Federal University of Pernambuco
Center of Informatics

Undergraduate in Computer Science

Activity recognition in noisy environments: an evaluation of the robustness of machine learning algorithms

Manuela Silveira Barbosa

B.Sc. Dissertation

Recife
December, 2017

Federal University of Pernambuco
Center of Informatics

Manuela Silveira Barbosa

Activity recognition in noisy environments: an evaluation of the robustness of machine learning algorithms

A B.Sc. Dissertation presented to the Center of Informatics of Federal University of Pernambuco in partial fulfillment of the requirements for the degree of Bacharel in Computer Science.

Advisor: *George Darmiton da Cunha Cavalcanti*

Recife
December, 2017

Acknowledgements

First of all, I would like to thank my parents, Pedro and Eliana who invested on my education, always highlighting the importance of education and encouraging me even in the most difficult times.

I would also like to thank my grandmother, Helena, for being so supportive, loving, and the purest example of kindness present in my life.

Thanks to my boyfriend, Nicolas, who stood by me in the tough moments, listening to all my problems, helping me when I was in need, distracting me and being beside a lover, an exceptional friend.

Moreover, I would like to say thank you to my colleges from university who stood by me in all the ups and downs and traced the same path, being part of the achievements and all the difficulties.

Furthermore, I would like to thank all my colleges at my internship for being by my side, asking if I needed help with something, supporting and understanding my absences.

I would like to thank all the professors that participated on my formation, transmitting their knowledge and helping me to overcome my difficulties, as well as, providing me the fundamentals to become a good professional and person.

Finally, I would like to thank professor George for all the support and guidance. He would always clarify my questions promptly and review thoroughly each part of the dissertation. Without his help none of this would be possible.

Education is the most powerful weapon which you can use to change the world

—NELSON MANDELA

Abstract

Studies in activity recognition have gained attention and matured in the past years. This is mainly due to advances in the sensor's technology and in the underlying activity discovery and recognition process. Activity recognition in smart home environments aims to identify which activities are currently happening and predict which activities are going to happen in order to monitor the residents status and promote comfort and well-being. The sensor's information extraction is the foundation of the activity recognition. If the data is not encrypted properly, it will become an easy target for interceptions and the resident will be more likely to suffer from malicious unauthorized attacks. Two of the major challenges in smart homes are privacy and security. In order to forbid hackers to retrieve resident data and determine which activities the resident is performing or learn the resident habits, one technique that has been studied is the insertion of noise in the data generated by the sensors. This work aims to analyze how real-world smart home datasets and machine learning techniques perform while being exposed to noise. Experiments are conducted over 10 different datasets. Using the 5-fold cross-validation method, each machine learning technique was evaluated against different noise ranges. Based on the results obtained, the machine learning algorithms suffered considerably from the noise insertion. Decision Tree, Random Forest, and K-NN achieved the best general accuracy and were affected significantly by the noise. SVM did not experience drastic changes when exposed to noise but had bad results in most cases and Naive Bayes can be considered the worst of all classifiers under those circumstances.

Keywords: Activity recognition, Smart homes, Noise, Sensors, Information extraction, Security, Privacy, Machine learning

Resumo

Estudos em reconhecimento de atividades ganharam bastante atenção e amadureceram nos últimos anos. Isto é devido principalmente a avanços na tecnologia implantada nos sensores e no processo de descoberta e reconhecimento de atividades. Reconhecimento de atividades aplicado a casas inteligentes visa identificar que atividades estão acontecendo e prever quais ainda vão se realizar com objetivo de monitorar o estado dos residentes e promover um maior conforto e bem estar. A extração de informações proveniente de sensores é o alicerce do reconhecimento de atividades. Se os dados não forem encriptados devidamente, eles se tornarão um alvo fácil para intercepções e os residentes estarão mais propícios a sofrer ataques maliciosos e não autorizados. Dois dos maiores desafios encontrados em casas inteligentes são privacidade e segurança. Com o intuito de impedir *hackers* de adquirir dados dos residentes e determinar quais atividades os residentes estão executando ou aprender os hábitos dos residentes, uma técnica que vem sendo estudada é a inserção de ruído nos dados gerados pelos sensores. Este trabalho analisa como bases de dados de casas inteligentes do mundo real e técnicas de aprendizagem de máquina se comportam quando expostas a ruídos. Os experimentos são conduzidos sobre 10 bases de dados diferentes. Utilizando o método de validação cruzada, analisa-se cada técnica de aprendizagem de máquina em relação a diferentes níveis de ruído. Baseado nos resultados obtidos, os algoritmos de aprendizagem de máquina sofreram consideravelmente com a inserção de ruído. Árvore de decisão, *Random Forest* e *K*-NN alcançaram as melhores acurácias gerais e foram afetados significativamente pelo ruído. SVM não apresentou mudanças drásticas quando exposto a ruídos mas obteve resultados ruins na maioria dos casos e *Naive Bayes* pode ser considerado o pior dos classificadores.

Palavras-chave: Reconhecimento de atividades, Casas inteligentes, Ruído, Sensores, Extração de informação, Segurança, Privacidade, Aprendizagem de máquina

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Motivation

Progresses in the sensor's technology and data mining techniques allowed the growth of the Internet of things. The Internet of things states that ordinary daily objects are going to become intelligent, sensing the context that it is inserted and making assumptions and decisions to improve the quality of life, the object overall performance, and its energy consumption.

Smarts houses are residences that possess a good amount of these intelligent ordinary objects. In the past, these objects were restrict to lighting and heating systems, nowadays most of the electrical objects can become intelligent somehow. The smart objects are able to recognize some actions performed by the residents and sense the environment itself, as well as communicate with each other and consequently facilitate the resident life.

There is no use in having plenty of sensors, a vast quantity of data if there is no information extraction associated with them. Activity recognition techniques aim to keep track of all the resident activities, it monitors the functional status of the resident in real-time. Doing so, it is possible to find patterns in the resident routine and maybe automate some tasks.

It is very important that residents remain independent in their own houses, activities considered fundamental also known as Activities of Daily Living(ADLs) such as cooking, drinking, taking medicine and grooming are essential for living a functional autonomous life. Above from that, the aging population, the cost of healthcare, and the opportunity of improving the quality of life of individuals with disabilities, increasing the support received from the environment, were some of the reasons that motivated further studies in this area.

The information exchanged between smart objects is very susceptible to attacks. Researchers have been concerned about security and privacy issues in the context of IOT. One of the methods they proposed in order to prevent attackers to steal sensitive information and infer the resident status is to add noise on the training set.

Most of the approaches that treat activity recognition in the literature focus on well scripted or pre-segmented data, which does not represent the real-world scenario. In order to be more realistic, there is a need to have an online activity recognition system that is able to overcome embedded errors and identify interleaved and concurrent activities.

Some studies interesting to point out that already explored activity recognition in the smart home scenario were the MavHome [1], the PlaceLab [2], the AwareHome [3] and the CASAS project [4]. They reiterate the importance of this field of study.

1.2 Objective

The main purpose of this work is to analyze the performance of the machine learning techniques in the context of activity recognition in real-world smart houses against some different noise percentage scenarios. This would prove whether inserting noise in the training data should be considered sufficient in order to camouflage the authentic data and forbid attackers to have access to the legitimate activities that the resident is performing. With the objective of validating this assumption, some metrics were established and the results obtained in the noise-free scenario and the scenarios with some percentage of noise inserted were compared as a manner of understanding the impact that noise can cause.

1.3 Methodology

The first step in the development of this work after choosing the machine learning area of study was to perform a literature review. This facilitated the overall understanding of the most basic concepts, main objectives and difficulties faced by the area, as well as made it possible to pursue an analysis of the existent techniques. After the conclusion of the literature review, one of the papers was selected to serve as guidance. The activity recognition technique proposed in this work is based on the paper "Activity recognition on streaming sensor data" written by Narayanan Krishnan and Diane Cook [5]. Based on ideas expressed in this paper, an activity recognition algorithm was implemented. After that, the decision regarding the work's specificity was made. It was decided to analyze the behavior of the algorithm and the machine learning models in different noise percentage scenarios. The machine learning techniques were also selected at this point. The proposed method was analyzed using 10 real-world data sets from the CASAS dataset repository [4] against 5 machine learning models in 6 noise percentage scenarios. The evaluation metrics were chosen and the results were interpreted and discussed.

1.4 Organization of the Dissertation

This work is organized as follows. Chapter 2 refers to the literature review and it is responsible for presenting all the basic concepts. It also gives the foundation necessary to understand the theory associated with the areas of studies related to this work and the motivation behind each one of them. Chapter 3 explains in detail the proposed method. It deciphers the components, steps, and assumptions made during the implementation. The experimental study can be seen in Chapter 4. It describes the datasets used in this work, as well as how the experiments were made, which evaluation metrics were chosen and discusses the results obtained. Chapter 5 sums up what could be learned by this work, if the objectives were met and suggests some improvements to consider for future works.

CHAPTER 2

Literature Review

This chapter is responsible for providing all the basic concepts necessary to fully understand this work. Section 2.1 explains what is Internet of Things and its connection to smart houses and activity recognition. Section 2.2 talks a little bit about an emerging area of study called Adversarial Machine Learning and what is its biggest concerns and objectives. Section 2.3 reveals what can be considered noise, what are the most common kinds of noise and how can noise be useful. Section 2.4 clarifies the purpose of machine learning as well as explains the functioning of some of the machine learning techniques used in this work, their advantages and disadvantages.

2.1 Internet Of Things

The term Internet of Things has become very popular in the last couple years. However, this concept is not entirely new. The idea of Ubiquitous computing was already disseminated in the early 1990s by Mark Weiser. Ubiquitous computing presupposes that computing is made to appear everywhere and anywhere. According to Weiser, the ordinary computers such as desktops and laptops are a transitional step towards achieving the real potential of information technology. He believes that the most profound technologies are those that disappear and that in the future the computers will be integrated with the world seamlessly in the most different forms and sizes [6].

On the other hand, Internet of Things states that everyday devices are going to become smarter and communicate with one another, sharing information and coordinating decisions [7]. In other words, the Internet will be more present in the real world daily life objects, such as televisions, fridges, stoves, lamps, washing machines, coffee makers and blenders.

Unlike the most common forms of communication nowadays which are Human-to-Human and Human-to-Machine, IoT is a concept to get devices connected to the Internet in a Machine-to-Machine interaction [8].

The sensors will provide context to the devices, communicate with each other and interact with people [7]. Doing so, the sensors are able to optimize their performance and simplify the resident's lifestyle by automating some tasks and saving time, energy and money [9].

Internet of things relies on the continuous advances in the sensor technology, the communication and information technology, pervasive computing and the Internet protocols compatible with the most heterogeneous things [10].

Data science plays a very important role to enhance IoT application intelligence. Data Science consists of a combination of different fields such as machine learning and data mining

with the objective of finding insights and patterns hidden on the raw data [9].

Researches estimate that by 2020, the total number of Internet-connected devices will be between 25 and 50 billion [11]. A problem that has to be faced is the enormous number of devices willing to connect to the Internet. To meet customer demands for smart objects, it is necessary to utilize a large addressing space, such as IPv6 [10].

IoT can be of great importance in order to improve the quality of people lives. Internet of things can be applicable to health-care, transportation, emergency response to natural disasters, agriculture, industrial manufacturing, education, smart homes and many other sectors [10].

In smart homes, Internet of things makes it possible for the resident's digital devices to interact with each other and for instance, open the garage when the resident is reaching home, prepare the coffee when waking up in the morning, turn off the TV when the resident is not watching and forgot it on, control the apartment temperature, turn all the lights off when the resident leaves the house and other applications. With the objective of making assumptions and control the digital devices in smart houses, activity recognition methods are fundamental to identify the resident's context.

Two of the most important requirements in relation to IoT are Security and privacy. Considering the resident ability to control and monitor physical objects, the systems have to be cautious and make sure the information does not fall into the wrong hands [10]. The way the data is transmitted makes the system vulnerable. This point will be more elaborated in the next subsection.

2.2 Adversarial Machine Learning

Machine learning algorithms have been progressively more adopted in security-related scenarios such as malware and spam detection, network intrusions, fraudulent transactions and other malicious activities [12][13]. However, attackers may take advantages of vulnerabilities exposed by machine learning algorithms to violate system security [14]. For instance, an attacker can modify or generate data from a spam filter to deceive the system and increase misclassification [15].

In machine learning, a classification problem takes the feature vector from the input samples and labels it with the most adequate class. The decision relies on how the model interprets the feature vector. In this context, adversarial samples are created or modified to force the machine learning model to classify them in a different class compared to their legitimate class [8]. An adversary may attempt to manipulate a system targeting the machine learning component by changing some of its inputs (i.e., poisoning the learner's classifications) in order to harm the system's detection capabilities or stealing private information (i.e., obtaining information from the learner, compromising the privacy of the system's users). If the attacker acquires some portion of the training examples he/she might manipulate the system into revealing information about the unknown examples [12][16].

This field of study aims to find effective machine learning techniques against an adversarial opponent, algorithms that can resist the attacks [16]. It is noticeable how much important became for the machine learning algorithms to operate well in adversarial settings. The term model resilience means how robust the system is to perturbations of its input. The more pertur-

bation is necessary to move a sample from its legitimate class to another class, the more robust the model is to adversarial manipulations, and this is exactly what this field is seeking [13].

Security is a significant challenge for the IoT implementations because of the lack of common standard and architecture. Securing data exchanges is necessary to avoid losing or compromising privacy. Heterogeneous networks suffer to guarantee security and privacy to the users [10]. The debate whether to accept or reject IoT involves the dualisms of "security versus freedom" and "comfort vs data privacy". The concern is that the personal data is collected and used by third parties without people agreement or for potentially damaging purposes [7].

In the smart house scenario, the adversary could try to control all the digital devices interconnected, unlocking the front door and turning off the robbery alarms for example.

One of the major directions machine learning have pursued regarding privacy is the randomization/perturbation of data. In this approach, the data is perturbed, noise is added to the data, the labels or attributes have its values swiped in the preprocessing phase [17]. The next subsection will explain in more detail noisy data.

2.3 Noisy Data

One usual problem identified in classification problems is the presence of noise in data. Real-world data is never perfect. In supervised problems, the noise changes the relationship between the features and the desired output. The model's performance depends on the training data quality and on the robustness of the algorithm against noise, which is its capability to build models less sensitive to data corruption. In noisy scenarios, the robustness is considered to be the most important performance result [18].

In order to diminish the impact of noise in the classification problem, some approaches have been studied. The use of robust learners, data polishing methods before training and noise filters that eliminate noisy instances from the training set are examples of alternatives [18].

It is important to point out that the presence of noise is not the only problem faced by the machine learning algorithms. The presence of small disjuncts and overlapping between classes might generate complex and nonlinear boundaries between classes that can result in the degradation of the classification performance. Nevertheless, noisy examples are considered to be samples from one class occurring in the safe areas of other class, not in the boundary.

Noise can be divided into two types. Class noise also known as label noise and attribute noise. Class noise takes place when an example has its label incorrectly assigned and attribute noise refers to corruptions in the attribute values (e.g., missing, unknown or erroneous attributes). Studies proved that attribute noises are more harmful than label noises [18].

However, noise is not always necessarily something bad. As described in the last subsection, researchers are studying the insertion of noise in order to contribute to the system security.

While checking the effect of noise on a specific classifier, noise needs to be generated. First, it is necessary to choose the place the noise will be introduced (i.e., attribute or label), the noise distribution used in the simulation (e.g., Gaussian, uniform) and the magnitude of the generated noise, how much of the training set will be manipulated.

The label noise insertion can be performed following one of the noise schemes in the literature. The uniform class noise consists of randomly replacing the class label of x% of the

examples. Besides that technique, the pairwise class noise insertion is also a well-known approach. Given a pair of classes (X,Y) and a noise level x, an instance with label X has x/100 probability to be mislabeled as Y [18].

To analyze the robustness of the classifier, a comparison between the original training set and the data generated by the different noise magnitudes needs to be executed. Therefore, those classifiers whose results are not so distant from the noise-free classifier are considered to be the most robust.

2.4 Machine Learning

There are some tasks in computer science in which the algorithm is unknown, there is no set of instructions to transform the input in the desired output. However, if a good amount of data is available, we can rely on the examples and construct machine learning algorithms intelligent enough that have the ability to learn from the data and get insights from it [19].

Classification problems are very common in Machine learning. In this type of problem, based on the data examples used in the training phase, the machine learning algorithm tries to categorize the examples belonging to the test set in the more satisfactory class, taking into account the resemblance between this example and the other members of the different classes.

Each specific learning algorithm has a distinct inductive bias and makes different assumptions. This subsection presents the machine learning algorithms used in this work.

2.4.1 Decision Tree

Decision trees are one of the most widely used machine learning techniques for inductive inference. Decision trees can be represented by a series of if-then rules to help human readability. Each node of the tree represents a test of some attribute of the instance and each branch descending corresponds to one of the possible values for that attribute. The classification of an instance starts at the tree's root and after testing the correspondent attribute, you move down until you reach one of the leaf nodes, which provides the classification [20].

The root of the tree and the attribute test order is determined by evaluating, using statistical tests, which attribute performs better, which one is the most useful for classifying examples. The best attribute, the one that has the most information gain is selected.

Information gain is basically the reduction of entropy while partitioning examples in relation to an attribute. Entropy characterizes the impurity of a collection of examples.

Some of the advantages regarding the decision tree usage are their easy interpretation and implementation and their capability to show possible paths not yet considered.

The disadvantages that can be associated with the decision tree are that the calculations can become very complex and time-consuming, they are not robust to noisy, meaning that a small change can cause a totally different outcome and the algorithm is considered greedy since it never backtracks to reconsider earlier choices.

2.4.2 *K*-Nearest Neighbors

K-Nearest Neighbors is an instance-based learning algorithm. There is not an effort to discover a target function that maps the examples to its classes. Every time a new testing instance needs to be evaluated, the distance between this instance and the others from the training set is calculated. Because the algorithm only processes the data when a new instance comes to be classified, the instance-based methods are referred as "lazy" learning algorithms. The fact that the instances can be represented as points in a Euclidean space is an assumption used by this method. Each test instance will have its distance calculated from all the training instances. The distance function can be the Euclidean distance for example. The K most similar neighbors (i.e., the K closest ones) are going to determine the class. The majority vote can be used among the K neighbors in order to classify the test instance.

K-NN is also simple to implement, is flexible to features and distance choices, there is no training cost, and it handles multi-class naturally. Some drawbacks found in the *K*-NN approach are that the cost can be high since all computation takes place in classification time, the performance depends on the number of dimensions and on the size of the database, the need to determine the value of parameter K, it does not learn from the training data, the algorithm may not generalize well and might not be robust to noise, and that it considers all attributes in order to identify the similar training examples. In a scenario in which the classification relies on only a portion of the attributes, the distance could be larger than it was supposed to be.

2.4.3 Naive Bayes

The Naive Bayes is a supervised learning method based on a probabilistic approach to inference. It assumes that there is a probability distribution and that an optimal solution can be found [20]. It classifies an example in a specific class based on the probability of this example belong to that class.

The Naive Bayes implementation is simple, it requires a small number of training data and it does not only returns the prediction but also the degree of certainty. However, it considers that attributes are independent what actually might not be the reality. It also relies on initial knowledge of many probabilities, otherwise, these probabilities are estimated, the computational cost, in order to find the optimal hypothesis, is significant and if there are no occurrences of a class label in relation to an attribute value, the probability estimated is going to be zero.

2.4.4 Random Forest

Random Forest is an ensemble approach composed of several independent decision trees combined with aggregation and bootstrap ideas. It is based on the divide and conquer paradigm, in which a group of weak learners gather together to form a strong model.

Random Forest was created in order to fix the overfitting seen in general decision trees that have grown very deep.

If a tree is very sensitive to noise in its training set, the majority of the others trees might not be assuming that they are not correlated. When taking the majority vote between the trees, the noise would not affect as much, making this machine learning algorithm more robust to

noise and leading to best performance. The number of trees can vary. Usually, a few hundred to several thousand are chosen according to the size and nature of the training set. In a Random Forest, the learning algorithm selects a random subset of the features at each candidate split in the learning process. This avoids that few strong features get selected in most of the trees, otherwise, this would make them become correlated. Random Forests are considered efficient on large datasets and can achieve better results than simple decision trees by reducing overfitting. However, they are more complex than a simple decision tree and more computationally expensive. They also do not provide an easy visualization and understanding of the model.

2.4.5 Support Vector Machine

Support Vector Machine (SVM), is a discriminant-based method. This algorithm aims to find the boundary separating the classes. The examples are represented as points in a n-dimensional space, where n is the number of features. SVM tries to map the examples in order to separate examples from different classes even further finding the hyperplane that best segregates the classes. To identify the best hyperplane among the possible hyperplanes, the margin (i.e., the distance between the nearest data point of both classes and the hyperplane) must be taken into consideration. It is considered to be the best hyperplane the one that segregates well the classes and has the maximum margin. If the margin is large the chances of miss-classification are lowered and the algorithm is more robust. Some advantages associated with SVM are its performance in high dimensional spaces, its generalization capability and the fact that it works well in clear margin of separation. Some of the drawbacks are the training time can be really time-consuming if the database is very large, it does not perform so well when the target classes are overlapping, it is difficult to find the appropriate kernel function and to understand the final model and the algorithm is complex.

CHAPTER 3

Proposed Method

The activity recognition technique proposed in this work can be divided into two phases, training (Figure 3.1), and testing (Figure 3.2). It is important to point out that both flows have some components in common and they also have a preprocessing phase in order to prepare the data to be trained or tested. Each component is explained in more detail in the following subsections.

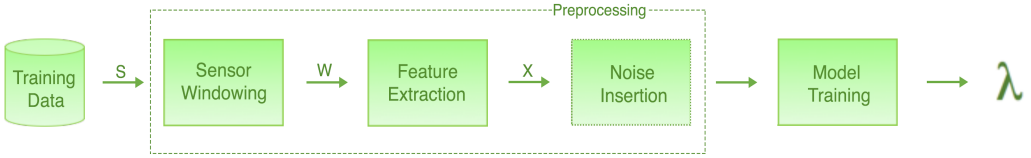


Figure 3.1: Overview of the proposed framework training phase

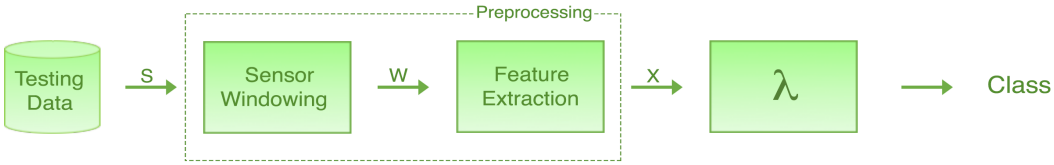


Figure 3.2: Overview of the proposed framework testing phase

However, some notations must be defined and some explanations must be given. First of all, the datasets adopted in this work consists of sensor events. Let $\{S_1, S_2, \ldots, S_N\}$ represent the sequence of sensor events. Each sensor event is composed of calendar day, time of the day, sensor name, and sensor status. All of the datasets selected in this work were annotated. Therefore, in addition to the fields mentioned above, some sensor events have an activity label indicating the beginning or ending of an activity.

A fragment of one of the datasets used in this work can be seen in Figure 3.3. The first column represents the calendar day composed of year, month and day. The second column refers to the time of the day, which consists of hour, minute, second and millisecond. The third column identifies the sensor. T101 is a specific temperature sensor found in the apartment, per example. The fourth column expresses the sensor value. It can be nominal like ON/OFF or numeric such as "35" seen in the first row. Some sensor events have an activity associated with it. In the figure, it is noticeable the beginning and ending of the sleep activity and the beginning of the bed to toilet transition.

```
2011-06-15  00:16:25.586279 T101    35
2011-06-15  00:22:40.981283 BATV005 9540
2011-06-15  01:01:36.427998 BATV007 9560
2011-06-15  01:03:39.14962  MA009   ON  Sleep="begin"
2011-06-15  01:03:40.284526 MA009   OFF
2011-06-15  01:17:37.377261 BATV102 3170
2011-06-15  01:40:02.937855 MA009   ON
2011-06-15  01:40:05.20224  MA009   OFF
2011-06-15  01:40:48.900973 MA009   ON
2011-06-15  01:40:50.015327 MA009   OFF Sleep="end"
2011-06-15  01:41:27.905913 MA007   ON
2011-06-15  01:41:29.789869 MA007   OFF
2011-06-15  01:41:30.712847 MA007   ON
2011-06-15  01:41:31.842606 MA007   OFF
2011-06-15  01:41:40.183566 BATV008 9540
2011-06-15  01:41:40.205759 MA008   ON  Bed_Toilet_Transition="begin"
2011-06-15  01:41:41.296767 MA008   OFF
2011-06-15  01:41:43.385244 MA008   ON
```

Figure 3.3: Fragment of the HH103 dataset

In Figures 3.1 and 3.2, the S going out of the datasets correspond to the sequence of sensor events. The windows generated by the sensor windowing component, are represented by W. When the windows pass through the feature extractor component, the feature vectors obtained are defined as X. The symbol λ represents the machine learning technique. The machine is trained with the training examples after the noise insertion component and is tested with the testing data preprocessed resulting in the predicted class.

In the training phase, initially, sensor events are going to pass through a preprocessing phase, in which the data is organized in windows, feature vectors are extracted and the label noise is inserted. The raw sensor data doesn't have a very strict structure, in which every row has the exact number of columns making it difficult to train most machine learning algorithms without any kind of preprocessing.

3.1 Sensor windowing

There are different ways of processing streams of data. The sensor stream can be equally divided into windows according to a specific time interval. A problem observed with this technique is the possibility of having windows with a small number of sensor events, containing no activities and windows with more than one activity inside. It is recommended to use the time-based windowing when dealing with constant time rate sensors, such as accelerometers and gyroscopes [21]. The approach adopted in this work, known as sensor event based windowing, rather than taking into consideration the time in order to create the windows, uses the sensors itself with that exact purpose.

The idea of the sensor event based windowing is to divide the sensor stream into windows containing an equal number of sensors events. The sequence $\{S_1, S_2, \ldots, S_N\}$ is reorganized and results in the creation of the windows $\{W_1, W_2, \ldots, W_M\}$, where M is the number of windows generated from the sequence of sensor events according to the window length desired. Doing this, all the windows will have the same number of sensor events and can vary their duration. The last sensor event in the window will determine the window activity label. However, this method has some drawbacks. For example, the relevance of all the sensors is the same, inde-

pendently of when the sensors were triggered. Another concern important to point out is the difficulty in determining the window length in order to sufficiently define the window context [5]. Figure 3.4 illustrates how sensor event based windowing works assuming that the window length is 4.

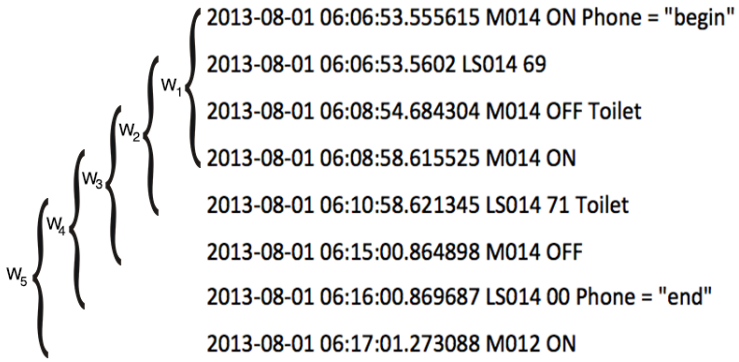


Figure 3.4: Example of sensor based windowing with window length=4

As can be seen in Figure 3.4, the dataset fragment resulted in 5 windows (M=5). The window W_4, for example, goes from the fourth line until the eighth line since the window length was set to 4. Moreover, its associated activity label is "Phone" based on the last sensor event found in the eighth line.

3.2 Feature Extraction

Once the sensor window W_i is defined, the next step is to generate a fixed dimensional feature vector X_i. The vector is composed of the time of the first sensor event within the window, the time of the last event within the window, the span time between the last sensor event and the first sensor event, and a bag of all sensors indicating which ones have been triggered.

- **Time based features** The first sensor event and last sensor event time are computed by calculating the POSIX timestamp and multiplying it by 1000. In the end, this will represent the number of milliseconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC), Thursday, 1 January 1970. The POSIX was used in order to take into consideration different days and not only the time of the day in the calculations. Thereby, the span time is the subtraction of the last sensor event by the first sensor event.

- **Bag of sensors** The bag of sensors is nothing more than a count of how many times each sensor was triggered in that specific window. There is no differentiation between the status of ON/OFF in the count. Whatever status was obtained, either ON/OFF or a numeric value, it will sum 1 in the correspondent sensor count. As mentioned before, the bag of sensors size will be equal to the number of sensors in the dataset. For instance, if

the database has 30 sensors, the dimension of X_i would be 33 since there are 30 distinct sensors and there are 3 time-based features. Each X_i is labeled with the last sensor event activity within the window.

- **Activity labeling** Some special cases should be mentioned regarding the window labeling. Figure 3.4 displays a hypothetical sensor based windowing scenario in which the window length is set to 4. This fragment results in 5 windows and most of the special cases can be identified in this example. The first window ends in between the "Toilet" activities. Since there is no activity indication in the last sensor event of this window we have to look at the previous activity within the windows. Even though the last activity was "Toilet", since it is not a begin-end block this activity was not taken into consideration. However, W_1 is inside the "Phone" begin-end block explicitly seen in the first line. Therefore, W_1 label is "Phone". W_2 activity label is "Toilet" since it is explicitly written in its last sensor event. The activity class corresponding to W_3 is "Phone" for the same reason noticed in W_1. W_4 activity label is "Phone", even though the last sensor event says that it was the end of the "Phone" activity, the label was chosen to still be designated. Lastly, the ultimate sensor event within the window W_5 is not inside any begin-end block, it does not correspond to any of the known activities. Whereas, it was introduced a new type of activity denominated "Other". Typical real-world settings tend to have the class "Other" as dominant, that is why its introduction is considered to be of great importance [5].

Taking into consideration the window W_4 in Figure 3.4, the time of the first sensor event within the window is 2013-08-01 06:08:58.615525, the time of the last sensor event is 2013-08-01 06:16:00.869687. These values are converted to milliseconds and the subtraction of them represents the span time between the first and last sensor events. The bag of sensors for this window would have 3 dimensions taking into account that there are only 3 different sensors in this database. The bag of sensors for W_4 is {M014:2, LS014:2, M012:0}. The reason why sensors M014 and LS014 have value 2 is due to the fact that each of these sensors was triggered twice in the window. Sensor M012 was triggered only after the end of window W_4. For that reason, it was not triggered within the window and its value is 0.

After the generation of all feature vectors. They are normalized (rescaling the range of features to [0, 1]). The details referring to the normalization are treated in Section 4.2. When the normalization process is over, the noise is inserted in the training sets. This process is described in more detail next.

3.3 Noise Introduction Mechanism

In order to analyze the behavior of the datasets and understand how the most different machine learning algorithms are affected by noise, the label noise insertion technique was chosen. A percentage of the training data had its label switched for another random activity label according to the noisy rate desired. As explained in Sections 2.2 and 2.3 the insertion of noise can encrypt the data as a manner of preventing the interceptor to get ahold of all the

resident real activity information, obfuscating the truth. Even though the data might still be intercepted, it should be useless since it does not represent the real information.

3.4 Model Training

The machine learning algorithms chosen to train the datasets having as input value the feature vectors generated by the preprocessing phase were: Decision Tree, *k*-Nearest Neighbors, Naive Bayes, Random Forest, and Support Vector Machine. These classifiers were chosen according to the results obtained in the paper "Do we Need Hundreds of Classifiers to Solve Real World Classification Problems?" [22]. This article evaluated 179 classifiers over almost the whole UCI machine learning database. They concluded that the best results achieved were from the Random Forest, followed by the SVM classifier. Decision Trees, Nearest Neighbors, and Bayesian classifiers were far behind. However, we decided to still take them into consideration in order to explore other learning paradigms. All of them were described in more detail in Section 2.4.

The model is trained with the normalized feature vectors and results in the machine learning trained model designated λ.

3.5 Model Testing

For each noise range, all the folds corresponding to that noise range are trained and tested. In order to test the data according to a specific classifier. The sensor events (S) also have to be preprocessed. However, in this case no noise is inserted. The data is split into windows (W) and have its features extracted (X) and normalized following the same rules applied to the training data. The test data serves as the input for the machine learning model generated by the training phase (λ) and the predicted class is obtained for each feature vector.

CHAPTER 4

Experimental Study

This section aims to describe how the experiments were performed and discuss the results obtained. Section 4.1 outlines which datasets were used and their characteristics. Section 4.2 is responsible for defining the experiment parameters. The evaluation metrics established are more detailed in Section 4.3 and in the last Section, the results are presented and discussed.

4.1 Datasets

All the datasets used in this work were obtained in the CASAS (Center for Advanced Studies in Adaptive Systems) website. CASAS serves to meet research needs around testing of the technologies using real data through the use of a smart homes environment located on the WSU Pullman campus [4]. In total, 10 datasets were selected. The testbeds are annotated and contain either 1 or 2 residents only and they represent the resident's daily life. The experiments were not based on scripted activities in a laboratory setting, in which activities are well segmented. This work's approach reflects the real world scenario since it relies on sensor streams. The residents perform their daily routine without being given explicit instruction regarding what to do. Making it possible to investigate more complex scenarios such as interleaved activities, concurrent activities performed by multiple residents and the presence of embedded errors. The datasets analyzed were: HH103, HH105, HH110, HH124, HH125, HH126, HH129, Kyoto 2008, Kyoto Spring 2009 and Tulum 2009. Among the datasets chosen, the minimum number of sensor events is 20952 and the maximum is 486912. The minimum number of activities per dataset is 5 and the maximum is 35. Table 4.1 shows some of the datasets characteristics and more detailed information regarding the activity count in each dataset can be checked on the Appendix.

Data Set	#Residents	#Sensor Events	#Activities
HH103	1	167183	33
HH105	1	225822	31
HH110	1	138331	26
HH124	1	76028	22
HH125	1	216255	35
HH126	1	189814	33
HH129	1	216292	33
Kyoto 2008	2	20952	5
Kyoto Spring 2009	2	138039	16
Tulum 2009	2	486912	11

Table 4.1: Datasets characteristics

The smart homes have the most different kinds of sensors. Some types of sensors are motion sensors, temperature sensors, burner sensors, hot water sensors, cold water sensors, magnetic door sensors and light sensors. An example of the layout of one of the apartments can be examined in Figure 4.1

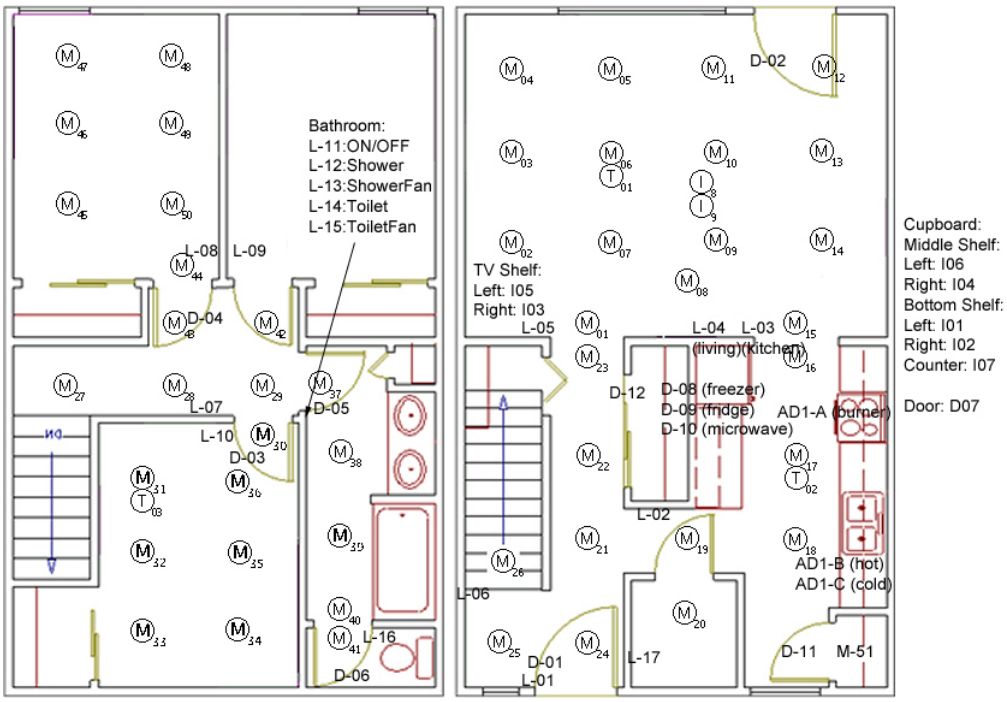


Figure 4.1: Layout of the Kyoto apartment

4.2 Experiment Parameters

This section talks about parameters set in the experiment implementation. As Section 3.1 explained, choosing the window length in order to accommodate a good number of sensor events and designate the context of the window well is a hard task. According to the book Activity Learning, windows lengths between 5 and 30 are more common when talking about fixed-sized windows in sensor event based windowing [21]. Since the objective of this work was not to identify the most optimal window size, there were no implementation or investigation with this purpose. Based on the book and other works which explored more this concept, the window size chose for the experiments was 30.

The normalization of the feature vectors was done using the following formula:

$$X_n = \frac{(X - X_{min})}{(X_{max} - X_{min})}$$

Where X_{min} represents the minimum feature value and X_{max} represents the maximum feature value.

Table 4.2 shows how the feature vector looks like before the normalization and after the normalization. The first line represents the feature vector before the normalization and the second row represents the vector after the normalization process. The columns present in the feature vector visible in the figure are respectively the time of the first sensor event, the time of the last sensor event, the span time between the last and the first sensor event and the following numbers correspond to the bag of sensors. Meaning that this dataset has only 10 sensors. Thus, the feature vector is composed of 13 columns and have one activity label associated with it.

	Start Time	End Time	Span Time	Bag of Sensors
!Normalized	1238667135081.328	1238667163077.2659	27995.93798828125	1, 0, 3, 1, 0, 0, 1, 1, 2, 1
Normalized	0.7703268188170178	0.7016197378551695	0.7009158756572871	0.5, 0.0, 1.0, 0.5, 0.0, 0.0, 0.5, 0.0, 0.5, 0.5

Table 4.2: Example of feature vector structure

After normalizing the feature vectors based on windows of length 30, the introduction of noise was executed. Each training fold had its labels switched according to a certain noise rate. The noise rates analyzed were 10%, 20%, 30%, 40% and 50%. The label substitution is done by randomly choosing a new label among all the activities labels in the dataset and assigning its value to one of the examples in the training fold.

All reported experiments used the stratified 5-fold cross-validation. For each experiment performed, one machine learning technique was evaluated according to all the noise ranges. All the experiments with respect to the same database have the same input since the same fold is used.

The Decision Tree classifier was set in order to choose the best split at each node. The maximum depth was not determined, the nodes are expanded until all leaves are pure or until all leaves contain less than a specific number of samples required to split an internal node, which in the case was 2. In the *k*-Nearest Neighbors classifier the number of neighbors was set to 3, the weights were left with their default value which is uniform, where all points in each neighborhood are weighted equally. The Naive Bayes classifier assumed a Gaussian distribution since we are dealing with continuous data. The number of jobs of the Random Forest was set to 2, while the number of trees remained 10. The Support Vector Machine had its kernel function set to Radial Basis Function (RBF).

4.3 Evaluation Metrics

To evaluate the machine learning performance due to the different noise rates some evaluation metrics were established. For each fold of each noise rate the general accuracy, the accuracy, the precision, the recall, the f-measure and the confusion matrix were calculated. Before explaining each evaluation metric, some abbreviations must be defined:

- **True Positive (TP)** : Consider TP as being an example predicted of class C and its real label is indeed C.

- **True Negative (TN)** : Consider TN the example that does not belong to the class C and it was correctly predicted as not being from the class C.
- **False Positive (FP)** : Consider FP one example predicted C but its ground truth says that this example is actually not from C, it belongs to another class.
- **False Negative (FN)** : Consider FN as being an example that is part of class C but was predicted otherwise, judged to be from a class different than C.

The general accuracy is calculated by dividing the number of examples correctly classified by the total number of examples. The following formula represents the general accuracy

$$Accuracy = \frac{\sum TP + \sum TN}{\sum TP + \sum TN + \sum FP + \sum FN}$$

The other metrics are calculated in relation to each class and the mean value is taken among the activity classes. It is important to point out that the "Other" activities were not considered in the calculations. For each class different than "Other", the values of TP, TN, FP, and FN are calculated.

The accuracy is a metric similar to the general accuracy described above. The only difference is the fact that it is calculated per activity class and the mean is taken between the activities. In other words, each class will have its accuracy and the mean will be computed.

The precision is the number of examples correctly classified as belonging to the class divided by all the examples classified as being part of that class. Whence, the precision is defined as

$$Precision = \frac{TP}{TP + FP}$$

The recall is the number of examples correctly classified from that specific class divided by the total number of examples that should have been classified from that class. Its mathematical equation can be examined below.

$$Recall = \frac{TP}{TP + FN}$$

The f-measure is calculated using the precision and recall computed values. The formula which represents F-Measure is

$$F - Measure = \frac{2 \times precision \times recall}{precision + recall}$$

The confusion matrix was also generated in order to analyze which classes the classifiers were having more trouble with. The confusion matrix evidences how many examples of each class were correctly classified and for which classes the examples were misclassified. One dimension of the matrix represents the predicted classification and the other dimension refers to the correct classification, the ground truth. The correct predictions reside in the confusion matrix diagonal.

After calculating all these metrics for each fold, the mean was calculated among the folds of the same noise range to get the average performance of the machine learning algorithm according to the specific noise percentage.

Having this information, a depth investigation in relation to the behavior of each machine learning algorithm in the most varied scenarios is made possible.

4.4 Results and discussion

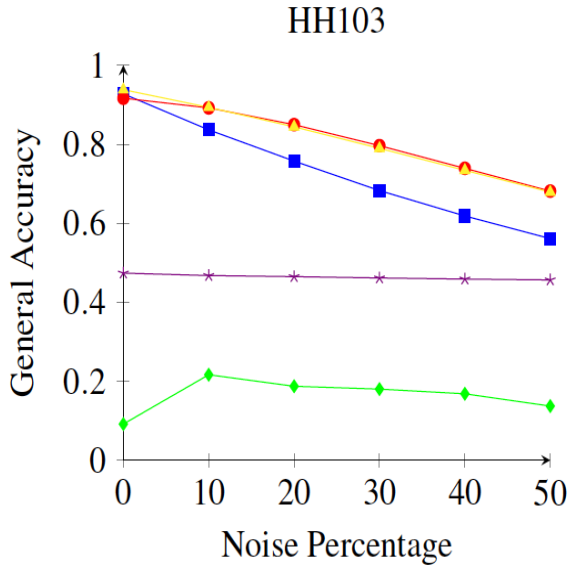
The machine learning techniques which had the best outputs in the noise-free scenario were Random Forest, *K*-NN and Decision Tree. Naive Bayes and SVM did not have good results.

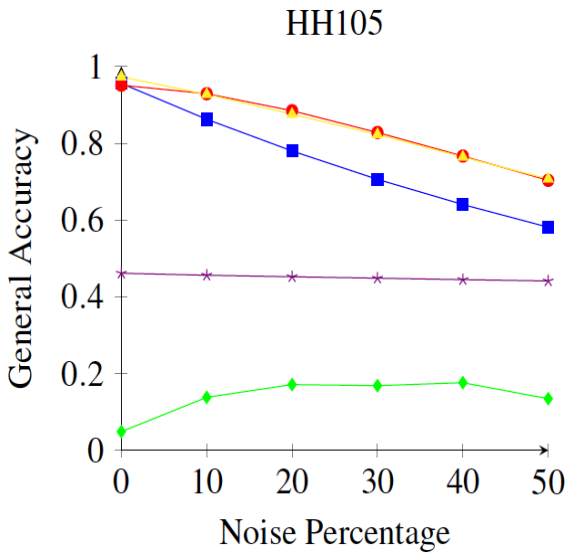
Figure 4.3 illustrates the general accuracy value of each dataset in relation to the increase in the noise range. The X-axis in each plot represents the noise percentage inserted in the training set and the Y-axis consists of the general accuracy obtained in a range of 0 to 1 (0 meaning 0% and 1 meaning 100%). Each line in the line chart stands for a different machine learning technique. The blue line represents the Decision tree, the red line appears as the *K*-NN, the yellow line act in place of the Random Forest, the green line corresponds to the Naive Bayes and the purple line represents the SVM technique. Each marker on top of the lines represents the general accuracy mean among the 5 folds.

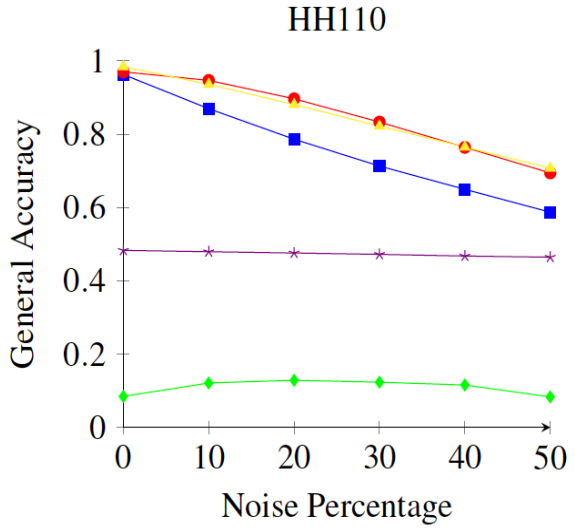
From the figure, it can be inferred that for the most machine learning techniques, the general accuracy decreases with the noise range insertion increase, except the Naive Bayes technique which had its general accuracy increased with the increment of noise insertion in some cases.

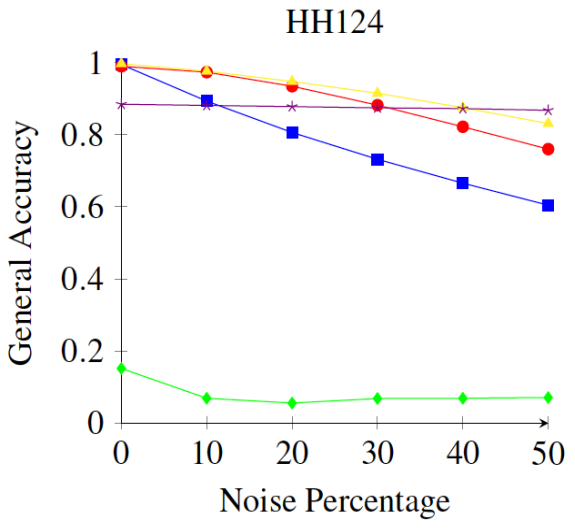
Another interesting fact to point out is that SVM, even though did not perform very well in most scenarios, was the algorithm which suffered less impact from the noise insertion. This can be explained by SVM robustness. The databases in which SVM had the best general accuracy happens to be the smaller ones, HH124, Kyoto 2008 and Kyoto 2009 Spring.

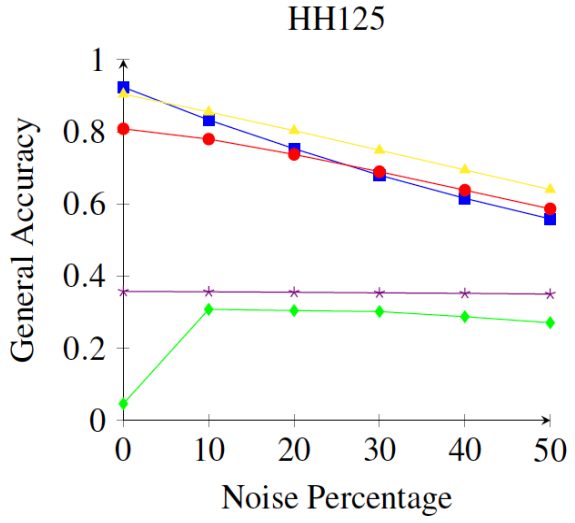
Although the *K*-NN and Random Forest obtained very similar results in a noise-free scenario, it is noticeable that in most of the datasets, the Random Forest achieves better results in the 50% noise scenario.

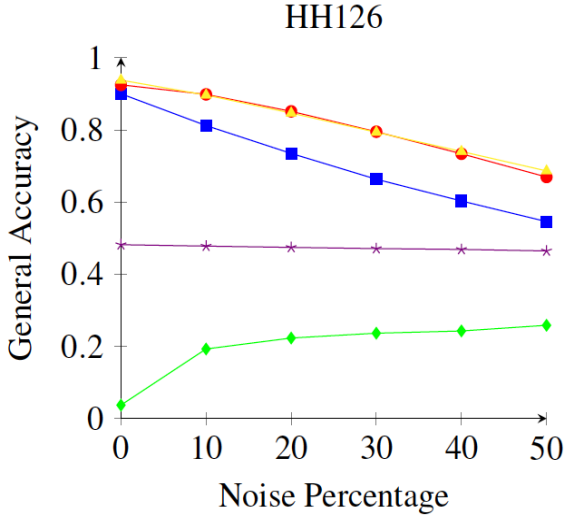

HH103
General Accuracy
Noise Percentage
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50


HH105
General Accuracy
Noise Percentage
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50


HH110
General Accuracy
Noise Percentage
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50


HH124
General Accuracy
Noise Percentage
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50


HH125
General Accuracy
Noise Percentage
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50


HH126
General Accuracy
Noise Percentage
0
0.2
0.4
0.6
0.8
1
0
10
20
30
40
50

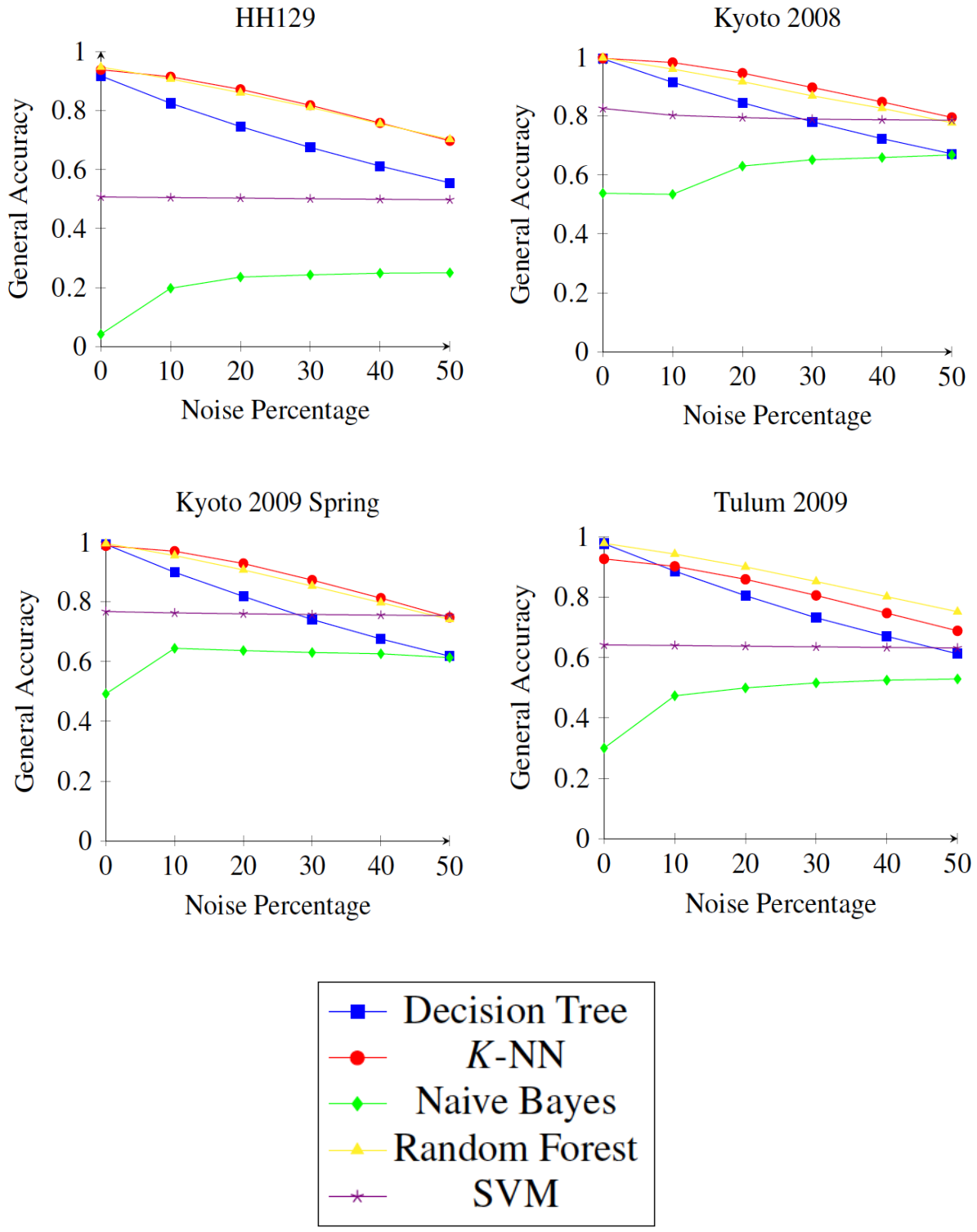


Figure 4.3: General Accuracy per noise range plots

Figure 4.4 shows the other metrics calculated from the Kyoto 2009 Spring dataset.

The top left sub-figure shows the accuracy against the noise percentage. All the machine learning algorithms scored high in this metric even with the noise insertion. Since there was not an expressive change between the algorithms and the noise percentage, this metric is not relevant for further comparisons. I believe this metric is biased. It is calculated in relation to each class and it considers everything that is not from that class as true negative even though the label is different from the ground truth. The accuracy takes into consideration the value of true negatives helping the final high accuracy value.

The top right sub-figure of Figure 4.4 represents the precision according to the different noise percentages. It can be analyzed from this sub-figure that the *K*-NN, Random Forest, and Decision tree had the best scores in a noise-free scenario. When looking at the 50% noise scenario the decision tree had the worst result, the noise insertion really influenced its performance. The SVM had its precision constant among all the noise ranges and the Naive Bayes suffered little effect from noise after the 20% noise range.

The bottom left sub-figure of Figure 4.4 correspond to the recall metric. The line chart is very similar to the precision one. However, the recall was not as affected by the noise insertion as the precision was. This can be observed just by comparing the values obtained in the 50% scenario in both charts. The SVM recall was smaller and maintained constant from all the noise ranges. Unlike the precision line chart, the SVM recall was worse than the *K*-NN, Decision Tree, and Random Forest even on the 50% scenario. The same can not be seen in the precision plot. Where the SVM precision in the 50% noise scenario was the best among the other algorithms.

The bottom right sub-figure of Figure 4.4 reproduces the F-Measure behavior according to the different noise scenarios. Since the F-Measure takes into consideration the precision and the recall it is reasonable that the plot is a mix of the other two. One difference interesting to point out is that the SVM F-Measure in the 50% scenario was better than the Decision Tree but worse than the Random Forest and *K*-NN.

Table 4.3 illustrates the maximum general accuracy achieved in the noise-free scenario and 50% noise scenario in all datasets. The Random Forest is the machine learning technique which has the best results in most of the datasets in both scenarios. All the datasets have a high general accuracy in the noise-free scenario. The maximum achieved in the noisy free scenario was 0.996 by the Random Forest in the Kyoto 2008 dataset which happens to be the dataset with fewer activities and the one with the smallest number of sensor events. The dataset with the smallest maximum general accuracy was the HH125, the one with the highest number of activities. In the 50% scenario, the highest general accuracy was achieved by SVM in the HH124 dataset. And the smallest maximum general accuracy between the datasets in the 50% scenario was 0.639 scored by the SVM in the HH125 dataset. It can be concluded based on this table that even though the data suffered a noise insertion of 50% the general accuracy was affected but not as much as expected. If the machine learning module suffered an attack the malicious program could still achieve an accuracy of 0.639 minimum and 0.867 maximum. The value 0.867 is a very good general accuracy, the attacker could recognize most of the activities generated by the resident, highlighting the danger still present after the noise insertion.

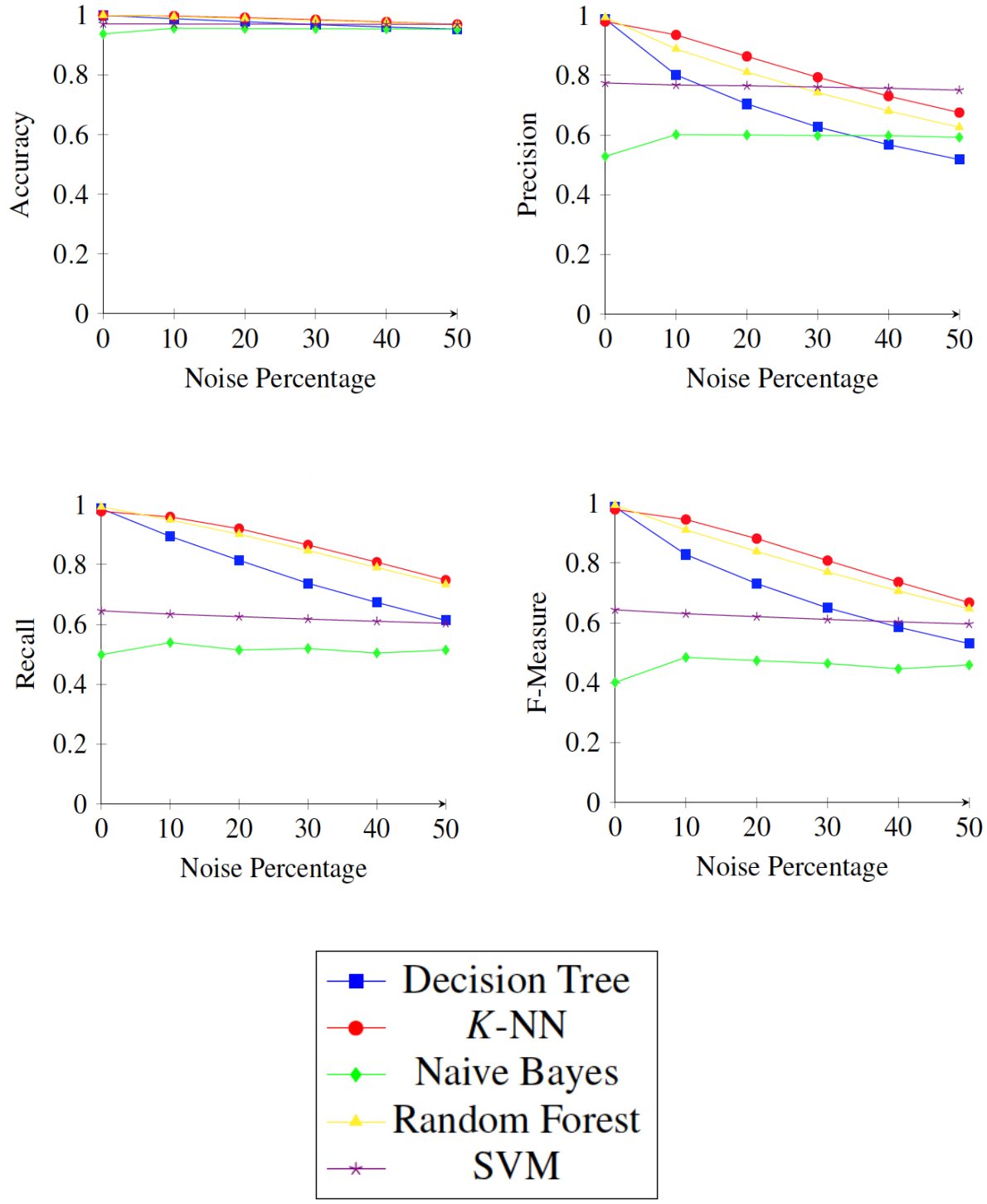


Figure 4.4: Accuracy, Precision, Recall and F-Measure of Kyoto 2009 Spring dataset

Dataset	**0% Noise**	**50% Noise**
HH103	0.938 (RF)	0.681 (*K*-NN)
HH105	0.973 (RF)	0.706 (RF)
HH110	0.984 (RF)	0.707 (RF)
HH124	0.996 (RF)	0.867 (SVM)
HH125	0.923 (DT)	0.639 (RF)
HH126	0.937 (RF)	0.686 (RF)
HH129	0.947 (RF)	0.702 (RF)
Kyoto 2008	0.996 (RF)	0.794 (*K*-NN)
Kyoto 2009 Spring	0.994 (RF)	0.753 (SVM)
Tulum 2009	0.978 (RF)	0.751 (RF)

Table 4.3: Maximum general accuracy per dataset, 0% and 50% noise rate

CHAPTER 5

Conclusion

This work's main objective was to analyze the performance of machine learning techniques in the context of recognizing activities on smart homes against several noise insertion scenarios.

The datasets were obtained from the CASAS project. A total number of 10 datasets were selected. They consist of sensor streams and represent the routine of the residents in each apartment. The data was not scripted, it corresponds to the real-world scenario.

The activity recognition was based on a sensor windowing approach. The features were extracted from each window and each training set was submitted to noise insertions equivalent to the percentages of 10%, 20%, 30%,40% and 50%. The model was trained with all the noise percentages, including the noise-free scenario and tested with the accurate data. The machine learning models selected were Random Forest, Naive Bayes, Decision Tree, SVM and *K*-NN. The result obtained from the machine learning module was the predicted class.

The metrics chosen to evaluate the machine learning techniques performance according to the noise percentages were general accuracy, accuracy, precision, recall, f-measure, and confusion matrix.

The ideal result would be if the machine gets a big amount of activities right in a noise-free scenario and that it miserably get almost all the activities wrong in the noisy scenarios.

The results obtained have shown that inserting noise on the datasets did not appear to be as efficient in order to mask the resident's legitimate activities. Even though the data was manipulated, the general accuracy rate can still be considered high in some of the databases on a 50% scenario. This proves that this approach is not sufficient and did not assure the security and privacy promised.

Future works intend to test different window sizes to check if this would affect the results, consider more machine learning techniques such as Multi-Layer Perception, as well as investigate the reason why SVM did not achieve good results in almost all cases even though it excels in most of the machine learning problems, and examine whether the attribute noise insertion would have superior results or not in comparison to the class noise insertion implied in this work.

APPENDIX A

Appendix

Class	Count	Class	Count	Class	Count	Class	Count
Bathe	3483	Eat_Breakfast	1430	Other	27502	Wash_Breakfast_Dishes	2188
Bed_Toilet_Transition	4075	Eat_Dinner	2010	Personal_Hygiene	14589	Wash_Dinner_Dishes	3427
Cook	509	Eat_Lunch	1207	Phone	1221	Wash_Dishes	166
Cook_Breakfast	11031	Enter_Home	5355	Read	1280	Wash_Lunch_Dishes	4233
Cook_Dinner	18680	Evening_Meds	868	Relax	1710	Watch_TV	8270
Cook_Lunch	10687	Exercise	124	Sleep	11859	Work_At_Table	1971
Dress	3451	Leave_Home	5240	Sleep_Out_Of_Bed	1127		
Eat	10	Morning_Meds	1075	Toilet	18376		

Table A.1: HH103 Activity Count

Class	Count	Class	Count	Class	Count	Class	Count
Bathe	4030	Eat_Breakfast	1250	Morning_Meds	2237	Wash_Breakfast_Dishes	1419
Bed_Toilet_Transition	1872	Eat_Dinner	1075	Other	73867	Wash_Dinner_Dishes	1774
Cook	15437	Eat_Lunch	1548	Personal_Hygiene	9893	Wash_Dishes	7783
Cook_Breakfast	11006	Enter_Home	4918	Phone	1994	Wash_Lunch_Dishes	4531
Cook_Dinner	8065	Entertain_Guests	1197	Relax	10818	Watch_TV	5241
Cook_Lunch	5584	Evening_Meds	753	Sleep	6133	Work	2123
Dress	6250	Groom	1589	Sleep_Out_Of_Bed	4759	Work_At_Table	6502
Eat	1547	Leave_Home	4820	Toilet	15777		

Table A.2: HH105 Activity Count

Class	Count	Class	Count	Class	Count	Class	Count
Bathe	1229	Enter_Home	653	Relax	18212	Work_At_Table	10784
Bed_Toilet_Transition	1677	Evening_Meds	2436	Sleep	15421	Work_On_Computer	15462
Cook_Breakfast	2368	Groom	2106	Sleep_Out_Of_Bed	3593		
Cook_Dinner	225	Leave_Home	586	Take_Medicine	4582		
Dress	5487	Morning_Meds	2464	Toilet	11389		
Drink	81	Other	30297	Wash_Breakfast_Dishes	2617		
Eat_Breakfast	1359	Personal_Hygiene	3876	Wash_Dinner_Dishes	110		
Eat_Dinner	316	Read	640	Work	332		

Table A.3: HH110 Activity Count

Class	Count	Class	Count	Class	Count
Bed_Toilet_Transition	737	Leave_Home	1825	Wash_Dinner_Dishes	32
Cook_Dinner	216	Other	60198	Wash_Dishes	37
Dress	179	Personal_Hygiene	147	Watch_TV	364
Eat_Breakfast	84	Phone	146	Work	60
Eat_Dinner	125	Sleep	1478	Work_At_Table	56
Enter_Home	1228	Sleep_Out_Of_Bed	357	Work_On_Computer	385
Entertain_Guests	6626	Toilet	1252		
Groom	397	Wash_Breakfast_Dishes	66		

Table A.4: HH124 Activity Count

Class	Count	Class	Count	Class	Count	Class	Count
Bathe	1296	Eat_Breakfast	366	Other	21809	Toilet	10525
Bed_Toilet_Transition	1099	Eat_Dinner	418	Personal_Hygiene	14494	Wash_Breakfast_Dishes	5235
Cook	1672	Eat_Lunch	364	Phone	1514	Wash_Dinner_Dishes	8143
Cook_Breakfast	10530	Enter_Home	6086	Read	640	Wash_Dishes	17700
Cook_Dinner	30095	Entertain_Guests	13995	Relax	542	Wash_Lunch_Dishes	1797
Cook_Lunch	9129	Evening_Meds	1334	Sleep	2658	Watch_TV	7251
Dress	5718	Groom	18133	Sleep_Out_Of_Bed	2776	Work	718
Drink	6314	Leave_Home	2971	Step_Out	1159	Work_On_Computer	7187
Eat	92	Morning_Meds	1328	Take_Medicine	1138		

Table A.5: HH125 Activity Count

Class	Count	Class	Count	Class	Count	Class	Count
Bathe	508	Eat_Breakfast	1979	Other	62864	Wash_Breakfast_Dishes	2578
Bed_Toilet_Transition	391	Eat_Dinner	721	Personal_Hygiene	8689	Wash_Dinner_Dishes	804
Cook	402	Eat_Lunch	753	Phone	1958	Wash_Dishes	3328
Cook_Breakfast	8726	Enter_Home	6148	Read	4430	Wash_Lunch_Dishes	1118
Cook_Dinner	7234	Entertain_Guests	5222	Relax	195	Watch_TV	11667
Cook_Lunch	5123	Evening_Meds	1060	Sleep	5406	Work_On_Computer	7891
Dress	7763	Groom	6315	Sleep_Out_Of_Bed	311		
Drink	6018	Leave_Home	7282	Step_Out	1106		
Eat	92	Morning_Meds	604	Toilet	11099		

Table A.6: HH126 Activity Count

Class	Count	Class	Count	Class	Count	Class	Count
Bathe	508	Eat_Breakfast	1979	Other	62864	Wash_Breakfast_Dishes	2578
Bed_Toilet_Transition	391	Eat_Dinner	721	Personal_Hygiene	8689	Wash_Dinner_Dishes	804
Cook	402	Eat_Lunch	753	Phone	1958	Wash_Dishes	3328
Cook_Breakfast	8726	Enter_Home	6148	Read	4430	Wash_Lunch_Dishes	1118
Cook_Dinner	7234	Entertain_Guests	5222	Relax	195	Watch_TV	11667
Cook_Lunch	5123	Evening_Meds	1060	Sleep	5406	Work_On_Computer	7891
Dress	7763	Groom	6315	Sleep_Out_Of_Bed	311		
Drink	6018	Leave_Home	7282	Step_Out	1106		
Eat	92	Morning_Meds	604	Toilet	11099		

Table A.7: HH129 Activity Count

Class	Count	Class	Count	Class	Count	Class	Count	Class	Count
Cook	3272	Eat	278	Other	5379	Relax	1078	Sleep	10916

Table A.8: Kyoto 2008 Activity Count

Class	Count	Class	Count	Class	Count
Cleaning	1316	R1_work_at_dining_room_table	4407	R2_sleep	12044
R1_bed_to_toilet	1874	R2_bed_to_toilet	1961	R2_watch_TV	10038
R1_breakfast	5681	R2_breakfast	8692	R2_work_at_computer	8570
R1_groom	5313	R2_groom	7434	Wash_bathtub	219
R1_sleep	13715	R2_prepare_dinner	17257		
R1_work_at_computer	32907	R2_prepare_lunch	6582		

Table A.9: Kyoto 2009 Spring Activity Count

Class	Count	Class	Count	Class	Count
Cleaning	1316	R1_work_at_dining_room_table	4407	R2_sleep	12044
R1_bed_to_toilet	1874	R2_bed_to_toilet	1961	R2_watch_TV	10038
R1_breakfast	5681	R2_breakfast	8692	R2_work_at_computer	8570
R1_groom	5313	R2_groom	7434	Wash_bathtub	219
R1_sleep	13715	R2_prepare_dinner	17257		
R1_work_at_computer	32907	R2_prepare_lunch	6582		

Table A.10: Tulum Activity Count

Bibliography

[1] MavHome: Managing an Adaptative Versatile Home. http://ailab.wsu.edu/mavhome/.

[2] The PlaceLab. http://web.mit.edu/cron/group/house_n/placelab.html.

[3] Aware Home Research Initiative. http://www.awarehome.gatech.edu.

[4] CASAS: Center for Advanced Studies in Adaptive Systems. http://casas.wsu.edu.

[5] Krishnan, Narayanan C. ; Cook, Diane J. Activity recognition on streaming sensor data. *Pervasive and Mobile Computing*, 10:138–154, February 2014.

[6] Mark Weiser. The computer for the 21 st century. *Scientific american*, 265(3):94–105, 1991.

[7] Friedemann Mattern and Christian Floerkemeier. From the internet of computers to the internet of things. *From active data management to event-based systems and more*, pages 242–259, 2010.

[8] Terence KL Hui, R Simon Sherratt, and Daniel Díaz Sánchez. Major requirements for building smart homes in smart cities based on internet of things technologies. *Future Generation Computer Systems*, 76:358–369, 2017.

[9] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P Sheth. Machine learning for internet of things data analysis: A survey. *Digital Communications and Networks*, 2017.

[10] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials*, 17(4):2347–2376, 2015.

[11] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1(2011):1–11, 2011.

[12] Ziv Katzir and Yuval Elovici. Quantifying the resilience of machine learning classifiers used for cyber security. *Expert Systems with Applications*, 2017.

[13] Patrick McDaniel, Nicolas Papernot, and Z Berkay Celik. Machine learning in adversarial settings. *IEEE Security & Privacy*, 14(3):68–72, 2016.

[14] Huang Xiao, Battista Biggio, Blaine Nelson, Han Xiao, Claudia Eckert, and Fabio Roli. Support vector machines under adversarial label contamination. *Neurocomputing*, 160:53–62, 2015.

[15] Rui Zhang and Quanyan Zhu. Secure and resilient distributed machine learning under adversarial environments. In *Information Fusion (Fusion), 2015 18th International Conference on*, pages 644–651. IEEE, 2015.

[16] Ling Huang, Anthony D Joseph, Blaine Nelson, Benjamin IP Rubinstein, and JD Tygar. Adversarial machine learning. In *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pages 43–58. ACM, 2011.

[17] Francisco-Javier González-Serrano, Ángel Navia-Vázquez, and Adrián Amor-Martín. Training support vector machines with privacy-protected data. *Pattern Recognition*, 72:93–107, 2017.

[18] García, Salvador ; Luengo, Julián ; Herrera, Francisco. *Data Preprocessing in Data Mining*. Springer, 2015.

[19] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[20] T.M. Mitchell. *Machine Learning*. McGraw-Hill international editions - computer science series. McGraw-Hill Education, 1997.

[21] Krishnan, Narayanan C. ; Cook, Diane J. *Activity Learning:Discovering, Recognizing, and Predicting Human Behavior from Sensor Data*. Wiley, 2015.

[22] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems. *J. Mach. Learn. Res*, 15(1):3133–3181, 2014.