Forgetting Methods for White Box Learning

Anthony D'Amato[1] and Matthieu Boussard[1]

Craft ai[1], Paris, France
{anthony.damato,matthieu.boussard}@craft.ai

Abstract. In the Internet of Things (IoT) domain, being able to propose a contextualized and personalized user experience is a major issue. The explosion of connected objects makes it possible to gather more and more information about users and therefore create new, more innovative services that are truly adapted to users. To attain these goals, and meet the user expectations, applications must learn from user behavior and continuously adapt this learning accordingly. To achieve this, we propose a solution that provides a simple way to inject this kind of behavior into IoT applications by pairing a learning algorithm (C4.5) with Behavior Trees. In this context, this paper presents new forgetting methods for the C4.5 algorithm in order to continuously adapt the learning.

Keywords: Learning, Forgetting, Decision Trees, Classification, IoT, Data Mining, C4.5

1 Introduction

In the Internet of Things (IoT) domain, services must be more intelligent, to the extent that proposing a personalized service to the user has become one of the principal challenges; or to put it another way, ensuring that applications give the user an impression of uniqueness, by learning his/her behavior and acting in consequence, has become a primary objective. Moreover, as an end user never has the same behavior over the course of a year, a service that learns a specific characteristic of his/her behavior must be capable of adapting. In this domain, smart devices must quickly provide the service they claim to provide, otherwise they will have no utility in the eyes of their user, which reinforces the idea that the learning method offered within a service must rapidly satisfy end user needs.

This paper proposes an adaptive learning method for developers with a *White Box* approach. The *White Box* learning notion is a fully controllable, predictable, powerful and understandable learning algorithm, that does not just deliver an output depending on an input but also visually displays how this output was found. In contrast, *Black Box* algorithms remain very powerful, but are harder to explain. For instance, in Deep Learning algorithms, understanding how the weight of an artificial neuron will impact the output can be tricky. For the *White Box* approach, we choose the Behavior Trees (BTs) [4] mechanism widely used for Artificial Intelligence in video games [10] which are well suited to the characteristics of the *White Box* (Table 1) [8]. Introducing learning in BTs should not break this *White Box* approach. For instance, Q-Learning algorithms can be integrated in BTs[7, 6] but this integration is too *Black Box* by nature and has a very slow convergence. Learning algorithms that are quite capable of such characteristics are classifiers and more precisely algorithms inducing Decision Trees because they provide an easy way to read the rules generated. We chose the famous C4.5 classifier algorithm [13] to introduce learning in BTs while maintaining a *White Box* approach. Figure 1 shows the integration of learning in an AI designed by a Behavior Tree. All the learning capacity is concentrated in a limited part of the BT and as we use a classifier inducing a Decision Tree, this part also provides a graphical view.

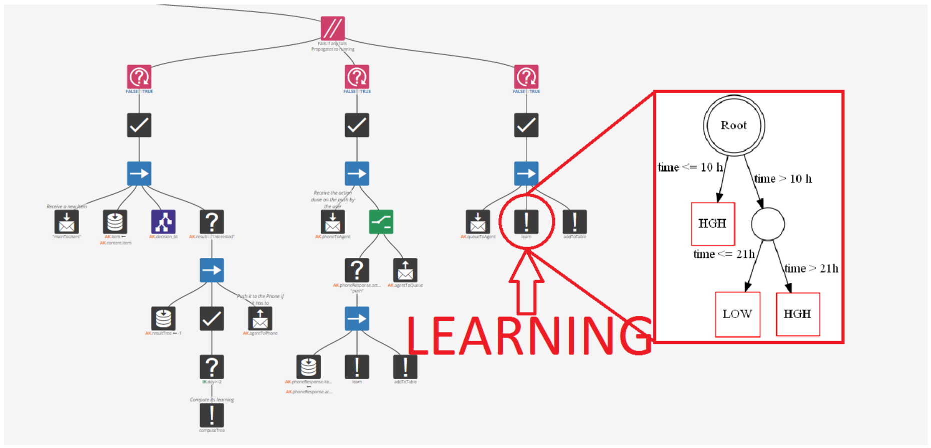

Fig. 1. Integration of learning in a Behavior Tree

This paper is organized as follows: Section 2 explains the choice of the C4.5 algorithm among other classifiers and adds some details on the algorithm itself; 3 proposes new forgetting methods to add adaptability to C4.5; and 4 presents an evaluation of each of these methods and their performance.

	State Machines	Fuzzy Logic/ Markov Systems	Goal-Oriented Behavior	Rule-Based Systems	Behavior Trees
Simplicity	++	-	+	-	++
Separation of Game Design and Programming	++	+	+	++	++
Flexibility	-	+	+	+	++
Behavior Quality	-	+	+	++	++
Efficiency	++	-	-	++	+
Conclusion	Very Simple	Probabilistic	Great for planning	Limits not reached yet	High flexibility

Table 1. Comparison of decision-making techniques.

2 Related Work

As previously mentioned, our research into a *White Box* learning led us to choose the classification algorithm C4.5. This algorithm induces a Decision Tree from a training set and is based on the information gain concept. It was identified as one of the top 10 algorithms in Data Mining [15] and other work led by Gracia and Herrera [9] used statistical tests over 30 database to show that the C4.5 algorithm is the best classifier, when compared to CN2, Naive Bayes, 1-Nearest Neighbors and a Kernel classifier.

Let D be the training set composed of samples x_i for the C4.5 algorithm such that:

$$D = \{x_1, x_2, \ldots, x_n\} \tag{1}$$

The samples are themselves composed of a set of predictive variables A named *attributes* and a predicted variable Y, that can have multiple values named *classes*. For example with T attributes as the input, the first sample is defined by:

$$x_1 = A_1 \cup \{y_1\} \ with \ A_1 = \{a_{1,1}, a_{1,2}, \ldots, a_{1,T}\}$$

In the above $a_{i,j}$ is the value of the j^{th} attribute of the i^{th} sample and y_j the predicted output value of the i^{th} sample.

To choose the right algorithm, which would form the foundation of our learning algorithm, we had three main criteria. The learning algorithm must have the following attributes: the *White Box* property, by inducing decision trees (2.1); the ability to handle multiple attribute values and continuous attributes (2.2); the ability to deal with dynamic training sets (2.3).

2.1 A *White Box* Learning

Since BTs are *White Box*, adding learning to them must keep this characteristic and therefore, the rules induced by the learning algorithm must be understandable. The first methods we considered were the Case Based Reasoning algorithms. Case Based Reasoning [11] is a process that aims to solve new problems using a database of solutions. Another option we considered was Boosting Algorithms, which are based on the creation of highly accurate prediction rules thanks to the combination of weak and inaccurate rules. One of the most famous and the most used is the AdaBoost algorithm introduced by Freund and Schapire in 1995[14]. Yet, neither Case based Reasoning nor Boosting algorithms provide simple visual output, something that Decision Trees do very well indeed.

2.2 Multiple attribute values and continuous attributes

A few years before the creation of the C4.5 algorithm, CART [3] was published. This algorithm handles both numerical and categorical variables and induces both a Decision Tree and a Regression Tree. Therefore, CART can propose continuous outputs thanks to this Regression Tree. However, CART is not adapted to attributes with multiple values because it creates binary trees, whereas C4.5 is well adapted for this. Indeed, if an attribute has more than two possible values the tree induced by C4.5 will have a branch for each possible value.

The predecessor of C4.5, ID3 (Iterative Dichotomiser 3) [12] induces also a Decision Tree from a training set but it was very hard to use on continuous data because searching for the best split was time consuming. The handling of continuous attributes by C4.5 was one of the numerous improvements on ID3.

2.3 Data Streams

C4.5 is used for the creation of Decision Trees from a static database. However, in our application the training set changes every time a new sample from the end users behavior arrives, which is a situation that is similar to Data Stream Mining [2]. A Data Stream consists of a sequence of data items arriving at high frequency, generated by a process that is subject to unknown changes. One of the major priorities of Data Stream Mining is its ability to adapt to these changes by predicting the goal or setting up forgetting methods on the incoming data. As in our application of learning there is a constant follow-up of the end user, and the use of C4.5 algorithm must be equivalent to a Data Stream Mining by applying forgetting methods without this data rate constraint, but with the problem of constant adaptation to the incoming information.

3 Forgetting Methods

As the end user behavior is variable, the learning must be able to adapt efficiently. However, the C4.5 algorithm is not suited to fast adaptation because, in order to counterbalance previous training samples, the algorithm needs as many new cases as previously encountered. For instance, if the training set is composed of 50 days of constant behavior no behavior variation during 50 days and if the user decides suddenly to change all his previous decisions, then it will take another 50 days for C4.5 to induce a tree with the new user behavior. Indeed the samples from the first 50 days give more information about the user behavior than the new ones because they are more numerous. And as C4.5 is based on information gain measurement, the first behavior is still induced by the algorithm. Therefore, in this section several methods are proposed to adapt the learning to variations. This paper proposes two new methods for forgetting, namely the *Random Forgetting*(3.2) method and the *Leaf Forgetting*(3.3) method.

3.1 Sliding Window

This method is based on the current algorithms used in the application of Clustering Algorithms on Data Stream [5] which consists in applying a Sliding Window to the incoming stream. This method makes it possible to remember all the recent events that occurred inside this window.

For the Sliding Window method, a maximum length N for the training set is chosen and if this upper limit is reached, every time a new sample is added the oldest (x_1) is deleted. Let z be a new sample, if $Card(D) = N$:

$$D = (D \setminus \{x_1\}) \cup \{z\} \tag{2}$$

3.2 Random Forgetting

For the newly introduced Random Forgetting method we define a maximum size N for the training set and once this ceiling is reached, every new case added to the data set leads to the random deletion of a stored case. Given the function $rand(N)$ picking randomly an integer between 0 and $N-1$ and a new sample z, if $Card(D) = N$:

$$D = \left(D \setminus \left\{x_{rand(N)}\right\}\right) \cup \{z\} \tag{3}$$

The approach with this method is, compared with the Sliding Window, to introduce an unpredictable deletion in the training set in order not to forget necessary the oldest samples, which could be significantly important for the learning. It allows to remember important events relatively spaced in time.

3.3 Leaf Forgetting

The second method introduced in this paper is termed Leaf Forgetting. Here, each sample in the training set has at its disposal a weight, named w.

Once the tree is induced, the training set can be partitioned in accordance with the leaves of this tree. Let L_i be the set of training samples reaching the *i-th* leaf and k the total number of leaves, then D can be defined as follows:

$$D = \bigcup_{i=1}^{k} L_i \tag{4}$$

When a new sample is joined to the training set, it checks in which leaf of the previous tree this sample is arriving. All the data stored in this specific leaf of the tree have their weights increased by an update function f. Then if the weight of a case is higher than a maximum w_{max}, it is deleted from the training set, which can be defined by:

Let y be a new sample for the data set:

$$y \in L_i \Rightarrow \forall x \in L_i,\ w_x = f(w_x) \tag{5}$$

Let O be the set in which are the samples that must be deleted from the training set defined by:

$$O = \{z, w_z > w_{max}\} \tag{6}$$

Then the new training set is defined by:

$$D = D \backslash O \tag{7}$$

For a fast adaptation, it is important that no cluster be strongly preponderant on others and this is where this method has an advantage over others because increasing weights in each leaf favors over time a balance between learned behaviors.

4 Results

During this experiment, agents designed with BTs control the temperature in each room of an inhabited house and learn the temperature the user desires. For this simulation we focus on one room.

The available data are the outdoor temperature (*OutTemp*), the user presence (*Presence*), the time (*Time*)and whether or not the user is working on this day (*Working*). The temperature (*Temp*) in a room has 3 values: *"Low"*, *"Medium"* and *"High"*. Each time the user decides to change the temperature, the state of each sensor plus the desired action can be added to the training set:

$$A = \{\mathit{OutTemp},\ \mathit{Presence},\ \mathit{Time},\ \mathit{Working}\}$$

and

$$Y = \{\mathit{Temp}\}$$

Furthermore, every 10 minutes the agent learns the current situation but also computes the tree previously induced and applies its decision for the temperature in the room. If the end user is not satisfied with this decision and changes the temperature to a preferred level, then the agent does not make any further decisions until its decision is the same as the users. These user interactions are recorded for the experiment. Then, when the day is over, we apply a forgetting algorithm and C4.5 is computed on the current training set.

The metric used to measure the efficiency of our smart agent is the number of user actions that occur during the simulation. This measure shows the number of actions an end user has to perform in order that the agent acts on his/her behalf. Indeed, the fewer interactions the user has to perform, the more efficient the learning adaptation.

4.1 Adaptation speed

As a user can rapidly change his/her behavior, the learning algorithm must be able to adapt quickly to match the new behavior. The speed of adaptation is a strong criteria for forgetting because it is necessary to limit the number of human actions with the device that are required for it to suit his preferences. Performance in terms of speed for each forgetting method is evaluated in the following simulation:

This simulation (Figure 2) lasts 100 days during which the user has a constant behavior for 50 days and then inverses completely his previous actions if the user preferred a *"High"* level, he switches to *"Low"* and vice versa. The simulated user behavior is the following: the user is working from Monday to Friday and is not working at the weekend. His/her behavior changes, depending on whether the current day is a working day, or at the weekend. The user likes to sleep in a warm room and turn off the heating when he/she goes to work. At the weekend, when the user is at home, he/she likes to set the temperature at a medium level. For this first experiment the input of the classification algorithm is limited to:

$$A = \{Presence,\ Time,\ Working\}$$

and

$$Y = \{Temp\}$$

During this experiment five simulations are launched:

- Without any forgetting method
- Using a Sliding Window with $N = 720$ ($\simeq$ 5 days)
- Using Random Forgetting with $N = 720$ ($\simeq$ 5 days)
- Using Random Forgetting with a length of 20 days $N = 2900$ ($\simeq$ 20 days)
- Using Leaf Forgetting with $f(w) = w + 1$ and $w_{max} = 100$

Results As we can see in the Figure 2, without a forgetting algorithm, the number of human actions is not only high but it also takes a long time to learn new behavior. The explanation for this very long adaptation is that the training set has accumulated a lot of samples about the user behavior during the 50 first days, which produces a decision tree that is difficult to modify. Indeed, as the

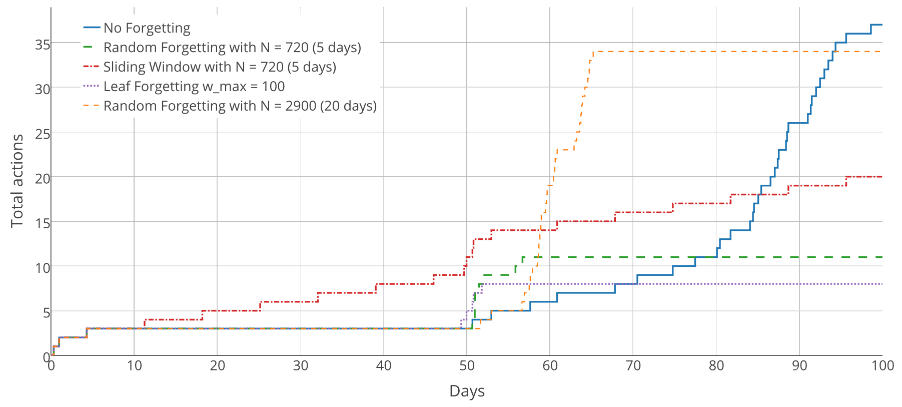

Fig. 2. Total number of user actions over a period of 100 days with a behavior change at the 50th day (the lower the better).

algorithm keeps all samples, it needs as many new samples about the inverse behavior as the samples accumulated during 50 days.

The Sliding Window is strongly dependent of its length: if it is small, its adaptation is fast but it forgets past actions like the behaviors during the weekends. Indeed, as we can see in Figure 2, every 7 days the user changes the temperature. This is due to the Sliding Window length that is less than five days, then at the end of the working interval, the weekend behavior is totally forgotten and must be learned again. With a larger size the Sliding Window would have remembered older events but would have had slower adaptation.

Random Forgetting is also dependent on N, indeed as we can see, the smaller N , the faster it becomes. This is explained by the fact that the smaller N is the higher the probability deletion of an old undesirable behavior becomes, and consequently it takes less time to learn a new behavior. For Random Forgetting with $N = 2900$ ($\simeq$ 20 days), as N is high, the number of samples representing the old behavior takes time to decrease and therefore a lot of user actions are necessary to delete this previous behavior.

Leaf Forgetting has the best performance in this experiment and therefore has the fastest adaptation of all methods. Increasing weights only in the leaves of the current behavior leads to explicitly forgetting targeted behavior and therefore makes it possible to rapidly counter the previous behaviors the user had. Through this forgetting method it is possible to balance the number of samples in each leaf induced by C4.5 and thus, it ensures that few leaves are preponderant over others.

4.2 Memorization capacity

The device continuously learning the user behavior must not only have fast adaptation, but must also keep in memory events likely to occur a long time after they first occur. For example, in summer there are no low temperature levels equivalent to those likely to occur in winter, and therefore, even if the learning has to be adapted to the current season, it must not forget what it has learned during the previous season when adapting to the current one. Performance concerning the memorization of each forgetting methods is evaluated in the following simulation:

This simulation (Figure 3) lasts 2 years during which the user adapts the inside temperature in accordance with the outdoor temperature [1]. For this experiment, the outdoor temperature is added in the input:

$$A = \{OutTemp,\ Presence,\ Time,\ Working\}$$

and

$$Y = \{Temp\}$$

During this experiment, three simulations are launched:

- Using Random Forgetting with $N = 720$ ($\simeq$ 5 days)
- Using Random Forgetting with a length of 20 days $N = 2900$ ($\simeq$ 20 days)
- Using Leaf Forgetting with $f(w) = w + 1$ and $w_{max} = 100$

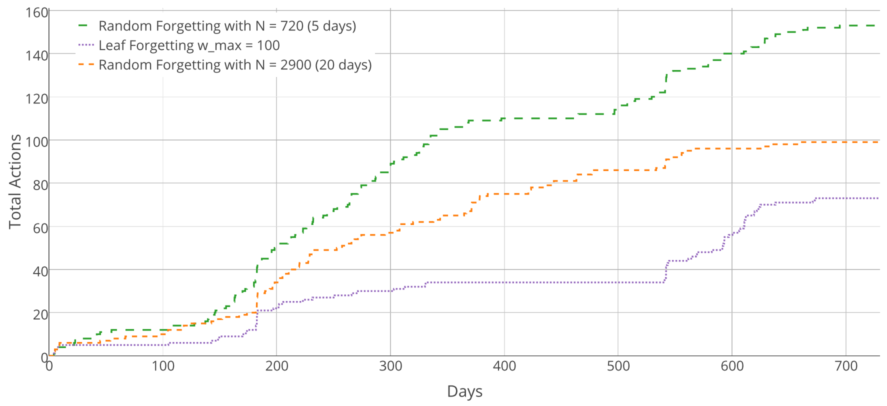

Fig. 3. Total number of user actions over a two years period considering outdoor temperature (the lower the better).

Results As previously mentioned, the Random Forgetting method is strongly dependent on its length and this simulation highlights this constraint. In addition, as we can see,during the first year the Leaf Forgetting is faster in learning the user behavior than the Random Forgetting. This is related to the results concerning the first experiment and the fast adaptation of Leaf Forgetting.

What is interesting in this simulation is to see that Leaf Forgetting has a strong control of the house from the end of the first year to the middle of the second year, a domain where the Random Forgetting flunks. But on the 545^{th} day, it has forgotten the previous year. This can be explained by the deletion of the leaves corresponding to the behavior on the 180^{th} day. Indeed, the number of samples in the corresponding leaves is not high enough to be relevant compared to all the new samples accumulated during the following year and these leaves are no longer induced.

4.3 Noise resistance

Sometimes the user can have a different behavior for a few moments, or some disturbances can occur, such as the interaction with someone unfamiliar with the house. These kinds of events are isolated and must not have an impact on the future, unless they are definitive. Performance considering these noisy events for each forgetting method is evaluated in the following simulation:

This third simulation (Figure 4) has the same inputs as the first but adds noise on some days. The simulated user also has the same behavior but it is possible that for one or two days the user will completely change his/her behavior.

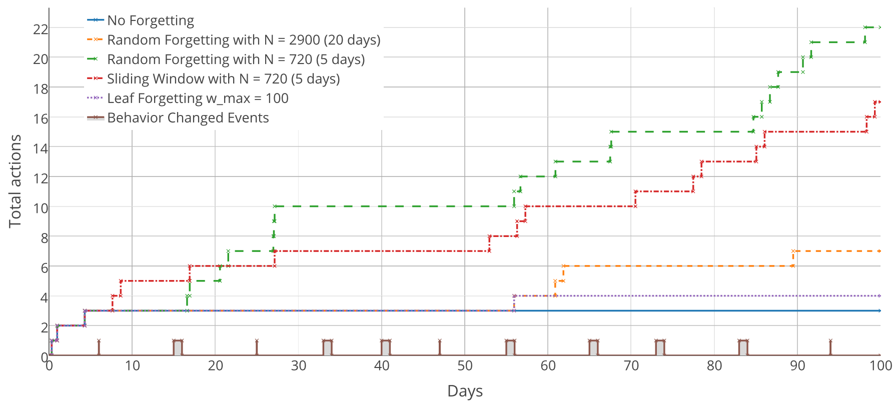

Fig. 4. Number of total user actions over 100 days with random behavior changes (the lower the better).

Results Without any forgetting method, the learning is strongly resistant to noise because it accumulated enough samples to reinforce the learned behavior.

Concerning the forgetting methods, the Random Forgetting and the Sliding Window with short lengths are not resistant to noise because they do not have enough samples in their training set. Indeed, we can see that with a higher N Random Forgetting has a resistance to noise that is far better.

Of all methods tested, Leaf Forgetting is the method the most resistant to noise. The user interacted on a single occasion during the 100 day period, once the behavior had been learned. This is explained by the constant reinforcement of behaviors previously learned and the balance created between each leaf.

5 Conclusion and Future Works

In this paper we have presented several forgetting methods for the C4.5 algorithm and analyzed their performance. In the Internet of Things, proposing personalized and contextualized applications leads to the use of algorithms capable of adapting with forgetting methods. Forgetting is necessary in the IoT domain because of the user behavior inconsistency and the necessity to offer the user an impression of uniqueness. The methods implemented to propose such a learning must have a *White Box* namely a controllable and understandable learning. In this paper, three new methods are proposed, Sliding Window, Random Forgetting and Leaf Forgetting. As we can see from the results of a series of experiments, the Leaf Forgetting method has the best performance and meets the expectations of a learning that can be quickly adapted to the user.

In the future, an improvement for Leaf Forgetting would be to integrate the weight of every leaf directly into the computation of the C4.5 algorithm, like Data Stream Mining algorithms do with incremental classifiers [5]. This approach should improve the performance in terms of memorization and the balance between each leaf by normalizing the information of the leaves.

Another perspective to this work is to apply the *White Box* learning approach to new domains. We would now like to apply it to Data Streams. This will bring two new challenges: the larger number and size of samples and their incoming rate. The application of Leaf Forgetting to Data Streams will highlight its strengths and weaknesses with this type of data and will lead to improvement perspective.

References

[1] Raw air temperature at Murdoch, Australia, 11/01/13 to 11/01/15. http://wwwmet.murdoch.edu.au. Accessed: 2015-11-01.

[2] A. Bifet. *Adaptive Learning and Mining for Data Streams and Frequent Patterns.* PhD thesis, Universitat Politecnica de Catalunya, 2009.

[3] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees.* Wadsworth and Brooks, Monterey, CA, 1984.

[4] A. Champandard. Understanding behavior tree. http://aigamedev.com/open/article/bt-overview/. Accessed: 2015-09.

[5] X. H. Dang, V. C. Lee, W. K. Ng, and K. L. Ong. Incremental and adaptive clustering stream data over sliding window. In *Proceedings of the 20th International Conference on Database and Expert Systems Applications*, DEXA '09, pages 660–674, Berlin, Heidelberg, 2009. Springer-Verlag.

[6] R. de Pontes Pereira and P. M. Engel. A framework for constrained and adaptive behavior-based agents. *CoRR*, abs/1506.02312, 2015.

[7] R. Dey and C. Child. Ql-bt: Enhancing behaviour tree design and implementation with q-learning. In *CIG*, pages 1–8. IEEE, 2013.

[8] P. Erler. Decision-making in cosmonautica. http://www.makinggames.biz/news/decision-making-in-cosmonautica,6020.html. Accessed: 2016-01.

[9] S. Garcia and F. Herrera. An extension on "statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons. *Journal of Machine Learning Research*, 9:2677–2694, Dec. 2008.

[10] D. Isla. Handling complexity in the Halo 2 AI. GDC, 2005.

[11] J. L. Kolodner. An introduction to Case-Based reasoning. *Artif. Intell. Rev.*, 6(1):3–34, 1992.

[12] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, Mar. 1986.

[13] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[14] R. E. Schapire. A brief introduction to boosting. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, IJCAI '99, pages 1401–1406, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[15] X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, Dec. 2007.