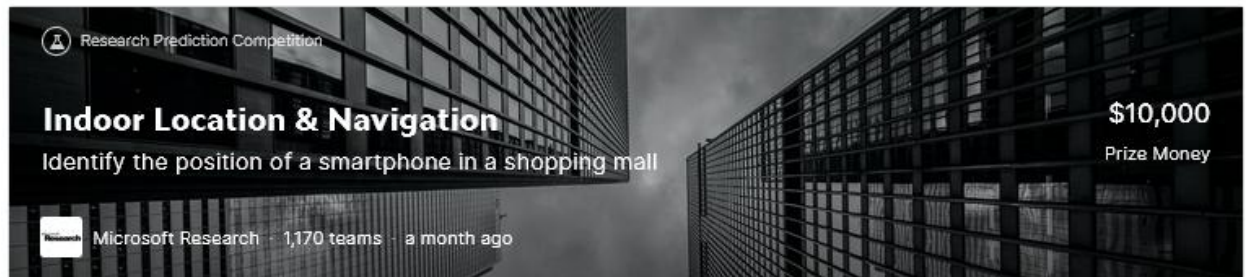


Indoor Location & Navigation

(Identify the position of a smartphone in a shopping mall)

Artur Makhanko, Advanced Data Mining course project



<https://www.kaggle.com/c/indoor-location-navigation/>

Current positioning indoor solutions have relatively poor accuracy, particularly in multi-level buildings, or generalize poorly to small datasets. Additionally, GPS was built for a time before smartphones. Today's use cases often require more granularity than is typically available indoors.

In this competition, our task is to predict the indoor position of smartphones based on real-time sensor data, provided by indoor positioning technology company XYZ10 in partnership with Microsoft Research. We'll locate devices using "active" localization data, which is made available with the cooperation of the user. We'll work with a dataset of nearly 30,000 traces from over 200 buildings.

Methods we are going to use:

0. Feature selection

1. Gradient boosting trees (LightGBM)

2. LSTM (Keras)

3. Postprocessing (Cost minimization, snap to grid)

Let us look at the available data first.

Train directory contains sites (shopping malls) directories that consist of floor directories. And each floor directory contains txt files, that have information about paths (smartphones). **Metadata** directory contains floor map, its size and geo information for each site and each floor.

The dataset for this competition consists of dense indoor signatures of WiFi, geomagnetic field, iBeacons etc., as well as ground truth (waypoint) (locations) collected from hundreds of

buildings in Chinese cities. The data found in path trace files (*.txt) corresponds to an indoor path between position p_1 and p_2 walked by a site-surveyor.

During the walk, an Android smartphone is held flat in front of the surveyors body, and a sensor data recording app is running on the device to collect IMU (accelerometer, gyroscope) and geomagnetic field (magnetometer) readings, as well as WiFi and Bluetooth iBeacon scanning results. In addition to raw trace files, floor plan metadata (e.g., raster image, size, GeoJSON) are also included for each floor.

- train - training path files, organized by site and floor; each path files contains the data of a single path on a single floor
- test - test path files, organized by site and floor; each path files contains the data of a single path on a single floor, *but without the waypoint (x, y) data*; the task of this competition is, for a given site-path file, predict the floor and waypoint locations at the timestamps given in the `sample_submission.csv` file
- metadata - floor metadata folder, organized by site and floor, which includes the following for each floor:
 - `floor_image.png`
 - `floor_info.json`
 - `geojson_map.json`
- `sample_submission.csv` - a sample submission file in the correct format; each has a unique `id` which contains a site id, a path id, and the timestamp within the trace for which to make a prediction; see the [Evaluation page](#) for the required integer mapping of floor names

Example of the floor with ground truth plotted over it:



Example of the data we are expected to train on:

Data Explorer
55.84 GB

metadata
test
00ff0c9a71cc37a2...
01c41f1aeba5c48c...
030b3d94de8aca...
0389421238a7e2...
04029880763600...
0412d582bb8a2c...
046cfa46be49fc1...
049bb468e7e166...
04b259d70f2b50...
053526f9012ca71...
055255f16b549ed...
05a6d4cdf3d1eb9...
05d052dde78384...
06832e9b9ca24ff...
06882da3694b71...
068e4f6926e78ff...
07db4eac1fc8df50...

< 00ff0c9a71cc37a2ebdd0f05.txt (2.79 MB) [Download] [Full Screen]

This preview is truncated due to the large file size. Create a Notebook or download this file to see the full content. [Download]

```
# startTime 000000000000
# SiteID:5da1389e4db8ce0c98bd0547 SiteName:和达城商场
# Brand:OPPO Model:PBCM10 AndroidName:8.1.0 APILevel:27
# type:1 name:BMI160 Accelerometer version:2062600 vendor:BOSCH resolution:0.
# type:4 name:BMI160 Gyroscope version:2062600 vendor:BOSCH resolution:0.00106811
# type:2 name:AK09911 Magnetometer version:1 vendor:AKM resolution:0.
# type:35 name:BMI160 Accelerometer Uncalibrated version:2062600 vendor:BOSCH resol
# type:16 name:BMI160 Gyroscope Uncalibrated version:2062600 vendor:BOSCH resol
# type:14 name:AK09911 Magnetometer Uncalibrated version:1 vendor:AKM resol
# VersionName:v20191105-nightly-3-g9486d0e VersionCode:390
000000000143 TYPE_ACCELEROMETER -0.32388306 2.2018127 8.9592285 2
000000000143 TYPE_MAGNETIC_FIELD 37.600708 1.9195557 -21.879578 3
000000000143 TYPE_GYROSCOPE 0.030273438 -0.024505615 -0.23849487 3
000000000143 TYPE_ROTATION_VECTOR 0.06012629 0.063835315 0.6487433 3
000000000143 TYPE_MAGNETIC_FIELD_UNCALIBRATED 0.0 -28.941345 -363.4308
000000000143 TYPE_GYROSCOPE_UNCALIBRATED 0.019836426 0.03567505 -0.2297821
000000000143 TYPE_ACCELEROMETER_UNCALIBRATED -0.15509033 2.0767212 9.088516
```

0. Feature selection

For the first two phases we are going to use only the wifi/beacon-features. The rest of the features are better to use in the post-processing phase, while wifi/beacon-features create a good baseline.

Basically we are going to use only wifi/beacon, timestamp and waypoint data to generate our solution.

First let us preprocess the initial wifi/beacon data, so that we can pass it into gradient boosting tree. Code with the description of the dataset creating process is given under the next link:

As an input we will use the initial training data, which is available in the description of the competition.

We will use such a dataset (or, to be more exact, datasets) to train our LGBM model.

The idea is following:

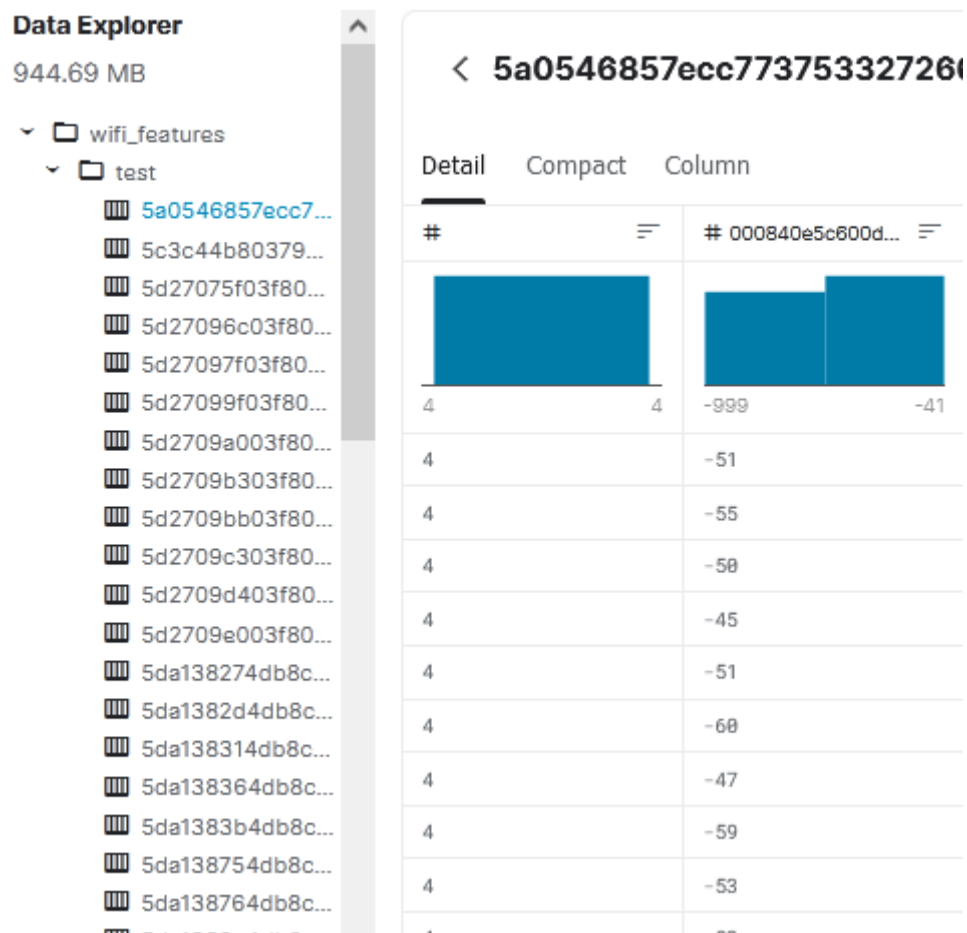
The RSSI (for both the beacon and the wifi) values are a measure of signal strength and hence likely correlated with distance from the phone to wifi/beacon access point. Because there are many different signals that the phone picks up there should be a identifiable signature of signal strengths that can be used to estimate the phone position.

In the training set the position of a block of wifi/beacon signals can be estimated by finding the temporally nearest waypoint. This could be used as a label for training data for use with

regressors/classifiers. The phone acceleration etc could be used to improve upon the labels, but this seems like it would be a lot of work and I would ignore it for an initial solution.

For the test set we would find the temporally nearest wifi/beacon block, to each of the prediction time points, and use the classifiers/regressor to estimate its position and floor. Again this could be improved by then applying the phones acceleration etc data to estimate the position offset between when the wifi/beacon signals were recorded and the target timestamp.

Example of the data from the dataset:



Code for processing the data:

https://github.com/282075uwr/ADM_P/blob/main/wifi_features.ipynb

Still, we are going to preprocess the dataset even more in order to feed it into the LSTM.

Another thing we may do is to create one unified dataset, which is making one dataframe to feed into one model. As an input we are going to use the previously generated dataset. It will be constructed the next way:

bssid_* features:

These features are WiFi BSSID ordered by RSSI strength.

For example, bssid0 has the strongest RSSI for the x, y, floor. bssid1 is the second.

rss*_ features:

For the response to bssid_*, this is the rssi value for the bssid.

filename:

Filename represents the siteid. We can use this siteid for training if we want to.

Example of the data:

< **5a0546857ecc773753327266_test.csv** (1.29 MB)

Detail Compact Column

A bssid_0	A bssid_1	A bssid_2	A bssid_3
365544221bea97f5... 3% 67b0fd33f6477b11... 2% Other (284) 95%	b2337b25e7d1df04... 2% 345c70f0084ce5d5... 2% Other (289) 97%	03431cb0601b5e99... 2% b2546cae6e588d3... 2% Other (288) 96%	d73000724c487cf2... 2% b2546cae6e588d38... 1% Other (290) 97%
eebf5db207eec2f3e041 f92153d789270f346821	323607d8444900d64151 ee06d164738ac727bbce	7805f319f3f591986eff e78c5b41143180278f2d	02a1be3a5dab38320f87 9489d8a1e0f2a72768b3
1d1d62dcf72481cc9580 fed3b724f0d27015aaf1	bd9bc0a2092c040bfe6b a12f8aafac24e83b312a	d771612396c3e2e557e9 86fafd9fc2c56a99d3cd	13b7aeaf441f21614814 81fe67eace721cff07ab
6bc91b3951089c3a2253 96608b138ca178479924	b26914599f6d9ba16b43 975394e1eeb9d82f4bab	b2546cae6e588d38618e acc557dd0385812197cf	1d1d62dcf72481cc9580 fed3b724f0d27015aaf1
de53ffe7e3c71c9ed5c8 45fa50e0521efa5f3685	1d1d62dcf72481cc9580 fed3b724f0d27015aaf1	bccd6a9054f8649ad43f e96b766687fb769b064f	f64c13fd10a07bca1bf2 b7bd7a80630632ce62c9
1d1d62dcf72481cc9580 fed3b724f0d27015aaf1	a929157f3cc32a433b02 ad7d7876e9a1678d3944	6bc91b3951089c3a2253 96608b138ca178479924	000840e5c600de293cea 57f13326f273c86c3988
7805f319f3f591986eff e78c5b41143180278f2d	12911a64fecf13f2e9fb 0aaed554621e3b0bacde	b26914599f6d9ba16b43 975394e1eeb9d82f4bab	b2546cae6e588d38618e acc557dd0385812197cf
7805f319f3f591986eff e78c5b41143180278f2d	d84cce12fbfba61bf930 123050f61a11e2a29310	b2546cae6e588d38618e acc557dd0385812197cf	b26914599f6d9ba16b43 975394e1eeb9d82f4bab

Code for such preprocessing is available under the next link:

https://github.com/282075uwr/ADM_P/blob/main/unified_wifi.ipynb

Now let us train an ensemble of LGBMs (one model for each floor) and a LSTM trained on the unified features.

1. LGBM Training

We use the next configuration (found with optuna and manual search).

```
lgb_params_1 = {'objective': 'root_mean_squared_error',
                'boosting_type': 'gbdt',
                'n_estimators': 50000,
                'learning_rate': 0.01,
                'num_leaves': 90,
                'colsample_bytree': 0.4,
                'subsample': 0.6,
                'subsample_freq': 2,
                'bagging_seed': SEED,
                'reg_alpha': 8,
                'reg_lambda': 2,
                'random_state': SEED,
                'n_jobs': -1
                }
```

Example of LGBM training process (for one building) can be found under the next link:

https://github.com/282075uwr/ADM_P/blob/main/LGBM.ipynb

2. LSTM Training

Model:

```
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='swish')(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(0.2)(x)
x = L.Dense(256, activation='swish')(x)

x = L.Reshape((1, -1))(x)
x = L.BatchNormalization()(x)
x = L.Dropout(0.2)(x)
#x = L.LSTM(128, dropout=0.03, recurrent_dropout=0.03, return_sequences=True, activation='swish')(x)
x = L.LSTM(64, dropout=0.2, return_sequences=False, activation='swish')(x)


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

Example of LSTM training process can be found under the next link:


https://github.com/282075uwr/ADM_P/blob/main/LSTM.ipynb

3. Postprocessing (Cost minimization, snap to grid)

The results we have obtained are bearable, but sometimes they provide not really desired results (e.g. phone being teleported inside of the wall, etc.) Maybe it is not that intuitive, but, taking into consideration the discussion of the winner's solutions, almost entire top 20% of the participants (including the second place) didn't go further than LSTM or LGBM in the baseline phase. The most important phase of the competition has happened with the postprocessing of the predictions, not the predictions itself. My combination for postprocessing has allowed me to enter the top 15%, but I was too lazy to somehow improve on it, taking into consideration the coming projects deadlines for the university.



Indoor Location & Navigation
 Identify the position of a smartphone in a shopping mall
 Research · a month ago

Top 14%
 161/1170
 ...

Anyway, let us discuss the postprocessing.

1. Merge the results.

For "Ensembling", we used two notebooks with best scores. And for the "Comparative Method" we got help from the remaining eleven notebooks (we generated them by changing the LSTM or LGBM configs a bit). If you read the code carefully, you will see that we have only allowed some of the results of these eleven notebooks to affect the overall result. That is, they can only be effective if their X and Y values are between the X and Y of the two main notebooks. This is a very important point.

The code is available under the next link:

https://github.com/282075uwr/ADM_P/blob/main/comparative.ipynb

2. Floor model

- Construct a floor-bssid-max(RSSI) frequency table for each site
- Give each floor 1 point for every bssid-RSSI pair that falls within the max(RSSI) previously observed on that floor
- Take the highest-voted floor.

The code is so basic that there is really no need for showing it. A lot of loops and table imports, nothing special.

3. Cost minimalization:

We are going to use the location prediction shared by competition hosts based on the magn/acc data.

Let us define the next cost function:

$$L(X_{1:N}) = \sum_{t=1}^N \alpha_t \|X_t - \hat{X}_t\|^2 + \sum_{t=1}^{N-1} \beta_t \|(X_{t+1} - X_t) - \Delta \hat{X}_t\|^2$$

And, again, quoting the original cost minimalization idea from their webinar

where \hat{X}_t is absolute position predicted by machine learning and $\Delta \hat{X}_t$ is relative position predicted by sensor data.

Since the cost function is quadratic, the optimal X is solved by linear equation $QX = c$, where Q and c are derived from above cost function. Because the matrix Q is tridiagonal, each machine learning prediction is corrected by *all* machine learning predictions and sensor data.

The optimal hyperparameters (α and β) can be estimated by expected error of machine learning and sensor data, or just tuned by public score.

The code is available under the next link:

https://github.com/282075uwr/ADM_P/blob/main/cost_min.ipynb

4. Snap to Grid:

In order not to repeat ourselves we will use the modified version of this step in the summary. To describe the main idea, we assume that the space where we make predictions isn't continuous, but discrete. Thus we generate some points and if the length (defined by some function) between a generated waypoint and our prediction is shorter than some threshold – we take the position of the waypoint and replace the position of the prediction with a newly obtained data.

Where one can **go further**? We can modify it in the next ways:

1. Use dynamic algorithms.

For example, 11th place solution differs from mine only by the snap to grid model part. And this is ~100 places between us.

2. Create new waypoints:

2.1 Creating the waypoint grid:

https://github.com/282075uwr/ADM_P/blob/main/cost_min.ipynb

2.2 Shrinking the created grid (push further from the walls):

https://github.com/282075uwr/ADM_P/blob/main/shrinking_the_corridors.ipynb

3. Utilizing more advanced cost optimization functions

4. Playing more with the initial dataset.

5. Looking for more data leakage (f.e. limiting oneself to the selected pathes by the leaked phone id)






Results:































Submissions are evaluated on the **mean position error** as defined as:

$$\text{mean position error} = \frac{1}{N} \sum_{i=1}^N \left(\sqrt{(\hat{x}_i - x_i)^2 + (\hat{y}_i - y_i)^2} + p \cdot |\hat{f}_i - f_i| \right)$$

where:

- N is the number of rows in the test set
- \hat{x}_i, \hat{y}_i are the predicted locations for a given test row
- x_i, y_i are the ground truth locations for a given test row
- p is the floor penalty, set at 15
- \hat{f}_i, f_i are the predicted and ground truth integer floor level for a given test row

159	▼ 33	Mike Kim		5.22453	10	1mo
160	▼ 12	Nemuri		5.23180	18	1mo
161	▼ 7	majoraregalia		5.24013	90	1mo
162	▲ 5	Marek Nurzynski		5.24463	82	1mo
163	▲ 6	Duong Anh Kiet		5.24697	15	1mo

1	—	Track me if you can	  	1.49717	322	1mo
2	—	MYRCJ	    	2.19892	513	1mo
3	▲ 2	Zidmie		2.48923	476	1mo
4	▲ 2	ELQMH	    	2.68029	497	1mo
5	▲ 2	chris		2.77024	134	1mo
6	▼ 2	Boyrin Vjacheslav	 	2.81548	109	1mo
7	▲ 1	Isamu & Matt & ryles & T...	    	2.82709	403	1mo
8	▲ 1	Michał Stolarczyk		3.07282	167	1mo
9	▲ 1	higepon & saito	 	3.10582	364	1mo
10	▲ 2	cuML		3.12258	163	1mo
11	▼ 8	Ouranos & Vicens	 	3.15829	478	1mo
12	▲ 2	Hugues		3.19888	154	1mo
13	▼ 2	Olaf Placha		3.35958	108	1mo