Generating n bit of data

In [2]:

```
n = int(input('Enter Number of bits : '))
count = 0
i = n
string = 'bit_'
total_number = 2 ** n
```

In [3]:

```
value = list()
dictionary = dict()

while i >= 1 :
    key = string + str(i)
    d = 2 ** count

    while len(value) != total_number:
        for j in range(d):
            value.append(0)
        for j in range(d):
            value.append(1)

    dictionary[key] = value
    value = list()
    count = count + 1
    i = i - 1
```

In [4]:

```
#dictionary
```

In [5]:

```
#list(dictionary.items())
```

In [6]:

```
l = list(dictionary.items())
#l
```

In [7]:

```
reversed_dictionary = dict()
i = n-1
while i >= 0:
    reversed_dictionary[l[i][0]] = l[i][1]
    i = i - 1
#reversed_dictionary
```

In [8]:

```
dictionary = reversed_dictionary
#dictionary
```

In [9]:

```
output = dictionary['bit_1']
#output
```

In [100]:

```
import pandas as pd
```

```
df = pd.DataFrame(data=dictionary)
df
```

Out[100]:

	bit_1	bit_2	bit_3	bit_4	bit_5
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1
20	1	0	1	0	0
21	1	0	1	0	1
22	1	0	1	1	0
23	1	0	1	1	1
24	1	1	0	0	0
25	1	1	0	0	1
26	1	1	0	1	0
27	1	1	0	1	1
28	1	1	1	0	0
29	1	1	1	0	1
30	1	1	1	1	0
31	1	1	1	1	1

In [101]:

```
df['Output'] = output
df
```

Out[101]:

	bit_1	bit_2	bit_3	bit_4	bit_5	Output
0	0	0	0	0	0	0
1	0	0	0	0	1	0

2	bit_0	bit_2	bit_3	bit_4	bit_5	Output
3	0	0	0	1	1	0
4	0	0	1	0	0	0
5	0	0	1	0	1	0
6	0	0	1	1	0	0
7	0	0	1	1	1	0
8	0	1	0	0	0	0
9	0	1	0	0	1	0
10	0	1	0	1	0	0
11	0	1	0	1	1	0
12	0	1	1	0	0	0
13	0	1	1	0	1	0
14	0	1	1	1	0	0
15	0	1	1	1	1	0
16	1	0	0	0	0	1
17	1	0	0	0	1	1
18	1	0	0	1	0	1
19	1	0	0	1	1	1
20	1	0	1	0	0	1
21	1	0	1	0	1	1
22	1	0	1	1	0	1
23	1	0	1	1	1	1
24	1	1	0	0	0	1
25	1	1	0	0	1	1
26	1	1	0	1	0	1
27	1	1	0	1	1	1
28	1	1	1	0	0	1
29	1	1	1	0	1	1
30	1	1	1	1	0	1
31	1	1	1	1	1	1

In [102]:

```
df = df.drop('Output',axis=1)
df
```

Out[102]:

	bit_1	bit_2	bit_3	bit_4	bit_5
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	0	1	1
4	0	0	1	0	0
5	0	0	1	0	1
6	0	0	1	1	0
7	0	0	1	1	1
8	0	1	0	0	0

	bit_1	bit_2	bit_3	bit_4	bit_5
9	0	1	0	0	1
10	0	1	0	1	0
11	0	1	0	1	1
12	0	1	1	0	0
13	0	1	1	0	1
14	0	1	1	1	0
15	0	1	1	1	1
16	1	0	0	0	0
17	1	0	0	0	1
18	1	0	0	1	0
19	1	0	0	1	1
20	1	0	1	0	0
21	1	0	1	0	1
22	1	0	1	1	0
23	1	0	1	1	1
24	1	1	0	0	0
25	1	1	0	0	1
26	1	1	0	1	0
27	1	1	0	1	1
28	1	1	1	0	0
29	1	1	1	0	1
30	1	1	1	1	0
31	1	1	1	1	1

In [103]:

```
df.iloc[0,0]
```

Out[103]:

```
0
```

In [104]:

```
df.shape
```

Out[104]:

```
(32, 5)
```

In [105]:

```
n = df.shape[0]
m = df.shape[1]

print('number of total rows :',n)
print('number of features :',m)
```

```
number of total rows : 32
number of features : 5
```

Train Test

In [106]:

```
train_percentage = 60
```

```
test_percentage = 100 - train_percentage

print('Train Percentage :',train_percentage)
print('Test Percentage :',test_percentage)
```

```
Train Percentage : 60
Test Percentage : 40
```

In [107]:

```
import math

no_of_train_data = math.ceil(( total_number * train_percentage ) / 100)
no_of_test_data = total_number - no_of_train_data

print('No of Train Data :',no_of_train_data)
print('No of Test Data :',no_of_test_data)
```

```
No of Train Data : 20
No of Test Data : 12
```

In [108]:

```
n = no_of_train_data # row
m = df.shape[1] # column

print('number of train rows :',n)
print('number of features :',m)
```

```
number of train rows : 20
number of features : 5
```

Initializing weight with random number

In [109]:

```
import numpy as np

np.random.seed(113)
w = np.random.rand(n,m)
print(w)
```

```
[[0.85198549 0.0739036  0.89493176 0.43649355 0.12767773]
 [0.57585787 0.84047092 0.43512055 0.69591056 0.6846381 ]
 [0.70064837 0.77969426 0.64274937 0.96102617 0.10846489]
 [0.79610634 0.83258008 0.26600836 0.83668539 0.53212691]
 [0.51690756 0.09858771 0.91886899 0.66665849 0.17477948]
 [0.21769151 0.46787528 0.43589124 0.88935448 0.22259927]
 [0.58901937 0.27720157 0.52572218 0.25935711 0.52894863]
 [0.31214075 0.54416225 0.2420565  0.09423802 0.18946638]
 [0.15028533 0.89444684 0.3007521  0.27286447 0.00647975]
 [0.59801345 0.79435088 0.59862107 0.61498669 0.87010577]
 [0.72948669 0.76516178 0.98117598 0.52135838 0.53482608]
 [0.08298453 0.01905823 0.26417891 0.77072226 0.96964001]
 [0.3147297  0.49847345 0.2032428  0.68422641 0.8370478 ]
 [0.77362072 0.33219103 0.13979055 0.18148193 0.77215136]
 [0.12510639 0.81139091 0.69946877 0.69293721 0.659838  ]
 [0.93853729 0.84554181 0.28678798 0.72945576 0.40825844]
 [0.70259877 0.2926497  0.70089211 0.09127827 0.36000804]
 [0.08585043 0.48548256 0.24627121 0.67633576 0.82430543]
 [0.17854142 0.0199978  0.73323042 0.6815786  0.79907516]
 [0.21139877 0.982588   0.45313877 0.64182862 0.33975144]]
```

In [110]:

```
w.shape
```

Out[110]:

```
(20, 5)
```

In [111]:

```
w[0,0]
```

Out[111]:

```
0.8519854927300882
```

Initializing distance with 0

In [112]:

```
closest_node = np.zeros(total_number,dtype='int')
closest_node
```

Out[112]:

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

Adjusting weight in Training

In [113]:

```
d_list = list()
learning_rate = 0.5
distance = 0
neighbor_int = 3
neighbor_float = 3

while neighbor_float >= 0.0000001 :

    # calculating distance from input node to every output nodes
    for i in range(n):
        for j in range(n):
            for k in range(m):
                distance = distance + ((df.iloc[i,k]-w[j,k]) ** 2)
            d_list.append((j,distance))
            distance = 0

        # sorting the distances in ascending order
        for ii in range(len(d_list)):
            for jj in range(ii+1,len(d_list)):
                if d_list[jj][1] < d_list[ii][1]:
                    temp = d_list[ii]
                    d_list[ii] = d_list[jj]
                    d_list[jj] = temp

        # saving the closest node
        closest_node[i] = d_list[0][0]

        # updating weights of closest node and it's neighbor nodes
        for ii in range(neighbor_int+1):
            node = d_list[ii][0]
            for k in range(m):
                w[node][k] = w[node][k] + learning_rate * (df.iloc[i,k] - w[node][k])

        d_list = list()

    neighbor_float = neighbor_float - learning_rate * neighbor_float
    neighbor_int = int(np.ceil(neighbor_float))
```

In [114]:

```
closest_node[:no_of_train_data]
```

Out[114]:

```
array([ 7, 13,  8, 11,  4,  4,  5, 18,  1, 19,  3,  9, 15, 10,  2, 14, 12,
        6, 12, 17])
```

In [115]:

```
print('Training Input No.','\t','   Output node')
for i in range(no_of_train_data):
    print('\t',i,'\t\t\t',closest_node[i])
```

```
Training Input No.      Output node
  0      7
  1      13
  2      8
  3      11
  4      4
  5      4
  6      5
  7      18
  8      1
  9      19
  10      3
  11      9
  12      15
  13      10
  14      2
  15      14
  16      12
  17      6
  18      12
  19      17
```

Clustering testing nodes

In [116]:

```
# calculating distance from input node to every output nodes
for i in range(no_of_train_data,total_number):
    for j in range(n):
        for k in range(m):
            distance = distance + ((df.iloc[i,k]-w[j,k]) ** 2)
        d_list.append((j,distance))
        distance = 0

    # sorting the distances in ascending order
    for ii in range(len(d_list)):
        for jj in range(ii+1,len(d_list)):
            if d_list[jj][1] < d_list[ii][1]:
                temp = d_list[ii]
                d_list[ii] = d_list[jj]
                d_list[jj] = temp

    # saving the closest node
    closest_node[i] = d_list[0][0]
    d_list = list()
```

In [117]:

```
closest_node[no_of_train_data:]
```

Out[117]:

```
array([12,  4,  0, 18,  1, 19,  8,  3, 15, 10,  2, 14])
```

In [118]:

```
print('Testing Input No.','\t','   Output node')
for i in range(no_of_train_data,total_number):
    print('\t',i,'\t\t\t',closest_node[i])
```

```
Testing Input No.      Output node
  20      12
  21      4
```

```
  22      0
  23      18
  24      1
  25      19
  26      8
  27      3
  28      15
  29      10
  30      2
  31      14
```

In [119]:

```
print('All Input Node','\t','   Output node')
for i in range(total_number):
    print('\t',i,'\t\t\t',closest_node[i])
```

```
All Input Node      Output node
  0      7
  1      13
  2      8
  3      11
  4      4
  5      4
  6      5
  7      18
  8      1
  9      19
  10      3
  11      9
  12      15
  13      10
  14      2
  15      14
  16      12
  17      6
  18      12
  19      17
  20      12
  21      4
  22      0
  23      18
  24      1
  25      19
  26      8
  27      3
  28      15
  29      10
  30      2
  31      14
```

In [10]:

```
# n = 4
# for i in range(40):
#     n = n - 0.5 * n
#     print(n)
```