Choosing Machine Learning Algorithms for Anomaly Detection in Smart Building IoT Scenarios

Fernando Almaguer-Angeles, John Murphy, Liam Murphy, and A. Omar Portillo-Dominguez
School of Computer Science, University College Dublin, Ireland
fernando.almaguerangeles[at]ucdconnect.ie
{j.murphy, liam.murphy, andres.portillodominguez}[at]ucd.ie

***Abstract*—Internet of Things (IoT) systems produce large amounts of raw data in the form of log files. This raw data must then be processed to extract useful information. Machine Learning (ML) has proved to be an efficient technique for such tasks, but there are many different ML algorithms available, each suited to different types of scenarios. In this work, we compare the performance of 22 state-of-the-art supervised ML classification algorithms on different IoT datasets, when applied to the problem of anomaly detection. Our results show that there is no dominant solution, and that for each scenario, several candidate techniques perform similarly. Based on our results and a characterization of our datasets, we propose a recommendation framework which guides practitioners towards the subset of the 22 ML algorithms which is likely to perform best on their data.**

I. Introduction

The Internet has always been in evolution, from the World Wide Web, going through the Web 2.0 and branching into the Internet of Things (IoT) [1]. The first step of this evolution was characterized by linked HTML static documents. The novelty in the second part was a two-way communication, enabling users participation and the opportunity to interact and collaborate through blogs, social networking, wikis, among others. IoT is an Internet-based paradigm that will allow any common object the ability to sense and interact with one another or with services to accomplish an objective.

According to [2], [3] IoT was born right after mobile phones got the ability to sense and interact with one another, turning them into smartphones, and its explosive growth after 2007. Three years later in 2010, the number of connected devices to the Internet exceeded the current global population for the first time. On the one hand, [2] also forecast that by 2020 there will be an average of 6.58 connected devices per person or 50 billion devices around the globe. On the other hand, IoT data will reach 500 zettabytes by 2019 [4]. Consequently, IoT is gradually impacting every business area and industry field [5].

Due to the speed with which data is being generated, it is important to analyze it and extract useful information efficiently. For this, we used machine learning (ML), which is an efficient way to process great amounts of data. There are many different ML algorithms available, each suited to different types of scenarios. We focused on those that are able to distinguish between two categories - binary classification - which allows for anomaly detection. Furthermore, there are many different ML algorithms that can do anomaly detection, so the real question is, what technique to use in our data to extract useful information.

We analyzed 22 state-of-the-art supervised ML classification algorithms on different IoT smart-building scenarios. Our aim is to provide a recommendation framework which guides practitioners who have little or no ML expertise towards the subset of the 22 ML algorithms which is likely to perform best on their data.

The rest of the paper is organized as follows. In section II, we review the work related to the comparison of the performance of different ML algorithms. In section III we describe how we modified real-world smart building datasets to inject anomalies, and also our experimental scenario. Section IV details our results, and we discuss in section V how our recommendations depend on the characteristics of the data, and some possible future directions for this work.

II. Related Work

There are different studies about how to measure and compare the performance of ML classification algorithms. In [6], the authors are comparing the area under the ROC (Receiver Operating Characteristics) curve or auROC with accuracy. Their motivation was to give a formal argument to why auROC should be preferred over accuracy. Through their results, they confirm empirically that auROC is statistically consistent and more discriminating measure than accuracy.

A similar analysis is carried out in [7], where the authors compare the relationship between Precision-Recall (P-R) and auROC curves. They talk about what cases auROC and P-R curves are good for. Also, they mention why using accuracy results can be misleading. Their results, among other things, showed that there is an equivalence between the auROC and P-R curves, where a curve dominates in auROC space if and only if it dominates in P-R space, which means that auROC can be represented in a P-R curve, but not the other way around.

In [8], the authors assess several metrics for the quality of classification performance. They made families of metrics by clustering the relationships between measures. Their results showed that auROC is genuinely different and a compact measure. While in [9], although they are using auROC as one of several metrics, they do not use it as the main decision-making metric. Instead, they used *Classification Rate* and *Cohens Kappa* to make their decision.

In [10], the authors do not consider one metric to choose the best ML algorithm for a scenario, what they do is to consider several metrics and model the output as a multiple criteria decision making. Among the metrics they used are: accuracy, true positive rate, true negative rate, mean absolute error (MAE), precision, F-measure, auROC, kappa statistic, and computation time.

The research done in [11] was to assess the impact of training ML with imbalanced datasets (DS), and all their results were compared with the auROC metric.

In terms of anomaly detection, in [12] the authors analyzed an autonomous vehicle system to detect anomalies, with the help of a ML algorithm, to ensure the secure control between vehicles and instruments. Their results showed that a neighbor comparison and historical comparison are useful to predict anomalies within their proposed algorithm. Moreover, in [13] the authors evaluated some techniques suitable to identify real-time anomalies within an IoT network with the aim of offering practitioners a reference about when such techniques might be more appropriate.

III. Methodology

In this section we are going to describe our approach, specifically the experimental environment setup, the evaluation, the DS characteristics and the rules to inject anomalies, as well as the framework to choose a subset of the ML algorithms that is likely to perform best (in terms of anomaly detection) on the data.

A. Experimental environment setup

All the experiments were performed in an isolated test environment so, the server resources were fully dedicated to the experiments. This environment was composed of a server Dell Precision T5500 with an: Intel Xeon E5-620 CPU at 2.40GHz; 4 cores; 8 siblings, 25GB of RAM, 500GB of HDD at 7200 rpm; 6Gb/s, GPU NVIDIA GF108GL [Quadro 600]; clock at 33MHz; width: 64 bits; size=128, running Linux Ubuntu 18.04 LTS, and Python 3.6. The library installation was made with command *pip3 install packageName −−user*. Also, the resources (time, CPU, RAM) measurements were made with standard Linux command lines.

B. Evaluation

We evaluated all the 22 ML supervised classification algorithms provides by the scikit-learn python library [14], using their default settings [6], [9], [15], [16]. The only exception was the NearestNeighbors.radiusNeighborsClassifier algorithm, where we used the *outlier_label=1* option. We selected this library because it is freely accessible and highly-used in the industry.

To try to improve the classification process, or reduce the resources consumption, like CPU, RAM, and execution time. We used two feature selection (FS): low variance (LV), and tree-based (TB), from the same library as the ML, and same settings (default). It is worth mentioning that, because of the FS nature, they did not always produce an output as can be seen in Tables I and II.

Table I
Tracking

DS	Records	Anomalies	Attributes	LV	TB
CMU	31,780	8.48%	34	33	8
NCSU	267,685	8.29%	4	-	2
CU	623,207	32.51%	11	-	-

The metric to compare the classification performance from the different algorithms is the area under the Receiver Operating Characteristic (ROC) curve or auROC. This metric provides a single measure of a ML's performance [6], [8].

C. Datasets characteristics and anomalies injection

As we were unable to find smart building datasets with anomalies (binary DS), we analyzed real-world DS to create comprehensive rules that can mimic the behavior of a real scenario and thus add anomalies. We split the DS in two categories:

1) Tracking: These datasets are characterized by the position, recorded by sensors, of users in a place. The specifics of each DS like the number of records, anomalies, attributes, and the attributes reduction after the FS, can be found in Table I.

cmu/supermarket [17]: It has 34 attributes, of which 30 are magnetometers readings, the other ones are a magnetometer ID and the X, Y, and Z coordinates.

The injection of anomalies is considered when a user has a position with a negative X or Z coordinates.

ncsu/mobilitymodels [18]: The DS contains five folders (two university campi [NCSU and KAIST], New York City, Disney World [Orlando], and North Carolina state fair), inside these folders are several files that track specific users at that site, e.g. for the *kaist* site we have 92 files that represent 92 users. These files are composed of three attributes, a timestamp, and X, Y coordinates.

To inject an anomaly, we are considering the identification (ID) of the user to be a prime number and the user must have negative X and Y coordinates. In the case of *New York*, we are considering the not prime numbers and the same coordinates.

Because this is a classification task, we are not going to use the time-series column of this DS.

cu/rssi [19]: This DS consists of three files, each one containing 11 attributes, the first two attributes are IDs, the third one can be either *dev1* or *dev2*, the fourth one can be either *down*, *left*, *right*, *up* or *0.8*, *1.5*, *2.5*, *3.5*, *3.8*, *23*, the fifth could be another identifier and can be either something between *10 - 20* or be *-16* or *16*, the remaining six are the measurements of the sensors.

After analyzing the data, we consider it an anomaly when the position in column four is *up* and the readings from the sensors are negative, from column seven to 11.

2) Coexisting Time: In these DS, a data point is two users in the same space with a duration. The specifics of each DS like the number of records, anomalies, attributes, and the attributes reduction after the FS, can be found in Table II.

upmc/content [20]: Different folders have several files which represent, with an ID, either a fixed location or a student. Within these files, there is an ID, that represents an interaction with that user, and two timestamps, the first one represents the start of the interaction between this ID (this registry) and the files' ID (a location or a student), the second timestamp represents the end of the interaction.

We defined a threshold for the interactions like:

Pub:	4	hours
ShopWindow:	20	minutes
SuperMarket:	2	hours
CommercialCenter:	3	hours
CollegePorter:	20	minutes
LabReception:	20	minutes
Students:	4	hours

If any of these thresholds is exceeded, then an anomaly is added.

tecnalia/humanet [21]: It has seven attributes, the first two are users IDs, the third and fourth ones are the time and date when the interaction started, in the same way, the fifth and sixth ones say when the interaction ended, the seventh one represents the state of the device and can be either *horizontal*, *vertical and static* or *vertical and moving*.

We considered someone that is in the status *vertical and static* and who remained for more than 10 seconds in that position as an anomaly.

unimi/pmtr [22]: It has only one file that contains as first attribute an ID, followed by a second ID, a starting timestamp and an ending timestamp, for a total of four attributes.

To add an anomaly, we look for a prime number in the second column, and if the interaction lasted for more than 120 seconds, we consider it an anomaly.

upb/mobility2011 [23]: It has a file called *interactions.dat.txt* that contains as first attribute an ID, then a second ID, a starting timestamp, an ending timestamp, and two other attributes.

Whenever the difference between the second ID and the first one is a prime number, and the interaction persisted for more than five seconds, we consider it an anomaly.

upb/hyccups [24]: It has a file called *full_output.txt* that contains an ID, a second ID, a timestamp, and another attribute which we are using as the time of the interaction.

For the anomaly injection, we used this criterion: If the first ID times the second ID is a prime number, and the interaction lasted for more than 281,000 seconds, we consider it an anomaly.

copelabs/usense [25]: The DS contains nine folders that represent nine users. Each folder has a file called *Social-Strength.dat*. Within this file there are interactions with other users and they are characterized by a time stamp, an ID of another user, an encounter duration, an average encounter duration, and two more attributes.

If the result of adding the IDs is a prime number, and the encounter duration lasted for more than 60 seconds, we consider it an anomaly.

Table II
COEXISTING TIME

DS	Records	Anomalies	Attributes	LV	TB
UPMC	41,587	0.89%	4	-	2
TECNALIA	1,000	39.86%	5	-	3
UNIMI	11,895	12.94%	4	-	3
mobility2011	1,463	11.62%	6	-	4
hyccups	8,427	4.55%	4	-	-
COPELABS	755,772	6.61%	6	-	2

D. Framework

The procedure we followed was, for each DS, apply to it the different FS mentioned in III-B. In this manner, we produce several DS to analyze: the one created in section III-C (which we will refer to as the pre-processing DS from now on) and the outputs from the FS. Later we divided each dataset into training and testing. We refer to an **experiment** as training a particular ML algorithm with the training DS and analyzing the testing DS. In this manner, we can iterate over the ML algorithms with this set (training, testing) of DS. Once we have analyzed every ML with the current DS, we can start processing the next DS.

The ML training was made by using two-thirds of the anomalous data points in the DS and the same amount of regular data points, so we can have a balanced training DS [6], [11]. We also used a three-fold cross-validation [6].

After each experiment, we collect the true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) (which are how well the algorithm classified the information) to calculate the auROC metric with this formula [11]:

$$auROC = \frac{1 + recall - FalsePositiveRate}{2} \tag{1}$$

Where *recall* is the percentage of positive instances correctly classified. Is defined as follow:

$$recall = \frac{TP}{TP + FN} \tag{2}$$

and the *false positive rate* is the percentage of negative instances misclassified. Is defined as follow:

$$FalsePositiveRate = \frac{FP}{FP + TN} \tag{3}$$

IV. RESULTS

As it can be seen in Table III, the column DS contains the different tracking datasets, the FS represents the feature selection mentioned in III-B; the only exception is the N that represents the pre-processing DS generated in section III-C, the remaining columns are the names and the results of the auROC for the different ML algorithms. In the same manner, we can see the different coexisting time DS in Table IV. Where TEC represents the *tecnalia* DS, COPEL the *copelabs* DS, HYC the *upb/hyccups*, and MOBIL the *upb/mobility2011* datasets. We will see later in this section why we are not showing the results for all the 22 ML algorithms.

Table III
TRACKING RESULTS

DS	FS	Bagging	Extra trees	Gradient boosting	Random forest	Decision tree
CMU	LV	**0.9996**	0.9936	0.9994	0.9960	0.9994
	N	1	0.9978	1	0.9980	1
	TB	0.9925	**0.9931**	0.9926	0.9924	0.9860
NCSU	N	**0.9979**	0.9974	0.9858	0.9974	0.9977
	TB	0.9911	0.9914	0.9742	0.9913	**0.9916**
CU	N	1	1	1	1	1

Table IV
COEXISTING TIME RESULTS

DS	FS	Bagging	Gradient boosting	Random forest	Decision tree
UPMC	N	0.8028	**0.8205**	0.7943	0.7660
	TB	0.6827	0.6730	0.6674	**0.6833**
UNIMI	N	0.8065	**0.8536**	0.7872	0.8018
	TB	0.7556	**0.7608**	0.7330	0.7536
TEC	N	0.7119	**0.7478**	0.7331	0.6881
	TB	0.6730	**0.6974**	0.6787	0.6612
COPEL	N	**0.9998**	0.9894	0.9994	0.9998
	TB	0.9843	0.9757	0.9818	**0.9856**
HYC	N	0.9905	**0.9974**	0.9898	0.9858
MOBIL	N	**0.6435**	0.6399	0.6114	0.5848
	TB	0.5851	**0.6237**	0.6008	0.5816

Table V
TRACKING: CPU AND MEMORY AVERAGE CONSUMPTION

ML	CPU%	Memory%	Average	Rank
Bagging	10.72	1.44	6.08	3
Extra trees	11.89	1.43	6.66	5
Gradient boosting	11.42	1.40	6.41	4
Random forest	9.30	1.41	5.36	1
Decision tree	9.99	1.65	5.82	2

The Table V represents the average CPU, and memory among the experiments shown in Table III, e.g. the line showing the bagging ML algorithm with a 10.72 CPU% and 1.44% memory consumption is the average among the different tracking DS (CMU, NCSU, and CU) and the FS output for the bagging algorithm. The same processing is present in the memory column. The average column is the average between the two previous ones. We used this last column to create a ranking of algorithms in the rank column, where the number one represents the first place, the number two is the second best, and so on.

We show in Table VI the same kind of ranking analysis as in Table V. The difference is this table represents time instead of CPU or memory, i.e. the column seconds shows the average time consumption among the different DS (CMU, NCSU, CU, including the FS) per algorithm. The column rank shows a ranking, where the number one represents the first place, the number two is the second best, and so on.

Table VI
TRACKING: AVERAGE TIME

ML	Seconds	Rank
Bagging	8.71	4
Extra trees	8.62	3
Gradient boosting	10.62	5
Random forest	8.56	2
Decision tree	7.35	1

It is worth noting that the difference in the resources consumption among the pre-processing DS and the FS datasets. We saw no significant resource improvement - in the CPU, memory, or execution time - contradicting our initial expectations. Therefore, no resources improvement, neither in the CPU, memory, or execution time, as we initially thought in section III-B.

After having conducted the experiments and having collected all the auROC results, we noticed that for the CU DS there was a group of ML algorithms that performed perfectly, with a 100% measure of auROC, as we can see in Table III. Therefore, it is natural to recommend all these ML algorithms for DS with CU characteristics, and it is why we are not showing all the results.

This result led us to look for ML algorithms that performed well at all DS (pre-processing and the output from the FS). This search showed that the same ML algorithms that perform perfectly with the CU dataset, also performed well at all the tracking datasets, as can be seen in Table III. Thus, it is very likely that any of these ML algorithms will have a good performance at DS with similar characteristics as the tracking datasets we had analyzed in this paper. Additionally, if a practitioner is interested in the amount of resources these algorithms can consume, he or she should also consider Tables V and VI.

We tried to perform the same analysis for the coexisting time DS, but the results were not as even as with the tracking DS, as we can see in Table IV. Consequently we changed to a heuristic approach that allowed us to tell the best algorithms at all DS. We did this by gradually decreasing the auROC acceptance until the rows from the different DS started to be filled up. The results showed a similar list as in the tracking DS, with the exception of the extra trees algorithm. For this reason, the advice is to start working with these ML algorithms for DS with coexisting time characteristics. Furthermore, because of the wide auROC measurements, we will not perform a resources consumption analyzes, due to an unfair comparison across big differences in auROC scores.

V. CONCLUSIONS AND FUTURE WORK

As IoT systems produce ever-growing amounts of raw data, it is primordial to extract useful information from it. Finding an efficient manner to extract information is not a trivial problem, is time-consuming, and many machine learning methods exist for different scenarios. We proposed a recommendation framework for IoT smart building anomaly detection (where

the anomalies are defined as users in undesired situations), that gives practitioners a set of techniques best suited for their data. This framework has shown that for datasets that are characterized by the position of users in a place, the bagging, extra trees, gradient boosting, random forest, and decision tree classification algorithms are very likely to perform very well according to the area under the Receiver Operating Characteristics curve metric. Furthermore, once a practitioner is aware of these particular ML algorithms, he or she can start to tune the algorithm to increase the classification performance in their data.

As future work, we will confirm and extend our results, with new datasets and exploring new scenarios like other feature selection and other datasets types.

ACKNOWLEDGMENT

This work was supported, in part, by Science Foundation Ireland grant 13/RC/2094 and co-funded under the European Regional Development Fund through the Southern & Eastern Regional Operational Programme to Lero - the Irish Software Research Centre (www.lero.ie).

REFERENCES

[1] A. Whitmore, A. Agarwal, and L. Da Xu, "The Internet of ThingsA survey of topics and trends," *Information Systems Frontiers*, vol. 17, no. 2, pp. 261–274, 2015.

[2] D. Evans, "How the Next Evolution of the Internet Is Changing Everything," *Cisco Internet Business Solutions Group (IBSG)*, no. April, 2011. [Online]. Available: http://115.112.165.74:81/KrishnaAkalamkam/digitalmarketing/articles/TheInternetofThings{_}.pdf

[3] A. O. Portillo-Dominguez and V. Ayala-Rivera, "A requirements-based approach for the evaluation of emulated iot systems," in *2018 4th International Workshop on Requirements Engineering for Self-Adaptive, Collaborative, and Cyber Physical Systems (RESACS)*. IEEE, 2018, pp. 16–19.

[4] L. Lyu, J. Jin, S. Rajasegarar, X. He, and M. Palaniswami, "Fog-empowered anomaly detection in IoT using hyperellipsoidal clustering," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1174–1184, 2017.

[5] S. Brady, A. Hava, P. Perry, J. Murphy, D. Magoni, and A. O. Portillo-Dominguez, "Towards an emulated iot test environment for anomaly detection using nemu," in *2017 Global Internet of Things Summit (GIoTS)*. IEEE, 2017, pp. 1–6.

[6] J. Huang and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, 2005.

[7] J. Davis and M. Goadrich, "The relationship between Precision-Recall and ROC curves," *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pp. 233–240, 2006. [Online]. Available: http://portal.acm.org/citation.cfm?doid=1143844.1143874

[8] C. Ferri, J. Hernández-Orallo, and R. Modroiu, "An experimental comparison of performance measures for classification," *Pattern Recognition Letters*, vol. 30, no. 1, pp. 27–38, 2009. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167865508002687

[9] A. Fernández, S. García, J. Luengo, E. Bernadó-Mansilla, and F. Herrera, "Genetics-based machine learning for rule induction: State of the art, taxonomy, and comparative study," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 6, pp. 913–941, 2010.

[10] G. KOU, Y. LU, Y. PENG, and Y. SHI, "Evaluation of Classification Algorithms Using Mcdm and Rank Correlation," *International Journal of Information Technology & Decision Making*, vol. 11, no. 01, pp. 197–225, 2012. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0219622012500095

[11] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Information Sciences*, vol. 250, pp. 113–141, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2013.07.007

[12] M. H. A. Wibowo, H. Guo, and W. L. Goh, "Detecting anomalies in metro systems," *IEEE World Forum on Internet of Things, WF-IoT 2018 - Proceedings*, vol. 2018-January, pp. 302–307, 2018.

[13] S. Brady, D. Magoni, J. Murphy, H. Assem, and A. O. Portillo-Dominguez, "Analysis of machine learning techniques for anomaly detection in the internet of things," in *2018 IEEE Latin American Conference on Computational Intelligence (LA-CCI)*. IEEE, 2018, pp. 1–6.

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[15] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.

[16] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data," *ACM SIGKDD Explorations Newsletter - Special issue on learning from imbalanced datasets*, vol. 6, no. 1, pp. 20–29, 2004. [Online]. Available: http://doi.acm.org/10.1145/1007730.1007735{%}5Cnhttp://dl.acm.org/ft{_}gateway.cfm?id=1007735{&}type=pdf

[17] A. Purohit, S. Pan, K. Chen, Z. Sun, and P. Zhang, "CRAWDAD dataset cmu/supermarket (v. 2014-05-27)," Downloaded from https://crawdad.org/cmu/supermarket/20140527, Jul. 2018.

[18] I. Rhee, M. Shin, S. Hong, K. Lee, S. Kim, and S. Chong, "CRAWDAD dataset ncsu/mobilitymodels (v. 2009-07-23)," Downloaded from https://crawdad.org/ncsu/mobilitymodels/20090723, Jul. 2018.

[19] K. Bauer, E. W. Anderson, D. McCoy, D. Grunwald, and D. C. Sicker, "CRAWDAD dataset cu/rssi (v. 2009-05-28)," Downloaded from https://crawdad.org/cu/rssi/20090528, Sep. 2018.

[20] J. Leguay, P. Hui, J. Crowcroft, J. Scott, A. Lindgren, and T. Friedman, "CRAWDAD dataset upmc/content (v. 2006-11-17)," Downloaded from https://crawdad.org/upmc/content/20061117, Sep. 2018.

[21] J. M. Cabero, V. Molina, I. Urteaga, F. Liberal, and J. L. Martin, "CRAWDAD dataset tecnalia/humanet (v. 2012-06-12)," Downloaded from https://crawdad.org/tecnalia/humanet/20120612, Sep. 2018.

[22] P. Meroni, S. Gaito, E. Pagani, and G. P. Rossi, "CRAWDAD dataset unimi/pmtr (v. 2008-12-01)," Downloaded from https://crawdad.org/unimi/pmtr/20081201, Sep. 2018.

[23] R. I. Ciobanu and C. Dobre, "CRAWDAD dataset upb/mobility2011 (v. 2012-06-18)," Downloaded from https://crawdad.org/upb/mobility2011/20120618, Oct. 2018.

[24] ——, "CRAWDAD dataset upb/hyccups (v. 2016-10-17)," Downloaded from https://crawdad.org/upb/hyccups/20161017, Oct. 2018.

[25] S. Firdose, L. Lopes, W. Moreira, R. Sofia, and P. Mendes, "CRAWDAD dataset copelabs/usense (v. 2017-01-27)," Downloaded from https://crawdad.org/copelabs/usense/20170127, Oct. 2018.