Learning User Preferences for Adaptive Pervasive Environments: An Incremental and Temporal Approach

SARAH GALLACHER, ELIZA PAPADOPOULOU, NICK K. TAYLOR, and M. HOWARD WILLIAMS, Heriot-Watt University

Personalization mechanisms often employ behavior monitoring and machine learning techniques to aid the user in the creation and management of a preference set that is used to drive the adaptation of environments and resources in line with individual user needs. This article reviews several of the personalization solutions provided to date and proposes two hypotheses: (A) an incremental machine learning approach is better suited to the preference learning problem as opposed to the commonly employed batch learning techniques, (B) temporal data related to the duration that user context states and preference settings endure is a beneficial input to a preference learning solution. These two hypotheses are the cornerstones of the Dynamic Incremental Associative Neural NEtwork (DIANNE) developed as a tailored solution to preference learning in a pervasive environment. DIANNE has been evaluated in two ways: first, by applying it to benchmark datasets to test DIANNE's performance and scalability as a machine learning solution; second, by end-users in live trials to determine the validity of the proposed hypotheses and to evaluate DIANNE's utility as a preference learning solution.

Categories and Subject Descriptors: I.2.6 [**Artificial Intelligence**]: Learning—Connectionism and neural nets

General Terms: Algorithms, Design, Human Factors

Additional Key Words and Phrases: Context aware, incremental learning, personalization, pervasive environment, user preferences

ACM Reference Format:
Gallacher, S., Papadopoulou, E., Taylor, N. K., and Williams, M. H. 2013. Learning user preferences for adaptive pervasive environments: An incremental and temporal approach. ACM Trans. Autonom. Adapt. Syst. 8, 1, Article 5 (April 2013), 26 pages.
DOI: http://dx.doi.org/10.1145/2451248.2451253

5

1. INTRODUCTION

Computational technology is filtering into our everyday lives to a greater and greater degree. Nowadays it is common for a single person to own multiple computational devices such as laptops, desktop PCs, smartphones, and tablet devices. Additionally, smaller and more powerful mobile devices [Iphone 2012; Ipad 2012] are enabling users

This work was spported by the European Commission (DAIDALOS, PERSIST, and SOCIETIES projects). Apart from funding these projects, the European Commission has no responsibility for the content of this article.
S. Gallacher is currently affiliated with the Intel Collaborative Institute on Sustainable and Connected Cities (ICRI-Cities), University College London.
Authors' addresses: S. Gallacher (corresponding author) Intel Collaborative Institute on Sustainable and Connected Cities (ICRI-Cities), University College London; email: s.gallacher@ucl.ac.uk; E. Papadopoulou, N. K. Taylor, M. H. Williams, School of Mathematical and Computer Sciences (MACS), Heriot-Watt University, Scotland.

DOI: http://dx.doi.org/10.1145/2451248.2451253

to take computational technology with them wherever they go. Advances in networking such as wireless technologies, IPv6 [Deering and Hinden 1995], and the Internet of Things [Gershenfeld et al. 2004] are allowing our environments to be better connected while advances in embedded technologies are empowering everyday objects [Mistry and Maes 2009; ADIDAS_1 2005] with processing and communication power.

We are moving towards the view of the world that Weiser [1991] envisaged over 20 years ago and what he termed as *ubiquitous computing* (also termed pervasive computing or ambient intelligence). This paradigm painted a picture of environments filled with computational technology that could beneficially adapt to meet user needs. *Context awareness* plays a key role as it supports environment adaptation based on current environment and user state information. Another key concept is *personalization*. Personalization utilizes additional personal information about individual users (in addition to user context information) to enable environments to adapt differently for different users, or for the same user in different situations. This supports the argument of Satyanarayanan [2001] that personal user information is crucial for system decision taking, adding that "otherwise it would be impossible to determine which system actions will help rather than hinder the user." For example, one individual may prefer his smartphone to use the least expensive network when he his at home whereas another user may prefer her smartphone to use the network with best Quality-of-Service (QoS).

In pervasive computing, personalization mechanisms often use rules called *user preferences* to drive personal adaptations. A user preference outlines what an individual user prefers in some contextual situation. For example, the user may have a volume preference for some multimedia device that states "if I am at home and I am on my own, turn the volume up high, but if I am at home and I am not on my own, turn the volume down low." Personalization mechanisms will monitor the user's context and alter the multimedia volume accordingly on behalf of the user.

For each individual user, the quality of the personalization experience depends on the coverage and accuracy of the set of preferences that the user owns. Ideally a preference set should contain sufficient preferences to cover all possible environment adaptations. The user may wish to create and manage all preferences manually but it is usual to provide support for such tasks. This is because manually creating and managing the preference set can become a heavy burden depending on its size and complexity (both of which are likely to increase through time). Therefore, personalization systems typically include behavior monitoring and machine learning mechanisms to monitor context-dependent user behaviors, learn preferences, and update the preference set on behalf of the user.

In a personalization system that supports preference learning the selection of an appropriate machine learning technique is critical. It must be able to extract accurate preferences from monitored behaviors in a challenging, ever-changing environment where input sources come and go. It must be able to keep the preference set up to date by responding rapidly to changes in user behavior while at the same time remaining less sensitive to noise from small deviations in user behavior. Additionally, it must be possible to present the entire preference set to the-user. Therefore, it should be possible to interpret the internal knowledge of the chosen machine learning technique into a human-understandable form. This is most important to enable the end-user to have final control over personalized adaptations and to gain a mental picture of system intelligence [Dey 2009].

Personalization systems with preference learning functionality have been implemented in various pervasive projects, each adopting different machine learning techniques. Section 2 of this article outlines several of these projects and discusses their proposed preference learning solutions. Two trends have been observed: the majority of preference learning solutions employ batch machine learning techniques and do not

consider the temporal aspects of environmental states and preference settings. These two trends are questioned in this article with a view to whether they are best practice for preference learning.

Regarding the usage of batch learning algorithms, it is noted that the preference learning problem domain is essentially incremental in nature with inputs occurring one at a time through time. Research suggests that such problems are more naturally and flexibly handled by incremental algorithms. Regarding the exploitation of temporal data, it is noted that the sometimes preparational nature of human behaviors can introduce noise into the behavior history set which is often difficult for conventional preference learning systems to overcome. It is posed that the exploitation of temporal data relating to environmental states and preference settings can help overcome such issues. Based on these observations, two hypotheses are proposed in Section 3: (A) an incremental machine learning approach is better suited to the preference learning problem as opposed to the commonly employed batch learning techniques, and (B) temporal data related to the duration that user context states and preference settings endure is a beneficial input to a preference learning solution.

These two hypotheses are the cornerstones of the Dynamic Incremental Associative Neural NEtwork (DIANNE) which is presented in Section 4 in terms of its topology and capabilities. It is a single-layer neural network designed specifically for the task of preference learning in pervasive environments. Section 5 introduces DIANNE's temporal learning algorithm which is incremental and also exploits temporal information corresponding to environmental states and preference settings. Section 6 details how DIANNE has been evaluated, first for performance and scalability using benchmark datasets and second as a real-time preference learner in live user trials. The results of the both evaluations are presented and discussed. Finally Section 7 provides a conclusion to the work and reflects on the validity of the hypotheses.

2. OTHER PREFERENCE LEARNING SOLUTIONS

Since the early 90's, pervasive projects have utilized machine learning techniques for preference learning. However, many different machine learning techniques are available with each being suited to some problem domain. Therefore, personalization systems employ different machine learning techniques depending on personalization goals and required preference formats. Some systems focus on task-driven personalization where preferences are based on past sequences of behaviors. Other systems focus on context-driven personalization where preferences are based on past cooccurrences of context states and behaviors. Since the goals are different it is not surprising that a variety of machine learning techniques are employed within implicit personalization systems; however, there are several approaches that are more commonly used. Table I maps the various learning techniques and algorithms to the projects that employ them.

The authors' gained firsthand experience of designing and implementing a personalization system in the DAIDALOS project [Taylor et al. 2011]. In a process typically employed in other preference learning systems, the DAIDALOS system monitored user behaviors where the behavior altered the personalized state of some service/resource. These behaviors were then stored with a snapshot of the context state in which the behavior was performed. Once sufficient monitored data was collected, the C45 decision tree learning algorithm [Quinlan 1986] was then applied to extract context-dependent preferences indicating what behaviors the user performed in a given context. The decision tree output could be easily translated into human-understandable form enabling users to understand what had been learned about them. However, other projects have placed less focus on user understanding and hence adopted learning algorithms with more complex internal structures.

Table I. Machine Learning Techniques and Algorithms Employed for Preference Learning by Pervasive Projects

ML Technique	Algorithm	Project
Decision Tree Learning	C45	DAIDALOS
Artificial Neural Networks (ANNs)	Sparse Network of Winnows (SNoW)	GAIA
	Multi-layer perceptron	Adaptive Home
Stochastic Models	Hierarchical Hidden Markov Models (HHMM)	MavHome
	Hidden Markov Models (HMM)	Synapse
	Naïve Bayesian Classifier	Mobilife
Fuzzy Logic	Fuzzy rules	iDorm
		Ubisec
Rule Learning	Ripper algorithm	Mobilife
	Apriori algorithm	SPICE
Reinforcement Learning	Q Learning	Adaptive Home

The GAIA project [Ziebart et al. 2005] is developing a middleware infrastructure for smart homes and offices which it terms *active spaces*. It utilizes the Sparse Network of Winnows (SNoW) algorithm to extract context-dependent preferences from monitored behavior. Agents can then use these preferences to automatically adapt the environment appropriately as the user's context changes in the future. SNoW is essentially a neural network of perceptrons utilizing the WINNOW learning rule. It is specifically tailored for domains with large numbers of features that may not be known a priori and hence is well-suited to pervasive environments where the set of context features is potentially large and changes with time.

The adaptive home (or neural network home) project [Mozer 1998] utilizes reinforcement learning (Q-learning algorithm) and neural network (multilayer perceptron with backpropagation) techniques to learn the intentions of inhabitants within the smart home environment. The aim is to balance user requirements and energy conservation. To achieve this the adaptive home employs learning techniques to build models of future context states for future context prediction (e.g., future occupancy of an area or future hot water usage). User tasks are then analyzed against predicted future context states to proactively adapt the home appropriately in terms of future user and energy requirements.

The MavHome project [Youngblood et al. 2005] also focuses on the prediction of future user tasks to drive adaptations within a home environment. Several machine learning techniques such as sequential pattern discovery and Markov chains are used to identify commonly occurring patterns of behavior from stores of monitored historic behavior data. Once a model of behavior has been learned, an incremental prediction algorithm (Active-LeZi) is used to predict future behavior in real time (e.g., in a given context, when the user switches on the living room lights, she then switches on the TV).

The GAIA, adaptive home and, MavHome projects all have autonomy as their key goal with the intention of mitigating user interaction. However, the Synapse project [Si et al. 2005] heeds the advice of Barkhuus [2003] who argues that users want to enjoy autonomous behavior to a moderate degree without losing control. As a result, the Synapse personalization system performs environment adaptations under two modes: *active* and *passive*. Bayesian networks are employed to learn preferences that capture the relationships between context states and service usage behavior. This learned knowledge is then applied to personalize the user's environment through service provision. If a preference has a probability above some threshold, personalization operates in active mode and the service is started automatically. If the preference has a probability below some threshold, personalization operates in passive mode and the top five

potential services are presented to the user for manual selection. This approach aims to minimize incorrect personalization in uncertain situations while at the same time providing automation when appropriate.

In fact, finding the correct balance between automation and user control is a challenging personalization issue. If personalization is incorrect due to unsatisfactory learning techniques or a change in user behavior then it is desirable that the system provides mechanisms to identify and rectify the problem rapidly, mitigating user involvement. This places a requirement on preference learning processes to be able to rapidly accommodate new patterns of behavior into the preference set without requiring long periods of time to unlearn old preferences and learn new ones.

This can be a significant challenge when employing learning techniques that typically execute in batch mode. The nature of such techniques prevents any natural quick response to changes in user behavior since learning occurs during scheduled executions, between which no additional updates to internal knowledge are made. If user behavior were to change immediately after a scheduled learning execution, the preference set would remain erroneous until the next scheduled execution which may be hours or even days later. A rapid response solution is required to complement batch learning techniques. Many projects have identified this requirement and have implemented mechanisms that rapidly update the preference set in real time based on user feedback.

The Ubisec project [Groppe and Mueller 2005] employs fuzzy logic learning but also implements mechanisms to enable the quick accommodation of new information into its customization (or user) profile. If a conflicting behavior occurs (e.g., the device volume is set to mute by the system but the user unmutes the volume), the real-time profile evolution process analyzes the differences between the customization profile and the device status profile. A recommendation profile is generated from the differences and used to update the customization profile, subject to manual user approval. In this way the required updates are implicitly gathered from monitored manual device reconfiguration and do not require the user to understand complex profile GUIs.

Other projects such as SPICE [Cordier et al. 2006] and its predecessor Mobilife [Sutterer et al. 2007] also implement real-time response techniques. In both projects, rule learning techniques are used in conjunction with real-time updates based on explicit negative user feedback received as a consequence of undesired personalization.

The system implemented within the iSpace at Essex University [Hagras 2007] provides a rapid response mechanism that is not dependent on explicit user feedback. Once a preference set has been established during the initialization period, preferences can be modified, added, or deleted when user behavior changes. At such times, a nonintrusive cycle is entered where new or changing user behavior is specifically monitored and new preferences learned. Additionally, in a life-long learning phase the worst-performing preferences are periodically replaced by new ones to preserve system performance.

3. OBSERVATIONS AND HYPOTHESES

3.1. Incremental over Batch

The majority of the projects mentioned earlier utilize batch machine learning algorithms where preference learning is performed during scheduled executions, between which the batch algorithm is idle. This decision comes with several consequences due to the nature and limitations of batch learning algorithms. First, a batch algorithm requires a behavior history store to be maintained. This store could potentially become very large since all past instances must be retained for reprocessing on consecutive learning executions if catastrophic forgetting is to be avoided. Second, batch algorithms cannot naturally respond rapidly to changes in user behavior that occur

between learning executions. For this reason, several of the projects mentioned before have implemented rapid response mechanisms to keep the preference set up to date between learning executions, should the user behavior change. These rapid response mechanisms are often termed "incremental" by the projects that employ them due to the way in which they update the user profile in real time. However, it should be noted that incremental machine learning algorithms are rarely employed for preference learning.

This is surprising, considering the nature of the preference learning problem domain. In a pervasive system, examples of user behavior will typically become available and hence be monitored by the system one at a time, over time as the user interacts with his environment. The entire dataset will not be known a priori and learning should ideally continue (almost) indefinitely. One could question why incremental learning algorithms are rarely applied to a problem domain with an incremental nature. It is due to these characteristics that Giraud-Carrier [2000] proposes that user modeling tasks (such as preference learning) are essentially incremental tasks. He goes on to argue that although incremental tasks can be handled by nonincremental algorithms, the most natural and flexible way to handle an incremental task is with an incremental algorithm.

By utilizing an incremental algorithm, all new inputs would be processed immediately as they are received. The behavior history store would become redundant and updating of the user's preference set would not be restricted to cyclic executions. This would also remove the need for explicit rapid response mechanisms as the incremental algorithm itself would naturally be able to respond rapidly to changes in behavior. However, several key issues must be considered. How should new inputs be accommodated into internal knowledge and what if new inputs conflict with existing internal knowledge? Incremental algorithms also exhibit a learning curve since they learn from scratch with initially very little information. Initial outputs may be inaccurate and it may be difficult to determine when the algorithm has received sufficient input to be trusted.

Indeed there may be some trade-off between accuracy and responsiveness. However, as Webb et al. [2001] outline, often in an adaptive real-world domain, predictive accuracy comes second to various other factors such as CPU time. They also highlight that appropriate prompting techniques can greatly improve the user experience when predictive accuracy is lower. Essentially, the nature of incremental algorithms appears to be more in line with the nature of the preference learning problem domain. This has led to hypothesis (A): an incremental machine learning approach is better suited to the preference learning problem as opposed to the commonly employed batch learning techniques.

3.2. Temporal Information is Important

It is noted that none of the preference learning solutions reviewed previously makes use of temporal information related to the duration of user context states or preference settings. This is mainly due to their policies for context and behavior monitoring where an individual monitoring event occurs only when the user performs a behavior that alters the personalized state of some service/resource. Therefore we know that the user has performed behavior b in context c, but it does not tell us how long b and c endured. This disregard for temporal information means that such systems will have difficulty dealing with specific issues that can manifest due to the sometimes varied nature of human behavior. An example is the issue of *preparational actions (preactions)*.

A preaction is an interaction performed by the user in a previous context to prepare for entrance into a new context. For example, if the user is entering a lecture theater, he may mute his mobile phone in the corridor outside before he enters the lecture theater. Equally, when leaving the lecture theater, the user may unmute his mobile phone just before he leaves the lecture theater. Most preference learning solutions assume that

the current user context should be associated with the behaviors performed within and therefore they monitor behaviors and store them with a snapshot of the current user context. Hence they would learn a preference stating that the user prefers to mute his phone in the corridor and prefers to unmute his phone in the lecture theater, even though the user actually prefers the reverse.

If we reconsider the lecture theater scenario again from a temporal perspective, the user mutes his mobile phone outside the lecture theater and then enters the lecture theater. Therefore, the "mute" state prevails for only a short time period in the context outside the lecture theater, but prevails for a much longer time period in the context inside the lecture theater. The temporal duration of the cooccurring context and preference states provides important information that naturally leads one to conclude that the mute state is more strongly associated to the context inside the lecture theater where it prevailed for a greater temporal duration.

Taking another example; in some context the user sets his device to always use the least expensive network. This state prevails for several minutes before the user alters his device to always select the network with the best QoS. This state prevails for a number of weeks. Note that the behaviors only occur once in some context. With no additional temporal information, both actions could be equally associated to the context state. This contrasts with the natural assumption that the second setting is more strongly associated to the context due to its longer duration. Again, the temporal duration of the cooccurring context and preference states provides valuable information. This has led to hypothesis (B): temporal data related to the duration that user context states and preference settings endure is a beneficial input to a preference learning solution.

Of course, one could always pose scenarios where the temporal information also introduces noise, for example, if the user performs a behavior and then gets distracted, allowing the behavior to endure in the current context for an abnormal length of time. Such noise is inevitable in a system that exploits temporal information. Identifying when the user is distracted would be extremely challenging; therefore, to gain the benefits that temporal information can provide (such as overcoming preactions) the system should provide mechanisms that can undo distraction noise in a rapid fashion.

4. DYNAMIC INCREMENTAL ASSOCIATIVE NEURAL NETWORK (DIANNE)

DIANNE [Gallacher 2011] is a Dynamic Incremental Associative Neural NEtwork specifically designed for the problem domain of preference learning in pervasive environments. Its primary goal is to learn user preferences by associating the user's context and the user's preference settings. It implements two key concepts in line with the hypotheses highlighted before.

First, it implements an incremental approach to learning. It processes inputs as they occur in real time; it does not need to reprocess past inputs and hence it is not reliant on any behavior history store. By processing inputs in real time DIANNE can rapidly respond to new inputs including changes in user behavior and hence there is no need for any explicit rapid response mechanisms. Of course, many alternative incremental learning algorithms already exist [Hertz et al. 1991; Fisher 1987; Schlimmer and Granger 1986; Michalski et al. 1986], including incremental versions of decision tree building algorithms [Schlimmer and Fisher 1986] like the one implemented by the authors in the DAIDALOS project. However, to our knowledge, none of the incremental algorithms that already exist utilizes temporal data and hence does not adhere to the second hypothesis.

Second, DIANNE takes advantage of temporal data relating to the duration that user context states and preference settings endure. This is implemented through a temporal reinforcement learning policy that continuously alters the strength of the associations between context states and preference settings over time. As highlighted

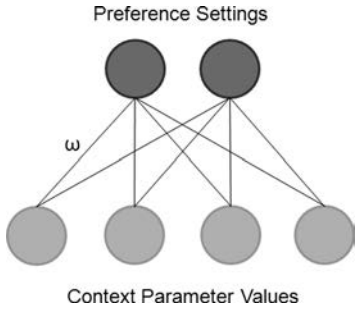

Fig. 1. Linear connections between context parameter values and preference settings.

in Section 3.2, it is proposed that the time a preference setting endures in some context is just as important as the fact that the preference setting was observable in the context. Therefore the strength of associations learned by DIANNE are not only based on the simultaneous occurrence of preference settings and context states but also the period of time that the simultaneous occurrence of preference settings and context states endured.

4.1. DIANNE Topology

DIANNE topology is a single-layer network although for ease it is described in terms of two layers: the *context layer* and the *preference layer*. The context layer contains *context nodes*, each representing some context parameter value (e.g. "home" is a value of the context parameter "location"). The preference layer contains *preference nodes*, each representing some setting of a preference (e.g. "high" is a setting of the preference "volume"). In its simplest form, DIANNE is a linearly connected network of context and preference nodes as shown in Figure 1.

The linear connections represent the direct influences that individual context parameter values have on preference settings. Consider that it is changes in the context parameter values that affect what preference settings are implemented. For example, if we have the following preference (written in IF-THEN-ELSE format for clarity):

IF <location = home>
THEN [volume = high]
ELSE IF <location = work>
THEN [volume = low]

we can see that the context parameter values "home" and "work" determine whether the implemented volume preference setting is "high" or "low".

A weight exists on each linear connection that represents the strength of the association between a specific context parameter value (context node) and a specific preference setting (preference node). A greater weight value means a stronger association. The DIANNE algorithm updates the weights on each connection over time based on the activity of the context node and the preference node. This manipulation of weights enables DIANNE to learn.

4.2. Node Activations

All nodes in DIANNE have a binary activation depending on the truth of the related context parameter value or preference setting in the user's real-world environment. Context updates (from the user's context provider) and preference updates (captured from the user's personalizable services) ensure that the network nodes always reflect

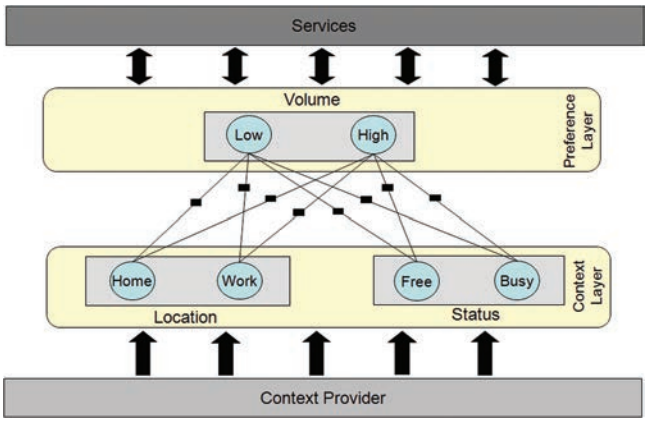

Fig. 2. DIANNE internal structure and interaction with external sources.

the current real-world state. Therefore, the context layer provides a representation of the user's context state and the preference layer provides a representation of the user's preference settings. Figure 2 illustrates how DIANNE interacts with external input sources.

Since each context node only represents one context parameter value it may be the case that several context nodes exist in the network, all relating to the same overall context parameter (e.g., there may be multiple context nodes each representing a different value of the context parameter "location"). It is assumed that a user's context model can only have one true value for each context parameter at a time. To implement such a constraint, the context nodes relating to the same context parameter are grouped together into *context node groups* (as shown in Figure 2) where the activation of the member context nodes is mutually exclusive. In each context group, the active node is always the one that represents the context parameter value that is currently true in the real-world environment.

Each context node has an associated *input potential*. The input potential is the value that the context node pushes into the network. As the activity of context nodes change so do their input potentials and hence the overall input to the network. The input potential of context node β_i is defined as

$$ip(\beta_i) = \begin{cases} 1 & \text{if } \beta_i \text{ is active} \\ 0 & \text{if } \beta_i \text{ is inactive.} \end{cases} \quad (1)$$

The same grouping policy is also applied to preference nodes since it is also assumed that only one preference setting of a particular preference can be true at a time. Hence preference nodes relating to the same preference are grouped together into *preference node groups* (also shown in Figure 2) where the activation of the member preference nodes is also mutually exclusive.

However, the activation of preference nodes is driven by two factors: (a) real-world input from services and (b) the sum of weighted connections, reflecting DIANNE output. Each preference node has an *output potential* that indicates how strongly DIANNE believes that this preference node should be active given the current context. The higher the potential, the stronger the belief that the node should be active. Each preference node has some number n of inputs where n is the total number of context nodes. The output potential of a preference node is the sum of its inputs; therefore the

output potential of α_j at time t is defined as

$$op\left(\alpha_j^t\right) = \sigma\left(\sum_{i=0}^{n} w_{ji}^t \beta^t i\right) = \sigma\left(w_{j1}^t \beta_1^t + w_{j2}^t \beta_2^t + \cdots + w_{jn}^t \beta_n^t\right), \tag{2}$$

where σ is the dynamic ramped squashing function with a variable gradient that maps the output potential from the possibly very large range of values to a finite range of values between -1 and $+1$. It is dynamic because of the continuous, life-long incremental nature of DIANNE learning. If a static gradient were used, saturation points could eventually be reached over time (e.g., if a preference setting always endures in some context) meaning that further, equivalent updates to outcome potentials would serve no purpose. Therefore σ is implemented as a dynamic squashing function with a variable gradient so saturation does not occur over time.

For each preference group, the gradient of σ is defined based on the output potentials of the group's winner node α_{win} (i.e., the node with the highest output potential in the group) and the group's loser node α_{lose} (i.e., the node with the lowest output potential in the group). The ramped function σ is defined as:

$$\sigma = \left\{ \begin{array}{ll} 1 & : \text{if op}(\alpha_{win}) \geq \text{highlimit} \\ grad \bullet op\left(\alpha_j^t\right) & \\ -1 & : \text{if op}(\alpha_{lose}) \leq \text{lowlimit} \end{array} \right\},$$

where *highlimit* and *lowlimit* dictate the dynamically adjustable positive and negative saturation points of σ. The *grad* variable is the gradient of the slope of σ defined as

$$grad = \frac{y_2 - y_1}{x_2 - x_1} = \frac{1 - (-1)}{\text{highlimit} - \text{lowlimit}} = \frac{2}{\text{highlimit} - \text{lowlimit}}.$$

The highlimit and lowlimit variables dictate the saturation points of σ. The variable highlimit dictates the positive saturation point while lowlimit dictates the negative saturation point. At time t_0 highlimit is initialized to $+\phi$ and lowlimit is initialized to $-\phi$ where ϕ is some constant greater than zero. In current DIANNE implementations ϕ is initialized to 10. The value of ϕ only affects the frequency with which the gradient of the dynamic squashing function updates. During DIANNE operation if $op(\alpha_{win}) \geq$ highlimit or $op(\alpha_{lose}) \leq$ lowlimit the highlimit variable is increased by ϕ while lowlimit is decreased by ϕ. The new value of *grad* is then calculated.

Each preference group implements mutually exclusive activation in a winner-takes-all fashion where the *winner node* (the node with the greatest output potential in the preference group) is activated. The winner node is the node that DIANNE believes should be active and implemented in the real world. In the majority of cases the winner node will also be the active node; however, sometimes this may not be the case. This situation indicates a change in user behavior from what has been observed and learned in the past. For example, DIANNE may identify the "low" setting of some volume preference as the winner node but the user may have manually changed the setting to "high". DIANNE handles such conflicts as part of its algorithm, described in Section 5.

It is noted that DIANNE could lend itself to alternative topologies. For example, each outcome node group (preference grouping) could be represented as an independent network. In doing so the context layer of each independent network could be tailored to contain nodes that are most influential to each outcome node group, enabling DIANNE to take advantage of a priori information of the problem domain. The challenges of

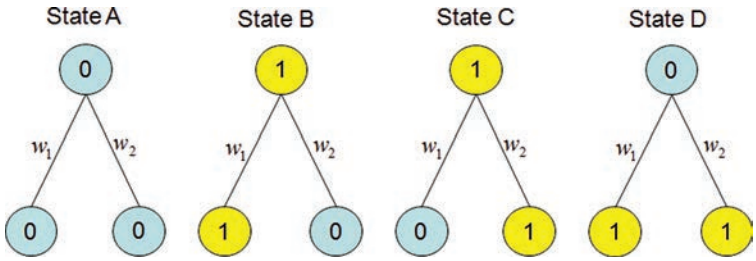

Fig. 3. XOR network states.

this topology would be the synchronization of mutual exclusivity in the context layers of multiple independent networks as well as the overhead in managing multiple independent networks.

4.3. Weight Manipulations

Each connection between a context node β (prenode) and a preference node α (postnode) in DIANNE has an associated weight value. The weight value determines the strength of the connection between β and α (and hence the strength of the association between the real-world values they represent). It is the manipulation of these weights that allows DIANNE to learn. The plasticity of weights is dependent on the activity of pre-and postnodes and follows Hebbian [Hebb 1949] learning policies. A weight will increase if the positive activity of β leads to the positive activity of α, decrease if the positive activity of β leads to the negative activity of α, and stay the same if β is not active.

The weight of synapse w_{mn} at time T is defined as

$$\begin{aligned} w_{mn}^{T} &= \left(w_{mn}^{T-1} + activity^{T}(\alpha_m, \beta_n)\right), \\ &= \left(\sum_{t=0}^{T} activity^{t}(\alpha_m, \beta_n)\right) \end{aligned} \tag{3}$$

where:

$$activity^{t}(x, y) = \begin{cases} +1 & : \text{if } \beta_n \text{ is active at time } t \text{ and} \\ & \quad \alpha_m \text{ is active at time } t; \\ -1 & : \text{if } \beta_n \text{ is active at time } t \text{ and.} \\ & \quad \alpha_m \text{ is not active at time } t; \\ 0 & : \text{if } \beta_n \text{ is not active at time } t \end{cases}$$

In many artificial neural networks weights are initialized randomly and modified to converge on some target function during the course of the training process. However, due to the "learning from scratch", incremental nature of DIANNE, it is natural to initialize the weights to zero giving them an initial state that is neither excitory nor inhibitory. DIANNE learning can continue in a life-long fashion, therefore the weights will not converge on some definitive target value. Some may converge towards upper or lower bounds ($\pm\infty$) for periods of time but equally others may not.

4.4. DIANNE Generality

(1) Nonlinear Problems. A well-documented constraint of single-layer neural networks is their inability to handle nonlinear problems such as XOR. As a single-layer network, DIANNE will not be able to represent XOR; however, in this problem domain

(context-dependent preference learning) DIANNE will never need to solve the XOR problem. Consider the four XOR states shown in Figure 3.

It is possible to represent state A in DIANNE. When all context nodes are inactive DIANNE can make no prediction so it will not return an outcome node in these circumstances. States B and C can also be represented by DIANNE. When input is available (i.e., some context nodes are active) DIANNE will always return the most likely outcome node for each node group. If there is only one outcome node in a group, that outcome node will always be returned in response to input. With that in mind, state D will never need to be represented by DIANNE as when input is available output will always be provided.

(2) Continuous Inputs. DIANNE has been designed to handle discrete inputs. Due to the fact that each network node is binary and represents only one value, continuous inputs would result in the creation of numerous network nodes and would decrease performance significantly in a manifestation of overfitting. Since many context attributes can relate to continuous sensor inputs we must consider how this data is handled in DIANNE. As is the case with many other learning algorithms it is necessary to discretize continuous inputs in a preprocessing step before they are presented to DIANNE. In the network example shown in Figure 2 it is assumed that discretized symbolic locations such as "home" and "work" have already been inferred from continuous raw sensor inputs such as GPS coordinate, signal strengths, etc. as is typical in many pervasive systems.

4.5. DIANNE Capacity

The capacity of a network is often expressed as a function of N, the number of nodes the network contains. However, DIANNE groups nodes into mutually exclusive sets. Therefore we must define the capacity of DIANNE in terms of both network nodes and groups.

First we will consider the number of possible input patterns P_{in} that the network can handle. A context group c contains some n number of context nodes. Since the activation of each node in c is binary the total number of possible node activation patterns in c is 2^n. However, we must discount any activation patterns that violate the mutually exclusive constraint on the activation of nodes within the same group. Therefore, if c contains n nodes the number of possible node activation patterns for c is

$$(n + 1).$$

If we have k context node groups then

$$\begin{aligned} P_{in} &= \big((n_1 + 1) \times (n_2 + 1) \times \cdots \times (n_k + 1)\big) \\ &= \prod_{i=1}^{k} (n_i + 1). \end{aligned}$$

Linear connections, binary node activations, and the single-layer architecture of DIANNE mitigate against internal hidden complexity. Therefore each input pattern P_{in}^{i} has one associated output pattern P_{out}^{i}. In other words,

$$P_{out} = P_{in}.$$

Therefore DIANNE storage capacity is equal to the number of possible input patterns (i.e., the number of possible context situations) P_{in}.

5. DIANNE TEMPORAL LEARNING ALGORITHM

DIANNE temporal learning algorithm is based on the two hypotheses outlined in Section 3. It handles inputs incrementally and implements a temporal reinforcement

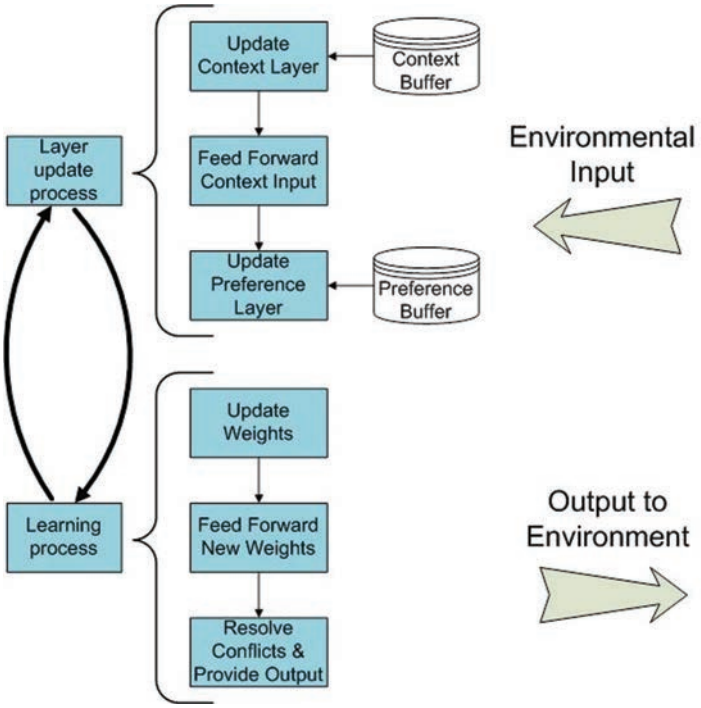

Fig. 4. Illustration of the main processes involved in the DIANNE temporal learning algorithm.

policy which dictates that network weights are updated (i.e., learning occurs) on a temporal basis in a real-time, repeated cycle. Unlike most conventional neural systems, DIANNE learns associations between vectors based on the duration of vector state cooccurrences rather than the fact that they cooccurred at one instance in time.

The DIANNE learning algorithm iterates in a continuous cycle with a frequency of one second, thereby learning in real time in a life-long manner. Figure 4 illustrates the temporal DIANNE learning algorithm. Initial algorithm designs tended towards an asynchronous approach with each network node operating in its own thread, updating itself depending on asynchronous inputs from the environment and other nodes. However, synchronization issues quickly appeared and network accuracy suffered as a result. The synchronized approach proposed may be slightly less reactive (up to one second slower) than an asynchronous approach; however, the required reaction time of DIANNE in a pervasive system is often application dependent and will also be influenced by the performance capabilities of the pervasive space.

The DIANNE algorithm can be split into two major processes. The *layer update process* is concerned with processing input (if any) received from the environment since the previous iteration. The *learning process* is concerned with updating DIANNE weights and providing output (if any) to the environment.

5.1. The Layer Update Process

The main purpose of this process is to ensure that the network nodes correctly reflect the environmental state before any learning updates occur. The context nodes should reflect the user's current context and the preference nodes should reflect the user's current preference settings. Failure to perform this process correctly will result in the network associating incorrect context and preference nodes. Between iterations, all

context changes received from the context provider are stored in a *context buffer* and all preference changes received from the user's personalizable services are stored in a *preference buffer*. The buffers then provide input to the three subprocesses involved in the layer update process.

(1) Update Context Layer. The first subprocess involves updating the context layer based on the contents of the context buffer. This ensures that the context layer reflects the user's current contextual state (for example, since the last algorithm iteration, the user may have changed location). The pseudocode for this process is detailed next.

```
while context buffer contains more inputs
  if context node group c_m exists in DIANNE
    if context node β_i ∈ c_m
      set activity of β_i to active
      set activity of all other nodes in c_m to inactive
    else
      create new context node β_new in c_m
      connect β_new to all existing preference nodes and initialise new
      weights to zero
      set activity of β_new to active
      set activity of all other nodes in c_m to inactive
  else
    create new context node group c_new
    create new context node β_new in c_new
    connect β_new to all existing preference nodes and initialise new
    weights to zero
    set activity of β_new to active
  update input potentials of all nodes in c_m based on equation (1)
```

(2) Feed Forward Context Input. The second subprocess feeds the new context inputs forward through the network to the preference layer where the preference nodes are updated accordingly. The preference layer will then represent what the network believes to be the preferred preference settings in this context. The pseudocode for this process is detailed next.

```
for each preference node group o_n
  for each preference node α_j ∈ o_n
      calculate output potential based on equation (2)
  activate winner node in o_n
```

(3) Update Preference Layer. The final subprocess involves updating the preference layer based on the contents of the preference buffer. In the previous subprocess the outcome vector was updated based on new context input to reflect what the network believes to be true in this context. By contrast, this subprocess updates the preference nodes based on new user inputs to reflect any changes in preference settings made by the user. The pseudocode for this process is detailed new.

```
while preference buffer contains more inputs
  if preference node group o_n exists in DIANNE
    if preference node α_j ∈ o_n
      set activity of α_j to active
      set activity of all other nodes in o_n to inactive
    else
      create new preference node α_new in o_n
      connect α_new to all existing context nodes and initialise new
```

```
      weights to zero
      set activity of α_new to active
      set activity of all other nodes in o_n to inactive
  else
    create new preference node group o_new
    create new preference node α_new in o_new
    connect α_new to all existing context nodes and initialise new weights
    to zero
    set activity of α_new to active
```

5.2. The Learning Process

Once the DIANNE layers reflect the current real-world state of context and preference settings, the learning process can execute to strengthen and weaken associations between nodes, enabling DIANNE to learn. Within the learning process, two learning rules are utilized. First, the Hebbian learning rule is used for normal, temporally driven updating of the synapse weights. This learning always occurs in each iteration of the DIANNE algorithm, incrementally increasing and decreasing network weights based on pre-and postnode activations. Secondly, an error-driven policy is utilized to update synapse weights during conflict resolution. This learning only occurs when conflicts exist between network knowledge and real-world states and is based on the potentials of the conflicting network nodes. There are three subprocesses involved in the overall learning process. Each is described in detail next.

(1) Update Weights. First, the synapse on each network connection is updated based on the Hebbian learning rule. The pseudocode for the process of updating all synapses is detailed in the following.

```
for each connection (α_j β_i)
  for each weight w_ji
      update value of w_ji based on equation (3)
```

(2) Feed Forward New Weights. As in the layer update process, a feed forward subprocess is now required to feed the new synaptic weights forward to the preference layer, allowing the preference nodes to reflect network output, postlearning. The pseudocode for this process is detailed now.

```
for each preference node group o_n
  for each preference node α_j ∈ o_n
    calculate output potential based on equation (2)
  identify winner node in o_n
```

(3) Resolve Conflicts and Provide Output. At this stage DIANNE can now provide output to the environment as appropriate and resolve conflicts within the network. The pseudocode for the process of resolving conflicts and providing output to the environment is detailed next.

```
for each preference node group o_n
  if winner node is active
    if winner node is a new winner node
      send output to service
    else
      resolve conflict between network and real world
```

At this point in the algorithm cycle, each preference group has a winner node and an active node. The winner node indicates the preference setting that the network believes should be implemented while the active node indicates the preference setting

that has actually been implemented by the user. If the winner node and active node are the same node we can say that what DIANNE believes to be true is actually true in the real world, hence there is no conflict. In such a situation DIANNE can now provide output to the environment if appropriate.

The current winner node of this preference group is checked against the winner node of the previous algorithm iteration. If they are different this means that a new winner node has been identified for this context and must be communicated to the environment. In this case DIANNE broadcasts the new winner node as output. If the winner node is the same as the previous iteration, the environment already knows about and conforms to this winner node and hence DIANNE does not need to communicate the winner node again.

A conflict occurs within the network when the winner node is not the same node as the active node, that is, what the network believes should be implemented is not what is actually implemented. This usually occurs when the user changes her preferred preference setting in some context. To resolve this conflict the difference between the output potentials of the winner node and the active node must be decreased. The Hebbian learning rule will not decrease the difference in a sufficient time frame. For example, if Hebbian were applied, the active node would need to endure in the context state for a comparable length of time as the winner node had in the past before the active node could gain the higher output potential of the two and become the winner. Instead the *stochastic gradient descent* learning rule (also called the *incremental gradient descent* learning rule (also called the *incremental gradient descent* learning rule) is utilized to reduce the difference between the node potentials. This learning rule is the basis of other incremental algorithms such as WINNOW [Littlestone 1988] and the pocket algorithm [Gallant 1990]. When applied in DIANNE, the potential of the active node is *boosted* (by a factor of the difference between the potentials of the active and winner nodes) so that it can better compete with the output potential of the winner node. The larger the difference between the potentials of the active and winner nodes, the more significantly the active node potential will be boosted. Really entrenched behaviors will require several conflicts before new behaviors overcome old ones; however, one-instance learning is less desirable.

This change in output potential is reflected down through the weights on the connections between the active node and the currently activated context nodes. The weights on these connections are boosted by some equal value that is determined by dividing the active node boost value by the number of weights to be updated. This boosting process only occurs once when a conflict is first identified in some context. On subsequent algorithm iterations only Hebbian learning is applied as usual to reduce the possibility of one-instance learning.

It should be noted that DIANNE has no specific learning rate parameter. The rate at which the network learns is influenced by both the magnitude of the Hebbian weight updates (± 1) and the frequency with which the DIANNE algorithm repeats in the ongoing cycle (1 second). Changing the value of these constants will not have any great effect on the accuracy of DIANNE. They influence how frequently the gradient of the dynamic squashing function (of each preference group) will be recalculated.

The use of two learning rules (Hebbian for temporal reinforcements and stochastic gradient descent for conflict resolution) is key to dealing with noise introduced by the temporal data itself (i.e., if the user gets distracted after setting some preference, hence the preference endures in a context for a longer time than it should). For example, if the user sets the volume of some service to "low" and then becomes distracted, the "low" node will be regularly reinforced in this context in line with the Hebbian rule and will eventually become the group node with the highest output potential in this context. When the user eventually changes the service volume (e.g., to "high") this will create a

Table II. Summary of Benchmark Dataset Characteristics

Dataset Name	Completeness	# Instances	# Attributes	# Class Values	Entropy
CANCER	Incomplete	286	9	2	0.73
CANCERW	Complete	699	10	2	0.93
VOTE	Incomplete	435	16	2	0.96
LYMPH	Complete	148	18	4	1.28
HEART	Complete	267	22	2	0.73

conflict situation. In this case the "high" node's output potential will be more radically altered in line with the stochastic gradient descent rule, enabling the "high" node to more rapidly compete with the "low" node without having to endure for a comparable time in the same context.

6. DIANNE EVALUATION

DIANNE has been evaluated in two ways. To evaluate performance and scalability as a machine learning solution, DIANNE has been applied to benchmark datasets. DIANNE performance is presented and discussed and comparisons are drawn against the performance of other machine learning algorithms applied to the same datasets. To evaluate DIANNE's utility as a preference learner in a real-time pervasive environment DIANNE has also been deployed as a preference learner in live user trials. The user trials have provided real-time and temporal data to evaluate the incremental and temporal nature of DIANNE.

6.1. DIANNE Benchmark Evaluation

The goal of this process was to determine DIANNE performance and scalability over benchmark datasets. A total of five datasets were chosen from the UCI Machine Learning Repository [UCI 2007] based on their attribute and class values and their past usage in the evaluation of other machine learning algorithms. They are as follows.

—Breast Cancer (CANCER) This dataset was provided by the University Medical Centre, Ljubljana, Slovenia for the problem of predicting the reoccurrence of breast cancer five years after the removal of a tumor.
—Breast Cancer Wisconsin (CANCERW) This was provided by the University of Wisconsin Hospitals for the problem of predicting whether a lump is cancerous.
—Congressional Voting (VOTE) This was provided by the Congressional Quarterly Almanac for the problem of predicting whether a member of Congress will vote democrat or republican.
—Lymphography (LYMPH) This was provided by the University Medical Centre, Ljubljana, Slovenia for the problem of determining the type of cancer in lymphography.
—SPECT Heart (HEART) This was provided by the Medical College of Ohio for the problem of diagnosing cardiac SPECT images.

The selected datasets include complete and incomplete examples and have varying characteristics as summarised in Table II.

A test harness was developed to control the DIANNE benchmark tests. The test harness interacted with DIANNE, providing inputs and collecting outputs for comparison as illustrated in Figure 5. The datasets were stored as individual scripts which were then read into the evaluator one at a time.

When a dataset was fed into the evaluator, first it was randomly divided into 70% training and 30% testing subsets. The training subset was then fed into DIANNE one

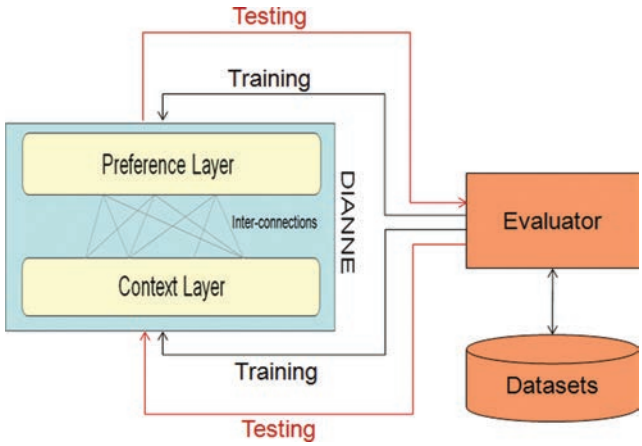

Fig. 5. DIANNE benchmark test harness.

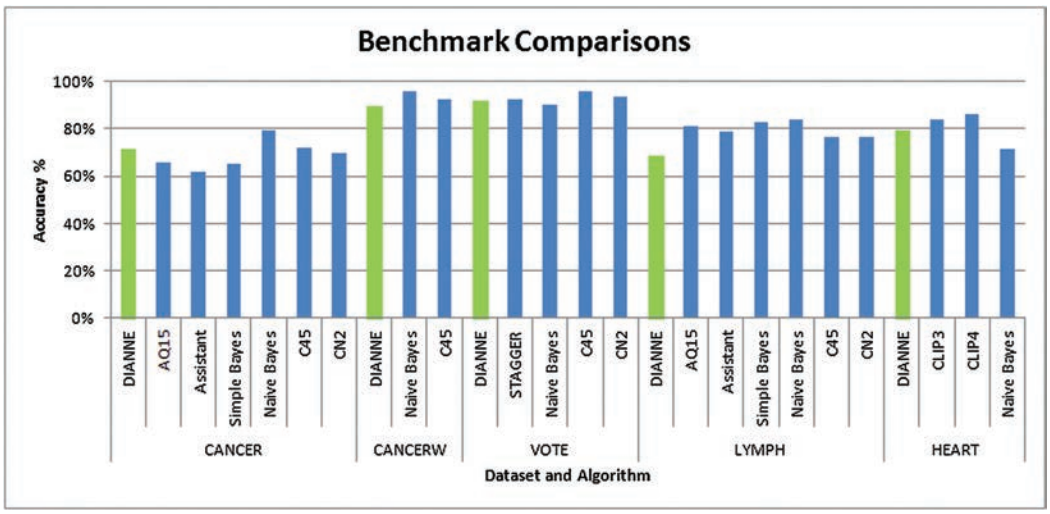

Fig. 6. DIANNE performance, compared against other algorithms across benchmark datasets.

instance at a time with the attributes feeding into the context layer and the class value feeding into the preference layer. DIANNE processed each instance incrementally and updated internal knowledge accordingly.

The testing subset was then fed into DIANNE one instance at a time. The attributes were fed into the context layer and based on this new vector DIANNE returned a class value as output to the evaluator. The evaluator checked the output against the correct class value for that instance and kept a tally of the number of correct and incorrect outputs received from DIANNE. This entire test harness process was repeated ten times for each dataset to give ten percentage accuracies which were then averaged to give one overall percentage accuracy for each dataset.

6.1.1. Results. Figure 6 presents the results of DIANNE on all five datasets. It also shows how the performance of DIANNE compared with that of other well-cited algorithms where comparable results (using the same training and testing set proportions) were available for the same benchmark datasets.

As can be seen, many well-cited machine learning algorithms, both batch and incremental in nature, are available for comparison. Over the five benchmark datasets DIANNE achieves accuracy figures as good, if not better, than other algorithms. Compared to batch algorithms, DIANNE performs comparably with C45 and outperforms CN2, simple bayes, and assistant on the CANCER dataset. The naive Bayes algorithm is outperformed on the HEART and VOTE datasets. These are encouraging results since DIANNE does not utilize a priori knowledge of the entire dataset and does not reprocess past training data. Compared to incremental algorithms, DIANNE outperforms AQ15 on the CANCER dataset and achieves accuracies comparable to that of the STAGGER algorithm on the VOTE dataset.

6.2. DIANNE Real-Time Evaluation

DIANNE was also evaluated in a real-world pervasive environment with end-users. The goal of this evaluation was to test DIANNE's utility as a preference learning solution in a pervasive personalization domain with real-time, incremental inputs. This evaluation was based on a personalized television experience where the learning challenge for DIANNE was to incrementally learn context-dependent viewing preferences for each individual trial participant.

Each trial participant was asked to make several visits to various screens placed around a University department building. On certain visits the participant could choose a channel to watch. They were not given any guidance on how they should make their channel choices and were free to watch channels, switch between channels, and change their minds as they pleased. The learning challenge for DIANNE was to incrementally associate user context with channel selection.

During the trial, participants were also asked to reconsider their channel selections allowing them to change their viewing behaviors if desired. The learning challenge for DIANNE then became one of incrementally adapting internal knowledge to appropriately learn any new overriding behaviors. In this instance, the scope of the trial to facilitate preactions was limited. However, further analysis facilitating preactions is pending and will be reported in future work.

The user trials took place over a two-week period in April 2011. A total of 24 people took part in the trials. Specifically, 75% of the participants were male and 71% of participants were aged between 26 and 35. A trial environment was set up as illustrated in Figure 7. Three plasma screens, A, B, and C (each attached to a PC), were positioned at different locations in a University department building and connected to the local ethernet network. Each screen could play one of 3 different channels. The trial server was a PC connected to the local wireless network. It managed access to the three screens and provided a service interface for user control devices, such as the remote control, to utilize.

The remote control provided a GUI with buttons for the various channels so that trial participants could select channels. Inputs received through the GUI were forwarded to the trial server which directed them to the screen closest to the user. The user's location was provided by an RFID system that identified the location of the RFID tag that the trial participant wore. DIANNE was hosted on the trial server. Here it received context updates (related to the user's location) from the RFID system and preference updates (regarding channel selection) from the remote control GUI. These updates were then processed in real time by DIANNE's context and preference layers. In this way DIANNE could perform incremental learning based on live, temporal inputs. DIANNE outputs, indicating a channel selection, were forwarded to the appropriate screens so that the screens would automatically adapt based on the current user context. Note that each trial participant performed the trial individually and a separate DIANNE was generated for each individual in each trial. This is in line with the intended

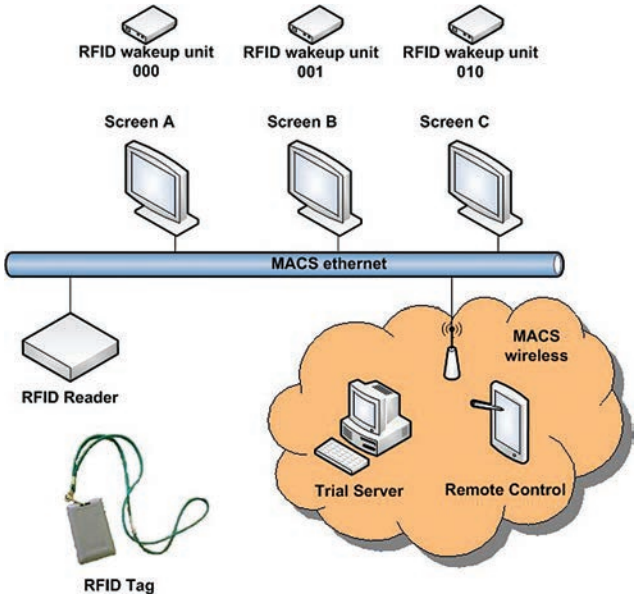

Fig. 7. Trial environment network diagram.

deployment of the DIANNE in a pervasive system where there is a one-to-one mapping between user and DIANNE. Therefore, the trials did not consider conflicts between the preferences of multiple individuals. Such conflicts should be handled by resource sharing mechanisms that are external to DIANNE.

A trial script was produced and strictly adhered for each trial participant. This ensured that all trials were equivalent and helped to reduce any bias that could be imparted on the participant by the trial coordinator. The trial script contained the following steps.

(1) The participant was given an information sheet to read which outlined the format of the trial.
(2) The participant was given an RFID tag to wear around his/her neck, a clipboard with a trial sheet, and the remote control device so he/she could control the screens.
(3) *Primary Selection Circuit*—The participant visited each screen, selected a channel, and wrote the channel, number in a box beside the screen name on his/her trial sheet.
(4) *Primary Test Circuits (PT1, PT2, PT3)*—The participant completed a further three circuits of the screens. During the circuits, if a screen didn't show the correct channel (i.e., the channel that the participant selected and wrote in the box beside that screen name) the participant had to correct the screen using the remote control to set the screen to the correct channel. If a screen did show the correct channel then the participant was not required to perform any action.
(5) *Secondary Selection Circuit*—The participant visited each screen, selected a channel (which may or may not be the channel they selected on the primary selection circuit), and wrote the channel number in a box beside the screen name on his/her trial sheet.
(6) *Secondary Test Circuits (ST1, ST2, ST3, ST4, ST5)*—The participant completed a further five circuits of the screens. As with the primary test circuits if the screen did

Table III. Generated Dataset Descriptions

Dataset	Description
Tester Observations (TO)	During the primary and secondary test circuits the tester notes what channel is automatically presented each time the participant comes into proximity of a screen and whether or not this is the correct channel.
Monitored Behaviour (MB)	All interactions between the participant and the remote control GUI are captured. When the participant interacts with the GUI to select a channel on some screen the channel number is stored with the current location of the participant (provided by the RFID reader). This creates a list of channel-location pairs representing the channel selections made on each screen. This dataset is typical of the monitored user behaviour histories gathered for preference learning in many pervasive systems.
Temporal Monitored Behaviour (TMB)	This dataset is an extension of the Monitored Behaviour dataset. Temporal information is included to represent the duration that each channel-location pair prevails. To achieve this the latest channel-location pair is duplicated in the dataset for every second that it prevails. Therefore this dataset represents the channel selections made on each screen and how long the participant watches the various channels on each screen.

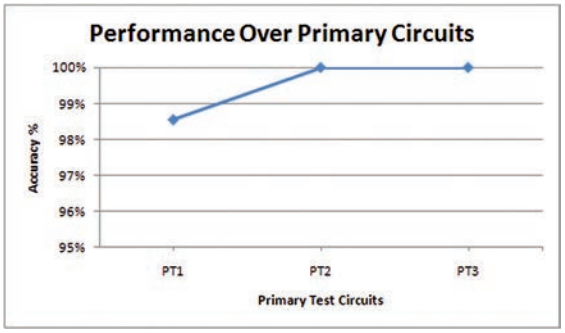

Fig. 8. Graph illustrating the percentage accuracy of DIANNE over the three primary test circuits.

not show the correct channel the participant corrected the screen using the remote control. If the screen did show the correct channel no further action was required.

During each trial a number of datasets were generated for later analysis and processing. Table III describes the datasets that were generated during each trial.

6.2.1. Results. The TO datasets were analyzed to investigate the accuracy and learning rate of DIANNE over the primary and secondary test circuits. On each primary test circuit the tester noted if DIANNE drove the presentation of the correct channel to the participant at each screen. Looking at these figures over 24 trials we can identify a percentage accuracy for DIANNE on each primary test circuit. Figure 8 illustrates that DIANNE retains a high percentage accuracy over the three primary test circuits (PT1, PT2, and PT3) and also shows a slight increase in accuracy as DIANNE improves internal knowledge over time with each circuit.

During the secondary test circuits the tester noted if DIANNE drove the presentation of the correct channel to the participant at each screen. Taking these figures over 24 trials we can identify a percentage accuracy for DIANNE on each secondary test circuit.

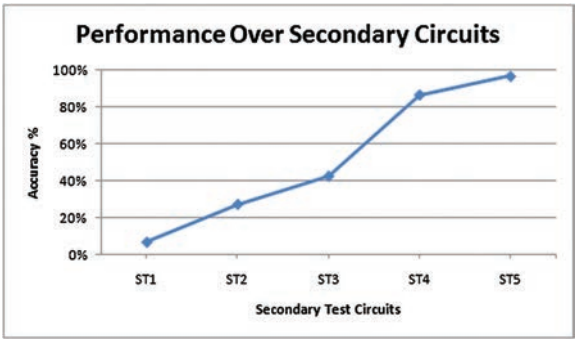

Fig. 9. Graph illustrating the percentage accuracy of DIANNE over the five secondary test circuits.

Figure 9 illustrates that DIANNE accuracy is below 10% on ST1 but increases rapidly to an accuracy above 95% by ST5.

The initial low accuracy value of 10% is expected due to the unpredictable change in behavior (new channel selections) on the secondary selection circuit (that took place between PT3 and ST1). DIANNE recovers from this unpredictable behavior change over time, as illustrated by the curve, and returns to high accuracy values within 4 test circuits. This compares favorably with other approaches tackling concept drift such as Kolter and Maloof [2007] where the recovery period is around 20 timesteps.

The Monitored Behavior (MB) datasets are a list of action-context pairs that can be applied to other learning algorithms to give a comparison with DIANNE performance. In this instance the C45 decision tree building algorithm was chosen as the algorithm for comparison as it has been utilized by the authors for preference learning in the DAIDALOS project due to both its accuracy and tree-based output which can be translated into human-readable form.

At the end of each trial DIANNE and C45 algorithms were tested to see if they could correctly predict the participant's secondary viewing preferences (i.e., the channel selections that the participant made during his/her secondary selection circuit). The MB dataset was applied to the C45 algorithm and from the tree-based output an IF-THEN-ELSE preference rule was generated indicating a channel number for each location. This was compared with the participant's actual secondary viewing preferences to give an accuracy figure for the C45 algorithm. The preferences held in DIANNE's final state were also compared with the participant's secondary viewing preferences to give a final accuracy figure for DIANNE.

In addition, the Temporal Monitored Behavior (TMB) dataset was also applied to the C45 algorithm at the end of each trial. The TMB dataset included extra context-action pairs extending the MB dataset with temporal information. The TMB dataset replicated DIANNE input allowing the C45 algorithm to take advantage of temporal data. The reason for this additional dataset is to ensure that the C45 algorithm was not hampered by less inputs or data than DIANNE. As with the MB dataset, the TMB dataset was applied to the C45 algorithm and from the tree-based output an IF-THEN-ELSE preference rule was generated indicating a channel number for each location. This was compared with the participant's secondary viewing preferences to give an accuracy figure for the C45 algorithm operating on a temporal dataset. Taking the accuracies of the DIANNE, C45(MB) and C45(TMB) over all 24 trials gives an average accuracy for each algorithm, illustrated in Figure 10.

The graph shows that DIANNE is over three times more accurate at learning the participant's secondary viewing preferences as the C45 algorithm on the MB dataset. The graph also shows that there is some improvement in the accuracy of the C45

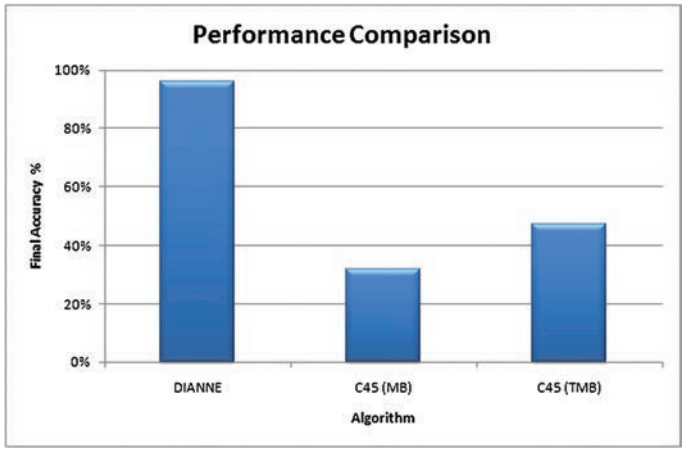

Fig. 10. Graph illustrating the final accuracy of DIANNE and the C45 algorithm on a nontemporal dataset (MB) and the C45 algorithm on a temporal dataset (TMB) over all 24 trials.

algorithm on the TMB dataset suggesting that temporal information is of benefit. However, even with added temporal information the C45 algorithm still only achieves an accuracy of less than 50%, roughly half that of DIANNE at 96%. In the case of the TMB datasets both algorithms have essentially received the same input except DIANNE processes inputs in real time as they occur throughout the trials whereas the C45 algorithm processes all inputs in batch after the trial is completed.

Looking more closely at the IF-THEN-ELSE preferences generated by the C45 algorithm on the MB and TMB datasets it can be seen that they often portray the participant's primary viewing preferences (i.e., the channels he/she selected during the primary selection circuit). This is understandable since most participants spent significantly longer time selecting channels on their first primary selection circuit than on their second, meaning that their primary channel selections often appear more frequently in the MB and TMB datasets.

It is understood that these test results do not provide a like for like comparison as the C45 algorithm is not designed for changing rules, but rather for static situations where recent data is not weighed more heavily than less recent data. No tree pruning techniques have been applied in this instance and it is understood that C45 favors problem domains with larger numbers of attributes. However, the MB and TMB datasets illustrate a typical preference learning situation. User preferences are not static and often change over time for various reasons. What this comparison illustrates is that algorithms such as C45 are perhaps not best suited to the preference learning problem domain. By comparison DIANNE can respond more rapidly to changes in behavior, returning to a high performance accuracy in an acceptable time frame (in this case, within a few test circuits). This reflects the findings of Segal and Kephart [2000] when comparing incremental and batch learning techniques for the Swiftfile system. DIANNE can also match many nonnetwork-based algorithms in terms of its translatability into human-understandable form. This is highly important in the pervasive domain where internal knowledge should be available for presentation to end-users.

7. CONCLUSION

An analysis of existing preference learning systems provides two key observations. First, incremental machine learning algorithms are rarely used for preference learning even though they seem to better align with the incremental nature of the problem

domain. Second, temporal data relating to the duration with which context states and preference settings cooccur is typically ignored. Contrary to typical trends, this article proposes two hypotheses: A and B. Hypothesis A proposes that an incremental learning algorithm is preferential over batch algorithms for preference learning. Hypothesis B proposes that temporal information is a beneficial input to the preference learning process.

DIANNE is presented as a tailored solution to the challenge of preference learning in pervasive environments. It conforms with the two hypotheses to provide a platform to test their validity. DIANNE is a single-layer neural network that learns associations between context and preference states to identify what preference settings to apply in a given context. The exclusion of hidden layers allows for rapid and noncomplex updating of internal knowledge. Although this means that DIANNE cannot represent nonlinear problems such as XOR it is shown that the latter case will never need to be handled in this problem domain. A single-layer topology also enables the network to be easily translated into a human-understandable form which is a key requirement for end-users.

DIANNE is an incremental learning system processing inputs one at a time as they occur in real time. Hence it can respond rapidly to changes in user behavior, it is not dependent on a store of user behavior history, and it does not need to reprocess past inputs. It employs a temporal learning algorithm that executes continuously in a lifelong manner, taking advantage of temporal preference information to overcome user behavior anomalies such as preparational actions. Associations between context and preference outcomes are strengthened and weakened based on the temporal duration in which a context state renders some preference setting to be implemented.

An incremental conflict resolution scheme enables DIANNE to deal with unpredictable changes in user behaviors. Conflicts are resolved at one instance in time based on current knowledge and without the need to reprocess past inputs. The stochastic gradient descent learning rule is used in conjunction with the Hebbian learning rule to accommodate new behaviors into internal network knowledge in a reasonable time frame without giving the perception of one-instance learning.

DIANNE was tested and evaluated both in terms of performance and scalability as a machine learning algorithm and in terms of its utility as a preference learning solution in a real-time environment. Benchmark datasets were applied to DIANNE to simulate situations with multiple attributes. The performance of DIANNE on such datasets was presented and compared with that of other well-cited algorithms, both batch and incremental in nature. Overall, DIANNE performed favorably across all datasets and in some cases outperformed its counterparts.

DIANNE was also deployed in live user trials to evaluate its utility as a preference learner in a real-time environment with real end-users. The results showed that DIANNE is able to identify initial viewing preferences very rapidly, meaning that the participant is almost always shown the correct channel at each screen on the primary test circuits. The results also show that during the secondary test circuits, DIANNE accuracy dropped immediately after the user changed his/her viewing behavior. However, the accuracy steadily increased to almost 100% over the course of the secondary test circuits, showing that DIANNE can rapidly recover from unpredictable shifts in user behavior.

When comparing DIANNE with the C45 tree building algorithm on temporal and nontemporal datasets, the results showed that DIANNE outperformed the C45 algorithm regardless of whether the temporal or nontemporal dataset was used. However, C45 performance is improved with the temporal dataset suggesting that extra temporal information is of benefit. These results reflect the findings of several other works

that concluded an incremental learning system is the optimal solution in incremental problem domains such as preference learning in pervasive environments.

Reflecting on Hypothesis A, the results seem to indicate that an incremental learning algorithm can respond more rapidly to new inputs and behavior changes. Based on the findings of these trials one could conclude that the hypothesis is correct. Reflecting on Hypothesis B, the trial results seem to indicate that this hypothesis is also correct as an improvement in performance is observable when temporal data was included as input to the C45 batch algorithm.

As well as a stand-alone implementation of DIANNE, it has also been fully integrated as a key preference learning system within the Personal Smart Space (PSS) platform developed by the PERSIST project [Crotty et al. 2008]. The capabilities of DIANNE were successfully demonstrated at a final project review and DIANNE is now being utilized as a key preference learning system in the SOCIETIES project [SOCIETIES 2010].

ACKNOWLEDGMENTS

The authors wish to thank their colleagues in these projects.

REFERENCES

ADIDAS_1. 2005. The adidas_1 running shoe review. http://www.gizmag.com/go/3810

BARKHUUS, L. 2003. Is context-aware computing taking control away from the user? Three levels of interactivity examined. In *Proceedings of the 5th Annual Conference on Ubiquitous Computing (UbiComp'03)*. Lecture Notes in Computer Science, vol. 2864, Springer, 149–156.

CORDIER, C., CARREZ, F., VAN KRANENBURG, H., LICCIARDI, C., VAN DER MEER, J., SPEDALIERI, A., LE ROUZIC, J. P., AND ZORIC, J. 2006. Addressing the challenges of beyond 3g service delivery: The spice service platform. In *Proceedings of Workshop on Applications and Services in Wireless Networks (ASWN'06)*.

CROTTY, M., TAYLOR, N. K., WILLIAMS, M. H., FRANK, K., ROUSSAKI, I. AND RODDY, M. 2008. A pervasive environment based on personal self-improving smart spaces. In *Proceedings of Architectures and Platforms for AMI, Workshop on European Conference on Ambient Intelligence (AmI'08)*. 58–62.

DEERING, S. AND HINDEN, R. 1995. Internet protocol version 6 (IPv6) specification, rfc 1883. http://www.ietf.org/rfc/rfc1883.txt

DEY, A. K. 2009. Modeling and intelligibility in ambient environments. *J. Ambient Intell. Smart Environ. 1,* 1, 57–62.

FISHER, D. H. 1987. Knowledge acquisition via incremental conceptual clustering. *Mach. Learn. 2*, 2, 139–172.

GALLACHER, S. 2011. Learning preferences for personalisation in pervasive environments. Ph.D. thesis. Heriot-Watt University, Scotland.

GALLANT, S. I. 1990. Perceptron-Based learning algorithms. *IEEE Trans. Neural Netw. 1*, 2, 179–191.

GERSHENFELD, N., KRIKORIAN, R., AND COHEN, D. 2004. The internet of things. *Sci. Amer. 291*, 4, 76–81.

GIRAUD-CARRIER, C. 2000. A note on the utility of incremental learning. *AI Comm. 13*, 4, 215–223.

HAGRAS, H. 2007. Embedding computational intelligence in pervasive spaces. *IEEE Pervas. Comput. 6*, 3, 85–89.

HEBB, D. O. 1949. *The Organization of Behaviour*. Wiley & Sons, New York.

HERTZ, J., KROGH, A., AND PALMER, R. G. 1991. *Introduction to the Theory of Neural Computation*. Addison-Wesley. Apple iPad specification. http://www.apple.com/uk/ipad, IPHONE 2012. Apple iPhone specification. http://www.apple.com/iphoneIPAD 2012.

IPAD 2012. Apple iPad specification. http://www.apple.com/uk/ipad

KOLTER, J. Z. AND MALOOF, M. A. 2007. Dynamic weighted majority: An ensemble method for drifting concepts. *J. Mach. Learn. Res. 8*, 2755–2790.

LITTLESTONE, N. 1988. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Mach. Learn. 2*, 4, 285–318.

MICHALSKI, R. S., MOZETIC, I., HONG, J., AND LAVRAC, N. 1986. The multi-purpose incremental learning system aq15 and its testing application to three medical domains. In *Proceedings of the 5th International Conference on Artificial Intelligence*. 1041–1045.

MISTRY, P. AND MAES, P. 2009. Sixth sense: A wearable gestural interface. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques*.

MOZER, M. C. 1998. The neural network house: An environment that adapts to its inhabitants. In *Proceedings of the American Association for Artificial Intelligence Spring Symposium on Intelligent Environments*. 110–114.

QUINLAN, J. R. 1986. Induction of decision trees. *Mach. Learn. 1*, 1. 81–106.

SATYANARAYANAN, M. 2001. Pervasive computing: Vision and challenges. *IEEE Personal Comm. 8*, 4, 10–17.

SCHLIMMER, J. C. AND FISHER, D. 1986. A case study of incremental concept induction. In *Proceedings of the 5th National Conference on Artificial Intelligence.* 496–501.

SCHLIMMER, J. C. AND GRANGER, R. H. 1986. Incremental learning from noisy data. *Mach. Learn. 1*, 3, 317–354.

SEGAL, R. B. AND KEPHART, J. O. 2000. Incremental learning in swiftfile. In *Proceedings of the 17th International Conference on Machine Learning*. 863–870.

SI, H., KAWAHARA, Y., MORIKAWA, H., AND AOYAMA, T. 2005. A stochastic approach for creating context-aware services based on context histories in smart home. In *Proceedings of the 3rd International Conference on Pervasive Computing, Exploiting Context Histories in Smart Environments (ECHISE'05).*

SOCIETIES. 2010. The societies eu fp7 project website. http://www.ict-societies.eu

SUTTERER, M., COUTAND, O., DROEGEHORN, O., AND DAVID, K. 2007. Managing and delivering context-dependent user preferences in ubiquitous computing environments. In *Proceedings of the International Symposium on Applications and the Internet Workshops (SAINTW'07).*

TAYLOR, N. K., ROBERTSON, P., FARSHCHIAN, B. A., DOOLIN, K., ROUSSAKI, I. G., MARSHALL, L., MULLINS, R., DRUESEDOW, S., AND DOLINAR, K. 2011. Pervasive computing in daidalos. *IEEE Pervas. Comput. 10*, 1, 74–81.

UCI. 2007. Online machine learning repository. http://archive.ics.uci.edu/ml

WEBB, G. I., PAZZANI, M. J., AND BILLUS, D. 2001. Machine learning for user modeling. *User Model. User-Adapt. Interact. 11*, 19–29.

WEISER, M. 1991. The computer for the 21st century. *Sci. Amer. 265*, 3, 94–104.

YOUNGBLOOD, G. M., HOLDER, L. B., AND COOK, D. J. 2005. Managing adaptive versatile environments. *Pervas. Mobile Comput. 1*, 4, 373–403.

ZIEBART, B. D., ROTH, D., CAMPBELL, R. H., AND DEY, A. K. 2005. Learning automation policies for pervasive computing environments. In *Proceedings of the 2nd International Conference on Autonomic Computing (ICAC'05).* 193.

Received October 2011; revised April 2012; accepted August 2012