Python Numpy

What is NumPy?

NumPy, short for Numerical Python, is one of the most important foundational packages for numerical computing in Python

Numpy is a general-purpose array-processing package.
Numpy is the core library or fundamental package for scientific computing in Python.
Numpy provides a high-performance multidimensional array object, and tools for working with these arrays and provides fast array-oriented arithmetic operations and flexible broadcasting capabilities.
Numpy has linear algebra, Fourier transform, and random number capabilities
Numpy has tools for integrating with different databases and C/C++ and Fortran code efficiently ·
A C API for connecting NumPy with libraries written in C, C++, or FORTRAN.
NumPy's library of algorithms written in the C language can operate on this memory without any type checking or other overhead.
Also has tools for reading/writing array data to disk and working with memory-mapped files

It contains various features like:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Extension package to Python for multi-dimensional arrays
- Closer to hardware (efficiency)
- Designed for scientific computation (convenience)
- Also known as *array oriented computing*
- Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.

Numpy -based algorithms are generally 10 to 100 times faster (or more) than their pure Python counterparts and use significantly less memory.

```
>>> import numpy as np
>>> my_arr = np.arange(1000000)
```

```
>>> my_list = list(range(1000000))
    # Now let's multiply each sequence by 2:
>>> %time for _ in range(10): my_arr2 = my_arr * 2
    CPU times: user 20 ms, sys: 50 ms, total: 70 ms
    Wall time: 72.4 ms
>>> %time for _ in range(10): my_list2 = [x * 2 for x in my_list]
    CPU times: user 760 ms, sys: 290 ms, total: 1.05 s
    Wall time: 1.05 s
```

Some of the sub-packages of numpy

Sub-Package	Purpose	Comments
core	basic objects	all names exported to numpy
lib	additional utilities	all names exported to numpy
linalg	basic linear algebra	old Linear Algebra from Numeric
fft	discrete Fourier transforms	old FFT from Numeric
random	random number generators	old Random Array from Numeric
distutils	enhanced build and distribution	improvements built on standard distutils
testing	unit-testing	utility functions useful for testing
f2py	automation wrapping of Fortran code	a useful needed by SciPy

Object Essentials

- NumPy provides two fundamental objects:
  - a N-dimensional array object (ndarray) and
  - an universal function object (ufunc).

  In addition, there are other objects that build on top of these

- An N-dimensional array is a homogeneous collection of "items" indexed using N integers.

- There are two essential pieces of information that define an N-dimensional array:
  1) the shape of the array, and
  2) the kind of item the array is composed of.

- The **shape of the array** is a tuple of N integers (one for each dimension) that provides information on how far the index can vary along that dimension.

- The other important information describing an array is the **kind of item the array is composed of**. Because every ndarray is a homogeneous collection of exactly the same data-type, every item takes up the same size block of memory, and each block of memory in the array is interpreted in exactly the same way.

Data-type Descriptor

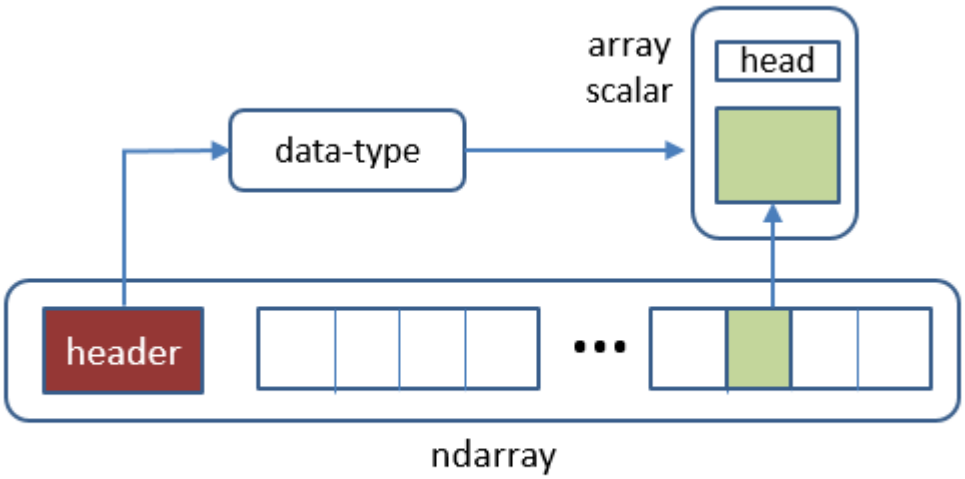


Fig Conceptual diagram of 3 objects

Conceptual diagram showing the relationship between the three fundamental objects used to describe the data in an array:

1) the ndarray itself

2) the data-type object that describes the layout of a single fixed-size element of the array

3) the array-scalar Python object that is returned when a single element of the array is accessed

ndarray = block of memory + indexing scheme + data type descriptor

- raw data : one dimensional contiguous block of memory
- indexing scheme : how to locate an element
- data type descriptor : how to interpret an element

- In Numpy, number of dimensions of the array is called rank of the array.
- A tuple of integers giving the size of the array along each dimension is known as shape of the array.
- An array class in Numpy is called as **ndarray**.
- Elements in Numpy arrays are accessed by using square brackets and can be initialized by using nested Python Lists.
- All arrays in NumPy are indexed starting at 0 and ending at M-1 following the Python convention

```
>>> import numpy as np
>>> np.arange(10)        # one dimensional array
    array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.random.randint(10,20)
    15
>>> np.random.randint( 10,20,(2,3) )       # two dimensional array
     array( [ [14, 14, 10],
              [12, 16, 17] ] )
>>> np.array( [ [1,2,3],[4,5,6] ] )
     array( [ [1, 2, 3],
              [4, 5, 6] ] )
>>> c =  [ [1,2,3],[4,5,6] ] )
>>> np.array ( c )
      array( [ [1, 2, 3],
               [4, 5, 6] ] )
```

```
>>>  X=np.array( c )      # can store in one more variable as
>>> x = np.random.randint( 10,20,(2,3) )
>>> x
    array( [ [11, 11, 18],
             [11, 13, 10]])
>>> x * 2
    array( [ [22, 22, 36],
             [22, 26, 20] ] )
>>> x + 10                              # x value will not change
          array( [ [21, 21, 28],         # we can store in a different variable, if needed
                   [21, 23, 20] ] )
>>> x
    array( [ [11, 11, 18],
             [11, 13, 10]])
```

Arithmetic Operations with NumPy Arrays

Arrays are important because they enable you to express batch operations on data without writing any for loops. NumPy users call this vectorization. Any arithmetic operations between equal-size arrays applies the operation element-wise

```
>>> import numpy as np
>>> arr = np.array([[1., 2., 3.], [4., 5., 6.]])
>>> arr
    array( [ [ 1., 2., 3.],
             [ 4., 5., 6.] ] )
>>> arr * arr
    array( [ [ 1., 4., 9.],
             [ 16., 25., 36.] ] )
>>> 1 / arr
    array( [ [ 1. , 0.5 , 0.3333],
             [ 0.25 , 0.2 , 0.1667] ] )
```

```
>>> arr ** 0.5
    array( [ [ 1. , 1.4142, 1.7321],
             [ 2. , 2.2361, 2.4495]])
>>> arr2 = np.array([[0., 4., 1.], [7., 2., 12.]])
>>> arr2
    array( [ [ 0., 4., 1.],
             [ 7., 2., 12.] ] )
>>> arr2 > arr
    array( [ [False, True, False],
             [ True, False, True]], dtype=bool)
```

```
>>> import numpy as np
>>> a = np.array ( [ [1,2,3] , [4,5,6] ] )
>>> a
    array( [ [ 1,2,3],
             [4, 5, 6] ] )
>>> a.shape
    (2, 3)
>>> a.dtype
    dtype('int32')
```

```
>>> a.ndim
    2
>>> b = np.array ( [ [10,20], [30,40],[50,60] ] )
>>> b.shape
      (3, 2)
>>> b.ndim
       2
>>> c=np.array ( [1,2,3], dtype=complex)
      array( [ 1.+0.j, 2.+0.j, 3.+0.j ] )
```

- ***The dtype attribute***, the data type of the ndarray array, and the type of the data type have been described previously.
  >>> np.arange(4, dtype=float) ***output:*** array([0., 1., 2., 3.])
  >>> np.arange(4, dtype='D') #'D' indicates *double-precision number*
  Output: array([0.+0.j, 1.+0.j, 2.+0.j, 3.+0.j])
  >>> np.array([1.22,3.45,6.779], dtype='int8')
  output: array([1, 3, 6], dtype=int8)

- ***Ndim attribute,*** the number of array dimensions
  >>> a = np.array([[1,2,3], [7,8,9]])
  >>> print(a.ndim) ***Output:*** 2

- ***Shape attribute,*** the scale of the array object, for the matrix, ie n rows and m columns, the shape is a tuple (tuple)

  >>> print(a.shape) ***output:***(2, 3)

Basic indexing (slicing)

- ❑ Indexing is also sometimes called slicing and slicing for an ndarray works very similarly as it does for other Python sequences.
- ❑ Slicing for an ndarray works very similarly as it does for other Python sequences

But there are three big differences:

- Slicing can be done over multiple dimensions
- Exactly one ellipsis object can be used to indicate several dimensions at once
- Slicing cannot be used to expand the size of an array (unlike lists).

```
>>> import numpy as np
>>> a = np.array ( [ [1,2,3,4,5] ,
                     [4,6,8,10,12] ] )
>>> a[0,0]
    1
>>> a[1, 2]
     8
>>> a[-1, : 3]
     array ( [ 4, 6, 8] )
>>> a[0, 2 : 4]
     array ( [ 3, 4] )
```

```
>>> import numpy as np
>>> arr = np.arange(10)
>>> arr[5]
    5
>>> arr[5:8]
     array( [5, 6, 7] )
>>> x = arr[5:8]
>>> x
      array( [5, 6, 7] )
>>> arr[5:8] = 12
>>> arr
array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
>>> x
      array([12, 12, 12])
```

```
>>> arr
array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
>>> x
     array([12, 12, 12])
>>> x[1]=12345
>>> x
array([   12, 12345,    12])
>>> arr
array([ 0,  1,  2,  3,  4, 12, 12345, 12,  8,
9])
 #if there is a change in array slice , it
will reflect in original array
```

```
# how to change in array slice without
reflecting in original array
>>> b=arr[5:8].copy()
>>> b
    array([   12, 12345,    12])
>>> arr
    array([0,1,2,3,4,12,12345,12,8,9])
>>> b[1]=200
>>> b
    array([ 12, 200,  12])
>>> arr
    array([0,1,2,3,4,12,12345,12,8,9])
>>> b[:]=66     #bare slicing
>>> b
    array([66, 66, 66])
```

```
>>> import numpy as np
>>> a = np.array ( [ [1,2,3,4,5] ,
                     [4,6,8,10,12] ] )
>>> a[0,0]
    1
>>> a[1, 2]
     8
>>> a[-1, : 3]
     array ( [ 4, 6, 8] )
>>> a[0, 2 : 4]
     array ( [ 3, 4] )
```

```
>>> a[:2,:1]
        array( [ [1],
                 [4] ] )
>>> a[:2,1:]
        array( [ [ 2,  3,  4,  5],
                 [ 6,  8, 10, 12] ] )
>>> a[ : , :1 ]
        array( [ [1],
                 [4] ] )
>>> a[:2,1:]=0
>>> a
        array( [ [1, 0, 0, 0, 0],
                 [4, 0, 0, 0, 0] ] )
```

Objective: Explain the ufunc methods of numpy package in python
Outcome: Outline the ufunc methods of numpy package in python

Boolean Indexing

Topper	Sub1	Sub2	Sub3
Rahul	30	29	24
John	30	28	29
Kumar	30	27	28
Rahul	29	28	26
John	30	30	25

```
>>> import numpy as np
>>> topper=np.array( ['Rahul','John','Kumar','Rahul','John'] )
>>> topper
        array( ['Rahul', 'John', 'Kumar', 'Rahul', 'John'], dtype='<U5')
>>> marks = np.array( [ [30,29,24], [30,28,29],[30,27,28],[29,28,26], [30,30,25] ] )
>>> marks
array( [ [30, 29, 24],
       [30, 28, 29],
       [30, 27, 28],
       [29, 28, 26],
       [30, 30, 25] ] )
>>> marks[topper=='Rahul']
    array( [ [30, 29, 24],
             [29, 28, 26] ] )
>>> marks[topper=='Rahul',2:]
   array( [ [24],
            [26] ] )
>>> marks[topper=='Rahul',1:]
        array( [ [29, 24],
                 [28, 26] ] )
```

```
>>> marks[topper=='Rahul',2]
        array( [24, 26] )
>>> marks[topper=='Rahul',3]
        Error: index out of bound
>>> topper!='Rahul'
    array([False,  True,  True, False,  True])
>>> marks[~(topper=='Rahul')]
        array( [ [30, 28, 29],
                 [30, 27, 28],
                 [30, 30, 25] ] )
>>> cond=topper=='Rahul‘
>>> marks[~cond]    # ~ invert the condition
        array( [ [30, 28, 29],
                 [30, 27, 28],
                 [30, 30, 25] ] )
>>>cond1=(topper=='Rahul')|(topper=='kumar')
>>> cond1
        array([ True, False, False,  True, False])
>>> marks[cond1]
        array( [ [30, 29, 24],
                 [29, 28, 26] ] )
```

Memory layout of ndarray

NumPy internally stores data in a contiguous block of memory, independent of other built-in Python objects.
NumPy arrays also use much less memory than built-in Python sequences.

A contiguous array is just an array stored in an unbroken block of memory: to access the next value in the array, we just move to the next memory address.

Consider the 2D array arr = np.arange(12).reshape(4,3). It looks like this:

0 (0,0)	1 (0,1)	2 (0,2)
3 (1,0)	4 (1,1)	5 (1,2)
6 (2,0)	7 (2,1)	8 (2,2)
9 (3,0)	10 (3,1)	11 (3,2)

In the computer's memory, the values of arr are stored like this:

0	1	2	3	4	5	6	7	8	9	10	11

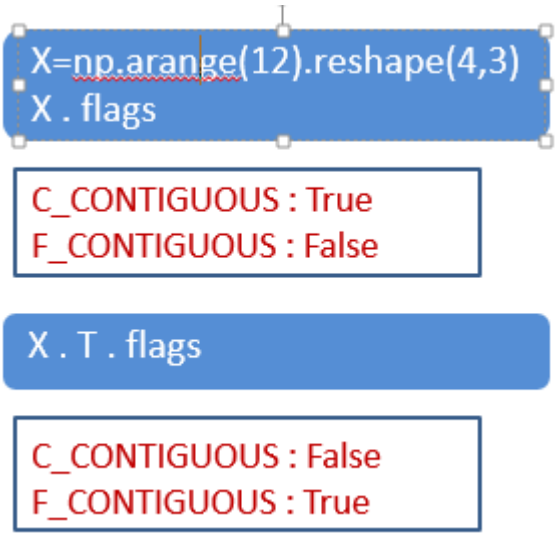
This means arr is a **C contiguous** array because the *rows* are stored as contiguous blocks of memory. The next memory address holds the next row value on that row. If we want to move down a column, we just need to jump over three blocks (e.g. to jump from 0 to 4 means we skip over 1,2 and 3).

Transposing the array with arr.T means that C contiguity is lost because adjacent row entries are no longer in adjacent memory addresses. However, arr.T is **Fortran contiguous** since the *columns* are in contiguous blocks of memory:

Performance-wise, accessing memory addresses which are next to each other is very often faster than accessing addresses which are more "spread out" (fetching a value from RAM could entail a number of neighbouring addresses being fetched and cached for the CPU.) This means that operations over contiguous arrays will often be quicker.

Each block represents a chunk of memory that is needed for representing the underlying array element. For example, each block could represent the 8 bytes needed to represent a double-precision floating point number.

Array takes 12 blocks of memory shown as a single, **contiguous segment.**
In other words, to move through computer memory sequentially, the last index is incremented first, followed by the second-to-last index and so forth. Some of the algorithms in NumPy that deal with N-dimensional arrays work best with this kind of data.



The flags attribute holds information about the memory layout of the array. The C_CONTIGUOUS field in the output indicates whether the array was a C-style array. This means that the indexing of this array is done like a C array. This is also called row-major indexing in the case of 2D arrays. This means that, when moving through the array, the row index is incremented first, and then the column index is incremented.

Universal Functions: Fast Element-wise Array Functions

A universal function, or *ufunc*, is a function that performs elementwise operations on data in ndarrays. You can think of them as fast vectorized wrappers for simple functions that take one or more scalar values and produce one or more scalar results.

Many ufuncs are simple elementwise transformations, like sqrt or exp:

```
arr = np.arange(10)

np.sqrt(arr)

np.exp(arr)
```

ufuncs stands for "Universal Functions" and they are NumPy functions that operates on the `ndarray` object.

Universal functions in Numpy are simple mathematical functions. It is just a term that we gave to mathematical functions in the Numpy library. Numpy provides various universal functions that cover a wide variety of operations.

These functions include standard trigonometric functions, functions for arithmetic operations, handling complex numbers, statistical functions, etc. Universal functions have various characteristics which are as follows-

- These functions operates on **ndarray** (N-dimensional array) i.e Numpy's array class.
- It performs fast element-wise array operations.
- It supports various features like array broadcasting, type casting etc.
- Numpy, universal functions are objects those belongs to numpy.ufunc class.
- Python functions can also be created as a universal function using **frompyfunc** library function.
- Some **ufuncs** are called automatically when the corresponding arithmetic operator is used on arrays. For example when addition of two array is performed element-wise using '+' operator then np.add() is called internally.

These are referred to as *unary* ufuncs. Others, such as add or maximum, take 2 arrays (thus, *binary* ufuncs) and return a single array as the result:

Why use ufuncs?

- ufuncs are used to implement *vectorization* in NumPy which is way faster than iterating over elements.
- They also provide broadcasting and additional methods like reduce, accumulate etc. that are very helpful for computation.
- ufuncs also take additional arguments, like:

- where boolean array or condition defining where the operations should take place.
- dtype defining the return type of elements.
- out output array where the return value should be copied.

What is Vectorization?

Converting iterative statements into a vector based operation is called vectorization.

It is faster as modern CPUs are optimized for such operations.

Ufunc functions (unary, binary)

Rounding Decimals

trunc() fix()	Remove the decimals, and return the float number closest to zero.
around()	increments preceding digit or decimal by 1 if >=5 else do nothing.
floor()	rounds off decimal to nearest lower integer.
ceil()	rounds off decimal to nearest upper integer.

```
>>> import numpy as np
>>> x = np.array ( [ -2, -1, 0, 1, 2 ] )
>>> abs(x)
    array( [2, 1, 0, 1, 2 ] )
>>> y = np.array ([-5.1686, 3.6667])
>>> abs(y)
     array([5.1686, 3.6667])
```

```
>>> np.trunc ([-5.1686, 3.6667])
    array([-5., 3.])
>>> np.around ([5.1686, 3.6667])
    array([5., 4.])
>>> np.floor ([5.1686, 3.6667])
    array([6., 3.])
>>> np.ceil ([5.1686, 3.6667])
    array([6., 4.])
```

Logs

log2()	perform log at the base 2.
log10()	perform log at the base 10.
log()	perform log at the base e.

```
>>> import numpy as np
>>> x= [1, 2, 3]
>>> np.exp(x)
    array[ 2.71828183,  7.3890561,
20.08553692]
>>> np.exp2(x)
     array([2. , 4. , 8.]
>>> np.log(x)
     array([0. , 0.69314718,
1.09861229])
```

```
>>> np.log2(x)
    array([0.     , 1.    , 1.5849625])
>>> np.log10(x)
     array([0. , 0.30103 , 0.47712125])
>>> np.linspace(0,10,5)
     array([ 0. , 2.5, 5. , 7.5, 10. ])
>>> np.linspace(0,10,3)
     array([ 0., 5., 10.])
>>> np.linspace(0, np.pi, 3)
     array([0. , 1.57079633, 3.14159265])
```

Trigonometric Functions

sin(), cos() and tan()	values in radians and produce the corresponding sin, cos and tan values.
arcsin(), arccos() and arctan()	Finding angles from values of sine, cos, tan. E.g. sin, cos and tan inverse
hypot()	Finding hypotenues using pythagoras theorem in NumPy.

Hyperbolic Functions

sinh(), cosh() and tanh()	take values in radians and produce the corresponding sinh, cosh and tanh values.
arcsinh(), arccosh() and arctanh()	Finding angles from values of hyperbolic sine, cos, tan. E.g. sinh, cosh and tanh inverse

Inverse Functions - unary

```
>>> import numpy as np
>>> np.sin(np.pi/2)
     1.0
>>> arr = np.array([np.pi/2, np.pi/3,np.pi/4])
>>> np.sin(arr)
     array([1. , 0.8660254 , 0.70710678])
>>> np.deg2rad([90, 180])
     array([1.57079633, 3.14159265])
>>> np.rad2deg([np.pi/2, np.pi])
     array([ 90., 180.])
```

```
>>> np.arcsin(1.0)
     1.5707963267948966
>>> arr = np.array([1, -1])
>>> np.arcsin(arr)
     array([ 1.57079633, -1.57079633])
>>> np.sinh(np.pi/2)
     2.3012989023072947
>>> base = 6
>>> height = 8
>>> np.hypot(base, height)
```

Summations

axis=1	NumPy will sum the numbers in each array.
cumsum()	Cummulative sum means partially adding the elements in array.

Products

prod()	To find the product of the elements in an array
axis=1	NumPy will return the product of each array.
cumprod()	Cummulative product means taking the product partially.

Differences

diff()	A discrete difference means subtracting two successive elements.

```
>>> import numpy as np
>>> arr1 = np.array([1, 2, 3])
>>> arr2 = np.array([4, 5, 6])
>>> arr1+arr2
    array([5, 7, 9])
>>> np.add(arr1, arr2)
    array([5, 7, 9])
>>> np.sum([arr1, arr2])
    21
 >>> np.sum([arr1, arr2], axis=1)
    array([ 6, 15])
```

```
>>> np.cumsum(arr1)
    array([ 6, 15])
>>> np.cumsum(arr1)
    array([1, 3, 6], dtype=int32)
>>> np.prod(arr1)
    6
>>> np.prod([arr1, arr2])
    720
>>> np.prod([arr1, arr2], axis=1)
    array([  6, 120])
```

Finding LCM (Lowest Common Multiple)

lcm()	The Lowest Common Multiple is the least number that is common multiple of both of the numbers.
reduce()	To find the Lowest Common Multiple of all values in an array

Finding GCD (Greatest Common Denominator)

gcd()	The GCD (Greatest Common Denominator), also known as HCF (Highest Common Factor) is the biggest number that is a common factor of both of the numbers.
reduce()	To find the Highest Common Factor of all values in an array

```
>>> import numpy as np
>>> arr2 = np.array([4, 5, 6])
>>> np.cumprod(arr2)
    array([  4,  20, 120], dtype=int32)
>>> np.diff(arr2)
    array([1, 1])
```

```
>>> num1 = 4, num2 = 6
>>> np.lcm(num1, num2)
12
>>> np.gcd(num1, num2)
2
>>> arr1 = np.array([1, 2, 3, 4])
>>> arr2 = np.array([3, 4, 5, 6])
>>> np.union1d(arr1, arr2)
array([1, 2, 3, 4, 5, 6])
>>> np.intersect1d(arr1, arr2)
array([3, 4])
```

Simple Arithmetic

add()	sums the content of two arrays, and return the results in a new array.
subtract()	subtracts the values from one array with the values from another array, and return the results in a new array.
multiply()	multiplies the values from one array with the values from another array, and return the results in a new array.
divide()	divides the values from one array with the values from another array, and return the results in a new array.
power()	rises the values from the first array to the power of the values of the second array, and return the results in a new array.
mod() remainder()	return the remainder of the values in the first array corresponding to the values in the second array, and return the results in a new array.
divmod()	return both the quotient and the the mod. The return value is two arrays, the first array contains the quotient and second array contains the mod.
absolute() abs()	do the same absolute operation element-wise but we should use absolute() to avoid confusion with python's inbuilt math.abs()

Ufunc Unary

Function	Description
`abs, fabs`	Compute the absolute value element-wise for integer, floating-point, or complex values
`sqrt`	Compute the square root of each element (equivalent to `arr ** 0.5`)
`square`	Compute the square of each element (equivalent to `arr ** 2`)
`exp`	Compute the exponent e^x of each element
`log, log10, log2, log1p`	Natural logarithm (base *e*), log base 10, log base 2, and log(1 + x), respectively
`sign`	Compute the sign of each element: 1 (positive), 0 (zero), or –1 (negative)
`ceil`	Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number)
`floor`	Compute the floor of each element (i.e., the largest integer less than or equal to each element)
`rint`	Round elements to the nearest integer, preserving the `dtype`
`modf`	Return fractional and integral parts of array as a separate array
`isnan`	Return boolean array indicating whether each value is `NaN` (Not a Number)
`isfinite, isinf`	Return boolean array indicating whether each element is finite (non-`inf`, non-`NaN`) or infinite, respectively
`cos, cosh, sin, sinh, tan, tanh`	Regular and hyperbolic trigonometric functions
`arccos, arccosh, arcsin, arcsinh, arctan, arctanh`	Inverse trigonometric functions
`logical_not`	Compute truth value of `not x` element-wise (equivalent to `~arr`).

Ufunc Binary

Function	Description
`add`	Add corresponding elements in arrays
`subtract`	Subtract elements in second array from first array
`multiply`	Multiply array elements
`divide, floor_divide`	Divide or floor divide (truncating the remainder)
`power`	Raise elements in first array to powers indicated in second array
`maximum, fmax`	Element-wise maximum; `fmax` ignores `NaN`
`minimum, fmin`	Element-wise minimum; `fmin` ignores `NaN`
`mod`	Element-wise modulus (remainder of division)
`copysign`	Copy sign of values in second argument to values in first argument

ndarray attributes

```
>>> import numpy as np
>>> a=np.array([1.,2.,3.])
>>> a.ndim
    1
>>> a.shape
    (3,)
>>> a.dtype
    dtype('float64')
```

```
>>> c=np.array([1,2,3],dtype=float)
>>> c
   array([1., 2., 3.])
>>> d=np.array([1,2,3], dtype=complex)
>>> d
    array([1.+0.j, 2.+0.j, 3.+0.j])
>>> d.real
    array([1., 2., 3.])
>>> d.imag
    array([0., 0., 0.])
```

```
>>> import numpy as np
>>> d=np.array([true,false])
Traceback (most recent call last):
NameError: name 'true' is not
defined

>>> d=np.array([True,False])
>>> d.dtype
   dtype('bool')
```

```
>>> d=np.array(['true','false'])
>>> d.dtype
    dtype('<U5')
>>> e=np.array(['welcome','bye'])
>>> e.dtype
    dtype('<U7')
>>> f=np.array(['lockdown','mask'])
>>> f.dtype
    dtype('<U8')
```

```
>>> import numpy as np
>>> f=np.array(['lockdown','mask'])

>>> f.flags
  C_CONTIGUOUS : True
  F_CONTIGUOUS : True
  OWNDATA : True
  WRITEABLE : True
  ALIGNED : True
  WRITEBACKIFCOPY : False
  UPDATEIFCOPY : False
```

C CONTIGUOUS (C) the data is in a single, C-style contiguous segment;

F CONTIGUOUS (F) the data is in a single, Fortran-style contiguous segment;

OWNDATA (O) the array owns the memory it uses or if it borrows it from another object;

WRITEABLE (W) the data area can be written to;

ALIGNED (A) the data and strides are aligned appropriately for the hardware;

UPDATEIFCOPY (U) this array is a copy of some other array, When this array is deallocated, the base array will be updated with the contents of this

```
>>> import numpy as np
>>> a=np.arange(5)
>>> a
    array([0, 1, 2, 3, 4])
>>> a.dtype
     dtype('int32')
>>> a.itemsize
      4
>>> a.nbytes
      20
```

```
>>> a.size
      5
>>> a.size*a.itemsize
      20
>>> b=np.arange(7,dtype=np.float)
>>> b
      array([0., 1., 2., 3., 4., 5., 6.])
>>> b.itemsize
     8
>>> b.nbytes
       56
```

```
>>> import numpy as np
>>> c=np.arange(7,dtype=complex)
>>> c
    array([0.+0.j, 1.+0.j, 2.+0.j, 3.+0.j,
4.+0.j, 5.+0.j, 6.+0.j])
>>> c.dtype
     dtype('complex128')
>>> c.nbytes
      112
```

```
>>> d=np.complex(7)
>>> d
    (7+0j)
>>> x=np.array([[10,20,30],[40,50,60]])
>>> x.flat[2]
     30
>>> x.flat[5]
     60
>>> x.flat[7]
     IndexError: index 7 is out of bounds
     for axis 0 with size 6
```

Array creator functions

Function	Description
array	Convert input data (list, tuple, array, or other sequence type) to an ndarray either by inferring a dtype or explicitly specifying a dtype; copies the input data by default
asarray	Convert input to ndarray, but do not copy if the input is already an ndarray
arange	Like the built-in range but returns an ndarray instead of a list
ones, ones_like	Produce an array of all 1s with the given shape and dtype; ones_like takes another array and produces a ones array of the same shape and dtype
zeros, zeros_like	Like ones and ones_like but producing arrays of 0s instead
empty, empty_like	Create new arrays by allocating new memory, but do not populate with any values like ones and zeros
full, full_like	Produce an array of the given shape and dtype with all values set to the indicated "fill value" full_like takes another array and produces a filled array of the same shape and dtype
eye, identity	Create a square N × N identity matrix (1s on the diagonal and 0s elsewhere)

```
>>> import numpy as np
>>> a=np.ones((3,3))
>>> a
    array([[1., 1., 1.],
           [1., 1., 1.],
           [1., 1., 1.]])
>>> np.zeros((2,2))
     array([[0., 0.],
            [0., 0.]])
```

```
>>> np.eye(3)
    array([[1., 0., 0.],
           [0., 1., 0.],
           [0., 0., 1.]])
>>> np.diag(np.array([1,2,3,4]))
     array([[1, 0, 0, 0],
            [0, 2, 0, 0],
            [0, 0, 3, 0],
            [0, 0, 0, 4]])
```

```
>>>import numpy as np
>>>np.full((3, 3), 6, dtype = 'float')
    array([[6., 6., 6.],
           [6., 6., 6.],
           [6., 6., 6.]])
>>> np.random.rand(4)
array([0.88995885, 0.38294717,
0.26089306, 0.44507376])
```

```
>>> np.empty((2,2))
    array([[0.88995885, 0.38294717],
           [0.26089306, 0.44507376]])
>>> b=np.empty((2,3))
>>> c=np.ones_like(b)
>>> c
    array([[1., 1., 1.],
           [1., 1., 1.]])
# c takes the size of b but fills the value
  with 1 floating values
```

ravel, flatten

ravel and flatten converts the ndarray to 1d array

ravel()

i) **Return only reference/view of original array**
ii) **If you modify the array you would notice that the value of original array also changes**
iii) **ravel is faster than flatten() as it does not occupy any memory**
iv) **ravel is a library-level function**

flatten()

i) **return copy of original array**
ii) **if you modify any value of this array value of original array is not affected**
iii) **flatten() is comparatively slower than ravel() as it occupies memory**
iv) **flatten is a method of an ndarray object**

```
>>> import numpy as np
>>> a=np.arange(12).reshape(2,6)
>>> a
    array([[ 0,  1,  2,  3,  4,  5],
           [ 6,  7,  8,  9, 10, 11]])
>>> b=np.arange(12).reshape(3,4)
>>> b
    array([[ 0,  1,  2,  3],
           [ 4,  5,  6,  7],
           [ 8,  9, 10, 11]])
```

```
>>> c=np.arange(12).reshape(4,3)
>>> c
       array([[ 0,  1,  2],
              [ 3,  4,  5],
              [ 6,  7,  8],
              [ 9, 10, 11]])
>>> c.ravel()
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> c.flatten()
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

Difference between ravel and flatten

```
>>> import numpy as np
>>> c=np.arange(12).reshape(4,3)
>>> c
    array([[ 0,  1,  2],
           [ 3,  4,  5],
           [ 6,  7,  8],
           [ 9, 10, 11]])
>>> d=c.ravel()
>>> d
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
>>> d[0]=30
# after d[0]=30 assigned
>>> d
array([30,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,
11])

>>> c
    array([[30,  1,  2],
           [ 3,  4,  5],
           [ 6,  7,  8],
           [ 9, 10, 11]])
# original array changes
```

```
>>> import numpy as np
>>> x=np.arange(12).reshape(4,3)
>>> x
    array([[ 0,  1,  2],
           [ 3,  4,  5],
           [ 6,  7,  8],
           [ 9, 10, 11]])
>>> y=x.flatten()
>>> y
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
>>> x[0]=30
# after x[0]=30 assigned
>>> x
array([30,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,
11])

>>> y
    array([[0,  1,  2],
           [ 3,  4,  5],
           [ 6,  7,  8],
           [ 9, 10, 11]])
# original array  does not change
```

diagonal

```
>>> import numpy as np
>>> a=np.arange(25).reshape(5,5)
>>> a
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
>>> a.diagonal()
      array([ 0,  6, 12, 18, 24])
```

```
>>> a.diagonal(1)
    array([ 1,  7, 13, 19])
>>> a.diagonal(-1)
    array([ 5, 11, 17, 23])
>>> a.diagonal(-3)
     array([15, 21])
>>> a.diagonal(3)
    array([3, 9])
>>> a.diagonal(6)
     array([], dtype=int32
```

transpose, min, max

```
>>> import numpy as np
>>> a=np.arange(25).reshape(5,5)
>>> a.T
array([[ 0,  5, 10, 15, 20],
       [ 1,  6, 11, 16, 21],
       [ 2,  7, 12, 17, 22],
       [ 3,  8, 13, 18, 23],
       [ 4,  9, 14, 19, 24]])
```

```
>>> a=np.array([11,20,13,11,14])
>>> np.max(a)
20
>>> np.min(a)
11
>>> np.argmax(a)
1
>>> np.argmin(a)
0
>>> np.where(a==np.min(a))
 #to return multiple minimum values if it is
present
(array([0, 3], dtype=int64),)
```

Copy

```
>>> import numpy as np
>>> x = np.array([[0,1], [2,3]])
```

	>>> y = x	>>> y = x . copy( )
	[[0 1] [2 3]]	[[0 1] [2 3]]
>>> y		
>>> x[0][1]=5 >>> x	[[0 5] [2 3]]	[[0 5] [2 3]]
>>> y	[[0 5] [2 3]]	[[0 1] [2 3]]
>>> x.fill(0) >>> x	[[0 0] [0 0]]	[[0 0] [0 0]]
>>> y	[[0 0] [0 0]]	[[0 1] [2 3]]

If y=x used
Change in x does not reflect in y

If y=x.copy( )
Change in x => changes in y

reshape, resize

Both reshape and resize change the shape of numpy array.

The difference is resize will affect the original array

```
>>> import numpy as np
>>> a=np.arange(12).reshape(3,4)
>>> a
    array([[ 0,  1,  2,  3],
           [ 4,  5,  6,  7],
           [ 8,  9, 10, 11]])
>>> a.reshape(4,3)
>>> a
     array([[ 0,  1,  2,  3],
            [ 4,  5,  6,  7],
            [ 8,  9, 10, 11]])
```

```
>>> a.resize((4,3))
>>> a
    array([[ 0,  1,  2],
           [ 3,  4,  5],
           [ 6,  7,  8],
           [ 9, 10, 11]])
# Original variable value will not change by using reshape
But resize will change the original variable shape
```

asarray, tolist

asarray( ) – is used to convert an input to an array

The input could be a list, tuple, ndarray etc.

tolist( ) – to convert an array to a list

```
>>> import numpy as np
>>> L1=[1,2,3]
>>> type(L1)
     <class 'list'>
>>> x=np.asarray(L1)
>>> x
    array([1, 2, 3])
>>> type(x)
    <class 'numpy.ndarray'>
```

```
>>> a=np.array([[10,20],[30,40]])
>>> type(a)
    <class 'numpy.ndarray'>
>>> b=a.tolist()
>>> type(b)
     <class 'list'>
>>> type(a)
     <class 'numpy.ndarray'>
```

tostring( ), tobytes( )

```
tostring() and tobytes() - used for creating a byte array from string

>>> import numpy as np        # Convert a 2d array into a string of bytes
>>> a = np.arange(4).reshape(2, 2)
>>> a
[[0 1]
 [2 3]]
>>> a.tostring()
>>> a.tobytes()
```

Merging arrays

axis=1 , along columns

axis=-, along rows

np.concatenate((a,b),axis=1)) concatenate a and b along columns

```
>>> import numpy as np
>>> a=np.array([1,2,3])
>>> b=np.array([5,6])
>>> np.concatenate([a,b])
    array([1, 2, 3, 5, 6])
>>> np.append(a,b)
    array([1, 2, 3, 5, 6])
>>> np.hstack([a,b])
    array([1, 2, 3, 5, 6])
>>> np.vstack([a,b])
    error dimension should match
```

```
>>> c=np.array( [8,9,10] )
>>> np.vstack([a,c])
    array([[ 1,  2,  3],
           [ 8,  9, 10]])
>>> np.stack((a,c),axis=0)
    array([[ 1,  2,  3],
           [ 8,  9, 10]])
>>> np.stack((a,c),axis=1)
    array([[ 1,  8],
           [ 2,  9],
           [ 3, 10]])
```

astype, dtype

astype() - changes the dtype of the given array object

```
>>> import numpy as np
>>> arr = np.array([10, 20, 30, 40, 50])
>>> arr
      [10 20 30 40 50]
>>> arr.dtype
      dtype('int32')
>>> arr = arr.astype('float64')
>>> arr.dtype
      dtype('float64')
```

```
>>> arr1 = arr.astype('complex')
>>> arr1.dtype
      dtype('complex128')
>>> arr1
      array([10.+0.j, 20.+0.j, 30.+0.j, 40.+0.j,
             50.+0.j])
>>> arr.dtype
      dtype('float64')
>>> arr
      array([10, 20, 30, 40, 50])
```

Negative( )

```
>>> import numpy as np
>>> a=10
>>> np.negative(a)
      -10
>>> b= np.array([[2, -7, 5], [-6, 2, 0]])
>>> b
      array([[ 2, -7,  5],
             [-6,  2,  0]])
>>> c =  np.negative(b)
>>> c
      array([[-2,  7, -5],
             [ 6, -2,  0]])
```

```
>>> np.where(c>0,2,c)
      array([[-2,  2, -5],
             [ 2, -2,  0]])
>>> np.where(c<0,2,c)
      array([[2, 7, 2],
             [6, 2, 0]])
>>> c
      array([[-2,  7, -5],
             [ 6, -2,  0]])
# original array not affected
```

NumPy Datatypes

Type	Type code	Description
int8, uint8	i1, u1	Signed and unsigned 8-bit (1 byte) integer types
int16, uint16	i2, u2	Signed and unsigned 16-bit integer types
int32, uint32	i4, u4	Signed and unsigned 32-bit integer types
int64, uint64	i8, u8	Signed and unsigned 64-bit integer types
float16	f2	Half-precision floating point
float32	f4 or f	Standard single-precision floating point; compatible with C float
float64	f8 or d	Standard double-precision floating point; compatible with C double and Python float object
float128	f16 or g	Extended-precision floating point
complex64, complex128, complex256	c8, c16, c32	Complex numbers represented by two 32, 64, or 128 floats, respectively
bool	?	Boolean type storing True and False values
object	0	Python object type; a value can be any Python object
string_	S	Fixed-length ASCII string type (1 byte per character); for example, to create a string dtype with length 10, use 'S10'
unicode_	U	Fixed-length Unicode type (number of bytes platform specific); same specification semantics as string_ (e.g., 'U10')

Itemset, put

itemset, put -insert /update scalar into an array

```
>>> import numpy as np
>>>x=np.array([[10,20,30],[40,50,60],
                [70,80,90]])
>>>x
    array([[10,  20,  30],
           [ 40,  50,  60],
           [ 70,  80,  90]])
>>> x[0][0]=100
>>> x
     array([[100,  20,  30],
            [ 40,  50,  60],
            [ 70,  80,  90]])
```

```
>>> x.itemset(1,200)
>>> x              #x[1] changes to 200
    array([[100,  200,  30],
           [ 40,  50,  60],
           [ 70,  80,  90]])

>>> x.itemset((2,2),900)
>>> x           # x[2][2] changes to 900
     array( [ [100, 200,  30],
              [ 40,  50,  60],
              [ 70,  80, 900] ] )
```

```
>>> import numpy as np
>>> x
      array( [ [100, 200,  30],
               [ 40,  50,  60],
               [ 70,  80, 900] ] )

>>> x.put(4,5)
>>> x
        array( [ [100, 200,  30],
                 [ 40,   5,  60],
                 [ 70,  80, 900] ] )
```

```
>>> x.put((2,6),100)          #multiple indices can be updated at the same time
>>> x
       array([[100, 200, 100],
              [ 40,   5,  60],
              [100,  80, 900]])

>>> >>> x.put((2,6),(1000,2000))
>>> x
        array([[ 100,  200, 1000],
               [  40,    5,   60],
               [2000,   80,  900]])
#multiple indices can be updated  with multiple corresponding values at a time
```

Compress

Compress(condition) - The method returns the elements (or sub-arrays along the given axis) of self where condition is true.

```
>>> import numpy as np
>>> x=np.array([0,1,2,3])
>>> x.compress(x > 2)
    array([3])
>>> x.compress(x > 1)
    array([2, 3])
```

```
>>> x.compress(x >= 1)
    array([1, 2, 3])
>>> x.compress(x < 2)
    array([0, 1])
```

tofile( )

tofile() – store the content in mentioned file

```
>>>  import numpy as np
>>>  d=np.array([10,20,30])
>>>  d.tofile("hi.txt",sep="  ",format="%d")
       # stores as 10 20 30
>>>  d.tofile("hi.txt",sep=":",format="%03d")
       # stores as 010:020:030
>>>  d.tofile("hi.txt",sep=":",format="%s")
       # stores as 10:20:30
>>>  d.tofile("hi.txt",sep=",,",format="%s")
       # stores as 10,,20,,30
```

If sep is the empty string, then write the file in binary mode.

If sep is any other string, write the array in simple text mode separating each element with the value of the sep string.

save, load

save - saving into .pny file

```
>>> import numpy as np
>>> a = np.arange(10)
>>>np.save('testfile', a)
    # the array is saved in the file
      testfile.npy
```

load - saving into .pny file

```
# the array is loaded into b
>>>b = np.load('testfile.npy')
>>>print("b is printed from testfile.npy")
>>> b
     [ 0 1 2 3 4 5 6 7 8 9]
```

squeeze, sort

```
#squeeze () will reduce dimension
 to level down
>>> import numpy as np
>>> x=np.array( [ [ [0,1],[2,3] ] ] )
>>> x.ndim
     3
>>> x.squeeze()
      array( [ [0, 1],
               [2, 3] ] )
```

```
>>> x.ndim
     3
>>> y=x.squeeze()
>>> y.ndim
     2
>>>a=np.array([6.7,3.4,1.1,2.1,2.3,5.6,7.8,2.2])
>>> a.sort()
>>> a
     array([1.1, 2.1, 2.2, 2.3, 3.4, 5.6, 6.7, 7.8])
```

Array calculations

input array[[10 20 30], [60 50 40], [90 80 70]]

```
>>> import numpy as np
>>>x = np.array([[10,20,30],[60,50,40],
[90,80,70]])
      [[10 20 30]
       [60 50 40]
       [90 80 70]]
>>>np.sum(x)
      450
>>>np.sum(x,  axis=0)
        #Sum of each column
      [160 150 140]
```

```
>>>np.sum(x,  axis=1)
         #Sum of each row
    [ 60 150 240]

>>>np.max(x)
       #maximum element of a matrix
    90

>>>np.min(x)
       #minimum element of a matrix
     10
```

```
>>> import numpy as np
>>>np.argmax(x)
    #index of maximum element in a
      matrix
       6

>>>np.argmin(x)
      #index of minimum element in a
       matrix
       0
```

```
>>>np.argmax(x.axis=1)
      #index of maximum element in
      each row of a matrix 30 60 90
       [2 0 0]

>>>np.argmin(x.axis=1)
      #index of minimum element in
     each column of a matrix 10 40 70
        [0 2 2]
```

```
>>> import numpy as np
>>>np.argmax(x.axis=0)
     #index of maximum element in
   each column of a matrix 90 80 70
     [2 2 2]

>>>np.argmin(x.axis=0)
    #index of minimum element in
  each column of a matrix 10 20 30
     [0 0 0]
```

```
>>> np.swapaxes(x,0,1)
   #interchange two axis of an array
    array([[10, 60, 90],
           [20, 50, 80],
           [30, 40, 70]])

# np.swapaxes(x,0,1)  gives same as
above , and in both cases original array
will not change
```

```
>>> import numpy as np
>>>A=np.array([[1,2],[3,4]])
>>>B=np.array([[2,2],[3,3]])
     matrix A
          [[1 2]
           [3 4]]
     matrix B
          [[2 2]
           [3 3]]
```

```
>>>np.add(A,B)
    #element wise addition of matrix
     [[3 4]
     [6 7]]
>>>np.subtract(A,B)
    #element wise subtraction
     [[-1  0]
     [ 0  1]]
>>>np.divide(A,B)
     #element wise division of matrix
     [[0.5       1.       ]
     [1.        1.33333333]
```

Matrix A

[[2 2]

[3 4]]

Matrix B

[[1 2]

[3 3]]

```
>>> import numpy as np
>>>np.multiply(A,B)
     #element wise multiplication of
     matrix
     [[ 2  4]
     [ 9 12]]

>>>np.dot(A,B)
    #product of matrices
     [[ 8  8]
     [18 18]]
```

```
>>>np.sqrt(A)
     [[1.        1.41421356]
     [1.73205081 2.        ]]

>>>A.T
     #transpose of given matrix
     [[1 3]
     [2 4]]
```

set operations

```
>>> import numpy as np
>>> array1 = np.array([0, 10, 20, 40, 60, 80])
>>> array2 =np.array ([10, 30, 40, 50, 70])
>>> print("Union of 2 arrays")
>>> np.union1d(array1, array2)

>>> print("Intersection of 2 arrays")
>>> np.intersect1d(array1, array2)

print("Difference of first array to second")
>>> np.setdiff1d(array1, array2)

print("Difference of second array to first")
>>> np.setdiff1d(array1, array2)
```

Output:

```
Union of 2 arrays
array( [ 0 10 20 30 40 50 60 70 80] )

Intersection of 2 arrays
 array( [ 10,40] )

Difference of first array to second"
array( [ 0 20 60 80] )

Difference of first array to second"
array( [ 30 50 70] )
```

```
>>> #Extract all odd numbers from arr
>>> arr = np.arange(10)
>>> arr
    array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
>>> arr [ arr % 2 == 1]
    array([1, 3, 5, 7, 9])

>>> #replace odd numbers with -1
>>> arr[arr % 2 == 1] = -1
>>> arr

array([ 0, -1, 2, -1, 4, -1, 6, -1, 8, -1])
```

```
#replace odd terms with -1 in new array
withour affecting original array

>>> arr = np.arange(10)

>>> out = np.where(arr % 2 == 1, -1, arr)

>>> print("Result: ",out)
>>> print("Original: ",arr)

Result: [ 0 -1 2 -1 4 -1 6 -1 8 -1]
Original: [0 1 2 3 4 5 6 7 8 9]
```

Replace the negative values in numpy array with 0

```
import numpy as np
x = np.array([-1, -4, 0, 2, 3, 4, 5, -6])
print("Original array:")
print(x)

print("Replace the negative values with 0:")
x[x < 0] = 0
print(x)
```

Output:

```
Original array:
[-1 -4 0 2 3 4 5 -6]

Replace the negative values with 0:
[0 0 0 2 3 4 5 0]
```

NumPy program to test element-wise for NaN of a given array

```
import numpy as np
a = np.array([1, 0, np.nan, np.inf])
print("Original array")
print(a)

print("Test element-wise for NaN:")
print( np.isnan(a) )
```

Output:

```
Original array
[ 1. 0. nan inf]

Test element-wise for NaN:
[False False True False]
```

NumPy program to test element-wise for positive or negative infinity

```
import numpy as np
a = np.array([1, 0, np.nan, np.inf])
print("Original array")
print(a)

print("Test element-wise for positive or
negative infinity :")
print( np.isinf(a) )
```

Output:

```
Original array
[ 1. 0. nan inf]

Test element-wise for positive or
negative infinity:
[False False False True]
```

NumPy program to test a given array element-wise for finiteness (not infinity or not a Number)

```
import numpy as np
a = np.array([1, 0, np.nan, np.inf])
print("Original array")
print(a)

print("Test element-wise for finiteness :")
print( np.isfinite(a) )
```

Output:

```
Original array
[ 1. 0. nan inf]


Test element-wise for finiteness:
[True True False False]
```

NumPy program to test element-wise for complex number, real number of a given array. Also test whether a given number is a scalar type or not.

```
>>> import numpy as np
>>> a = np.array([1+1j, 1+0j, 4.5, 3, 2, 2j])
>>> print("Original array")
>>> print(a)
>>> print("Checking for complex number:")
>>> print(np.iscomplex(a))
>>> print("Checking for real number:")
>>> print(np.isreal(a))
>>> print("Checking for scalar type:")
>>> print(np.isscalar(3.1))
>>> print(np.isscalar([3.1]))
```

Output:

```
Original array
[ 1.0+1.j 1.0+0.j 4.5+0.j 3.0+0.j
2.0+0.j 0.0+2.j]
Checking for complex number:
[ True False False False False True]
Checking for real number:
[False True True True True False]
Checking for scalar type:
True
False
```

Machine Learning

Introduction to Machine learning

In simple words, we can say that machine learning is the competency of the software to perform a single or series of tasks intelligently without being programmed for those activities.

This is part of **Artificial Intelligence**. Normally, the software behaves the way the programmer programmed it; while machine learning is going one step further by making the software capable of accomplishing intended tasks by using statistical analysis and predictive analytics techniques.

How Machine Learning Works?
Machine Learning is a technique which works intelligently by using some complex algorithms and set of predefined rules. It uses the past data to read the patterns and then based on the analysis it generates the relevant data or performs the intended task abiding the defined rules and algorithms.

As an example, whenever we typed in something on the search bar, the search engines uses this machine learning technique to display the related contents. It intelligently reads the vast data on the web, indexes, ranking, and based on the defined rules and algorithm it displays the search results.

A major focus of machine learning research is to automatically learn to recognize complex patterns and make intelligent decisions based on data.

Examples of Machine Learning

- **Google and other search engines** providers use machine learning techniques to populate the relevant data. Google recommends so many keywords when we try to search in search box. Ranking web pages.
- **Diagnosis of Deadly Diseases**
  The machine learning technique is being used in the field of healthcare domain. The software helps to detect the deadly diseases such as cancers by reading the past data of the patients and matching that with the symptoms defined in the algorithms.

- **Face Reorganization and Tagging Features**

In social media accounts, the software suggests the name of your friends present in the images for tagging. This is possible by using the machine learning techniques. It recognizes the faces

of the people present in the images by using the past data of those users and suggests you the accurate names.

- **Detection of Spam Emails**
  The software model identifies the email messages based on the nature of the content it carries and decides whether it is a spam or a genuine email to be allowed to the inbox. The software reads the past data feeds and the set of rules and algorithms to identify the nature of the emails.

- **Displaying the Related Advertisements**
  While reading an online article or seeing images or videos, you may have noticed that the related advertisements from various advertisers to buy or see a product keep on coming on the side bars and elsewhere on the web pages. That is again an example of machine learning. The software recognizes the data used in the articles and shows the related products which the users may be interested in buying or seeing.

- **Speech Recognition**
  To recognize spoken words (Subtitle in youtube videos, individual speakers, vocabularies, microphone characteristics, background noise etc).

Some successful applications of machine learning are,

- Learning to recognize spoken words.
- Learning to drive an autonomous vehicle.
- Learning to classify new astronomical structures.
- Learning to play world-class games.

Some more examples of machine learning tasks include:

- Identifying the zip code from handwritten digits on an envelope
- Determining whether a tumor is benign based on a medical image
- Detecting fraudulent activity in credit card transactions
- Identifying topics in a set of blog posts

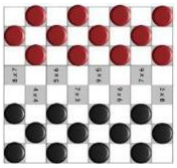
- Segmenting customers into groups with similar preferences
- Detecting abnormal access patterns to a website

Definition: A field of study that gives the computers the ability to learn without being explicitly programmed. (Arthus Samuel)

Definition: A computer program is said to **learn** from experiences E with respect to some class of tasks T and performance P, if its performance at tasks in T, as measured by P, improves with experience E (Tom Mitchell)

For example, a computer program that learns to play checkers might improve its performance as measured by its ability to win at the class of tasks involving playing checkers games, through experience obtained by playing games against itself. In general, to have a well-defined learning problem, we must identity these three features:

- the class of tasks represented as T,
- the measure of performance to be improved represented as P,
- and the source of experience represented as E.

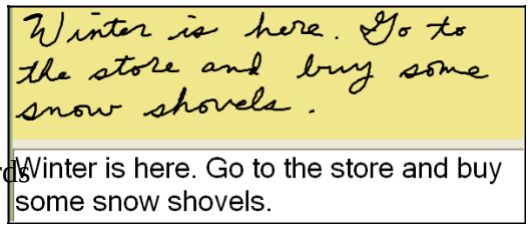


A checkers learning problem

- *Task T*: playing checkers
- *Performance measure P*: percent of games won against opponents
- *Training experience E*: playing practice games against itself

We can specify many learning problems in this fashion, such as learning to recognize handwritten words, or learning to drive a robotic automobile autonomously.

A handwriting recognition learning problem

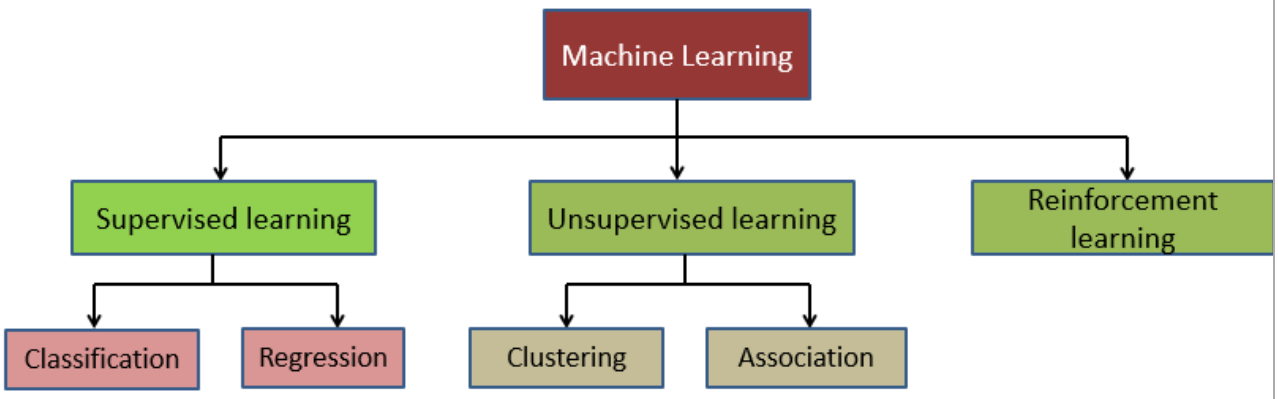


- *Task T*: recognizing and classifying handwritten words within images
- *Performance measure P*: percent of words correctly classified
- *Training experience E*: a database of handwritten words with given classifications

A robot driving learning problem

- T: driving on public four-lane highways using vision sensors
- P: average distance traveled before an error (as judged by human overseer)
- E: a sequence of images and steering commands recorded by observing a human driver

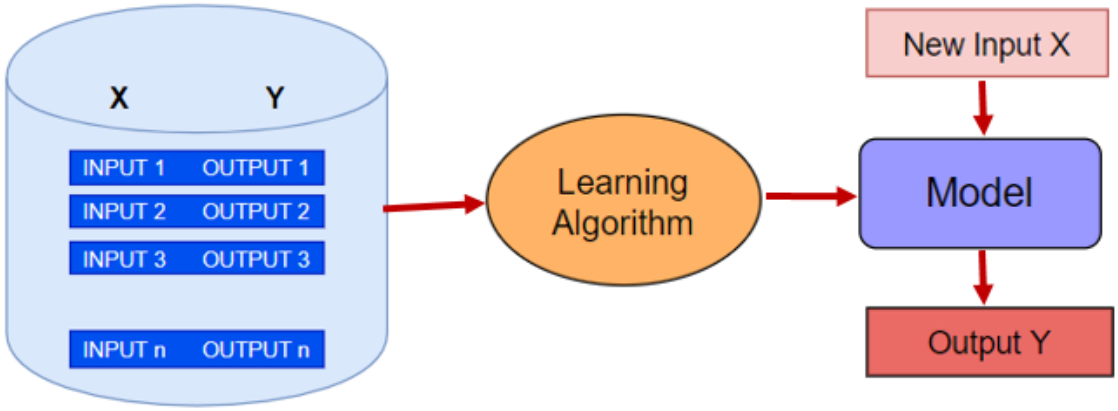
Learning can happen in 3 methods



Supervised Learning

Supervised learning is a machine learning method in which models are trained using labelled data
Models need to find the mapping function to map the input variable (X) with the output variable (Y)

$$Y=f(x)$$



Algorithm:

Given:

- **a set of input features $X_1,\ldots\ldots,X_n$**
- **a target feature Y**
- **a set of training examples where the values for the input features and the target features are given for each example**
- **a new example, where only the values for the input features are given**

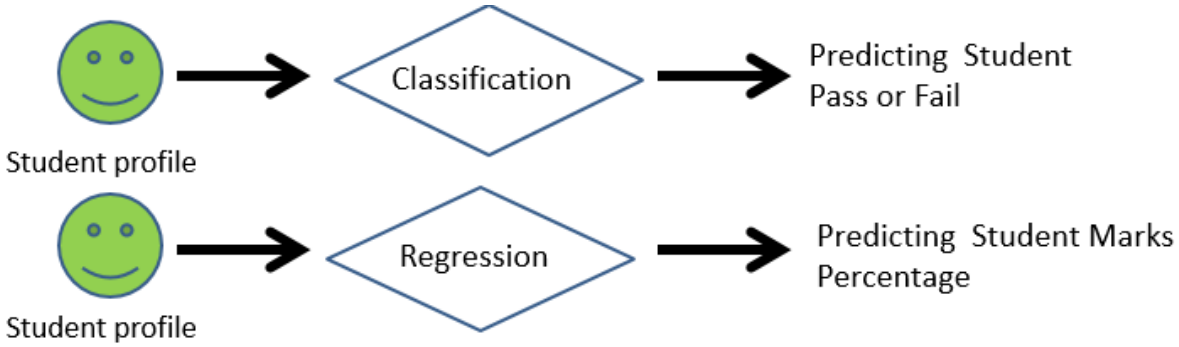
Predict the values for the target features for the new example

- **Classification when Y is discrete**
- **Regression when Y is continuous**

Types of Supervised Learning

- **Regression**
- **Classification**

Regression – The dependent variables are numerical
Classification – The dependent variables are categorical



PARAMETER	CLASSIFICATION	REGRESSION
Basic	Mapping function is used for mapping of values to predefined classes	Mapping function is used for mapping of values to continuous output
Involves prediction of	Discrete values	Continuous values
Nature of the prediction	Unordered	Ordered
Method of calculation	By measuring accuracy	By measurement of root mean square error
Example Algorithms	Decision tree, logistic regression, etc	Regression tree (Random Forest), Linear Regression, etc.

The output feature Y can be discrete or continuous.
Discrete – classification

1. Tomorrow rains or not
2. By giving different symptoms of patient and predicts whether the patient has particular disease or not

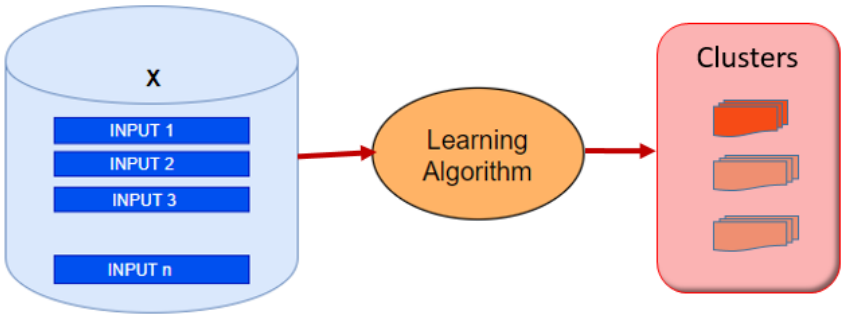
Continuous- regression

1. Predicting price of a house, which is a real valued number, by givining different features like location, built in area, no. of bedrooms etc.

Credit – scoring

Unsupervised Learning

Unsupervised learning is a machine learning method in which patterns inferred from unlabelled data
Goal is to find the structure and patterns from the input data by its own



In supervised learning, the aim is to learn a mapping from the input to an output whose correct values are provided by a supervisor. In unsupervised learning, there is no such supervisor and we only have input data. The aim is to find the regularities in the input.

Types of Supervised Learning

- Clustering
- Association

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group than those in other groups.

Association rule mining finds interesting associations and relationships among large sets of data items. This rule shows how frequently a itemset occurs in a transaction. A typical example is Market Based Analysis

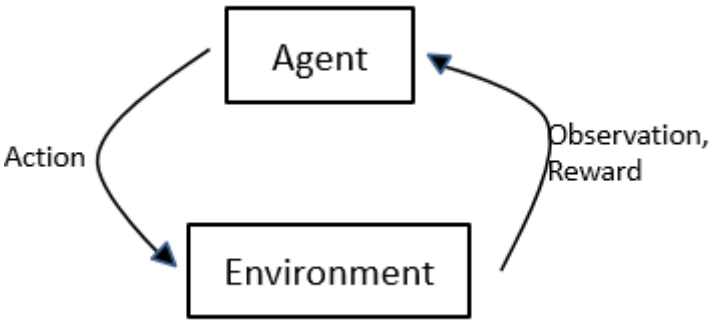
Reinforcement Learning

Supervised Learning: Immediate feedback (labels provided for every input)
Unsupervised Learning: No feedback (No labels provided)
Reinforcement Learning : Delayed scalar feedback (a number called reward)

- RL deals with agents that must sense and act upon their environment
- Every action has some impact in the environment, and the environment provides rewards that guides the learning algorithm

Examples:

- Learning to walk/ fly/ ride a vehicle
- Robot-soccer
- Share Investing
- Scheduling
- AlphaFo / Super Mario

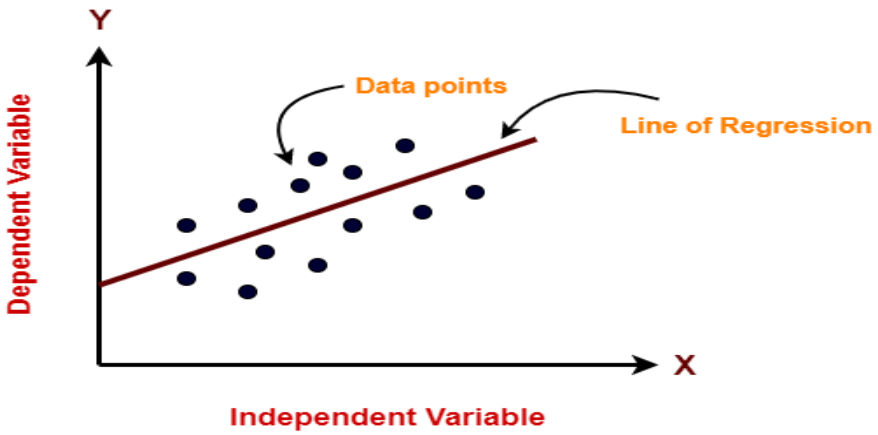


Linear Regression Introduction

In Machine Learning,

- Linear Regression is a supervised machine learning algorithm
- It tries to find out the best linear relationship that describes the data you have
- It assumes that there exists a linear relationship between a dependent variable and independent variable(s)
- The value of the dependent variable of a linear regression model is a continuous value i.e. real numbers

Linear regression model represents the linear relationship between a dependent variable and independent variable(s) via a sloped straight line.



The sloped straight line representing the linear relationship that fits the given data best is called as a regression line. It is also called as best fit line.

Examples

- . Predicting the price of a used car

  Dependent variable - Output – car price

  Independent variables - Inputs - car attributes

  - brand, year, engine capacity, mileage etc

- Agricultural decision problem

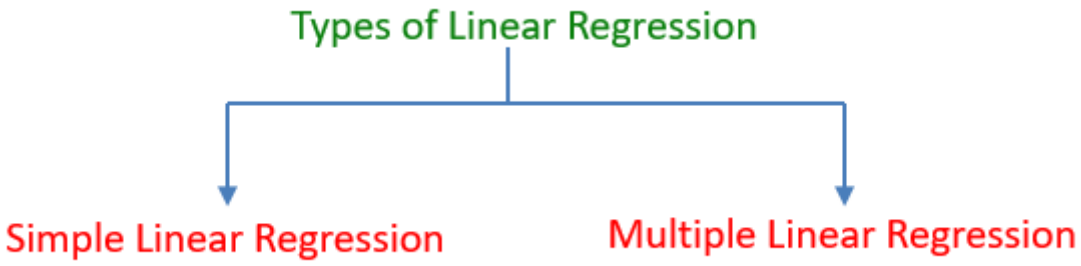
  Dependent variable - crop yield in bushels per acre

  Independent variables-

  - amount of fertilizers and pesticides applied per acre
  - amounts of rainfall
  - average temperatures during the months of the growing season

Types of Linear Regression

Based on the number of independent variables, there are two types of linear regression



Simple Linear Regression

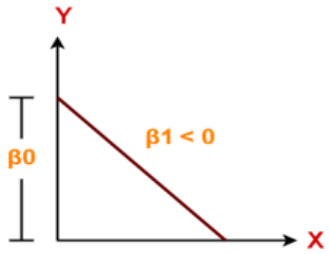
In simple linear regression, the dependent variable depends only on a single independent variable.
For simple linear regression, the form of the model is-

$$Y = \beta_0 + \beta_1 X$$

Y is a dependent variable
X is an independent variable
β_0 and β_1 are the regression coefficients
β_0 is the intercept or the bias that fixes the offset to a line
β_1 is the slope or weight that specifies the factor by which X has an impact on Y

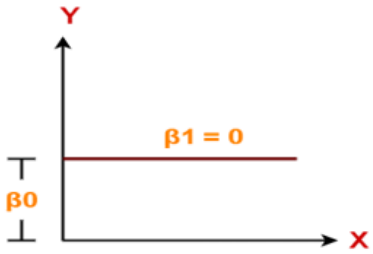
There are following 3 possible cases

Case-01: $\beta_1 < 0$



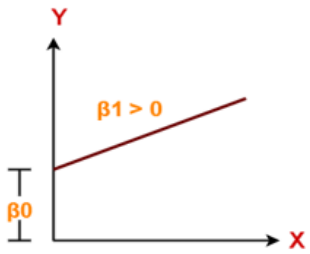
X has negative impact on Y
If X increases, Y will decrease
and vice-versa

Case-02: $\beta_1 = 0$



X has no impact on Y
If X changes, no change in Y

Case-03: $\beta_1 > 0$



X has positive impact on Y
If X increases, Y will increase
and vice-versa.

34.2 Linear Regression – Case Study

Objective: Understand Linear Regression with a Case Study
Outcome: Outline Linear Regression with a Case Study

Problem Statement

Five randomly selected students took a math aptitude test before they began their statistics cours
The Statistics Department has three questions.

- What linear regression equation best predicts statistics performance, based on mat aptitude scores?
- If a student made an 80 on the aptitude test, what grade would we expect her to make in statistics?
- How well does the regression equation fit the data?

How to find the Regression Equation

- x_i column shows scores on the aptitude test
- y_i column shows statistics grades
- The last two columns show deviations scores - the difference between the student' score and the average score on each test.

Student	x_i	y_i	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})^2$	$(y_i - \bar{y})^2$	$(x_i - \bar{x})^2(y_i - \bar{y})^2$
1	95	85	17	8	289	64	136
2	85	95	7	18	49	324	126
3	80	70	2	-7	4	49	-14
4	70	65	-8	-12	64	144	96
5	60	70	-18	-7	324	49	126
Sum	390	385			730	630	470
Mean	78	77					

The regression equation is a linear equation of the form: $\hat{y} = b_0 + b_1 x$

Regression coefficient (b_1):

$$b_1 = \Sigma [(x_i - x)(y_i - y)] / \Sigma [(x_i - x)^2]$$
$$b_1 = 470/730$$
$$b_1 = 0.644$$

Regression slope (b_0):

$$b_0 = y - b_1 * x$$
$$b_0 = 77 - (0.644)(78)$$
$$b_0 = 26.768$$

Therefore, the regression equation is: $\hat{y} = 26.768 + 0.644x$

If a student made an 80 on the aptitude test, the estimated statistics grade ($\hat{y}$) would be:

$$\hat{y} = b_0 + b_1 x$$
$$\hat{y} = 26.768 + 0.644x = 26.768 + 0.644 * 80$$
$$\hat{y} = 26.768 + 51.52 = 78.288 \quad \text{Grade}$$

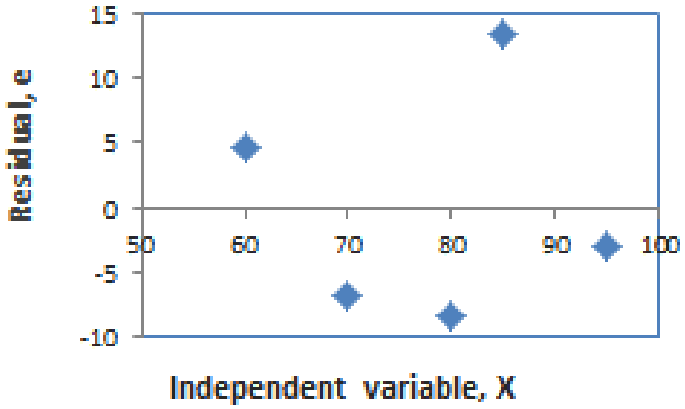
The difference between the observed value of the dependent variable (y) and the predicted value ($\hat{y}$)
is called the **residual** (e).

Each data point has one residual.

Residual = Observed value - Predicted value
$e = y - \hat{y}$

x	y	ŷ	e
60	70	65.411	4.589
70	65	71.849	-6.849
80	70	78.288	-8.288
85	95	81.507	13.493
95	85	87.945	-2.945

The residual plot shows a fairly random pattern - the first residual is positive, the next two a negative, the fourth is positive, and the last residual is negative. This random pattern indicates th a linear model provides a decent fit to the data.



Multiple Linear Regression

- In multiple linear regression, the dependent variable depends on more than one independent variables.
- For multiple linear regression, the form of the model is-

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_3 X_3 + \ldots\ldots + \beta_n X_n$$

Here,

Y is a dependent variable

$X_1, X_2, \ldots, X_n$ are independent variables

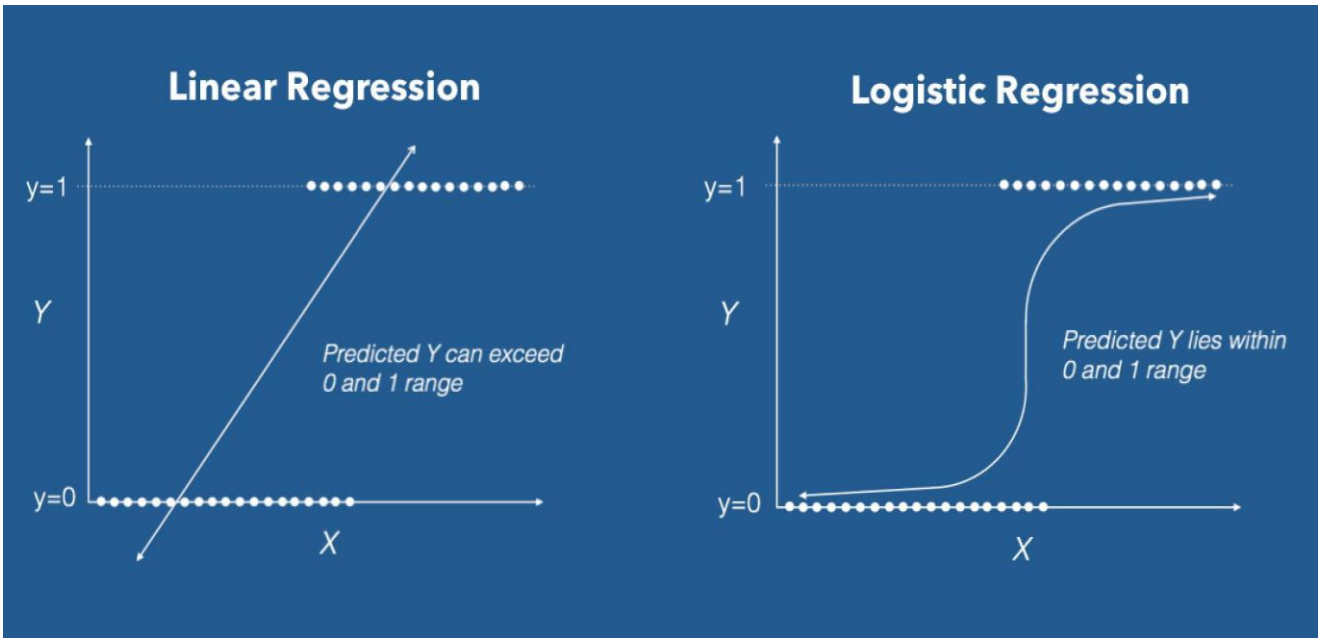
$\beta_0, \beta_1, \ldots, \beta_n$ are the regression coefficients

β_j $(1<=j<=n)$ is the slope or weight that specifies the factor by which X_j has an impact on Y

Limitation of Linear Regression

- When the response variable has only 2 possible values, it is desirable to have a model that predicts the value either as 0 or 1 or as a probability score that ranges between 0 and 1.
- Linear regression does *not* have this capability. Because, If you use linear regression to model a binary response variable, the resulting model may not restrict the predicted Y values within 0 and 1.

This is where logistic regression comes into play. In logistic regression, you get a probability score that reflects the probability of the occurrence of the event.



Problem statement 1:

Whether employee will be promoted or not

Problem statement 2:

After hike what will be his salary

Variables:

Education Qualification

Number of years in a current position
Total number of years experience
Age
Salary

Linear Regression: Problem statement 2

Logistic Regression: Problem statement 1

Examples

- Which category of the products is most interesting to this customer?
- Is this movie a romantic, comedy , documentary or thriller?
- Is this review written by a customer or a robot?
- Will the customer buy this product?
- Is this email spam or not spam?
- Whether loan can be approved or not for this applicant?

	Whether employee will be promoted or not	After hike what will be his salary
Type of Learning	**Classification**	**Regression**
Independent variables	Education Qualification No. of years in current position Total years of experience Age Salary	Education Qualification No. of years in current position Total years of experience Age
Dependant variables	Promoted or not	Salary

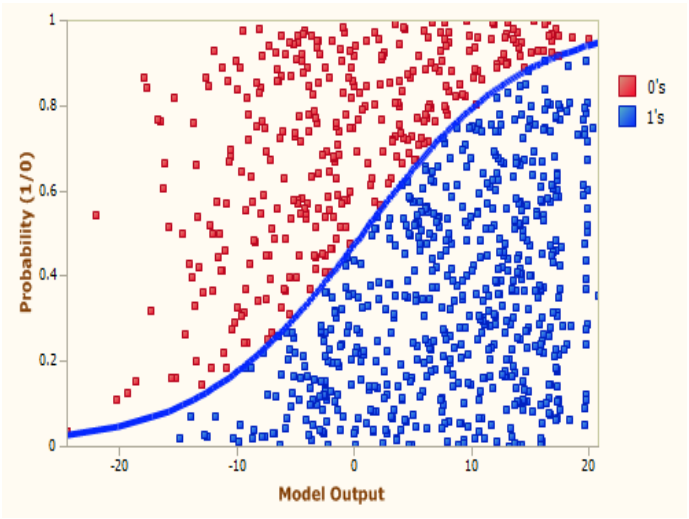
We should be able to express how inputs and output are related mathematically.
That mathematical representation is called as **model.**

Nature of the dependent variables differentiates regression and classification problem, where both are under supervised learning

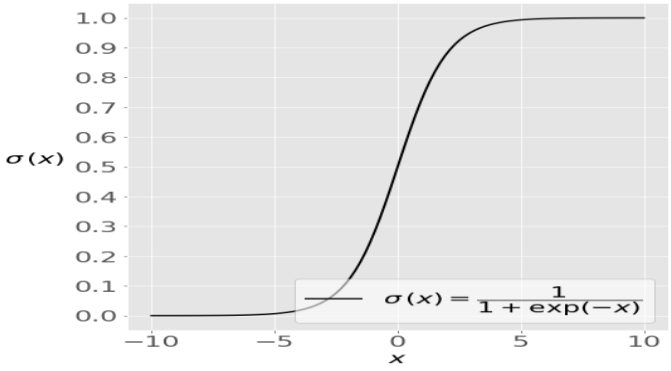
LOGISTIC REGRESSION

- A fundamental classification technique
- Predicts the probability of a categorical dependent variable

- The dependent variable is a binary variable that contains data coded as 1 (yes, success, etc.) or 0 (no, failure, etc.).
- In other words, the logistic regression model predicts P(Y=1) as a function of X.
- Somewhat similar to polynomial and linear regression.
- It is also known as a generalized linear model (GLM).
- Logistic regression is a transformation of the linear regression model that allows us to probabilistically model binary variables.



Logistic Regression

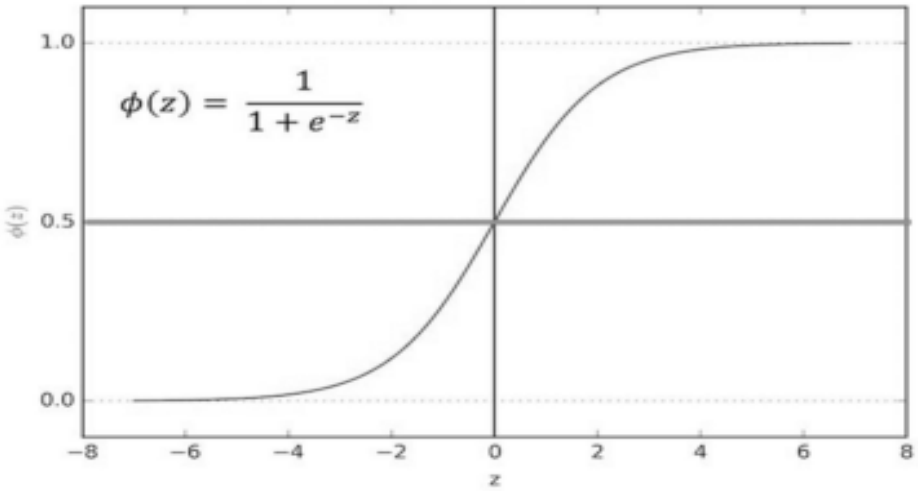


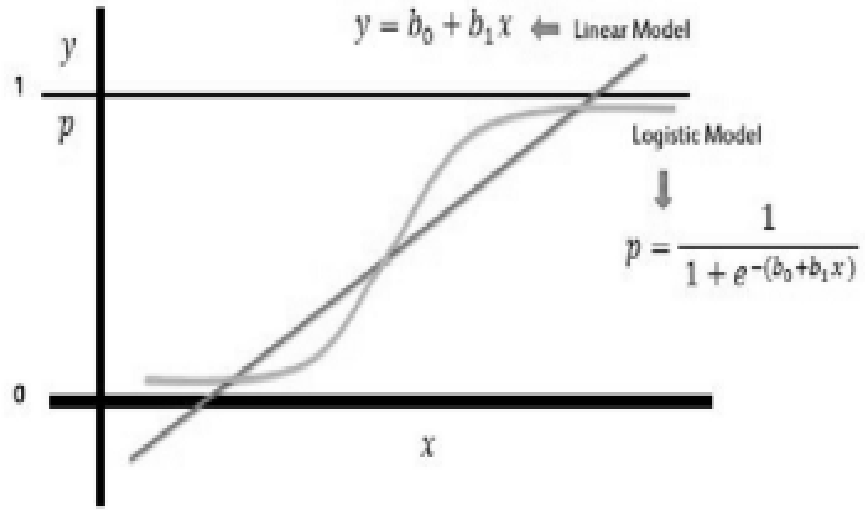
Sigmoid function
(or S-shaped curve) of some variable x

The sigmoid function also known as the logistic function is going to be the key to using logistic regression to perform classification.

The sigmoid function takes in any value and outputs it to be between 0 and 1.

This means we can take our linear regression solution and place it into the sigmoid function and it looks something like below





How threshold=0.5

The sigmoid function $\vartheta(z)$ or p:

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

Here are some other properties about sigmoid function or $\vartheta(z)$ or p:

1. When z=0, $\vartheta = 0.5$

$$\theta(z) = \frac{1}{1+e^{0}} = \frac{1}{1+1} = .5$$

2. When z is very large, θ is approximately 1

$$\theta(z) \approx \frac{1}{1+0} = 1$$

3. When z is very small/negative , θ is approximately

$$\theta(z) \approx \frac{1}{1+\infty} = 0$$

Model Representation

- *Let set of independent variables* $\boldsymbol{x} = (x_1, \ldots, x_r)$,
  where r *is the number of predictors (or inputs),*
- *start with the known values of the predictors* $\boldsymbol{x}_i$ *and the corresponding actual response (or output)* y_i *for each observation* $i = 1, \ldots, n$.
- *y - the target variable / dependent variable / output variable*

- *In a classification problem,*
  Y can take only discrete values for given set of features(or inputs), X.
  (output variable which takes the values 0 or 1)
- *A key difference from linear regression is that the output value being modeled is a binary values (0 or 1) rather than a numeric value.*

- *Goal: Goal is to find the* ***logistic regression function*** $(\boldsymbol{x})$ *such that the* ***predicted responses*** $(\boldsymbol{x}_i)$ *are as close as possible to the* ***actual response*** y_i *for each observation* $i = 1, \ldots, n$.

- *Logistic regression is a linear classifier,*
  So we will use a linear function
  $(\boldsymbol{x}) = b_0 + b_1x_1 + \cdots + b_rx_r$, *also called the* ***logit***.
  $b_0, b_1, \ldots, b_r$ - ***estimators*** / ***predicted weights*** / ***coefficients***

- *The logistic regression function* $(\boldsymbol{x})$ *is the sigmoid function of* $(\boldsymbol{x})$: $(\boldsymbol{x}) = 1 / (1 + \exp(-f(\boldsymbol{x}))$.
  As such, it's often close to either 0 or 1.

The process of calculating the best weights using available observations is called ***model training*** *or* ***fitting***.

To get the best weights,
usually we use maximize the ***log-likelihood function (LLF)*** *for all observations* $i = 1, \ldots, n$.

*This method is called the **maximum likelihood estimation** and is represented by the equatic*

Types of Logistic Regression

- *Binary Logistic Regression: The target variable has only two possible outcomes such as Spam or Not Spam, Cancer or No Cancer.*
- *Multinomial Logistic Regression: The target variable has three or more nominal categories such as predicting the type of Wine.*
- *Ordinal Logistic Regression: The target variable has three or more ordinal categories su as restaurant or product rating from 1 to 5.*

Linear Vs Logistic

LINEAR REGRESSION	LOGISTIC REGRESSION
A linear approach that models the relationship between a dependent variable and one or more independent variables	A statistical model that predicts the probability of an outcome that can only have two values
Used to solve regression problems	Used to solve classification problems (binary classification)
Estimates the dependent variable when there is a change in the independent variable	Calculates the possibility of an event occurring
Output value is continuous	Output value is discrete
Uses a straight line	Uses an S curve or sigmoid function
Ex : predicting the GDP of a country, predicting product price, predicting the house selling price, score prediction	Ex : predicting whether an email is a spam or not, predicting whether the credit card transaction is fraud or not, predicting whether the customer will take loan or not

Clustering Introduction

- **Clustering** is the classification of objects into different groups, or more precisely, the partitioning of a data set into subsets (clusters), so that the data in each subset (ideally) share some common trait - often according to some defined distance measure

- **Example**

  Web document Search: A web search engine often returns thousands of pages in response to a broad query, making it difficult for users to browse to identify relevant information.

  Clustering methods can be used to automatically group the retrieved documents into a list of meaningful categories**.**

In Clustering , training will be done without any output.
And grouping will be done with out any predefined labels

Clustering is an example of unsupervised learning.

The process of grouping a set of physical or abstract objects into classes of *similar* objects is called **clustering.**

Although classification is an effective means for distinguishing groups or classes of objects, it requires the often costly collection and labeling of a large set of training tuples or patterns, which the classifier uses to model each group.

Clustering can also be used for **outlier detection**.

Applications of outlier detection include the detection of credit card fraud and the monitoring of criminal activities in electronic commerce. For example, exceptional cases in credit card transactions, such as very expensive and frequent purchases, may be of interest as possible fraudulent activity.

Applications of Clustering

- Recommendation engines
- Market segmentation
- Social network analysis
- Search result grouping
- Medical imaging
- Image segmentation
- Anomaly detection

Types of Clustering

- Clustering falls under unsupervised learning
- They are different types of clustering methods, including:
  - Partitioning methods
  - Hierarchical **clustering**
  - Fuzzy **clustering**
  - Density-based **clustering**
  - Model-based **clustering**
- K-means is under Partitioning method of clustering

Analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters).

K-MEANS CLUSTERING

K-means clustering is an algorithm to classify or to group the objects based on attributes/features into K number of group.

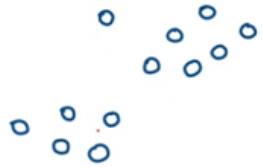
K is positive integer number.

- The k-means algorithm is an algorithm to cluster n objects based on attributes into k partitions, where $k < n$.

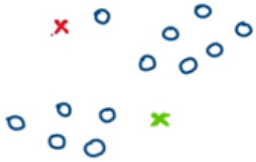
- An algorithm for partitioning (or clustering) N data points into K disjoint subsets Sj containing data points so as to minimize the sum-of-squares criterion

  where xn is a vector representing the the nth data point and
  uj is the geometric centroid of the data points in Sj.

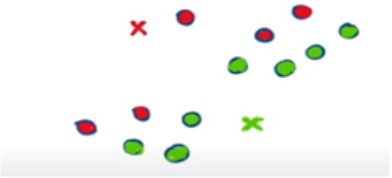
- The grouping is done by minimizing the sum of squares of distances between data and the corresponding cluster centroid.



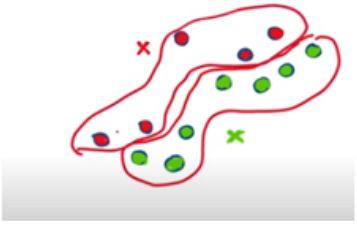
Ex: These are data points, we don't have clusters assigned to the data points. We will randomly assign 2 points. Let's k=2



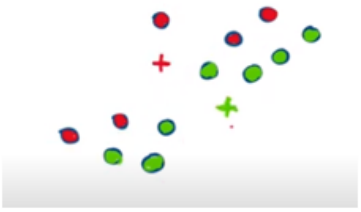
We randomly pick 2 cluster points, 2 cluster centroids one is red and another is green



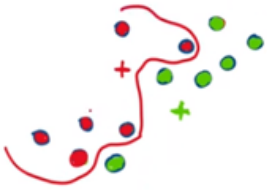
Based on the nearest cluster available to a data point we will assign a cluster to each of the data point. This is the assignment after first cluster centre.



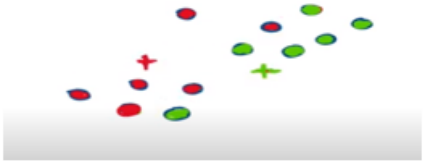
Red are in one cluster and green are in another cluster. Now we will compute mean of all the five points in red and similarly for green



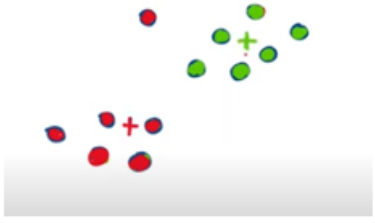
Now red and green had come to another place.

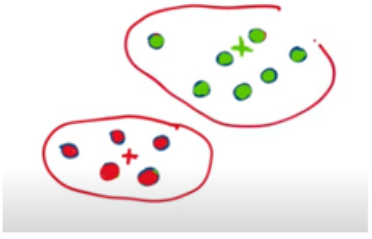


Find the mean

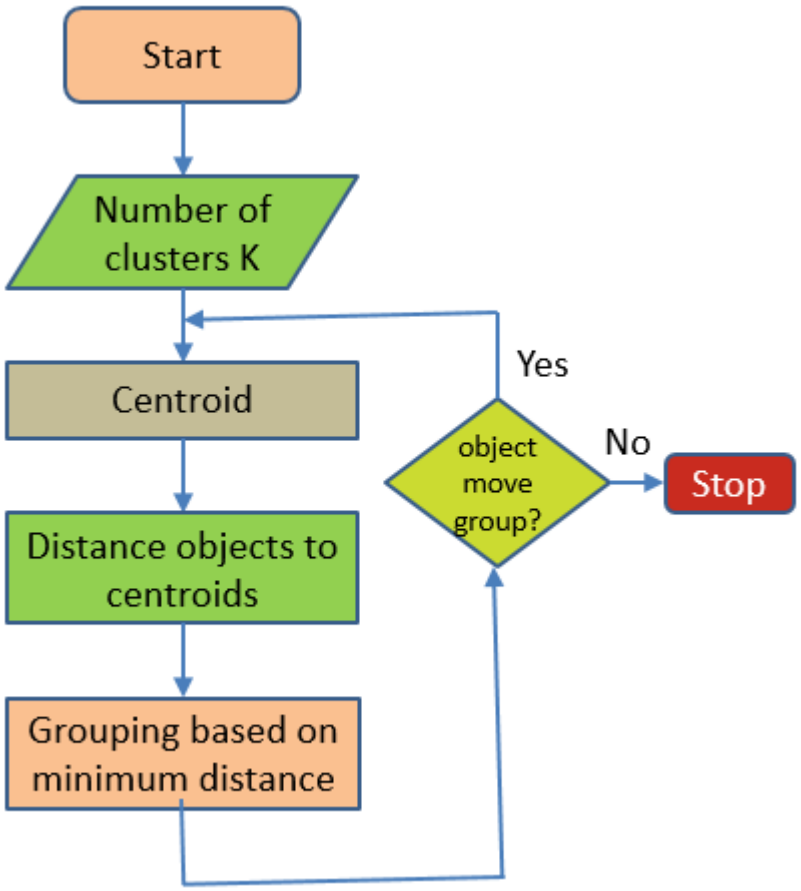


Recomputed centroids





K-means works for circular shape.



Step 1: Begin with a decision on the value of k =number of clusters

Step 2:
Put any initial partition that classifies the data into k clusters. You may assign the training samples randomly, or systematically as the following:

- Take the first k training sample as single element clusters
- Assign each of the remaining (N-k) training sample to the cluster with the nearest centroid. After each assignment, recompute the centroid of the gaining cluster.

Step 3: Take each sample in sequence and compute its distance from the centroid of each of the clusters. If a sample is not currently in the cluster with the closest centroid, switch this sample to that cluster and update the centroid of the cluster gaining the new sample and the cluster losing the sample.

Step 4: Repeat step 3 until convergence is achieved, that is until a pass through the training sample causes no new assignments.

- Given a *K*, find a partition of *K clusters* to optimize the chosen partitioning criterion (cost function)
  - global optimum: exhaustively search all partitions
- The *K-means* algorithm: a heuristic method
  - K-means algorithm (MacQueen'67): each cluster is represented by the centre of the cluster and the algorithm converges to stable centroids of clusters.
  - K-means algorithm is the simplest partitioning method for clustering analysis and widely used in data mining applications

- Given the cluster number *K*, the *K-means* algorithm is carried out in three steps after initialization:
- Initialisation: set seed points (randomly)

1) Assign each object to the cluster of the nearest seed point measured with a specific distance metric
2) Compute new seed points as the centroids of the clusters of the current partition (the centroid is the centre, i.e., *mean point*, of the cluster)
3) Go back to Step 1), stop when no more new assignment (i.e., membership in each cluster no longer changes)

Objective: Explain the different types of Machine Learning
Outcome: Outline the different types of Machine Learning

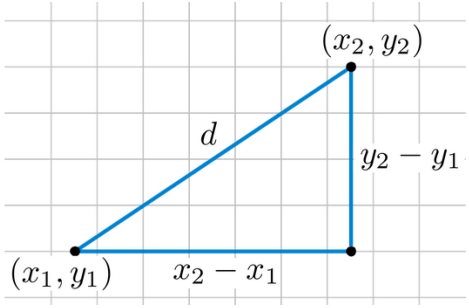
Case-study

We have 4 medicines as our training data points object and each medicine has 2 attributes. Each attribute represents coordinate of the object. We have to determine which medicines belong to cluster 1 and which medicines belong to the other cluster.

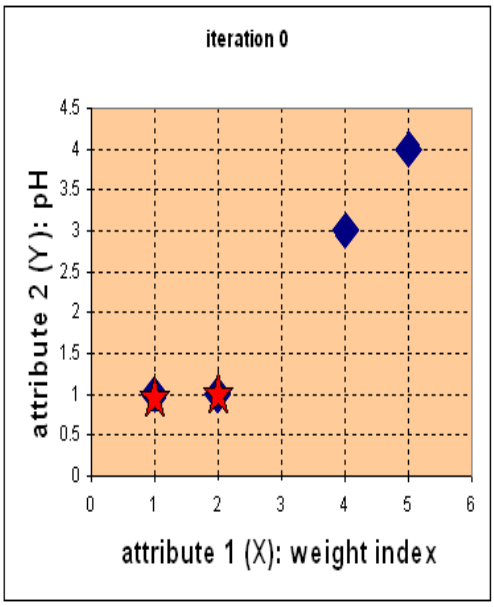
Object	Attribute1(X): weight index	Attribute2(Y): pH
Medicine A	1	1
Medicine B	2	1
Medicine C	4	3
Medicine D	5	4

Euclidean Distance formula between two points (x_1, y_1) and (x_2, y_2) are

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



Step 1:
Initial value of centroids :
Suppose we use medicine A and medicine B as the first centroids.
Let c_1 and c_2 denote the coordinate of the centroids, then c_1=(1,1) and c_2=(2,1)



Objects-Centroids distance :

we calculate the distance between cluster centroid to each object.
Let us use Euclidean distance, then we have distance matrix at iteration 0 is

$$\mathbf{D}^0 = \begin{bmatrix} 0 & 1 & 3.61 & 5 \\ 1 & 0 & 2.83 & 4.24 \end{bmatrix} \quad \begin{matrix} \mathbf{c}_1 = (1,1) & group-1 \\ \mathbf{c}_2 = (2,1) & group-2 \end{matrix}$$

$$\begin{matrix} A & B & C & D \end{matrix}$$

$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \quad \begin{matrix} X \\ Y \end{matrix}$$

Each column in the distance matrix symbolizes the object.

The first row of the distance matrix corresponds to the distance of each object to the first centroid and the second row is the distance of each object to the second centroid.

For example, distance from medicine
$C = (4, 3)$ to the first centroid $C_1(1,1)$ is ,

$$\sqrt{(4-1)^2+(3-1)^2} = 3.61$$

and its distance to the second centroid is , $C_2(2,1)$ is

$$\sqrt{(4-2)^2+(3-1)^2} = 2.83$$

etc.

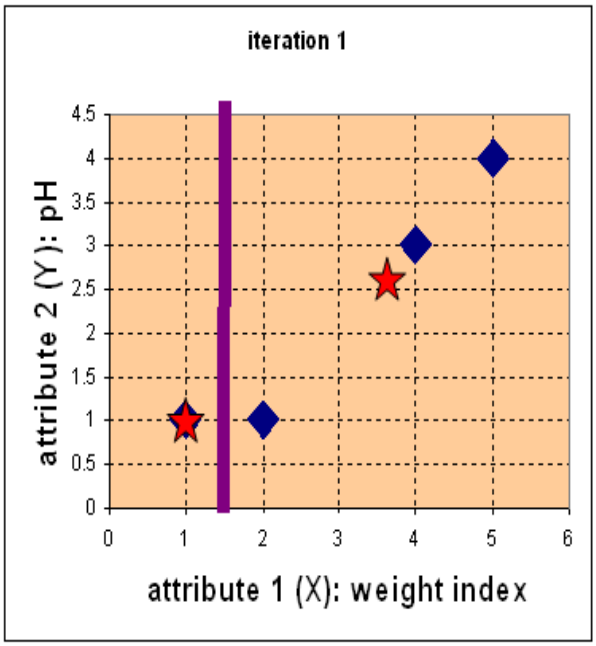
Step 2:

Objects clustering :

We assign each object based on the minimum distance.

Medicine A is assigned to group 1, medicine B to group 2, medicine C to group 2 and medicine D to group 2.

The elements of Group matrix below is 1 if and only if the object is assigned to that group.



$$G^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \quad \begin{matrix} group-1 \\ group-2 \end{matrix}$$
$$A \quad B \quad C \quad D$$

Iteration-1, Objects-Centroids distances :
The next step is to compute the distance of all objects to the new centroids.
Similar to step 2, we have distance matrix at iteration 1 is

$$D^1 = \begin{bmatrix} 0 & 1 & 3.61 & 5 \\ 3.14 & 2.36 & 0.47 & 1.89 \end{bmatrix} \quad \begin{matrix} c_1 = (1,1) & group-1 \\ c_2 = (\frac{11}{3}, \frac{8}{3}) & group-2 \end{matrix}$$
$$A \quad B \quad C \quad D$$
$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \quad \begin{matrix} X \\ Y \end{matrix}$$

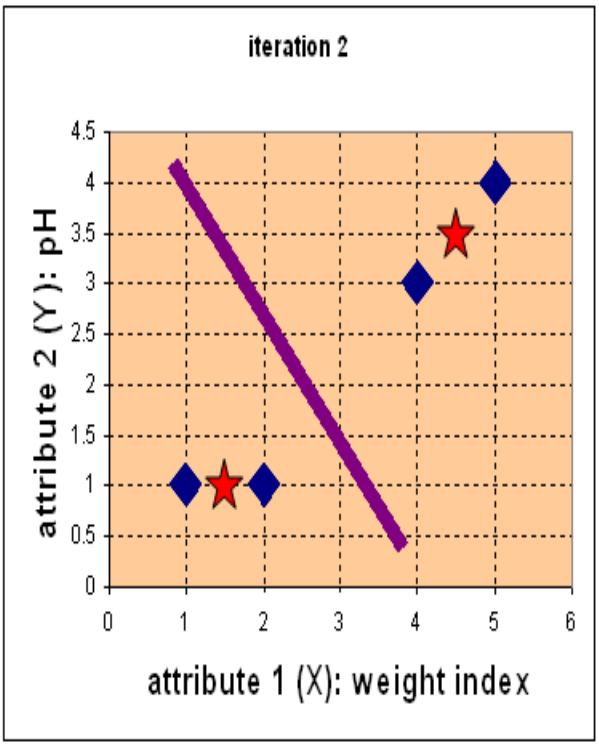
Iteration-1, Objects clustering: Based on the new distance matrix, we move the medicine B to Group 1 while all the other objects remain. The Group matrix is shown below

$$G^1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{matrix} group-1 \\ group-2 \end{matrix}$$
$$A \quad B \quad C \quad D$$

Iteration 2, determine centroids: Now we repeat step 4 to calculate the new centroids coordinate based on the clustering of previous iteration. Group1 and group 2 both has two members, thus the new centroids are

$$c_1 = (\frac{1+2}{2}, \frac{1+1}{2}) = (1\tfrac{1}{2}, 1) \qquad c_2 = (\frac{4+5}{2}, \frac{3+4}{2}) = (4\tfrac{1}{2}, 3\tfrac{1}{2})$$

and



Iteration-2, Objects-Centroids distances :
Repeat step 2 again, we have new distance matrix at iteration 2 as

$$D^2 = \begin{bmatrix} 0.5 & 0.5 & 3.20 & 4.61 \\ 4.30 & 3.54 & 0.71 & 0.71 \end{bmatrix} \begin{matrix} c_1 = (1\frac{1}{2}, 1) & group-1 \\ c_2 = (4\frac{1}{2}, 3\frac{1}{2}) & group-2 \end{matrix}$$

$$\begin{matrix} A & B & C & D \end{matrix}$$

$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \begin{matrix} X \\ Y \end{matrix}$$

Iteration-2, Objects clustering: Again, we assign each object based on the minimum distance.

$$G^2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{matrix} group-1 \\ group-2 \end{matrix}$$

$$\begin{matrix} A & B & C & D \end{matrix}$$

We obtain result that $G^2 = G^1$

.

Comparing the grouping of last iteration and this iteration reveals that the objects does not move group anymore.

Thus, the computation of the k-mean clustering has reached its stability and no more iteration is needed.

We get the final grouping as the results as:

Object	Feature1(X): weight index	Feature2(Y): pH	Group (Result)
Medicine A	1	1	1
Medicine B	2	1	1
Medicine C	4	3	2
Medicine D	5	4	2

35.1 Clustering

*Objective: Explain the **importance of Clustering***
*Outcome: Outline the **importance of Clustering***

Clustering Introduction

- **Clustering** is the classification of objects into different groups, or more precisely, the partit a data set into subsets (clusters), so that the data in each subset (ideally) share some comm often according to some defined distance measure

- **Example**

  Web document Search: A web search engine often returns thousands of pages in re to a broad query, making it difficult for users to browse to identify relevant information.

  Clustering methods can be used to automatically group the retrieved documents into a meaningful categories**.**

In Clustering , training will be done without any output.
And grouping will be done with out any predefined labels

Clustering is an example of unsupervised learning.

The process of grouping a set of physical or abstract objects into classes of *similar* objects **clustering.**

Although classification is an effective means for distinguishing groups or classes of objects, it the often costly collection and labeling of a large set of training tuples or patterns, which the uses to model each group.

Clustering can also be used for **outlier detection**.

Applications of outlier detection include the detection of credit card fraud and the monitoring of activities in electronic commerce. For example, exceptional cases in credit card transactions, suc expensive and frequent purchases, may be of interest as possible fraudulent activity.

Applications of Clustering

- Recommendation engines
- Market segmentation
- Social network analysis

- Search result grouping
- Medical imaging
- Image segmentation
- Anomaly detection

Types of Clustering

- Clustering falls under unsupervised learning
- They are different types of clustering methods, including:
  - Partitioning methods
  - Hierarchical **clustering**
  - Fuzzy **clustering**
  - Density-based **clustering**
  - Model-based **clustering**
- K-means is under Partitioning method of clustering

Analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called cluster) are more similar (in some sense or another) to each other than to those in other groups (clusters).

K-MEANS CLUSTERING

K-means clustering is an algorithm to classify or to group the objects based on attributes/features into number of group.

K is positive integer number.

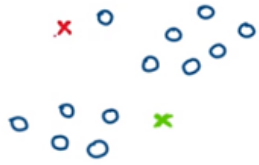
- The k-means algorithm is an algorithm to cluster n objects based on attributes into partitions, where $k < n$.

- An algorithm for partitioning (or clustering) N data points into K disjoint subsets containing data points so as to minimize the sum-of-squares criterion

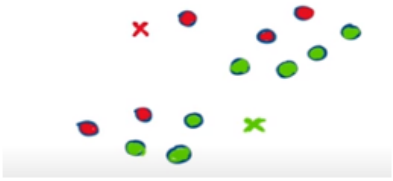
  where xn is a vector representing the the nth data point and
  uj is the geometric centroid of the data points in Sj.

- The grouping is done by minimizing the sum of squares of distances between data and th corresponding cluster centroid.

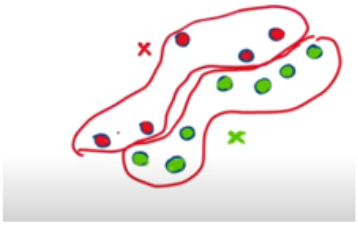
Ex: These are data points, we don't have clusters assigned to the data points. We will randomly assign 2 points. Let's k=2



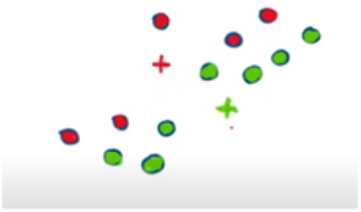
We randomly pick 2 cluster points, 2 cluster centroids one is red and another is green



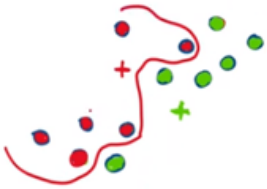
Based on the nearest cluster available to a data point we will assign a cluster to each of the data point. This is the assignment after first cluster centre.



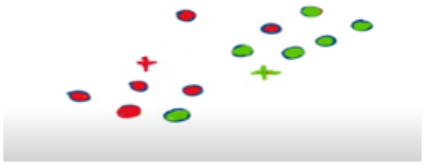
Red are in one cluster and green are in another cluster. Now we will compute mean of all the five points in red and similarly for green



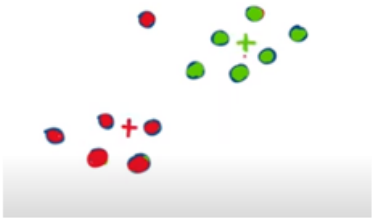
Now red and green had come to another place.

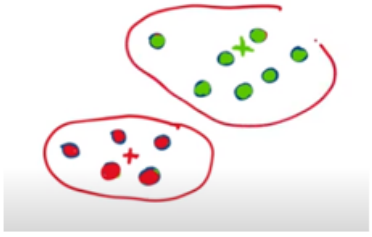


Find the mean

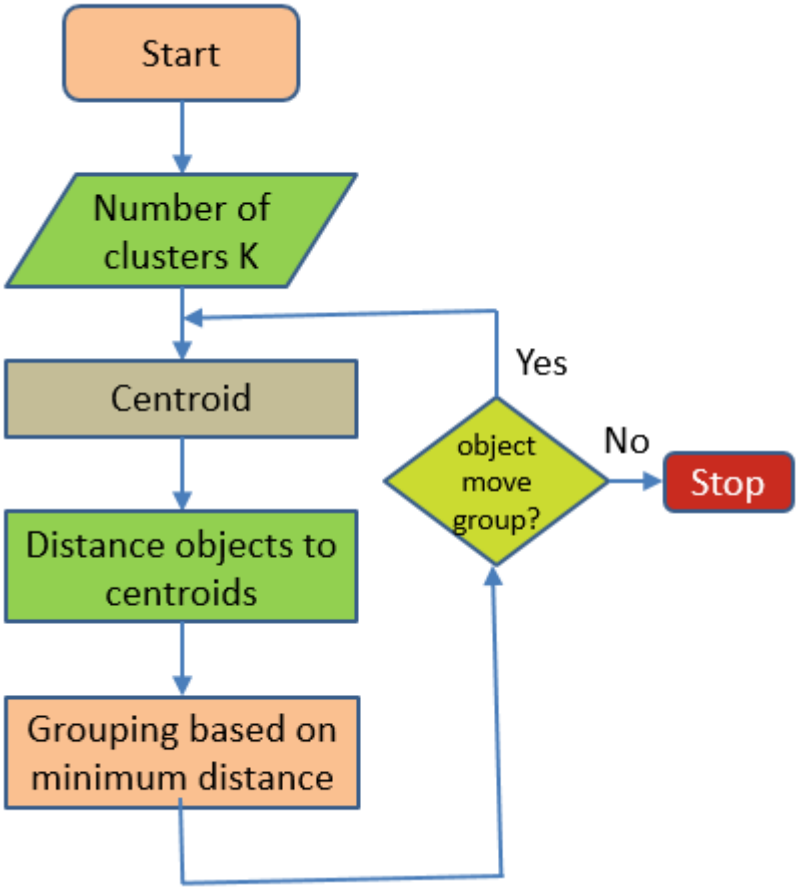


Recomputed centroids





K-means works for circular shape.



Step 1: Begin with a decision on the value of k =number of clusters

Step 2:
Put any initial partition that classifies the data into k clusters. You may assign the training samp randomly, or systematically as the following:

- Take the first k training sample as single element clusters
- Assign each of the remaining (N-k) training sample to the cluster with the nearest centroid. Af each assignment, recompute the centroid of the gaining cluster.

Step 3: Take each sample in sequence and compute its distance from the centroid of each of clusters. If a sample is not currently in the cluster with the closest centroid, switch this sample that cluster and update the centroid of the cluster gaining the new sample and the cluster losing sample.

Step 4: Repeat step 3 until convergence is achieved, that is until a pass through the training samp causes no new assignments.

- Given a *K*, find a partition of *K clusters* to optimize the chosen partitioning criterion (cost
  - global optimum: exhaustively search all partitions
- The *K-means* algorithm: a heuristic method
  - K-means algorithm (MacQueen'67): each cluster is represented by the centre of tl and the algorithm converges to stable centroids of clusters.
  - K-means algorithm is the simplest partitioning method for clustering analysis ar used in data mining applications

- Given the cluster number *K*, the *K-means* algorithm is carried out in three steps after initiali
- Initialisation: set seed points (randomly)

4) Assign each object to the cluster of the nearest seed point measured with a specific distan
5) Compute new seed points as the centroids of the clusters of the current partition (the centr centre, i.e., *mean point*, of the cluster)
6) Go back to Step 1), stop when no more new assignment (i.e., membership in each cluster changes)

35.2 Clustering

Objective: Explain the different types of Machine Learning
Outcome: Outline the different types of Machine Learning

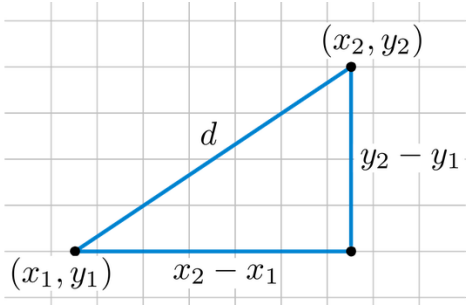
Case-study

We have 4 medicines as our training data points object and each medicine has 2 attributes. Each attribute represents coordinate of the object. We have to determine which medicines belong to cluster 1 and which medicines belong to the other cluster.

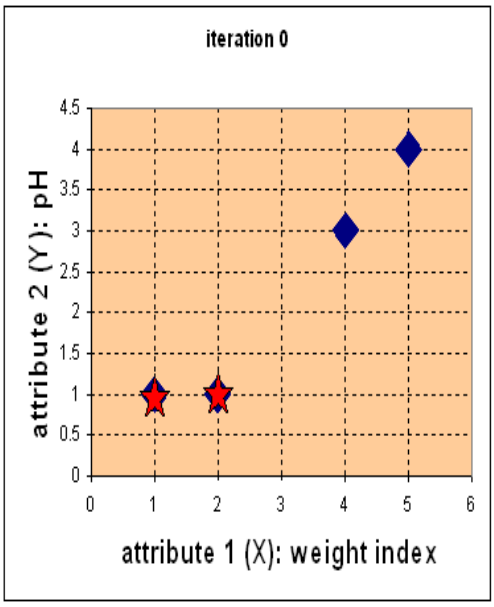
Object	Attribute1(X): weight index	Attribute2(Y): pH
Medicine A	1	1
Medicine B	2	1
Medicine C	4	3
Medicine D	5	4

Euclidean Distance formula between two points (x_1, y_1) and (x_2, y_2) are

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$



Step 1:
Initial value of centroids :
Suppose we use medicine A and medicine B as the first centroids.
Let c_1 and c_2 denote the coordinate of the centroids, then c_1=(1,1) and c_2=(2,1)



Objects-Centroids distance :

we calculate the distance between cluster centroid to each object.
Let us use Euclidean distance, then we have distance matrix at iteration 0 is

$$\mathbf{D}^0 = \begin{bmatrix} 0 & 1 & 3.61 & 5 \\ 1 & 0 & 2.83 & 4.24 \end{bmatrix} \quad \begin{matrix} \mathbf{c}_1 = (1,1) & group-1 \\ \mathbf{c}_2 = (2,1) & group-2 \end{matrix}$$

$$\begin{matrix} A & B & C & D \end{matrix}$$
$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \quad \begin{matrix} X \\ Y \end{matrix}$$

Each column in the distance matrix symbolizes the object.

The first row of the distance matrix corresponds to the distance of each object to the first centroid and the second row is the distance of each object to the second centroid.

For example, distance from medicine
C = (4, 3) to the first centroid $C_1(1,1)$ is ,

$$\sqrt{(4-1)^2+(3-1)^2} = 3.61$$

and its distance to the second centroid is , $C_2(2,1)$ is

$$\sqrt{(4-2)^2+(3-1)^2} = 2.83$$

etc.

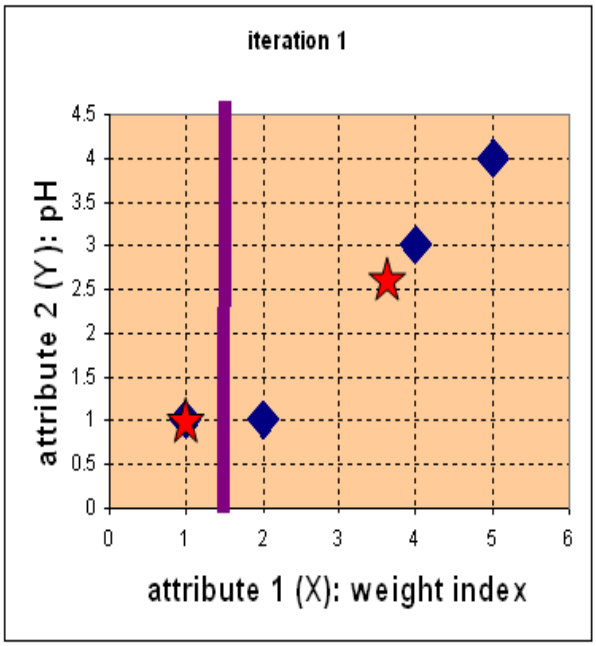
Step 2:

Objects clustering :

We assign each object based on the minimum distance.

Medicine A is assigned to group 1, medicine B to group 2, medicine C to group 2 and medicine D to group 2.

The elements of Group matrix below is 1 if and only if the object is assigned to that group.



$$G^0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \quad \begin{matrix} group-1 \\ group-2 \end{matrix}$$
$$A \quad B \quad C \quad D$$

Iteration-1, Objects-Centroids distances :
The next step is to compute the distance of all objects to the new centroids.
Similar to step 2, we have distance matrix at iteration 1 is

$$D^1 = \begin{bmatrix} 0 & 1 & 3.61 & 5 \\ 3.14 & 2.36 & 0.47 & 1.89 \end{bmatrix} \quad \begin{matrix} c_1 = (1,1) & group-1 \\ c_2 = (\frac{11}{3}, \frac{8}{3}) & group-2 \end{matrix}$$
$$A \quad B \quad C \quad D$$
$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \quad \begin{matrix} X \\ Y \end{matrix}$$

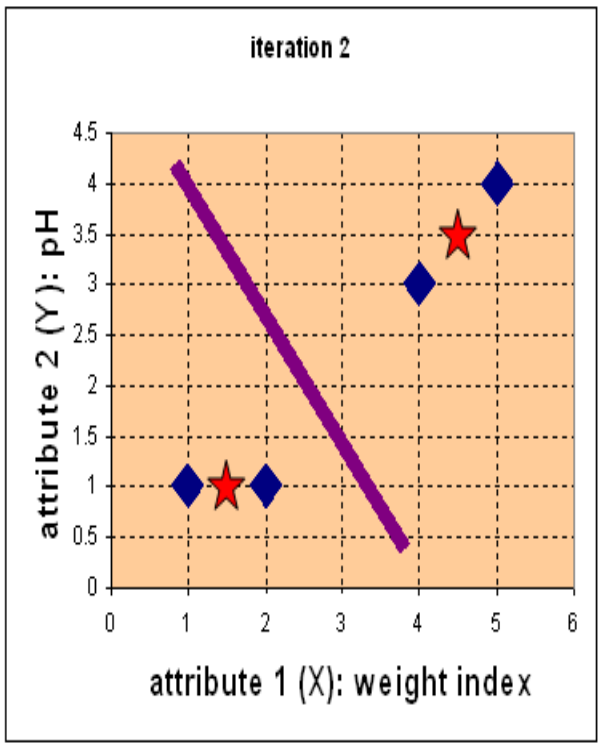
Iteration-1, Objects clustering: Based on the new distance matrix, we move the medicine B to Group 1 while all the other objects remain. The Group matrix is shown below

$$G^1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{matrix} group-1 \\ group-2 \end{matrix}$$
$$A \quad B \quad C \quad D$$

Iteration 2, determine centroids: Now we repeat step 4 to calculate the new centroids coordinate based on the clustering of previous iteration. Group1 and group 2 both has two members, thus the new centroids are

$$c_1 = (\frac{1+2}{2}, \frac{1+1}{2}) = (1\tfrac{1}{2}, 1) \qquad c_2 = (\frac{4+5}{2}, \frac{3+4}{2}) = (4\tfrac{1}{2}, 3\tfrac{1}{2})$$

and



Iteration-2, Objects-Centroids distances :
Repeat step 2 again, we have new distance matrix at iteration 2 as

$$D^2 = \begin{bmatrix} 0.5 & 0.5 & 3.20 & 4.61 \\ 4.30 & 3.54 & 0.71 & 0.71 \end{bmatrix} \begin{matrix} c_1 = (1\frac{1}{2}, 1) & group-1 \\ c_2 = (4\frac{1}{2}, 3\frac{1}{2}) & group-2 \end{matrix}$$

$$\begin{matrix} A & B & C & D \end{matrix}$$

$$\begin{bmatrix} 1 & 2 & 4 & 5 \\ 1 & 1 & 3 & 4 \end{bmatrix} \begin{matrix} X \\ Y \end{matrix}$$

Iteration-2, Objects clustering: Again, we assign each object based on the minimum distance.

$$G^2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{matrix} group-1 \\ group-2 \end{matrix}$$

$$\begin{matrix} A & B & C & D \end{matrix}$$

We obtain result that $G^2 = G^1$

.

Comparing the grouping of last iteration and this iteration reveals that the objects does not move group anymore.

Thus, the computation of the k-mean clustering has reached its stability and no more iteration is needed.

We get the final grouping as the results as:

Object	Feature1(X): weight index	Feature2(Y): pH	Group (Result)
Medicine A	1	1	1
Medicine B	2	1	1
Medicine C	4	3	2
Medicine D	5	4	2