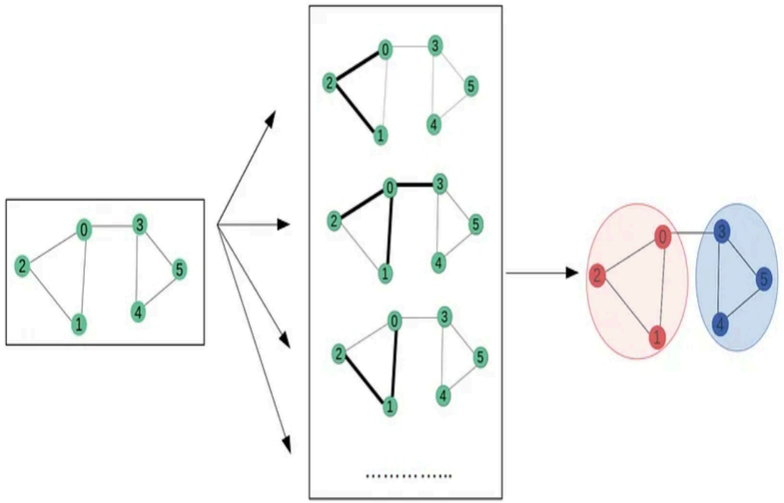
Graph algorithms in AI

Neural Networks have gained massive success in the last decade. However, early variants of Neural Networks could only be implemented using regular or Euclidean data, while a lot of data in the real world have underlying graph structures which are non-Euclidean. The non-regularity of data structures have led to recent advancements in Graph Neural Networks. In the past few years, different variants of Graph Neural Networks are being developed with Graph Convolutional Networks (GCN) being one of them. GCNs are also considered as one of the basic Graph Neural Networks variants.

Applications of GCN



Graph data processing

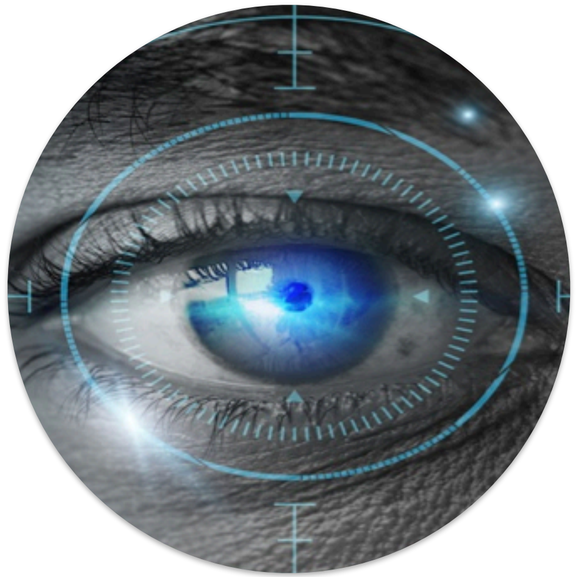


Image processing

Building Graph Convolutional Networks

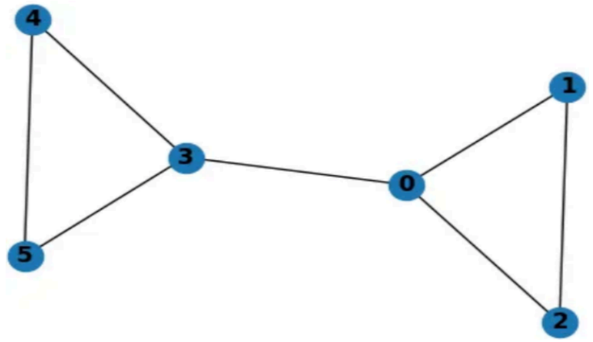
Let's start by building a simple undirected graph (G) using NetworkX.B

```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import fractional_matrix_power

import warnings
warnings.filterwarnings("ignore", category=UserWarning)


#Initialize the graph
G = nx.Graph(name='G')

#Create nodes
#In this example, the graph will consist of 6 nodes.
```

```python
#Each node is assigned node feature which corresponds to the node name
for i in range(6):
    G.add_node(i, name=i)


#Define the edges and the edges to the graph
edges = [(0,1),(0,2),(1,2),(0,3),(3,4),(3,5),(4,5)]
G.add_edges_from(edges)

#See graph info
print('Graph Info:\n', nx.info(G))

#Inspect the node features
print('\nGraph Nodes: ', G.nodes.data())

#Plot the graph
nx.draw(G, with_labels=True, font_weight='bold')
plt.show()
```

create_graph.py hosted with ❤ by GitHub view raw

Output:



Graph Info:

Name: G

Type: Graph

Number of nodes: 6

Number of edges: 7

Average degree: 2.3333

Graph Nodes: [(0,{‘name’:0}),(1,{‘name’:1}),(2,{‘name’:2}),(3,{‘name’:3}),(4,{‘name’:4}),(5,{‘name’:5})]

```python
#Get the Adjacency Matrix (A) and Node Features Matrix (X) as numpy array
A = np.array(nx.attr_matrix(G, node_attr='name')[0])
X = np.array(nx.attr_matrix(G, node_attr='name')[1])
X = np.expand_dims(X,axis=1)

print('Shape of A: ', A.shape)
print('\nShape of X: ', X.shape)
print('\nAdjacency Matrix (A):\n', A)
print('\nNode Features Matrix (X):\n', X)
```

adj_matrix.py hosted with ❤ by GitHub view raw

Output:

Shape of A: (6, 6)

Shape of X: (6, 1)

Adjacency Hatrix (A):

[[0. 1. 1. 1. 0. 0.]
[1. 0. 1. 0. 0. 0.]
[1. 1. 0. 0. 0. 0.]
[1. 0. 0. 0. 1. 1.]
[0. 0. 0. 1. 0. 1.]
[0. 0. 0. 1. 1. 0.]]

Node Features Matrix (X):

[[0]
[1]
[2]
[3]
[4]
[5]]

```
#Dot product Adjacency Matrix (A) and Node Features (X)
AX = np.dot(A,X)
print("Dot product of A and X (AX):\n", AX)
```

dot_prod.py hosted with ❤ by GitHub view raw

Output:

Dot product of A and X (AX):

[[6.]

[2.]

[1.]

[9.]

[8.]

[7.]]

```python
#Add Self Loops
G_self_loops = G.copy()

self_loops = []
for i in range(G.number_of_nodes()):
    self_loops.append((i,i))

G_self_loops.add_edges_from(self_loops)

#Check the edges of G_self_loops after adding the self loops
print('Edges of G with self-loops:\n', G_self_loops.edges)

#Get the Adjacency Matrix (A) and Node Features Matrix (X) of added self-lopps graph
A_hat = np.array(nx.attr_matrix(G_self_loops, node_attr='name')[0])
print('Adjacency Matrix of added self-loops G (A_hat):\n', A_hat)

#Calculate the dot product of A_hat and X (AX)
AX = np.dot(A_hat, X)
print('AX:\n', AX)
```

self_loop.py hosted with ❤ by GitHub view raw

1.1K | 9

Output:

Edges of G with self-loops:

[(0, 1), (0, 2), (0, 3), (0, 0), (0, 1), (1, 2), (1, 1), (2, 2), (3, 4), (3, 5), (3, 3), (4, 5), (4, 4), (5, 5)]

Adjacency Matrix of added self-loops G (A_hat):

[[1. 1. 1. 1. 0. 0.]
[1. 1. 1. 0. 0. 0.]
[1. 1. 1. 0. 0. 0.]
[1. 0. 0. 1. 1. 1.]
[0. 0. 0. 1. 1. 1.]
[0. 0. 0. 1. 1. 1.]]

AX:

[[6.]
[3.]
[3.]
[12.]
[12.]
[12.]]

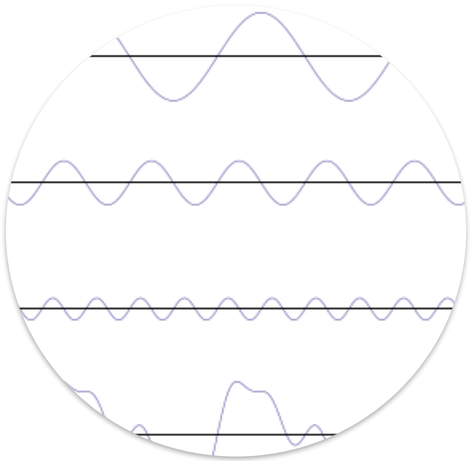
Normalized features = DAX

If we compare DAX with AX, we will notice that:

AX	DAX:
[[6.]	[[1.2]
[3.]	[0.75]
[3.]	[0.75]
[12.]	[2.4]
[12.]	[3.]
[12.]]	[3.]]

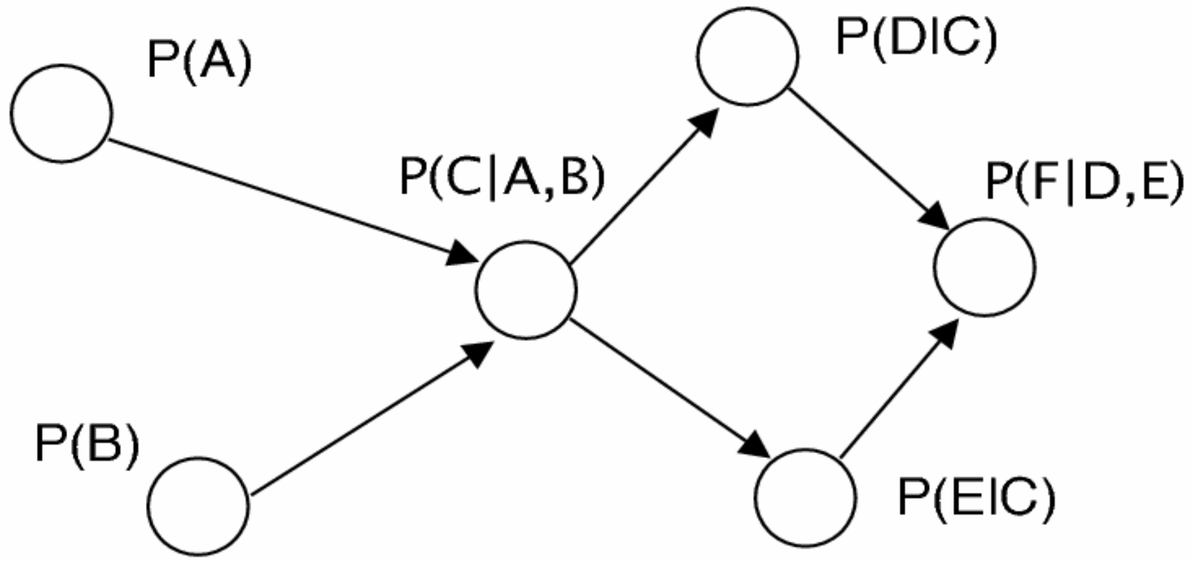
Applications of BP

Medical data analysis



Sound analysis

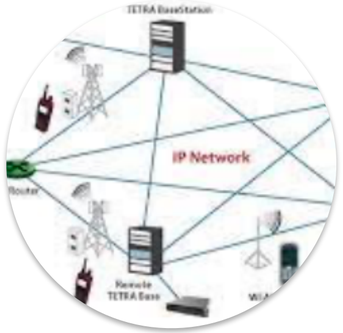
Face processing


P(A)
P(B)
P(C|A,B)
P(D|C)
P(E|C)
P(F|D,E)

```python
# تعریف مدل گرافیکی
model = BayesianModel([('A', 'B'), ('C', 'A'), ('C', 'B')])

# تعیین جدول احتمال شرطی برای متغیرها
cpd_A = TabularCPD(variable='A', variable_card=2, values=[[0.5], [0.5]])
cpd_B = TabularCPD(variable='B', variable_card=2, values=[[0.9, 0.1], [0.1, 0.9]],
                   evidence=['A', 'C'], evidence_card=[2, 3])
cpd_C = TabularCPD(variable='C', variable_card=3, values=[[0.3, 0.5, 0.2], [0.6, 0.2, 0.2], [0.1, 0.2, 0.7]])

# افزودن جدول‌های احتمال شرطی به مدل
model.add_cpds(cpd_A, cpd_B, cpd_C)

# ساخت شیء محاسبه برای الگوریتم انتشار باور
infer = BeliefPropagation(model)

# محاسبه توزیع شرطی متغیر B با توجه به A و C
joint_prob = infer.query(variables=['B'], evidence={'A': 0, 'C': 1})
b_prob = joint_prob.values
```

```
import numpy as np

# Define the Bayesian Network
network = {
    'Gender': {'parents': [], 'probabilities': [0.5, 0.5]},
    'Hair': {'parents': ['Gender'], 'probabilities': [[0.6, 0.4], [0.2, 0.8]]},
    'Glasses': {'parents': ['Gender'], 'probabilities': [[0.8, 0.2], [0.3, 0.7]]},
    'Smiling': {'parents': ['Gender', 'Hair'], 'probabilities': [[[0.9, 0.1], [0.6, 0.4]], [[0.7, 0.3], [0.4, 0.6]]]}
}

# Define the evidence (observed variables)
evidence = {'Gender': 0, 'Hair': 0}

# Initialize the belief (message) for each variable
beliefs = {}
for variable in network:
    beliefs[variable] = np.ones(len(network[variable]['probabilities']))

# Iterate over the network to propagate the belief (message)
for i in range(10):
    for variable in network:
        # Calculate the incoming message for the variable from its parents
        incoming_message = np.ones(len(network[variable]['probabilities']))
        for parent in network[variable]['parents']:
            parent_prob = beliefs[parent] * network[variable]['probabilities'][network[parent]['probabilities'] == evidence[parent]]
            incoming_message *= parent_prob.sum(axis=0)
        incoming_message /= incoming_message.sum()

        # Update the belief (message) for the variable
        beliefs[variable] = incoming_message

# Print the final belief for each variable
for variable in beliefs:
    print(f'{variable}: {beliefs[variable]}')
```

Applications of generative clustering algorithm

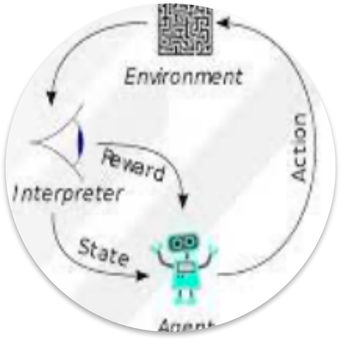


Network architecture

Social communication analysis

Machine learning

Image processing



Reinforcement learning

Clustering methods :

Probabilities methods

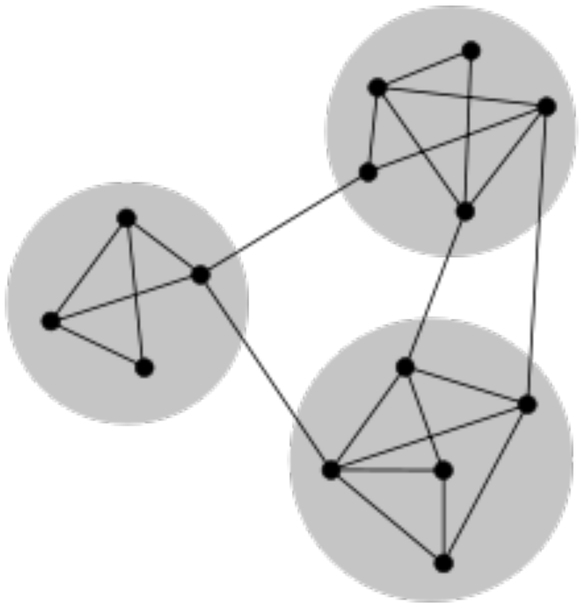
Distance-based methods

Density-based methods

Grid-based methods

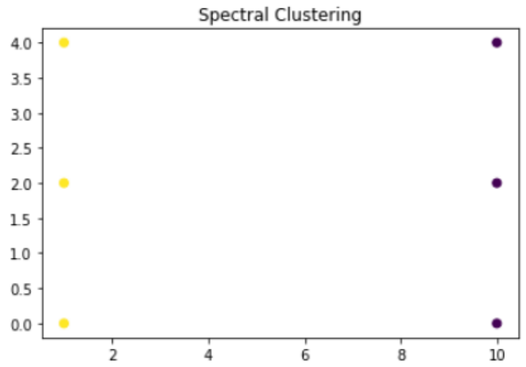
Factorization techniques

Spectral methods

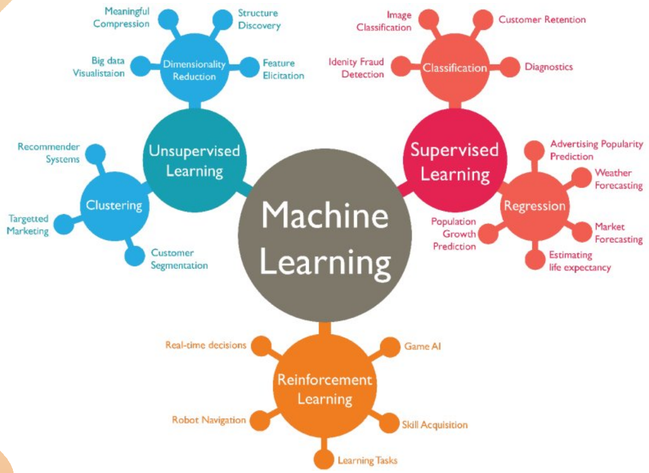


```python
import numpy as np
from sklearn.cluster import SpectralClustering
import matplotlib.pyplot as plt

# تعریف داده‌های آموزشی
X = np.array([[1, 2], [1, 4], [1, 0], [10, 2], [10, 4], [10, 0]])

# ایجاد شیئ مدل خوشه‌بندی
model = SpectralClustering(n_clusters=2, affinity='nearest_neighbors', n_neighbors=2)

# اعمال الگوریتم خوشه‌بندی بر روی داده‌های آموزشی
y_pred = model.fit_predict(X)

# ترسیم نمودار خوشه‌بندی
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.title('Spectral Clustering')
plt.show()
```

```
/home/iman/.local/lib/python3.10/site-packages/sklearn/manifold/_spectral_embedding.py:274: UserWarning: Graph is no
t fully connected, spectral embedding may not work as expected.
  warnings.warn(
```

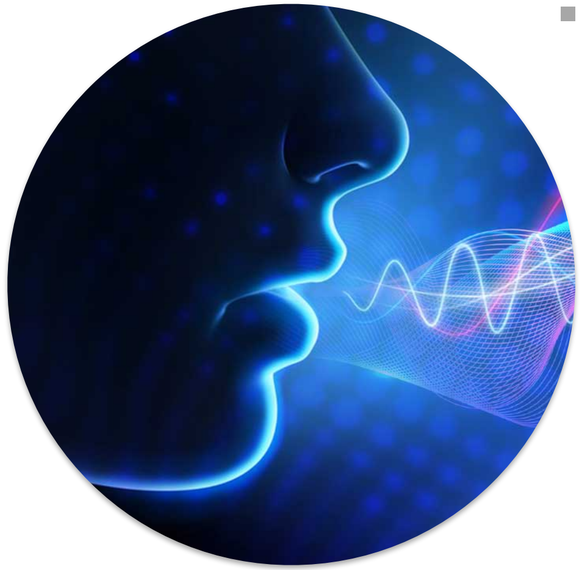


KNN algorithm

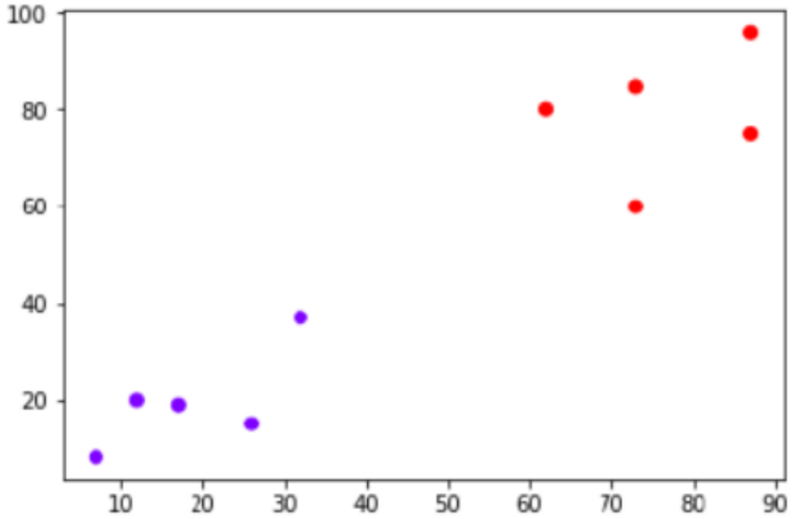


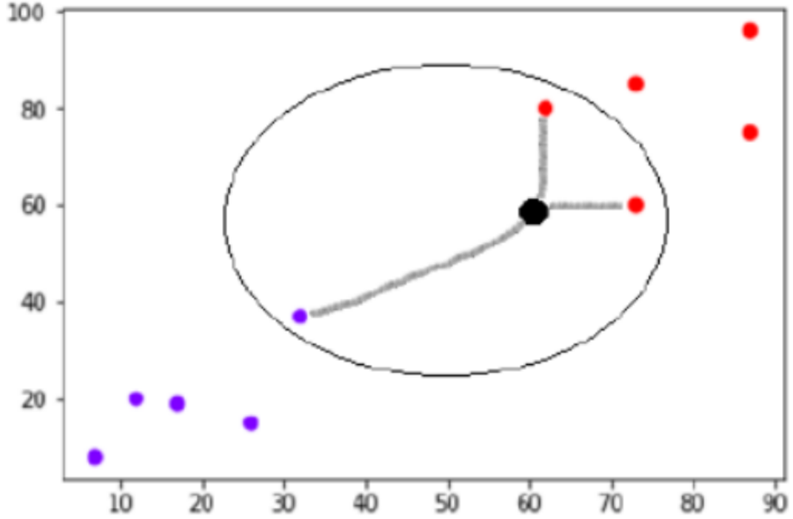
Machine Learning

Handwriting recognition



voice recognition


100
80
60
40
20
10
20
30
40
50
60
70
80
90


100
80
60
40
20
10
20
30
40
50
60
70
80
90

Get the Data

** Read the 'KNN_Project_Data csv file into a dataframe **

[8]
```
readfile = pd.read_csv('/home/iman/Desktop/KNN_Project-Data')

df = pd.DataFrame(readfile)
```

Check the head of the dataframe.

[9]
```
readfile.head()
```

...

	XVPM	GWYH	TRAT	TLLZ	IGGA	HYKR	EDFS	GUUB	MGJM	JHZC	TARGET CLASS
0	1636.670614	817.988525	2565.995189	358.347163	550.417491	1618.870897	2147.641254	330.727893	1494.878631	845.136088	0
1	1013.402760	577.587332	2644.141273	280.428203	1161.873391	2084.107872	853.404981	447.157619	1193.032521	861.081809	1
2	1300.035501	820.518697	2025.854469	525.562292	922.206261	2552.355407	818.676686	845.491492	1968.367513	1647.186291	1
3	1059.347542	1066.866418	612.000041	480.827789	419.467495	685.666983	852.867810	341.664784	1154.391368	1450.935357	0
4	1018.340526	1313.679056	950.622661	724.742174	843.065903	1370.554164	905.469453	658.118202	539.459350	1899.850792	0

Train Test Split

Use train_test_split to split your data into a training set and a testing set.

[10]
```
from sklearn.model_selection import train_test_split
```

[11]
```
X = df.iloc[:,:-1].values

Y = df.iloc[:,10].values

x_train , x_test , y_train , y_test = train_test_split(X , Y )
```

Using KNN

Import KNeighborsClassifier from scikit learn.

[1]
```
from sklearn.neighbors import KNeighborsClassifier
```

Create a KNN model instance with n_neighbors=1

```
[12] knn = KNeighborsClassifier(n_neighbors=1)
```

Fit this KNN model to the training data.

```
[14] knn.fit(x_train , y_train)
```

```
... KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1)
```

Predictions and Evaluations

Let's evaluate our KNN model!

Use the predict method to predict values using your KNN model and X_test.

```
[17]
pre = knn.predict(x_test)
pre
```

```
array([0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1,
       1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0,
       1, 1, 1, 0, 0, 0, 1, 0])
```