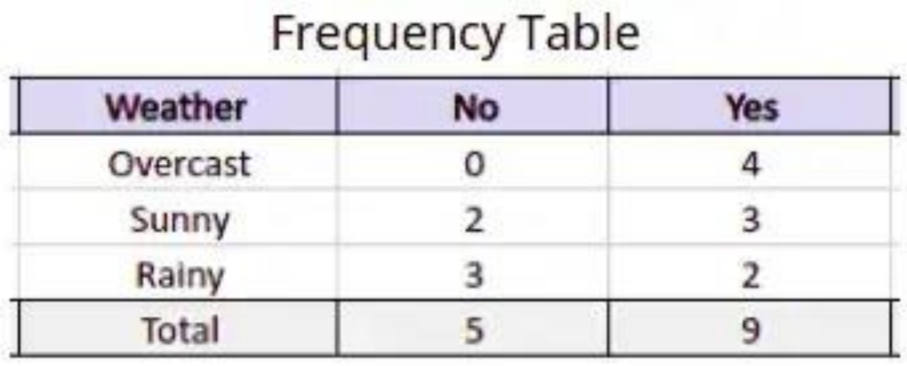
Given an example of **weather conditions and playing sports**, we need to calculate the probability of playing sports and not playing sports, finding out whether the statement has dependent or independent variables.



Weather	Play
Sunny	No
Sunny	No
Overcast	Yes
Rainy	yes
Rainy	yes
Rainy	no
Overcast	yes
sunny	no
sunny	yes
rainy	yes
sunny	yes
Overcast	yes
Overcast	yes
Rainy	no

1st Example

Set frequency and likelihood tables **to simplify prior and posterior probability calculations.** The Frequency table shows how often labels appear for each feature



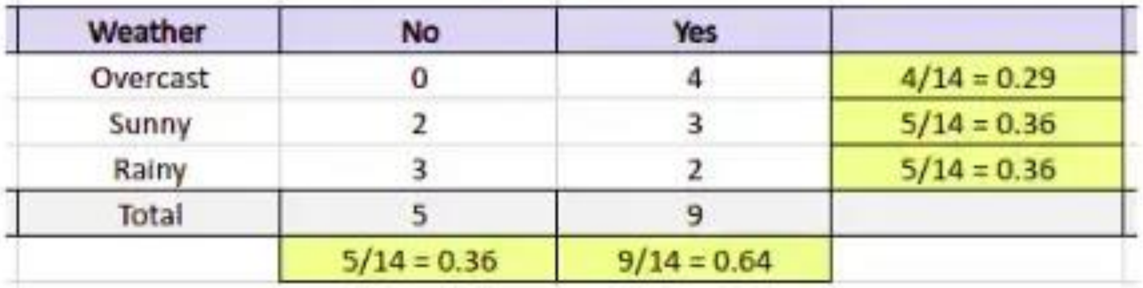
Frequency Table

Weather	No	Yes
Overcast	0	4
Sunny	2	3
Rainy	3	2
Total	5	9

1st Example

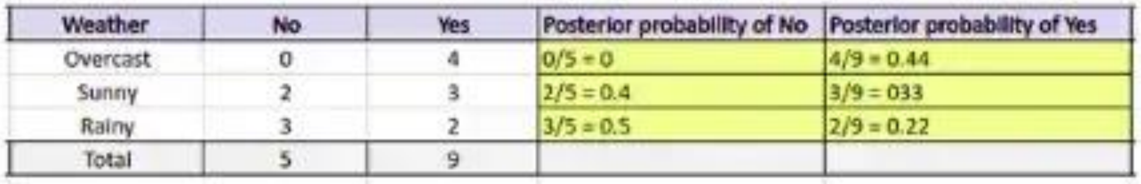
Table 1 shows the **prior probabilities** of labels, and Table 2 shows the **posterior probability**.

Table 1



Weather	No	Yes	
Overcast	0	4	4/14 = 0.29
Sunny	2	3	5/14 = 0.36
Rainy	3	2	5/14 = 0.36
Total	5	9	
	5/14 = 0.36	9/14 = 0.64	

Table 2



Weather	No	Yes	Posterior probability of No	Posterior probability of Yes
Overcast	0	4	0/5 = 0	4/9 = 0.44
Sunny	2	3	2/5 = 0.4	3/9 = 033
Rainy	3	2	3/5 = 0.5	2/9 = 0.22
Total	5	9		

1st Example

Probability of Playing:

We have the formula for the Naive Bayes classification, which is

P(Yes | Overcast) = P(Overcast | Yes) P(Yes) / P (Overcast).

Now, let's calculate the prior probabilities:

P(Overcast) = 4/14 = 0.29

P(Yes)= 9/14 = 0.64

The next step is to find the posterior probability, which can be easily calculated by:

P(Overcast | Yes) = 4/9 = 0.44

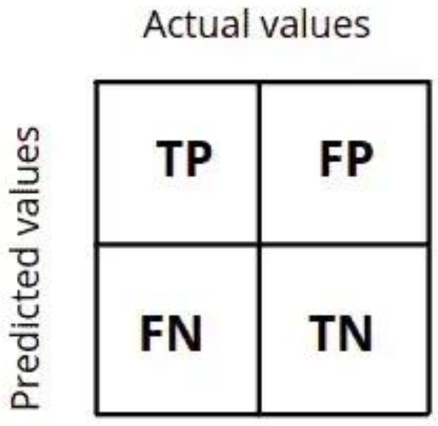
Once we have the posterior and prior probabilities, we can put them back in our main formula to calculate **the probability of playing when the weather is overcast.**

- P(Yes | Overcast) = 0.44 * 0.64 / 0.29 = 0.98

Evaluating Naive Bayes Classification performance

Confusion Matrix for Binary classification

The confusion matrix is also known as the error matrix. It is a table layout that allows visualization of the performance of a classification algorithm.



Predicted values \ Actual values		
	TP	FP
	FN	TN

TP: True positive
FP: False positive
FN: False negative
TN: True negatve

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

Evaluating Naive Bayes Classification performance

There are two complementary pairs of measurements that can help us to interpret the performance of a classifier, namely **sensitivity and specificity**, and **precision and recall**. Their definitions are shown next, followed by some explanation.

$$\text{Sensitivity} = \frac{\#TP}{\#TP + \#FN}$$

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

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

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

They can be combined to give a single measure, the F_1 measure, which can be written in terms of precision and recall as:

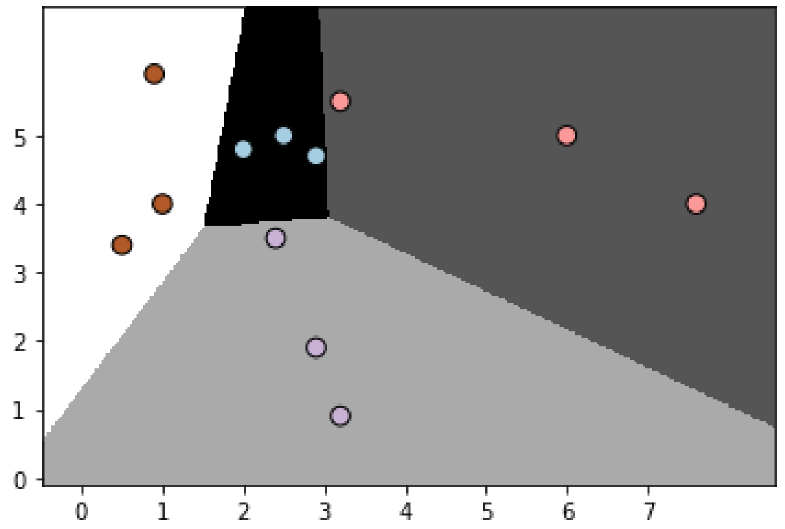
$$F_1 = 2\frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

Logistic Regression with Python

Using modules:

import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

Dataset:



X = np.array([[2, 4.8], [2.9, 4.7], [2.5, 5], [3.2, 5.5], [6, 5], [7.6, 4], [3.2, 0.9], [2.9, 1.9], [2.4, 3.5], [0.5, 3.4], [1, 4], [0.9, 5.9]])
y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])

Logistic Regression with Python

Create the logistic regression classifier:

Classifier_LR = linear_model.LogisticRegression(solver='liblinear', C=75)

Solver: {‘lbfgs’, ‘liblinear’, ‘newton-cg’, ‘newton-cholesky’, ‘sag’, ‘saga’}
Algorithm to use in the optimization problem.
For small datasets, ‘liblinear’ is a good choice

C : *float, default=1.0*
Inverse of regularization strength; must be a positive float. Like in support vector machines.

Logistic Regression with Python

Train this classifier:

Classifier_LR.fit(X, y)

visualize the output:

```
def Logistic_visualize(Classifier_LR, X, y):
    min_x, max_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    min_y, max_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0
    mesh_step_size = 0.02
    x_vals, y_vals = np.meshgrid(np.arange(min_x, max_x, mesh_step_size), np.arange(min_y, max_y,
mesh_step_size))
    output =Classifier_LR.predict(np.c_[x_vals.ravel(), y_vals.ravel()])
    output = output.reshape(x_vals.shape)
    plt.figure()
    plt.pcolormesh(x_vals, y_vals, output, cmap=plt.cm.gray)
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='black', linewidth=1,cmap=plt.cm.Paired)
    plt.xlim(x_vals.min(), x_vals.max())
    plt.ylim(y_vals.min(), y_vals.max())
    plt.xticks((np.arange(int(X[:, 0].min() - 1), int(X[:, 0].max() + 1),1.0)))
    plt.yticks((np.arange(int(X[:, 1].min() - 1), int(X[:, 1].max() + 1),1.0)))
    plt.show()
Logistic_visualize(Classifier_LR, X, y)
```

Building a NB Classifier in Python

- **Step 1:** *import sklearn :* Scikit-learn which is one of the best machine learning modules in Python.

- **Step 2:** Import Scikit-learn's dataset : we use the Breast Cancer Wisconsin Diagnostic Database. The dataset includes various information about breast cancer tumors, as well as classification labels of **malignant or benign**. The dataset has **569 instances**, or data, on **569 tumors** and includes information on **30 attributes**, or features, such as the **radius of the tumor, texture, smoothness, and area**

```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()   # load the dataset.
```

Building a NB Classifier in Python

list of important **dictionary keys**:

Classification label names(target_names)

The actual labels(target)

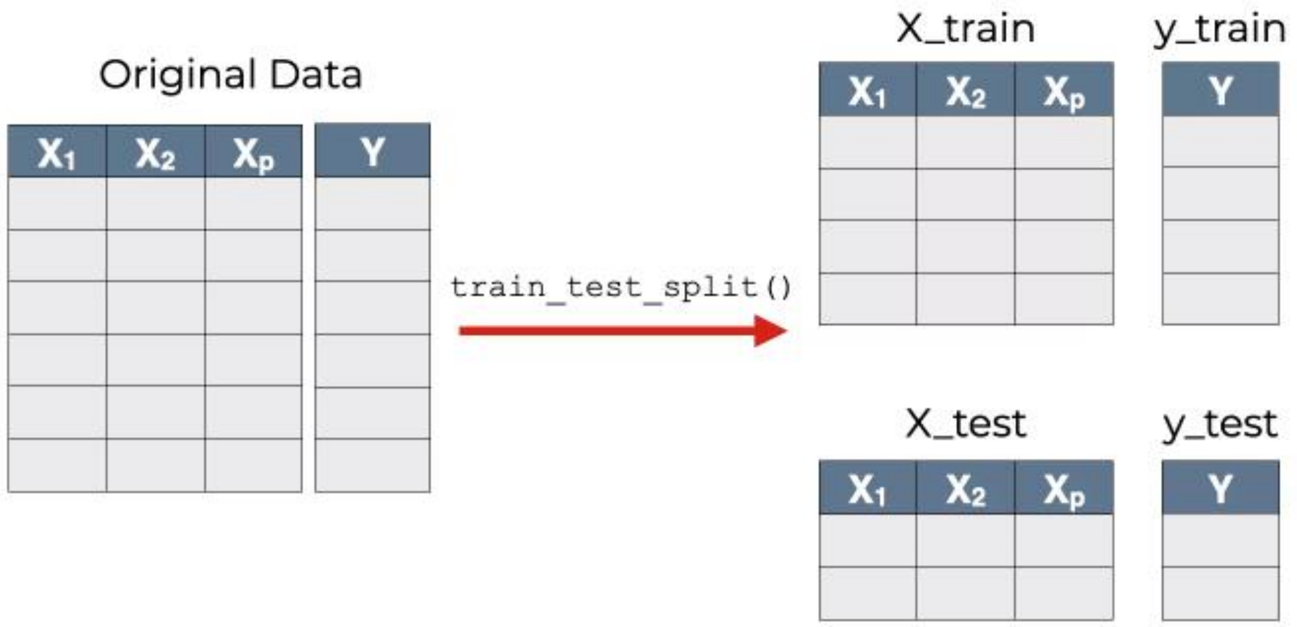
The attribute/feature names(feature_names)

label_names = data['target_names']
labels = data['target']
feature_names = data['feature_names']
features = data['data']

#For label names 0 represents malignant cancer and 1 represents benign cancer.

Building a NB Classifier in Python

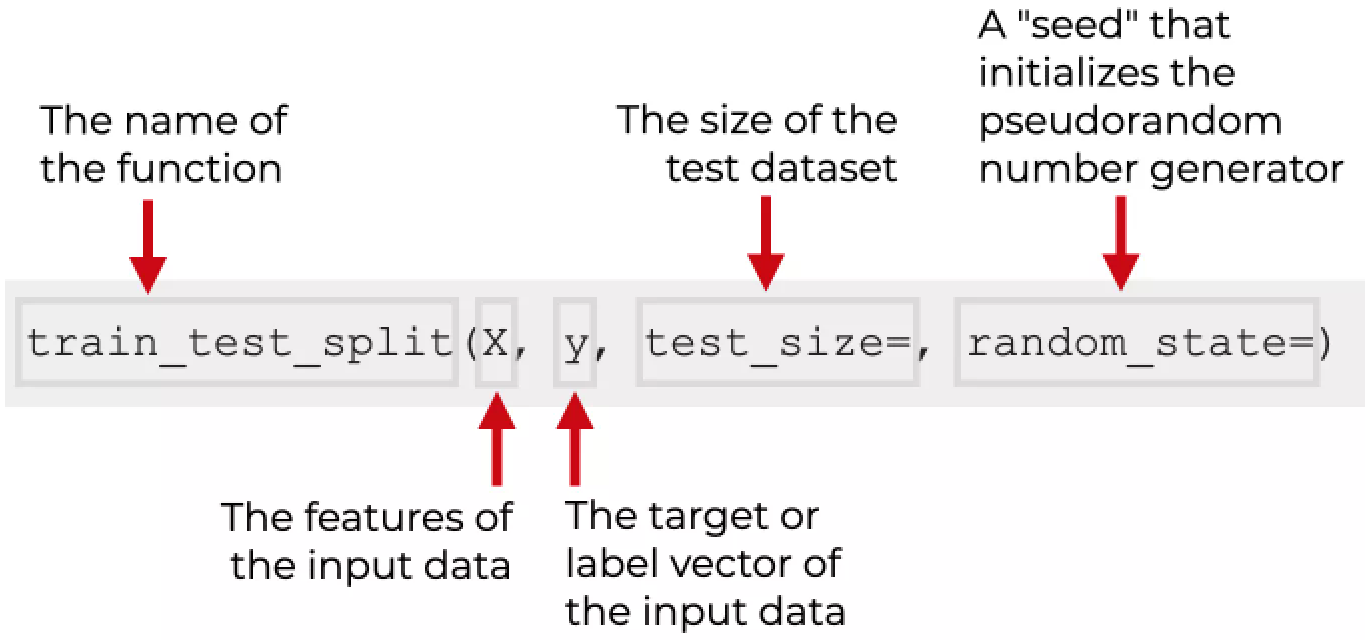
- **Step 3:** Organizing data into sets: we will divide our data into two parts namely a training set and a test set. Splitting the data into these sets is very important because we have to test our model on
- the unseen data.



Building a NB Classifier in Python

from sklearn.model_selection import train_test_split

train, test, train_labels, test_labels = train_test_split(features,labels,test_size = 0.40, random_state = 42)


The name of the function
The size of the test dataset
A "seed" that initializes the pseudorandom number generator
train_test_split(X, y, test_size=, random_state=)
The features of the input data
The target or label vector of the input data

we are using 40 % of the data **for testing** and the remaining data would be used for training the model.

Building a NB Classifier in Python

- **Step 4**: Building the model: We are going to use Naïve Bayes algorithm for building the model.

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()               #initialize the model
#train the model by fitting it to the data by using gnb.fit()
model = gnb.fit(train, train_labels)
```

Building a NB Classifier in Python

- **Step 5**: Evaluating the model and its accuracy: evaluate the model by making predictions on our test data

```
preds = gnb.predict(test)
print(preds)
# 0s and 1s are the predicted values for the tumor classes
```

Comparing test_labels and preds to find out the accuracy of our model.

```
from sklearn.metrics import accuracy_score
print(accuracy_score(test_labels,preds))

# Result is 0.951754385965 (nearly 95%)
```

Next Week

- Decision Tree
- K-Nearest Neighbors (KNNs)

Thank you for your participation ☺