Phần I : Ngôn ngữ lập trình Python

Python là ngôn ngữ được sử dụng phổ biến trong các ứng dụng về Machine Learning.

Cài đặt Python

Cài đặt Python theo hướng dẫn tại trang chủ.

Cài đặt các thư viện machine learning : từ cửa sổ cmd gõ lệnh:

```
pip install matplotlib pandas sklearn tensorflow keras
```

Trên windows, có thể download và cài đặt WinPython, sẽ có đầy đủ các thư viện trên.

1. Biến số

Biến số trong Python được khai báo bằng cách khởi tạo giá trị cho tên biến

```
bienso = <giatri>
```

Ví dụ :

```
>> x = 1
>> x += 1
>> ten = 'Nguyễn Văn An'
```

2. Vào ra dữ liệu

In dữ liệu ra màn hình

```
print(<danh sách giá trị ngăn cách nhau bởi dấu phảy>)
```

Ví dụ:

```
>> z = 1
>> y = 2
>> z = x + y
>> print(x, '+', y, '=', z)
```

Vào dữ liệu từ bàn phím:

```
giatri = input('Nhập giá trị cho biến : ')
```

Ví dụ:

```
>> x = input('x=')
>> x = float(x)
```

3. Dữ liệu kiểu số

Tương tự các ngôn ngữ lập trình, Python có 2 kiểu dữ liệu số : số nguyên (**int**) và số thập phân (**float**)

```
>> x = 1    # số nguyên
>> y = 1.2  # số thập phân
>> z = 1e6  # số thập phân
```

Các phép tính trên dữ liệu số

- Phép tính số học **+ - * /**
- Phép lấy phần dư **%**
- Phép lũy thừa ****** (hai dấu * viết liền nhau)
- Làm tròn số **round(x,n)** : làm tròn đến n chữ số thập phân sau dấu phảy
- Lấy giá trị tuyệt đối **abs**

Trong python 2, phép / hai số nguyên cho kết quả là số nguyên, trong python 3 phép chia 2 số nguyên cho kết quả là số thập phân. Phép **//** (2 dấu / liền nhau) được dùng để lấy kết quả số nguyên khi chia 2 số nguyên trong python 3

Tương tự một số ngôn ngữ lập trình, python cho phép dùng các phép tính rút gọn **+= , -=, *=, /=** …

Ví dụ:

```
>> 1 + 2
      3

>> 3 - 2
      1

>> 3 * 4
      12

>> 1/2
      0.5

>> 2 ** 10
      1024

>> 10 // 3
      3

>> 10 % 3
      1
```

Các hàm toán học:
Các hàm toán học nằm trong thư viện math của python

- Các hàm lượng giác : **sin, cos, tan, asin, acos, atan**
- Hàm căn bậc 2 : **sqrt**
- Hàm logarith : **log10** (cơ số 10), **log** (cơ số e)

Ví dụ:

```
>> import math
```

```
>> print(math.sin(math.pi/2))
       1.0

>> print(math.sqrt(4))
       2.0
```

4. Dữ liệu kiểu Boolean

Kiểu Boolean (logic) trong Python có 2 giá trị là **True** và **False**. Các phép tính tạo ra dữ liêu kiểu Boolean:

- Các phép so sánh : lớn hơn (>), nhỏ hơn (<), lớn hơn hoặc bằng (>=), nhỏ hơn hoặc bằng (<=), bằng nhau (==), khác nhau (!=)
- Phép phủ định **not**
- Phép và **and**
- Phép hoặc **or**

Ví dụ:

```
>> 1 == 2-1
      True

>> 1 > 2
      False

>> not (1 < 2)
      False

>> (1 < 2) and (2 <3)
     True

>> (1 > 2) or (2 >3)
     False
```

5. Dữ liêu kiểu String

Các giá trị dữ liệu String được đặt trong 2 dấu nháy đơn (') hoặc 2 dấu nháy kép (")

```
hoten = 'Nguyễn Văn An'
diachi = "Hà Nội"
```

Các hàm và phép tính thông dụng về dữ liệu văn bản trong Python:

- Hàm **str** : chuyển đổi số (và các kiểu dữ liệu khác) sang dữ liệu văn bản
- Phép **+** 2 chuỗi : ghép 2 chuỗi văn bản thành một
- Hàm **lower** chuyển văn bản sang chữ thường

- Hàm **upper** chuyển văn bản sang chữ hoa
- Hàm **replace** : thay nội dung một xâu con trong xâu văn bản
- Hàm **split** : tách một xâu thành các xâu con

Ví dụ

```
>> str(1000)
     '1000'

>> 'Chào ' + 'bạn'
     'Chào bạn'

>> 'Chào bạn'.lower()
      'chào bạn'

>> 'Chào bạn'.upper()
     'CHÀO BẠN'

>> 'Tôi sống ở Hà Nội'.replace('Hà Nội', 'Huế')
 'Tôi sống ở Huế'

>> 'Hà Nội'.split()
      ['Hà', 'Nội']

>> 'Hà Nội, Việt Nam'.split(',')
      ['Hà Nội', 'Việt Nam']
```

Substring:

Với một xâu văn bản, chúng ta có thể lấy ra từng đoạn nhỏ của xâu đó. Mỗi đoạn này gọi là một xâu con.

Cú pháp để lấy một xâu con từ một xâu văn bản có trước:

text[start:end]

Trong đó các chỉ số `start, end` là các chỉ số bắt đầu và kết thúc của các kí tự được lấy ra từ xâu gốc. Lưu ý là các kí tự sẽ chỉ được lấy từ vị trí `start` đến `end-1`

Nếu giá trị `end` được bỏ qua thì xâu con sẽ được lấy từ vị trí `start` đến hết xâu gốc:
text[start:] ~ text[start: len(text)]

Nếu giá trị start được bỏ qua thì xâu con sẽ được lấy từ đầu xâu gốc:
text[:end] ~ text[0: end]

Nếu các chỉ số start, end có giá trị âm, chúng được hiểu là tính từ cuối xâu trở về, ví dụ:
text[-2 : -1] ~ text[len(text)-2 : len(text)-1]

Ví dụ 1:

```
>> text = 'Chào bạn'
>> print(text[0:4])
     Chào
```

```
>> print(text[5:])
     bạn

>> print(text[:4])
     Chào

>> print(text[-1])
     n

>> print(text[-3:])
     Bạn
```

6. Dữ liệu kiểu List:

Kiểu dữ liệu List dùng để chứa một dãy nhiều phần tử. Khi khai báo giá trị, các phần từ của List được đặt trong cặp dấu [ ] , ví dụ:

```
>> dayso = [1, 3, 5, 7, 9]
>> danhsach_hocsinh = ["Nguyễn Văn An", "Nguyễn Chí Cường", "Nguyễn
Mạnh Tuấn"]
```

Các phần tử của một List có thể không cùng kiểu dữ liệu:

```
>> diachi = [322, "Tây Sơn", "Hà Nội"]
```

Các hàm và phép tính thông dụng trên dữ liệu kiểu List:

- Hàm **`len`** : Lấy số phần tử của một List
- Hàm **`append`** : Thêm phần từ vào List
- Hàm **`remove`** : Xóa một phần tử khỏi List
- Phép **+** 2 danh sách : tạo thành một List mới bằng cách ghép 2 danh sách thành phần
- Hàm **`extend`** : Ghép một List con vào cuối một List.
- Hàm **reverse** : Đảo ngược một List
- Phép **`in`** : kiểm tra một giá trị có nằm trong List không
- Phép **`not in`** : Kiểm tra một giá trị có không nằm trong List không

Tương tự với kiểu dữ String, kiểu List cũng cho phép truy nhập đến từng phần tử qua chỉ số, đồng thời cho phép lấy ra từng đoạn con theo các chỉ số đầu và cuối:

- **`lst[i]`** : phần tử thứ `i` của List
- **`lst[start:end]`** : đoạn con chứa các phần tử từ chỉ số `start` đến `end-1`
- **`lst[start:]`** : đoạn con chứa các phần tử từ chỉ số `start` đến cuối List
- **`lst[: end]`** : đoạn con chứa các phần tử từ chỉ số `0` đến `end-1`

Ví dụ:

```
>> lst = [1, 2, 3, 4, 5]
>> print(len(lst))
     5

>> lst.append(6)
```

```
>> print(lst)
    [1,2,3,4,5,6]

>> lst.remove(3)
>> print(lst)
    [1,2,4,5,6]

>> lst = lst + [7, 8]
>> print(lst)
    [1,2,4,5,6,7,8]

>> lst.extend([9, 10])
>> print(lst)
    [1,2,4,5,6,7,8,9,10]

>> 1 in lst
    True

>> 3 not in lst
    True

>> print(lst[5])
    7

>> print(lst[1:4])
    [2,4,5]

>> print(lst[:3])
    [1,2,3]

>> print(lst[6:])
    [8,9,10]

>> print(lst[-2:])
    [9,10]

>> lst.reverse()
>> print(lst)
    [10,9,8,7,6,5,4,2,1]
```

7. Kiểu dữ liệu Tuple

Dữ liệu kiểu Tuple tương tự với kiểu List, chỉ khác là các phần tử sau khi khai báo thì không sửa (thêm, xóa) được. Các phần tử của dữ liệu kiểu Tuple được đặt trong cặp dấu ( ) :

```
dayso = (1, 3, 5, 7, 9)
danhsach_hocsinh = ("Nguyễn Văn An", "Nguyễn Chí Cường", "Nguyễn Mạnh
Tuấn")
```

Dữ liệu kiểu Tuple có thể dùng để gán cho một bộ nhiều biến số:

```
x, y = 1, 2
```

8. Dữ liệu kiểu Set

Kiểu dữ liệu Set dùng để chứa các phần tử của một tập hợp. So với kiểu List, kiểu Set có điểm khác:

- Trong Set không có 2 phần tử cùng giá trị. Nếu một phần tử đã có trong Set thì khi thêm mới phần tử cùng giá trị vào Set, chúng chỉ được tính là một

- Không thể dùng chỉ số để truy nhập các phần tử trong Set như cách làm với List

Các hàm và phép tính trên Set:

- Hàm **`add`** : thêm giá trị mới vào Set
- Hàm **`remove`** : xóa phần tử trong Set
- Phép **`in`** : kiểm tra giá trị có nằm trong Set không
- Phep **`not in`** : kiểm tra giá trị có không nằm trong Set không
- Hàm **`union`** : hợp của 2 Set
- Hàm **`intersection`** : giao của 2 Set

Ví dụ 1:

```
>> s1 = set([1, 2])
>> s1.add(3)
>> s1.remove(1)

>> 1 not in s1
    True

>> 2 in s1
    True

>> s2 = set([3, 4])
>> s3 = set.union(s1, s2)
>> print(s3)
    {2, 3, 4}

>> s4 = set.intersection(s1, s2)
>> print(s4)
    {3}
```

9. Dữ liệu kiểu Dictionary

Các phần tử của kiểu dữ liệu Dictionary được khai báo theo từng cặp và đặt trong cặp dấu { }, các cặp được ngăn cách nhau bởi dấu phảy, hai giá trị trong cặp ngăn cách nhau bởi dấu hai chấm:

```
d = {"một" : 1, "hai" : 2, "ba" : 3}
```

Lấy một phần tử từ Dictionary:

```
d[key]
d.get(key, <gia_tri_mac_dinh>)
```

10. Lệnh điều khiển if:

Các dạng cấu trúc if:

- ```
  if <điều kiện> :
      <Lệnh>
  ```

- ```
  if <điều kiện> :
      <Lệnh 1>
  else:
      <Lệnh 2>
  ```

- ```
  if <điều kiện 1> :
      <Lệnh 1>
  elif <điều kiện 2>  :
      <Lệnh 2>
  elif <điều kiện 3>  :
      <Lệnh 3>
  else:
      <Lệnh 4>
  ```

Lưu ý:

- Cuối các dòng của lệnh `if` hoặc `else` là dấu 2 chấm (`:`). Python sử dụng dấu hai chấm để bắt đầu cho một khối lệnh con, bạn sẽ quen dần với điều này.

- Các dòng lệnh bên dưới `if` hoặc `else` cần được viết lùi vào một số khoảng trắng (thường là 4 khoảng trắng, hoặc 1 phím tab) so với dòng chứa lệnh `if/else`, nếu có nhiều hơn một dòng lệnh thì chúng phải được viết thẳng hàng.

Ví dụ:

```
x = input('x = ')
x = int(x)

if x % 2 == 0:
    print('Số chẵn')
else:
    print('Số lẻ')
```

11. Vòng lặp for

Cấu trúc for:

```
for bienLap in tapGiaTri:
    <khoi_lenh>
```

Các cách biểu diễn tập giá trị lặp:

- Liệt kê các thành phần. Ví dụ [1, 2, 3, 4, 5]. Đây thực chất là dữ liệu List.
```

- Khoảng "`range(end)`" : bao gồm các số tự nhiên từ 0 đến trước `end`. Lưu ý, khoảng này không chứa giá trị `end`, tức giá trị cuối của khoảng là `end-1`

- Khoảng "`range(start, end)`" : bao gồm các số nguyên bắt đầu từ `start` đến trước `end`.

- Khoảng "`range(start, end, increment)`" : bao gồm các số nguyên từ `start` đến trước `end` và tăng đều theo khoảng cách `increment`

Ví dụ:

Tính tổng các số từ 1 đến 100

```
S = 0
for i in range(1, 101):
    S += i

print('S = ', S)
```

12. Vòng lặp while

Cấu trúc while:

```
while dieukien:
        <khoi_lenh>
```

Ví dụ : Đổi số thập phân sang nhị phân

```
x = input("x=")
x = int(x)

s = ''
while x > 0:
    i = x % 2
    s = str(i) + s
    x = x // 2

print(s)
```

13. Hàm

Cấu trúc khai báo hàm:

```
def tenham(<danh_sach_bien>):
    <noi_dung_ham>
    return <ket_qua>
```

Ví dụ : viết hàm tính chu vi & diện tích một hình chữ nhật

```
def calcAreaAndPerimeter(width, height):
    S = width * height
    P = 2*(width + height)
    return S, P

S, P = calcAreaAndPerimeter(5, 4)
print('S = ', S)
```

```
print('P = ', P)
```

14. Truy nhập File

Đọc file:

Đọc từng dòng của file:

```
f = open('ten_file')
for line in f:
    print(line)
f.close()
```

Đọc tất cả các dòng :

```
f = open('ten_file')
lines = f.readlines()
f.close()
```

Đọc toàn bộ file vào một biến String:

```
f = open('ten_file')
content = f.read()
print(content)
f.close()
```

Ghi file :

```
f = open('ten_file', 'w')   # Dùng 'a' thay cho 'w' để ghi tiếp vào cuối file
f.write('Line 1\n')
f.write('Line 2\n')
f.close()
```

Phần II : Các thư viện xử lý số liệu của Python

1. Numpy

Numpy là thư viện cho phép xử lý các mảng số (một chiều & nhiều chiều). So với kiểu List có sẵn của Python, các mảng của numpy khác ở điểm là các phần tử của mảng có cùng kiểu dữ liệu (int, float, string, …), do đó tốc độ xử lý trên các mảng này nhanh hơn so với xử lý trên kiểu List.

Một số hàm thường sử dụng của numpy:

- Hàm **numpy.array(lst, dtype)** : chuyển một đối tượng kiểu List sang một mảng có kiểu dữ liệu *dtype*. Các giá trị *dtype* thường dùng:
    - ✓ `numpy.float`
    - ✓ `numpy.float32`
    - ✓ `numpy.float64`
    - ✓ `numpy.int`
    - ✓ `numpy.uint`
    - ✓ `numpy.uint8`
    - ✓ `numpy.int8`
    - ✓ `numpy.uint16`

- ✓ numpy.int16
- ✓ numpy.int32
- ✓ numpy.uint32
- ✓ numpy.int64
- ✓ numpy.uint64

Nếu kiểu dữ liệu không được chỉ định, numpy sẽ dựa vào giá trị các phần tử của List để chọn kiểu dữ liệu phù hợp.

- Hàm **numpy.zeros(shape, dtype)**: tạo một mảng có kích thước bằng với *shape* và kiểu dữ liệu *dtype*, các phần tử của mảng được khởi tạo bằng 0. Giá trị của *shape* có thể là một số - khi đó mảng tạo ra là mảng một chiều, hoặc một bộ số - khi đó mảng tạo ra là mảng nhiều chiều. Giá trị mặc định của *dtype* là *numpy.float*

- Hàm **numpy.ones(shape) :** tương tự hàm *zeros* chỉ khác là các phần từ của mảng được khởi tạo bằng 1

- Hàm **numpy.concatenate((arr1, arr2, …,))** : Ghép các mảng lại thành một

- Hàm **arr.shape, arr.dtype** : trả về kích thước và kiểu dữ liệu của một mảng

- Hàm **arr.reshape(new_shape)**: tạo ra một mảng mới bằng cách xếp lại dữ liệu của mảng *arr* đã có theo kích thước mới *new_shape*

Ví dụ :

```
>> import numpy

>> X = numpy.array([1,2,3,4])
>> X
    array([1, 2, 3, 4])

>> X2 = numpy.array([[1,2],[3,4]])
>> X2
    array([[1, 2],
           [3, 4]])

>> X3 = numpy.ones(4)
>> X3
    array([ 1.,  1.,  1.,  1.])

>> X4 = numpy.zeros((2,2), dtype=numpy.uint8)
>> X4
    array([[0, 0],
           [0, 0]], dtype=uint8)

>> X4.shape
    (2, 2)

>> X4.dtype
    dtype('uint8')

>> X4.reshape((1, 4))
    array([0, 0, 0, 0])

>> X5 = numpy.ones((2,2), dtype=numpy.uint8)
>> X5
   array([[1, 1],
```

```
          [1, 1]], dtype=uint8)

>> X6 = numpy.concatenate((X4, X5))
>> X6
   array([[0, 0],
          [0, 0],
          [1, 1],
          [1, 1]], dtype=uint8)
```

Truy nhập phần tử của mảng trong numpy:

- Với mảng một chiều, cách truy nhập phần tử của mảng tương tự cách truy nhập phần tử của List
- Với mảng nhiều chiều, một phần tử được truy nhập bởi danh sách các chỉ số nằm trong một cặp dấu ngoặc vuông duy nhất : arr[i_1, i_2, …, i_n]

  Tương tự mảng một chiều hay List, có thể cách dùng các chỉ số dạng hai chấm(*start:end*, *start:*, *:end*, …) để truy nhập tới từng đoạn nhỏ trong mảng nhiều chiều.

Ví dụ:

```
>> import numpy

>> X = numpy.array([[1,2],[3,4]])

>> X[1,1]
    4

>> X[:, 0]
    array([1, 3])

>> X[1, :]
    array([3, 4])
```

Các phép tính trên mảng của numpy:

- Các phép tính số học : Nếu hai mảng có cùng kích thước thì các phép tính **+, -, *, /, **, %, //**, … có thể được thực hiện trên hai mảng đó. Kết quả là một mảng có cùng kích thước mà mỗi phần tử là kết quả của phép toán được thực hiện trên từng cặp phần tử của hai mảng đầu vào. Các phép tính kiểu này gọi là **element-wise operation**.
- Các phép tính logic : Một mảng trong numpy có thể được so sánh với một giá trị số (int, float), hoặc với một mảng cùng kích thước, kết quả trả về là một mảng kiểu *boolean* cùng kích thước với mảng gốc, phép so sánh cũng được thực hiện theo kiểu element-wise.

  Kết quả của phép tính logic có thể được dùng làm chỉ số để truy nhập đến mảng (một hình thức filter dữ liệu). Chi tiết tham khảo phần ví dụ.
- Các hàm toán học : tương ứng với các hàm toán học trong thư viện math của Python (**sin, cos, tan, asin, acos, sqrt, floor, ceil, log, log10, exp**,…), numpy cung cấp các hàm tương tự với cùng tên như trên, nhưng thực hiện trên các mảng dữ liệu. Kết quả trả về cũng là một mảng theo cách xử lý *element-wise*.
- Hàm **numpy.sum, numpy.mean** : Tính tổng / trung bình của một mảng dữ liệu. Theo mặc định, toàn bộ các phần tử của mảng (một chiều & nhiều chiều) sẽ được gộp lại thành một danh sách và phép tính tổng/trung bình sẽ thực hiện trên danh sách này để cho kết quả là một giá trị số duy nhất. Nếu muốn tính tổng/trung bình theo từng hàng/cột của mảng, cần chỉ định tham số

axis cho hàm này, giá trị của *axis* là chỉ số của chiều dữ liệu (hàng/cột …) mà phép tính tổng/trung bình sẽ thực hiện dọc theo chiều dữ liệu này.

Ví dụ:

```
>> import numpy

>> X1 = numpy.array([[1,2],[3,4]])
>> X1
    array([[1, 2],
           [3, 4]])

>> X2 = numpy.array([[5,6],[7,8]])
    array([[5, 6],
           [7, 8]])

>> X1 + 1
    array([[2, 3],
           [4, 5]])

>> X1 + X2
    array([[ 6,  8],
           [10, 12]])

>> X3 = X1 * X1
>> X4 = numpy.sqrt(X3)
>> X4
    array([[ 1.,  2.],
           [ 3.,  4.]])

>> numpy.sum(X1)
    10

>> numpy.mean(X2, axis=0)
    array([ 6.,  7.])

>> numpy.mean(X2, axis=1)
    array([ 5.5,  7.5])

>> indexes = (X1 >= 2) & (X2 % 2 == 0)
>> indexes
    array([[False, True],
           [False,  True]], dtype=bool)

>> X1[indexes]
    array([2, 4])
```

Pandas

Pandas là thư viện cho phép xử lý các bảng dữ liệu có cấu trúc. Một bảng dữ liệu bao gồm nhiều cột, mỗi cột tương ứng với một trường của đối tượng dữ liệu. Cách mô tả này tương tự cách lưu trữ dữ liệu trong file excel, csv và trong database

id	price	quantity
1	50000	10
2	100000	15
3	80000	12

Ví dụ về một bảng dữ liệu

Trong pandas, khái niệm bảng dữ liệu tương đương với đối tượng DataFrame

- Khởi tạo một bảng dữ liệu:

```
>> from pandas import DataFrame

>> table1 = DataFrame({
                'id' : [1, 2, 3],
                'price' : [50000, 100000, 80000],
                'quantity' : [10, 15, 12]
        })

>> table1
            id   price  quantity
         0   1   50000      10
         1   2   100000     15
         2   3   80000      12
```

- Đọc dữ liệu từ file csv:
  Giả sử dữ liệu về một bảng được lưu trong file `table.csv` dưới dạng sau:

```
id,price,quantity
1,50000,10
2,100000,15
3,80000,12
```

  Để đọc dữ liệu từ file, pandas cung cấp hàm **read_csv :**

```
>> import pandas
>> table = pandas.read_csv('table.csv')
```

  Kết quả là bảng dữ liệu hoàn toàn tương tự như khai báo theo kiểu DataFrame ở phía trên.

- Lưu bảng dữ liệu vào file csv:

```
>> table.to_csv('test.csv', index=False)
```

  Tham số *index* đặt bằng False để cột chỉ số của bảng (cột do pandas tự sinh) không ghi vào file csv.

- Truy nhập dữ liệu trong bảng:
  - Truy nhập đến một hàng:

```
>> row0 = table.loc[0]
>> row0
      id              1
      price       50000
      quantity       10
      Name: 0, dtype: int64
```

```
>> row0[0]
      1

>> row0.price  # tương đương với row0['price']
      50000
```

- Truy nhập đến một cột :

```
>> price_col = table.price  # tương đương với table['price']
>> price_col
      0     50000
      1    100000
      2     80000
      Name: price, dtype: int64

>> price_col[0]
      50000
```

- Truy nhập toàn bộ bảng dữ liệu dưới dạng mảng numpy 2 chiều :

  Cách truy nhập theo từng hàng hay từng cột giúp chương trình dễ đọc nhưng tốc độ xử lý khá chậm. Khi cần xử lý trên nhiều hàng hay nhiều cột, cách nhanh nhất là truy nhập đến mảng dữ liệu 2 chiều nằm bên trong mỗi bảng:

```
>> data = table.values
>> data
    array([[     1,  50000,     10],
               [     2, 100000,     15],
               [     3,  80000,     12]], dtype=int64)
```

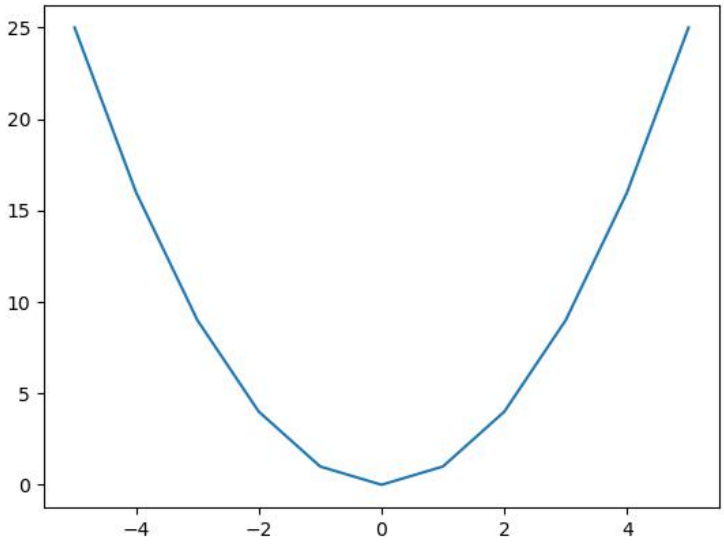
Matplotlib

Matplotlib là thư viện cho phép vẽ các đồ thị.
Dạng đồ thị hay sử dụng nhất là đồ thị dạng đường (line) biểu hiện mối quan hệ giữa 2 đại lượng:

```
import matplotlib.pyplot as plt

X = list(range(-5, 6)))
Y = [x*x for x in X]

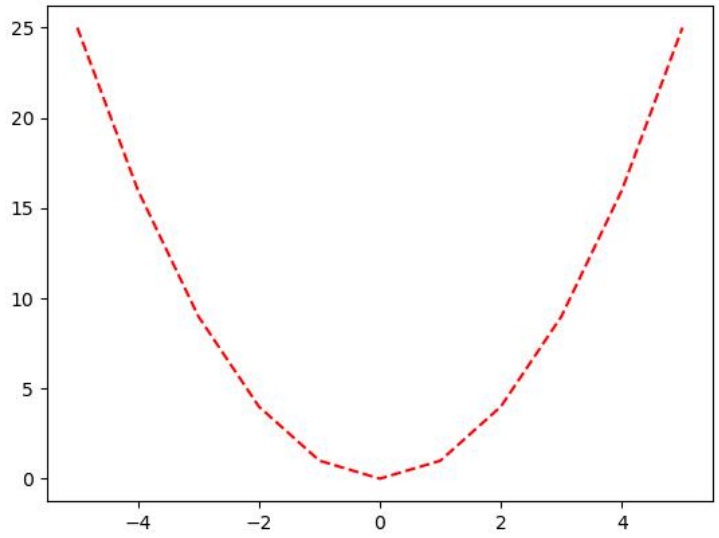
plt.plot(X,Y)
plt.show()
```



Đồ thị vẽ bằng matplotlib

- Chọn màu và dạng đường vẽ

```
plt.plot(X,Y, '--', color='red')
plt.show()
```



Chọn màu và dạng của đường vẽ

Việc chọn màu được thực hiện qua tham số *color*, trong hàm *plot*. Dạng đường vẽ được chọn qua kí hiệu mô tả đặc trưng của đường:

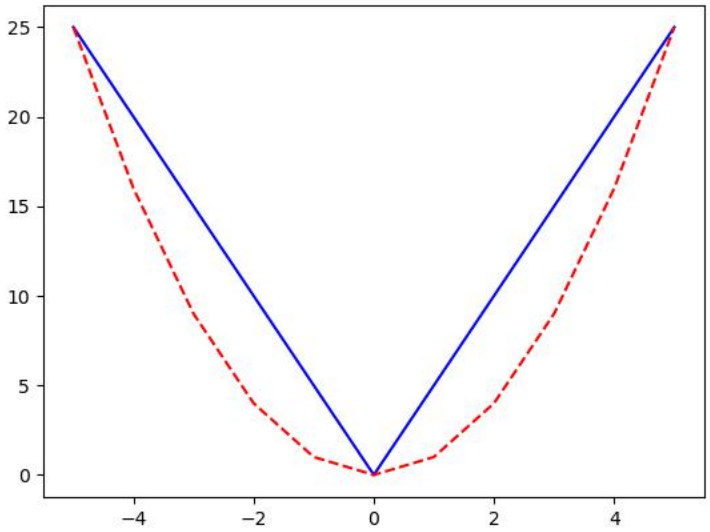
- '-' : nét liền
- '--' : nét đứt
- '.' : từng điểm

- Vẽ nhiều đường trên một đồ thị:
  Cách đơn giản nhất để vẽ nhiều đường trên một đồ thị là dùng nhiều lệnh *plot* liên tiếp nhau trước khi gọi một lênh *show* duy nhất.

```
import matplotlib.pyplot as plt

X = list(range(-5, 6))
Y1 = [5*abs(x) for x in X]
Y2 = [x*x for x in X]

plt.plot(X, Y1, color='blue')
plt.plot(X, Y2, '--', color='red')
plt.show()
```



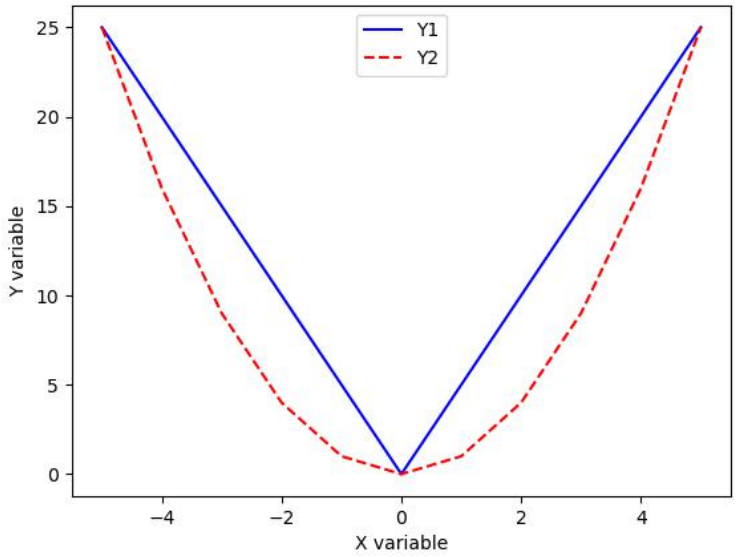
Vẽ nhiều đường trên một đồ thị

- Thêm chú thích:
  Các lệnh thêm chú thích:
  - `xlabel` : Tiêu đề cho trục nằm ngang
  - `ylabel` : Tiêu đề cho trục thẳng đứng
  - `legend` : Hiện tên các đường trên đồ thị

```
import matplotlib.pyplot as plt

X = list(range(-5, 6))
Y1 = [5*abs(x) for x in X]
Y2 = [x*x for x in X]

plt.plot(X, Y1, color='blue', label='Y1')
plt.plot(X, Y2, '--', color='red', label='Y2')
plt.xlabel('X variable')
plt.ylabel('Y variable')
plt.legend(loc='upper center')
plt.show()
```



Thêm chú thích cho đồ thị

Phần III : Các thuật toán cơ bản

1. Linear regression

Linear regression là thuật toán dùng để xấp xỉ đầu ra một hệ thống /đối tượng theo một tổ hợp tuyến tính của các đầu vào:

$$Y \approx A.X + B$$

Ví dụ 1: Dân số thế giới qua các năm
https://en.wikipedia.org/wiki/World_population_estimates

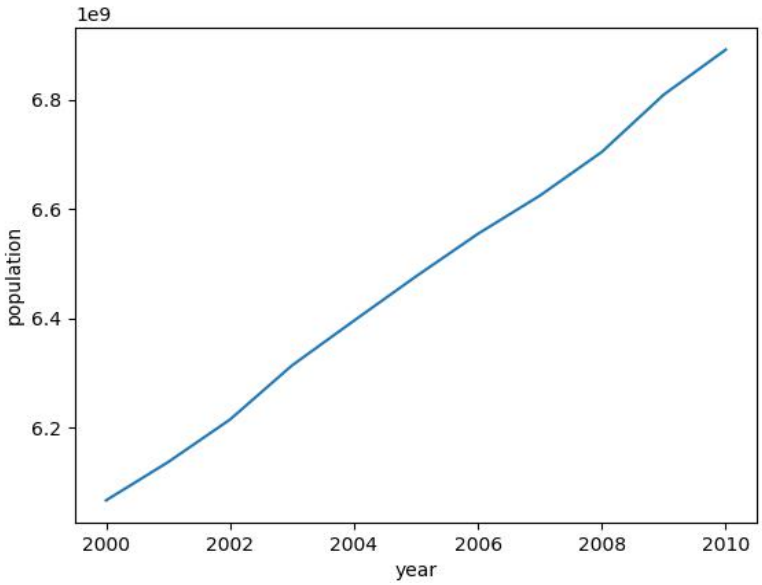
```
    years = [2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009,
2010]

    populations = [
        6067000000,
        6137000000,
        6215000000,
        6314000000,
        6396000000,
        6477000000,
        6555000000,
        6625000000,
        6705000000,
        6809972000,
        6892319000
    ]
```

Sử dụng thư viện matplotlib để vẽ đồ thị giữa các biến:

```
    import matplotlib.pyplot as plt
    plt.plot(years, populations)
    plt.xlabel('year')
```

```
plt.ylabel('population')
plt.show()
```



Đồ thị dân số thế giới qua các năm vẽ bằng matplotlib

Trên đồ thị, có thể thấy dân số thế giới tăng qua các năm theo một đường thẳng. Nếu y là dân số thế giới và x là các năm, mối quan hệ này có dạng:

$y \approx a.x + b$

Xác định hệ số của Linear Regression từ dữ liệu

Cần xác định các hệ số a, b khi có tập dữ liệu thể hiện mối quan hệ giữa biến vào x và biến ra y. Tập dữ liệu này được cho ở dạng N cặp giá trị (x_i, y_i) , i =1,2 ..N

Phương pháp dùng để xác định hệ số a, b là phương pháp bình phương tối thiểu. Phương pháp này tìm đường thẳng sao cho tổng bình phương sai lệch cua các điểm (x_i, y_i) với mô hình đường thẳng đó là nhỏ nhất:

$$sse = \sum_{i=1}^{N} (ax_i + b - y_i)^2 \longrightarrow \min$$

sse là kí hiệu của *sum square eror*

Các hệ số a, b được tìm bằng cách giải hệ phương trình tìm điểm cực tiểu của *sse:*

$$\frac{\partial sse}{\partial a} = 0$$
$$\frac{\partial sse}{\partial b} = 0$$

Nếu đặt :

$$S_{xx} = \sum x_i^2, \quad S_{xy} = \sum x_i y_i, \quad S_x = \sum x_i, \quad S_y = \sum y_i$$

Thì hệ phương trình tìm a, b là:

$$S_{xx}a + S_x b = S_{xy}$$

$$S_x a + Nb = S_y$$

Từ đó, chúng ta tìm được:

$$a = \frac{NS_{xy} - S_x S_y}{NS_{xx} - S_x^2}, \quad b = \frac{aS_x - S_y}{N}$$

Chương trình để tính hệ số a, b :

```
X = years
Y = populations

N = len(X)
Sxx, Sxy, Sx, Sy = 0, 0, 0, 0

for i in range(N):
    Sxx += X[i] * X[i]
    Sxy += X[i] * Y[i]
    Sx += X[i]
    Sy += Y[i]

a = (N*Sxy - Sx*Sy)/(N*Sxx - Sx*Sx)
b = (Sy - a*Sx)/N

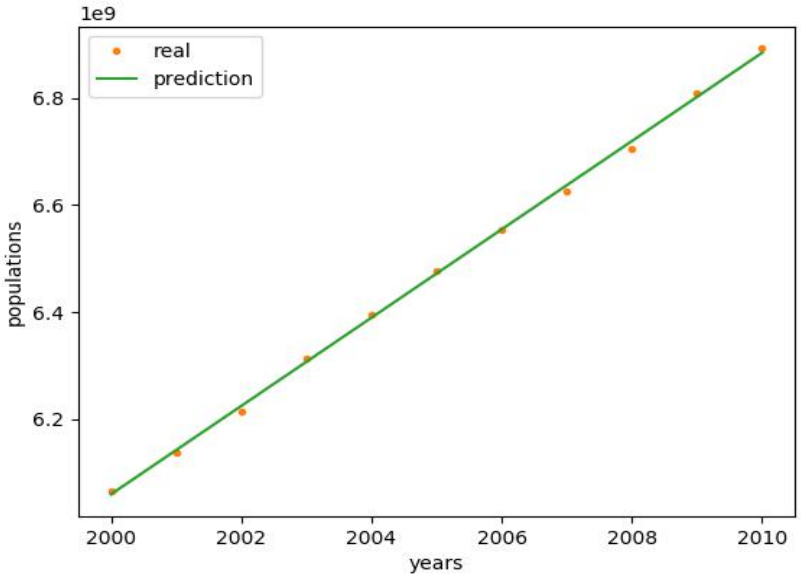
print('a=', a)
print('b=', b)
```

Kết quả tìm được:

a = 82.45 x 10^6, b = -158.84 x 10^9

Để kiểm tra, chúng ta vẽ đồ thị giữa dữ liệu thật và mô hình

```
Y_predict = [a*X[i]+b for i in range(N)]
plt.plot(X, Y, '.', label='real')
plt.plot(X, Y_predict, '-', label='prediction')
plt.xlabel('years')
plt.ylabel('populations')
plt.legend(loc='upper left')
plt.show()
```



Kết quả giữa mô hình và dữ liệu thật

Dùng mô hình để dự đoán

Sau khi đã có mô hình, chúng ta có thể dùng để dự đoán giá trị đầu ra với các đầu vào không có trong tập dữ liệu. Ví dụ, để dự đoán dân số thế giới trong các năm 2011 và 2012

```
print('2011 population : ', a*2011 + b)
print('2012 population : ', a*2012 + b)
```

Các giá trị dự đoán là:

- 2011 population : 6966816436
- 2012 poplation : 7049266281

Giá trị thật của dân số thế giới trong các năm 2011, 2012:

- 2011 real population : 6997998760
- 2011 real population : 7080072417

Sai số dự đoán tương ứng là 0.445% và 0.435%

Sử dụng thư viện Scikitlearn

Scikitlearn cho phép xây dựng mô hình Linear Regresion (và nhiều mô hình khác) một cách nhanh chóng.

```
from sklearn import linear_model

years = [2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009,
2010]

populations = [
   6067000000,
   6137000000,
   6215000000,
   6314000000,
```

```
    6396000000,
    6477000000,
    6555000000,
    6625000000,
    6705000000,
    6809972000,
    6892319000
]

X = [[x] for x in years]
Y = populations

model = linear_model.LinearRegression()
model.fit(X, Y)
a = model.coef_[0]
b = model.intercept_

print('a=', a)
print('b=', b)

print('2011 population : ', model.predict(2011)[0])
print('2012 population : ', model.predict(2012)[0])
```

Lưu ý là dữ liệu vào X cho mô hình Linear Regression của Scikitlearn là một mảng 2 chiều, vì mỗi phần tử x được xem là 1 vector thuộc tính. Chúng ta sẽ thấy rõ hơn điều này ở ví dụ sau.

Ví dụ 2:
Bảng dữ liệu sau thể hiện mối quan hệ giữa hàm lượng mỡ máu theo tuổi và cân nặng

Weight (kilograms)	Age (Years)	Blood fat content(mg/l)
84	46	354
73	20	190
65	52	405
70	30	263
76	57	451
69	25	302
63	28	288
72	36	385
79	57	402
75	44	365
27	24	209
89	31	290
65	52	346
57	23	254
59	60	395
69	48	434
60	34	220
79	51	374
75	50	308
82	34	220
59	46	311

67	23	181
85	37	274
55	40	303
63	30	244

Xây dựng mô hình Linear Regression để ước tính hàm lượng mỡ trong máu theo tuổi và cân nặng

$$y = a_1x_1 + a_2x_2 + b$$

Trong đó:

y : hàm lượng mỡ máu
x_1: tuổi
x_2 : cân nặng

Để thực hiện, chúng ta chia tập dữ liệu thành 2 phần : training set và test set theo tỉ lệ 70:30, sau đó dùng scikitlearn để dựng mô hình Linear regression cho 2 biến đầu vào.

```
from sklearn import linear_model

X = [[84,46], [73,20], [65,52], [70,30], [76,57],
     [69,25], [63,28], [72,36], [79,57], [75,44],
     [27,24], [89,31], [65,52], [57,23], [59,60],
     [69,48], [60,34], [79,51], [75,50], [82,34],
     [59,46], [67,23], [85,37], [55,40], [63,30]]

Y = [354, 190, 405, 263, 451,
     302, 288, 385, 402, 365,
     209, 290, 346, 254, 395,
     434, 220, 374, 308, 220,
     311, 181, 274, 303, 244 ]

N = len(X)
Ntrain = int(0.7*N)

Xtrain = X[:Ntrain]
Ytrain = Y[:Ntrain]

model = linear_model.LinearRegression()
model.fit(Xtrain, Ytrain)

Xtest = X[Ntrain:]
Ytest = Y[Ntrain:]
Ypredict = model.predict(Xtest)

for i in range(len(Xtest)):
    print('y=', Ytest[i], ', predict=', round(Ypredict[i]))
```

Kết quả chạy chương trình:

```
y= 374 , predict= 396.0
y= 308 , predict= 387.0
y= 220 , predict= 316.0
y= 311 , predict= 351.0
y= 181 , predict= 246.0
y= 274 , predict= 333.0
y= 303 , predict= 317.0
```

```
y= 244 , predict= 276.0
```

2. Logistic regression

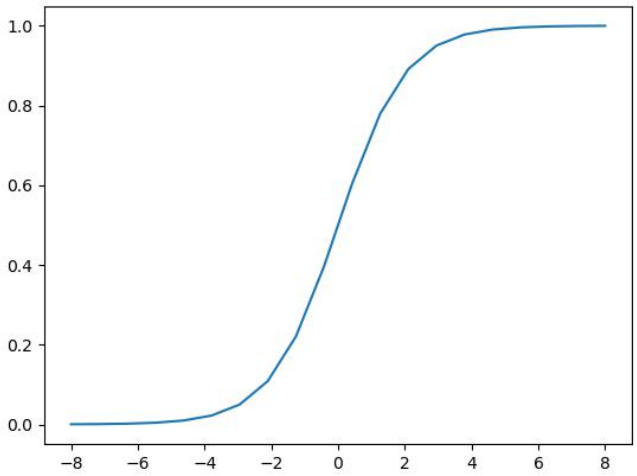
Logistic regression là thuật toán dùng để dự đoán một sự kiện có xảy ra hay không và xác suất xảy ra sự kiện là bao nhiêu.

Đầu vào của mô hình có thể có một hoặc nhiều biến số (tương tự Linear regression), đầu ra của mô hình là một giá trị nằm trong đoạn [0,1] thể hiện xác suất sẽ xảy ra sự kiện cần dự đoán.

Hàm logistic

Hàm logistic là hàm dùng để tạo ra giá trị trong khoảng [0,1] ở đầu ra của mô hình Logistic Regression

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



Đồ thị hàm logistic

Ý nghĩa của hàm logistic:

Nếu sự tăng giảm của một đại lượng dẫn đến khả năng xảy ra một sự kiện tăng hay giảm tương ứng thì hàm logistic là mô hình đơn giản nhất cho mối quan hệ giữa chúng.

Ví dụ :

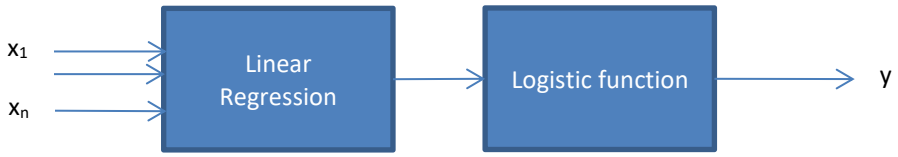
- Hàm lượng mỡ máu và nguy cơ mắc bệnh tim mạch
- Thu nhập hàng tháng của một người và khả năng người đó mua một loại sản phẩm
- Số ngày đến lớp của sinh viên và khả năng qua bài thi cuối kì

Biểu diễn toán học đầy đủ của Logistic Regression:

Có n biến số có thể đo được giá trị X_1, X_2, … X_n. Cần xác định xác suất xảy ra một sự kiện Y theo giá trị của các biến số đã biết. Nếu sự tăng giảm của mỗi biến số X_i dẫn đến khả năng sự kiện Y xảy ra tăng hoặc giảm một cách tương ứng, thì quan hệ giữa giá trị của các biến đầu vào và sự kiện cần dự đoán có thể được biểu diễn qua mô hình:

$$p(y) = \frac{1}{1 + e^{-(a_1 x_1 + a_2 x_2 + ... + a_n x_n)}}$$

Có thể xem mô hình Logistic Regression ở trên là nối tiếp của mô hình Linear Regression với một hàm logistic



Mô hình phân tách của Logistic Regression

Ví dụ 1:

Ước tính khả năng qua bài thi cuối kì theo thời gian học ôn thi của sinh viên. Nguồn : Wikipedia

Số giờ học	0.5	0.75	1	1.25	1.5	1.75	2	2.25	2.5	2.75	3	3.25	3.5	3.75	4	4.25	4.5	4.75	5	5.5
Qua kì thi	0	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1

Bảng trên là số giờ học ôn thi và kết quả thi của 20 sinh viên. Gọi x là số giờ học ôn thi, p là xác suất qua kì thi, thì mối quan hệ giữa x và p có thể được thể hiện qua mô hình:

$$p = \frac{1}{1 + e^{-(ax+b)}}$$

Chúng ta phải tìm các hệ số a, b dựa vào bảng dữ liệu đã cho.
Để thực hiện điều này, chúng ta dùng phương pháp bình phương cực tiểu, tương tự như trong Linear Regression.

Nếu p_i là xác suất dự đoán ứng với một giá trị đầu vào x_i, gọi y_i là kết quả thực sự (0/1) thì chúng ta mong muốn tối thiểu sai số:

$$E = \frac{1}{2}\sum (p_i - y_i)^2$$

Để có sai số tối thiểu thì đạo hàm của E theo các tham số a, b phải bằng 0, trong trường hợp này:

$$\frac{\partial E}{\partial a} = \sum x_i e_i p_i (1 - p_i)$$
$$\frac{\partial E}{\partial b} = \sum e_i p_i (1 - p_i)$$

Trong đó $e_i = p_i - y_i$, là các sai số thành phần.

Việc giải hệ phương trình trên là tương đối khó khăn, do đó chúng ta dùng phương pháp "di chuyển ngược hướng đạo hàm" (Gradient descent) để tìm các giá trị a,b làm sai số E đạt nhỏ nhất

Phương pháp Gradient Descent:

Phương pháp Gradient Descent là phương pháp tìm giá trị nhỏ nhất bằng phép lặp nhiều bước. Phương pháp này dựa trên kết quả của giải tích là một hàm số nhiều biến sẽ giảm giá trị khi các biến đầu vào thay đổi theo hướng ngược với hướng của đạo hàm tại vị trí đang xét:

$$f(X+\Delta X) < f(X) \text{ nếu } \Delta X = -k\frac{\partial f}{\partial X}, \text{ trong đó k > 0}$$

Nội dung phương pháp cụ thể như sau:

- Khởi tạo ngẫu nhiêu giá trị ban đầu cho các biến
- Với mỗi bước, thay đổi các biến một lượng tỉ lệ với đạo hàm của hàm số tại vị trí hiện tại. Hệ số tỉ lệ là số âm, độ lớn của hệ số tỉ lệ được gọi là **learning rate (lr)**
- Khi giá trị của hàm số không giảm thêm nữa thì dừng lại

Áp dụng phương pháp trên, chúng ta tìm hệ số a, b cho ví dụ trên như sau:

- Khởi tạo a, b
- Với mỗi bước, cập nhật giá trị mới cho a, b:

$$a \leftarrow a - lr.\sum x_i e_i p_i (1-p_i)$$
$$b \leftarrow b - lr.\sum e_i p_i (1-p_i)$$

Trong đó **lr** là giá trị của learning rate , thường chọn trong khoảng 0.001 đến 0.1

Chương trình xây dựng mô hình Logistic Regression cho ví dụ 1 như sau:

```
import math

data = [
                (0.50, 0),
                (0.75, 0),
                (1.00, 0),
                (1.25, 0),
                (1.50, 0),
                (1.75, 0),
                (1.75, 1),
                (2.00, 0),
                (2.25, 1),
                (2.50, 0),
                (2.75, 1),
                (3.00, 0),
                (3.25, 1),
                (3.50, 0),
                (4.00, 1),
                (4.25, 1),
                (4.50, 1),
                (4.75, 1),
                (5.00, 1),
                (5.50, 1)
        ]
```

```
n = len(data)
X = [data[i][0] for i in range(0,n)]
Y = [data[i][1] for i in range(0,n)]

epochs = 1000                       # epochs
lr = 0.01                           # Learning rate
a = 0
b = 0

for step in range(0, epochs):
   da, db = 0, 0
   sse = 0
   for i in range(n):
         x = X[i]
         y = Y[i]
         p = 1/(1+math.exp(-(b+a*x)))
         e = p - y
         da += e*p*(1-p)*x
         db += e*p*(1-p)
         sse += e*e

   a += -lr*da
   b += -lr*db
   rmse = math.sqrt(sse/n)
   print('Step %d: rmse=%0.2f, a=%0.2f, b=%0.2f'%(step, rmse, a, b))
```

Kết quả sau 1000 bước lặp:

```
a = 0.91, b = -2.41
```

Để kiểm tra lại, chúng ta so sánh kết quả giữa mô hình và giá trị thực:

```
P = [1/(1+math.exp(-(b + a*X[i]))) for i in range(0,n)]
print('x\t y\t Prob\t Predict')

for i in range(0,n):
   print('%0.2f\t %d\t %0.2f\t %d'%(X[i], Y[i], P[i], round(P[i])))
```

Kết quả:

```
x       y       Prob    Predict
0.50    0       0.12    0
0.75    0       0.15    0
1.00    0       0.18    0
1.25    0       0.22    0
1.50    0       0.26    0
1.75    0       0.31    0
1.75    1       0.31    0
2.00    0       0.36    0
2.25    1       0.41    0
2.50    0       0.47    0
2.75    1       0.52    1
3.00    0       0.58    1
3.25    1       0.63    1
3.50    0       0.68    1
4.00    1       0.77    1
4.25    1       0.81    1
4.50    1       0.84    1
4.75    1       0.87    1
```

```
5.00    1       0.89    1
5.50    1       0.93    1
```

Trong 20 mẫu, có 4 mẫu là mô hình cho kết quả khác với thực tế, độ khớp của mô hình là 80%

Sử dụng thư viện Scikitlearn

```
from sklearn.linear_model import LogisticRegression

data = [
                (0.50, 0),
                (0.75, 0),
                (1.00, 0),
                (1.25, 0),
                (1.50, 0),
                (1.75, 0),
                (1.75, 1),
                (2.00, 0),
                (2.25, 1),
                (2.50, 0),
                (2.75, 1),
                (3.00, 0),
                (3.25, 1),
                (3.50, 0),
                (4.00, 1),
                (4.25, 1),
                (4.50, 1),
                (4.75, 1),
                (5.00, 1),
                (5.50, 1)
        ]

n = len(data)
X = [data[i][0:1] for i in range(0,n)]
Y = [data[i][1] for i in range(0,n)]

model = LogisticRegression(C=5)
model.fit(X, Y)

print('a = ', model.coef_)
print('b = ', model.intercept_)

score = model.score(X, Y)
print('score = ', score)

P = model.predict_proba(X)
print('x\t y\t Prob\t Predict')

for i in range(0,n):
   x, y, p = X[i][0], Y[i], P[i][1]
   print('%0.2f\t %d\t %0.2f\t %d'%(x, y, p, round(p)))
```

Kết quả cũng tương tự như trong chương trình sử dụng Gradient Descent ở phía trên.

Ví dụ 2: Dự đoán kết quả nộp hộp sơ cao học tại một trường đại học. Nguồn : UCLA

Các sinh viên sau khi tốt nghiệp đại học có thể nộp hồ sơ để xin học tiếp chương trình cao học. Các thông tin có được từ hồ sơ:

```
gre  gpa   rank
380  3.61    3
660  3.67    3
800  4.00    1
640  3.19    4
520  2.93    4
760  3.00    2
```

Trong đó : gre là điểm bài thi chuyên ngành, gpa là điểm trung bình đại học, rank là xếp hạng của trường đại học (cao nhất là 1, thấp nhất là 4). Dữ liệu đầy đủ có thể download từ file binary.csv

Cần xây dựng mô hình để dự đoán khả năng một sinh viên có thể được nhận vào chương trình cao học hay không căn cứ các thông tin đã có trong hồ sơ.

Trước hết chúng ta download file dữ liệu về, sau đó tìm cách tạo ra các bộ giá trị X, Y cho mô hình Logistic Regression. Để đọc file csv, cách nhanh nhất là sử dụng thư viện pandas.

```
import pandas as pd

df = pd.read_csv('binary.csv')
X = df[['gre','gpa', 'rank']].values
Y = df['admit'].values
```

Sau khi đã có bộ dữ liệu, chúng ta chia ra thành tập train và tập test theo tỉ lệ 70:30

```
N = len(X)
Ntrain = int(0.7 * N)
Xtrain = X[:Ntrain]
Ytrain = Y[:Ntrain]
Xtest = X[Ntrain:]
Ytest = Y[Ntrain:]
```

Tiếp theo, chúng ta dùng Scikitlearn để tạo mô hình Logistic Regression như ở ví dụ 1

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(Xtrain, Ytrain)
print('train score = ', model.score(Xtrain, Ytrain))

Yval = model.predict(Xtest)
print('testcore = ', model.score(Xtest, Ytest))
```

Kết quả cho thấy độ khớp của mô hình với tập train và test chỉ khoảng 70%. Điều này có nghĩa mô hình không thực sự hợp lý. Việc đánh giá các sinh viên theo kết quả điểm là không đầy đủ, hội đồng tuyển sinh sẽ xem xét cả các thông tin khác trong hồ sơ không thể hiện dưới dạng các dữ liệu số như ở trên.

Ví dụ 3:

Dự đoán khả năng mua hàng của khách hàng. Nguồn : LogisticRegressionAnalysis.com

Một tạp chí cần dự đoán khả năng một khách hàng với các thông tin nhân khẩu biết trước sẽ mua ấn bản của tạp chí hay không.

Income	Is Female	Is Married	Has College	...
24000	1	0	1	...
75000	1	1	1	...
46000	1	1	0	...
70000	0	1	0	...

Các thông tin mà tạp chí biết về các khách hàng gồm : thu nhập, giới tình, tình trạng kết hôn, học vấn, ... Chi tiết có trong file dữ liệu có thể tải về KidCreative.csv. Cần xây dựng mô hình Logistic Regression để dự đoán khả năng mua tạp chí của một khách hàng.

Tương tự như ở ví dụ 2, chúng ta download file dữ liệu về và dùng pandas đọc file để tách lấy các bộ giá trị X, Y cho mô hình

```
import pandas as pd
from sklearn.linear_model import LogisticRegression

df = pd.read_csv('KidCreative.csv')
values = df.values
Y = values[:, 1]
X = values[:, 2:]
```

Các bước còn lại tương tự ví dụ 2:

```
N = len(X)
Ntrain = int(0.7 * N)
Xtrain = X[:Ntrain]
Ytrain = Y[:Ntrain]
Xtest = X[Ntrain:]
Ytest = Y[Ntrain:]

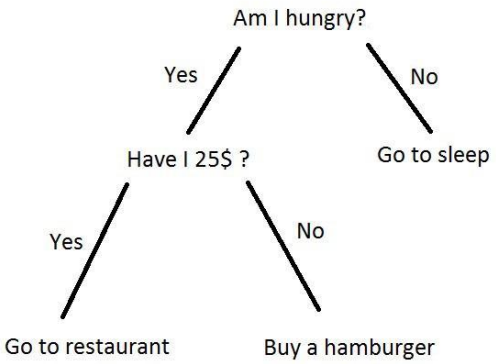
model = LogisticRegression()
model.fit(Xtrain, Ytrain)
print('train score = ', model.score(Xtrain, Ytrain))

Yval = model.predict(Xtest)
print('testcore = ', model.score(Xtest, Ytest))
```

Trong ví dụ này, độ khớp của mô hình trên tập train và test đều đạt trên 88%, như vậy mô hình ước lượng là khá hợp lí.

3. Decision tree

Decision tree là phương pháp ra quyết định theo các nhánh của một cây quyết định



Một ví dụ về Decision Tree

Mỗi nốt cuối (lá) của cây quyết định là một trong những quyết định từ tập cho trước. Mỗi nốt trung gian là một trạng thái cần đưa ra quyết định tiếp theo.

Cây quyết định có thể được xây dựng bằng kinh nghiệm (như ở hình vẽ trên), hoặc được dựng từ một tập dữ liệu cho trước. Giống như các phương pháp khác trong Machine Learning, việc dựng ra cây quyết định được thực hiện theo nguyên lý làm cho độ khớp giữa mô hình và thực tế là cao nhất.

Nếu tập dữ liệu là giới hạn, về lí thuyết có thể tạo ra cây quyết định khớp hoàn toàn với tất cả các giá trị trong tập dữ liệu (cho độ dài của cây tăng lên cho đến khi khớp tất cả các mẫu). Tuy nhiên, nếu cây có độ dài lớn thì khi dự đoán kết quả của một mẫu dữ liệu không nằm trong tập huấn luyện, tỉ lệ sai sẽ khá lớn. Hiện tượng này gọi là **overfitting** có nghĩa độ khớp với tập huấn luyện rất cao nhưng khi dùng mô hình trong thực tế lại gặp sai số lớn.

Để tránh hiện tượng overfitting khi dựng cây quyết định từ một tập dữ liệu, thông thường độ dài tối đa của cây sẽ bị giới hạn.

Ví dụ 1:

Xây dựng cây quyết định để cho biết có nên ra ngoài chơi thể thao không tùy theo điều kiện thời tiết
Nguồn : Decision Trees for Classification: A Machine Learning Algorithm

Day	Outlook	Temperature	Humidity	Wind	Play Golf
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Khi dựng cây quyết định, tại bước đầu chúng ta cần chọn gốc của cây. Nguyên lý thực hiện là chọn gốc của cây sao cho ở mỗi trạng thái được phân tách ra (2 nốt trái-phải) , có thể sớm đi đến quyết định cuối cùng nhất. Trong trường hợp lí tưởng, có thể ra quyết định ngay ở nhánh bên dưới.

Cụ thể, chúng ta dùng khái niệm Entropy để đo mức độ không đồng nhất của một trạng thái:

$$H(S) = \sum_x p(x) \log_2(\frac{1}{p(x)})$$

Trong đó x là các giá trị có thể của đầu ra của đối tượng, p(x) là xác suất đầu ra nhận giá trị bằng x.

Ý nghĩa của Entropy:

Entropy thể hiện độ bất định của một trạng thải. Trạng thái có entropy càng lớn sẽ càng khó đưa ra một quyết định hợp lý (có nhiều lựa chọn), trạng thái có entropy thấp có thể dễ đưa ra một quyết định hợp lý.

Với bài toán quyết định 0/1 hay Yes/No, trạng thái nào có thể đưa ra quyết định rõ ràng (p(0)=1 hoặc p1(1)=1) thì Entropy bằng 0, trạng thái nào mà khả năng đưa ra 2 quyết định là ngang nhau (50/50) thì Entropy là 1, lúc đó độ bất định là cao nhất.

Ngoài Entropy, có thể sử dụng Gini để đo độ không đồng nhất/độ bất định của một trạng thái:

$$G(S) = 1 - \sum_x p_x^2$$

Gini và Entropy khá tương đồng nhau và có thể dùng thay thế lẫn nhau trong việc xây dựng cây quyết định.

Trở lại ví dụ trên, trong 14 ngày có 9 ngày quyết định là Yes, 5 ngày quyết định là No, Entropy ở trạng thái gốc là:

$$S = \frac{9}{14} \log_2(\frac{14}{9}) + \frac{5}{14} \log_2(\frac{14}{5}) = 0.94$$

Mong muốn của chúng ta là sau khi chọn ra gốc của cây quyết định, 2 trạng thái bên trái và phải có độ bất định nhỏ nhất, để có thể sớm đưa ra quyết định cuối cùng. Để đo mức hợp lý của việc phân chia cây, chúng ta dùng độ giảm Entropy/Gini trước và sau khi phân chia:

$$\Delta H = H(S) - \sum_i p_i H(S_i)$$

Trong đó S_i là các trạng thái sau khi phân chia (2 trạng thái bên trái và bên phải), p_i là xác suất xảy ra của mỗi trạng thái, được tính bằng tỉ lệ của số mẫu trong tập dữ liệu thuộc về mỗi trạng thái chia cho tổng số mẫu dữ liêu ở trạng thái cha.

Giả sử chúng ta chọn gốc dựa trên trạng thái của gió (Wind=Weak/Strong). Số mẫu trong tập dữ liệu nằm ở 2 trạng thái này là 8 và 6:

Wind = Weak	Wind = Strong	Total
8	6	14

Ở trạng thái Wind=Weak, có 6 mẫu cho quyết định Yes, 2 mẫu cho quyết định No, như vậy Entropy ở trạng thái này là:

$$H(S_{WeakWind}) = \frac{6}{8}\log_2(\frac{8}{6}) + \frac{2}{8}\log_2(\frac{8}{2}) = 0.811$$

Tương tự, ở trạng thái Wind=Strong, số mẫu cho quyết định Yes là 3, số mẫu cho quyết định No là 3, Entropy ở trạng thái này là:

$$H(S_{StrongWind}) = \frac{3}{6}\log_2(\frac{6}{3}) + \frac{3}{6}\log_2(\frac{6}{3}) = 1.0$$

Mức giảm Entropy sau khi phân chia:

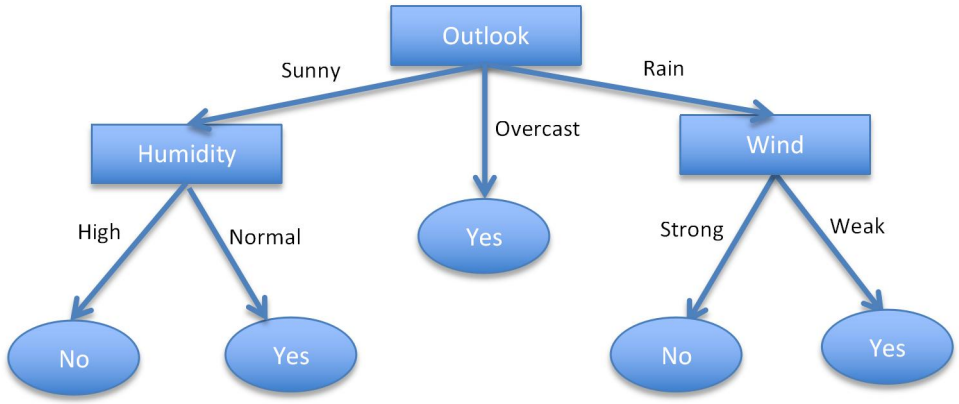
$$\Delta H = H(S) - (\frac{8}{14}H(S_{WeakWind}) + \frac{6}{14}H(S_{StrongWind})) = 0.048$$

Tương tự, chúng ta tính mức giảm Entropy nếu chọn gốc phân chia theo các biến đầu vào khác:

- Phân chia theo Outlook : ΔH = 0.246
- Phân chia theo Temperature : ΔH = 0.029
- Phân chia theo Humidity : ΔH = 0.151

Như vậy ở bước đầu, chọn phân chia theo Outlook là hợp lí nhất, vì làm giảm Entropy nhiều nhất.

Sau khi đã chọn gốc là Outlook, chúng ta tiếp tục thực hiện phân chia ở các nhánh dưới tương tự theo nguyên lý trên, kết quả cuối cùng sẽ thu được cây quyết định như sau:



Cây quyết định được dựng ra trong ví dụ 1

Ví dụ 2: Bài toán Iris

Iris là một loài hoa với 3 chủng khác nhau : setosa, versicolor, virginica. Mỗi chủng có kích thước các phần của bông hoa khác nhau. File iris.csv chứa 150 mẫu giá trị kích thước các phần bông hoa của 3 chủng hoa trên.

sepal_length	sepal_width	petal_length	petal_width	species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa
7	3.2	4.7	1.4	versicolor
6.4	3.2	4.5	1.5	versicolor
6.9	3.1	4.9	1.5	versicolor
5.5	2.3	4	1.3	versicolor
6.5	2.8	4.6	1.5	versicolor
6.3	3.3	6	2.5	virginica
5.8	2.7	5.1	1.9	virginica
7.1	3	5.9	2.1	virginica
6.3	2.9	5.6	1.8	virginica
6.5	3	5.8	2.2	virginica

Một số mẫu dữ liệu trong file iris.csv

Cần dựng một cây quyết định để phân biệt 3 chủng hoa trên dựa vào kích thước các bộ phận trên bông hoa.

Với bài toán này, giá trị đầu ra có thể nhận là một trong 3 chủng hoa, các biến đầu vào là các biến liên tục, để xử lý chúng ta chia khoảng giá trị của chúng theo các mốc, sau đó xử lý tương tự như các biến rời rạc.

Nếu xây dựng cây quyết định bằng tay như ở ví dụ 1 sẽ mất nhiều thời gian. Chúng ta sử dụng thư viện scikitlearn để dựng cây quyết định. Dữ liệu được chia thành 2 tập training/test theo tỉ lệ 70/30 để kiểm tra mô hình có bị overfitting không.

```
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

def get_class(species):
    return {
        'setosa': 0,
        'versicolor': 1,
        'virginica': 2
    }[species]

df = pd.read_csv('iris.csv')
values = df.values
X = np.array(values[:, :-1], dtype=np.float)
Y = [get_class(species) for species in values[:, -1]]

Xtrain = np.concatenate((X[:35], X[50:85], X[100:135]))
Ytrain = Y[:35] + Y[50:85] + Y[100:135]
Xtest = np.concatenate((X[35:50], X[85:100], X[135:]))
Ytest = Y[35:50] + Y[85:100] + Y[135:]

model= DecisionTreeClassifier()
model.fit(Xtrain, Ytrain)
```

```
    print('train score = ', model.score(Xtrain, Ytrain))
    print('test score = ', model.score(Xtest, Ytest))
```

Kết quả độ khớp trên tập training, test là 100% và 97.7%.
Trong một số trường hợp, một cây quyết định sẽ cho kết quả không đủ chính xác. Để có kết quả chính xác hơn, có thể tạo nhiều cây quyết định và tổng hợp kết quả của các cây lại (phương pháp Random Forest)

Hiển thị cây quyết định dưới dạng hình ảnh:

Sau khi đã dựng xong cây quyết định, chúng ta có thể vẽ hình ảnh của cây để quan sát:

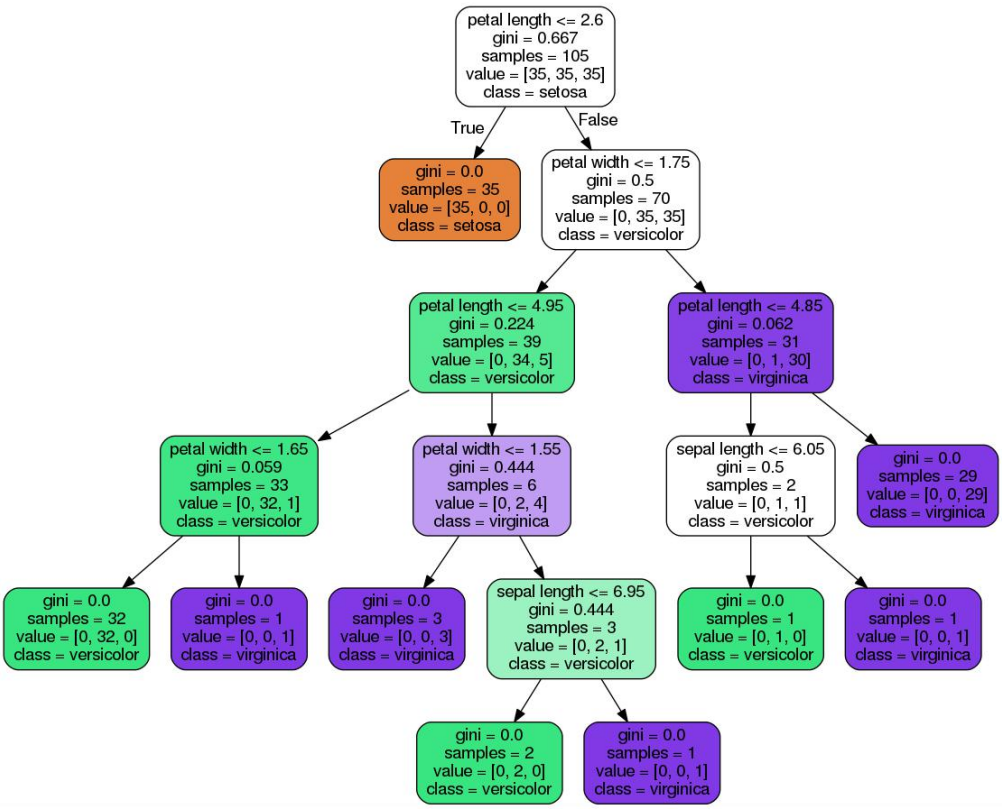
```
    import pydotplus
    import sklearn

    class_names=['setosa', 'versicolor', 'virginica']
    feature_names=['sepal length','sepal width','petal length','petal
width']

    dot_data = sklearn.tree.export_graphviz(model, out_file=None,
                          feature_names=feature_names,
                          class_names=class_names,
                          filled=True, rounded=True)

    graph = pydotplus.graph_from_dot_data(dot_data)
    f = open('tree.png', 'wb')
    f.write(graph.create_png())
    f.close()
```

Ảnh cây quyết định được lưu trong file tree.png có dạng như sau



Ảnh của cây quyết định trong ví dụ iris

4. K-nearest neighbour

Thuật toán K-nearest neighbour dựa vào các bộ dữ liệu trong tập dữ liệu mẫu mà khoảng cách đến bộ dữ liệu đang cần xử lý là ngắn nhất, để đưa ra kết quả . Đầu tiên chọn ra k mẫu trong tập dữ liệu mẫu gần nhất với bộ dữ liệu đang cần xử lý, sau đó tổng hợp kết quả của k mẫu này lại (lấy trung bình, chọn giá trị xuất hiện nhiều nhất …)

Ví dụ: Sử dụng tập dữ liệu iris ở phần trước, xây dựng chương trình phân loại hoa dựa trên thuật toán k-nearest neighbour với k=3

```
import numpy as np
import pandas as pd

df = pd.read_csv('iris.csv')
values = df.values
X = np.array(values[:, :-1], dtype=np.float)
Y = values[:, -1]

Xtrain = np.concatenate((X[:35], X[50:85], X[100:135]))
Ytrain = np.concatenate((Y[:35], Y[50:85], Y[100:135]))
Xtest = np.concatenate((X[35:50], X[85:100], X[135:]))
Ytest = np.concatenate((Y[35:50], Y[85:100], Y[135:]))

Ntrain = len(Xtrain)
Ntest = len(Xtest)

K = 3

def predict(x):
    distances = [(np.sum((x-Xtrain[i])*(x-Xtrain[i])), i)
                    for i in range(Ntrain)]

    distances = sorted(distances)
    y_samples = [Ytrain[distances[i][1]] for i in range(K)]

    predict = None
    Nmax = 0
    map_counts = {}

    for species in y_samples:
        map_counts[species] = map_counts.get(species,0) + 1
        if map_counts[species] > Nmax:
            predict = species
            Nmax = map_counts[species]

    return predict

Ypredict = [predict(Xtest[i]) for i in range(Ntest)]
Ncorrect = len([i for i in range(Ntest) if Ypredict[i] == Ytest[i]])
print('test score = ', Ncorrect/Ntest)
```

Sử dụng thư viện scikitlearn

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
```

```
df = pd.read_csv('iris.csv')
values = df.values
X = np.array(values[:, :-1], dtype=np.float)
Y = values[:, -1]

Xtrain = np.concatenate((X[:35], X[50:85], X[100:135]))
Ytrain = np.concatenate((Y[:35], Y[50:85], Y[100:135]))
Xtest = np.concatenate((X[35:50], X[85:100], X[135:]))
Ytest = np.concatenate((Y[35:50], Y[85:100], Y[135:]))

Ntrain = len(Xtrain)
Ntest = len(Xtest)

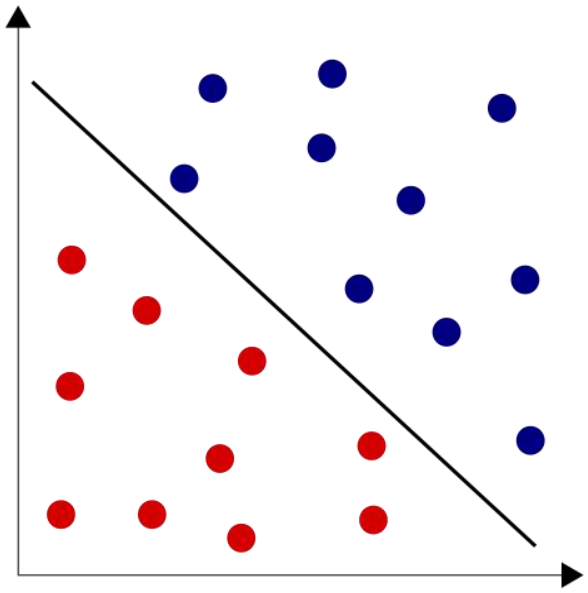
model = KNeighborsClassifier(n_neighbors=3)
model.fit(Xtrain, Ytrain)

score = model.score(Xtest, Ytest)
print('test score = ', score)
```

5. Support Vector Machine (SVM)

Support Vector Machine (SVM) là thuật toán phân loại 2 nhóm đối tượng bằng cách tìm một đường thẳng (mặt phẳng) để phân chia không gian thuộc tính thành 2 nửa sao cho các mẫu của 2 nhóm nằm ở 2 phía của đường thẳng và khoảng cách từ các điểm thuộc tính đến đường thẳng phân chia là lớn nhất.

SVM có thể không dùng đường thẳng phân chia mà dùng một số dạng đường cong (kernel) để phân chia 2 nhóm. Trong phần này, chúng ta chỉ quan tâm đến dạng phân chia sử dụng đường thẳng.



Ảnh minh họa SVM. Nguồn : Wikipedia

Trong trường hợp các điểm thuộc tính của 2 nhóm không phân tách được ra khỏi nhau và không tồn tại đường thẳng phân chia để toàn bộ mỗi nhóm ở một phía của đường thẳng, SVM sẽ tìm đường thẳng để sai số phân loại là nhỏ nhất (gần giống cách làm của Linear Regression).

Ví dụ 1:

Phân tách 2 chủng versicolor và virginica trong tập iris dựa trên 2 thuộc tính petal_length và petal_width.

Tương tự như ví dụ ở các mục trước, đầu tiên chúng ta chuẩn bị dữ liệu cho các tập train và test:

```
import pandas as pd

df = pd.read_csv('iris.csv')
values = df.values

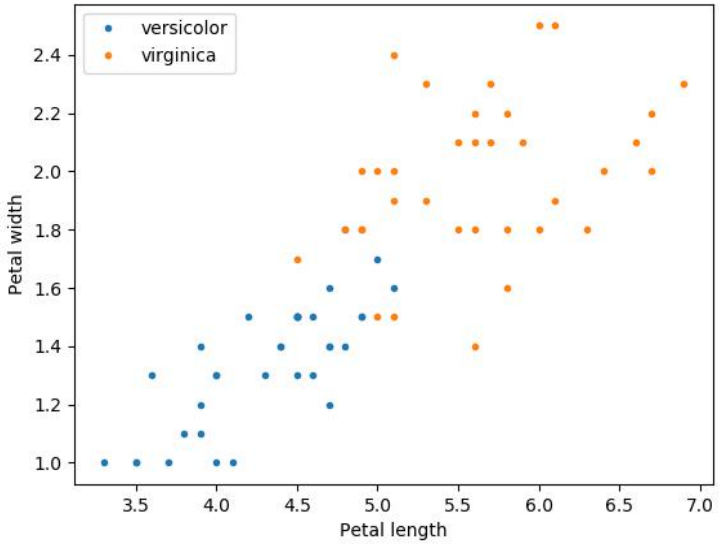
X1 = values[50:100, 2:4]
X2 = values[100:150, 2:4]

X1train, X1test = X1[:35], X1[35:]
X2train, X2test = X2[:35], X2[35:]
```

Với 2 thuộc tính của đối tượng, chúng ta có thể biểu diễn các mẫu dữ liệu của 2 nhóm trên mặt phẳng:

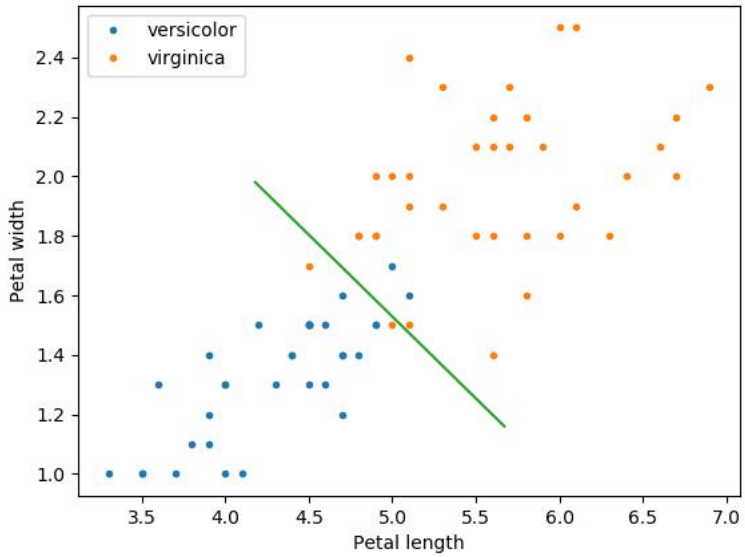
```
import matplotlib.pyplot as plt

plt.plot(X1train[:,0], X1train[:,1], '.', label='versicolor')
plt.plot(X2train[:,0], X2train[:,1], '.', label='virginica')
plt.xlabel('Petal length')
plt.ylabel('Petal width')
plt.legend(loc='upper left')
plt.show()
```



Biểu diễn phân bố thuộc tính 2 nhóm trên mặt phẳng

Dựa vào đồ thị, có thể thấy 2 nhóm không tách biệt hẳn nhau, chúng ta tìm đường thẳng phân chia để sai số nhỏ nhất. Có thể kẻ một đường thẳng gần đúng ở vùng giữa màn hình. Dùng chuột di chuyển và quan sát tọa độ (x,y) ở phía dưới bên phải cửa sổ matplotlib, chúng ta có thể tìm được đường thẳng gần đúng bằng cách định vị 2 điểm bất kì, ví dụ 2 điểm (4.65 , 1.8) và (5.36, 1.42)



Dùng tay xác đinh gần đúng đường thẳng phân chia

Dùng phương trình đường thẳng phân chia để làm hàm dự đoán:

```
def predict(petal_length, petal_width):
   distance = (petal_width-1.16)/(1.98-1.16) + \
                   (petal_length-5.67)/(5.67-4.18)
   return 1 if distance > 0 else 0

Y1predict = [predict(x[0], x[1]) for x in X1test]
Y2predict = [predict(x[0], x[1]) for x in X2test]

N1correct = len([1 for i in range(len(X1test)) if Y1predict[i] == 0])
N2correct = len([1 for i in range(len(X2test)) if Y2predict[i] == 1])
Ncorrect = N1correct + N2correct

print('score = ', Ncorrect/(len(X1test)+len(X2test)))
```

Ví dụ 2:

Dùng toàn bộ 4 thuộc tính để xây dựng mô hình SVM. Sử dụng thư viện scikitlearn.

```
import numpy as np
import pandas as pd
from sklearn import svm

def get_class(species):
    return {
        'setosa': 0,
        'versicolor': 1,
        'virginica': 2
    }[species]

df = pd.read_csv('iris.csv')
values = df.values
X = np.array(values[:, :-1], dtype=np.float)
Y = [get_class(species) for species in values[:, -1]]

Xtrain = np.concatenate((X[:35], X[50:85], X[100:135]))
Ytrain = Y[:35] + Y[50:85] + Y[100:135]
```

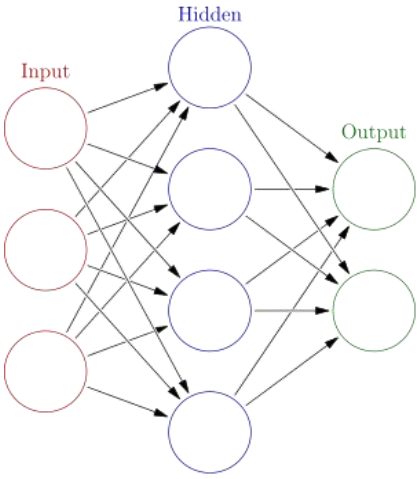
```
Xtest = np.concatenate((X[35:50], X[85:100], X[135:]))
Ytest = Y[35:50] + Y[85:100] + Y[135:]

model = svm.SVC(kernel='linear')
model.fit(Xtrain, Ytrain)

print('test score = ', model.score(Xtest, Ytest))
```

6. Neural Network

Neural network là hệ thống xử lý thông tin gồm nhiều lớp (layer), kết quả xử lý của lớp trước là đầu vào của lớp sau.

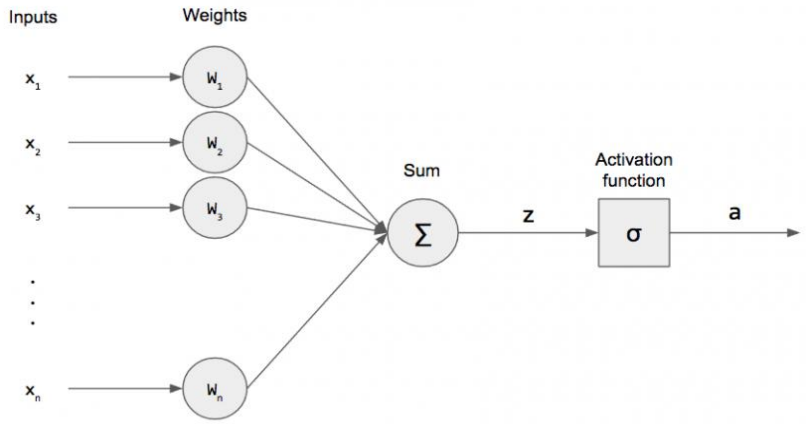


Cấu trúc của Neural Network. Nguồn : Wikipedia

Đầu vào của neural network là các giá trị đã biết, đầu ra của neural network là giá trị cần dự đoán. Giữa đầu vào và đầu ra là các lớp ẩn (hidden layers)

Perceptron:

Perceptron là mô hình đơn giản nhất của neural network, chỉ có 1 lớp với 1 neuron xử lý thông tin.



Cấu trúc của một Perceptron

Một perceptron có các đầu vào x_1, x_2, …., x_n và có một đầu ra y. Cách xử lý thông tin của Perceptron như sau:

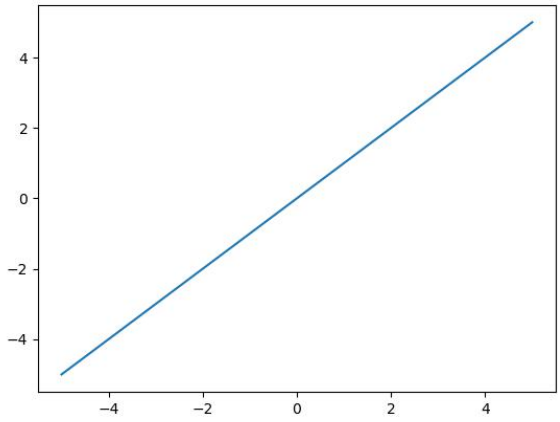
- Mỗi đầu vào được gắn một trọng số , thể hiện mức độ quan trọng của đầu vào đó tới kết quả ở đầu ra
- Các đầu vào được nhân với trọng số tương ứng, sau đó các giá trị này được cộng lại
- Giá trị tổng được đi qua một hàm biến đổi, gọi là "Activation function", giá trị kết quả của hàm này là đầu ra của hệ thống

Mô tả toán học của Perceptron:

$$y = \sigma(a_1 x_1 + a_2 x_2 + ... + a_n x_n)$$

Với các hàm σ khác nhau, Perceptron trở thành các hệ thống mà chúng ta đã biết:

- σ(x) = x : Hàm identity,

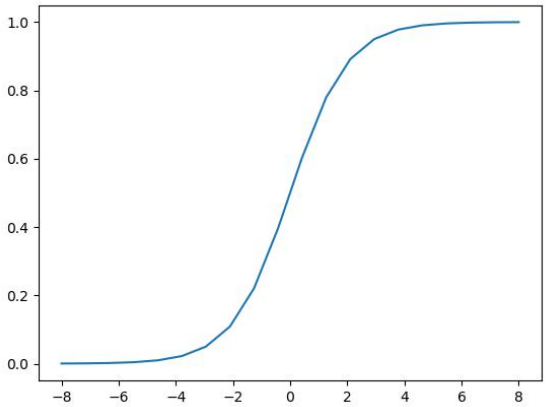


Đồ thị hàm identity

Khi đó, Perceptron trở thành mô hình Linear Regression

- σ(x) = sigmoid(x):

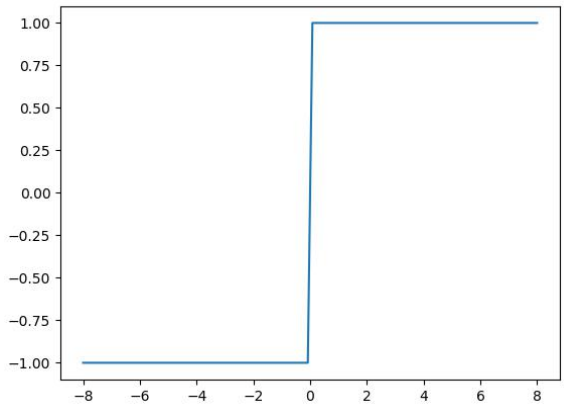
$$sigmoid(x) = \frac{1}{1+e^{-x}}$$



Đồ thị hàm sigmoid

Khi đó, Perceptron trở thành mô hình Logistic Regression

- σ(x) = sign(x) : Hàm dấu:



Đồ thị hàm sign

Khi đó, Perceptron trở thành mô hình SVM

Đầu ra của Perceptron có thể là giá trị liên tục hay giá trị rời rạc. Nếu đầu ra là giá trị liên tục, chúng ta có bài toán xấp xỉ hàm giá trị. Nếu đầu ra là giá trị rời rạc, chúng ta có bài toán phân loại dữ liệu.

Perceptron cho bài toán phân loại:

Trong bài toán phân loại, đầu ra của mô hình là chỉ số của nhóm mà đối tượng thuộc về. Hàm activation cho bài toán phân loại thường là hàm sigmoid, vì giá trị của hàm nằm trong đoạn [0,1] do đó các giá trị 0 hay 1 được dùng để đặc trưng cho chỉ số của 2 nhóm cần phân biệt

Ngoài hàm sigmoid, có thể dùng hàm tanh(hay tansig):

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

Trong bài toán phân loại mà số nhóm lớn hơn 2, hàm activation thường sử dụng là hàm softmax. Hàm này thực chất là một bộ k hàm con, trong đó k là số nhóm đối tượng:

$$f_i(x_1, x_2, ..., x_k) = \frac{e^{x_i}}{e^{x_1} + e^{x_2} + ... + e^{x_k}}$$

Ý nghĩa của hàm softmax là tạo ra phân phối tỉ lệ trên k nhóm sao cho tổng các tỉ lệ thành phần bằng 100%. Hàm softmax có thể xem là mở rộng của hàm sigmoid cho bài toán phân loại với số nhóm lớn hơn 2.

Nếu sử dùng hàm softmax là hàm activation thì đầu ra của hệ thống phải mã hóa theo kiểu một vector k thành phần, trong đó vector mã hóa của nhóm i có giá trị bằng 1 tại vị trí i, các vị trí còn lại đều bằng 0, ví dụ:

- Nhóm 0 : (1, 0, 0, 0, …)
- Nhóm 1: (0, 1, 0, 0 , …..)
- Nhóm 2 : (0, 0, 1, 0, …..)

Cách mã hóa đầu ra ở trên gọi là **“One Hot encoding”**

Perceptron cho bài toán xấp xỉ giá trị hàm:

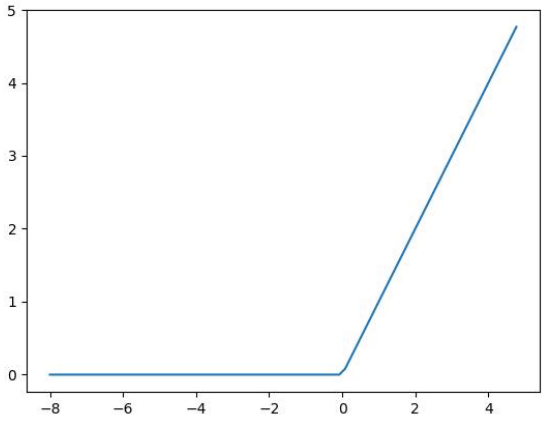
Trên thực tế có nhiều hệ thống mà đầu ra là một hàm số phức tạp của nhiều đầu vào mà việc xây dựng công thức chính xác là khó khăn. Trong trường hợp đó, Neural Network được dùng để tạo ra một mô hình có thể xấp xỉ hàm giá trị của đối tượng. Từ một tập dữ liệu các tham số vào-ra của hệ thống, người ta chỉnh định các tham số của Neural Network để sai số giữa mô hình và thực tế là nhỏ nhất. Quá trình này gọi là quá trình huấn luyện mạng Neuron, tương tự quá trình huấn luyện các mô hình đã được trình bày ở các phần trước.

Các hàm activation được dùng trong bài toán xấp xỉ:

- Hàm identity σ(x) =x . Trong trường hợp này mô hình Neural Network trở thành Linear Regression

- Hàm σ(x) = relu(x) :

$$relu(x) = \begin{cases} x & ,x >= 0 \\ 0 & ,x < 0 \end{cases}$$

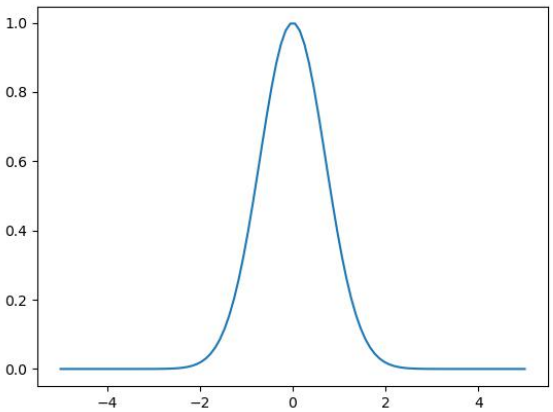


Đồ thị hàm relu

Ý nghĩa của việc dùng hàm relu là tìm cách xấp xỉ hàm giá trị bằng từng đoạn tuyến tính, thay vì chỉ dùng một đường thẳng/mặt phẳng duy nhất để xấp xỉ hàm giá trị như trong trường hợp của Linear Regression. Việc xấp xỉ từng đoạn này sẽ cho kết quả chính xác hơn.

- Hàm "radial basis function" σ(x) = rbf(x):

$$rbf(x) = e^{-\alpha(x-c)^2}$$



Đồ thị hàm rbf

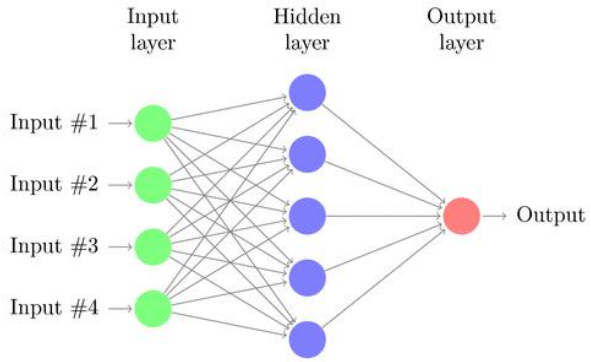
Việc dùng hàm rbf cũng gần giống như dùng hàm relu, tức tìm cách chia không gian giá trị thành nhiều miền nhỏ và xấp xỉ trên các miền đó. Hàm rbf được dùng cho bài toán xấp xỉ với nhiều tham số đầu vào, khi đó mỗi miền giá trị là một hình cầu.

Mạng neuron với nhiều lớp (Multilayer Perceptron - MLP):

Nếu mạng Neuron chỉ có một lớp (Perceptron) thì khả năng phân loại hay xấp xỉ hàm sẽ không đáp ứng được nhiều trường hợp trong thực tế. Để tăng độ chính xác cho mạng Neuron, có thể dùng nhiều lớp ghép nối tiếp nhau.

Với các bài toán phân loại hay xấp xỉ hàm mà số đầu vào tương đối nhỏ (từ 10 trở lại) thì mạng Neuron 2 lớp có thể đáp ứng được trong đa số các trường hợp.

Trong trường hợp này, ở lớp giữa của mạng neuron sẽ là N_h (number of hidden) neuron, mỗi neuron có cấu trúc tương đương với một perceptron. Sau đó, kết quả của N_h neuron này sẽ được tổng hợp lại ở một lớp tiếp theo (output layer) để cho ra kết quả cuối cùng.



Cấu hình thông dụng của Neural Network

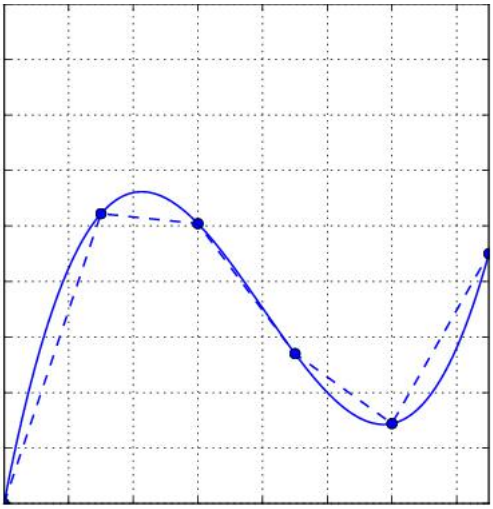
Ở cấu hình thông dụng này, số đầu vào và đầu ra đã cố định (yêu cầu của bài toán), chỉ có số neuron ở lớp giữa (N_h) là tham số có thể lựa chọn. Nếu chọn N_h nhỏ thì mô hình sẽ không khớp được chính xác với tập dữ liệu huấn luyện (hiện tượng **underfitting**). Nếu chọn N_h lớn thì mô hình sẽ rất khớp với tập dữ liệu huấn luyện, nhưng khi sử dụng trong thực tế thì xuất hiện sai số bất thường (**overfitting**)

Ý nghĩa của việc dùng nhiều neuron:

Với bài toán xấp xỉ hàm giá trị:

Có thể hiểu việc dùng nhiều Neuron là để chia không gian giá trị thành nhiều miền nhỏ, ở mỗi miền sẽ có một Neuron đóng vai trò chính trong việc xấp xỉ hàm giá trị trong miền đó.

Ví dụ, nếu dùng hàm relu là hàm activation thì việc dùng N_h neuron sẽ tạo ra một mô hình mà đầu ra là một hàm gồm N_h đoạn thẳng ghép lại với nhau:



Xấp xỉ hàm giá trị bất kì bằng nhiều neuron với hàm activation relu

Về lí thuyết, có thể xấp xỉ hàm với độ chính xác tùy ý bằng cách tăng số đoạn nhỏ (tương ứng với N_h).

Với bài toán phân loại:

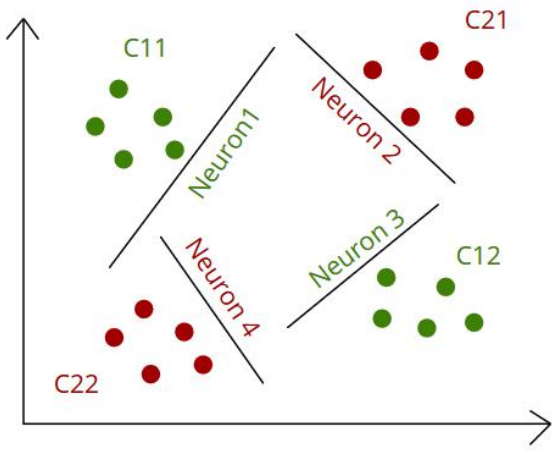
Có thể hiểu việc dùng nhiều Neuron tương đương với việc dùng nhiều SVM để phân tách ra các nhóm con của mỗi nhóm dữ liệu, sau đó tổng hợp kết quả lại. Việc này giúp khắc phục nhược điểm của SVM là không phân tách được 2 nhóm dữ liệu đan xen vào nhau (như chúng ta đã gặp ở ví dụ trong phần nói về SVM)

Cụ thể ở lớp ẩn (hidden layer):

- Neuron 1 : Phân tách nhóm con $C_{11} \subset C_1$ với phần còn lại của không gian dữ liệu
- Neuron 2 : Phân tách nhóm con $C_{21} \subset C_2$ với phần còn lại của không gian dữ liệu
- Neuron 3 : Phân tách nhóm con $C_{12} \subset C_1$ với phần còn lại của không gian dữ liệu
- Neuron 4 : Phân tách nhóm con $C_{22} \subset C_2$ với phần còn lại của không gian dữ liệu
- ……

Ở lớp ra (output layer):

- Nếu đối tượng thuộc C_{11} hoặc C_{12}, …. : kết luận đối tượng thuộc nhóm 1
- Nếu đối tượng thuộc C_{21} hoặc C_{22}, …. : kết luận đối tượng thuộc nhóm 2



Cách phân loại nhóm dữ liệu của mạng neuron

Việc phân tách một nhóm con với phần còn lại của không gian dữ liệu là dễ dàng hơn so với phân tách 2 nhóm lớn, đặc biệt khi số neuron sử dụng đủ lớn.

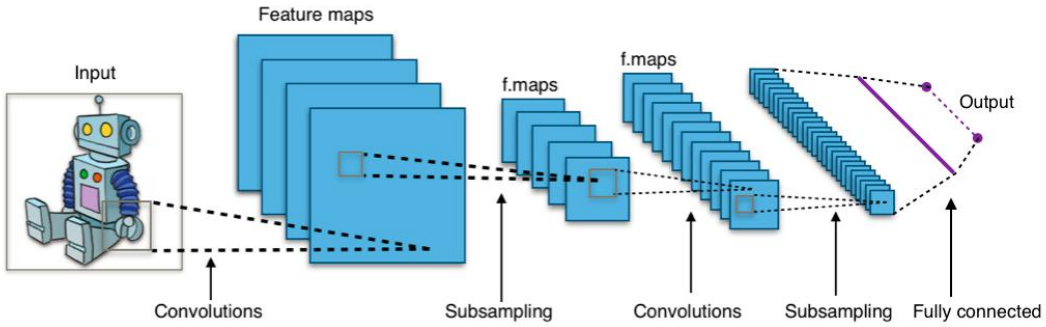
Mạng neuron với rất nhiều lớp (Deep Neural Network)

Trong các bài toán mà số lượng biến đầu vào rất lớn, ví dụ xử lý ảnh (đầu vào là một ma trận điểm ảnh width x height x 3), hoặc xử lý ngôn ngữ tự nhiên (đầu vào là một văn bản) thì mạng neuron với một vài lớp cũng không thể xử lý được.

Trong các bài toán đó, cần sử dụng các mạng neuron với số lớp rất lớn. Các mạng đó được gọi là Deep Neural Network, việc nghiên cứu về các mạng này gọi là Deep Learning.

Convolutional Neural Network (CNN):

Convolutional Neural Network (CNN) được sử dụng ban đầu cho mục đích xử lý ảnh, sau đó mở rộng cho các lĩnh vực khác.



Cấu trúc của CNN. Nguồn : Wikipedia

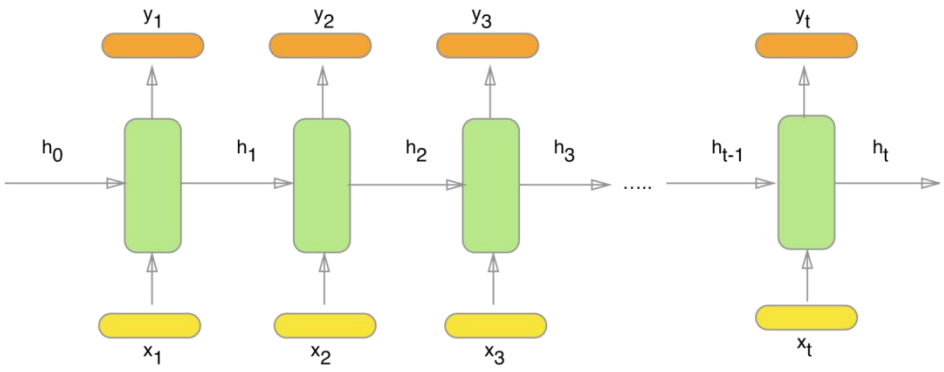
Hàm cơ bản sử dụng trong CNN là hàm convolution 2D. Hàm này được dùng để phát hiện sự tương tự nhau của 2 mảng dữ liệu 2 chiều. Trong xử lý ảnh truyền thống, hàm convoltion 2D được dùng để so sánh một template ảnh có sẵn với các vùng của một bức ảnh để phát hiện sự tồn tại của một đối tượng (template) trong ảnh.

Việc dùng một template để xác định sự có mặt của đối tượng trong ảnh sẽ gặp khó khăn khi kích thước ảnh, tỉ lệ ảnh thay đổi hoặc đối tượng trong ảnh bị biến dạng (do thay đổi độ sáng , góc chụp). Việc này tương tự như kho khăn gặp phải khi dùng một neuron cho bài toán xấp xỉ hàm hay phân loại.

CNN giải quyết vấn đề này bằng cách tạo ra rất nhiều layer, mỗi layer sẽ có các bộ lọc (filter) mà thực chất là các template để phát hiện sự có mặt của đổi tượng ở các scale khác nhau. Ngoài ra, giá trị của các bộ lọc (filter) này được tự học từ tập dữ liệu huấn luyện chứ không phải chuẩn bị trước như các template trong xử lý ảnh truyền thống.

Recurrent Neural Network (RNN):

RNN được sử dụng trong các bài toán mà thứ tự của các biến đầu vào là quan trọng, ví dụ như một tín hiệu theo thời gian, hoặc một đoạn văn bản (thứ tự sắp xếp của các từ quyết định đến ý nghĩa của câu).



Cấu trúc của RNN.

Trong RNN, đầu ra của mỗi neuron tại một thời điểm được kết hợp với đầu vào tại thời điểm tiếp theo để tính toán đầu ra của bước tiếp. Như vậy RNN là một hệ có trạng thái, tức đầu ra không chỉ phụ thuộc đầu vào mà còn phụ thuộc trạng thái của quá khứ.

Mỗi phần tử trong RNN là một phần tử nhớ, thường sử dụng theo cấu trúc Long short-term memory (LSTM). Việc dùng cấu trúc này chủ yếu liên quan đến tính khả thi trong việc huấn luyện RNN. RNN do đó đôi khi còn được gọi là LSTM network.

Huấn luyện mạng neuron

Việc huấn luyện mạng neuron cũng tuân theo nguyên tắc như huấn luyện các mô hình Machine learning mà chúng ta đã biết. Quá trình huấn luyện thực hiện trên tập dữ liệu training, sau đó được kiểm chứng trên tập test.

Trong quá trình huấn luyện, các tham số được chỉnh định sao cho sai lệch giữa đầu ra của mạng neural và giá trị đầu ra thật là nhỏ nhất.

Vì mạng neuron thực chất là một hàm của nhiều biến vào, do đó quá trình chỉnh định tham số có thể dùng theo nguyên lý của phương pháp Gradient Descent như chúng ta đã tìm hiểu ở phần về Logistic Regression.

$$W \longleftarrow W - lr.\frac{\partial E}{\partial W} \quad , \quad E = \sum (f(x_i) - y_i)^2$$

Trong đó (x_i, y_i) là các giá trị trong tập dữ liệu dùng để huấn luyện mạng neuron, lr là learning rate, f là hàm mô tả quan hệ vào ra của mạng neuron, W là các tham số cần tìm của mạng neuron.

Ví dụ 1:

Phân loại 3 chủng hoa trong tập dữ liệu iris bằng Neural Network.

Chúng ta có thể dùng đối tượng MLPClassifier trong thư viện Scikitlearn để xây dựng mạng neuron cho bài toán phân loại:

```
from sklearn.neural_network import MLPClassifier

model = MLPClassifier(hidden_layer_sizes=(...), activation='...',
          max_iter = ..., tol=..., verbose=True/False)
```

```
model.fit(Xtrain, Ytrain)       # huấn luyện
model.predict(Xtest)            # dùng model sau huấn luyện để dự đoán
```

Các tham số:

- **hidden_layer_sizes** : mảng kích thước của các hidden layers
- **activation** : hàm activation, nhận một trong các giá trị : {'identity', 'logistic', 'tanh', 'relu'}, mặc định là relu
- **max_iter** : số bước lặp tối đa, mặc định là 200
- **tol** : nếu sai số ở các bước liên tiếp không nhỏ hơn giá trị này thì dừng quá trình training. Giá trị này nên chọn đủ nhỏ, hoặc bằng -1
- **verbose** : hiển thị log ở từng bước hay không

Ngoài ra hàm còn các tham số khác, có thể tham khảo theo tài liệu của scikitlearn

Chương trình đầy đủ cho bài toán iris:

```
import numpy as np
import pandas as pd
from sklearn.neural_network import MLPClassifier

def get_class(species):
    return {
        'setosa': [1, 0, 0],
        'versicolor': [0, 1, 0],
        'virginica': [0, 0, 1]
    }[species]


df = pd.read_csv('iris.csv')
values = df.values
X = np.array(values[:, :-1], dtype=np.float)
Y = np.array([get_class(species) for species in values[:, -1]])

Xtrain = np.concatenate((X[:35], X[50:85], X[100:135]))
Ytrain = np.concatenate((Y[:35], Y[50:85], Y[100:135]))
Xtest = np.concatenate((X[35:50], X[85:100], X[135:]))
Ytest = np.concatenate((Y[35:50], Y[85:100], Y[135:]))

model = MLPClassifier(hidden_layer_sizes=(5,), activation='tanh',
                      max_iter=10000, tol=1e-7, verbose=True)

model.fit(Xtrain, Ytrain)

score = model.score(Xtest, Ytest)
print('test score = ', score)
```

Sử dụng thư viện keras:

Dùng scikitlearn để tạo mô hình Neural Network khá nhanh và đơn giản. Tuy nhiên scikitlearn không tối ưu về hiệu năng và không cho phép dùng GPU cho các bài toán lớn. Do vậy, chúng ta nên tìm hiểu cách tạo mạng Neural bằng thư viện keras.

Thư viện keras là lớp abstract phía trên của các thư viện về Deep Learning như tensorflow (Google), Theano, CNTK (Microsoft). Keras cho phép tạo ra các mô hình Neural Network khá nhanh và độc lập với Backend sử dụng (tensorflow/Theano/CNTK, ….)

Để tạo mô hình Neural Network cho bài toán phân loại, chúng ta dùng mô hình Sequencial (nối tiếp các lớp của) keras:

```
from keras.layers import Dense, Activation
from keras.optimizers import Adam

model = Sequential()
model.add(Dense(5, input_dim=4, activation='relu'))
model.add(Dense(3, activation='softmax'))
```

Đoạn chương trình trên tạo ra một mạng neuron với 4 đầu vào, 5 neuron ở lớp ẩn và 3 đầu ra. Hàm activation ở đầu ra là softmax, vì bài toán phân loại ở đây có 3 nhóm đối tượng.

Tiếp theo, chúng ta chọn chỉ tiêu tối ưu cho model:

```
from keras.optimizers import Adam
adam = Adam(lr=0.005)
model.compile(optimizer=adam, loss='categorical_crossentropy',
                              metrics=['accuracy'])
```

Đoạn chương trình trên gắn bộ tối ưu Adam cho model. Adam là thuật toán tối ưu tham số cải tiến so với Gradient Descent. Giá trị learning rate được chọn ở trên là 0.005

Tham số loss là hàm mục tiêu tối ưu, ở trên dùng **categorical_crossentropy**. Các bài toán phân loại thường dùng hàm mục tiêu là categorical_crossentropy**,** thay cho mse (mean square error) vì cho tốc độ hội tụ nhanh hơn. Công thức tính categorical_crossentropy có thể tham khảo trong tài liệu hướng dẫn sử dụng của keras.

Tham số metrics là chỉ tiêu đo kết quả, ở trên dùng accuracy, tức độ chính xác (tính bằng tỉ lệ giữa số mẫu cho kết quả đoán đúng trên tổng số mẫu)

Sau khi đã xây dựng model, chúng ta huấn luyện model trên tập dữ liệu training:

```
model.fit(Xtrain, Ytrain, epochs=2000, shuffle=True)
```

Các tham số:

- epochs : số bước lặp
- shuffle : hoán vị bộ dữ liệu ở mỗi bước hay không, để tránh tình trạng hội tụ về cực tiểu địa phương

Các tham số khác có thể tham khảo trong tài liệu hướng dẫn của keras

Các chỉ tiêu loss và accuracy sẽ được hiển thị trong quá trình huấn luyện model. Kết thúc quá trình huấn luyện, chúng ta kiểm tra độ chính xác trên tập dữ liệu test:

```
_, score = model.evaluate(Xtest, Ytest)
print('test score = ', score)
```

Chương trình đầy đủ:

```
import numpy as np
```

```
import pandas as pd
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam

def get_class(species):
    return {
        'setosa': [1, 0, 0],
        'versicolor': [0, 1, 0],
        'virginica': [0, 0, 1]
    }[species]


df = pd.read_csv('iris.csv')
values = df.values
X = np.array(values[:, :-1], dtype=np.float)
Y = np.array([get_class(species) for species in values[:, -1]])

Xtrain = np.concatenate((X[:35], X[50:85], X[100:135]))
Ytrain = np.concatenate((Y[:35], Y[50:85], Y[100:135]))
Xtest = np.concatenate((X[35:50], X[85:100], X[135:]))
Ytest = np.concatenate((Y[35:50], Y[85:100], Y[135:]))

model = Sequential()
model.add(Dense(10, input_dim=4, activation='relu'))
model.add(Dense(3, activation='softmax'))

adam = Adam(lr=0.005)
model.compile(optimizer=adam, loss='categorical_crossentropy',
metrics=['accuracy'])

model.fit(Xtrain, Ytrain, epochs=2000, shuffle=True)

_, score = model.evaluate(Xtest, Ytest)
print('test score = ', score)
```

Ví dụ 2: Lượng năng lượng thế giới tiêu thụ qua các năm từ 1990 đến 2007 :

1990	1991	1992	1993	1994	1995	1996	1997	1998
8761	8818	8830	8923	8993	9215	9447	9540	9593
1999	2000	2001	2002	2003	2004	2005	2006	2007
9789	10016	10114	10330	10688	11171	11489	11835	12152

Đơn vị năng lượng tiêu thụ là MTOE (million tons of oil equivalent). Nguồn dữ liệu : Enerdata

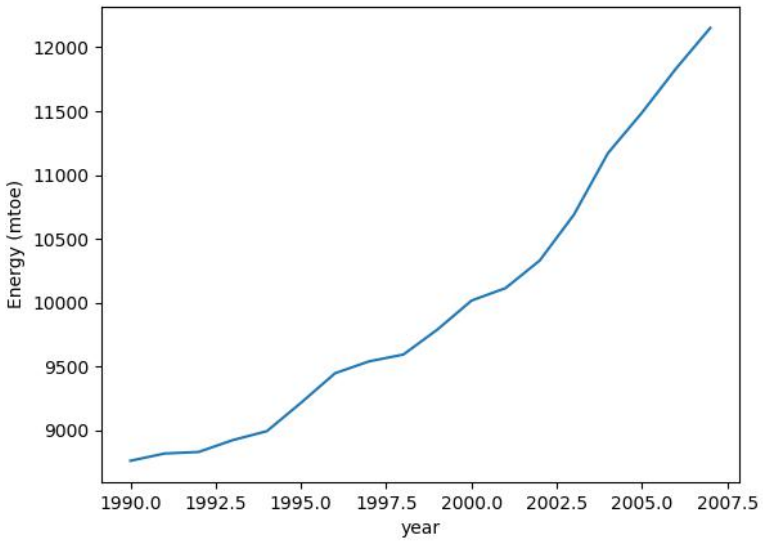
Xây dựng mạng neural để đưa ra dự đoán năng lượng tiêu thụ của thế giới trong mỗi năm dựa trên năng lượng của các năm trước.
Trước tiên, chúng ta vẽ biểu đồ tiêu thụ năng lượng qua các năm:

```
import matplotlib.pyplot as plt

years = list(range(1990, 2008))

energies= [8761, 8818, 8830, 8923, 8993, 9215, 9447, 9540, 9593, 9789,
          10016, 10114, 10330, 10688, 11171, 11489, 11835, 12152 ]
```

```
plt.plot(years, energies)
plt.xlabel('years')
plt.ylabel('energies')
plt.show()
```



Đồ thị năng lượng tiêu thụ của thế giới qua các năm

Dựa vào đồ thị có thể thấy năng lượng thế giới tiêu thụ qua các năm không tăng theo một đường thẳng như dân số thế giới trong ví dụ về Linear Regression.

Mô hình Linear Regression vẫn có thể sử dụng nhưng cho kết quả không chính xác. Chúng ta sử dụng mạng neuron với hàm activation relu để đưa ra hàm xấp xỉ năng lượng tiêu thụ qua các năm.

Về nguyên tắc của dự đoán, chúng ta chỉ được dùng thông tin đã biết của quá khứ để ước đoán giá trị trong tương lai. Như vậy mỗi khi dự đoán năng lượng tiêu thụ trong một năm, chúng ta phải xây dựng một mạng neuron với dữ liệu huấn luyện trong các năm trước.

Để minh họa, ở đây chúng ta chỉ xây dựng một mạng neuro với dữ liệu từ năm 1990 đến 2005, và dùng mạng neuron này dự đoán cho năm 2006 và 2007

Sử dụng scikitlearn để xây dựng mạng neuron cho bài toán xấp xỉ hàm:

Trong scikitlearn, đối tượng MLPRegressor được dùng cho việc xấp xỉ hàm. Cách xây dựng mạng neuron này gần giống với MLPClassifier ở ví dụ 1.

```
from sklearn.neural_network import MLPRegressor

model = MLPRegressor(hidden_layer_sizes=(...), activation='...',
          max_iter = ..., tol=..., verbose=True/False)

model.fit(Xtrain, Ytrain)       # huấn luyện
model.predict(Xtest)            # dùng model sau huấn luyện để dự đoán
```

Chương trình đầy đủ:

```
from sklearn.neural_network import MLPRegressor
```

```
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt

years = list(range(1990, 2008))

energies = [8761, 8818, 8830, 8923, 8993, 9215, 9447, 9540, 9593, 9789,
          10016, 10114, 10330, 10688, 11171, 11489, 11835, 12152 ]

X = [[x] for x in years]
Y = [y/1000 for y in energies]
Xtrain = X[:-2]
Ytrain = Y[:-2]
Xtest = X[-2:]
Ytest = Y[-2:]

scaler = MinMaxScaler()
scaler.fit(Xtrain)

Xtrain = scaler.transform(Xtrain)
Xtest = scaler.transform(Xtest)

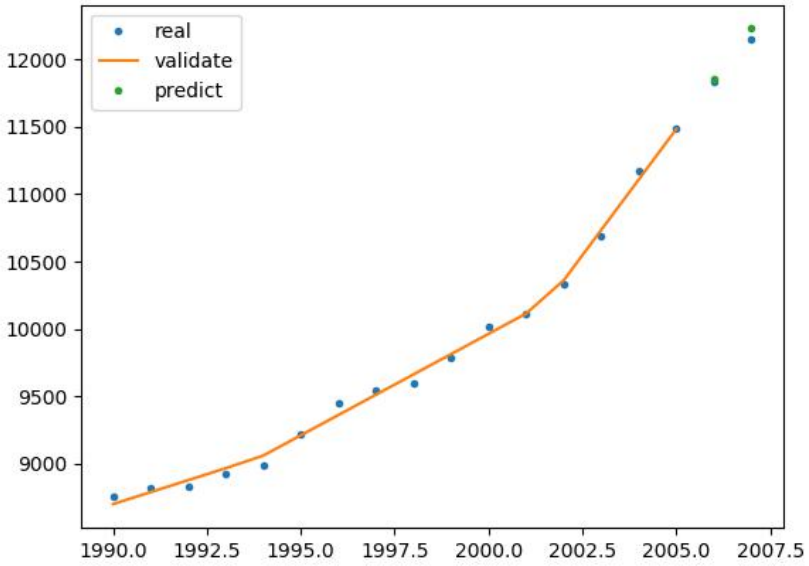
model = MLPRegressor(50, max_iter=10000, tol=-1, verbose=True)
model.fit(Xtrain, Ytrain)

Ypredict = model.predict(Xtest)
for i in range(2):
    predict = round(1000 * Ypredict[i])
    err = round(100 * (predict - energies[-2+i])/energies[-2+i], 1)
    print('Year {}, real : {}, predict : {}, err(%) : {}'.format
            (years[-2+i], energies[-2+i], predict, err))

plt.plot(years, energies, '.', label='real')
plt.plot(years[:-2], 1000 * model.predict(Xtrain), label='validate')
plt.plot(years[-2:], 1000 * model.predict(Xtest), '.', label='predict')
plt.legend(loc='upper left')
plt.show()
```

Kết quả chạy chương trình:

```
Year 2006, real : 11835, predict : 11856.0, err(%) : 0.2
Year 2007, real : 12152, predict : 12228.0, err(%) : 0.6
```



Dữ liệu thật và dự đoán của mô hình

Một vài lưu ý:

- Kết quả các lần chạy có thể không giống nhau, do hiện tượng khởi tạo biến ngẫu nhiên. Có trường hợp mạng neuron suy biến thành đường thẳng và kết quả chỉ tương đương với Linear Regression

- Đầu vào và đầu ra của bài toán xấp xỉ hàm nên được chuẩn hóa với công cụ MinMaxScaler của scikitlearn để đưa khoảng giá trị của các biến về [0,1]

Sử dụng Keras:

Tương tự ví dụ 1, có thể dùng Keras để xây dựng mạng neuron

```
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam

years = list(range(1990, 2008))

energies = [8761, 8818, 8830, 8923, 8993, 9215, 9447, 9540, 9593, 9789,
          10016, 10114, 10330, 10688, 11171, 11489, 11835, 12152 ]

X = [[x] for x in years]
Y = [y/1000 for y in energies]
Xtrain = X[:-2]
Ytrain = Y[:-2]
Xtest = X[-2:]
Ytest = Y[-2:]

scaler = MinMaxScaler()
scaler.fit(Xtrain)

Xtrain = scaler.transform(Xtrain)
Xtest = scaler.transform(Xtest)
```

```
model = Sequential()
model.add(Dense(50, input_dim=1, activation='softplus'))
model.add(Dense(1))

adam = Adam(lr=0.01)
model.compile(optimizer=adam, loss='mse')

model.fit(Xtrain, Ytrain, epochs=5000)

Ypredict = model.predict(Xtest)
for i in range(2):
    predict = round(1000 * Ypredict[i][0])
    err = round(100 * (predict - energies[-2+i])/energies[-2+i],1)
    print('Year {}, real : {}, predict : {}, err(%) : {}'.format
            (years[-2+i], energies[-2+i], predict, err))

plt.plot(years, energies, '.', label='real')
plt.plot(years[:-2], 1000 * model.predict(Xtrain), label='validate')
plt.plot(years[-2:], 1000 * model.predict(Xtest), '.', label='predict')
plt.legend(loc='upper left')
plt.show()
```

Ví dụ 3:

Nhận dạng chữ số viết tay với mạng neuron. Nguồn : Handwritten Digit Recognition With Keras

Kho dữ liệu hình ảnh MNIST chứa các chữ số viết tay từ 0 đến 9. Mỗi ảnh có kích thước 28 x 28 pixel. Cần xây dựng mạng neuron để từ một file ảnh cho ra kết quả là chữ số trong file ảnh đó.

Download tập dữ liệu ảnh MNIST :

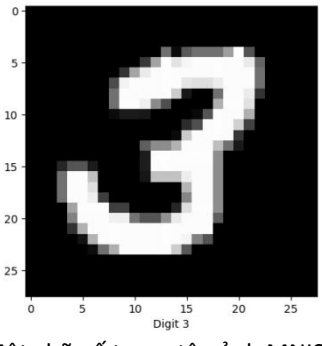
```
from keras.datasets import mnist
(Xtrain, ytrain), (Xtest, ytest) = mnist.load_data()
```

Keras sẽ tự động download MNIST dataset từ Amazon S3 về máy.

Hiển thị một ảnh trong tập dữ liệu ảnh:

```
import matplotlib.pyplot as plt
from random import randint

index = randint(0, len(Xtrain)-1)
img = Xtrain[index]
plt.imshow(img, cmap=plt.get_cmap('gray'))
plt.xlabel('Digit ' + str(ytrain[index]))
plt.show()
```



Một chữ số trong tập ảnh MNIST

Chuẩn hóa dữ liệu:

```
from keras.utils import to_categorical

num_pixels = Xtrain.shape[1] * Xtrain.shape[2]
Xtrain = Xtrain.reshape(Xtrain.shape[0],num_pixels).astype('float32')
Xtest = Xtest.reshape(Xtest.shape[0], num_pixels).astype('float32')

Xtrain = Xtrain / 255
Xtest = Xtest / 255

ytrain = to_categorical(ytrain)
ytest = to_categorical(ytest)
num_classes = ytest.shape[1]
```

Xây dựng mạng neuron:

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam

model = Sequential()
model.add(Dense(num_pixels, input_dim=num_pixels,
kernel_initializer='normal', activation='relu'))

model.add(Dense(num_classes, kernel_initializer='normal',
                activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
                metrics=['accuracy'])
```

Huấn luyện mạng và kiểm tra mô hình trên tập dữ liệu test:

```
model.fit(Xtrain, ytrain, validation_data=(Xtest, ytest), epochs=10,
                batch_size=200, verbose=2)
scores = model.evaluate(Xtest, ytest, verbose=0)
print("score=", scores[1])
```

Kết quả score = 0.9818, hay độ chính xác trên tập dữ liệu test là 98.18%

Cuối cùng, chúng ta lưu lại model vào file để sử dụng về sau:

```
f = open('mnist.json', 'w')
f.write(model.to_json())
f.close()

model.save_weights('mnist.h5')
```

Cấu trúc model được lưu vào file mnist.json, giá trị các tham số trong model được lưu vào file mnist.h5

Chương trình đầy đủ :

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam
```

```
from keras.utils import to_categorical

(Xtrain, ytrain), (Xtest, ytest) = mnist.load_data()

num_pixels = Xtrain.shape[1] * Xtrain.shape[2]
Xtrain = Xtrain.reshape(Xtrain.shape[0], num_pixels).astype('float32')
Xtest = Xtest.reshape(Xtest.shape[0], num_pixels).astype('float32')

Xtrain = Xtrain / 255
Xtest = Xtest / 255

ytrain = to_categorical(ytrain)
ytest = to_categorical(ytest)
num_classes = ytest.shape[1]

model = Sequential()
model.add(Dense(num_pixels, input_dim=num_pixels,
                kernel_initializer='normal', activation='relu'))

model.add(Dense(num_classes, kernel_initializer='normal',
                activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
                metrics=['accuracy'])

model.fit(Xtrain, ytrain, validation_data=(Xtest, ytest), epochs=10,
                batch_size=200, verbose=2)

scores = model.evaluate(Xtest, ytest, verbose=0)
print("score=", scores[1])

f = open('mnist.json', 'w')
f.write(model.to_json())
f.close()

model.save_weights('mnist.h5')
```

Sử dụng model đã lưu để nhận biết một ảnh chữ viết tay

Chúng ta dùng paint để vẽ một ảnh chữ viết tay:

Ảnh chữ số được vẽ trong chương trình Paint

Tiếp theo chúng ta xây dựng chương trình để đưa ra chữ số đã được viết trong ảnh:

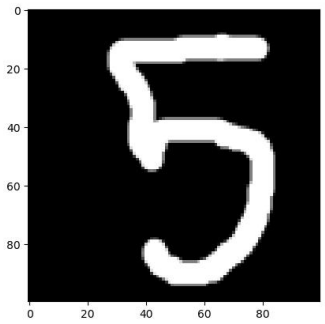
Tải ảnh từ file:

```
from scipy import misc
import matplotlib.pyplot as plt
```

```
img = misc.imread('5.png', flatten=True)
plt.imshow(img, cmap=plt.get_cmap('gray'))
plt.show()
```

Ảnh sẽ được hiển thị trong cửa sổ matplotlib. Chúng ta thấy màu sắc của ảnh (chữ đen/nền trắng) ngược với các ảnh trong MNIST (chữ trắng/nền đen), do đó cần đảo ngược lại màu sắc cho ảnh:

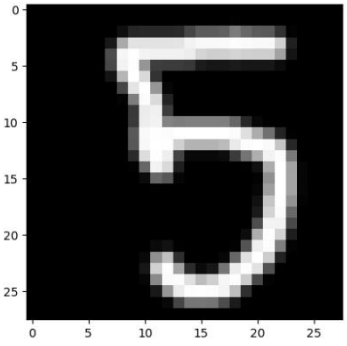
```
img = 255 -img
plt.imshow(img, cmap=plt.get_cmap('gray'))
plt.show()
```



Đảo ngược màu sắc của ảnh

Ngoài ra kích thước của ảnh không đúng như trong tập MNIST (28x28), chúng ta cần scale lại ảnh về đúng kích thước 28x28:

```
img = misc.imresize(img, (28,28))
plt.imshow(img, cmap=plt.get_cmap('gray'))
plt.show()
```



Ảnh sau khi được scale về kích thước 28x28

Từ dữ liệu ảnh, chúng ta chuyển thành vector có kích thước 784 (28^2), sau đó chuẩn hóa dữ liệu (về khoảng [0,1] như trong phần training model):

```
img = img.reshape(-1).astype('float32')
img = img/255
```

Sau khi đã có dữ liệu, chúng ta load model từ các file json và h5 đã được lưu trong quá trình training model:

```
from keras.models import model_from_json
```

```
f = open('mnist.json')
json_content = f.read()
f.close()

model = model_from_json(json_content)
model.load_weights('mnist.h5')
```

Cuối cùng, chúng ta dùng model để phán đoán chữ số trong ảnh:

```
import numpy as np
X = np.array([img])
y = model.predict(X)
print('Digit : ', np.argmax(y[0]))
```

Kết quả:

```
Digit :  5
```

Chương trình đầy đủ:

```
import sys
import numpy as np
from scipy import misc
import matplotlib.pyplot as plt
from keras.models import model_from_json

img = misc.imread(sys.argv[1], flatten=True)
img = 255 - img
img = misc.imresize(img, (28,28))
img = img.reshape(-1).astype('float32')
img = img/255

f = open('mnist.json')
json_content = f.read()
f.close()

model = model_from_json(json_content)
model.load_weights('mnist.h5')

X = np.array([img])
y = model.predict(X)
print('Digit : ', np.argmax(y[0]))
```

Cách sử dụng chương trình trên: Đặt tên file là digit_recog.py, để file trong cùng thư mục với các file mnist.json và mnist.h5, sau đó từ cửa sổ cmd ở thư mục chứa file, gõ lệnh:

```
python digit_recog.py ten_file_anh
```

Tên của file ảnh được truyền vào từ tham số của dòng lệnh, kết quả chữ số trong ảnh sẽ được in ra màn hình.

Có thể thấy nếu dùng Paint để vẽ các chữ số thì độ chính xác không cao như kết quả đã kiểm tra trên tập test của MNIST (98.18%), lí do là do ảnh vẽ bằng Paint tương đối khác so với các mẫu trong MNIST nên mạng neuron đã train có nhiều trường hợp sẽ phán đoán sai.

Sử dụng Convolutional Neural Network:

Với các bài toán xử lý ảnh với kích thước nhỏ như MNIST thì mạng neural nhiều lớp thông thường (MLP) có thể xử lý được, nhưng khi kích thước ảnh lớn, mạng MLP sẽ có số lượng tham số rất lớn và khả năng training là rất khó, ngoài ra còn có thể xảy ra hiện tượng overfitting.

Trong trường hợp đó cần dung đến CNN. Chương trình dưới đây minh họa cách tạo một CNN đơn giản cho bài toán MNIST

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.optimizers import Adam
from keras.utils import to_categorical


(Xtrain, ytrain), (Xtest, ytest) = mnist.load_data()
Xtrain = Xtrain.reshape(Xtrain.shape[0], 28, 28, 1).astype('float32')
Xtest = Xtest.reshape(Xtest.shape[0], 28, 28, 1).astype('float32')

Xtrain = Xtrain / 255
Xtest = Xtest / 255

ytrain = to_categorical(ytrain)
ytest = to_categorical(ytest)
num_classes = ytest.shape[1]


model = Sequential()
model.add(Conv2D(32, (5, 5), input_shape=(28, 28, 1),
activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
model.fit(Xtrain, ytrain, validation_data=(Xtest, ytest), epochs=10,
batch_size=200, verbose=2)

f = open('mnist_cnn.json', 'w')
f.write(model.to_json())
f.close()

model.save_weights('mnist_cnn.h5')
```

Sử dụng model đã train để phán đoán:

```
import sys
import numpy as np
from scipy import misc
import matplotlib.pyplot as plt
from keras.models import model_from_json

img = misc.imread(sys.argv[1], flatten=True)
```

```
img = 255 - img
img = misc.imresize(img, (28,28))
img = img.reshape(28,28,1).astype('float32')
img = img/255

f = open('mnist_cnn.json')
json_content = f.read()
f.close()

model = model_from_json(json_content)
model.load_weights('mnist_cnn.h5')

X = np.array([img])
y = model.predict(X)
print('Digit : ', np.argmax(y[0]))
```

Ví dụ 4: Phân loại tài liệu

Mạng neuron cũng có thể được dùng để xử lý văn bản. Ở ví dụ này, chúng ta tìm hiểu cách dùng mạng neuron để phân loại tài liệu theo chủ đề nội dung của tài liệu.

Trước tiên, chúng ta tìm hiểu một số cách mã hóa văn bản thành dữ liệu dạng số.

BOW (Bag of word):

Mô hình Bag of word dựa trên một từ điển là một danh sách các từ thường gặp trong các tài liệu. Mỗi tài liệu được đặc trưng bằng số lượng xuất hiện của các từ trong tài liệu

Ví dụ từ điển : {1 : một, 2 : hai, 3: ba …., chín : 9, mười : 10, trăm : 11, lẻ : 12, mươi : 13}
Nội dung văn bản : một trăm hai mươi hai
Mã hóa BOW : v = [(1,1), (11,1), (2,2), (13,1)]

TF-IDF (term frequency - inverse document frequency):

TF-IDF là một dạng biến đổi của BOW. Trong BOW, số lượng từ xuất hiện trong một tài liệu sẽ phụ thuộc vào độ dài tài liệu, ngoài ra một số từ xuất hiện nhiều trong hầu hết các tài liệu (các từ nối, đệm). Hiện tượng này dẫn đến số lần xuất hiện của một từ không phản ánh đúng mức độ quan trọng của từ trong tài liệu.

Với TF-IDF, hai điểm trên được khắc phục như sau:

- Sử dụng term frequency (TF) : tf của một từ trong một tài liệu được tính bằng số lần xuất hiện của từ đó trong tài liệu chia cho số lần xuất hiện lớn nhất của một từ trong tài liệu đó. Công thức thường sử dụng :

$$tf(w,d) = 0.5 + 0.5\frac{f_{w,d}}{\max(f_{w',d}, w' \in d)}$$

  Có thể thấy hàm tf sẽ không bị ảnh hưởng bởi độ dài khác nhau của các tài liệu

- Sử dụng inverse document frequency (IDF) : idf của một từ trong một kho tài liệu được tính theo công thức:

$$idf(w) = \log \frac{N}{N_w}$$

Trong đó N là tổng số tài liệu, N_w là số tài liệu có chứa từ w
IDF sẽ làm các từ hay thương xuyên xuất hiện trong ngôn ngữ (từ nối) không được coi là quan trọng trong tài liệu đang xét

TF-IDF được tính bằng tích số của TF và IDF:

$$tfidf(w,d) = tf(w,d).idf(w)$$

Sau khi biết cách tính tf-idf, chúng ta dùng cách mã hóa này để tạo dữ liệu vào cho mạng neuron trong bài toán phân loại tài liệu.

Chuẩn bị dữ liệu :

Các tài liệu cần phân loại là các tin ngắn lấy từ các bài trên báo Dân trí. File chứa các tài liệu này có thể dowload về : news_headlines.zip File download về giải nén ra sẽ chứa 3 file văn bản chứa các tin ngắn thuộc 3 chủ để khác nhau:

- xahoi.txt : Các tin về chủ đề chính trị, xã hội
- kinhdoanh.txt : Các tin về chủ đề kinh doanh
- thethao.txt : Các tin về chủ đề thể thao

Một số ví dụ về các mẩu tin trong các file dữ liệu :

- Chính trị xã hội:
  - Sáng nay (6/7), một ô tô tải đang lưu thông trên tuyến đường tránh Nam Hải Vân (Đà Nẵng) bất ngờ mất lái, lao vào rào chắn công trình, khiến 2 người thương vong.
  - Thanh tra TP Đà Nẵng đã chuyển 451 hồ sơ về nguồn gốc đất thuộc ranh giới khu vực dự án Ga đường sắt ở quận Liên Chiểu sang Cơ quan Cảnh sát điều tra - Công an thành phố.
- Kinh doanh :
  - Một bé trai người Hàn Quốc được nằm nghỉ và uống một hộp sữa tại bệnh viện San Francisco. Sau đó, bố mẹ bé đã phát hoảng khi nhận được hóa đơn viện phí lên tới 18.836 USD (tương đương gần 440 triệu đồng).
  - Mở bán giữa thời điểm cơn sốt đất nền tại Huế đang lên cao, Royal Park Huế được giới BĐS xem là "miếng bánh ngon" không thể bỏ lỡ. Điều gì đang là hấp lực với các nhà đầu tư?
- Thể thao :
  - Ngày hôm qua, Real Madrid đã chính thức công bố chiêu mộ thành công hậu vệ cánh phải Alvaro Odriozola từ Real Sociedad. Đây là tân binh đầu tiên mà HLV Lopetegui mang về ở Bernabeu.
  - Trước đối thủ yếu hơn, gần như chắc chắn HLV Hoàng Anh Tuấn sẽ lại sử dụng đội hình 2, nhằm giúp các cầu thủ trụ cột có thể lực tốt nhất trước trận quan trọng gặp chủ nhà U19 Indonesia.

Để xử lý văn bản, trước hết chúng ta cần cài đặt một số thư viện:

```
import pip

pip.main(["install", "gensim"])
pip.main(["install", "pyvi"])
```

Thư viện gensim được dùng cho việc xử lý ngôn ngữ , cung cấp các hàm tính BOW, TF-IDF và nhiều hàm về xử lý văn bản. Thư viện pyvi cung cấp các hàm tách từ và xác định từ loại tiếng Việt.

Với một văn bản tiếng Việt đầu vào, trước hết chúng ta cần tách ra từng từ của văn bản, bỏ đi các từ chứa các kí tự đặc biệt:

```
import re
from pyvi import ViTokenizer, ViPosTagger

special_regex = r'.*[0-9()\[\]{}\+\-\*\'/.%~`@#$%^&*|,\\?—="].*'

def word_tokenize(sentence):
    words, postags = ViPosTagger.postagging(
                  ViTokenizer.tokenize( sentence.lower()))
    filter_words = []

    for i in range(len(words)):
        if postags[i] not in ['N','V','A'] : continue
        if re.match(special_regex, words[i]): continue
        filter_words.append(words[i])

    return filter_words
```

Hàm word_tokenize ở trên tách văn bản thành các từ, xác định từ loại của từng từ và chỉ giữ lại các từ thuộc một trong 3 từ loại danh từ/ động từ / tính từ, vì 3 từ loại này quyết định nội dung chủ yếu của văn bản

Tiếp đến chúng ta đọc toàn bộ tài liệu từ các file txt giải nén từ file đã download về. Để tiện phân chia các tập training và test ở bước sau, mỗi chủ đề chỉ lấy 100 tin ngắn:

```
topics = ['xahoi' , 'kinhdoanh', 'thethao']
topic_names = ['Xã hội', 'Kinh doanh', 'Thể thao']
num_classes = len(topics)

documents = []
labels = []

for i in range(len(topics)):
    f = open(topics[i] + '.txt', encoding='utf8')
    documents.extend(f.readlines()[:100])
    labels.extend([i]*100)
    f.close()
```

Dùng gensim để tạo từ điển, tính toán Bag of word và tf-idf trên danh sách tài liệu đã tạo ra:

```
import gensim
from gensim import models

processed_docs = list(map(word_tokenize, documents))
dictionary = gensim.corpora.Dictionary(processed_docs)
dictionary.filter_extremes(no_below=3, no_above=0.5, keep_n=100000)
dict_size = len(dictionary)
```

```
bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
tfidf = models.TfidfModel(bow_corpus)
```

Tạo tập dữ liệu training và test:

```
import numpy as np
from keras.utils import to_categorical

X = []
Y = []

for i in range(len(processed_docs)):
    bow_vector = tfidf[bow_corpus[i]]
    wordvec = np.zeros(dict_size)

    for index, value in bow_vector:
        wordvec[index] = value

    X.append(wordvec)
    Y.append(to_categorical(labels[i], num_classes))

X_train = np.array(X[:70] + X[100:170] + X[200:270])
Y_train = np.array(Y[:70] + Y[100:170] + Y[200:270])

X_test = np.array(X[70:100] + X[170:200] + X[270:300])
Y_test = np.array(Y[70:100] + Y[170:200] + Y[270:300])
```

Xây dựng mô hình mạng neuron với keras:

```
from keras.utils import to_categorical
from keras.optimizers import Adam

model = Sequential()
model.add(Dense(5, input_dim=dict_size, activation='sigmoid'))
model.add(Dense(num_classes, activation='softmax'))

adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-6)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],
                optimizer=adam)

model.fit(X_train, Y_train, epochs=500, shuffle=True)
_, score = model.evaluate(X_test, Y_test)
print('score = ', score)
```

Kết quả cho score = 0.86, tức độ chính xác trên tập dữ liệu test là 86%

Dự đoán chủ đề của một đoạn tin không có trong tập dữ liệu:

```
text = """Chiều ngày 7/7, một cơn mưa rào bất ngờ đổ xuống xứ Huế giải
          nhiệt sau những ngày nắng nóng lên đến 40 độ C, thỏa mãn "cơn
          khát" mưa của người dân."""

processed_text = word_tokenize(text)
bow = dictionary.doc2bow(processed_text)
bow_vector = tfidf[bow]

wordvec = np.zeros(dict_size)
for index, value in bow_vector:
    wordvec[index] = value
```

```
predict = model.predict(np.array([wordvec]))
categ = np.argmax(predict[0])
print('Chủ đề : ', topic_names[categ])
```

Kết quả :

```
Chủ đề :  Xã hội
```

Chương trình đầy đủ :

```
import os
import re
import numpy as np
import gensim
from gensim import models
from pyvi import ViTokenizer, ViPosTagger

from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import Adam
from keras.utils import to_categorical

special_regex = r'.*[0-9()\[\]{}\+\-\*\'/.%~`@#$%^&*|,\\?—="].*'

def word_tokenize(sentence):
    words, postags = ViPosTagger.postagging(
                        ViTokenizer.tokenize(sentence.lower()))

    filter_words = []

    for i in range(len(words)):
        if postags[i] not in ['N','V','A'] : continue
        if re.match(special_regex, words[i]): continue
        filter_words.append(words[i])

    return filter_words

topics = ['xahoi' , 'kinhdoanh', 'thethao']
topic_names = ['Xã hội', 'Kinh doanh', 'Thể thao']
num_classes = len(topics)

documents = []
labels = []

for i in range(len(topics)):
    fn = os.path.join('news_headlines', topics[i] + '.txt')
    f = open(fn, encoding='utf8')
    documents.extend(f.readlines()[:100])
    labels.extend([i]*100)
    f.close()

processed_docs = list(map(word_tokenize, documents))
dictionary = gensim.corpora.Dictionary(processed_docs)
dictionary.filter_extremes(no_below=3, no_above=0.5, keep_n=100000)
dict_size = len(dictionary)

bow_corpus = [dictionary.doc2bow(doc) for doc in processed_docs]
tfidf = models.TfidfModel(bow_corpus)
```

```
X = []
Y = []

for i in range(len(processed_docs)):
    bow_vector = tfidf[bow_corpus[i]]
    wordvec = np.zeros(dict_size)
    for index, value in bow_vector:
        wordvec[index] = value
    X.append(wordvec)
    Y.append(to_categorical(labels[i], num_classes))

X_train = np.array(X[:70] + X[100:170] + X[200:270])
Y_train = np.array(Y[:70] + Y[100:170] + Y[200:270])

X_test = np.array(X[70:100] + X[170:200] + X[270:300])
Y_test = np.array(Y[70:100] + Y[170:200] + Y[270:300])

model = Sequential()
model.add(Dense(5, input_dim=dict_size, activation='sigmoid'))
model.add(Dense(num_classes, activation='softmax'))

adam = Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-6)
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],
               optimizer=adam)

model.fit(X_train, Y_train, epochs=500, shuffle=True)
_ , score = model.evaluate(X_test, Y_test)
print('score = ', score)

text = """Chiều ngày 7/7, một cơn mưa rào bất ngờ đổ xuống xứ Huế giải
         nhiệt sau những ngày nắng nóng lên đến 40 độ C,
         thỏa mãn "cơn khát" mưa của người dân."""

processed_text = word_tokenize(text)
bow = dictionary.doc2bow(processed_text)
bow_vector = tfidf[bow]

wordvec = np.zeros(dict_size)
for index, value in bow_vector:
    wordvec[index] = value

predict = model.predict(np.array([wordvec]))
categ = np.argmax(predict[0])
print('Chủ đề : ', topic_names[categ])
```