Received Signal Strength Indicator based geolocation, a machine learning approach

Anthony Houdaille, Maël Fabien, and Alexandre Bec

Télécom ParisTech, 2018

Abstract. In this study, we will focus on predicting geolocation from Received Signal Strength Indicator (RSSI). The train set is provided by Sigfox. We will show that random forest regressors perform better than other classifiers. We will also explore a dependent variable modeling. Since the latitude and the longitude seem to be correlated in the given dataset, we also used the predicted latitude as an input for predicting longitude, and this method improved the accuracy metric by 6.7%.

Keywords: RSSI · Geolocation · Machine Learning.

1 The training set

The aim of this study is to provide a geolocation estimation using Received Signal Strength Indicator in the context of IoT. The aim is to allow a geolocation of low consumption connected devices that use the Sigfox network. State of the art models are able to be precise to the nearest kilometer in urban areas, and around ten kilometers in less populated areas.

1.1 Features

The training set provided by Sigfox is a 39250 rows data set. The features are the following :

- messid : The ID of the message.
- bsid : The ID of the base station.
- did : The ID of the device that sent the message.
- nseq : Details of the repetition. Incremented each time a message is sent.
- time_ux : The timestamp of the moment the message was sent.
- rssi : The received signal strength indicator, measured in dBm.
- bs_lat : The latitude of the base station that received the message.
- bs_lng : The longitude of the base station that received the message.

Another data set is provided and contains per message ID, the exact location, measured by GPS, at which the message was sent.

1.2 Building the feature matrix

One of the first task is to build a data set that can be exploited for machine learning applications.

In the first data set, each rows corresponds to one message received by one base station. Therefore, there are duplicated rows that could be concatenated in a one-hot encoder. Indeed, we will create one-hot encoders for :

- The Received Signal Strength of each base station
- Each base station latitude and longitude
- The "time ux" at which the message was received for each base station
- The n-seq, a measure that assesses the efficiency of the message transmitted
- An estimated distance measure based on the pathloss lognormal shadowing model
- The distance between the barycenter of the base stations and each base station

We also create additional features including :

- The total number of base stations which received a given message
- Additional transformations of the distance and the RSSI.

The distance measuring according to the pathloss lognormal is defined by :

$$P_{rx} = P_{tx} + K_c + 10\eta log\frac{d}{d_0}$$

where P_{rx} is the transmitted power (in dBm), P_{tx} the received power, d_0 the reference distance, and d the transmitter-receiver distance, to be isolated.

Finally, we concatenate the two data sets, scale it, group the rows by message ID, and split the data into a training and a test set. The final training data set is a 6068 x 1569 matrix.

The feature matrix construction is achieved the following way :

```
def feat_mat_const(df_mess_train, df_mess_test):

    df = pd.concat([df_mess_train, df_mess_test], axis=0).reset_index()
    df = df.drop(['Unnamed: 0', 'index'], axis=1)

    one_hot = pd.get_dummies(df['bsid'])
    one_hot_nseq = pd.get_dummies(df['nseq'])
    one_hot_empty = one_hot.rename(columns={x:'isbs' + str(y)
    for x,y in zip(one_hot.columns,range(1, len(one_hot.columns)+1))})
    one_hot_rssi = one_hot.rename(columns={x:'rssi' + str(y)
    for x,y in zip(one_hot.columns,range(1, len(one_hot.columns)+1))})
    one_hot_distance_power = one_hot.rename(columns={x:'distpwr' + str(y)
    for x,y in zip(one_hot.columns, range(1, len(one_hot.columns)+1))})
    one_hot_lat = one_hot.rename(columns={x:'lati' + str(y)
    for x,y in zip(one_hot.columns, range(1, len(one_hot.columns)+1))})
```

```
one_hot_lng = one_hot.rename(columns={x:'lngi' + str(y)
for x,y in zip(one_hot.columns, range(1, len(one_hot.columns)+1))})
one_hot_distance_station = one_hot.rename(columns={x:'distbs' + str(y)
for x,y in zip(one_hot.columns,range(1, len(one_hot.columns)+1))})
one_hot_time = one_hot.rename(columns={x:'time' + str(y)
for x,y in zip(one_hot.columns,range(1, len(one_hot.columns)+1))})

df['NBAntennes'] = 1

df = df.join(one_hot_rssi)
df.iloc[:,9:] = df.iloc[:,9:].multiply(df["rssi"], axis="index")

power_up_dbm = -51.70
lambda_onde = (2.8 * 10 ** 8)/ (868 * 10 ** 6)
kc = 20 * np.log((4 * np.pi)/ lambda_onde)
etha = 4
df['distpwr'] = 10**((power_up_dbm + kc - df['rssi']) / ( 10 * etha))
df = df.join(one_hot_distance_power)
df.iloc[:,261:] = df.iloc[:,261:].multiply(df["distpwr"], axis="index")

lat_dis = np.mean(df['bs_lat'])
lng_dis = np.mean(df['bs_lng'])
df['distbs'] = np.arccos( np.sin(np.radians(df['bs_lat'])) *
np.sin(np.radians(lat_dis))
+ np.cos(np.radians(df['bs_lat'])) *
np.cos(np.radians(lat_dis)) * np.cos( np.radians(lng_dis - df['bs_lng'])))
*6378137/1000
df = df.join(one_hot_distance_station)
df.iloc[:,529:] = df.iloc[:,529:].multiply(df["distbs"], axis="index")

df = df.drop('rssi',axis = 1)
df = df.drop('distpwr',axis = 1)
df = df.drop('bsid',axis = 1)
df = df.drop('distbs', axis=1)

df = df.join(one_hot_nseq)
df = df.drop('nseq',axis = 1)

df = df.join(one_hot_time)
df.iloc[:,788:] = df.iloc[:,788:].multiply(df["time_ux"], axis="index")

df = df.join(one_hot_lat)
df.iloc[:,1047:] = df.iloc[:,1047:].multiply(df["bs_lat"], axis="index")

df = df.join(one_hot_lng)
```

```
    df.iloc[:,1306:] = df.iloc[:,1306:].multiply(df["bs_lng"], axis="index")

    df = df.drop(['bs_lat', 'bs_lng', 'time_ux'], axis=1)

    df = df.loc[:, (df != 0).any(axis=0)]
    df['sum_dist'] = np.sum(df.filter(like='dist'), axis=1)
    df['avg_dist'] = np.mean(df.filter(like='dist'), axis=1)
    df['sum_rssi'] = np.sum(df.filter(like='rssi'), axis=1)
    df['avg_rssi'] = np.mean(df.filter(like='rssi'), axis=1)
    df['avg_lat'] = np.mean(df.filter(like='lati'), axis=1)
    df['avg_lng'] = np.mean(df.filter(like='lngi'), axis=1)
    df['sum_lat'] = np.sum(df.filter(like='lati'), axis=1)
    df['sum_lng'] = np.sum(df.filter(like='lngi'), axis=1)

    df = pd.concat([df[['messid', 'NBAntennes', 'did']],
    (df.drop(['messid', 'NBAntennes', 'did'], axis=1)-
    df.drop(['messid', 'NBAntennes', 'did'], axis=1).mean())/
    df.drop(['messid', 'NBAntennes', 'did'], axis=1).std()], axis=1)

    df_train = df[:39250]
    df_test = df[39521:]

    df_train = pd.concat([df_train, pos_train], axis=1)
    df_train = df_train.groupby(['messid', 'did', 'lat', 'lng']).sum().reset_index()

    cols = list(df_train.columns.values)
    cols.pop(cols.index('lat'))
    cols.pop(cols.index('lng'))
    df_train = df_train[cols+['lat', 'lng']]

    df_test = df_test.groupby(['messid', 'did']).sum().reset_index()

    return df_train, df_test
```

1.3 Exploratory data analysis

Some distributions can be plotted in order to understand the data set.

The empirical distribution of the RSSI in the data set looks like a skewed normal. In the decibel milliwatts measure, a value closer to 0 indicates a stronger signal. Compared to signal strength in signal processing, those values are relatively small.

On average, six base stations received each message. This gives us an idea of how many points we will be able to rely on for each message geolocation estimation.

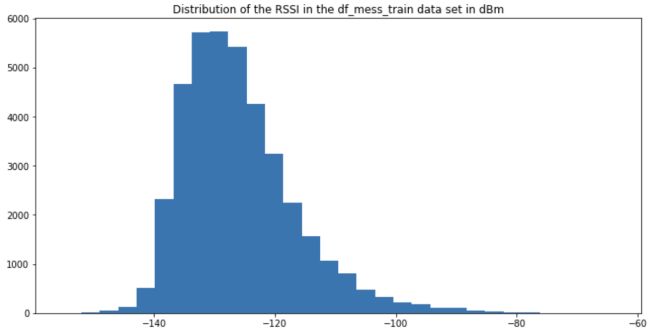

Fig. 1. Distribution of the RSSI in the data set

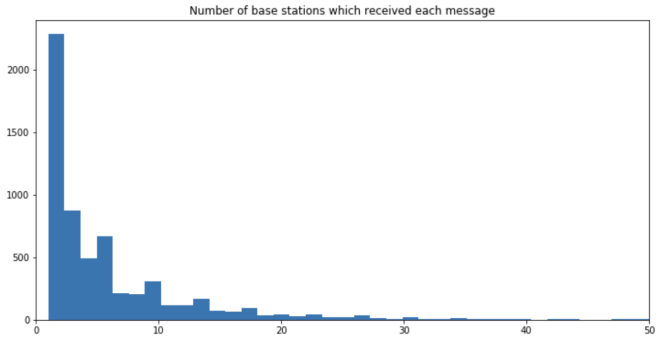

Fig. 2. Number of base stations which received each message

Since the data set relies on the geographical datas, we can plot them graphically using the Folium library. The data points are located within the Denver area (Colorado, US).

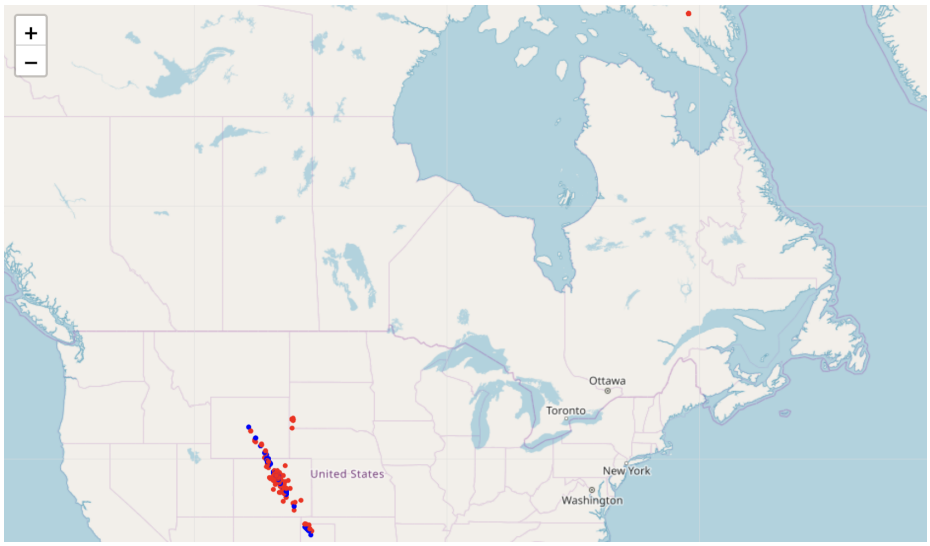

Fig. 3. Map of the location of the base stations (red) and messages (blue)

We observe that some base stations are not lying within the same geographic area than the messages. This is problematic since those base stations have all received at least one signal.

To deal with those outliers, we should re-locate the base stations to the barycenter of the observations. We do indeed not have any better information that could allow us to correct the position more precisely.

After leading a Principal Component Analysis, we can take a closer look at the circle of correlation to understand the influence of each column on the latitude and longitude to estimate. The key elements from this analysis are the following :

- the latitude and the longitude are obviously highly correlated with the base station coordinates
- the device id is close to uncorrelated to the locations, which seems intuitive
- it is quite similar for the nseq
- the base station id might have some impact on the final location, since it gives us information on where the message has been received. This information is also caught in the base station latitude and longitude.

- the RSSI and the time-ux seem to bring some insights, but with less confidence for the time-ux. This can be explained by the fact that a slight error in the time-ux means a large error on the estimated distance.

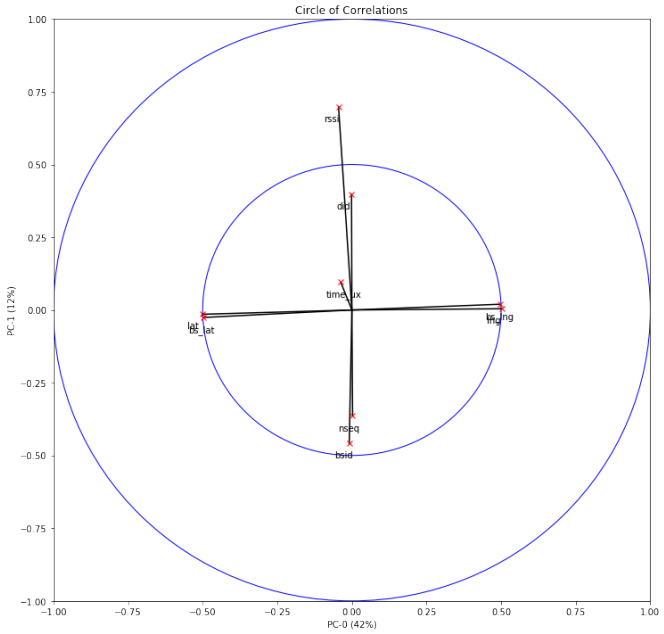

Fig. 4. Circle of correlation

2 Building the predictions

We need to predict both the latitude and the longitude of the geolocation of the message. This is a multi target regression case. We will explore two approaches :

- Considering both targets as independent
- Considering both targets as dependent, and modelling a single target based on the prediction of the other.

For both cases, we have tested several regressors, among which :

- A K-Nearest Neighbors regressor
- A Linear regressor
- A Support Vector regressor
- A Decision tree regressor

- A Random Forest regressor
- An Extra Trees regressor
- An XGBoost regressor
- And an Artificial Neural Network

Other approaches producing less relevant results will not be presented here. The metrics we are considering is the 80% percentile of the precision error on the geolocation.

2.1 Independence hypothesis

The results of the different regressors are summarized in the Table 1. We noticed that the devices that sent the messsages were different in the test and the train sets provided. Therefore, to avoid overfitting, we need to split the train set in two sets that both contain different devices.

Table 1. 80% percentile error achieved per model

Model	Error (m)
Extra Trees Regressor	2315
Random Forest Regressor	2760
KNN Regressor	3235
Decision Tree Regressor	3347
XGBoost	4149
Linear Model Regressor	4266
Support Vector Regressor	5393
Artificial Neural Network	82143

Dimension reduction techniques have also been tested, among which :

- a Principal Component Analysis
- a Truncated SVD

The Truncated SVD technique was inspired by the Latent Semantic Analysis (LSA) in Natural Language Processing in which we usually deal with sparse matrices. It ended up improving the accuracy of the XGBoost Regressor.

However, the best regressor was the Extra Trees. The error cumulative probability is presented in Figure 5.

Looking at the relative feature importance in the feature matrix on the whole feature matrix, including the one hot encoders, we notice that the distance measure of the different base stations appears among the most significant features. This shows that the theoretical foundations of our feature engineering is necessary.

We observe that our method manages pretty well to identify the cluster effects in some areas of the city.

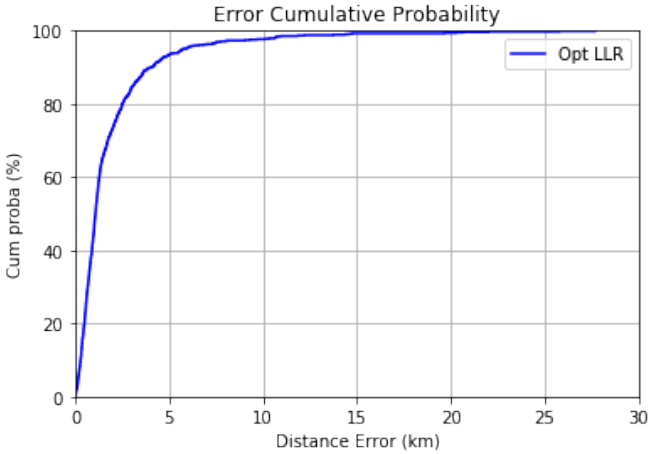

Fig. 5. Error Cumulative Probability

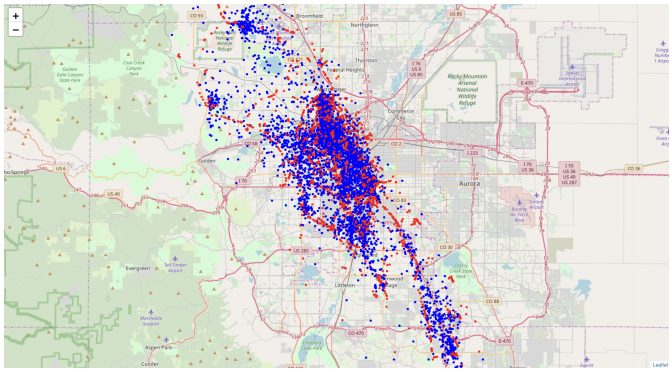

Fig. 6. Prediction map of the geolocation of the message ID (blue) vs. true position (red)

2.2 Relaxed independence hypothesis

Visually, the independence hypothesis would hold if the targets (latitude and longitude) were more or less uncorrelated. However, in our case, we can observe a clear relationship illustrated by the red linear regression line.

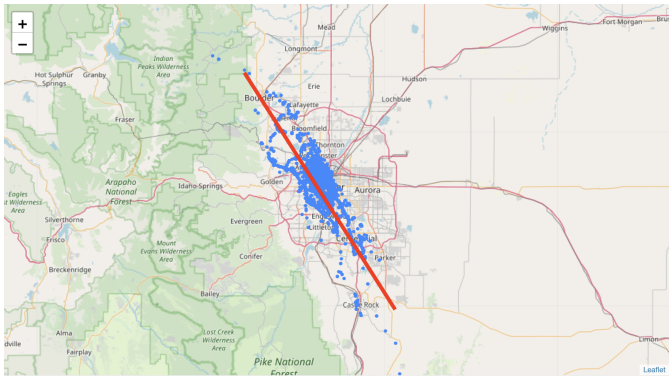

Fig. 7. Map of the location of the base stations, linear regressor

This has a strong implication : by predicting one of the targets, for example the latitude, we have additional information to predict the second one, in such case the longitude.

There are two ways to implement this :

- predicting the latitude, and adding to the feature matrix before predicting the longitude
- predicting the longitude, and adding to the feature matrix before predicting the latitude

It turns out that using the predicted longitude to predict latitude is the best option. Graphically, this can be justified. The longitude is the biggest source of position variation in the data. Therefore, if we have an idea, more or less precise of the longitude of the base station, it is easier to predict the latitude.

Adjusting our best extra trees model, this produces an error of 2161 meters instead of 2315 meters, which is a 6.7% error improvement.

3 Conclusion

We have shown through this brief paper that Sigfox devices can be used for geolocation in the context of IoT at an 80% cumulative error of around 2900 meters. There is space for improvements in our approach, and deep learning techniques could also help reach a higher degree of precision in the position estimates.

We have also shown that a more theoretical feature importance remains important even in the domain of RSSI. The best model probably lies somewhere between machine learning and signal processing.

Finally, we have shown that modeling dependent targets in this special example does not significantly impact the result.