Overview

It demonstrats how to utilize the unified Wi-Fi dataset.
The Neural Net model is not optimized, there's much space to improve the score.

In this notebook, I refer these two excellent notebooks.

- wifi features with lightgbm/KFold by @hiro5299834
  I took some code fragments from his notebook.
- Simple 👌 99% Accurate Floor Model 💯 by @nigelhenry
  I use his excellent work, the "floor" prediction.

It takes much much time to finish learning.
And even though I enable the GPU, it doesn't help.
If anybody knows how to make it better, can you please make a comment?

Thank you!

In [1]:
```
import numpy as np
import pandas as pd
import scipy.stats as stats
from pathlib import Path
import glob
import pickle

import random
import os

from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import StandardScaler, LabelEncoder

import tensorflow as tf
import tensorflow.keras.layers as L
import tensorflow.keras.models as M
import tensorflow.keras.backend as K
import tensorflow_addons as tfa
from tensorflow_addons.layers import WeightNormalization
from tensorflow.keras.callbacks import ReduceLROnPlateau, ModelCheckpoint, EarlyStopping
```

options

We can change the way it learns with these options.
Especialy **NUM_FEATS** is one of the most important options.
It determines how many features are used in the training.
We have 100 Wi-Fi features in the dataset, but 100th Wi-Fi signal sounds not important, right?
So we can use top Wi-Fi signals if we think we need to.

In [2]:
```
# options

N_SPLITS = 10

SEED = 2021

NUM_FEATS = 20 # number of features that we use. there are 100 feats but we don't need to use all of th
em

base_path = '/kaggle'
```

In [3]:
```
def set_seed(seed=42):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)
    session_conf = tf.compat.v1.ConfigProto(
        intra_op_parallelism_threads=1,
        inter_op_parallelism_threads=1
    )
    sess = tf.compat.v1.Session(graph=tf.compat.v1.get_default_graph(), config=session_conf)
    tf.compat.v1.keras.backend.set_session(sess)

def comp_metric(xhat, yhat, fhat, x, y, f):
    intermediate = np.sqrt(np.power(xhat-x, 2) + np.power(yhat-y, 2)) + 15 * np.abs(fhat-f)
    return intermediate.sum()/xhat.shape[0]
```

In [4]:
```
feature_dir = f"{base_path}/input/indoorunifiedwifids"
train_files = sorted(glob.glob(os.path.join(feature_dir, '*_train.csv')))
test_files = sorted(glob.glob(os.path.join(feature_dir, '*_test.csv')))
subm = pd.read_csv(f'{base_path}/input/indoor-location-navigation/sample_submission.csv', index_col=0)
```

In [5]:
```
with open(f'{feature_dir}/train_all.pkl', 'rb') as f:
  data = pickle.load( f)

with open(f'{feature_dir}/test_all.pkl', 'rb') as f:
  test_data = pickle.load(f)
```

In [6]:
```
# training target features

BSSID_FEATS = [f'bssid_{i}' for i in range(NUM_FEATS)]
RSSI_FEATS  = [f'rssi_{i}' for i in range(NUM_FEATS)]
```

In [7]:
```
# get numbers of bssids to embed them in a layer

wifi_bssids = []
for i in range(100):
    wifi_bssids.extend(data.iloc[:,i].values.tolist())
wifi_bssids = list(set(wifi_bssids))

wifi_bssids_size = len(wifi_bssids)
print(f'BSSID TYPES: {wifi_bssids_size}')

wifi_bssids_test = []
for i in range(100):
    wifi_bssids_test.extend(test_data.iloc[:,i].values.tolist())
wifi_bssids_test = list(set(wifi_bssids_test))

wifi_bssids_size = len(wifi_bssids_test)
print(f'BSSID TYPES: {wifi_bssids_size}')

wifi_bssids.extend(wifi_bssids_test)
wifi_bssids_size = len(wifi_bssids)
```

```
BSSID TYPES: 61206
BSSID TYPES: 33042
```

In [8]:
```
# preprocess

le = LabelEncoder()
le.fit(wifi_bssids)
le_site = LabelEncoder()
le_site.fit(data['site_id'])

ss = StandardScaler()
ss.fit(data.loc[:,RSSI_FEATS])
```

Out[8]:
```
StandardScaler()
```

In [9]:
```
data.loc[:,RSSI_FEATS] = ss.transform(data.loc[:,RSSI_FEATS])
for i in BSSID_FEATS:
    data.loc[:,i] = le.transform(data.loc[:,i])
    data.loc[:,i] = data.loc[:,i] + 1

data.loc[:, 'site_id'] = le_site.transform(data.loc[:, 'site_id'])

data.loc[:,RSSI_FEATS] = ss.transform(data.loc[:,RSSI_FEATS])
```

In [10]:
```
test_data.loc[:,RSSI_FEATS] = ss.transform(test_data.loc[:,RSSI_FEATS])
for i in BSSID_FEATS:
    test_data.loc[:,i] = le.transform(test_data.loc[:,i])
    test_data.loc[:,i] = test_data.loc[:,i] + 1

test_data.loc[:, 'site_id'] = le_site.transform(test_data.loc[:, 'site_id'])

test_data.loc[:,RSSI_FEATS] = ss.transform(test_data.loc[:,RSSI_FEATS])
```

In [11]:
```
site_count = len(data['site_id'].unique())
data.reset_index(drop=True, inplace=True)
```

In [12]:
```
set_seed(SEED)
```

The model

The first Embedding layer is very important.
Thanks to the layer, we can make sense of these BSSID features.

We concatenate all the features and put them into LSTM.

If something is theoritically wrong, please correct me. Thank you in advance.

In [13]:
```
def create_model(input_data):

    # bssid feats
    input_dim = input_data[0].shape[1]

    input_embd_layer = L.Input(shape=(input_dim,))
    x1 = L.Embedding(wifi_bssids_size, 64)(input_embd_layer)
    x1 = L.Flatten()(x1)

    # rssi feats
    input_dim = input_data[1].shape[1]

    input_layer = L.Input(input_dim, )
    x2 = L.BatchNormalization()(input_layer)
    x2 = L.Dense(NUM_FEATS * 64, activation='relu')(x2)

    # site
    input_site_layer = L.Input(shape=(1,))
    x3 = L.Embedding(site_count, 2)(input_site_layer)
    x3 = L.Flatten()(x3)


    # main stream
    x = L.Concatenate(axis=1)([x1, x3, x2])

    x = L.BatchNormalization()(x)
    x = L.Dropout(0.3)(x)
    x = L.Dense(256, activation='relu')(x)

    x = L.Reshape((1, -1))(x)
    x = L.BatchNormalization()(x)
    x = L.LSTM(128, dropout=0.3, recurrent_dropout=0.3, return_sequences=True, activation='relu')(x)
    x = L.LSTM(16, dropout=0.1, return_sequences=False, activation='relu')(x)


    output_layer_1 = L.Dense(2, name='xy')(x)
    output_layer_2 = L.Dense(1, activation='softmax', name='floor')(x)

    model = M.Model([input_embd_layer, input_layer, input_site_layer],
                    [output_layer_1, output_layer_2])

    model.compile(optimizer=tf.optimizers.Adam(lr=0.001),
                  loss='mse', metrics=['mse'])

    return model
```

In [14]:
```
score_df = pd.DataFrame()
oof = list()
predictions = list()

oof_x, oof_y, oof_f = np.zeros(data.shape[0]), np.zeros(data.shape[0]), np.zeros(data.shape[0])
preds_x, preds_y = 0, 0
preds_f_arr = np.zeros((test_data.shape[0], N_SPLITS))

for fold, (trn_idx, val_idx) in enumerate(StratifiedKFold(n_splits=N_SPLITS, shuffle=True, random_state
=SEED).split(data.loc[:, 'path'], data.loc[:, 'path'])):
    X_train = data.loc[trn_idx, BSSID_FEATS + RSSI_FEATS + ['site_id']]
    y_trainx = data.loc[trn_idx, 'x']
    y_trainy = data.loc[trn_idx, 'y']
    y_trainf = data.loc[trn_idx, 'floor']

    tmp = pd.concat([y_trainx, y_trainy], axis=1)
    y_train = [tmp, y_trainf]

    X_valid = data.loc[val_idx, BSSID_FEATS + RSSI_FEATS + ['site_id']]
    y_validx = data.loc[val_idx, 'x']
    y_validy = data.loc[val_idx, 'y']
    y_validf = data.loc[val_idx, 'floor']

    tmp = pd.concat([y_validx, y_validy], axis=1)
    y_valid = [tmp, y_validf]

    model = create_model([X_train.loc[:,BSSID_FEATS], X_train.loc[:,RSSI_FEATS], X_train.loc[:,'site_i
d']])
    model.fit([X_train.loc[:,BSSID_FEATS], X_train.loc[:,RSSI_FEATS], X_train.loc[:,'site_id']], y_trai
n,
                validation_data=([X_valid.loc[:,BSSID_FEATS], X_valid.loc[:,RSSI_FEATS], X_valid.loc[:,
'site_id']], y_valid),
                batch_size=128, epochs=1000,
                callbacks=[
                ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=3, verbose=1, min_delta=1e-4
, mode='min')
                , ModelCheckpoint(f'{base_path}/RNN_{SEED}_{fold}.hdf5', monitor='val_loss', verbose=0,
 save_best_only=True, save_weights_only=True, mode='min')
                , EarlyStopping(monitor='val_loss', min_delta=1e-4, patience=5, mode='min', baseline=No
ne, restore_best_weights=True)
            ])

    model.load_weights(f'{base_path}/RNN_{SEED}_{fold}.hdf5')
    val_pred = model.predict([X_valid.loc[:,BSSID_FEATS], X_valid.loc[:,RSSI_FEATS], X_valid.loc[:,'sit
e_id']])

    oof_x[val_idx] = val_pred[0][:,0]
    oof_y[val_idx] = val_pred[0][:,1]
    oof_f[val_idx] = val_pred[1][:,0].astype(int)

    pred = model.predict([test_data.loc[:,BSSID_FEATS], test_data.loc[:,RSSI_FEATS], test_data.loc[:,'s
ite_id']]) # test_data.iloc[:, :-1])
    preds_x += pred[0][:,0]
    preds_y += pred[0][:,1]
    preds_f_arr[:, fold] = pred[1][:,0].astype(int)

    score = comp_metric(oof_x[val_idx], oof_y[val_idx], oof_f[val_idx],
                        y_validx.to_numpy(), y_validy.to_numpy(), y_validf.to_numpy())
    print(f"fold {fold}: mean position error {score}")

    break # for demonstration, run just one fold as it takes much time.

preds_x /= (fold + 1)
preds_y /= (fold + 1)

print("*+"*40)
# as it breaks in the middle of cross-validation, the score is not accurate at all.
score = comp_metric(oof_x, oof_y, oof_f, data.iloc[:, -5].to_numpy(), data.iloc[:, -4].to_numpy(), data
.iloc[:, -3].to_numpy())
oof.append(score)
print(f"mean position error {score}")
print("*+"*40)

preds_f_mode = stats.mode(preds_f_arr, axis=1)
preds_f = preds_f_mode[0].astype(int).reshape(-1)
test_preds = pd.DataFrame(np.stack((preds_f, preds_x, preds_y))).T
test_preds.columns = subm.columns
test_preds.index = test_data["site_path_timestamp"]
test_preds["floor"] = test_preds["floor"].astype(int)
predictions.append(test_preds)
```

```
/opt/conda/lib/python3.7/site-packages/sklearn/model_selection/_split.py:668: UserWarning: The least
populated class in y has only 1 members, which is less than n_splits=10.
  % (min_groups, self.n_splits)), UserWarning)
```

```
Epoch 1/1000
1815/1815 [==============================] - 136s 72ms/step - loss: 2729.8407 - xy_loss: 2726.3058 -
floor_loss: 3.5362 - xy_mse: 2726.3058 - floor_mse: 3.5362 - val_loss: 85.3923 - val_xy_loss: 81.8580
- val_floor_loss: 3.5344 - val_xy_mse: 81.8580 - val_floor_mse: 3.5344
Epoch 2/1000
1815/1815 [==============================] - 127s 70ms/step - loss: 223.9748 - xy_loss: 220.4333 - fl
oor_loss: 3.5415 - xy_mse: 220.4333 - floor_mse: 3.5415 - val_loss: 70.9965 - val_xy_loss: 67.4622 -
val_floor_loss: 3.5344 - val_xy_mse: 67.4622 - val_floor_mse: 3.5344
Epoch 3/1000
1815/1815 [==============================] - 128s 71ms/step - loss: 186.3053 - xy_loss: 182.7956 - fl
oor_loss: 3.5097 - xy_mse: 182.7956 - floor_mse: 3.5097 - val_loss: 66.5667 - val_xy_loss: 63.0323 -
val_floor_loss: 3.5344 - val_xy_mse: 63.0323 - val_floor_mse: 3.5344
Epoch 4/1000
1815/1815 [==============================] - 128s 70ms/step - loss: 156.6663 - xy_loss: 153.1252 - fl
oor_loss: 3.5411 - xy_mse: 153.1252 - floor_mse: 3.5411 - val_loss: 54.7970 - val_xy_loss: 51.2626 -
val_floor_loss: 3.5344 - val_xy_mse: 51.2626 - val_floor_mse: 3.5344
Epoch 5/1000
1815/1815 [==============================] - 130s 72ms/step - loss: 134.9542 - xy_loss: 131.4136 - fl
oor_loss: 3.5406 - xy_mse: 131.4136 - floor_mse: 3.5406 - val_loss: 49.2016 - val_xy_loss: 45.6672 -
val_floor_loss: 3.5344 - val_xy_mse: 45.6672 - val_floor_mse: 3.5344
Epoch 6/1000
1815/1815 [==============================] - 129s 71ms/step - loss: 121.0858 - xy_loss: 117.5678 - fl
oor_loss: 3.5180 - xy_mse: 117.5678 - floor_mse: 3.5180 - val_loss: 45.9466 - val_xy_loss: 42.4122 -
val_floor_loss: 3.5344 - val_xy_mse: 42.4122 - val_floor_mse: 3.5344
Epoch 7/1000
1815/1815 [==============================] - 127s 70ms/step - loss: 110.7925 - xy_loss: 107.2740 - fl
oor_loss: 3.5185 - xy_mse: 107.2740 - floor_mse: 3.5185 - val_loss: 42.9951 - val_xy_loss: 39.4608 -
val_floor_loss: 3.5344 - val_xy_mse: 39.4608 - val_floor_mse: 3.5344
Epoch 8/1000
1815/1815 [==============================] - 129s 71ms/step - loss: 99.2850 - xy_loss: 95.7436 - floo
r_loss: 3.5415 - xy_mse: 95.7436 - floor_mse: 3.5415 - val_loss: 40.7045 - val_xy_loss: 37.1701 - val
_floor_loss: 3.5344 - val_xy_mse: 37.1701 - val_floor_mse: 3.5344
Epoch 9/1000
1815/1815 [==============================] - 128s 71ms/step - loss: 91.3670 - xy_loss: 87.8340 - floo
r_loss: 3.5330 - xy_mse: 87.8340 - floor_mse: 3.5330 - val_loss: 39.7930 - val_xy_loss: 36.2586 - val
_floor_loss: 3.5344 - val_xy_mse: 36.2586 - val_floor_mse: 3.5344
Epoch 10/1000
1815/1815 [==============================] - 127s 70ms/step - loss: 82.6977 - xy_loss: 79.1865 - floo
r_loss: 3.5112 - xy_mse: 79.1865 - floor_mse: 3.5112 - val_loss: 38.7777 - val_xy_loss: 35.2434 - val
_floor_loss: 3.5344 - val_xy_mse: 35.2434 - val_floor_mse: 3.5344
Epoch 11/1000
1815/1815 [==============================] - 128s 71ms/step - loss: 75.7084 - xy_loss: 72.1843 - floo
r_loss: 3.5241 - xy_mse: 72.1843 - floor_mse: 3.5241 - val_loss: 41.6324 - val_xy_loss: 38.0981 - val
_floor_loss: 3.5344 - val_xy_mse: 38.0981 - val_floor_mse: 3.5344
Epoch 12/1000
1815/1815 [==============================] - 130s 72ms/step - loss: 69.9684 - xy_loss: 66.4466 - floo
r_loss: 3.5218 - xy_mse: 66.4466 - floor_mse: 3.5218 - val_loss: 35.3302 - val_xy_loss: 31.7958 - val
_floor_loss: 3.5344 - val_xy_mse: 31.7958 - val_floor_mse: 3.5344
Epoch 13/1000
1815/1815 [==============================] - 129s 71ms/step - loss: 64.4142 - xy_loss: 60.8843 - floo
r_loss: 3.5299 - xy_mse: 60.8843 - floor_mse: 3.5299 - val_loss: 34.4395 - val_xy_loss: 30.9051 - val
_floor_loss: 3.5344 - val_xy_mse: 30.9051 - val_floor_mse: 3.5344
Epoch 14/1000
1815/1815 [==============================] - 129s 71ms/step - loss: 60.9259 - xy_loss: 57.3923 - floo
r_loss: 3.5336 - xy_mse: 57.3923 - floor_mse: 3.5336 - val_loss: 36.2234 - val_xy_loss: 32.6890 - val
_floor_loss: 3.5344 - val_xy_mse: 32.6890 - val_floor_mse: 3.5344
Epoch 15/1000
1815/1815 [==============================] - 131s 72ms/step - loss: 56.7305 - xy_loss: 53.2067 - floo
r_loss: 3.5237 - xy_mse: 53.2067 - floor_mse: 3.5237 - val_loss: 32.9840 - val_xy_loss: 29.4496 - val
_floor_loss: 3.5344 - val_xy_mse: 29.4496 - val_floor_mse: 3.5344
Epoch 16/1000
1815/1815 [==============================] - 131s 72ms/step - loss: 53.5774 - xy_loss: 50.0561 - floo
r_loss: 3.5212 - xy_mse: 50.0561 - floor_mse: 3.5212 - val_loss: 31.3678 - val_xy_loss: 27.8334 - val
_floor_loss: 3.5344 - val_xy_mse: 27.8334 - val_floor_mse: 3.5344
Epoch 17/1000
1815/1815 [==============================] - 130s 71ms/step - loss: 51.0886 - xy_loss: 47.5495 - floo
r_loss: 3.5391 - xy_mse: 47.5495 - floor_mse: 3.5391 - val_loss: 32.0633 - val_xy_loss: 28.5289 - val
_floor_loss: 3.5344 - val_xy_mse: 28.5289 - val_floor_mse: 3.5344
Epoch 18/1000
1815/1815 [==============================] - 128s 70ms/step - loss: 48.7772 - xy_loss: 45.2407 - floo
r_loss: 3.5365 - xy_mse: 45.2407 - floor_mse: 3.5365 - val_loss: 31.7013 - val_xy_loss: 28.1670 - val
_floor_loss: 3.5344 - val_xy_mse: 28.1670 - val_floor_mse: 3.5344
Epoch 19/1000
1815/1815 [==============================] - 131s 72ms/step - loss: 46.0759 - xy_loss: 42.5587 - floo
r_loss: 3.5172 - xy_mse: 42.5587 - floor_mse: 3.5172 - val_loss: 30.5862 - val_xy_loss: 27.0518 - val
_floor_loss: 3.5344 - val_xy_mse: 27.0518 - val_floor_mse: 3.5344
Epoch 20/1000
1815/1815 [==============================] - 127s 70ms/step - loss: 44.5626 - xy_loss: 41.0293 - floo
r_loss: 3.5333 - xy_mse: 41.0293 - floor_mse: 3.5333 - val_loss: 31.5709 - val_xy_loss: 28.0365 - val
_floor_loss: 3.5344 - val_xy_mse: 28.0365 - val_floor_mse: 3.5344
Epoch 21/1000
1815/1815 [==============================] - 142s 78ms/step - loss: 42.3605 - xy_loss: 38.8158 - floo
r_loss: 3.5447 - xy_mse: 38.8158 - floor_mse: 3.5447 - val_loss: 31.3791 - val_xy_loss: 27.8448 - val
_floor_loss: 3.5344 - val_xy_mse: 27.8448 - val_floor_mse: 3.5344
Epoch 22/1000
1815/1815 [==============================] - 133s 73ms/step - loss: 41.1424 - xy_loss: 37.5991 - floo
r_loss: 3.5433 - xy_mse: 37.5991 - floor_mse: 3.5433 - val_loss: 28.4463 - val_xy_loss: 24.9119 - val
_floor_loss: 3.5344 - val_xy_mse: 24.9119 - val_floor_mse: 3.5344
Epoch 23/1000
1815/1815 [==============================] - 133s 74ms/step - loss: 39.5329 - xy_loss: 36.0124 - floo
r_loss: 3.5205 - xy_mse: 36.0124 - floor_mse: 3.5205 - val_loss: 29.8392 - val_xy_loss: 26.3048 - val
_floor_loss: 3.5344 - val_xy_mse: 26.3048 - val_floor_mse: 3.5344
Epoch 24/1000
1815/1815 [==============================] - 137s 75ms/step - loss: 37.9622 - xy_loss: 34.4312 - floo
r_loss: 3.5310 - xy_mse: 34.4312 - floor_mse: 3.5310 - val_loss: 28.7268 - val_xy_loss: 25.1924 - val
_floor_loss: 3.5344 - val_xy_mse: 25.1924 - val_floor_mse: 3.5344
Epoch 25/1000
1815/1815 [==============================] - 146s 80ms/step - loss: 37.0578 - xy_loss: 33.5341 - floo
r_loss: 3.5237 - xy_mse: 33.5341 - floor_mse: 3.5237 - val_loss: 30.7665 - val_xy_loss: 27.2321 - val
_floor_loss: 3.5344 - val_xy_mse: 27.2321 - val_floor_mse: 3.5344

Epoch 00025: ReduceLROnPlateau reducing learning rate to 0.00010000000474974513.
Epoch 26/1000
1815/1815 [==============================] - 138s 76ms/step - loss: 33.8569 - xy_loss: 30.3180 - floo
r_loss: 3.5389 - xy_mse: 30.3180 - floor_mse: 3.5389 - val_loss: 36.9109 - val_xy_loss: 33.3765 - val
_floor_loss: 3.5344 - val_xy_mse: 33.3765 - val_floor_mse: 3.5344
Epoch 27/1000
1815/1815 [==============================] - 134s 74ms/step - loss: 32.5913 - xy_loss: 29.0521 - floo
r_loss: 3.5392 - xy_mse: 29.0521 - floor_mse: 3.5392 - val_loss: 45.9923 - val_xy_loss: 42.4579 - val
_floor_loss: 3.5344 - val_xy_mse: 42.4579 - val_floor_mse: 3.5344
fold 0: mean position error 27.549837684159932
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+
mean position error 175.7266515006742
*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+*+
```

In [15]:
```
all_preds = pd.concat(predictions)
all_preds = all_preds.reindex(subm.index)
```

Fix the floor prediction

So far, it is not successfully make the "floor" prediction part with this dataset.
To make it right, we can incorporate @nigelhenry's excellent work.

In [16]:
```
simple_accurate_99 = pd.read_csv('../input/simple-99-accurate-floor-model/submission.csv')

all_preds['floor'] = simple_accurate_99['floor'].values
```

In [17]:
```
all_preds.to_csv('submission.csv')
```

That's it.

Thank you for reading all of it.

I hope it helps!

Please make comments if you found something to point out, insights or suggestions.