Indoor Localization using Graph Neural Networks

Paper : https://www.colibri.udelar.edu.uy/jspui/bitstream/20.500.12008/30375/1/LGLC21.pdf
Code : GitHub - facundolezama19/indoor-localization-gnn: Notebooks con los códigos utilizados para el proyecto final del curso.

INDEX

Introduction

Introduction

- **Background**
  - 스마트폰 접근성 ↑ , wifi 연결 유지 필요성 ↑
  - Device(phone)이 Access Point(AP)의 신호에 접근하고, 이는 고정되어 있기 때문에 실시간으로 사람의 위치를 추정할 수 있다.
  - Building is divided into non-overlapping zones, objective is to estimate the corresponding zone based on the power measurements
  - 목적 : input(device가 주변 AP로 부터 받은 전력 정보가 포함된 벡터) → 해당 구역이 어느 zone인지.
- **Approach**
  - Node : APs
  - Edge : distance between nodes(RSSI)
  - We propose a way to construct the graph using only the RSSI measurements (and not the floor plan)
- **Evaluate**
  - Dataset : MNAV and UjiIndoorLoc

Method

Method

❑ Spatial Information

- **Given RSSI of the n_{AP} , APs as received by the device.**
- **To learn how to map these values to the corresponding zone.**
  - $X \in \mathbb{R}^{n_{AP} \times F_{in}}$, F_{in} : 2.4GHz, 5GHz 등 대역폭 개수
  - $\Phi : \mathbb{R}^{n_{AP} \times F_{in}} \rightarrow \{1, 2, \dots, n_z\}$, n_z : *possible zones*
- **We define a graph where nodes are the APs and edges are related to the distance between them.**
- **즉 device가 APs로 부터 수신한 RSSI 에 따라 어느 건물, 혹은 몇 층 등의 possible zone 을 맞추는 것.**

RSSI Received Signal Strength Indication, Receive Signal Strength Indicator 수신 신호 강도

□ 잡음이 포함된 무선/RF `수신 신호 세기`에 대한 매우 일반적인 명칭

- 간단하게 측정하고 확인할 수 있는 수신 전파 신호의 세기를 말함 .
- 즉, 수신 전력(단위 : [dBm])을 나타냄 .. 통상, 안테나 및 수신기 회로 내부 손실은 제외

출처 : **[정보통신기술용어해설]**

Method

❑ Spatial Information

- If F_{in} = 2, obtain two weights per edge (단, 여기에서는 하나만 사용했을 때와 두 대역폭 모두 사용했을 때 차이가 크지 않기 때문에 2.4GHz에 해당하는 단일 가중치만 사용한다.)
- Positive weights만 고려하기 위해 minimum RSSI value는 모든 측정 단계에서 삭제한다. 이는 edge가 사라지게 할 수도 있는데 먼 거리를 효과적으로 반영할 수 있다.
- 결과로 나온 그래프가 꼭 대칭일 필요는 없다.

Method

❑ Spatial Information Process(Graph 구축 단계)

1. 특정 AP_i 가 주어진다면 AP_j(AP_i에 대한 측정 값이 있는 모든 instance)를 불러온다
2. Filtering (not over threshold) : only consider those instances that were obtained in an area near to the AP_i
3. In filtered subset, estimate the distance between AP_i and any other AP_j.
4. RSSI measurement : 3번 과정에서 구한 distance들의 평균
5. 이후 모든 node(=AP)에 대해 반복하여 $RSSI$ update.
6. 최종 RSSI = edge weight

Model

❑ Graph Neural Network

- RSSI measurement : signal
- Each node i has vector x_i
- 목적 : signal에 따라 $n_z(possible\ zones)$로 분류 하는 것.

- CNN과 흡사.
  - Convolution layer : called GSO(Graph Shift Operator) , S ∈ $\mathbb{R}^{n_{AP} \times n_{AP}}$
    - $layer(S)$: $n_{AP} \times n_{AP}$ matrix representation of the graph
  - Aggregate at each node the information of its neighbors
    - $SX = Y \rightarrow S^K X = S(S^{K-1}X)$, $S: matrix(graph), X: Input, K = hope$
    - $if\ single-layer\ GNN \rightarrow Y = \sigma\left(\sum_{k=0}^{K-1} S^k X H_k\right)$, k=signal K=hope, H=output dimension
- Classifier layer : output dimension n_z
- Use softmax function, Cross entropy loss function

Data Preprocess

Data Preprocess

1. Data 초기 형태

	location	wifi-dc:a5:f4:43:85:c0	wifi-dc:a5:f4:43:27:e0	wifi-f8:4f:57:ab:da:00	wifi-5c:a4:8a:4c:05:c0	wifi-1c:1d:86:ce:ef:b0	wifi-dc:a5:f4:43:79:20	wifi-c0:7b:bc:36:9e:10	wifi-1c:1d:86:9f:99:20	wifi-c0:7b:bc:36:af:40
0	location_7	-76.0	NaN	NaN	-91.0	-80.0	-74.0	-79.0	-78.0	-77.0
1	location_3	-81.0	-93.0	-88.0	-87.0	-94.0	-87.0	-85.0	-79.0	-79.0
2	location_8	-87.0	-90.0	NaN	-83.0	NaN	-79.0	-93.0	-83.0	-80.0
3	location_10	-84.0	-93.0	-88.0	-93.0	NaN	-89.0	-93.0	-87.0	-85.0
4	location_3	-78.0	-94.0	-90.0	-86.0	NaN	-88.0	-86.0	-79.0	-79.0

다음과 같은 location 에 따라 설치된 wifi 종류, 신호세기를 value로 가지는 table 형태의 데이터를 구축

- 여기서는 기본 제공되는 dataset(https://github.com/ffedee7/posifi_mnav/tree/master/data_analysis)

Data Preprocess

2. 전 처리

1) Fillna(0)을 통해 NaN값 처리
2) 신호세기에 +100을 하여 양수로 변환(도메인 지식에 근거)
3) Wifi 식별 번호를 AP_i 로 변환.

	location	wifi-dc:a5:f4:43:85:c0	wifi-dc:a5:f4:43:27:e0	wifi-f8:4f:57:ab:da:00	wifi-5c:a4:8a:4c:05:c0	wifi-1c:1d:86:ce:ef:b0	wifi-dc:a5:f4:43:79:20	wifi-c0:7b:bc:36:9e:10	wifi-1c:1d:86:9f:99:20	c
0	location_7	24.0	0.0	0.0	9.0	20.0	26.0	21.0	22.0	
1	location_3	19.0	7.0	12.0	13.0	6.0	13.0	15.0	21.0	
2	location_8	13.0	10.0	0.0	17.0	0.0	21.0	7.0	17.0	
3	location_10	16.0	7.0	12.0	7.0	0.0	11.0	7.0	13.0	
4	location_3	22.0	6.0	10.0	14.0	0.0	12.0	14.0	21.0	
...	...	...	...	...	...	...	...	...	...	
10464	location_7	23.0	17.0	17.0	15.0	21.0	30.0	23.0	25.0	
10465	location_8	14.0	6.0	15.0	6.0	13.0	16.0	14.0	17.0	
10466	location_2	36.0	46.0	33.0	0.0	0.0	35.0	13.0	7.0	
10467	location_8	19.0	0.0	0.0	23.0	8.0	20.0	10.0	17.0	
10468	location_5	25.0	20.0	15.0	0.0	17.0	27.0	31.0	31.0	

10469 rows × 16 columns

	location	AP1	AP2	AP3	AP4	AP5	AP6	AP7	AP8	AP9	AP10	AP11	AP12	AP13	AP14	AP15
0	location_7	24.0	0.0	0.0	9.0	20.0	26.0	21.0	22.0	23.0	16.0	27.0	32.0	31.0	27.0	8.0
1	location_3	19.0	7.0	12.0	13.0	6.0	13.0	15.0	21.0	21.0	16.0	29.0	25.0	34.0	29.0	0.0
2	location_8	13.0	10.0	0.0	17.0	0.0	21.0	7.0	17.0	20.0	9.0	27.0	22.0	31.0	21.0	0.0
3	location_10	16.0	7.0	12.0	7.0	0.0	11.0	7.0	13.0	15.0	0.0	8.0	9.0	30.0	14.0	0.0
4	location_3	22.0	6.0	10.0	14.0	0.0	12.0	14.0	21.0	21.0	15.0	34.0	20.0	33.0	28.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
10464	location_7	23.0	17.0	17.0	15.0	21.0	30.0	23.0	25.0	28.0	19.0	28.0	58.0	42.0	32.0	0.0
10465	location_8	14.0	6.0	15.0	6.0	13.0	16.0	14.0	17.0	16.0	17.0	26.0	37.0	27.0	11.0	6.0
10466	location_2	36.0	46.0	33.0	0.0	0.0	35.0	13.0	7.0	8.0	16.0	0.0	8.0	23.0	33.0	39.0
10467	location_8	19.0	0.0	0.0	23.0	8.0	20.0	10.0	17.0	28.0	10.0	35.0	13.0	42.0	26.0	0.0
10468	location_5	25.0	20.0	15.0	0.0	17.0	27.0	31.0	31.0	28.0	23.0	24.0	28.0	35.0	22.0	7.0

10469 rows × 16 columns

Data Preprocess

3. train, test dataset 준비

```
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
enc = OrdinalEncoder(dtype=np.int)
y = enc.fit_transform(df_data['location'].values.reshape(-1,1))
X = df_data.iloc[:,1:].values
```

```
y
```

```
array([[13],
       [ 9],
       [14],
       ...,
       [ 8],
       [14],
       [11]])
```

```
X
```

```
array([[24.,  0.,  0., ..., 31., 27.,  8.],
       [19.,  7., 12., ..., 34., 29.,  0.],
       [13., 10.,  0., ..., 31., 21.,  0.],
       ...,
       [36., 46., 33., ..., 23., 33., 39.],
       [19.,  0.,  0., ..., 42., 26.,  0.],
       [25., 20., 15., ..., 35., 22.,  7.]])
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_G = X_train.copy()
```

Label = location
- Location은 다중 범주형 변수이므로 ordinalencoder 사용

X = location을 제외한 table.
- 각 value는 신호세기에 해당함(RSSI measurement)

X_train, X_test, y_train, y_test split
- test_size = 0.2

Data Preprocess

4. Graph 구축

1. 특정 AP_i 가 주어진다면 AP_j(AP_i에 대한 측정 값이 있는 모든 instance)를 불러온다
2. Filtering (not over threshold) : only consider those instances that were obtained in an area near to the AP_i
3. In filtered subset, estimate the distance between AP_i and any other AP_j.
4. RSSI measurement : 3번 과정에서 구한 distance 들의 평균
5. 이후 모든 node(=AP)에 대해 반복하여 $RSSI$ update.
6. 최종 RSSI = edge weight

```
df_data_train = pd.DataFrame(X_G, columns=df_data.columns[1:])
df_G = pd.DataFrame(columns = ['from', 'to', 'weight'])

th = 10

for i in range(1,16):
  max_val = df_data_train['AP'+str(i)].max()
  df_aux_i = df_data_train[df_data_train['AP'+str(i)]  > (max_val - th)]
  df_aux_i = df_aux_i.drop('AP'+str(i), axis=1)
  df_aux_i.head()

  for k, v in df_aux_i.mean().items():
    df_G = df_G.append({'from':'AP'+str(i), 'to': k, 'weight': v}, ignore_index=True)
```

```
df_data_train
```

	AP1	AP2	AP3	AP4	AP5	AP6	AP7	AP8	AP9	AP10	AP11	AP12	AP13	AP14	AP15
0	20.0	0.0	0.0	13.0	12.0	6.0	8.0	15.0	13.0	14.0	30.0	23.0	26.0	19.0	0.0
1	23.0	0.0	0.0	56.0	29.0	18.0	9.0	17.0	34.0	0.0	15.0	7.0	24.0	7.0	0.0
2	29.0	26.0	12.0	16.0	33.0	29.0	32.0	32.0	22.0	13.0	0.0	18.0	28.0	22.0	0.0
3	24.0	10.0	17.0	10.0	21.0	30.0	18.0	23.0	20.0	22.0	34.0	35.0	31.0	21.0	0.0
4	19.0	0.0	17.0	7.0	19.0	21.0	17.0	27.0	16.0	20.0	24.0	44.0	34.0	29.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8370	18.0	9.0	0.0	11.0	11.0	27.0	6.0	16.0	19.0	10.0	27.0	28.0	32.0	20.0	0.0
8371	36.0	25.0	28.0	0.0	0.0	39.0	27.0	23.0	25.0	19.0	22.0	21.0	34.0	22.0	14.0
8372	20.0	0.0	0.0	8.0	6.0	20.0	0.0	19.0	29.0	23.0	26.0	24.0	39.0	35.0	0.0
8373	34.0	15.0	16.0	25.0	15.0	30.0	24.0	32.0	32.0	18.0	26.0	12.0	29.0	17.0	0.0
8374	29.0	12.0	17.0	13.0	0.0	20.0	10.0	24.0	32.0	0.0	17.0	9.0	34.0	21.0	0.0

8375 rows × 15 columns

Data Preprocess

4. Graph 구축

```
df_data_train = pd.DataFrame(X_G, columns=df_data.columns[1:])
df_G = pd.DataFrame(columns = ['from', 'to', 'weight'])

th = 10

for i in range(1,16):
  max_val = df_data_train['AP'+str(i)].max()
  df_aux_i = df_data_train[df_data_train['AP'+str(i)]  > (max_val - th)]
  df_aux_i = df_aux_i.drop('AP'+str(i), axis=1)
  df_aux_i.head()

  for k, v in df_aux_i.mean().items():
    df_G = df_G.append({'from':'AP'+str(i), 'to': k, 'weight': v}, ignore_index=True)
```

```
df_data_train
```

	AP1	AP2	AP3	AP4	AP5	AP6	AP7	AP8	AP9	AP10	AP11	AP12	AP13	AP14	AP15
0	20.0	0.0	0.0	13.0	12.0	6.0	8.0	15.0	13.0	14.0	30.0	23.0	26.0	19.0	0.0
1	23.0	0.0	0.0	56.0	29.0	18.0	9.0	17.0	34.0	0.0	15.0	7.0	24.0	7.0	0.0
2	29.0	26.0	12.0	16.0	33.0	29.0	32.0	32.0	22.0	13.0	0.0	18.0	28.0	22.0	0.0
3	24.0	10.0	17.0	10.0	21.0	30.0	18.0	23.0	20.0	22.0	34.0	35.0	31.0	21.0	0.0
4	19.0	0.0	17.0	7.0	19.0	21.0	17.0	27.0	16.0	20.0	24.0	44.0	34.0	29.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
8370	18.0	9.0	0.0	11.0	11.0	27.0	6.0	16.0	19.0	10.0	27.0	28.0	32.0	20.0	0.0
8371	36.0	25.0	28.0	0.0	0.0	39.0	27.0	23.0	25.0	19.0	22.0	21.0	34.0	22.0	14.0
8372	20.0	0.0	0.0	8.0	6.0	20.0	0.0	19.0	29.0	23.0	26.0	24.0	39.0	35.0	0.0
8373	34.0	15.0	16.0	25.0	15.0	30.0	24.0	32.0	32.0	18.0	26.0	12.0	29.0	17.0	0.0
8374	29.0	12.0	17.0	13.0	0.0	20.0	10.0	24.0	32.0	0.0	17.0	9.0	34.0	21.0	0.0

8375 rows × 15 columns

max_val = AP_i(column)마다 max 값 추출 → $Threshold = (\max - th)$

df_aux_i = subset(after filtering)
- AP_i(column)에서 max-th 보다 큰 값을 가진 instance만 추출 → $Filtering$
- 해당 instance 로만 구성된 subset 저장

df_aux_i = df_aux_i.drop('AP_i', axis=1)
- $table$에서 AP_i column 제거(AP_j만 얻기 위함)

df_aux_i.mean().items():
- k, v → 각 column명(AP_j) 와 평균값
- 따라서 반복문을 통해 AP_i → AP_j table(df_G) 생성

이걸 모든 node에 대해 반복.

Data Preprocess

4. Graph 구축

df_G

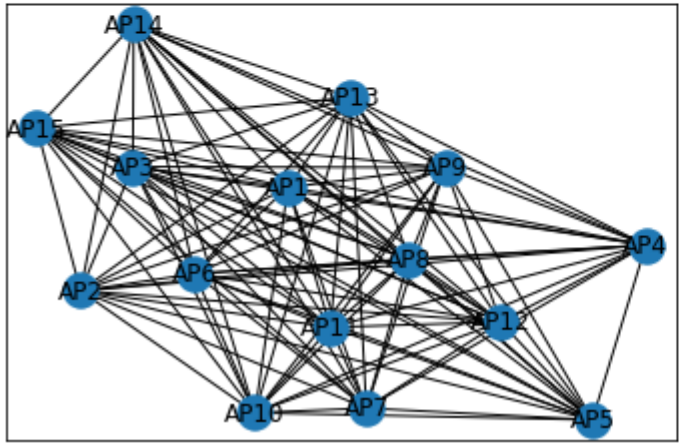
	from	to	weight
0	AP1	AP2	38.947368
1	AP1	AP3	44.631579
2	AP1	AP4	0.947368
3	AP1	AP5	1.263158
4	AP1	AP6	34.368421
...	...	...	...
205	AP15	AP10	8.166667
206	AP15	AP11	1.600000
207	AP15	AP12	3.616667
208	AP15	AP13	23.850000
209	AP15	AP14	36.916667

210 rows × 3 columns

```python
import networkx as nx
```

```python
G = nx.from_pandas_edgelist(df_G, source='from', target='to', edge_attr='weight')  # create_using=nx.DiGraph
```

```python
nx.draw_networkx(G)
plt.show()
```

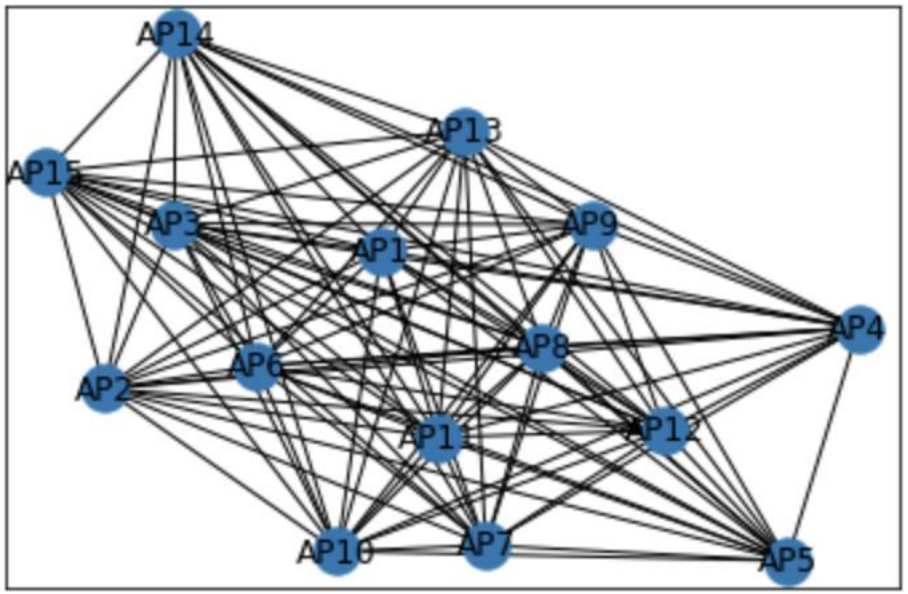


networkx 라이브러리를 이용하여 Graph 시각화

Graph 표현 : edge list
- threshold 값을 넘는 값들의 평균으로 edge(weight) 설정

Data Preprocess

4. Graph 구축




```
nx.to_numpy_array(G)
```

```
array([[ 0.        , 29.27777778, 28.63265306, 21.74137931, 15.02272727,
        30.25      , 27.5       , 30.86111111, 30.43181818, 24.76595745,
        24.51282051, 20.7037037 , 28.56818182, 41.83333333, 30.96666667],
       [29.27777778,  0.        , 31.92517007,  2.60344828,  7.93181818,
        53.875     , 28.475     , 20.22222222, 14.47727273, 22.91489362,
         9.15897436, 10.51851852, 14.76515152, 34.33333333, 47.56666667],
       [28.63265306, 31.92517007,  0.        ,  1.39655172,  3.63636364,
        34.5625    , 12.8       , 19.52777778, 17.27272727, 16.4893617 ,
        13.63589744, 10.2962963 , 17.25757576, 45.16666667, 33.73333333],
       [21.74137931,  2.60344828,  1.39655172,  0.        , 33.11363636,
         0.4375    , 14.775     , 21.05555556, 25.11363636, 15.36170213,
        18.29230769,  7.51851852, 12.96969697,  0.        ,  0.        ],
       [15.02272727,  7.93181818,  3.63636364, 33.11363636,  0.        ,
         6.8125    , 19.2       , 22.58333333, 15.15909091, 14.91489362,
         9.41538462, 18.11111111,  8.96212121,  0.        ,  0.4       ],
       [30.25      , 53.875     , 34.5625    ,  0.4375    ,  6.8125    ,
         0.        , 36.575     , 32.13888889, 24.        , 32.65957447,
        17.4       , 25.11111111, 16.32575758, 31.        , 39.26666667],
       [27.5       , 28.475     , 12.8       , 14.775     , 19.2       ,
        36.575     ,  0.        , 25.52777778, 21.31818182, 35.63829787,
        19.37435897, 21.03703704, 18.57575758,  0.        , 10.95      ],
       [30.86111111, 20.22222222, 19.52777778, 21.05555556, 22.58333333,
        32.13888889, 25.52777778,  0.        , 29.27272727, 26.78723404,
        24.44615385, 26.66666667, 23.32575758,  5.16666667, 13.36666667],
       [30.43181818, 14.47727273, 17.27272727, 25.11363636, 15.15909091,
        24.        , 21.31818182, 29.27272727,  0.        , 26.9787234 ,
        36.41025641, 24.66666667, 32.20454545, 18.83333333, 14.41666667],
       [24.76595745, 22.91489362, 16.4893617 , 15.36170213, 14.91489362,
        32.65957447, 35.63829787, 26.78723404, 26.9787234 ,  0.        ,
        17.83076923, 20.25925926, 16.85606061,  0.        ,  8.16666667],
       [24.51282051,  9.15897436, 13.63589744, 18.29230769,  9.41538462,
        17.4       , 19.37435897, 24.44615385, 36.41025641, 17.83076923,
         0.        , 26.7037037 , 31.6969697 ,  1.5       ,  1.6       ],
       [20.7037037 , 10.51851852, 10.2962963 ,  7.51851852, 18.11111111,
        25.11111111, 21.03703704, 26.66666667, 24.66666667, 20.25925926,
        26.7037037 ,  0.        , 24.10606061,  2.33333333,  3.61666667],
       [28.56818182, 14.76515152, 17.25757576, 12.96969697,  8.96212121,
        16.32575758, 18.57575758, 23.32575758, 32.20454545, 16.85606061,
        31.6969697 , 24.10606061,  0.        , 26.66666667, 23.85      ],
       [41.83333333, 34.33333333, 45.16666667,  0.        ,  0.        ,
        31.        ,  0.        ,  5.16666667, 18.83333333,  0.        ,
         1.5       ,  2.33333333, 26.66666667,  0.        , 36.91666667],
       [30.96666667, 47.56666667, 33.73333333,  0.        ,  0.4       ,
        39.26666667, 10.95      , 13.36666667, 14.41666667,  8.16666667,
         1.6       ,  3.61666667, 23.85      , 36.91666667,  0.        ]])
```

Data Preprocess

4. Graph 구축

```
nx.to_numpy_array(G)
```

```
array([[ 0.        , 29.27777778, 28.63265306, 21.74137931, 15.02272727,
        30.25      , 27.5       , 30.86111111, 30.43181818, 24.76595745,
        24.51282051, 20.7037037 , 28.56818182, 41.83333333, 30.96666667],
       [29.27777778,  0.        , 31.92517007,  2.60344828,  7.93181818,
        53.875     , 28.475     , 20.22222222, 14.47727273, 22.91489362,
         9.15897436, 10.51851852, 14.76515152, 34.33333333, 47.56666667],
       [28.63265306, 31.92517007,  0.        ,  1.39655172,  3.63636364,
        34.5625    , 12.8       , 19.52777778, 17.27272727, 16.4893617 ,
        13.63589744, 10.2962963 , 17.25757576, 45.16666667, 33.73333333],
       [21.74137931,  2.60344828,  1.39655172,  0.        , 33.11363636,
         0.4375    , 14.775     , 21.05555556, 25.11363636, 15.36170213,
        18.29230769,  7.51851852, 12.96969697,  0.        ,  0.        ],
       [15.02272727,  7.93181818,  3.63636364, 33.11363636,  0.        ,
         6.8125    , 19.2       , 22.58333333, 15.15909091, 14.91489362,
         9.41538462, 18.11111111,  8.96212121,  0.        ,  0.4       ],
       [30.25      , 53.875     , 34.5625    ,  0.4375    ,  6.8125    ,
         0.        , 36.575     , 32.13888889, 24.        , 32.65957447,
        17.4       , 25.11111111, 16.32575758, 31.        , 39.26666667],
       [27.5       , 28.475     , 12.8       , 14.775     , 19.2       ,
        36.575     ,  0.        , 25.52777778, 21.31818182, 35.63829787,
        19.37435897, 21.03703704, 18.57575758,  0.        , 10.95      ],
       [30.86111111, 20.22222222, 19.52777778, 21.05555556, 22.58333333,
        32.13888889, 25.52777778,  0.        , 29.27272727, 26.78723404,
        24.44615385, 26.66666667, 23.32575758,  5.16666667, 13.36666667],
       [30.43181818, 14.47727273, 17.27272727, 25.11363636, 15.15909091,
        24.        , 21.31818182, 29.27272727,  0.        , 26.9787234 ,
        36.41025641, 24.66666667, 32.20454545, 18.83333333, 14.41666667],
       [24.76595745, 22.91489362, 16.4893617 , 15.36170213, 14.91489362,
        32.65957447, 35.63829787, 26.78723404, 26.9787234 ,  0.        ,
        17.83076923, 20.25925926, 16.85606061,  0.        ,  8.16666667],
       [24.51282051,  9.15897436, 13.63589744, 18.29230769,  9.41538462,
        17.4       , 19.37435897, 24.44615385, 36.41025641, 17.83076923,
         0.        , 26.7037037 , 31.6969697 ,  1.5       ,  1.6       ],
       [20.7037037 , 10.51851852, 10.2962963 ,  7.51851852, 18.11111111,
        25.11111111, 21.03703704, 26.66666667, 24.66666667, 20.25925926,
        26.7037037 ,  0.        , 24.10606061,  2.33333333,  3.61666667],
       [28.56818182, 14.76515152, 17.25757576, 12.96969697,  8.96212121,
        16.32575758, 18.57575758, 23.32575758, 32.20454545, 16.85606061,
        31.6969697 , 24.10606061,  0.        , 26.66666667, 23.85      ],
       [41.83333333, 34.33333333, 45.16666667,  0.        ,  0.        ,
        31.        ,  0.        ,  5.16666667, 18.83333333,  0.        ,
         1.5       ,  2.33333333, 26.66666667,  0.        , 36.91666667],
       [30.96666667, 47.56666667, 33.73333333,  0.        ,  0.4       ,
        39.26666667, 10.95       , 13.36666667, 14.41666667,  8.16666667,
         1.6       ,  3.61666667, 23.85      , 36.91666667,  0.        ]])
```

```
W = nx.to_numpy_array(G)
# por último le saco la diagonal y la normalizo por su vector propio más grande
np.fill_diagonal(W,0)
```

W(weight) : n*n matrix

- 인접행렬로 표현됨.
- 대각행렬 값은 0으로 치환(self-loop 고려 x)

Model

Code : https://github.com/alelab-upenn/graph-neural-networks/blob/master/alegnn/modules/architectures.py

Model

❑ Graph Neural Network

- RSSI measurement : signal
- Each node i has vector x_i
- 목적 : signal에 따라 $n_z(possible\ zones)$로 분류 하는 것.

- CNN과 흡사.
  - Convolution layer : called GSO(Graph Shift Operator) , S ∈ $\mathbb{R}^{n_{AP} \times n_{AP}}$
    - $layer(S)$: $n_{AP} \times n_{AP}$ matrix representation of the graph
  - Aggregate at each node the information of its neighbors
    - $SX = Y \rightarrow S^K X = S(S^{K-1}X),\ S: matrix(graph), X: Input\ , K = hope$
    - $if\ single-layer\ GNN \rightarrow Y = \sigma\left(\sum_{k=0}^{K-1} S^k X H_k\right)$, k=signal K=hope, H=output dimension
- Classifier layer : output dimension n_z
- Use softmax function, Cross entropy loss function

Model

```
import alegnn.modules.architectures as architectures
import alegnn.utils.graphML as graphML
```

```
gnn_model = architectures.SelectionGNN(dimNodeSignals=[2, 20, 20], nFilterTaps=[3,3], bias=True, nonlinearity=torch.nn.ReLU, nSelectedNodes=[15, 15],
#gnn_model = architectures.SelectionGNN(dimNodeSignals=[1], nFilterTaps=[], bias=True, nonlinearity=torch.nn.ReLU, nSelectedNodes=[], poolingFunction=g
```

gnn_model = architectures.SelectionGNN(dimNodeSignals=[2,20,20], nFilterTaps=[3,3], bias=True, nonlinearity=torch.nn.ReLU,
nSelectedNodes=[15,15], poolingFunction=graphML.NoPool, poolingSize=[15,15], dimLayersMLP=[16],
GSO=torch.from_numpy(W).float())

dimNodesignals:(list of int) dimension of the signals at each layer(i.e. number of features at each node, or size of the vector supported at each node)

nFilterTaps (list of int): number of filter taps on each layer (i.e. nFilterTaps-1 is the extent of neighborhoods that are reached, for example K=2 is info from the 1-hop neighbors)

bias (bool): include bias after graph filter on every layer

nSelectedNodes (list of int): number of nodes to keep after pooling on each layer

dimLayersMLP (list of int): number of output hidden units of a sequence of fully connected layers after the graph filters have been applied
- bias=True 시 dim 1 증가

모델 아키텍쳐 : https://github.com/alelab-upenn/graph-neural-networks/blob/master/alegnn/modules/architectures.py

Train

Train

```
from sklearn.metrics import accuracy_score
loss = torch.nn.CrossEntropyLoss()

def train_model(model, train_data, test_data, batch_size=32, n_epochs=100, epsilon=0.005, weight_decay=1e-2):

    optimizer = torch.optim.Adam(model.parameters(), lr=epsilon, weight_decay=weight_decay)
    # scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=25, gamma=0.1)

    train_loader = torch.utils.data.DataLoader(dataset=train_data, batch_size=batch_size, shuffle=False)
    # Voy a mirar el loss en test en todo el conjunto de testing. Igual es bastante chico.
    test_loader = torch.utils.data.DataLoader(dataset=test_data, batch_size=len(test_data), shuffle=False)

    train_loss = []
    test_loss = []
    for epoch in range(n_epochs):
        for x_batch, y_batch in train_loader:
          if y_batch.shape[0] == batch_size:
            model.zero_grad()
            y_hat = model(x_batch)
            loss_result = loss(y_hat, y_batch.reshape(batch_size).type(torch.long))
            loss_result.backward()
            optimizer.step()
            # optimizer.zero_grad()
            train_loss.append(loss_result.item())
          else:
            break


        # scheduler.step()

        # Después de cada epoch miramos el test loss
        for x_batch, y_batch in test_loader:
            y_hattest = model(x_batch)
            test_loss.append(loss(y_hattest,y_batch.reshape(y_batch.shape[0]).type(torch.long)).item())

            #m = torch.nn.Softmax(dim=1)
            #output = m(y_hattest)

            #print(y_batch.shape)
            #print(y_batch.reshape(y_batch.shape[0]).shape)
            #print(y_hattest.shape)
            #print(np.array(torch.argmax(output, axis=1)).shape)
            #print(accuracy_score(y_batch.reshape(y_batch.shape[0]), np.array(torch.argmax(output, axis=1))))
            #print()

    return (model, train_loss, test_loss, y_hattest)
```

loss = torch.nn.CrossEntropyLoss()

optimizer = torch.optim.Adam(model.parameters(), lr=0.005, weight_decay = 1e-2)

- lr : learning rate
- weight_decay(가중치 감소) : weight Regularization

train_loader, test_loader = torch.utils.data.DataLoader() 사용

- batch_size 만큼 진행

Experimental Results

Experimental Results

❑ Dataset 1. UjiIndoorLoc

- This dataset was designed to test WLAN fingerprinting techniques.
- 4층 이상이고 최소 110,000m2의 면적을 포함하는 Jaume I 대학의 3개 건물에서 수집됨.
- Floor and BuildingID were used as labels.

1. BuildingID를 분류
2. BuildingID 별로 기기의 floor를 분류
3. 어느 빌딩, 몇 층인지 분류

TABLE I
CLASSIFIER PERFORMANCE ON EACH BUILDING SEPARATELY USING THE UJIINDOORLOC DATASET.

BuildingID	Floors	Test Accuracy	Instances	Number of APs
0	4	95.5 %	5249	69
1	4	82.4 %	5196	166
2	5	92.1 %	9492	53

BuildingID 별로 floor 분류

Experimental Results

1. **BuildingID를 분류 : 99% Accuracy**
2. **BuildingID 별로 기기의 floor를 분류 : Table 1**

TABLE I
CLASSIFIER PERFORMANCE ON EACH BUILDING SEPARATELY USING THE UJIINDOORLOC DATASET.

BuildingID	Floors	Test Accuracy	Instances	Number of APs
0	4	95.5 %	5249	69
1	4	82.4 %	5196	166
2	5	92.1 %	9492	53

BuildingID 별로 floor 분류

3. **어느 빌딩, 몇 층인지 분류**

   Label : BuildingID + Floors 총 13개.

   92.3% Accuracy (KNN : 85.5%)

Conclusions

Conclusion

- Indoor localization Application에 GNN을 적용해 봄.
  - Convolution 연산처럼 shift operation을 수행하는 GSO layer 사용.
  - GCN과는 전혀 다른 모델.
- 그래프 표현 : Edge list
  - RSSI를 이용하여 weight 정의
- 건물의 평면도(complete floor plan)가 없더라도 사용 가능한 기법
- 주요 Application
  - 기존 localization은 GPS 기반 위치추적이기 때문에 실내 위치추적은 불가능하다.
  - 그에 대한 solution 중 하나로 wifi(AP)의 RSSI 를 측정하여 Graph를 형성하고 GNN을 통해 학습시키는 방법 제시
    - 미아, 반려견 찾기
    - 실내 네비게이션(스타디움 좌석 안내, 쇼핑몰 위치 안내, 박물관 위치 안내)