

When the Power of the Crowd Meets the Intelligence of the Middleware : The Mobile Phone Sensing Case

Yifan Du, Valerie Issarny, Francoise Sailhan

▶ **To cite this version:**

Yifan Du, Valerie Issarny, Francoise Sailhan. When the Power of the Crowd Meets the Intelligence of the Middleware : The Mobile Phone Sensing Case. Operating Systems Review, 2019, 53 (1), pp.85-90. 10.1145/3352020.3352033 . hal-02238256

HAL Id: hal-02238256
https://inria.hal.science/hal-02238256

Submitted on 1 Aug 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

When the Power of the Crowd Meets the Intelligence of the Middleware: The Mobile Phone Sensing Case

Yifan Du[1], Valérie Issarny[1], and Françoise Sailhan[2]

[1]Inria Paris, France, {firstname.lastname}@inria.fr
[2]CNAM Paris, France, {firstname.lastname}@cnam.fr

Abstract

The data gluttony of AI is well known: Data fuels the artificial intelligence. Technologies that help to gather the needed data are then essential, among which the IoT. However, the deployment of IoT solutions raises significant challenges, especially regarding the resource and financial costs at stake. It is our view that mobile crowdsensing, *aka* phone sensing, has a major role to play because it potentially contributes massive data at a relatively low cost. Still, crowdsensing is useless, and even harmful, if the contributed data are not properly analyzed. This paper surveys our work on the development of systems facing this challenge, which also illustrates the virtuous circles of AI. We specifically focus on how intelligent crowdsensing middleware leverages on-device machine learning to enhance the reported physical observations.
Keywords: Crowdsensing, Middleware, Online learning.

1 Crowdsensing, IoT, AI & Middleware

Crowdsensing *aka* Mobile Phone Sensing (MPS) is a promising approach to observe real-world phenomena at a very large scale. The many MPS applications that have emerged over the years illustrate well the added value: micro-blogging [25], mobile social networking [16], quantified selves [23], urban tomography [1], environmental monitoring [26], transportation [4], or dynamic indoor map construction [6] all benefit from MPS. It is our perspective that MPS has been and will continue generating drastic changes in the way we approach science in the years to come. The development of citizen science illustrates well the trend [18]. The AI dependence on big data is another reason why MPS is likely to continue to grow: MPS has the potential to contribute the needed massive data at a relatively low cost.

In general, MPS holds the promise of enhancing our knowledge of the physical world. That is, MPS supports the IoT vision with the additional benefit that it does not require the costly deployment of dedicated sensors. For illustration, we refer to our background experience with the Ambiciti/SoundCity solution (http://ambiciti.io/), which features a MPS application and cloud-based platform for monitoring the individual and collective exposure to environmental pollution, and particularly noise [8]. The development of Ambiciti started in 2014 to result in the first launch of the application with the support of the city of Paris in summer 2015[1]. We have then shown that the assimilation of MPS observations allows generating street-level noise pollution maps that enhance traditional simulated maps [20], provided the calibration of the application [21].

Nonetheless, we need to admit that the above is one side of the coin, which is, the much positive one. MPS comes with hurdles too, and the underlying system must beat them. The major challenge facing effective crowdsensing is certainly being able to collect data of sufficient quality, starting with the ability to characterize the provided observations. Still referring to our experience with the Ambiciti solution, the analyses of the noise data collected in Paris over a one-year period in 2015-16 [9], and then in 2017 [11], have both highlighted that less than 10% of the observations actually contribute to the assimilation of relevant knowledge. Of course, one may consider that 10% of huge is still a valuable source of data. However, this incurs a significant waste of computing and networking resources, from device to cloud, which is not sustainable. And, in the –not so exceptional– case where the MPS application attracts a few committed users, then the knowledge from the collected observations is not worth the spending. *Participatory* sensing allows enhancing the data quality [10] but results in much less data than the *opportunistic* approach.

Following, we posit the need for developing intelligent middleware to support opportunistic MPS. The intelligent middleware that collects the sensing data on the device must act beyond merely interfacing with the embedded/connected sensors to transfer the data to the cloud. The middleware must as far as possible enhance locally the quality of the observations, from calibration to contextualization. While calibration may be achieved through regression analysis [19], contextualization requires prediction. The intelligent mid-

[1] https://www.inria.fr/en/centre/paris/news/launch-of-soundcity-mobile-application

dleware must implement soft/virtual sensors (as opposed to hardware/physical sensors) that run on the user device to further analyze and mine the data provided by the ever growing set of embedded cheap sensors. However, there is not a single implementation for soft sensors because the set of embedded sensors differs from one phone to the other, and the characteristic of the environment impacts on the inference of the observations.

This paper discusses our recent contribution in the area of self-adaptive intelligent middleware for MPS, which implements soft sensors that contextualize the observations that are sent to the cloud for global analyses. The intelligent middleware leverages online machine learning so that learning the context of the observations adapts to the available base sensors and the sensing environment. Prior to the presentation of our intelligent middleware solution in Section 3, we first highlight the importance of carefully selecting the sensor data *–aka features–* that inform the context definition, while distinguishing the many dimensions of a context.

2 Machine Learning to the Rescue

The accurate monitoring of the physical environment through crowdsensing requires knowing the location of the contributed observations, but such contextualization is not sufficient. The mobility of the user impacts the quality of the quantitative observations that mobile crowdsensing gathers [15], which may be inferred from machine learning over motion sensor data [5]. Knowing whether the smartphone/sensor is in-/out-pocket, in-/out-door and under-/on-ground is also essential because the device needs to be in a position that enables -yet does not interfere with- sensing the physical characteristics of the surrounding [21]. The literature investigates separately the inference of each of these context elements, while they are all equally important. Moreover, the proposed solutions do not account for the diversity of the contributing devices.

The features that classify: Today's smartphones embed an increasing number of sensors that may serve to contextualize the observations that the crowdsensors gather. Although the list of relevant sensors varies from one phone to another, high-end phones may provide the following *features*: *Light density*, *Abstract proximity* (the distance from an object to the screen of the device), *Magnetic strength*, *Temperature* of the ambient air, *Pressure*, *Humidity*, *GPS accuracy*, *GSM RSSI* value, *Wifi raw RSSI*, and *Abstract RSSI level* (i.e., the overall signal quality). We specifically focus on eliciting the features that best contribute to classify the observation context with respect to: in-/out-pocket (M_{pocket}), in-/out-door (M_{door}), and under-/on-ground (M_{ground}).

We leverage the $DATASET_1$ data set, which provides us with the supporting ground truth, for the above selection. $DATASET_1$ assembles labeled sensor data from a Crosscall Trekker-X3 phone, covering all the candidate features for all the scenarios to be classified. All the environmental sensors and network modules were active during the data collection. $DATASET_1$ comprises $20k$ instances and the amount of labeled data for each class is uniform. Each instance has three user-encoded labels, which represent the ground truth result for the three classifications.

We assess the significance of each candidate feature using the following scoring metrics [2]. *Information gain* is the expected amount of gained information *aka* reduction of entropy. *Gain ratio* is a ratio of the information gain and the attribute's intrinsic information, which reduces the bias towards multi-valued features that occur in information gain. *Gini* is the inequality among values of a frequency distribution. *Chi2* (χ^2) is the dependency between the feature and the class as measured by the chi-square statistic. Finally, *ReliefF* is the ability of an attribute to distinguish between classes on similar data instances. Table 1 provides the significance of the features to M_{pocket}, M_{door}, and M_{ground} within $DATASET_1$.

Feature/Metric	Info. G.	G. Ratio	Gini	χ^2	ReliefF
In-/Out-Pocket classification M_{pocket}					
Light density	0.310	0.155	0.169	3758.434	0.034
Proximity	0.931	0.720	0.478	17776.819	0.329
Magnetic strength	0.024	0.012	0.016	405.081	0.011
Temperature	0.344	0.172	0.213	273.650	0.097
Pressure	0.063	0.032	0.042	298.334	0.036
Humidity	0.096	0.048	0.064	1276.633	0.076
GPS accuracy	0.057	0.042	0.037	165.818	0.001
GSM RSSI value	0.017	0.008	0.011	256.149	0.034
Wifi raw RSSI	0.073	0.069	0.040	682.765	0.030
Abstract RSSI level	0.027	0.017	0.017	382.020	0.058
In-/Out-Door classification M_{door}					
Light density	0.255	0.127	0.157	5041.293	0.050
Proximity	0.098	0.076	0.063	1427.619	0.038
Magnetic strength	0.167	0.084	0.093	2633.329	0.008
Temperature	0.228	0.114	0.125	50.212	0.070
Pressure	0.127	0.064	0.077	341.875	0.127
Humidity	0.045	0.022	0.029	0.343	0.094
GPS accuracy	0.974	0.715	0.482	9085.097	0.996
GSM RSSI value	0.738	0.370	0.384	11211.066	0.157
Wifi raw RSSI	0.320	0.180	0.148	437.694	0.133
Abstract RSSI level	0.794	0.493	0.416	15416.226	0.293
Under-/On-Ground classification M_{ground}					
Light density	0.102	0.050	0.061	1662.044	0.009
Proximity	0.078	0.060	0.051	1599.679	0.007
Magnetic strength	0.017	0.008	0.011	297.901	0.006
Temperature	0.485	0.243	0.255	1905.520	0.255
Pressure	0.376	0.188	0.202	6136.352	0.224
Humidity	0.276	0.138	0.142	2475.257	0.161
GPS accuracy	0.434	0.318	0.222	4182.467	0.528
GSM RSSI value	0.463	0.232	0.262	8031.071	0.143
Wifi raw RSSI	0.181	0.170	0.086	5594.195	0.103
Abstract RSSI level	0.547	0.340	0.296	11586.949	0.250

Table 1: The significance of features to the classifications.

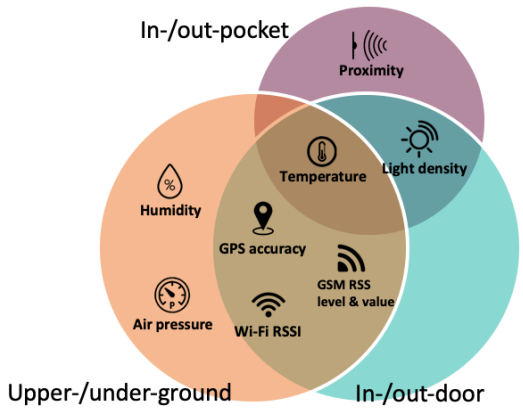
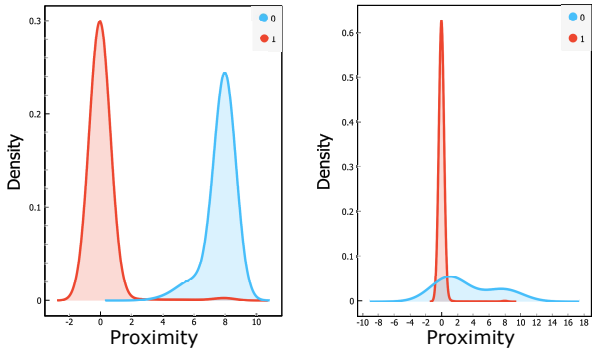


Figure 1: The features selection per classifier.



(a) 1 device contribution. (b) 3 devices contribution.

Figure 2: Distribution *wrt* In(1)/Out(0)-Pocket classification.

We note that the features that show a higher *information gain* and *gain ratio*, also show higher *Gini* and *ReliefF*. We then select the features that give both high *information gain* and high *gain ratio*, while we set the required threshold value to 0.1 for both. This leads to the selection depicted in Figure 1 for each of the three classifiers.

Training the classifiers: We may now train the three classifiers –M_{pocket}, M_{door}, and M_{ground}– using the most significant features associated with each of them. Using mixed training sources causes the risk of weakening the ability of classification due to the diversity of the feature values across devices and user behavior. As an illustration, Figure 2 shows the distribution of the proximity feature in the in/out-pocket scenarios in the case of a pure training set (2a: single user device) and a mixed training set (2b: three user devices), respectively. We observe that involving more devices and users in the training set creates interference between the distribution of the individual's features and results in blurring the features. The same phenomenon for light density has been observed in [13, 14]. Crucially, the single user-specific model outperforms the model trained on data pooled from several users [22]. Similarly, a model for each smartphone/sensor brand would provide a better classification. But, in practice, this is hardly feasible given the diversity of smartphones/sensors as it would require to perform training for each smartphone/sensor brand.Thus, we initialize our three classifiers, i.e., learning models, with $DATASET_1$ (20k entries) that we used for the feature selection.

Our classifiers must be effective both in terms of classification accuracy and time/space cost, especially with respect to their local update on the device. There are various algorithms eligible for the classification problem [17] although fewer are updatable. We have specifically selected six candidate updatable algorithms: *Hoeffding Tree –H.Tree* for short– (Very Fast Decision Tree), *IBk* (Instance Based K-nearest neighbors classifier), *KStar* (Instance-based Learner), *LWL* (Locally Weighted Learning), *updatable Naive Bayes* (*Naive-Bayes* for short), and *SGD* (Stochastic Gradient Descent) [24].

Metric/Model	H.Tree	IBk	KStar	LWL	NaiveBayes	SGD
M_{pocket}						
Size (kB)	16	1158	1157	1158	3	5
CVCA (%)	99.1538	99.469	99.350	99.149	98.999	99.180
OLR (ms)	0.020	3.809	0.081	4.344	0.012	0.123
IR (ms)	0.057	10.545	165.844	91.325	1.635	0.018
M_{door}						
Size (kB)	9	1612	1791	1764	4	6
CVCA (%)	100	100	100	100	100	100
OLR (ms)	0.036	5.150	0.062	5.813	0.011	0.172
IR (ms)	0.071	11.823	364.790	109.644	1.610	0.047
M_{ground}						
Size (kB)	13	1763	1763	1763	4	6
CVCA (%)	100	100	100	98.060	97.105	100
OLR (ms)	0.024	4.628	0.062	6.720	0.009	0.111
IR (ms)	0.061	15.160	238.916	128.149	1.223	0.018

Table 2: Initial learning models.

Table 2 compares the selected algorithms according to the same four metrics for our three classifications: M_{pocket}, M_{door} and M_{ground}. *Size* is the serialized model size of the initial classifier; it is an important metric due to the (relative) resource constraint of the mobile device and the fact that the size may increase as the model gets updated locally. *CVCA* (10-fold Cross Validation Classification Accuracy) characterizes the cross-validation split of the data into 10 folds where the learning model is tested by holding out examples from 1 fold at a time; the model is then induced from other 9 folds and examples from the 1 fold are classified and this is repeated for all the 10 folds. The classification accuracy is the proportion of correctly classified examples. *OLR* (Online Learning Runtime) indicates the time taken for updating a learning model with a fresh instance (i.e., user feedback). Finally, *IR* (Inference Runtime) indicates the time taken for carrying out an inference using an incoming feature vector.

The result for M_{pocket} in Table 2 shows that all the classifiers can provide a similar high *CVCA* of about 99%. However,

a significant difference appears among the sterilized sizes: *IBk*, *KStar* and *LWL* are storing training instances inside the learning model, which makes the size of the classifier proportional to the size of the training dataset. Instead, *H.Tree*, *NaiveBayes* and *SGD* require a much lower *Size*. *IBk* and *LWL* have an *OLR* greater than $3ms$, while it is less than $1ms$ for the other four. *IBk*, *KStar* and *LWL* all have much longer *IR* than *H.Tree*, *NaiveBayes* and *SGD*. A better cross validation result is discovered for M_{door}: All the algorithms provide the maximum classification accuracy in cross validation of 100%. However, although the dataset is unchanged compared to M_{pocket}, the serialized size of *IBk*, *KStar* and *LWL* increases due to the number of selected features. Besides, the *OLRs* do not change significantly, and are less than $1ms$ except for *IBk* and *LWL*. *IBk*, *KStar* and *LWL* still have a longer *IR* than the other three algorithms. Result for M_{ground} shows that *LWL* and *NaiveBayes* give lower classification accuracy in cross validation than other algorithms but still over 95%. The high storage cost remains for *IBk*, *KStar* and *LWL* as they require storing historical data. They further cost a much longer *IR* than the other three algorithms. As for *H.Tree*, *NaiveBayes* and *SGD*, both their *OLR* and *IR* remain below $1ms$, with the negligible exception of the *IR* of *NaiveBayes* at 1.223. Overall, *IBk*, *KStar* and *LWL* show the highest space and time costs, and we discard them. We specifically select *H.Tree* [7] that offers the highest accuracy and lowest space/time costs.

3 Empowering MPS Middleware with AI

SenseTogether is our middleware solution to opportunistic MPS, which is available at https://github.com/sensetogether/. We outline below the integration of our online learning approach to the contextualization of the contributed observations (see Figure 3), while detail may be found in [3].

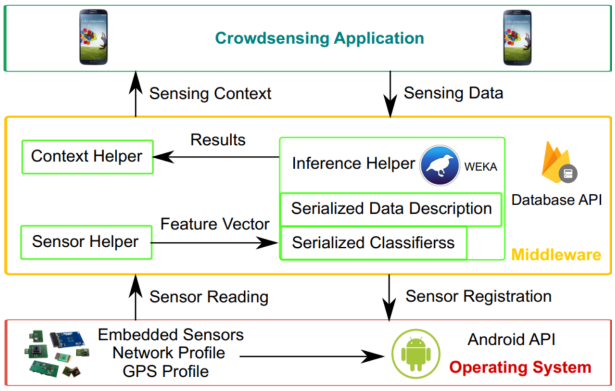


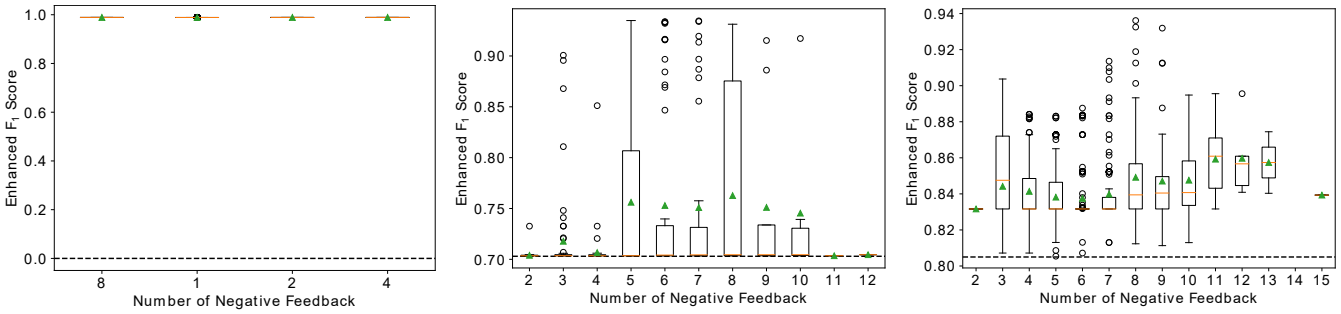
Figure 3: The *SenseTogether* middleware.

On-device learning: The initial classifiers are trained only once on a computer, and are deployed at the time of the installation of *SenseTogether*. Then, the *H.Tree* classifiers get updated across time so as to cope with: (i) the biases in the feature value across diverse device models, (ii) the difference in the availability of features depending on the device and user preferences, and (iii) the classification on new scenarios not covered during the initial training.

While the inference of the sensing context is running on the device, the middleware needs to collect the user feedback to assess the correctness of the inference result. The feedback is then converted to a labeled training instance that updates the current learning models. However, the requests for feedback should be limited as much as possible to minimize the burden on the user, while still enhancing the accuracy of the classifiers (in our case: M_{pocket}, M_{door} and M_{ground}) over time. We achieve this by applying a hierarchical inference and update of the three classifiers. The hierarchical algorithm follows from the predominant role of the in-pocket classifier over the two others, and of the in-door classifier over the under-ground one when sensing the physical environment. In more detail, a crowdsensed measurement is relevant for the analysis of most environmental phenomenon if out-pocket, while a in-pocket device has less opportunity to be contributing to the mobile crowdsensing. The in-door/out-door detection is meaningful only when the device is out-pocket and ready for sensing. Furthermore, the under-ground/on-ground case is a sub-scenario of the in-door situation. Also, while requesting the user's feedback about a single inference may be acceptable, requesting the feedback about three inferences is too much to ask. Practically, the *opportunistic* feedback from the user is collected using a permanent notification. The notification provides the user with information about the inferred context. Then, the user decides if and when to provide feedback upon incorrect inferences.

Performance evaluation: We evaluate our updatable approach using a new testing dataset $DATASET_2$. Similarly to $DATASET_1$, $DATASET_2$ contains $20k$ instances, each embedding three labels representing the ground truth, and covers all the relevant scenarios (i.e., in/out-pocket, in/out-door and under/on-ground) uniformly. Differently to $DATASET_1$, the environment sensors including temperature, humidity, pressure are not available on the contributing device Xiaomi Redmi Note 4, and the available sensors are from a distinct manufacturer. In addition, the user switches off the Wifi module from time to time. Furthermore, the data gathered for $DATASET_1$ and $DATASET_2$ correspond to two different physical environments as they were collected in two different city areas and at different months.

The F_1 score is a measure of a test accuracy considering both the precision and the recall of the test. The F_1 score ranges from 1 (best) to 0 (worst). We assess the F_1 score of the hierarchical algorithm according to the number of occurrences of negative user feedback (i.e., when the inference is wrong). We performed 500 experiments where the initial



(a) M_{pocket} (b) M_{door} (c) M_{ground}

Figure 4: Enhancement of the F_1 score according to the number of (hierarchical and negative) feedback occurrences.



Figure 5: Performance on an Android phone - red dots represent user feedback.

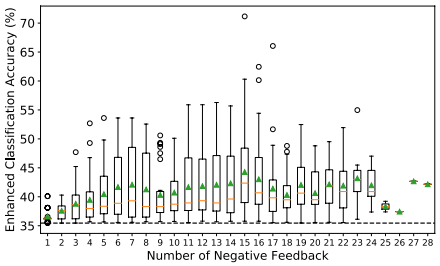


Figure 6: Multi-class *H.Tree* classification accuracy wrt # of feedback.

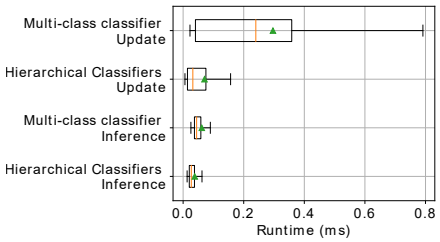


Figure 7: Execution time of a multi-class *H.Tree vs* hierarchical binary *H.Trees*.

classifiers trained with $DATASET_1$ are evaluated by simulating negative feedback for which we leverage $DATASET_2$. Figure 4 provides the F_1 score according to the number of (negative and hierarchical) feedback occurrences. Among the 15 occurrences, at most 4 are related to M_{pocket}, 12 to M_{door} and 15 to M_{ground}. The F_1 score gets an enhancement 100%, 90% and 71% of the time for M_{pocket}, M_{door} and M_{ground}, respectively. The enhancement of the F_1 score is the most significant for M_{pocket} and the least significant for M_{door}. Overall, 8 hierarchical feedback occurrences provide a high F_1 score for all the three classifiers.

The *SenseTogether* middleware requires around $100MB$ of memory on a smartphone (Qualcomm Snapdragon 636). The inference and update of contexts necessitate around $3MB$ of memory. The inference execution time is on average $0.2ms$, $0.1ms$ and $0.1ms$ for M_{pocket}, M_{door} and M_{ground}, respectively, while the execution time necessary to update the model is $7.3ms$, $7.5ms$ and $10.0ms$ on average. Summarizing, our approach allows adapting the tradeoffs between power consumption and accuracy, while inducing limited resource consumption on the smartphone. Figure 5 shows the CPU, memory, and energy consumption of running *SenseTogether* with machine learning performing hierarchical inference every second, and opportunistic update according to the user feedback. When receiving feedback from the user, the CPU usage slightly increases due to the update of the models. The memory contains the amount consumed by basic Android APP components and the computing does not obviously affect the memory. There is no impact on the network consumption and the level of energy consumption remains light.

Multi-class Classifier v.s. Hierarchical Classifiers: An alternative to our online learning per classifier would be performing a single and multi-class classification, which distinguishes 8 combinations of in/out-pocket, in/out-door and under/on-ground. We compare the two approaches using 100 runs of experiment. As illustrated in Figure 6, the mean of the enhanced accuracy (triangles in plot) of the multi-class classification always remains below 45%. Also, the initial and enhanced classification accuracy of a multi-class classifier is $1/2$ time lower compared to hierarchical classifiers. Besides, the multi-class classifier requires more feedback compared to the hierarchical classifiers: Up to 28 feedback occurrences are required. Also, with the multi-class classifier, the user must select among 7 options instead of one or two options. Finally, our hierarchical classifier involves a much shorter update and a slightly shorter inference compared to the multi-class classifier because the hierarchical classifiers do not always perform all the classifications (see Figure 7).

Overall, the hierarchical classifiers offer many advantages: (1) A classifier per context element results in a high classification accuracy; (2) Each classifier only relies on the most relevant features, which reduces the inference and update execution time; (3) A classifier is easily added/removed when a new context element is handled for the benefit of the crowd-

sensing application; (4) Hierarchical classifiers limit the number of inferences that are triggered; (5) The required user feedback is simple and reduced.

4 Conclusion

AI needs data, and data needs AI. We tackle both perspectives in our work where we aim at fostering the collection of high-quality observations from the contribution of the crowd. To do so, we leverage online machine learning so as to contextualize the gathered observation, in a way that is both resource-efficient and accounts for the specific of the crowdsensors –spanning the device characteristics, the end-user's behavior and the environment. Our work builds on the assumption that crowdsensing will be increasingly a significant source of data for AI. However, this also means attracting a large-enough crowd over time. This is known to be a hard problem and solutions lie in the ability for the crowdsensing application to self-adapt to the user's expected gain [12]. We are currently investigating such solutions where AI and intelligent middleware again have a major role to play.

References

[1] M. Chi, Z. Sun, Y. Qin, J. Shen, and J. A. Benediktsson. A novel methodology to label urban remote sensing images based on location-based social media photos. *Proc. IEEE*, 105(10), 2017.

[2] J. Demšar, T. Curk, A. Erjavec, C. Gorup, T. Hočevar, M. Milutinovič, M. Možina, M. Polajnar, M. Toplak, A. Starič, M. Štajdohar, L. Umek, L. Žagar, J. Žbontar, M. Žitnik, and B. Zupan. Orange: Data Mining Toolbox in Python. *J. Mach. Learn. Res.*, 14(1), 2013.

[3] Y. Du, V. Issarny, and F. Sailhan. User-centric context inference for mobile crowdsensing. In *Proc. ACM/IEEE IoTDI*, 2019.

[4] M. Elhamshary, M. Youssef, A. Uchiyama, H. Yamaguchi, and T. Higashino. Transitlabel: A crowd-sensing system for automatic labeling of transit stations semantics. In *Proc. ACM Conf. on Mobile Systems, Applications, and Services*, 2016.

[5] M. Elhoushi, J. Georgy, A. Noureldin, and M.J. Korenberg. A Survey on Approaches of Motion Mode Recognition Using Sensors. *IEEE Trans. Intelligent Transportation Systems*, 18(7), 2017.

[6] R. Gao, M. Zhao, T. Ye, F. Ye, G. Luo, Y. Wang, K. Bian, T. Wang, and X. Li. Multi-story indoor floor plan reconstruction via mobile crowdsensing. *IEEE Trans. Mobile Computing*, 15(6), 2016.

[7] Geoff H., L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proc. ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, 2001.

[8] S. Hachem, V. Mallet, R. Ventura, A. Pathak, V. Issarny, P. G. Raverdy, and R. Bhatia. Monitoring noise pollution using the urban civics middleware. In *Proc. IEEE BigDataService*, 2015.

[9] V. Issarny, V. Mallet, K. Nguyen, P-G Raverdy, F. Rebhi, and R. Ventura. Dos and Don'ts in Mobile Phone Sensing Middleware: Learning from a Large-Scale Experiment. In *Proc. ACM Middleware*, 2016.

[10] H. Jin, L. Su, H. Xiao, and K. Nahrstedt. Incentive mechanism for privacy-aware data aggregation in mobile crowd sensing systems. *IEEE/ACM Trans. Networking*, 26(5), 2018.

[11] B. Lefèvre, R. Agarwal, V. Issarny, and V. Mallet. Mobile Crowd-Sensing as a Resource for Contextualized Urban Public Policies: A Study using Three Use Cases on Noise and Soundscape Monitoring. *Cities & Health*, May 2019.

[12] B. Lefèvre and V. Issarny. Matching Technological & Societal Innovations: The Social Design of a Mobile Collaborative App for Urban Noise Monitoring. In *Proc. IEEE SmartComp*, 2018.

[13] M. Li, P. Zhou, and Y. Zheng. IODetector: A Generic Service for Indoor/Outdoor Detection. *ACM Trans. Sensor Networks*, 11(2), 2014.

[14] S. Li, Z. Qin, H. Song, C. Si, B. Sun, X. Yang, and R. Zhang. A lightweight and aggregated system for indoor/outdoor detection using smart devices. *Future Generation Computer Systems*, 2017.

[15] S. Liu, Z. Zheng, F. Wu, S. Tang, and G. Chen. Context-aware data quality estimation in mobile crowdsensing. In *Proc. IEEE INFOCOM*, 2017.

[16] Y. Meng, C. Jiang, T. Q. S. Quek, Z. Han, and Y. Ren. Social learning based inference for crowdsensing in mobile social networks. *IEEE Trans. Mobile Computing*, 17(8), 2017.

[17] 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. *J. Mach. Learn. Res.*, 12, 2011.

[18] Sharman Apt Russel. *Diary of a Citizen Scientist*. Oregon State University Press, 2014.

[19] F. Sailhan, V. Issarny, and O. Tavares Nascimento. Opportunistic Multiparty Calibration for Robust Participatory Sensing. In *Proc. IEEE MASS*, 2017.

[20] R. Ventura, V. Mallet, and V. Issarny. Assimilation of mobile phone measurements for noise mapping of a neighborhood. *Journal of the Acoustical Society of America*, 144(3), 2018.

[21] R. Ventura, V. Mallet, V. Issarny, P.G. Raverdy, and F. Rebhi. Evaluation and calibration of mobile phones for noise monitoring application. *Journal of the Acoustical Society of America*, 142(5), 2017.

[22] D. Weir, S. Rogers, R. Murray-Smith, and M. Lochtefeld. A user-specific machine learning approach for improving touch accuracy on mobile devices. In *Proc. ACM UIST*, 2012.

[23] G. M. Weiss, J. W. Lockhart, T. T. Pulickal, P. T. McHugh, I. H. Ronan, and J. L. Timko. Actitracker: a smartphone-based activity recognition system for improving health and well-being. In *Proc. IEEE DSAA*, 2016.

[24] I. H Witten, F. Eibe, M. A. Hall, and C. J. Pal. *Data mining practical machine learning tools and techniques*. Morgan Kaufmann, 2017.

[25] Z. Xu, Y. Liu, and N.Y Yen. Editorial for crowdsensing and intelligent sensing on mobile media analytics special issue. *Mobile Networks and Applications*, 22(2), 2017.

[26] M. Zappatore, A. Longo, and M.A. Bochicchio. Using mobile crowd sensing for noise monitoring in smart cities. In *Proc. IEEE SpliTech*, 2016.