I. Johnson and M. Hahsler. *arulesCBA: Classification Based on Association Rules*, 2019. URL https://CRAN.R-project.org/package=arulesCBA. R package version 1.1.5. [p]

H. Kim. *Discretization: Data Preprocessing, Discretization for Classification.*, 2012. URL https://CRAN.R-project.org/package=discretization. R package version 1.0-1. [p]

T. Kliegr. QCBA: Postoptimization of quantitative attributes in classifiers based on association rules. *arXiv preprint*, 2017. URL https://arxiv.org/abs/1711.10166. [p]

T. Kliegr. *Arc: Association Rule Classification*, 2018. URL https://CRAN.R-project.org/package=arc. R package version 1.2. [p]

T. Kliegr and J. Kuchar. Tuning hyperparameters of classification based on associations (CBA). In *Proceedings of the 19th Conference Information Technologies - Applications and Theory ITAT'19*. CEUR-WS.org, 2019. [p]

T. Kliegr, J. Kuchař, D. Sottara, and S. Vojíř. Learning business rules with association rule classifiers. In A. Bikakis, P. Fodor, and D. Roman, editors, *International Symposium on Rules and Rule Markup Languages for the Semantic Web (RuleML 2014): Rules on the Web. From Theory to Applications*, pages 236–250. Springer-Verlag, 2014. ISBN 978-3-319-09870-8. URL https://doi.org/10.1007/978-3-319-09870-8_18. [p]

J. Kuchar. *rCBA: CBA Classifier for R*, 2018. URL https://CRAN.R-project.org/package=rCBA. R package version 0.4.3. [p]

W. Li, J. Han, and J. Pei. CMAR: Accurate and efficient classification based on multiple class-association rules. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 369–376, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-1119-8. URL https://doi.org/10.1109/ICDM.2001.989541. [p]

B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, KDD'98, pages 80–86. AAAI Press, 1998. [p]

R. S. Mickalski, I. Mozetic, H. J., and H. Lavrack. The multi purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proceedings of the 5th National Conference on Artificial Intelligence*, 1986. [p]

P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006. ISBN 0321321367. [p]

K. Vanhoof and B. Depaire. Structure of association rule classifiers: a review. In *2010 International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 9–12, 2010. URL https://doi.org/10.1109/ISKE.2010.5680784. [p]

H. Yang, C. Rudin, and M. Seltzer. Scalable Bayesian rule lists. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3921–3930. JMLR. org, 2017. [p]

H. Yang, M. Chen, C. Rudin, and M. Seltzer. *sbrl: Scalable Bayesian Rule Lists Model*, 2019. URL https://CRAN.R-project.org/package=sbrl. R package version 1.2. [p]

X. Yin and J. Han. CPAR: Classification based on predictive association rules. In *Proceedings of the SIAM International Conference on Data Mining*, pages 369–376, San Franciso, 2003. SIAM Press. [p]

Michael Hahsler
Office of Information Technology and Department of Engineering Management, Information, and Systems
Bobby B. Lyle School of Engineering
Southern Methodist University
P. O. Box 750123, Dallas, TX 75275, USA
mhahsler@lyle.smu.edu

Ian Johnson
Google,
Boulder, CO, USA
ianjjohnson@icloud.com

Tomáš Kliegr
Department of Information and Knowledge Engineering

Faculty of Informatics and Statistics
University of Economics, Prague
Winston Churchill Sq. 4, Prague, Czech Republic
ORCiD https://orcid.org/0000-0002-7261-0380
first.last@vse.cz

Jaroslav Kuchař
Web Intelligence Research Group
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9, 160 00, Prague, Czech Republic
first.last@fit.cvut.cz

Indoor Positioning and Fingerprinting: The R Package ipft

by Emilio Sansano, Raúl Montoliu, Óscar Belmonte and Joaquín Torres-Sospedra

Abstract Methods based on Received Signal Strength Indicator (RSSI) fingerprinting are in the forefront among several techniques being proposed for indoor positioning. This paper introduces the R package **ipft**, which provides algorithms and utility functions for indoor positioning using fingerprinting techniques. These functions are designed for manipulation of RSSI fingerprint data sets, estimation of positions, comparison of the performance of different positioning models, and graphical visualization of data. Well-known machine learning algorithms are implemented in this package to perform analysis and estimations over RSSI data sets. The paper provides a description of these algorithms and functions, as well as examples of its use with real data. The **ipft** package provides a base that we hope to grow into a comprehensive library of fingerprinting-based indoor positioning methodologies.

Introduction

Intelligent spaces, as a particularity of the concept known as Ambient Intelligence (AmI) (Aarts and Wichert, 2009; Werner et al., 2005), where agents communicate and use technology in a non-intrusive way, have an interest in both open and closed environments. Since people spend 90% of time indoors (Klepeis et al., 2001), one of the most relevant aspects of AmI is indoor localization, due to the large number of potential applications: industrial and hospital applications, passenger transport, residences, assistance to emergency services and rescue, localization and support guide for the disabled, leisure applications, etc. It is expected that the global market for this type of location will grow from USD 7.11 billion in 2017 to USD 40.99 billion by 2022 (Research and markets, 2017), being among the key technologies in the future. This is a technology that has already awakened but that in a short period of time will suffer a big explosion, as happened with the systems of positioning by satellite in exteriors and its applications.

This paper introduces the R package **ipft** (Sansano, 2017), a collection of algorithms and utility functions to create models, make estimations, analyze and manipulate RSSI fingerprint data sets for indoor positioning. Given the abundance of potential applications for indoor positioning, the package may have a broad relevance in fields such as pervasive computing, Internet of Things (IoT) or healthcare, among many others.

The main progress in indoor location systems has been made during the last years. Therefore, both the research and commercial products in this area are new, and researchers and industry are currently involved in the investigation, development and improvement of these systems. We believe that the R language is a good environment for machine learning and data analysis related research, as its popularity is constantly growing [1], researchers related to indoor positioning have explicitly selected R as developing framework for their experiments (Quan et al., 2017; Harbicht et al., 2017; Popleteev et al., 2011), it is well maintained by an active community, and provides an ecosystem of good-quality packages that leverage its potential to become a standard programming platform for researchers. There are some open source applications and frameworks to build indoor positioning services, such as FIND [2], Anyplace [3] or RedPIN [4], based on fingerprinting techniques but, as far as we know, there is not any public framework or package that provides functions and algorithms to manipulate fingerprinting datasets and experiment with positioning algorithms.

RSSI (Received Signal Strength Indicator) positioning systems are based on measuring the intensities of the received radio signals of the emitting devices (beacons) that are available at a particular position, and comparing them with a previously built RSSI data set (yub Lee et al., 2013). RSSI is used to measure the relative quality of a received signal to a client device, and each chipset manufacturer is free to define their own scale for this term. The value read by a device is given on a logarithmic scale and can correspond to an instant reading or a mean of some consecutive readings.

In this scenario, a fingerprint is an RSSI feature vector composed of received signal values from different emitting devices or beacons, associated to a precise position. In the last years, this technique is becoming increasingly important for indoor localization (Liu et al., 2007; He and Chan, 2016), since Wi-Fi is generally available in indoor environments where GPS signals cannot penetrate, and the

[1] `https://stackoverflow.blog/2017/10/10/impressive-growth-r/`
[2] `https://www.internalpositioning.com#about`
[3] `https://anyplace.cs.ucy.ac.cy`
[4] `http://redpin.org`

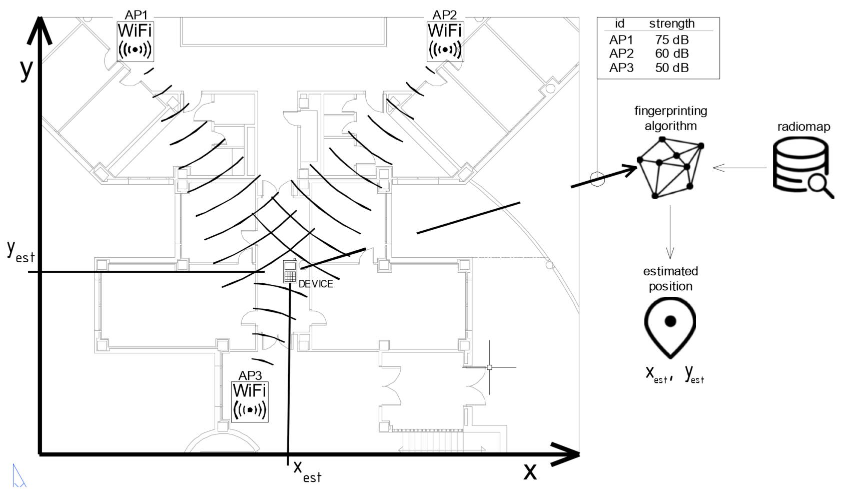

Figure 1: During the on-line phase, once the radio map has been built, the fingerprinting algorithm uses it to estimate the device's position by comparing the RSSI values heard by the device with the ones stored in the radio map.

wireless access points (WAPs) can be used as emitting devices (Li et al., 2006). Other types of indoor localization RF emitters, such as Bluetooth (Wang et al., 2013), RFID (Liu et al., 2011), or Ultra Wide Band (UWB) (Gigl et al., 2007), can be also used in combination with Wi-Fi access points or as a standalone positioning system.

The RSSI fingerprinting localization approach requires two phases of operation: a training phase, also known as off-line or survey phase, and a positioning phase, sometimes referred as on-line, runtime or tracking phase. In the training phase, multidimensional vectors of RSSI values (the fingerprints) are generated and associated with known locations. These measurements are used to build a data set (also known as radio map) that covers the area of interest. This data set can include, along with the collected RSSI values and the location coordinates, many other useful parameters, as the device type used in the measurements or its orientation. Later, during the positioning phase, an RSSI vector collected by a device is compared with the stored data to generate an estimation of its position (Figure 1).

Despite the increasing interest in RSSI positioning (Xiao et al., 2016), this topic has not been explicitly covered yet by any publicly available R package. The proposed package has been developed to provide users with a collection of fundamental algorithms and tools to manipulate RSSI radio maps and perform fingerprinting analysis. While fundamental algorithms and similarity measurement functions are implemented to provide a main framework for research and comparison purposes, these are highly customizable, to allow researchers to tailor those methods with their own parameters and functions.

This paper describes these algorithms and their implementation, and provides examples of how to use them. The remainder of the paper is structured as follows: Section Problem statement. Terminology and notation defines the fingerprinting problem statement and the nomenclature that will be used in the rest of the paper. An overview of the implemented algorithms is given in Section An overview of the implemented algorithms. Section Data wrangling outlines some data wrangling techniques included in the package. Section Positioning algorithms describes the implemented positioning algorithms. Section Beacon position estimation presents the included methods for access point position estimation. Then, Section Data clustering discuses some tools and functions included to create clusters or groups of fingerprints. Section Plotting functions illustrates the use of the plotting functions also included in the package. In all these sections, functions are described and explored using practical examples, and particular emphasis is placed on how to use them with real world examples and data sets. Finally, the paper is summarized in Section Summary.

Problem statement. Terminology and notation

This section provides a brief and general introduction to the principles of fingerprinting positioning, as well as a description of the notation and terminology that will be used in the next sections. The terms described here are related to general concepts of fingerprinting techniques, while the remaining of the paper describes the particular implementation of these concepts in the `ipft` package.

The main goal of the indoor localization techniques is to determine the position of a user in an indoor environment, where the GPS signal might not be received. This objective might require the use of an existing infrastructure, the deployment of a new one, the use of the so-called signals-of-opportunity (Yang et al., 2014), or even a combination of some of these techniques. Many of these techniques take advantage of the radio-frequency signals emitted by devices, whose position can be known or not, to estimate the user's position from the perceived strength of these signals. There are many kinds of devices that can be used for this purpose, such as Wi-Fi access points, bluetooth beacons, RFID or UWB devices, but for all of them, the information provided for a given position, the fingerprint, can be stored as a vector of received signal strength intensities (RSSI), whose length is determined by the number of detected emitters.

A radio map, or a fingerprinting data set, is composed of a set of collected fingerprints and the associated positions where the measurements were taken, and may contain some additional variables, such as the the type of device used or a time stamp of the observation, among any other useful data. Let $\mathcal{D}$ be a fingerprinting data set. Then:

$$\mathcal{D} = \{\mathcal{F}, \mathcal{L}\}$$

where $\mathcal{F}$ is the set of collected fingerprints and $\mathcal{L}$ is the set of associated locations.

For research purposes, a fingerprinting data set is usually divided into training and test sets. The training data set is used to store the fingerprints and location data to create models of the environment that can be used to estimate the position of a new fingerprint. The test data set is used to test the models obtained from the training data, and to compute the errors from the results of the position estimation.

Let $\mathcal{D}_{train}$ be a training data set:

$$\mathcal{D}_{train} = \{\mathcal{F}_{train}, \mathcal{L}_{train}\}$$

where

$$\mathcal{F}_{train} = \{\lambda_1^{tr}, \lambda_2^{tr}, ..., \lambda_n^{tr}\}$$

$$\mathcal{L}_{train} = \{\tau_1^{tr}, \tau_2^{tr}, ..., \tau_n^{tr}\}$$

$\mathcal{D}_{train}$ is composed of n fingerprints, stored as n vectors of RSSI measurements ($\lambda_i^{tr},\ i \in [1, 2, ..., n]$), and n locations ($\tau_i^{tr},\ i \in [1, 2, ..., n]$), stored as vectors, representing the position associated with its correspondent fingerprint. Each fingerprint consists of q RSSI values ($\rho_{h,i}^{tr}, h \in [1, ..., q]$), where q is the number of beacons considered when building the training set:

$$\lambda_i^{tr} = \left\{\rho_{1,i}^{tr}, \rho_{2,i}^{tr}, ..., \rho_{q,i}^{tr}\right\},\ i \in [1, ..., n]$$

and each associated position is composed of one or more values, depending on the number of dimensions to be considered and the coordinate system used. The position can be given as a vector of values representing its coordinates, although on multi-floor and multi-building environments labels can be used to represent buildings, floors, offices, etc. Let l be the number of dimensions of a position vector. Then:

$$\tau_i^{tr} = \left\{\nu_{1,i}^{tr}, \nu_{2,i}^{tr}, ..., \nu_{l,i}^{tr}\right\},\ i \in [1, ..., n]$$

The test data set is also composed of a collection of fingerprints associated to known positions. This data set is used for testing purposes, during research or during model building adjustments, to assess the model's performance by comparing its estimation of the positions with the ground truth.

The situation is different in real applications, where the goal is to estimate the unknown position of the receiver given the RSSI values detected at a particular location, using a previously built model. In this case, the test data set is just composed of a unique fingerprint, and the objective is to estimate the actual location of the receiver. Therefore, no information about its location is provided.

The test data set is composed of m observations:

$$\mathcal{D}_{test} = \{\mathcal{F}_{test}, \mathcal{L}_{test}\}$$

where

$$\mathcal{F}_{test} = \{\lambda_1^{ts}, \lambda_2^{ts}, ..., \lambda_m^{ts}\}$$

$$\mathcal{L}_{test} = \{\tau_1^{ts}, \tau_2^{ts}, ..., \tau_m^{ts}\}$$

To be able to compare the test observations with the training fingerprints, the number of RSSI values of its respective fingerprints has to be the same, and the position in the RSSI vector must represent the same beacon in both data sets. Therefore, each one of the m observations of the test data set is composed of a fingerprint with q RSSI values:

$$\lambda_j^{ts} = \left\{\rho_{1,j}^{ts}, \rho_{2,j}^{ts}, ..., \rho_{q,j}^{ts}\right\}, \ j \in [1, ..., m]$$

and a location vector with the same spatial dimensions as the training location vectors:

$$\tau_j^{ts} = \left\{\nu_{1,j}^{ts}, \nu_{2,j}^{ts}, ..., \nu_{l,j}^{ts}\right\}, \ j \in [1, ..., m]$$

The notation depicted above will be used in the remaining of the paper to represent the fingerprinting data. Symbols i and j will be used to represent iterations over the training and test data sets, respectively, while h will be used to iterate over the beacons present in each fingerprint.

An overview of the implemented algorithms

This section presents an introduction to the main functions, included in the **ipft**[5] package, that implement fingerprinting-based indoor localization methods. The package also provides two data sets for training and validation purposes that are briefly described in this section.

The `ipft` package implements three algorithms to build models to estimate the position of a receiver in an indoor environment. Two of these implementations are based on the well known k-Nearest Neighbors algorithm (*knn*) (Cover and Hart, 1967) to, given an RSSI vector, select the k most similar training examples from the radio map. The similarity between the RSSI value vectors can be measured, for example, as the *euclidean* distance between them, but other distance functions may be used (Torres-Sospedra et al., 2015b). The selection of a method to compute this measure can be provided to the function in two ways, either choosing one of the already implemented distance measurements (*euclidean*, *manhattan*, etc.), or by way of a reference to a function implemented by the user that returns the distance (the lower, the more similar or 'closer') between two matrices or vectors. Once the k neighbors are selected, the location of the user is estimated as the weighted average of the neighbors positions.

The first implementation, corresponding to the function `ipfKnn`, may behave in a deterministic way, finding the k more similar neighbors using a deterministic similarity function such as the euclidean or manhattan distances, or in a probabilistic way, using similarity functions such as LDG (Logarithmic Gaussian Distance) or PLGD (Penalized Logarithmic Gaussian Distance) (Cramariuc et al., 2016b), that are based upon statistical assumptions on the RSSI measurement error. The similarity function can be chosen from the set of implemented options or provided by the user via a custom function. This implementation is discussed in the Section The `ipfKnn` function.

The other implementation of the knn algorithm assumes a probabilistic nature for the received signal distribution (Roos et al., 2002) and uses collections of many fingerprints at each particular position, acquired during the training phase. Therefore, the radio map is composed of several groups, where a group is a set of fingerprints (vectors of RSSI values) that share the same location. Assuming that the RSSI value for a specific beacon can be modeled as a random variable following a normal distribution (Haeberlen et al., 2004), any of these collections, or groups, of fingerprints can be represented by the statistical parameters of this distribution, in this case, the mean and the standard deviation. This implies that the original data set can be transformed into a new type of data structure by storing the mean and the standard deviation of every detected beacon for every group. All the original data for a group is transformed into two vectors, one storing the means and the other the standard deviations. The trustworthiness of the data in the new data set will depend on the number of measurements for every location of the original data. It is assumed that the more measurements for a particular location, the more reliable will be their inferred statistical parameters.

The implementation of this probabilistic-based method takes the original radio map and a set of group indices, and fits these groups of measurements to a normal (Gaussian) distribution for every beacon and every location, so that the signal intensity distribution is determined by the mean

[5]The **ipft** package is available at CRAN and can be installed as any other R package:
```
> install.packages("ipft")
```
The package has to be loaded into the main environment to use it for the first time in an R session:
```
> library("ipft")
```

and the standard deviation of the Gaussian fit. Then, given a test fingerprint, the algorithm estimates its position by selecting the k most probable locations, making explicit use of the statistical parameters of the data stored in the radio map to optimize the probabilities in the assignment of the estimated position by computing a similarity function based on a summatory of probabilities. This approach is implemented through the `ipfProbabilistic` function and is described in the Section The `ipfProbabilistic` function.

Finally, the third implemented algorithm is based on a scenario where the location of the beacons is known, and an estimation of the fingerprint position can be made using the log-distance path loss model (Seybold, J.S., 2005). The strength of the received signal at a particular point can be modeled as a function of the logarithmic distance between the receiver and the emitter and some parameters related to the environment properties and the devices characteristics. Therefore, as this method uses an analytical model to evaluate the position, no radio map is needed to train a model to compare fingerprints with, since the position might be estimated from the fingerprint data and the position of the beacons. This method is implemented by the function `ipfProximity` and is described in Section The `ipfProximity` function.

The previous functions `ipfKnn`, `ipfProbabilistic` and `ipfProximity` create models based on the training data and parameters provided. These models can then be evaluated using the `ipfEstimate` function, that internally detects the algorithm to apply based on the model that receives as parameter.

The package also includes data from the IPIN2016[6] Tutorial data set. In the `ipftrain` data frame there are $n = 927$ observations, including the RSSI values for $q = 168$ wireless access points, the location, expressed in Cartesian coordinates, for the observation (x, y), and some other variables, as timestamps for the measurements or an identifier for the user who took the survey. The `ipftest` data frame contains $m = 702$ observations with the same structure, for testing and validation purposes. The fingerprints included in both data sets where taken in the same building and the same floor. The `ipfpwap` data frame contains the position of 39 of the WAPs included in the `ipftrain` and `ipftest` data sets. The unknown positions of the remaining WAPs are stored as `NA`. The characteristics of these data sets attributes are:

- `RSSI values`: Columns from 1 to 168. The values represent the strength of the received signal expressed in decibels, on a scale that ranges from −30dBm to −97dBm in the training set, and from −31dBm to −99dBm in the test set. The closer the value to zero, the stronger the signal.
- `position`: Columns 169 (X) and 170 (Y). The position given in Cartesian coordinates, with its origin in the same corridor where the data was acquired.
- `user id`: A numeric value from 1 to 8 to represent each of the 8 users that acquired the train data set. The test dataset was acquired by a different user, represented by the value 0.
- `timestamp`: The UNIX time stamp of the observation, in seconds.

There are some other publicly available indoor location data sets that have been used to develop and test this package and that are not included for size reasons, as the UJIIndoorLoc Data Set (Torres-Sospedra et al., 2015a) or the Tampere University data set (Cramariuc et al., 2016a).

The theoretical foundations of the algorithms and its uses are discussed in detail in Section Positioning algorithms. A description of the functions `ipfKnn`, `ipfProximity`, `ipfProbabilistic` and `ipfEstimate` is given while presenting some simulations to show how these algorithms can be useful in practice.

Data wrangling

An RSSI fingerprint is a vector composed of signal strength measurements from all the emitters received by a client device at a particular point, and can be measured in any unit of power. It is often expressed in decibels (dBm), or as percentage values between 1-100, and can be a negative or a positive value. Typically this values are stored as negative figures, where the strongest signals are closer to zero.

Some algorithms are sensitive to the scale of the data. For example, Neural Networks generally work better (**?**) with data scaled to a range between [0, 1] or [−1, 1], since unscaled data may slow down the learning process and the convergence of the network parameters and, in some cases, prevent the network from effectively learning the problem. Thus, the first step before the data can be fed to a positioning algorithm may involve some kind of transformation, depending on the characteristics of the original data.

The data sets included in this package represent the RSSI data from a set of wireless access points as negative integer numbers from −99 (weakest detected signal) to −30 (strongest detected signal).

[6]http://www3.uah.es/ipin2016/

When the RSSI of a WAP is not available, the value used is NA. This convention may be inconvenient for some calculations. For example, a similarity measure between two fingerprints as the euclidean distance will only take into account those WAPs that have been detected in both observations, causing a loss of information that otherwise could be utilized.

The **ipft** package contains some functions to manipulate and wrangle raw fingerprint data. The ipfTransform function mutates the given fingerprint data into a new data set with a specified range for the RSSI signals. The signature of the function is:

```
ipfTransform <- function(data, outRange = c(0, 1), outNoRSSI = 0, inRange = NULL,
                         inNoRSSI = 0, trans = "scale", alpha = 24)
```

where:

- data: The input data set with the original RSSI fingerprints.
- outRange: A numeric vector with two values indicating the desired range of the output data.
- outNoRSSI: The desired value for not detected beacons in the output data.
- inRange: A numeric vector with two values indicating the range of signal strength values in the input data. If this parameter is not provided, the function will infer it from the provided data.
- inNoRSSI: The value given to a not detected beacon in the original data.
- trans: The transformation to perform over the RSSI data, either 'scale' or 'exponential'.
- alpha: The α parameter for the exponential transformation.

The *scale* transformation scales the input data values to a range specified by the user. The feature scaling is performed according to Equation 1:

$$\rho_{h,i}^{out} = \begin{cases} a + b \cdot \rho_{h,i}^{in}, & \text{if } \rho_{h,i}^{in} \neq inNoRSSI \\ outNoRSSI, & otherwise \end{cases} \tag{1}$$

$$b = \frac{outMin - outMax}{inMin - inMax}$$
$$a = outMin - inMin \cdot b$$

where:

– $\rho_{h,i}^{out}$ and $\rho_{h,i}^{in}$ are the output and input RSSI values, respectively, for the h^{th} beacon from the i^{th} observation

– $outMax$ and $outMin$ are the maximum and minimum values, respectively, specified for the output by the outRange parameter.

– $inMax$ and $inMin$ are the maximum and minimum values, respectively, of the input data.

– $outNoRSSI$ and $inNoRSSI$ are the values assigned in the fingerprint to represent a not detected beacon for the output and input data, respectively, specified by the parameters outNoRSSI and inNoRSSI.

The *exponential* transformation (Torres-Sospedra et al., 2015b) changes the data according to the next equation:

$$\rho_{h,i}^{out} = \begin{cases} \exp(\frac{pos(\rho_{h,i}^{in})}{\alpha}), & \text{if } \rho_{h,i}^{in} \neq inNoRSSI \\ outNoRSSI, & otherwise \end{cases}$$

$$pos(\rho_{h,i}^{in}) = \begin{cases} \rho_{h,i}^{in} - inMin, & \text{if } \rho_{h,i}^{in} \neq inNoRSSI \\ 0, & otherwise \end{cases}$$

where α is a parameter for the exponential transformation. The authors establish α as a case-based parameter, and find that 24 is a good value for RSSI fingerprinting data, but they did not study the effects of α in the transformed data.

The following code scales the ipftrain and ipftest data sets RSSI data, stored in the columns 1:168, to a positive range of values, from 0 to 1, with NA representing a not detected WAP. As a not detected WAP is represented by a NA value in the original data, this has to be indicated to the function so it can transform these values to the desired output:

```
trainRSSI <- ipfTransform(ipftrain[, 1:168], outRange = c(0.1, 1), inNoRSSI = NA,
                          outNoRSSI = NA)
testRSSI <- ipfTransform(ipftest[, 1:168], outRange = c(0.1, 1), inNoRSSI = NA,
                         outNoRSSI = NA)
```

The `ipfTransform` function returns a new data set with the same structure (vector, matrix or data frame) as the input.

Positioning algorithms

This section describes three positioning algorithms implemented in the `ipft` package. The examples illustrating each description are based on the data previously scaled in Section Data wrangling .

The `ipfKnn` function.

The `ipfKnn` and `ipfEstimate` functions implement a version of the knn algorithm to select the k nearest neighbors (the k more similar vectors from the training set) to a given RSSI vector. Many different distance metrics (Torres-Sospedra et al., 2015b) can be used to compare two RSSI vectors and measure how 'near' or similar they are.

The distance metrics implemented in the package include some typical functions, as the L^1 norm, or manhattan distance, or the L^2, or euclidean distance. The L^u norm between two fingerprints with indices a and b is defined as follows:

$$L^u = \left(\sum_{h=1}^{q} |(\rho_{h,a} - \rho_{h,b}|^u\right)^{1/u}$$

The package also implements some fingerprinting specific distance estimation functions such as LDG and PLGD. The LGD between two RSSI vectors λ_i^{tr} and λ_j^{ts} of longitude q is given by:

$$LGD(\lambda_i^{tr}, \lambda_j^{ts}) = -\sum_{h=1}^{q} \log\ \max(G(\rho_{h,i}^{tr},\ \rho_{h,j}^{ts}),\ \epsilon)$$

where ϵ is a parameter to avoid logarithm of zero, as well as having one beacon RSSI value influence the LGD only above a certain threshold. $G(\rho_{h,i}^{tr},\ \rho_{h,j}^{ts})$ represents the Gaussian similarity between $\rho_{h,i}^{tr}$ and $\rho_{h,j}^{ts}$, defined as

$$G(\rho_{h,i}^{tr}, \rho_{h,j}^{ts}) = \begin{cases} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\rho_{h,i}^{tr} - \rho_{h,j}^{ts})^2}{2\sigma^2}\right), & \text{if } \rho_{h,i}^{tr} \neq 0 \text{ and } \rho_{h,j}^{ts} \neq 0 \\ 0, & otherwise \end{cases}$$

The σ^2 parameter represents the shadowing variance (Shrestha et al., 2013). Values for σ in the range between 4 and 10 dBm are usually good for indoor scenarios (Lohan et al., 2014).

The PLGD between two RSSI vectors λ_i^{tr} and λ_j^{ts} of longitude q is given as:

$$PLGD(\lambda_i^{tr}, \lambda_j^{ts}) = LGD(\lambda_i^{tr}, \lambda_j^{ts}) + \alpha(\phi(\lambda_i^{tr}, \lambda_j^{ts}) + \phi(\lambda_j^{ts}, \lambda_i^{tr}))$$

where $\phi(\lambda_i^{tr}, \lambda_j^{ts})$ is a penalty function for the beacons that are visible in the i^{th} training fingerprint but not in the j^{th} test fingerprint, $\phi(\lambda_j^{ts}, \lambda_i^{tr})$ is a penalty function for the beacons that are visible in the j^{th} test fingerprint but not in the i^{th} training fingerprint, and are defined as follows:

$$\phi(\lambda_i^{tr}, \lambda_j^{ts}) = \sum_{h=1}^{q} T_{max} - \rho_{h,i}^{tr},\ \text{for}\ \ 0 < \rho_{h,i}^{tr} \leq T_{max} \text{ and } r_i = 0)$$

$$\phi(\lambda_j^{ts}, \lambda_i^{tr}) = \sum_{h=1}^{q} T_{max} - \rho_{h,j}^{ts},\ \text{for}\ \ 0 < \rho_{h,j}^{ts} \leq T_{max} \text{ and } r_j = 0)$$

T_{max} is an upper threshold for the strength of the signal, and α is a scaling factor.

The similarity measurement method can be chosen by means of the parameter `method`, or by providing a custom function (parameters `FUN` and `...`). The signature of the `ipfKnn` function is:

```
ipfKnn <- function(train_fgp, train_pos, k = 3, method = 'euclidean',
                     weights = 'distance', norm = 2, sd = 5, epsilon = 1e-3,
                     alpha = 1, threshold = 20, FUN = NULL, ...)
```

where:

- `train_fgp`: A data frame of n rows and q columns containing the fingerprint vectors of the training set.
- `train_pos`: A data frame of n rows and l columns containing the positions of the training observations.
- `k`: The k parameter of the *knn* algorithm, the number of nearest neighbors to consider.
- `method`: The distance metric to be used by the algorithm. The implemented options are 'euclidean', 'manhatan', 'norm', 'LGD' and 'PLGD'
- `weights`: The weight function to be used by the algorithm. The implemented options are 'distance' and 'uniform'. The default 'distance' function calculate the weights from the distances as:

  $$w_{j,t} = \frac{1}{(1 + d_{j,t})\mathcal{W}_j}$$

  where $w_{j,t}$ is the weight assigned to the t^{th} ($t \in [1..k]$) neighbor of the j^{th} ($j \in [1..m]$) test observation, $d_{j,t}$ is the distance in the feature (RSSI) space between the t^{th} neighbor and the j^{th} test fingerprint, and $\mathcal{W}_j$ is a term used to normalize the values so that the total sum of the k weights is 1.

  The 'uniform' function assigns the same weight value to each neighbor:

  $$w_{j,t} = \frac{1}{k}$$

- `norm,sd,epsilon,alpha,threshold`: Parameters for the 'norm', 'LGD' and 'PLGD' methods.
- `FUN`: An alternative function provided by the user to compute the distance.
- `...`: Additional parameters for the function FUN.

For a training data set of n RSSI vectors (a data frame or a matrix named `tr_fingerprints`) and a data set of n position vectors (a data frame or a matrix named `tr_positions`), the code for fitting a knn model with a k value of 4 and the manhattan distance as the distance measurement method is:

```
knnModel <- ipfKnn(tr_fingerprints, tr_positions, k = 4, method = 'manhattan')
```

This function returns an S3 object of class `ipftModel` containing the following properties:

- `params`: A list with the parameters passed to the function.
- `data`: A list with the fingerprints and the location data of the radio map.

To estimate the position of a new fingerprint, the `ipfEstimate` function makes use of the previously obtained model. An `ipfModel` object holds the data model needed by the `ipfEstimate` function to apply the selected algorithm and returns an estimation of the test fingerprints positions. The signature of `ipfEstimate` is:

```
ipfEstimate <- function(ipfmodel, test_fgp, test_pos = NULL)
```

where:

- `ipfmodel`: An S3 object of class `ipfModel`.
- `test_fgp`: A data frame of m rows and q columns containing the fingerprints of the test set.
- `test_pos`: An optional parameter containing a data frame of m rows and l columns with the position of the test observations.

The `ipfEstimate` function returns an S3 object of the class `ipfEstimation` with the following elements:

- `location`: A $m \times l$ matrix with the predicted position for each observation in the `test` data set.
- `errors`: If the actual location of the test observations is passed in parameter `test_pos`, and the data that represents the position is numeric, this property returns a numeric vector of length n with the errors, calculated as the *euclidean* distances between the actual and the predicted locations.

- `confusion`: If the actual location of the test observations is passed in parameter `test_pos`, and the data that represents the position is a factor, the estimation of the actual position is performed as a classification task, and this property returns a confusion matrix summarizing the results of this classification.
- `neighbors`: A $m \times k$ matrix with the indices of the k selected neighbors for each observation in the `test` data set.
- `weights`: A $m \times k$ matrix containing the weights assigned by the algorithm to the selected neighbors.

The following `R` code shows an example of the usage of the `ipfKnn` function with the data set included in the package. This example takes the data previously scaled and generates a positioning model from the input data `trainRSSI` (the radio map) that is stored in `knnModel`. Then, the model is passed to the `ipfEstimate` function, along with the test data, to get an estimation of the position of the 702 test observations:

```
tr_fingerprints <- trainRSSI[, 1:168]
tr_positions    <- ipftrain[, 169:170]
knnModel        <- ipfKnn(tr_fingerprints, tr_positions, k = 7, method = "euclidean")
ts_fingerprints <- testRSSI[, 1:168]
ts_positions    <- ipftest[, 169:170]
knnEstimation   <- ipfEstimate(knnModel, ts_fingerprints, ts_positions)
```

Since the position of the test observations is known, the mean error for the 702 test observations can be calculated as follows:

```
> mean(knnEstimation$errors)
[1] 3.302739
```

The mean positioning error is one of the most common evaluation metrics used in indoor positioning (Liu et al., 2007) to assess the system's accuracy. This metric corresponds to the average Euclidean distance between the estimated locations and the true locations. As positions in the `ipftrain` and `ipftest` are expressed in meters, this metric represents the average error in meters for this scenario.

The neighbors selected from the training data set for the 6 first test fingerprints are:

```
> head(knnEstimation$neighbors)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7]
[1,]   71  176  126  125  127  771  130
[2,]   71  176  126  125  127  771  130
[3,]  465  914  915  913  217   77  218
[4,]  465  914  915  176  913  461  217
[5,]  176  126  125  771  130  127  914
[6,]   77  914  915  217  176  465  218
```

where each row of the output corresponds to the indices of the $k = 7$ more similar vectors from the training data set to the i^{th} vector of the `test` data set.

As an example of how to use `ipfKnn` with a custom function, the next code shows the definition of a *C++* function that implements a modified version of the manhattan distance. The function needs at least two parameters, the two matrices representing the training and test data sets. A third parameter is here introduced to represent a penalization value. This function penalizes the computed distance between two RSSI measurements when one of the beacons is not detected (represented by the value $\emptyset$), by multiplying the resulting distance by a factor F. Given two fingerprints λ_i^{tr} and λ_j^{ts} of length q, the *myD* distance is:

$$\text{myD}(\lambda_i^{tr}, \lambda_j^{ts}) = \sum_{h=1}^{q} \text{myd}(\rho_{h,i}^{tr}, \rho_{h,j}^{ts}),$$

where

$$\text{myd}(\rho_{h,i}^{tr}, \rho_{h,j}^{ts}) = \begin{cases} |\rho_{h,i}^{tr} - \rho_{h,j}^{ts}|, & \text{if}\, \rho_{h,i}^{tr} \neq \emptyset \,\text{and}\, \rho_{h,j}^{ts} \neq \emptyset \\ |\rho_{h,i}^{tr} - \rho_{h,j}^{ts}|F, & otherwise \end{cases}$$

The following code implements the `myD` function and shows an example of its usage with `ipfKnn`, as well as the results obtained. The function is coded in *C++* to improve its performance when using large data sets, although the method also accepts custom plain R functions. The myD function assumes that the fingerprints are in a positive range:

```
library('ipft')
library('Rcpp')
cppFunction('
  NumericMatrix myD(NumericMatrix train, NumericMatrix test, double F = 2.0) {
    NumericMatrix distanceMatrix(test.nrow(), train.nrow());
    double d = 0, pv = 0, rssi1 = 0, rssi2 = 0;
    for (int itrain = 0; itrain < train.nrow(); itrain++) {
      for (int itest = 0; itest < test.nrow(); itest++) {
        d = 0;
        for (int i = 0; i < train.ncol(); i++) {
          rssi1 = R_IsNA(train(itrain, i))? 0 : train(itrain, i);
          rssi2 = R_IsNA(test(itest, i))? 0 : test(itest, i);
          pv = (rssi1 != 0 && rssi2 != 0)? 1 : F;
          d = d + std::abs(rssi1 - rssi2) * pv;
        }
        distanceMatrix(itest, itrain) = d;
      }
    }
    return distanceMatrix;
  }'
)
customModel      <- ipfKnn(tr_fingerprints, tr_positions, k = 1, FUN = myD, F = 0.25)
customEstimation <- ipfEstimate(customModel, ts_fingerprints, ts_positions)

> head(customEstimation$neighbors)
     [,1]
[1,]  773
[2,]  773
[3,]  776
[4,]  773
[5,]  130
[6,]  130
```

The previous code outputs the selected neighbors for the first 6 observations in the test data set. As the ts_positions data frame contains the actual location of the observations, the absolute error committed by the model is returned in the ipfEstimation object:

```
> head(customEstimation$errors)
[1] 5.708275 5.708275 5.708275 5.708275 3.380000 3.380000

And the mean error with this custom similarity function is:

> mean(customEstimation$errors)
[1] 3.297342
```

An ipfEstimation object can be used directly to plot the Empirical cumulative distribution function of the error (function ipfPlotEcdf()) and the Probability density function (function ipfPlotPdf()). Figures 1 and 2 show the plots obtained from the following code:

```
> ipfPlotEcdf(customEstimation)
> ipfPlotPdf(customEstimation)
```

The plotting functions included in the package are described in detail in Section Plotting functions.

The ipfProbabilistic function.

Given the limitations of sensors accuracy (Luo and Zhan, 2014) and the irregular character of signal propagation (Ali et al., 2010), the RSSI vector stored for a particular position cannot have completely reliable and accurate information about the emitters signal strength. This uncertainty is generally modeled by a normal distribution (Haeberlen et al., 2004), but to do so many readings of the signals at the same position are needed to obtain a representative set of statistical parameters to model each RSSI present at that position.

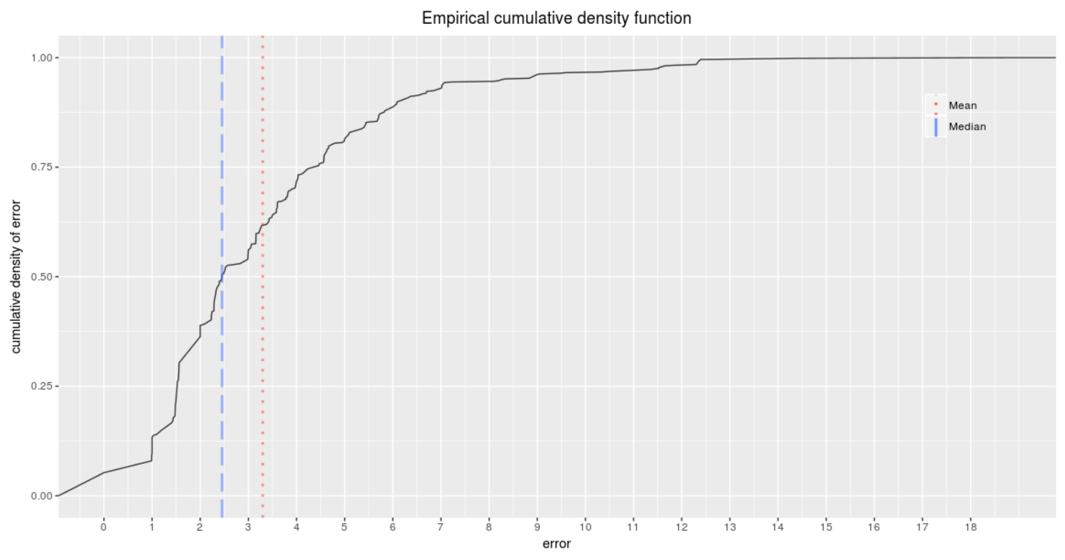

Figure 2: Funtion `ipfPlotEcdf`. Empirical cumulative distribution function of the error. The plot also shows the mean (red dotted line) and the median (blue dashed line) of the errors.

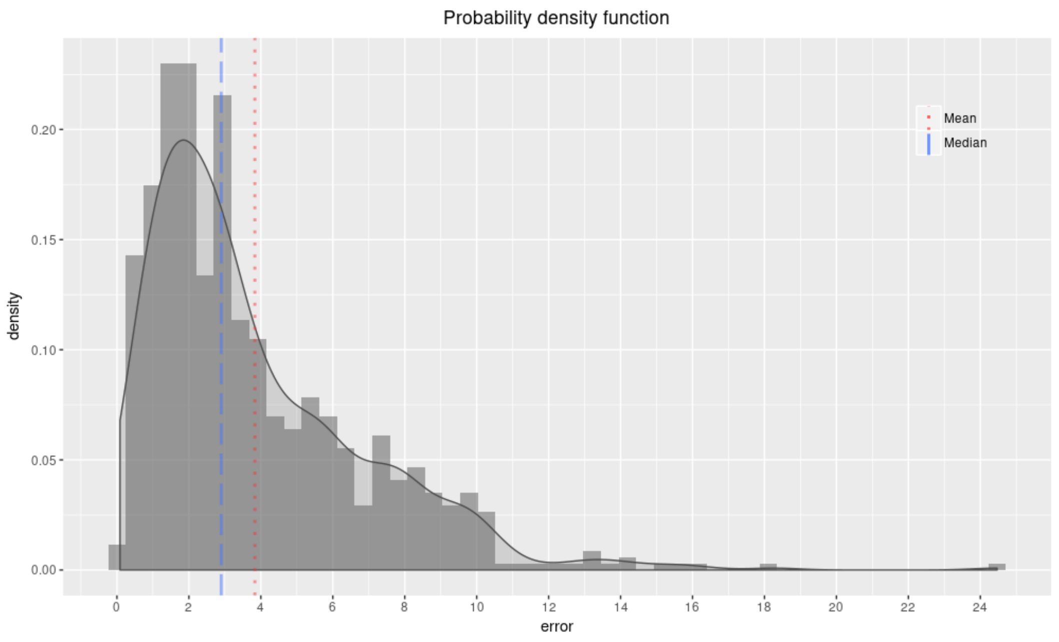

Figure 3: Funtion `ipfPlotPdf`. Probability density function. The plot shows the normalized histogram of the errors and its density function. The plot also shows the mean (red dotted line) and the median (blue dashed line) of the errors.

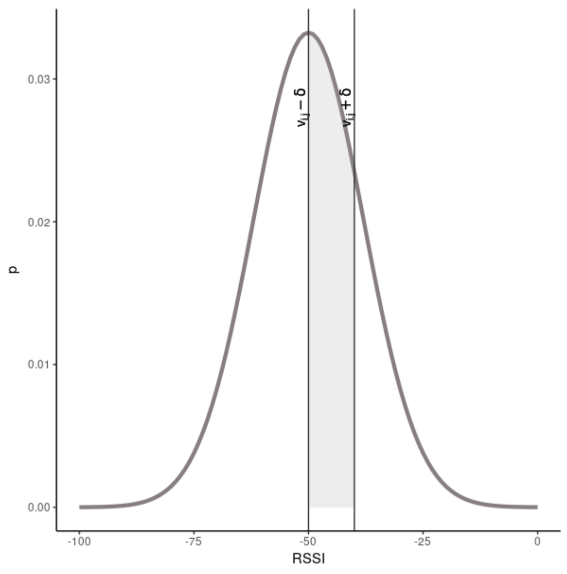

Figure 4: δ parameter for the probabilistic approach. This parameter sets the width of the discretization steps.

Thus, the initial collection of RSSI observations associated to a particular point is transformed into a pair of vectors containing the means and the standard deviations of the RSSI for each beacon, and then the complete training data is stored as a set of statistical parameters that can be used to infer the location of a test observation as the one that maximizes a probability function.

Let $\widehat{\mathcal{D}}_{train}$ be the new training set obtained from the previous procedure:

$$\widehat{\mathcal{D}}_{train} = \left\{\widehat{\mathcal{F}}_{train},\ \widehat{\mathcal{L}}_{train}\right\}$$

$$\widehat{\mathcal{F}}_{train} = \left\{\widehat{\lambda_1^{tr}}, \widehat{\lambda_2^{tr}}, ..., \widehat{\lambda_g^{tr}}\right\}$$

$$\widehat{\mathcal{L}}_{train} = \left\{\widehat{\tau_1^{tr}}, \widehat{\tau_2^{tr}}, ..., \widehat{\tau_g^{tr}}\right\}$$

where $\widehat{\mathcal{F}}_{train}$ is the set of statistical parameters obtained from the fingerprints of the training set, g is the number of groups of fingerprints with the same associated position, and $\widehat{\mathcal{L}}_{train}$ is the set of positions associated to each group. Each one of the g observations of the training data set is now composed of a fingerprint with q values:

$$\widehat{\lambda_i^{tr}} = \left\{\theta_{1,i}^{tr}, \theta_{2,i}^{tr}, ..., \theta_{q,i}^{tr}\right\},\ i \in [1, ..., g]$$

$$\theta_{h,i}^{tr} \sim \mathcal{N}(\mu_{h,i}, \sigma_{h,i}^2)$$

where $\mu_{h,i}$ and $\sigma_{h,i}^2$ are the mean and the variance, respectively, of the h^{th} RSSI of the i^{th} group of original fingerprints.

Let $\rho_{h,j}^{ts}$ be the h^{th} RSSI measurement of the j^{th} test fingerprint (λ_j^{ts}), and let $\mu_{h,i}$ and $\sigma_{h,i}^2$ be the mean and the standard deviation of the h^{th} beacon distribution obtained for the i^{th} position from the training set. The probability $p_{h,j}^{(i)}$, of observing $\rho_{h,j}^{ts}$ at the i^{th} position is:

$$p_{h,j}^{(i)} = \int_{\rho_{h,j}^{ts}-\delta}^{\rho_{h,j}^{ts}+\delta} \frac{1}{\sigma_{h,i}\sqrt{2\pi}}\, \mathrm{e}^{-\frac{x-\mu_{h,i}}{2\sigma_{h,i}^2}}\ dx$$

where δ is a parameter to allow the discretization of the normal distribution (Figure 4).

The set of all probabilities $p_{h,j}^{(i)},\ h \in [1, ..., q]$ obtained for a given test observation j, expresses the similarity between the observation measurement and the training data for a particular location. An evaluation of the total similarity for every location can be computed as a function of these individual probabilities, like its sum or its product. In the **ipft** package, this algorithm is implemented by the ipfProbabilistic and ipfEstimate functions, and by default uses the sum of probabilities as default operator to evaluate the similarity:

$$\psi_j^{(i)} = \sum_{h=1}^{p} p_{h,j}^{(i)}$$

where $\psi_j^{(i)}$ is the similarity between the j^{th} test observation and the i^{th} distribution from the training data set. The function to evaluate the similarity can be passed to ipfProbabilistic as a parameter.

As well as the ipfKnn and ipfProximity funtions, ipfProbabilistic returns a ipfModel object with the same data structure seen in Section The ipfKnn function, but with the difference that now the data property returns the probabilistic parameters that define the fitted distributions for every group of fingerprints on the training set. The clustering or grouping of the training data is performed by default over the location data provided by the user, but this behavior can be customized by passing a parameter with the columns over which to group the data, or by passing the group indices directly. The ipft package implements two functions (ipfGroup() and ipfCluster()) to perform clustering tasks. These functions are described in Section Data clustering.

The signature of the ipfProbabilistic function is:

```
ipfProbabilistic <- function(train_fgp, train_pos, group_cols = NULL, groups = NULL,
                             k = 3, FUN = sum, delta = 1, ...)
```

where train_fgp, train_pos and k have the same meaning and structure as described in Section The ipfKnn function, and, given n observations in the training set:

- groups: is a numeric vector of length n, containing the index of the group assigned to each observation of the training set. This parameter is optional.
- group_cols: is a character vector with the names of the columns to use as criteria to form groups of fingerprints. This parameter is optional.
- FUN: is a function to estimate a similarity measure from the calculated probabilities.
- delta: is a parameter to specify the interval around the test RSSI value to take into account when determining the probability.
- ...: are additional parameters for FUN.

The following code shows how to use the ipfProbabilistic function to obtain a probabilistic model from the ipftrain and ipftest data sets. The default behavior of ipfProbabilistic groups the training data attending at the position of each observation, in this case, its x and y coordinates:

```
> probModel <- ipfProbabilistic(tr_fingerprints, tr_positions, k = 7, delta = 10)
> head(probModel$data$positions)
     X     Y
1 -0.6 24.42
2 -0.6 27.42
3  0.0  0.00
4  0.4  0.00
5  0.4  3.38
6  0.4  6.81
```

Now the ipfModel$data property returns a list with 3 elements:

- means: a data frame with the means for every beacon and every group of fingerprints.
- sds: a data frame with the standard deviations for every beacon and every group of fingerprints.
- positions: a data frame with the position of each group of fingerprints.

To obtain an estimation from this model, the same code used in section The ipfKnn function can be used to produce the estimated locations:

```
> ts_fingerprints <- ipftest[, 1:168]
> ts_positions    <- ipftest[, 169:170]
> probEstimation  <- ipfEstimate(probModel, ts_fingerprints, ts_positions)
```

and their errors and its mean value:

```
> mean(probEstimation$errors)
[1] 6.069336
```

An alternative function can be passed to ipfProbabilistic. The following code uses the maximum value of the probabilities as the similarity measure, and passes a parameter to remove NAs from the data[7]:

```
> probModel      <- ipfProbabilistic(tr_fingerprints, tr_positions, k = 9, delta = 10,
+                                    FUN = max, na.rm = TRUE)
> probEstimation <- ipfEstimate(probModel, ts_fingerprints, ts_positions)
> mean(probEstimation$errors)
[1] 8.652321
```

The `ipfProximity` function.

When the location of the access points is known, it's possible to estimate the position of a fingerprint using the log-distance path loss model (Seybold, J.S., 2005). Given a set of q beacons, and a fingerprint vector $\lambda = \{\rho_1, \rho_2, ..., \rho_q\}$ of length q, this model is expressed as:

$$\rho_h = P_{1m,h} - 10\alpha \log_{10} d_h - \gamma, \ \ h \in [1, 2, ..., q]$$

where ρ_h is the value of the received signal from the h^{th} beacon, d_h is the distance from the observation to the beacon, $P_{1m,h}$ is the received power at 1 meter from the emitter, α is the path loss exponent, and $\gamma \sim \mathcal{N}(0, \sigma_\gamma^2)$ represents a zero mean Gaussian noise that models the random shadowing effects of the environment.

The estimator of the distance between the emitting beacon and the position where the signal is received is:

$$\hat{d}_h = 10^{\frac{\rho_h - P_{1m,h}}{10\alpha}}$$

This estimation follows a log-normal distribution that is:

$$\ln \hat{d}_h \sim \mathcal{N}(\ln d_h, \sigma_d^2)$$

where $\sigma_d = (\sigma_\gamma ln10)/(10\alpha)$.

The mean and the variance of the distribution are:

$$E[\hat{d}_h] = d_h \, e^{\sigma_d^2/2}$$
$$Var[\hat{d}_h] = d_h^2 \, e^{\sigma_d^2} \, (e^{\sigma_d^2} - 1)$$

Note that the variance grows quadratically with the distance, making the estimation less reliable as the distance becomes larger. Therefore, the distances estimated from different beacons will have different accuracies. To take this into account, the algorithm estimates the position of a fingerprint as a minimization problem of the overall squared error of the estimated distances. The objective function to minimize is:

$$\min_{\tau} J = \sum_{h=1}^{p} \omega_h (\hat{d}_h - \|s_h - \tau\|)^2$$

where τ is the position that minimizes the function, that is, the estimated position, q is the number of beacons present in the fingerprint, and $\omega_h = 1/Var[\hat{d}_h]$ are the weights.

The functions ipfProximity and ipfEstimate implement this design, and uses the Broyden-Fletcher-Goldfard-Shano algorithm (BFGS) (Broyden, 1969; Fletcher, 1970; Goldfarb, 1970; Shanno, 1970), a quasi-Newton method, to minimize the previous function to make an estimation of the fingerprint position. The accuracy of the estimation is strongly dependent on the reliability of the emitters positions. When these positions are unknown, they can be estimated with the function ipfEstimateBeaconPositions. Section Beacon position estimation details the implementation and usage of this function. The ipfProximity function returns an ipfModel object with the data needed by the ipfEstimate function to estimate a fingerprint position.

The signature of the ipfProximity function is:

```
ipfProximity <- function(bpos, rssirange = c(-100, 0), norssi = NA, alpha = 5,
```

[7]The ipfProbabilistic function takes into account the NAs contained in the data when using the default function (sum), but the user needs to manage this situation when a custom function is provided. In this example, the data is not previously transformed, is passed as it is, with NAs for not detected WAPs, to illustrate this situation.

```
                                wapPow1 = -30)
```

where:

- bpos: a matrix or a data frame containing the position of the beacons, in the same order as they appear in fingerprints.
- rssirange: the range of the RSSI data in the fingerprints.
- norssi: the value used to represent a not detected beacon.
- alpha: the path loss exponent (α).
- wapPow1: a numeric vector with the received power at one meter distance from the beacon ($P_{1m,h}$). If only one value is supplied, it will be assigned to all beacons.

In the following example, the goal is to estimate the position of the 702 fingerprints included in the test set, using the known position of the WAPs and the log-distance path loss model. The ipfpwap dataset contains the location of 39 of the 168 wireless access points of the ipftrain and ipftest data sets. The ipfProximity function returns a model that is used to estimate the position of the fingerprints. As the real position of the test fingerprints is known, this information can be also passed to the ipfEstimate function. Thus, the returned ipfEstimation object will contain, along with the estimated positions, the associated errors:

```
> proxModel      <- ipfProximity(ipfpwap, alpha = 4, rssirange = c(-100, 0),
+                               norssi = NA, wapPow1 = -32)
> fingerprints   <- ipftest[, 1:168]
> positions      <- ipftest[, 169:170]
> proxEstimation <- ipfEstimate(proxModel, ipftest[, 1:168], ipftest[, 169:170])
> mean(proxEstimation$errors)
[1] 8.0444
```

Positioning algorithms comparison

In a classical fingerprint-based positioning system, the radio map is constructed in accordance to the positioning algorithm to be used in the online phase. The knn algorithm follows a deterministic approach that performs well in most cases, while the probabilistic method is based on the assumption that there is enough training data for each particular position to obtain reliable parameters to model a distribution for each signal at each survey location. As regards to the proximity algorithm, it is based on two assumptions; first, the ability to realistically simulate the propagation model of the signal, and second, the known positions of the emitter beacons. These conditions are not met in many scenarios, where changes in occupation, for example, modify the propagation model and thus the performance of the positioning system.

To illustrate the previous considerations, Table 1 shows the mean and the quartile errors in meters for the implemented algorithms, computed using the dataset included in the package. In this particular case, given the characteristics of the training data, knn performs better than the rest.

		Quartile error (m)				
algorithm	mean error (m)	0%	25%	50%	75%	100%
knn	3.3027	0.15172	1.46891	2.61281	4.08992	19.84650
probabilistic	6.0693	0.14289	3.26988	5.63051	8.19933	17.93031
proximity	8.0444	2.49865	5.71055	7.42602	9.88427	20.12029

Table 1: Comparison of the algorithms' accuracy on the dataset included in the package

To compare the performance of the proposed implementation of the previous positioning algorithms, we ran a benchmark test of 1000 iterations on each function, using the dataset included in the package. The results for the model fitting functions are shown in Table 2. As it can be seen, the proximity and knn algorithms are the fastest, as expected, since their model fitting process basically consists in storing the training data for later processing during the estimation stage. In contrast, the probabilistic algorith has to fit a normal distribution for each signal received at each position, and thus, it takes longer to complete the process.

The outcomes are different when considering the results for the estimation function (Table 3). The position estimation for the probabilistic algorithm is faster that the rest. For the knn algorithm, the estimation process could be improved using clustering techniques to avoid comparing the test fingerprint with all the instances in the training set. With regards to the estimation process for the

function	elapsed (sec)	relative
ipfKnn	0.031	1.409
ipfProbabilistic	1035.446	47065.727
ipfProximity	0.022	1.000

Table 2: Performance comparison of the model building functions

proximity algorithm, the fact that the result is computed by solving an unconstrained nonlinear optimization through an iterative method highly penalyzes its performance.

model	function	elapsed (sec)	relative
knn	ipfEstimate	2508.079	2.998
probabilistic	ipfEstimate	836.651	1.000
proximity	ipfEstimate	28259.110	33.776

Table 3: Performance comparison of the estimation functions on each model

Beacon position estimation

If the actual position of the beacons is unknown, it can be estimated in many ways from the RSSI data. Two basic methods for estimation of the beacons location have been included in the `ipft` package through the `ipfEstimateBeaconPositions` function. The 'centroid' and the 'weighted centroid' methods.

Both methods use the fingerprint data to guess the position of the beacons. Let q be the number of beacons and $\tau^{\mathcal{B}}$ be the set of beacons locations:

$$\tau^{\mathcal{B}} = \left\{ \nu_{1,h}^{\mathcal{B}}, \nu_{2,h}^{\mathcal{B}}, \nu_{3,h}^{\mathcal{B}} \right\},\ h \in [1, 2, ..., q]$$

the position of the h^{th} beacon is given by:

$$\tau_h^{\mathcal{B}} = \left\{ \sum_{i=1}^{n} \omega_i \nu_{1,i}^{tr}, \sum_{i=1}^{n} \omega_i \nu_{2,i}^{tr}, \sum_{i=1}^{n} \omega_i \nu_{3,i}^{tr} \right\}$$

where n is the number of fingerprints in the training set. The value of ω_i is:

$$\omega_i = \frac{1}{n}$$

for the 'centroid' method and:

$$\omega_i = \frac{\rho_{h,i}^{tr}}{\sum_{l=1}^{n} \rho_{h,l}^{tr}}$$

for the 'weighted centroid' method. Since the biggest weights have to be assigned to the strongest RSSI values, the fingerprint vector values should be positive, or at least, positively correlated to the beacon received intensity. This is checked by the function implementation so the input data is internally transformed to a positive range when needed.

This is the signature of the `ipfEstimateBeaconPositions` function:

```
ipfEstimateBeaconPositions <- function(fingerprints, positions, method = 'wcentroid',
                                       rssirange = c(-100, 0), norssi = NA)
```

where:

- `fingerprints`: is a data frame with the fingerprint vectors as rows.
- `positions`: a data frame with the position of the fingerprints.
- `method`: the method to use by the algorithm, either 'centroid' or 'wcentroid'.
- `rssirange`: the range of the signal strength values of the fingerprints.
- `norssi`: the value assigned in the fingerprints to a non detected beacon.

The following code uses the function `ipfEstimateBeaconPositions` with the 'weighted centroid' method to estimate the position of the wireless access points, under the assumption that this position is unknown. Finally, the function `ipfProximity` estimates the positions of the first 6 test fingerprints:

```
> bc_positions   <- ipfEstimateBeaconPositions(ts_fingerprints, ts_positions,
                                           method = 'wcentroid')
> proxModel      <- ipfProximity(bc_positions, rssirange = c(0.1, 1),
+                               norssi = NA)
> proxEstimation <- ipfEstimate(proxModel, fingerprints[1:6,],
+                              positions[1:6,])
> proxEstimation$location
        V1       V2
1 1.686950 12.02117
2 1.686950 12.02117
3 1.654255 10.91767
4 1.682121 10.96035
5 1.711448 10.88966
6 1.695007 10.09507
```

Data clustering

Clustering techniques can be used with the aim of enhancing localization performance and reducing computational overhead (Cramariuc et al., 2016b). The `ipft` package includes some functions for cluster analysis and grouping of the fingerprinting and location data. These functions can be used to create or detect clusters based on the position of the observations, on its signal levels, or on any other criteria that might be useful to group the data by. Performing RSSI clustering before the positioning process groups a large number of reference points into various clusters that can be used to perform first-level classification. This allows to assess the testing point location by using only the fingerprints in the matched cluster rather than the whole radio map. Furthermore, given the amplitude atenuation that building partitions cause to electromagnetic signals, clusters usually can be related to physical spaces such as buildings, floors or even rooms.

The main function for clustering tasks is `ipfCluster`. The more basic usage of the function takes the provided data and uses the k-means algorithm to classify it into k disjoint sets of observations, by selecting a set of k cluster centers to minimize the sum of the squared distances between the data vectors and their corresponding centers.

The k-means clustering procedure begins with an initial set of randomly selected centers, and iteratively tries to minimize the sum of the squared distances. This makes the algorithm very sensitive to the arbitrary selection of initial centers, and introduces variability in the results obtained from one execution to another. Besides, the number of clusters has to be established beforehand, and that may be inconvenient in some scenarios.

The signature of the `ipftCluster` function is:

```
ipfCluster <- function(data, method = 'k-means', k = NULL, grid = NULL, ...)
```

where

- `data`: is a data frame with the data to cluster. When using the *k-means* method, the data frame must not contain any `NA` values.
- `method`: the algorithm used to create clusters. The implemented algorithms are 'k-means' for k-means algorithm, 'grid' for clustering based on spatial grid partition, and 'AP' for affinity propagation algorithm.
- `k`: a numeric parameter for k-means algorithm.
- `grid`: a numeric vector with the size of the grid for the grid algorithm.

When using the default k-means algorithm, the function behaves as a wrapper around the k-means function of the **stats** package, and therefore, the usage can be further customized by passing extra parameters, as the number of iterations or the algorithm to be used ("Hartigan-Wong" is the default).

The following example will find $k = 30$ clusters of similar fingerprints in the `ipftrain` dataset. First the data set of fingerprints is transformed to eliminate the `NA` values that represent a not detected beacon. Then, the data is passed to the `ipfCluster` function to find the 30 clusters using the 'MacQueen' algorithm:

```
> set.seed(1)
> cl_fingerprints <- ipfTransform(tr_fingerprints, inNoRSSI = NA, outNoRSSI = 0)
> clusterData     <- ipfCluster(cl_fingerprints, k = 30, iter.max = 20,
+                                algorithm = "MacQueen")
> head(clusterData$clusters)
[1] 3 3 3 3 3 3
```

The outcome of the ipfCluster function is a list containing the indices of the k clusters and its centroids. Given the previous example, clusterData$centers will return the k centroids, and clusterData$clusters will return the cluster index $i \in [1, .., k]$ for every observation in ipftrain.

The ipfCluster function includes an implementation of the affinity propagation (AP) algorithm (Frey and Dueck, 2007) that can be used to estimate the number of distinct clusters present in the radio map. AP does not require the number of clusters to be determined before running it. It finds members of the input set, known as 'exemplars', that are representative of clusters by creating the centers and the corresponding clusters based on the constant exchanging of reading similarities between the observations. This message-passing process continues until a good set of centers and corresponding clusters emerges.

The following code uses AP clustering to find groups of similar RSSI vectors from the ipftrain data set. With no further parametrization, it will classify the RSSI data into 43 distinct clusters:

```
> clusterData     <- ipfCluster(tr_fingerprints, method = 'AP')
> dim(clusterData$centers)
[1]  43 168
```

Now, clusterData$centers holds the 43 'exemplars', those RSSI vectors from the radio map that are representative of a cluster, and clusterData$clusters contains the indices that link every observation of the data set with its assigned cluster.

To perform a more simple grouping based on a precise set of variables, the ipfGroup function provides a method to group the data by column name. The function signature is:

```
ipfGroup <- function(data, ...)
```

where

- data: is a data frame with the data to group.
- ...: The variables to group the data by.

The ipfGroup function returns a numeric vector with the same length as the number of observations contained in the data data frame, containing the index of the group assigned to each observation. The following example groups the data according to the position of the observations, that in the ipftrain and ipftest datasets are represented by the columns 'X' and 'Y':

```
> groups <- ipfGroup(ipftrain, X, Y)
> head(groups)
[1]  4  4  4  4 22 22
> length(unique(groups))
[1]  41
```

Plotting functions

Indoor positioning generally involves statistical analysis of datasets, and the **ipft** provides some useful functions to produce graphs for exploring data. All the graphic functions included in the package are built upon the **ggplot2** package (Wickham, 2011), and return a ggplot object that can be plotted or further personalized with custom labels, theme, etc.

The ipfPlotPdf and the ipfPlotEcdf have already been introduced in Section The ipfKnn function. These functions will plot the probability density function and the empirical cumulative distribution function, respectively. Both functions take an ipfEstimation object to produce the plot, while the axis labels and plot tittle can be also supplied by the parameters xlab, ylab and tittle. Their respective signatures are:

```
ipfPlotPdf <- function(estimation, xlab = 'error', ylab = 'density',
                       title = 'Probability density function')

ipfPlotEcdf <- function(estimation, xlab = 'error',
                        ylab = 'cumulative density of error',
                        title = 'Empirical cumulative density function')
```

The function `ipfPlotLocation` will produce a plot of the location of the data. The following code shows its signature and presents an example of its use. The example calls the function with parameter `plabel` set to `TRUE`, to plot labels identifying each location, and `reverseAxis` set to `TRUE` to swap the axis. It also modifies the resulting object by changing the default **ggplot2** theme to the white one. The result is shown in Figure 5.

```
ipfPlotLocation <- function(positions, plabel = FALSE, reverseAxis = FALSE,
                            xlab = NULL, ylab = NULL, title = '')

library(ggplot2)
ipfPlotLocation(ipftrain[, 169:170], plabel = TRUE, reverseAxis = TRUE) + theme_bw()
```

The function `ipfPlotEstimation` plots the estimated position of the test observations based on an `ipfModel` object and an `ipfEstimation` object, as well as the actual position (parameter `testpos`), if known, and the position of the k selected fingerprints from the training set used to guess its location (parameter `showneighbors`). The green dots indicate the actual position of the observations, while the black dots indicate the estimated ones. The blue lines connect the estimated positions with the k neighbors from which the location has been estimated, and the red arrows connect the actual position of the fingerprint with the estimated one. The following code shows the function signature and provides an example of its usage. The result plot is shown in Figure 6:

```
ipfPlotEstimation <- function(model, estimation, testpos = NULL, observations = c(1),
                              reverseAxis = FALSE, showneighbors = FALSE,
                              showLabels = FALSE, xlab = NULL, ylab = NULL,
                              title = '')

library(ggplot2)
probModel <- ipfProbabilistic(ipftrain[, 1:168], ipftrain[, 169:170])
probEst   <- ipfEstimate(probModel, ipftest[, 1:168], ipftest[, 169:170])
ipfPlotEstimation(probModel, probEst, ipftest[, 169:170],
                  observations = c(61:62, 81:82), reverseAxis = TRUE,
                  showneighbors = TRUE, showLabels = TRUE) + theme_bw()
```

Summary

In this paper, the package **ipft** is presented. The main goal of the package is to provide researchers with a set of functions to manipulate, cluster, transform, create models and make estimations using indoor localization fingerprinting data. This package enables researchers to use a well established set of algorithms and tools to manipulate and model RSSI fingerprint data sets, and also allows them to customize the included algorithms with personalized parameters and functions to adapt the working mode to their particular research interests.

In this work some of the fundamental algorithms used in indoor fingerprinting localization techniques have been formally presented and illustrated, while detailed examples and information about its usage and implementation have been provided.

Future work

This package is an ongoing work, and future versions will implement new algorithms and tools with the aim of providing a base framework for researchers, and become a reference library for fingerprinting-based indoor positioning research.

In particular, future lines of work should consider the implementation of deep learning based algorithms. Many deep learning techniques can be exploited to try to obtain better positioning performance. Recurrent neural networks could be used to learn not only spatial but also temporal patterns of the received signals. Deep autoencoders can be implemented as a way to encode fingerprints and reduce their dimensionality to a few number of significant features. Their variational and generative extensions can be of use to better model the stochastic nature of RSSI data. These models can also be applied to generate new training data for deep learning-based clasisifiers, increasing the robustness of positioning systems and trying to address problems caused by heterogeneity of devices.

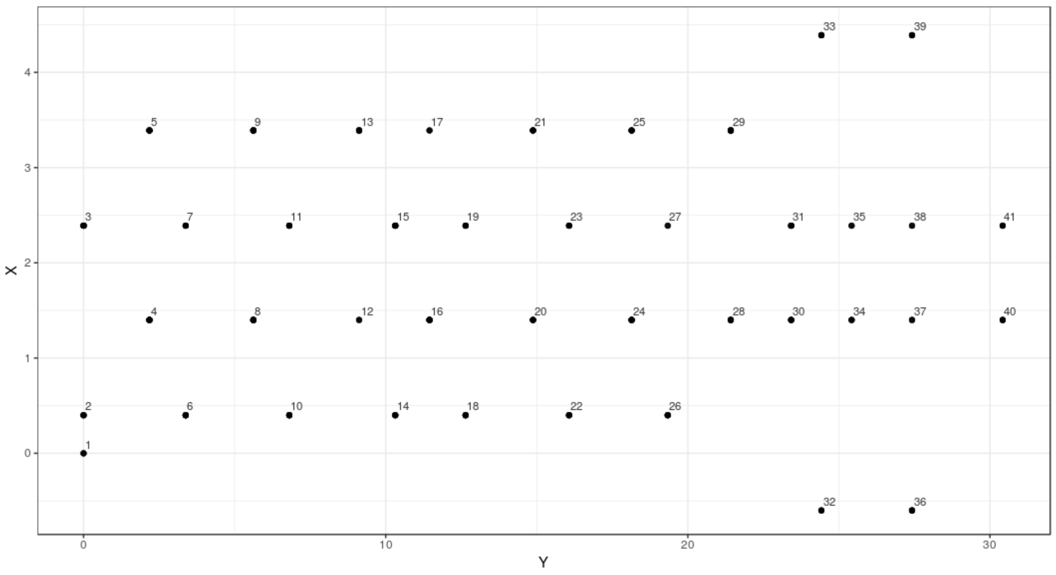

Figure 5: Location of fingerprints included in the `ipftrain` data frame. The labels indicate the group indices.

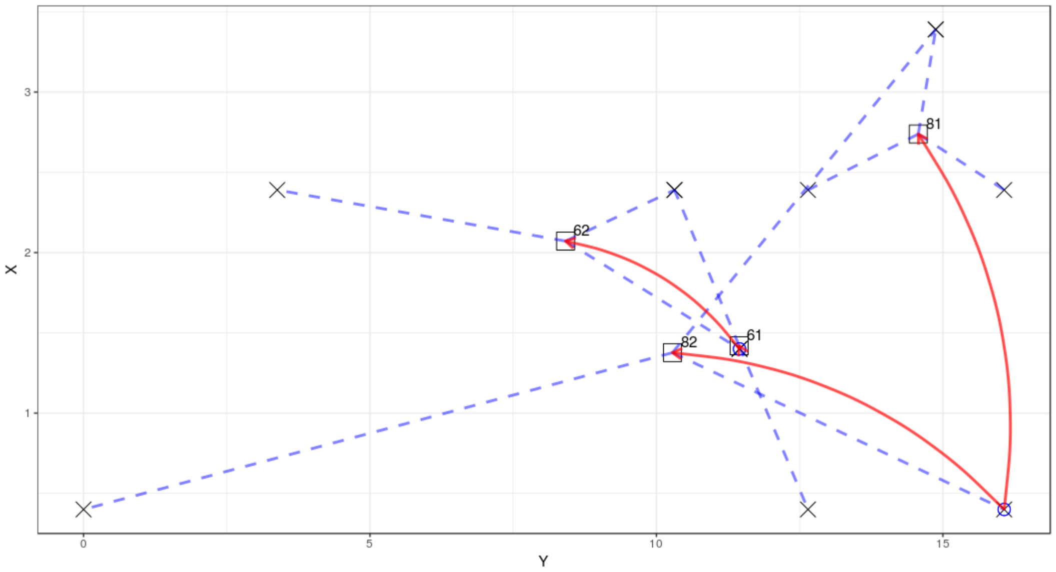

Figure 6: Estimated and actual positions of test observations 61, 62, 81 and 82 from the `ipftrain` data set. The circles indicate the actual positions of the observations. The squares show the estimated positions. The red arrows connect the actual positions with the estimated ones. The dashed lines connect the estimated positions with the *k* neighbors from which the location has been estimated, represented by the crosses.

Acknowledgements

The authors would like to thank the two anonymous reviewers for providing useful feedback that helped to improve the paper.

This work has been partially funded by the Spanish Ministry of Economy and Competitiveness through the "Proyectos I + D Excelencia" programme (TIN2015-70202-P) and by Jaume I University "Research promotion plan 2017" programme (UJI-B2017-45).

Bibliography

E. Aarts and R. Wichert. Ambient intelligence. In *Technology Guide*, pages 244–249. Springer-Verlag, 2009. URL https://doi.org/10.1007/978-3-540-88546-7_47. [p]

A. H. Ali, M. R. A. Razak, M. Hidayab, S. A. Azman, M. Z. M. Jasmin, and M. A. Zainol. Investigation of Indoor WIFI Radio Signal Propagation. In *Proceedings of the Symposium on Industrial Electronics and Applications, (ISIEA'10)*, pages 117–119, 2010. URL https://doi.org/10.1109/isiea.2010.5679486. [p]

C. Broyden. A new double-rank minimisation algorithm. preliminary report. In *Notices of the American Mathematical Society*, volume 16, page 670. American Mathematical Society 201 Charles ST, Providence, RI 02940-2213, 1969. [p]

T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. URL https://doi.org/10.1109/tit.1967.1053964. [p]

A. Cramariuc, H. Huttunen, and E. S. Lohan. Clustering benefits in mobile-centric wifi positioning in multi-floor buildings. In *2016 International Conference on Localization and GNSS (ICL-GNSS)*, pages 1–6. IEEE, 2016a. URL https://doi.org/10.1109/icl-gnss.2016.7533846. [p]

A. Cramariuc, H. Huttunen, and E. S. Lohan. Clustering Benefits in Mobile-Centric WiFi Positioning in Multi-Floor Buildings. In *Proceedings of the 6th International Conference on Localization and GNSS (ICL-GNSS'16)*, pages 1–6, 2016b. URL https://doi.org/10.1109/icl-gnss.2016.7533846. [p]

R. Fletcher. A new approach to variable metric algorithms. *The computer journal*, 13(3):317–322, 1970. [p]

B. J. Frey and D. Dueck. Clustering by Passing Messages between Data Points. *Science*, 315(5814): 972–976, 2007. URL https://doi.org/10.1126/science.1136800. [p]

T. Gigl, G. J. M. Janssen, V. Dizdarevic, K. Witrisal, and Z. Irahhauten. Analysis of a uwb indoor positioning system based on received signal strength. In *Proceedings of the 4th Workshop on Positioning, Navigation and Communication (PNC'07)*, pages 97–101, 2007. URL https://doi.org/10.1109/wpnc.2007.353618. [p]

D. Goldfarb. A family of variable-metric methods derived by variational means. *Mathematics of Computation*, 24(109):23–26, 1970. URL https://doi.org/10.1090/s0025-5718-1970-0258249-6. [p]

A. Haeberlen, E. Flannery, A. M. Ladd, A. Rudys, D. S. Wallach, and L. E. Kavraki. Practical robust localization over large-scale 802.11 wireless networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MobiCom'04)*, pages 70–84, 2004. URL https://doi.org/10.1145/1023720.1023728. [p]

A. B. Harbicht, T. Castro-Santos, W. R. Ardren, D. Gorsky, and D. J. Fraser. Novel, continuous monitoring of fine-scale movement using fixed-position radiotelemetry arrays and random forest location fingerprinting. *Methods in Ecology and Evolution*, 8(7):850–859, 2017. URL https://doi.org/10.1111/2041-210x.12745. [p]

S. He and S. H. G. Chan. Wi-Fi Fingerprint-Based Indoor Positioning: Recent Advances and Comparisons. *IEEE Communications Surveys & Tutorials*, 18(1):466–490, 2016. URL https://doi.org/10.1109/comst.2015.2464084. [p]

N. E. Klepeis, W. C. Nelson, W. R. Ott, J. P. Robinson, A. M. Tsang, P. Switzer, J. V. Behar, S. C. Hern, and W. H. Engelmann. The national human activity pattern survey (nhaps): a resource for assessing exposure to environmental pollutants. *Journal Of Exposure Analysis And Environmental Epidemiology*, 11:231 EP –, 2001. URL https://doi.org/10.1038/sj.jea.7500165. [p]