Fingerprinting indoor positioning based on Wi-Fi

Peng Wu, Ni Ke

August 24, 2018

Contents

1 Introduction

1.1 Indoor positioning overview

1.1.1 IOT, GNSS, IPS and its application

Indoor positioning [1] has been improved a lot and has recently witnessed an increasing market interest due to wide-scale use of Internet of Things (IOT) such as smartphones and other wireless devices in the last couple of years. Nowadays many applications include Global Navigation Satellite System (GNSS) receivers, are using GPS, GLONASS, Galileo, or BeiDou system service for global positioning. Affected by signal attenuation caused by construction materials, GNSS loses significant power in indoor environment. In addition, the multiple reflections at surfaces where indoor and outdoor connected will cause multi-path propagation leading to uncontrollable errors. So people come up with Indoor Positioning System (IPS) to precisely locate objects inside a building using lights, radio waves, magnetic fields, acoustic signals, or other sensory information collected by mobile devices.

1.1.2 location technology measuring principles and methods rss, Channel State Information (CSI), aoa, tof, tdoa, rtoa, fingerprinting, cooperative positioning

There are various signal metrics used for indoor location listed in the table below.

TABLE II
ADVANTAGES AND DISADVANTAGES OF DIFFERENT LOCALIZATION TECHNIQUES

Technique	Advantages	Disadvantages
RSSI	Easy to implement, cost efficient, can be used with a number of technologies	Prone to multipath fading and environmental noise, lower localization accuracy, can require fingerprinting
CSI	More robust to multipath and indoor noise,	It is not easily available on off-the-shelf NICs
AoA	Can provide high localization accuracy, does not require any fingerprinting	Might require directional antennas and complex hardware, requires comparatively complex algorithms and performance deteriorates with increase in distance between the transmitter and receiver
ToF	Provides high localization accuracy, does not require any fingerprinting	Requires time synchronization between the transmitters and receivers, might require time stamps and multiple antennas at the transmitter and receiver. Line of Sight is mandatory for accurate performance.
TDoA	Does not require any fingerprinting, does not require clock synchronization among the device and RN	Requires clock synchronization among the RNs, might require time stamps, requires larger bandwidth
RToF	Does not require any fingerprinting, can provide high localization accuracy	Requires clock synchronization, processing delay can affect performance in short ranger measurements
PoA	Can be used in conjunction with RSS, ToA, TDoA to improve the overall localization accuracy	Degraded performance in the absence of line of sight
Fingerprinting	Fairly easy to use	New fingerprints are required even when there is a minor variation in the space

Table 1. Advantages and disadvantages of different positioning techniques
link: https://arxiv.org/abs/1709.01015 a survey of indoor positioning

1.1.3 indoor positioning technologies: WLAN, RFID, Ultra-wideband, Bluetooth, inertial, Ultrasonic, hybrid technologies

Table 2. Comparison of IPS technologies

IPS	Technology	Accuracy	Cost (installation/ unit)	Advantages	Disadvantages	Complexity
RADAR	WiFi	2-3 m	L/L	Low price, existing infrastructure	Low accuracy, complex system	Medium
Ekahau	WiFi	1-3 m	H/L	Existing infrastructure, good mapping software	Expensive mapping software	Low
LANDMARC	RFID	2 m	H/L	Very cheap user units	Locating delay 7.5 s	Medium
Active Bat	Ultrasound	3 cm	H/L	Cheap user units, very precise	Requires a lot of beacons, medium battery life	High
Lok8	Ultrasound	10 cm	L/L	Smartphone user units, precise	Requires new infrastructure in every room	Medium
Topaz	Bluetooth	2 m	L/L	Low price	Locating delay 15-30 s	Medium
iBeacon	Bluetooth	0.5-3 m	H/L	Smartphone user units, ease of access	Requires a lot of beacons for better precision	Low
Beauregard et al.	Inertial	0.74-2.5 m	L/L	Cheap, map can be added post-hoc	Requires a detailed map for better precision	Medium
Ubisense	Ultra-wideband	15 cm	H/H	Very precise, very robust	Expensive installation and units	High
Leppäkoski et al.	Inertial +WiFi	3 m	L/L	Cheap, could work on smartphone	Low accuracy, time consuming installation	Medium
Cricket	Ultrasound + RF	10 cm	H/L	Cheap user units, provides privacy, precise	complex installation, low battery life	High
Han Zou et al.	Inertial +WiFi +Bluetooth	0.59 m	L/L	Cheap, precise	Requires detailed mapping	Medium

Link: https://ieeexplore.ieee.org/document/8400090/#_full-text-section overview of indoor positioning system technologies

1.2 Fingerprinting overview:link

We will use the fingerprinting [2] as our indoor position technique. It is one of the most popular schemes. It contains offline stage and online stage. In the offline stage, the system builds a database of thorough measurements from reference locations in target area. Then, in the online stage, the system will take the data to its model based on its database to predict the real-time location. Most of existing indoor fingerprinting systems exploit WiFi RSS values as fingerprints because of its simplicity and low hardware requirements. Some others use CSI data, which a litter sophisticated.

1.2.1 Advantages and disadvantages of Fingerprint Positioning based on WiFi

Compared with other indoor positioning systems, the WiFi Fingerprinting Positioning technology has advantages of low cost and high precision. Due to wide coverage of WiFi area worldwide, it's easy to detect the signal strength in any scenario where WiFi networks are deployed without additional hardware which makes the technology

cost low. In addition, the WiFi signal intensity is relatively stable compared to space-time attributes such as angle and arrival time which will cause more errors, so WiFi Fingerprinting Positioning technology can product preciser result.

WiFi Fingerprinting Positioning technology has disadvantage of high complexity. Since the input is RSS from various APs, the data storage cost and computation consumption can be relatively high than other traditional technologies. The method also requires a large amount of priori information as data support in order to improve the prediction accuracy, so its preliminary work costs high. In addition, the indoor environment is complex, the radio signals can be also affected by multipath effects, so there will be steady errors in the measurement.

1.2.2 Some popular algorithms for indoor positioning Fingerprinting studies: knn, svm, neural network

a) K Nearest Neighbor (KNN) is one of the simplest algorithms in machine learning, it has been widely used in indoor positioning for the low cost and high performance. KNN algorithm is comparing the received signal strength obtained by users with the signal strength value stored in fingerprint database, and picks k nearest data point from database according to certain metric. Finally it completes the positioning step by calculating the average of the k chosen fingerprinting data.

b) Support Vector Machine (SVM) is one of the most popular, most practical, and highest potential methods in statistical learning to analyze data used for classification and regression analysis, it turns the input space into a higher dimensional space by making nonlinear transform defined by inner product function, and calculates the optimal classification plane in this space.

c) As the principle of fingerprint data matching is similar to pattern recognition, a large number of artificial intelligence technologies used in pattern recognition can also be used in fingerprint positioning. Artificial neural network is one of the most important methods in machine learning to address this kind of issue. These artificial neural network methods includes multilayer perceptron, back propagation neural networks, multi-layer neural networks, and regression neural networks.

1.2.3 Our works compare to others:

1. Separate floor detection and position regression as two part;
2. Implement WKNN in matlab & DNN in jupyter notebook;
3. implement ensemble bagging method to get a 100% floor detection;
4. implement stack method to get better result.
5. Compare results of long-term data with the short-term data
6. Provide a CNN based approach for long-term data

2 Analysis

2.1 Data description

The two datasets are from the same research team, Tampere University of Technology[1] and Universitat Jaume I[2]. All the data are Wi-Fi database collected in a full crowdsourced mode (i.e., different devices, different users and no main indications).

In the first dataset [3], it mainly contain two files, one is coordinate file. Each row shows the (x,y,z) coordinate (in meters) where the measurements were done. These are the local coordinates, not the GPS coordinates (WGS84 or similar), so that they can be directly used for positioning studies.

Another is RSS file, This is a large file with NAP columns, showing the RSS level at which each of the NAP MAC addresses were heard in each measurement point. Each row corresponds to one measurement. The non-heard APs are set to +100 dBm, which is a fixed bogus value. If an MAC address is heard, then it is heard at a negative level (in dBm). For example, in the training RSS file, Access Point 2 was not heard (i.e., value of 100); Access Point 420 was heard at $-$ 84 dBm; and Access Point 489 was heard with $-$ 52 dBm.

The second dataset [4] was collected in university's library. For each of these reference points,which belong to two oors (3rd & 5th floors), 12 measurements were taken, six consecutive samples with the person who performed the collection facing one direction and another consecutive six facing the opposite direction.The measurements were organized into training and test datasets,and repeated each every month until completing 15 months of collection.

2.2 Visualization

Short term data, for the AP 492, visualize its distribution.

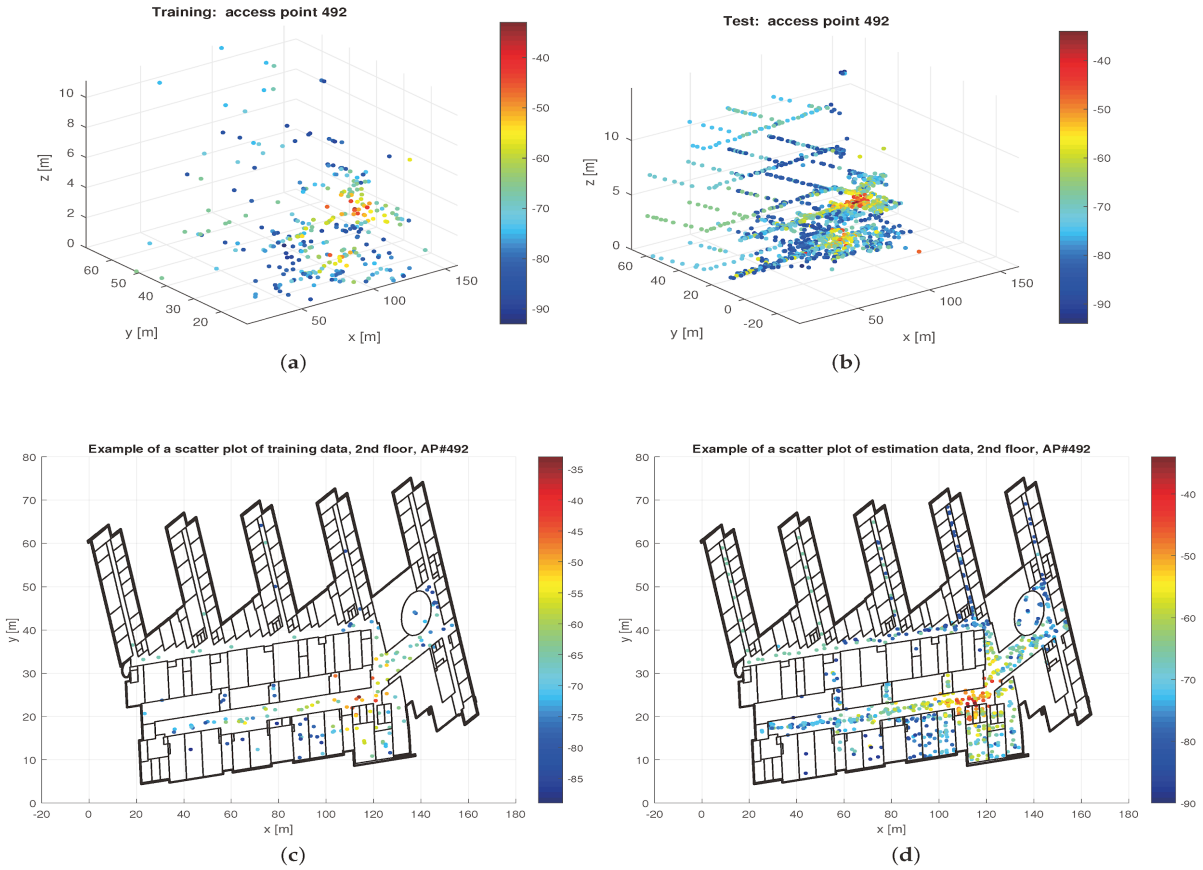


Figure 1. Examples of the 3D (up) and 2D (down) scatter diagrams (non-interpolated power maps) of AP 492. (a) 3D training data; (b) 3D test data; (c) 2D training data; (d) 2D test data.

Long term data, for AP 7, visualize dataset of each floor

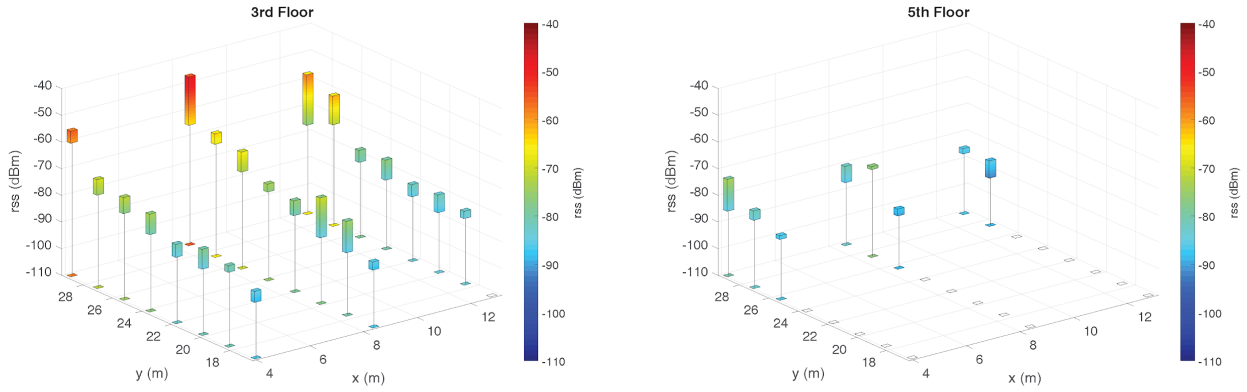
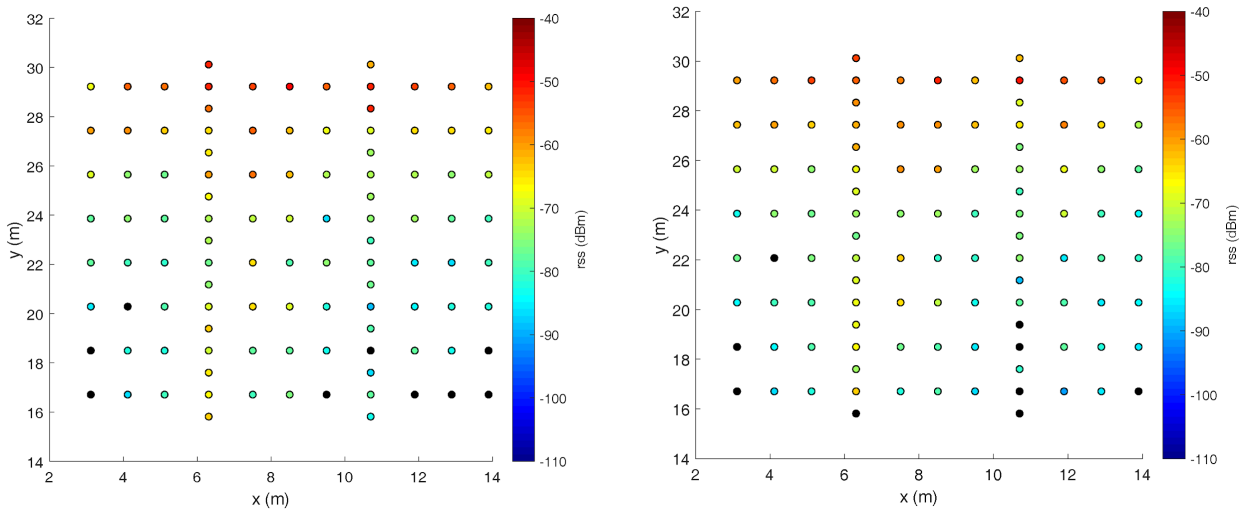


Figure 2. Mean and standard deviation values visualization for the 3rd (a) and 5th (b) floors. The vertical center of each bar represents the mean value, and the bar's size is adjusted to the standard deviation value.



(**a**) 1st Sample (**b**) 6th Sample

Figure 3. AP 7 RSS values for the first and last sample at collection, taking only samples from the 3rd floor and the "direct" direction. The black filling color indicates that the AP was not detected.

2.3 Benchmark

We have two metrics:

a) Floor detection rate is the percentage of the test points in which the floor was estimated correctly.

b) 2D mean error is the error between the estimated position and the true position when the estimate is at the same floor with the true position (incorrect floor estimates are ignored here).

Algorithm	Mean 2D Error (m)	Mean 3D Error (m)	Floor Detection (%)	Reference
Weighted centroid	10.64	11.57	83.19	[18]
Log-Gaussian probability ($\sigma = 10, N_{nn} = 3$)	10.18	11.19	82.92	[20,23]
Log-Gaussian probability ($\sigma = 7, N_{nn} = 1$)	9.78	11.03	85.29	[20,23]
RSS clustering (affinity propagation)	8.09	8.70	90.81	[20]
3D clustering (k-means)	17.35	24.73	72.90	[20]
UJI kNN algorithm (data=positive, dist=sorensen, $N_{nn} = 1, Not_{heard} = -103$)	8.45	8.73	92.26	[21]
UJI kNN algorithm (data=exponential, dist=neyman, $N_{nn} = 1, Not_{heard} = -103$)	8.60	9.02	91.98	[21]
UJI kNN algorithm (data=powed, dist=sorensen, $N_{nn} = 1, Not_{heard} = -103$)	8.65	8.92	92.99	[21]
RTLS@UM ($approach = 1, variant = 1, n = k1 = 5, k2 = 3$)	9.18	10.29	86.99	[31,32]
RTLS@UM ($approach = 1, variant = 3, n = 5, k1 = 1, k2 = 3$)	9.18	9.92	90.05	[31,32]
RBF ($N_{nn} = 1$), $distance = spearman$	9.77	10.32	86.51	[33]
Coverage area, pointwise defined (probability of AP match = 0.9)	10.03	9.44	86.64	[34]
Coverage area, distribution based (Gaussian distribution)	13.01	11.68	69.07	[36,37]

Figure 4. Benchmark positioning result on rank-based fingerprinting

See the benchmark result, 8–10 m in the best cases. And for floor detection, 92.9% is the best case. Link:[http://www.mdpi.com/2306-5729/2/4/32/htm#B15-data-02-00032].

2.4 Algorithms and Techniques: WKNN, DNN, AutoEncoder, CNN

In this project, we firstly implemented Weighted K Nearest Neighbor (WKNN) and Deep Learning Neural Network (DNN) compared to UJI previous studies. Then we implemented AutoEncoder based on Tensorflo w combined with DNN to improve the positioning accuracy. Since RSS is time series, we presented Convolutional neural network (CNN) for long term robust data.

WKNN:

For normal KNN algorithm, the input are Training RSS, Training Coordinate, Testing RSS, and K value. The output is the estimation value of Testing Coordinate by averaging the K neighbors coordinates. The best result is 10m accuracy by setting K equal to 2 using short term data. Review UJI KNN algorithm [5] 3D mean error in Benchmark, it's 8.7m, the improvement they make is changing distance measurements and preprocessing raw data before running algorithm.

The traditional/default distance measurement for KNN uses Euclidean distance to compare the difference between training RSS and testing RSS, but UJI uses Sorensen distance since it provides a better result. Another significant change is Fingerprint data representation. In UJI's report paper, they provide 4 ways of data representation which are Positive, 0-1 norm, Exponential, and Powed.

Positive values data representation is simply subtracting the minimum value of dataset. So new low value stands for lower strength signal, and high value represents for more strength signal.

$$Positive_i\left(x\right) = \begin{cases} (RSS_i - min) & If\ WAP_i\ is\ present\ in\ the\ fingerprint\ x\ and\ RSS_i \geq \tau \\ 0 & otherwise \end{cases}$$

Zero to one normalized representation corresponds to Positive values representation but all the values are normalized in the positive [0,1] range.

$$ZeroToOneNormalized_i\left(x\right) = \frac{Positive_i\left(x\right)}{-min}$$

A lot of studies have already proved that the RSS values in decibels dBm has a logarithmic scale instead of linear scale. So we introduce Exponential representation and powed representation.

$$Exponential_i\left(x\right) = \frac{\exp\left(\frac{Positive_i(x)}{\alpha}\right)}{\exp\left(\frac{-min}{\alpha}\right)}$$

$$Powed_i\left(x\right) = \frac{\left(Positive_i\left(x\right)\right)^{\beta}}{\left(-min\right)^{\beta}}$$

In Exponential representation and powed representation, signal fluctuations occur when the transmitter is increasing. For example, the variance of -104 dBm and -94 dBm is different to the variance of -11 dBm and -1 dBm.

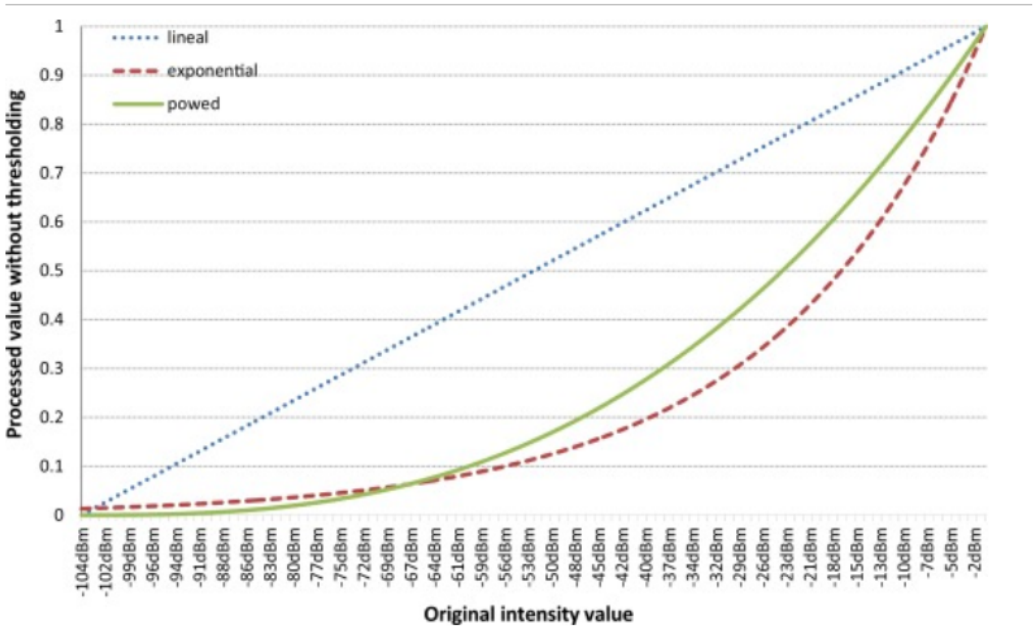


Figure 5. Visual meaning of the different representations used. Lineal stands for positive and normalized since both are proportional.

We improve UJI's KNN algorithm by multiplying a weight to each nearest neighbor instead of averaging all K neighbors. The weight is the inverse of distance between two neighboring RSS vectors. Our WKNN algorithm uses Sorensen distance as the measurement for choosing neighbors, and powed representation for data preprocessing. K value is set to 3 as we experienced so many times, and it gives us best result on accuracy which is 5.2 meters.

DNN:

Go to read my blog [http://jononearth.com/tags/deep-learning/] to check the detail for deep learning and neural network part.

neural networks are a class of machine learning algorithms to model complex patterns in datasets using multiple hidden layers and non-linear activation functions. It contains an input, passes it through multiple layers of hidden neurons(deep learning contains much more layers), and outputs a prediction representing the combined input of all the neurons.

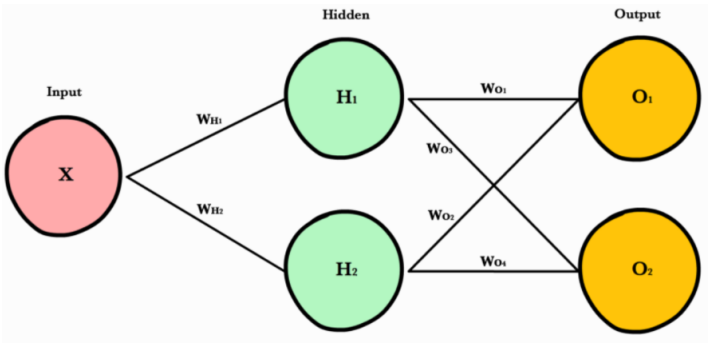
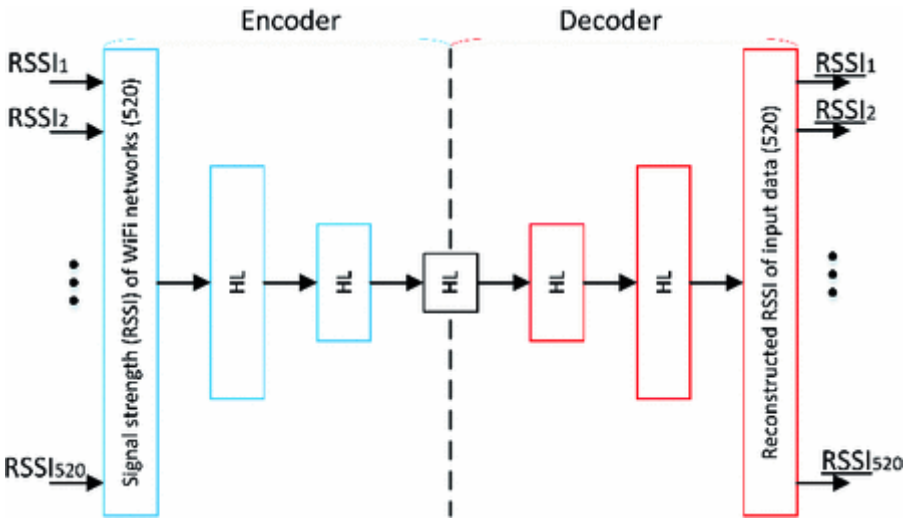


Figure 5. A simple Neural network model

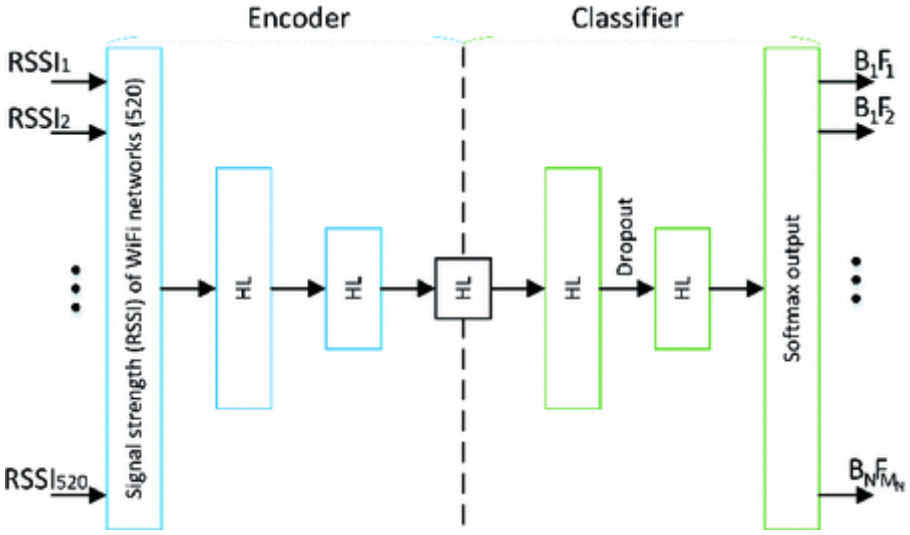
Neural networks are trained iteratively using optimization techniques like gradient descent. After every training, an error metric is calculated based on the difference between prediction and target.

the derivatives of this error metric are calculated and propagated back through the network using a technique called backpropagation. Each neuron's weights are the adjusted relative to how much they contributed to the total error. This process is repeated iteratively until the network error drops below an acceptable threshold.

AutoEncoder:



Stacked autoencoder (SAE) [6] used in DNN to determine floor and building. The input to SAE are signal strengths detected in a scan with one value for each network visible in the training database. The output of decoder is the reconstructed input from reduced representation. The HL stands for hidden layer and the numbers in parentheses represent the number of neurons in the layer



Architecture of DNN with SAE used to classify building and floor based on provided input WiFi scan. The already pre-trained encoder part is connected to classifier. The numbers in parentheses represent the number of neurons in the layer

CNN:

In traditional Neural Networks, we connect each input neuron to each output neuron in the next layer and learn the weights and biases of each neuron. However, in Convolutional Neural Networks[7], we use convolutions over the input layer to compute the output. Each layer applies different filters,during the training phase, a CNN automatically learns the values of its filters.

$$\begin{bmatrix} RSS_{1,1} & RSS_{1,2} & \dots & RSS_{1,T} \\ RSS_{2,1} & RSS_{2,2} & \dots & RSS_{2,T} \\ \vdots & \vdots & \vdots & \vdots \\ RSS_{N,1} & RSS_{N,2} & \dots & RSS_{N,T} \end{bmatrix}$$

The RSS data collected N times in one position

3 Methodology

3.1 Data Preprocessing:

a.missing values; -100,0,-103

In the dataset, if the signal is not heard by APs, give its value +100 dB. However, in real scenario, the signal is too small to be heard, the value is usually far less than -100dB, so in order to reduce the difference from +100 to -100, we will change the Non-heard APs to -100dB, and 0 dB, and compare the two changes.

b. test vs train; exchange

In this project, we change test sets as train sets. Because test sets is much larger than the train sets. You can see that in 5th floor, only have dozens of points.

Deep learning requires a lot of training data because of the huge number of parameters needed to be tuned by a learning algorithm. The problem with deep learning is that it starts off with a poor initial state and then some gradient based optimization (learning) algorithm is used to converge the network to an optimal solution, which might not necessarily be the global optimum. This process requires a lot of data, if the neural net starts off close to the required solution then it would consume less data during training[https://www.quora.com/Why-does-deep-learning-require-a-lot-of-data].

c. Coordinate transform;

For the first dataset, the coordinate is local, we change these coordinate to relative coordinate. The method is to

the minimum coordinate, like [x-x_{min}, y-y_{min}, z-z_{min}].

This data pre-processing method will improve the accuracy.

d. Scale

Why scale is very import?

When the models are distance based models, scale will have great effect on them. Scale can speed up the **gradient descent**, This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven. Besides, scale can avoid some features(which have big number) have more effect on the result and therefore, improve the accuracy.

3.2 Implementation

It mainly contain two parts:

1) Floor detection
2) Positioning in each floor

For the first dataset:

Using AutoEncoder to do the floor detection.

We first build the SAE containing encoder part and decoder part.

```
def encoder():
    model = Sequential()
    model.add(Dense(256, input_dim=input_size, activation='tanh', bias=True))
    model.add(Dense(128, activation='tanh', bias=True))
    model.add(Dense(64, activation='tanh', bias=True))
    return model
```

```
def decoder(e):
    e.add(Dense(128, input_dim=64, activation='tanh', bias=True))
    e.add(Dense(256, activation='tanh', bias=True))
    e.add(Dense(input_size, activation='tanh', bias=True))
    e.compile(optimizer='adam', loss='mse')
    return e
```

```
e = encoder()
d = decoder(e)
d.fit(train_X, train_X, nb_epoch=nb_epochs, batch_size=batch_size)
```

Then we remove the decoder part, add classifier.

```
def classifier(d):
    num_to_remove = 3
    for i in range(num_to_remove):
        d.pop()
    d.add(Dense(128, input_dim=64, activation='tanh', bias=True))
    d.add(Dense(128, activation='tanh', bias=True))
    d.add(Dense(num_classes, activation='softmax', bias=True))
    d.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
    return d
```

```
c = classifier(d)
c.fit(train_X, train_y, validation_data=(val_X, val_y), nb_epoch=nb_epochs, batch_size=batch_size)
```

The result we get is not bad, 91.38% , which is better than the most algorithms, but still have about 1% gap.

Using the DNN to train a simple mode to see if deep learning can improve the positioning accuracy. In the first floor, we get accuracy at 10m, which is not so good. Then we still use the AutoEncoder to train the a regression model. Get the 8.24m, which is much better but still is not the best result comparing to the benchmark positioning results.
future work: we only consider the first floor here, we should calculate the other floor. Maybe using transfer learning.

For the second long time dataset:
It is difficult to convert RSS data to RSS image data. First, we should group these data by the positions. For one position, RSS value was totally collected 348 times, this means the matrix is 348∗ 992.

```
train_group = train.groupby([448,449,450])
train_group.describe()
```

			0								1		...
			count	mean	std	min	25%	50%	75%	max	count	mean	...
448	449	450											
4.1254	16.691	3	348.0	-10.666667	29.237640	-95.0	0.00	0.0	0.0	0.0	348.0	-58.051724	...
		5	348.0	-68.528736	29.735274	-91.0	-84.00	-80.0	-75.0	0.0	348.0	0.000000	...
	18.479	3	348.0	-4.425287	19.557869	-94.0	0.00	0.0	0.0	0.0	348.0	-53.370690	...
		5	348.0	-60.887931	37.783801	-93.0	-86.00	-82.0	0.0	0.0	348.0	0.000000	...
	20.267	3	348.0	-7.727011	25.197723	-93.0	0.00	0.0	0.0	0.0	348.0	-66.264368	...
		5	348.0	-56.264368	39.984495	-94.0	-86.25	-81.0	0.0	0.0	348.0	0.000000	...
	22.055	3	348.0	-17.221264	35.326063	-94.0	0.00	0.0	0.0	0.0	348.0	-66.991379	...
		5	348.0	-41.362069	43.425858	-94.0	-86.00	0.0	0.0	0.0	348.0	0.000000	...
	23.843	3	348.0	-20.364943	37.341892	-94.0	0.00	0.0	0.0	0.0	348.0	-67.232759	...

To make better use of CNN, we split the matrix into 6∗ 992, as in real situation, the data were collected 6 times in one time for one position.

```
[[  0   0   0 ...,   0   0   0]
 [  0   0   0 ...,   0   0   0]
 [  0 -91   0 ...,   0   0   0]
 [  0   0   0 ...,   0   0   0]
 [  0 -92   0 ...,   0   0   0]
 [  0 -92   0 ...,   0   0   0]]
[[  0 -88 -84 ...,   0   0   0]
 [  0 -89 -86 ...,   0   0   0]
 [-95 -89 -87 ...,   0   0   0]
 [-94 -88 -87 ...,   0   0   0]
 [  0 -88 -85 ...,   0   0   0]
 [  0 -89 -89 ...,   0   0   0]]
[[  0 -93 -86 ...,   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   0   0]]
[[  0 -84   0 ...,   0   0   0]
 [  0 -86   0 ...,   0   0   0]
```

Then take these train values to the CNN model.

```
import keras
from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
from keras.layers import Dropout, Flatten, Dense
from keras.models import Sequential
from keras.optimizers import SGD, RMSprop, Adadelta, Adam
from keras import metrics

### TODO: Define your architecture.
model = Sequential()
# First Convolution Layer with patch size = 2, stride = 1, same padding, an depth = 16 and Relu activation
model.add(Conv2D(filters=16, kernel_size=2, padding='same', activation='relu',
                        input_shape = train5_x.shape[1:]))
# First Maxpool with kernel size = 2 and stride = 2
model.add(MaxPooling2D(pool_size=(2,2)))

# Second Convolution Layer with patch size = 2, stride = 1, same padding an depth = 32 and Relu activation
model.add(Conv2D(filters=32, kernel_size=2, padding='same', activation='relu'))
# Second Maxpool with kernel size = 2 and stride = 2
model.add(MaxPooling2D(pool_size=2))

# # Third Convolution Layer with patch size = 2, stride = 1, same padding an depth = 64 and Relu activation
# model.add(Conv2D(filters=64, kernel_size=2, padding='same', activation='relu'))
# # Third Maxpool with kernel size = 2 and stride = 2
# model.add(MaxPooling2D(pool_size=2), dim_ordering="tf")
# # add dropput to reduce overfitting
# # model.add(Dropout(0.5))

# Dense layer with 100 neurons and Relu activation
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
# Ouput layer with softmax activation with 133 neurons same as the category
model.add(Dense(2))

model.summary()
```

4 Refinement

4.1 Bagging

Bagging is method that can combine several weak learners into a strong one. In this project, we randomly choose 5 subsets from the original data and take them into a simple deep learning model to train. Then we create a vote system. For example, among the 5 classifiers we trained, 3 of them classify one data as floor 1, 2 of them classify the data as floor 4. The voting system then will consider the result is floor 1.

```python
# Create a random subsample from the dataset with replacement
from random import seed
from random import randrange

def subsample(x,y, ratio=0.5):
    sample_x = list()
    sample_y = list()
    n_sample = round(len(x) * ratio)
    while len(sample_x) < n_sample:
        index = randrange(len(x))
        sample_x.append(x[index])
        sample_y.append(y[index])
    sample_x = np.array(sample_x)
    sample_y = np.array(sample_y)
    return sample_x, sample_y
```

```python
def classifier():
    model = Sequential()
    model.add(Dense(256, input_dim=input_size, activation='tanh', bias=True))
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='tanh', bias=True))
    model.add(Dropout(0.5))
    model.add(Dense(num_classes, activation='softmax', bias=True))
    model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=['accuracy'])
    return model
```

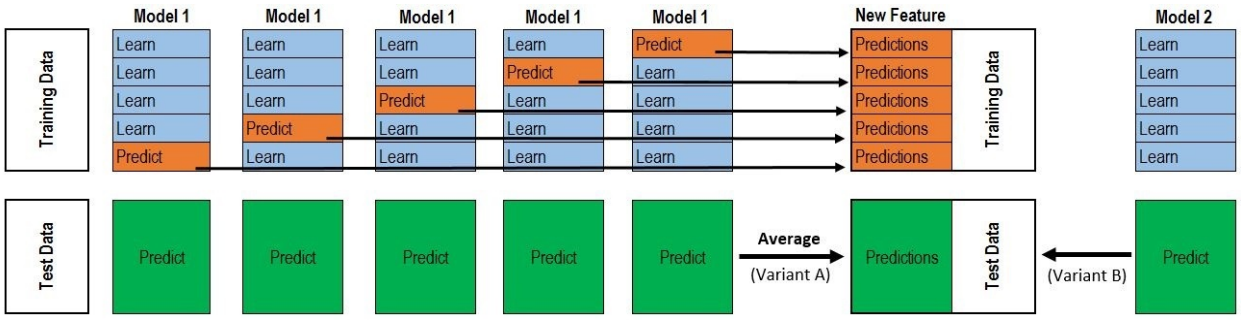
```python
from statistics import mode, StatisticsError
vote = []

for i in range(len(l2)):
    try:
        a = mode(l2[i])
        vote.append(a)
    except StatisticsError:
        vote.append(0)
```

The result is pretty well, get 100% accuracy, which can show the bagging's power.

4.2 Stacking

Stacking is a ensembling method used to combine information from different predictive models to create a new model. The stacked model(also called 2nd-level model as shown in figure) will outperform each of the individual models because it has the ability to highlight each base model where it performs best and discredit each base where it performs poorly. For this reason, stacking is the most effective when the base models are significantly different.



In the 1st-level model, we use 5 base model(2 Deep learning with different paraments, AutoEncoder, KNN, Decision Tree). For each base model in the training data, to do the K-fold cross validation, use its prediction as the new feature in the 2nd-level model. In the test data, for every model, it has the 5 predictions, then take the average of the prediction as the test data in the 2nd-level model.
In the 2nd-level, we create a simple deep learning with linear activation as our model. It achieves better result (accuracy at 7.2m) comparing to the base model(8∼ 10m).

```
blend_trainX = np.hstack((blend_train1,blend_train2, blend_train3,blend_train4,blend_train5))
blend_testX = np.hstack((blend_test1, blend_test2, blend_test3,blend_test4,blend_test5))

model_stack = Sequential()
model_stack.add(Dense(32, input_dim=10, activation='linear', bias=True))
model_stack.add(Dense(16, activation='linear', bias=True))
model_stack.add(Dense(2, activation='linear', bias=True))
model_stack.compile(optimizer='adam', loss='mean_squared_error',metrics=['accuracy'])
model_stack.fit(blend_trainX,trainY,nb_epoch=50)
predict_1 = model_stack.predict(blend_testX)
e_stack, e_stack_mean = accuracy(predict_1, testY)
print(e_stack_mean)
```

```
1264/1264 [==============================] - 0s - loss: 33.9529 - acc: 0.9699
Epoch 42/50
1264/1264 [==============================] - 0s - loss: 32.5108 - acc: 0.9699
Epoch 43/50
1264/1264 [==============================] - 0s - loss: 32.5502 - acc: 0.9715
Epoch 44/50
1264/1264 [==============================] - 0s - loss: 33.2535 - acc: 0.9707
Epoch 45/50
1264/1264 [==============================] - 0s - loss: 32.2443 - acc: 0.9699
Epoch 46/50
1264/1264 [==============================] - 0s - loss: 32.6447 - acc: 0.9715
Epoch 47/50
1264/1264 [==============================] - 0s - loss: 32.5811 - acc: 0.9691
Epoch 48/50
1264/1264 [==============================] - 0s - loss: 32.4924 - acc: 0.9676
Epoch 49/50
1264/1264 [==============================] - 0s - loss: 32.5852 - acc: 0.9707
Epoch 50/50
1264/1264 [==============================] - 0s - loss: 32.4005 - acc: 0.9699
7.19210026607
```

5 Results

1) Model evaluation

For the first dataset: we can see that using ensemble method is the best one that make the best prediction.

algorithms	Floor detection	Position accuracy(2D mean error)(1st floor)(m)
KNN	92%	8
WKNN	91%	5
Deep learning	87%	10
Autoencoder	91.38%	8
Ensemble	100%	7.19

For the second dataset: create a full-connected layer of neural network can temporary get a better result than CNN. However, applying CNN which normally used in computer vision to this long-term data can reduce the burden of computation and maybe can get a better result.

algorithms	Position accuracy(2D mean error)(5th floor)(m)
WKNN	2.1
DNN	3.3
CNN	3.4

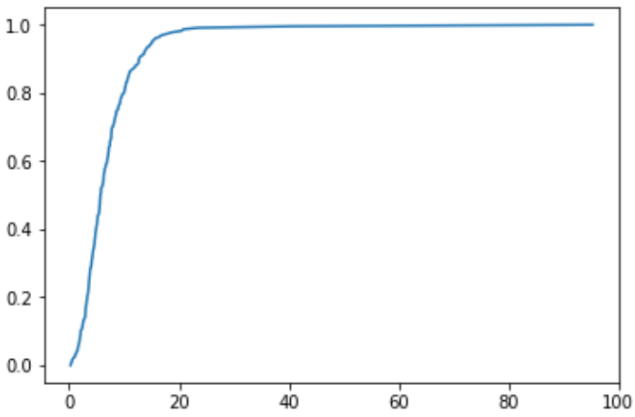
2) Justification

Compare to the benchmark, using the autoencoder and ensemble strategy can achieve a better results.

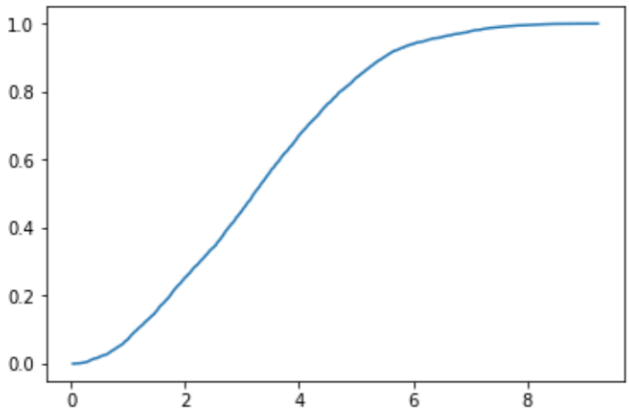
6 Conclusion

1) Free-form visualization

For the different dataset, the long-term dataset achieves a better result than short-term dataset. We can see this from the PDF figures. The variance of short-term data is larger than the long-term one. The long-term data is more robust to changes.



PDF of 1st floor position accuracy in first dataset



PDF of 3rd floor position accuracy in second dataset

2) Reflection

The process used for this project can be summarized using the following steps:

a) Indoor position introduction
b) Data exploration and visualization
c) Benchmark was created for project
d) Algorithm introduction
e) Data preprocessing

f) Train the floor detection classifier model and floor positioning regression model using different algorithms

The f part is the most difficult part for us. At first, we use the simple KNN algorithm to give it a try, but it can't precede the benchmark. Then we decide to use neural network. There are lots of paraments in a simple neural network. We didn't obtain a good result as first try. So, we keep trying looking for different paraments. Besides, using better strategy of deep learning for our project seems more important. At last, Autoencoder and ensemble is the strategy we are using for dataset1, CNN for dataset2. To understand and build stacking model (one method of ensemble) is time-consuming. For using CNN in long-term data which is very different from images data, data preprocessing is really difficult for us.

3) Improvement for future work

Though the project is almost done, but there are many things we can do in the future to improve the project.:

1) Better understand of deep learning in order to improve algorithms of it, since there are many parameters in it.
2) Train different model for each floor, since we only use the 1st floor as our example.
3) Using the transfer learning to deal with the problem of less data in some floor.
4) More details in dataset2 can be explored: The floor detection, different machine learning method to improve it accuracy, result changing in different month...
5) To collect our own data in real life
6) Combine different indoor positioning techniques to improve its accuracy, such as inertial.

The references can be check in this blog.

References

[1] Kin Leung Faheem Zafari, Athanasios Gkelias. A survey of indoor localization systems and technologies. *arXiv:1709.01015,.*

[2] Mladen Tomic Luka Batistić. Overview of indoor positioning system technologies. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO).*

[3] Joaquin; Richter Philipp; Leppäkoski Helena; Huerta Joaquin; Cramariuc Andrei Lohan, Elena Simona; Torres-Sospedra. Crowdsourced wifi database and benchmark software for indoor positioning.

[4] Richter Philipp Torres-Sospedra Joaquín Lohan Elena Simona Huerta Joaquín Mendoza-Silva, Germán Martín. Long-term wi-fi fingerprinting dataset and supporting material.

[5] Joaquín Huerta Joaquín Torres-Sospedra Raúl Montoliu Sergio Trilles, Óscar Belmonte. Comprehensive analysis of distance and similarity measures for wi-fi fingerprinting indoor positioning systems. *Expert Systems with Applications*, 42(23):263–9278, 2015.

[6] Jan Wietrzykowski Michał Nowicki. Low-effort place recognition with wifi fingerprints using deep learning. *arXiv:1611.02049*.

[7] Stanford CS class. Cs231n convolutional neural networks for visual recognition.

[8] Mustafa Elnainay Mai Ibrahim, Marwan Torki. Cnn based indoor localization using rss time-series.