[SHRS08] *Johannes Schmitt, Matthias Hollick, Christoph Roos, Ralf Steinmetz;* **Adapting the User Context in Realtime: Tailoring Online Machine Learning Algorithms to Ambient Computing.** In: Mobile Networks and Applications, vol. Volume 13, Number 6 / Dezember 2008, October 2008. ISSN 1383-469X (Print) 1572-8153 (Online).Seite 583 - 598

Mobile Netw Appl (2008) 13:583–598
DOI 10.1007/s11036-008-0095-8

Adapting the User Context in Realtime: Tailoring Online Machine Learning Algorithms to Ambient Computing

Johannes Schmitt · Matthias Hollick · Christoph Roos · Ralf Steinmetz

Published online: 3 October 2008

Abstract Ambient systems weave computing and communication aspects into everyday life. To provide self-adaptive services, it is necessary to acquire context information using sensors and to leverage the collected information for reasoning and classification of situations. To enable self-learning systems, we propose to depart from static rule-based decisions and first-order logic to define situations from basic context, but to build on machine-learning techniques. However, existing learning algorithms show substantial weaknesses if applied in highly dynamic environments, where we expect accurate decisions in realtime while the user is in-the-loop to give feedback to the system's recommendations. To address ambient and pervasive computing environments, we propose the FLORA—multiple classification (FLORA-MC) online learning algorithm. In particular, we enhance the FLORA algorithm to allow for (1) multiple classification and (2) numerical input values, while improving its concept drift handling capabilities; thus, making it an excellent choice for use in the area of ambient computing. The multiple classification allows context-aware systems to differentiate between multiple categories instead of taking binary decisions. Support for numerical input values enables the processing of arbitrary sensor inputs beyond nominal data. To provide the capability of concept drift handling, we propose the use of an advanced window adjustment heuristic, which allows FLORA-MC to continuously adapt to the user's behavior, even if her/his preferences change abruptly over time. In combination with the inherent characteristics of online learning algorithms, our scheme is very well suited for realtime application in the area of ambient and pervasive computing. We describe the design and implementation of FLORA-MC and evaluate its performance vs. state-of-the-art learning algorithms. We are able to show the superior performance of our algorithm with respect to reaction time and concept drift handling, while maintaining an excellent accuracy. Our implementation is available to the research community as a WEKA module.

Keywords ambient computing · pervasive computing · context-aware computing · machine learning

J. Schmitt · M. Hollick (✉) · C. Roos · R. Steinmetz
Multimedia Communications Lab (KOM),
Department of Electrical Engineering and
Information Technology, Technische Universität Darmstadt,
Merckstr. 25, 64283 Darmstadt, Germany
e-mail: matthias.hollick@kom.tu-darmstadt.de

1 Introduction

1.1 Introduction and motivation

The accurate determination of the context of users and devices is the basis for ambient computing systems. In order to adapt to changing demands and environments, the system needs to reason based on basic context facts to determine knowledge of higher level situations. Existing reasoning techniques include simple first-order logic and static rule-based systems to infer about the user's situation. However, to adequately match the requirements of ambient computing such as convenience and flexibility, the aforementioned techniques are considered to be too intrusive and too hard to configure and maintain. We focus our work on machine learning algorithms to aid in building and maintaining decision models. Designed to provide a more natural interaction between humans and computers, one of the key success factors of these systems is the ease of use, i.e., the system's ability to continuously adapt to the user's situation, the

immediate response to user feedback, etc. Existing machine learning approaches for context determination mostly rely on offline learning algorithms, which are highly accurate. However, accuracy comes at the expense of a slow model-building process, large memory consumption, and—for plain offline learning algorithms—the inability to handle concept drift, i.e., the seamless adaptation to changing user demands, appropriately (see [1]).

Application scenarios such as context-aware communications or—more in general—ambient computing and communications have a set of pivotal requirements, which would strongly benefit, if the model-adaptation process could be performed in realtime, thus conserving the precious resource of user attention and waiting time. However, there is a lack of adequate online learning algorithms, which could push context-aware systems into the realtime realm.

This is particularly true, if we consider advanced features of the learning algorithm, which are crucial to support ambient services. Additional requirements for learning algorithms to be applied in the aforementioned scenarios are, e.g., the support of multiple classifications, i.e., the ability of the model to differentiate between multiple result categories, the support to handle numerical input values, and the handling of concept drift, i.e., the very fast adaptation to abruptly changing user preferences over time.

Due to the lack of applicable online learning algorithms, offline learners in combination with window or weighting mechanisms are often used as alternatives (see [2]). When a concept drift occurs, these window mechanisms preferably delete old history elements (see also Section 2.2), which leads to problems with multiple categories, i.e., rarely used categories are not supported correctly by the resulting model. Instead of model updates, offline learners typically perform complete model building, which can be very costly, if the amount of samples or input values rises. Online learning algorithms are able to detect changes and adapt only the related parts of the model, thus providing for fast adaptation of the model.

Given the aforementioned assumptions and requirements of the ability to (1) classify context into multivariate categories (multiple classification), while (2) supporting arbitrary numerical input values and promptly adapting to changing user preferences (numerical input and concept drift), we discuss the following open research questions:

- How to combine the realtime model-building capabilities of online machine learning algorithms to allow classification of contexts into multiple categories?
- How to support numerical instead of nominal input, while maintaining the capability of online machine learning algorithms to handle concept drift appropriately?

1.2 Contribution

We answer these research questions by introducing the concepts to enhance online machine learning algorithms for use in the application domain of context-aware and ambient computing. In particular, our contribution is as follows:

- We design the FLORA-MC algorithm, which combines the online learning capabilities of the FLORA algorithm [3] with multiple classification.
- We allow for processing of numerical input values in FLORA-MC instead of nominal values only in FLORA.
- We present an extended window adjustment heuristic (WAH) to address the concept drift problem. Our heuristic is able to trade-off fast reaction time to concept drift vs. optimality of the classification accuracy.
- We enhance our WAH to be stable to context fluctuations by evaluating the Shannon-entropy of the classification process.
- We realize an implementation of FLORA-MC, which can be applied and compared with other algorithms within the commonly used data-mining framework WEKA [4].
- We perform extensive experimentation to analyze the performance of our algorithm in various representative settings.

To the best of our knowledge, our work is the first one to enhance online learning algorithms to provide an adequate feature set supporting advanced context-aware services; thus, pioneering the application of realtime model-building capabilities within ambient and pervasive systems.

1.3 Outline

In Section 2, we survey related work in literature and discuss the FLORA algorithm, which serves as the basis for our work. In Section 3, we describe the design and working of FLORA-MC and give implementation details. We put particular emphasis on the multiple classification of context and the window management to optimize the handling of concept drift. In Section 4, a thorough evaluation of the proposed algorithms is presented. This is followed by Section 5, which summarizes our findings and gives pointers to future work.

2 Background and related work

This section first introduces an application scenario representing the class of targeted ambient applications. Next, related work in general that deals with ambient and context-aware computing is surveyed. We then discuss the application of machine learning algorithms in context-aware

systems and introduce/define the terminology of these systems. We briefly survey the existing classes of algorithms with emphasis on online learning algorithms. In particular, we discuss the FLORA approach, which serves as the basis for our work.

2.1 Scenario and background

As a sample scenario we chose a context-aware multimedia communication system such as introduced by Görtz in [5]. The communication means in ambient environments are characterized by an abundance of different communication technologies, services, and applications such as voice over IP, instant messaging, email, video-conferencing, or telepresence/teleimmersion. These means are often coupled with technical or social implications. For instance, video-conferencing demands an appropriate technical infrastructure for all participants; or on the dimension of social context, the user might be embarrassed if her/his mobile phone rings during a business meeting. In our sample application, the goal is to optimally support the user in managing her/his communication needs, while being agnostic to the different communication channels as shown in Fig. 1.

Let us assume that Alice wants to talk to Bob. He might or might not be available or wanting to accept the call. If Alice realizes that she cannot reach Bob via phone, she might want to contact him via email, instant messaging, or leave a message on his voice mailbox. However, to do so, both Alice and Bob have to constantly update the configuration of their communication services; as of today automation of appropriate call handling requires setting of complex rules to govern the system's decisions.

Ambient support for a smart communication service could leverage a virtual assistant to control the handling of calls (see, e.g., Schmitt et al. [6]). In particular, the context-information about both communication partners and their environment can be used to determine the appropriate course of action. While being a simplistic example, the system's requirements are already very challenging and also capture key requirements of ambient computing systems. This includes but is not limited to appropriate

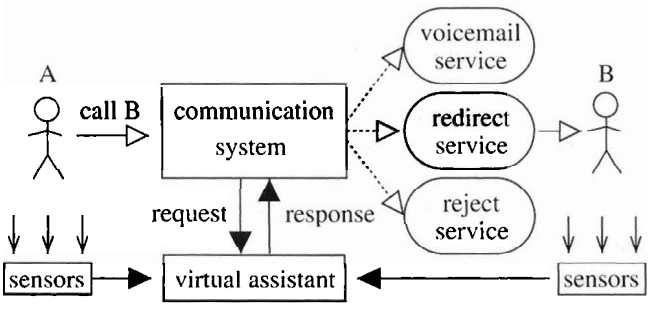


Figure 1 Sample application of a smart communication service

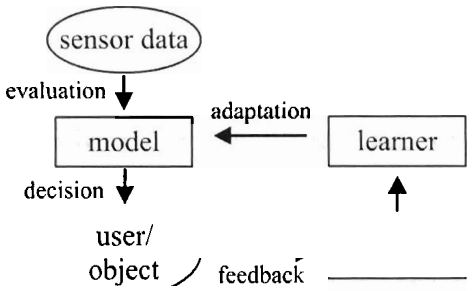


Figure 2 Architecture for context determination and model adaptation

context acquisition and evaluation in realtime in highly dynamic, complex environments. Furthermore, the service should be self-learning with minimal user interaction, and feedback of the user should be incorporated for future decisions immediately.

Fundamental work in the area of context-aware and pervasive computing has been pursued by Schilit [7] with the Mobile Context-Aware Computing system, Dey et al. [8, 9] with the Context Toolkit, and Schmidt et al. [10, 11] in the Technology for Enabling Awareness (TEA) architecture. A huge body of work has refined and extended on this work and pushed it closer to real-world applications. Recent efforts include the European projects Amigo [12] and Ambient Networks [13], which aim to realize a smart living environment and an ambient communication platform, respectively. In our work, we focus on the decision component within ambient computing systems. Our goal is to provide one building block for these, namely an optimally suited online machine-learning algorithm.

2.2 Applying machine learning for context-aware systems

In order to introduce a common terminology, Fig. 2 shows an abstracted architecture of a context-aware system, which is designed to adapt the applied decision model as follows.

1) Incoming sensor data is evaluated using the actual decision model.
2) As a result, the decision model returns the determined category, which can be used by the requesting instance, e.g., to trigger an event or to configure devices.
3) The object/user can feedback the system with information whether the decision has been correct, e.g., by entering the demanded category into the system.
4) The feedback in combination with the related sensor data forms a sample, which can be seen as a snapshot of the user demand.
5) The sample is used to update the model or to generate a new model in combination with a history of recent samples.

6) The adapted model is used for further evaluation-requests.

It is obvious that the decision model needs to be created and maintained to capture an accurate representation of the user demand for a dynamic set of available sensors and sensor-types. This model adaptation is performed by machine learning mechanisms, which determine the patterns between the input values (sensor data) and the output values (decisions/categories). Discovered patterns will be represented by model-specific expressions within the model. For our application domain, the model and the algorithm for model adaptation have to fulfil the following *requirements*.

a) Classification of a situation into a user-defined set of categories.
b) Reaction on changing user demand as fast as possible.
c) Evaluation of sensor data in realtime.
d) Low system-requirements and effort for model building.
e) Accurate decisions in presence of dynamic sets of sensor data (e.g., noise, missing data).

2.3 A brief survey on machine learning algorithms for context-aware systems

While several classes of machine learning algorithms exist, only a few of them are suited for our application domain. Figure 3 shows a classification of relevant algorithms. We next briefly discuss the feasibility of the individual algorithm classes for the given requirements.

Requirements (a) and (b) indicate the use of the class of *supervised learner*, whereas req. (c) prohibits the use of *lazy learners*. For our problem, we can thus chose from the *eager learners*: algorithms of the *online learner* class are designed to cater to req. (b) and (c), while algorithms of the *offline learner* class are particularly useful for req. (e).

Offline learners (or batch learners) are designed to handle common data-mining tasks. Due to their long history, many well-engineered algorithms are available, e.g., algorithms to generate support vector machines (SVM), Neural Networks (NN) or Decision Trees (DT). Enhancing *offline learners* with a meta-algorithms to introduce window management allows to fulfil reqs. (b) and (e). However, since most *offline learners* need to build up an entirely new model each time instead of incrementally updating the existing model, req. (d) cannot be easily fulfilled. We here omit the further discussion of other meta-algorithms such as n-way binary classification to address req. (a), since the violation of req. (d) outweighs the possible gain of these mechanisms for our scenario.

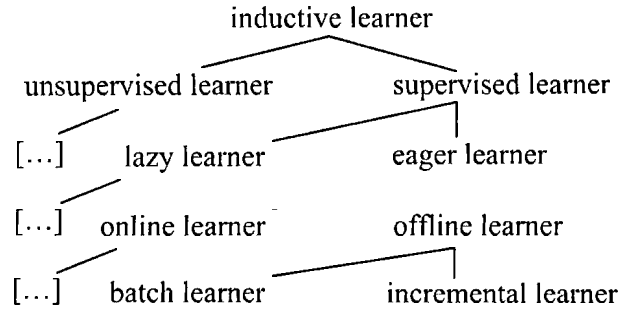


Figure 3 Overview of machine learning algorithms for context-aware systems

In contrast, *incremental learners* (such as algorithms for generation of Naive Bayes (NB) models) and *online learners* have the feature to update existing models for each new sample, thus, satisfying req. (d). *Online learners* use several sample (instance) storage strategies. *Online learner without instance storage* have good results regarding reqs. (b) and (d), however, often at the expense of lower classification accuracy compared with *offline learners*. *Partial instance storage* can be applied for *online learners*. Here, sets of training instances are managed within the model enhancing the model's "knowledge", which increased the classification accuracy.

2.4 Online learning algorithms and FLORA

The class of online learning algorithms provides promising features for our class of applications. Plain online learners, which do not have an instance management and do not store samples, enable fast adaptation to new concepts, because new samples overrule old and conflicting samples. Due to this behavior, however, they are typically not as precise as offline learners. Examples for online learning algorithms are STAGGER and Winnow [14]. To compensate for the imprecision of online learners, either statistic extensions and heuristics are applied or partial instance storage is implemented. Online learners featuring the latter extension can be described as hybrids between incremental and online learning algorithms. They store and process a set of current samples to compensate the loss of information, which results during model building (e.g. by discretization) [15]. Examples for algorithms using this concept are LAIR [16], HILLARY [17], and FLORA [3].

For our work, we have chosen FLORA ("FLOating Rough Approximation") as a basis, because it is considered a de-facto standard among online learning algorithms. FLORA has been incrementally developed and comes in four versions (FLORA-1 to FLORA-4). The algorithm generates a rule-based model, which has the ability to make binary decisions using discrete input data. A rule is related to at least one sample, which is stored within the included instance management. Rules are removed, if all referring samples are discarded by the instance management.

FLORA manages three different rule sets: the *Accepted DEScriptor-Set* (ADES) to represent positive rules, the *Potential DEScriptor-Set* (PDES) representing neutral rules, and the *Negative DEScriptor-Set* (NDES) representing negative rules.

The FLORA model can be instantaneously adapted, by creating/assigning into these sets or by moving rules between these sets appropriately. The decision process is performed by matching incoming samples against the ADES-Set, while the other sets are used during the model adaptation in order to generalize or depreciate rules.

FLORA-2 extends the instance management of the basic FLORA concept by adding a flexible window size, which is set by monitoring the model's stability and accuracy. During the initial training phase the window size increases, whereas the size is kept constant in steady state (i.e. for stable concepts). If a concept drift is detected the window size is decreased, thus, leading to fast adaptation to new concepts. However, if concept drift is mis-indicated, the classification accuracy suffers.

FLORA-3 introduces a "long-term mind" in order to handle incorrectly indicated concept drifts. During steady state, the current model (all rule-sets) is stored. If—after a concept drift—the latest concept matches a concept within the long term mind, the old concept is restored, thus shortening the learning cycle. This leads to increased accuracy and allows for a more sensitive configuration of the concept drift detection. A drawback of FLORA-3 is the memory intensive storage of old models, which depends on the number of models as well as on the size of the individual rule sets.

FLORA-4 has been designed to also handle noisy or contradicting feedback. To hinder the removal of valid concepts due to noise in samples (FLORA-1–3 reacts by invalidating even well-established concepts if encountering a single contradicting sample), the model allows a certain amount of contradiction. As a result, FLORA-4 is much more robust to noisy input data, however, resulting in a much slower convergence of the model, which is also hindering the swift reaction to concept drift.

2.5 Summary

We identified the key requirements of context-aware and pervasive computing for machine learning algorithms. Online algorithms fulfil most of these requirements, as they enable fast evaluation as well as fast model building. We identified FLORA as a lightweight and accurate candidate algorithm to operate with low memory consumption and providing an excellent basis for concept drift handling. For our intended extension, FLORA-3 and FLORA-4 add unnecessary complexity, thus, we base our work on FLORA-2.

3 FLORA-MC

We introduce FLORA-Multiple Classification (FLORA-MC) to address the aforementioned short-comings of existing machine learners. In particular, we detail the design and implementation of FLORA-MC with particular emphasis on the key mechanisms of the algorithm.

3.1 Algorithm design

FLORA-MC is based on FLORA-2 [3]. Main drawbacks of the FLORA family are the restrictions to nominal input values (sensor values for training instances) and binary output values (classifications) only, which are severe limitations for the envisioned application area. At the same time, the design principle of FLORA provides a powerful basis for our work, because the inherent characteristics of this learner are very promising. Our design, thus, is based on the construction principle of FLORA, but replicates the main design element, i.e., the storage for description items, to match the number of target categories. To ensure proper operation of FLORA-MC, we have to realize the handling of multiple descriptor sets instead of the unique sets in FLORA.

We next describe the internal working of our algorithm. The processing of input values of FLORA and FLORA-MC is based on classification rules, so-called description items (DI), which are composed out of different combinations of the attribute values of the training instances; e.g., a DI for FLORA-MC could be $size = small \wedge diameter = [2.2; 2.9] \wedge temperature = [30; 40] \wedge color = green$. Please note that only numerical values are represented using intervals, but nominal values can only be uniquely represented in the description item (i.e., $size = \{small, medium\}$ is not permitted). DIs are constructed or extended during the learning process (see Section 3.2 below for details on learning and forgetting). The FLORA family stores the obtained DIs in its ADES, PDES, and NDES sets. For FLORA-MC, the division in three subsets as well as the (logical) role of the sets is kept. However, we depart from the limitation of only one group of description sets (ADES, PDES, NDES) per category, which results in a more complex interrelation between the individual sets (see Fig. 4):

> The ADES-Set m contains all DIs, which describe category m, i.e., all corresponding training instances are representing category m.
> The NDES-Set m contains all DIs, which do *not* describe category m. Training instances are of category $1 \ldots n$ without m. However, this does not imply that the NDES DIs of category m form the union of all DIs of the ADES-Sets from $1 \ldots n$ without m.

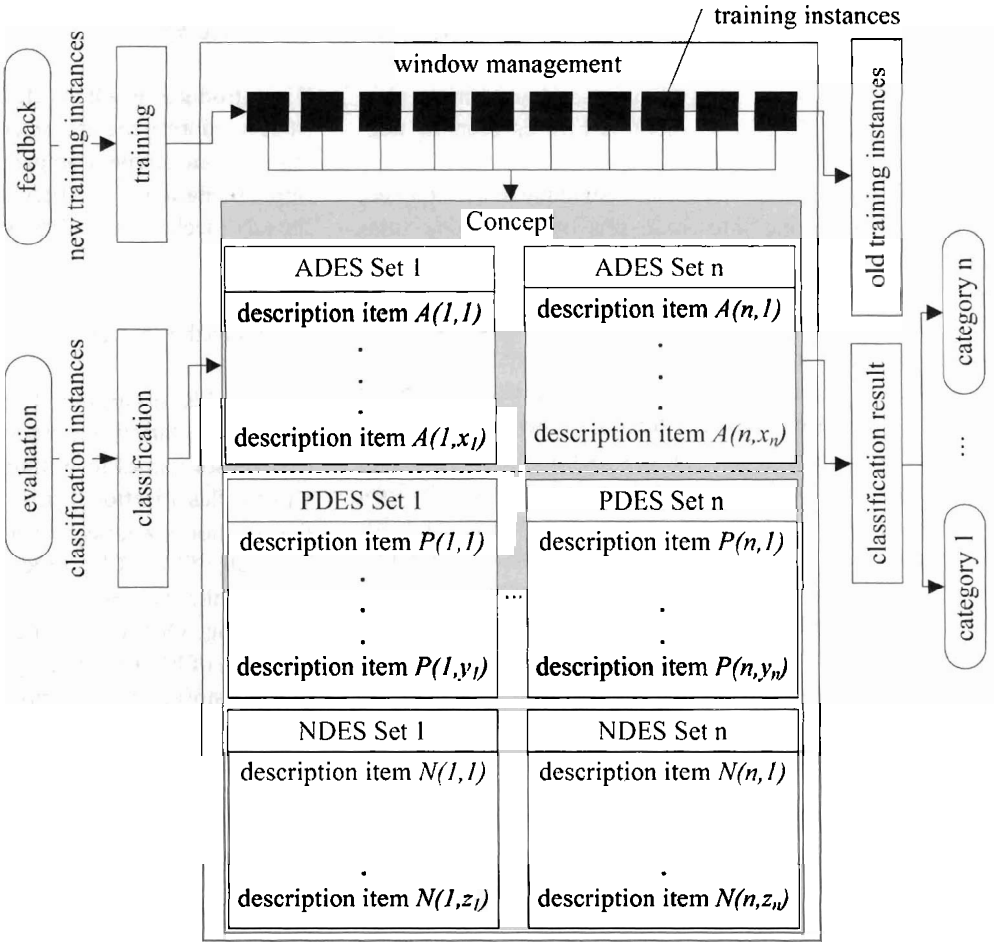


Figure 4 Working principle of FLORA-MC. The original FLORA algorithm manages only unique ADES, PDES, and NDES sets

– The PDES-Set m contains all DIs, which are supported by instances of category m as well as instances different from m.

Description items are only kept in the corresponding set, as long as training instances are within the range of the window, but are removed as soon as this is no longer true. The above example could be part of ADES for category "bad" and NDES for category "nice" and "fair" (the example is based on the STAGGER-methodology for evaluation of our learning algorithm, which is further elaborated in Section 4.1).

3.2 Learning and forgetting for multiple classifications

The key mechanisms of FLORA are the process of knowledge acquisition (learning) as well as the process of discarding knowledge (forgetting). The learning process is straightforward: new instances passed to the learning system enter the window and trigger a call of function *learn()* (see Fig. 5). For the ease of following the discussion, we keep the notation of [3] as far as possible.

Function *learn()* is called as *learn(I, $ADES_i$, $PDES_i$, NDES)* and works as follows. (1) A new training instance (sample) I for $ADES_j$ enters the system/window. (2) The algorithm checks, if instance I matches (nominal values are equal, numerical values fall into the given interval) an existing DI (ADI_i) in ADES-Set $ADES_j$. For a non-matching instance, the algorithm tries to generalize an existing DI (ADI_i) in ADES-Set $ADES_j$. For matching items, the supporting counter of the matching DI is incremented. The support counter ultimately determines, if DIs are either kept in the ADES-Set ($AP_i > 0$) or removed ($AP_i = 0$). Similar counters exist for NDES (NN_i). (3) The function *generalize()* is called to extend an existing DI with respect to its numerical values/intervals. Only DIs that have all nominal values in common with the new sample are considered.[1]

[1] Please note that the original FLORA algorithm performs the generalization differently: nominal values are directly removed from the DIs (we keep these nominal values and extend the intervals of the numerical values only). A weakness of FLORA's simple removal strategy is the easy over-generalization of concepts, which leads to inferior classification performance.

Notations:

$ADES$	summary of all ADES-Sets
$PDES$	summary of all PDES-Sets
$NDES$	summary of all NDES-Sets
I	new training instance
AP_i	num. of I supporting ADI_i
PP_i, PN_i	num. of I supporting PDI_i
NN_i	num. of I supporting NDI_i

Algorithm:

function learn(I, $ADES_j$, $PDES_j$, $NDES$)
  $MATCH := false$

  for $i := 1$ to $|ADES_j|$ do
    if match(I, ADI_i) then
      $AP_i := AP_i + 1$
      $MATCH := true$

  if not $MATCH$ then
    $G :=$ generalize(I, $ADES_j$, $NDES_j$)

  if not $MATCH$ and not G then
    include(I, $AP_i = 1$, $ADES_j$)

  for $i := 1$ to $|PDES_j|$ do
    if match(I, PDI_i) then
      $PP_i := PP_i + 1$

  for $i := 1$ to $|NDES_j|$ do
    if match(I, NDI_i) then
      delete(NDI_i, $NDES_j$)
      include(NDI_i, $PN_i = NN_i$, $PP_i = 1$, $PDES_j$)

Figure 5 FLORA-MC function: *learn()*

To avoid over-generalization, the NDES-Set j has to be jointly processed. (4) Only if an existing DI cannot be generalized, a novel DI in $ADES_j$ is created. The supporting counter is initialized to $AP_i = 1$. (5) A check is performed if the instance matches a DI in the PDES-Set. If yes, the supporting counter is incremented similar to step (2) for the $ADES_j$. (6) As a last step, the algorithm checks if the instance is equal to an existing DI of NDES-Set j. If yes, the DI is moved into PDES-Set j, with $PN_i = NN_i$ and $PP_i = 1$.

It is important to note that the function *learn()* is also applied on all NDES-Sets except NDES-Set j per instance. The modified pseudo-code has to exchange the NDES and ADES-Sets as well as the corresponding DIs. Additionally, counter PP_i is replaced by PN_i.

Please note the special role of the PDES-Set. In PDES, DIs are not included or generalized directly, but only if they are moved from ADES or NDES. E.g., if a DI has been created in ADES-Set m, and a novel instance for category $m + 1$ matches this DI, then it is moved to the PDES-Set, because the DI now supports an instance not only for m, but also for an instance other than m. Within the PDES-Set the item has a supporting counter (PP_i) for category m as well as a counter (PN_i) for categories other than m. Using the function *forget()*, the DI can be moved back to the ADES or NDES set.

Function *forget()* (see Fig. 6) is called to remove DIs from the descriptor sets, the call being *forget*(I, $ADES_i$, $PDES_i$, $NDES$). (1) The supporting counter AP_i is decremented. If the DI is no longer supported by any training sample in the window ($AP_i = 0$), the DI is removed from the ADES-Set. (2) The counter for PDES-Set j is also decremented. However, as soon as zero is reached, the DI is moved back into the NDES-Set j, i.e, the DI in question is only further supported from instances other than category j.

Again, the function *forget()* is also executed on the NDES-Sets, the changes in the pseudo-code are similar to the ones for function *learn()*.

To support numerical attribute values in training instances, our FLORA extension introduces the concept of (numerical) intervals. Matching of numerical input values is performed constantly (i.e. for each new instance feeding the learning process). A key problem of supporting numerical values instead of nominal values is to determine up to which distance a new element is considered to match an existing value or fall into an existing interval. FLORA-MC checks for each set, if the intervals/values match or can be joined (generalized) with an existing interval/value. Joining an interval in $ADES_j$ requires both intervals plus/minus a smoothing factor to overlap each other, while at the same time not contradicting any other DI in the NDES-Set j. We introduce a smoothing factor to serve as a distance metric.

Algorithm:

function forget(I, $ADES_j$, $PDES_j$, $NDES$)
  for $i := 1$ to $|ADES_j|$ do
    if match(I, ADI_i) then
      $AP_i := AP_i - 1$
    if $AP_i = 0$ then
      delete(ADI_i, $ADES_j$)

  for $i := 1$ to $|PDES_j|$ do
    if match(I, PDI_i) then
      $PP_i := PP_i - 1$
    if $PP_i = 0$ then
      delete(PDI_i, $PDES_j$)
      include(PDI_i, $NN_i = PN_i$, $NDES_j$)

Figure 6 FLORA-MC function: *forget()*

The smoothing parameter $smooth \in \{0, 1\}$ is user defined per numerical attribute. The generalization of the numerical value x or of an interval is performed as follows:

(1) $smoothing(smooth, n) = \sqrt{\frac{\ln(smooth)}{-0.5^n}}$, with n being a constant representing the number of elements from all training instances falling within the interval and *smooth* being the threshold for the matching function. The higher the occupation of the interval, the lower the likelihood that distant values are part of this interval, the smaller the smoothing factor.

(2) y as value for a training instance is said to generalize a numerical value x within an DI if and only if $x - smoothing(smooth, n) \leq y \leq x + smoothing(smooth, n)$. We are able to apply the same rules for an interval, if we extend the upper and lower bound of the interval by $smoothing(smooth, n)$.

The aforementioned functions are applied to allow for the multiple classification in FLORA-MC. In particular, the DIs in the ADES-Set form the basis for classifications.

Based on the classification sets as well as the fundamental functions *learn()* and *forget()*, we are able to easily realize a multiple classification capability. Each classification instance entering the system is compared with all DIs in all ADES-Sets only, which can in fact lead to multiple hits.

In contrast to the FLORA-1–4, a classification might fail due to the numerical nature of the input values in FLORA-MC. The original algorithm family would classify an instance non-matching to any DI as *false*, which is not feasible in FLORA-MC. Here, a minimal-distance-based approach is chosen to obtain the DI, which is closest to the sample. The function for the distance is determined as follows. (1) If a nominal value of an instance is not equal to the corresponding nominal value of a DI, the distance is incremented by one. (2) If a numerical value of an instance is not equal to the corresponding numerical value of a DI, the normalized distance between both numerical values is added to the distance. If the DI contains a numerical interval for an attribute, FLORA-MC computes the distance between the numerical value of the classification instance and the upper or lower boundary of the interval.

3.3 Window management

The number of training instances that are used to build up the target concept is managed via FLORA-MC's window size, which thus acts as the main control parameter of the algorithm. The appropriate choice and adaptation of the window size is paramount to guarantee a high quality of the classifications, while remaining sensible to concept drift. If the window is chosen too small, the entire learning process is characterized by uncertainty, which results in an excessive amount of mis-classifications. In contrast, an oversized window decreases the algorithm performance and leads to a much slower convergence in presence of concept drift.

FLORA-MC could perform optimally, if the window size is reduced as soon as a concept drift occurs, thus allowing for a fast learning process for the novel concept space. We propose a window adjustment heuristic (WAH), to first detect concept drift and second adjust the window size accordingly (see Fig. 7). Since our algorithm is only exposed to partial real-world context information (in form of training instances), however, this detection of concept drift is challenging. This is particularly true due to the complexity of detection the concept drift for numerical instead of nominal input values. We embed our WAH into the learning process and make use of the characteristic operational parameters of the learner (accuracy *Acc* and Shannon-entropy S of the classifications). The behavior of the WAH is as follows.

- The Shannon-entropy is a measure of the uncertainty associated with a random information and can be used to act as a switch to avoid mis-detections of concept drift. Uncertain learning situations such as during the initialization of the system are suspected for $S > S_{\text{threshold}}$. Here the main goal is to stabilize prediction performance, but not to (mistakenly) assume concept drift. We propose to increment the window by one

Notation:

$\|W\|$	window size
S	Shannon-entropy (using previous classifications)
Acc	current accuracy (using previous classifications)
Acc_{pre}	previous accuracy
$\sigma_{Acc_{pre}}$	standard deviation of Acc_{Pre}

User-defined parameters:

$Acc_{threshold}$	accuracy threshold
$S_{threshold}$	Shannon-entropy threshold

Algorithm:

if $(S > S_{threshold})$
  $|W| = |W| + 1$
else if $(Acc < Acc_{threshold})$ and $(Acc < (Acc_{pre} - \sigma_{Acc_{pre}}))$
  $|W| = 0.8 * |W|$
else if $(Acc < Acc_{threshold})$
  $|W| = |W| + 1$
else
  $|W| = |W|$

Figure 7 Window Adjustment Heuristic (WAH)

instance allowing FLORA-MC to acquire an additional sample, thus improving the prediction quality (if we assume stable concepts) and mitigating the uncertain learning situation.

A concept drift can be suspected, if the accuracy is momentarily falling below a defined threshold ($Acc < Acc_{\text{threshold}}$) while, at the same time, being smaller than the previous accuracy minus its standard deviation ($Acc < (Acc_{\text{pre}} - \sigma_{Acc_{\text{Pre}}})$). The window size should be significantly reduced, e.g., by 20%. The resulting reduction in 20% of active training samples results in a preference towards novel concepts, thus increasing the convergence speed of the algorithm.

A slowly decreasing classification accuracy ($Acc < Acc_{\text{threshold}}$) should be countered by incrementing the window size. The increase in number of samples allows the learner to increase its accuracy again.

If none of the above applies, the learning behavior is considered to be fine, thus, no additional samples are necessary and the window size is kept constant. Also, for each arriving sample, old samples are discarded in a FIFO manner.

A couple of key differences exist between our window management heuristics and the mechanisms present in the original FLORA-2 model. First, we use the Shannon-entropy to yield more stable results directly after a context switch has been detected, while original FLORA does not so, which might lead to constantly fluctuating window sizes in such situations. Second, FLORA-MC keeps the windows size constant if the context is detected as stable; FLORA-2 here decrements the window size by one, which caused problems for the classification of numerical values in our experiments. Also, the WAH of FLORA-2 makes use of the ADES-Set, which is not possible for numerical values and the multi-classification case. Additionally, FLORA-2 uses a decrease in *Acc* to detect concept drifts. Using numerical values, however, *Acc* is constantly changing, which would unnecessarily trigger the FLORA-2 mechanism. FLORA-MC, in contrast, includes the standard deviation of the recent *Acc* values for the detection of concept drifts. In summary, we added various supplemental mechanisms to the original WAH heuristics of FLORA-2 to avoid misclassifications in presence of numerical values and multi-classifications.

3.4 Implementation

We implemented FLORA-MC as a WEKA module, which is available to the community as open-source (see [18]). Implementation details can be found in [19].

4 Evaluation

This section comprises the description of the evaluation of FLORA-MC. We first introduce the goals of our study and specify the features of the system under test. Subsequently, the parameters of interest are introduced and the experimental design is devised. By means of a comparative analysis, we study the performance and accuracy of FLORA-MC vs. state-of-the-art offline learning algorithms including Naive Bayes (NB), decision tree (DT), artificial neural network (NN) and support vector machine (SVM). For the latter four learners, we include implementations with and without a meta-window adjustment heuristic (meta-WAH), to be able to evaluate the handling of concept drift appropriately. We chose the well-known STAGGER-concept to generate the workload for our simulation study; to be able to measure the extended features of FLORA-MC, we extend the STAGGER-concept correspondingly. Finally, we present the obtained results and analyse and interpret the findings.

4.1 Goals and methodology

Our goal is to study the performance and accuracy of FLORA-MC, i.e., the model-building time, the precision of the model predictions, and the reaction time to concept drift. We compare FLORA-MC with a representative set of offline learners.

- Naive Bayes (NB), which is a simple, yet highly efficient incremental probabilistic learner.
- Decision tree (DT), which represents the class of hierarchical-tree algorithms. We use a binary tree with NB-classifier.
- Artificial neural network (NN), which is an algorithm modelling the learning process of the human brain.
- Support vector machine (SVM), which detects patterns by mapping the input vectors into a high dimensional space and constructing maximal separating hyperplanes.

Obviously the above offline learners are not able to natively handle concept drift. To level the playing ground and allow for a fair comparison with our incremental online learner FLORA-MC, we augment the offline learners with a window heuristic (meta-WAH) similar to our WAH to detect concept drift. The interpretation of the results discusses the implications and efficacy of this approach.

We build on the STAGGER-methodology introduced by Schlimmer and Granger [20] for the evaluation of the algorithms, because it is commonly used to relate the performance between online and offline learners in literature (see e.g. [21]). Originally, the STAGGER-concept uses a simple domain of three nominal attributes, which are defined as follows.

- object *size* ∈ {(*s*)*mall*, (*m*)*edium*, (*l*)*arge*},
- object *color* ∈ {(*g*)*reen*, (*b*)*lue*, (*r*)*ed*}, and
- object *shape* ∈ {(*tr*)*iangular*, (*ci*)*rcular*, (*sq*)*uare*}.

The learning cycle consist of three distinct phases in the following order [3].

- Concept 1:
  $size = small \wedge color = red := positive$
- Concept 2:
  $color = green \vee shape = circular := positive$
- Concept 3:
  $size = (medium \vee large) := positive$

We modify the STAGGER-concept to suit our case. We keep the object *size* and *color*, but replace the object *shape* with two numerical attributes: *diameter* and *temperature*. We retain the three concepts (or phases), however, now representing a multi-variate learning scenario instead of a binary one, the class values being {*nice*, *fair*, *bad*}. The learning phases are defined as follows.

- Concept/phase 1:
  $size = s \wedge color = r \wedge dia. = [0; 2] \wedge temp. = [25; 45] := nice$
  $size = m \wedge color = g \wedge dia. = [2; 3] \wedge temp. = [0; 25] := fair$
  $size = l \wedge color = b \wedge dia. = ([0; 2] \vee [3; 5]) \wedge temp. = [0; 45] := bad$

- Concept/phase 2:
  $size = m \wedge color = G \wedge dia. = [2; 3] \wedge temp. = [0; 25] := nice$
  $size = l \wedge color = B \wedge dia. = ([0; 2] \vee [3; 5]) \wedge temp. = [0; 45] := fair$
  $size = s \wedge color = R \wedge dia. = [0; 2] \wedge temp. = [25; 45] := bad$

- Concept/phase 3:
  $size = m \wedge color = G \wedge dia. = [2; 3] \wedge temp. = [0; 25] := nice$
  $size = l \wedge color = B \wedge dia. = ([0; 2] \vee [3; 5]) \wedge temp. = [0; 45] := fair$
  $size = (s \vee m) \wedge color = (g \vee r) \wedge dia. = [2; 3] \wedge temp. = [25; 45] := bad$

Figure 8 visualizes our modified STAGGER-methodology. The boxes filled in grey indicate the class-value for the concept *nice*, vertical lines indicate the valuc *fair*, and horizontal lines the value *bad*. It is obvious that the transition from phase 1 to phase 2 changes all concepts (concept drift in 3/3 concepts). The transition between phase 2 and 3 only varies the concept for *bad*. Thus, the modified STAGGER-concept allows the evaluation of the learners for strong and weak concept drift.

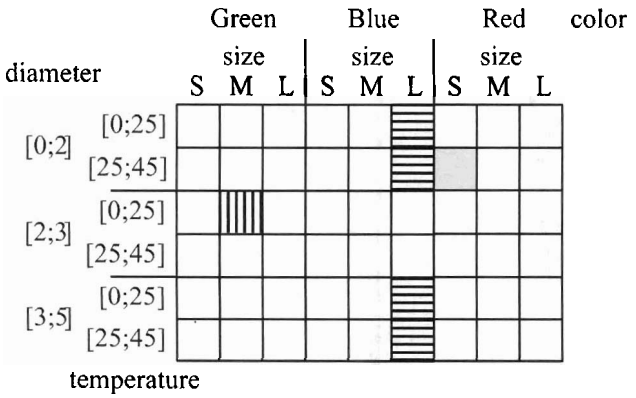
4.2 Experimental design and workload

We perform ten replications per simulation. The ten training sets are randomly created using the modified STAGGER-concept; however, to be able to compare the results, we re-use the same ten random STAGGER data sets for all algorithms. Each simulation run consists of the training of each individual training-instance (denoted as I, $I \in \{1 \ldots 150\}$), i.e., we perform 150 trainings per run, and subsequently 100 test-classifications per training to obtain the classification accuracy. The runtime results have been obtained using WEKA 3.4.7 [4] on a Pentium IV, 3.0 GHz PC with 1GByte main memory.

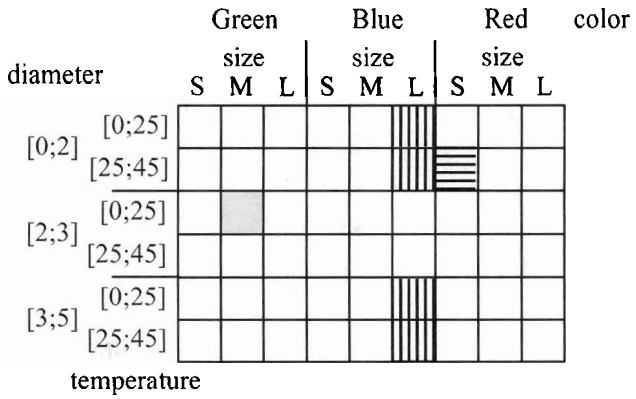
Our FLORA-MC algorithm can be tuned to fit specific scenarios. The influence of parameters such as window size, the limits of the Shannon-entropy to detect concept drift using our WAH, and the reduction rate of the window size if a concept drift is detected has been evaluated; please refer to [19] for an extensive set of experimental results covering all the individual building blocks of the FLORA-MC system. For the remainder of the experiments, we have chosen conservative settings to act as a baseline for FLORA-MC. The limit of the Shannon-entropy is set to 0.7, $Acc = 0.7$, the windows size is reduced by 20%, if WAH detects a concept drift. A second, more aggressive parameterization, reduces the windows size by 100%, i.e., discards all training samples if a concept drift is detected. If WAH does not work properly, the latter parameterization might lead to unstable classifications, because entire (valid) concepts might be dropped; however, this behavior has not been observed during the performed experiments. Further optimization of these parameters is possible and has been extensively discussed in [19] for the employed training set.

4.3 Results—comparative performance analysis

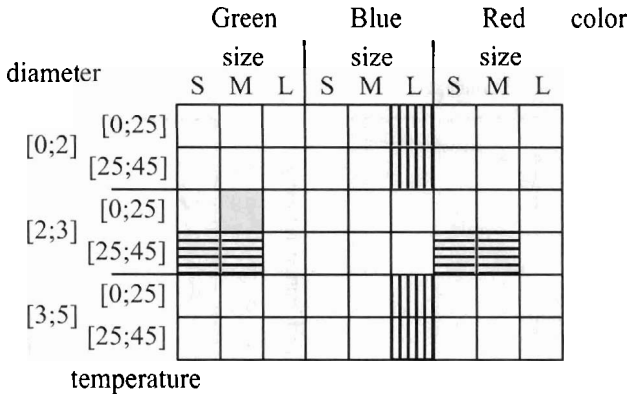
The obtained results for the classification accuracy of the tested algorithms are shown in Fig. 9. In particular, we analyse the performance of FLORA-MC vs. the plain offline learners and vs. offline learners that are extended with our meta-WAH (20% window reduction). Table 1 shows the corresponding results for the runtime-characteristics of the algorithms for model building and classification. We also study the performance of FLORA-MC vs. offline learners, if both use a more aggressive WAH/meta-WAH, which discards 100% of training samples as soon as a concept drift is detected (see Fig. 10 and Table 2). While the latter approach allows for a much faster convergence of the algorithms, it presents the danger of mis-detections of concept drifts; thus,



(a) Modified STAGGER-concept phase 1.



(b) Modified STAGGER-concept phase 2.



(c) Modified STAGGER-concept phase 3.

Figure 8 Visualization of the three phases of the modified STAGGER-concept, which serves as the workload for our performance evaluation

the implications of the aggressive WAH/meta-WAH have to be carefully considered, depending on the application.

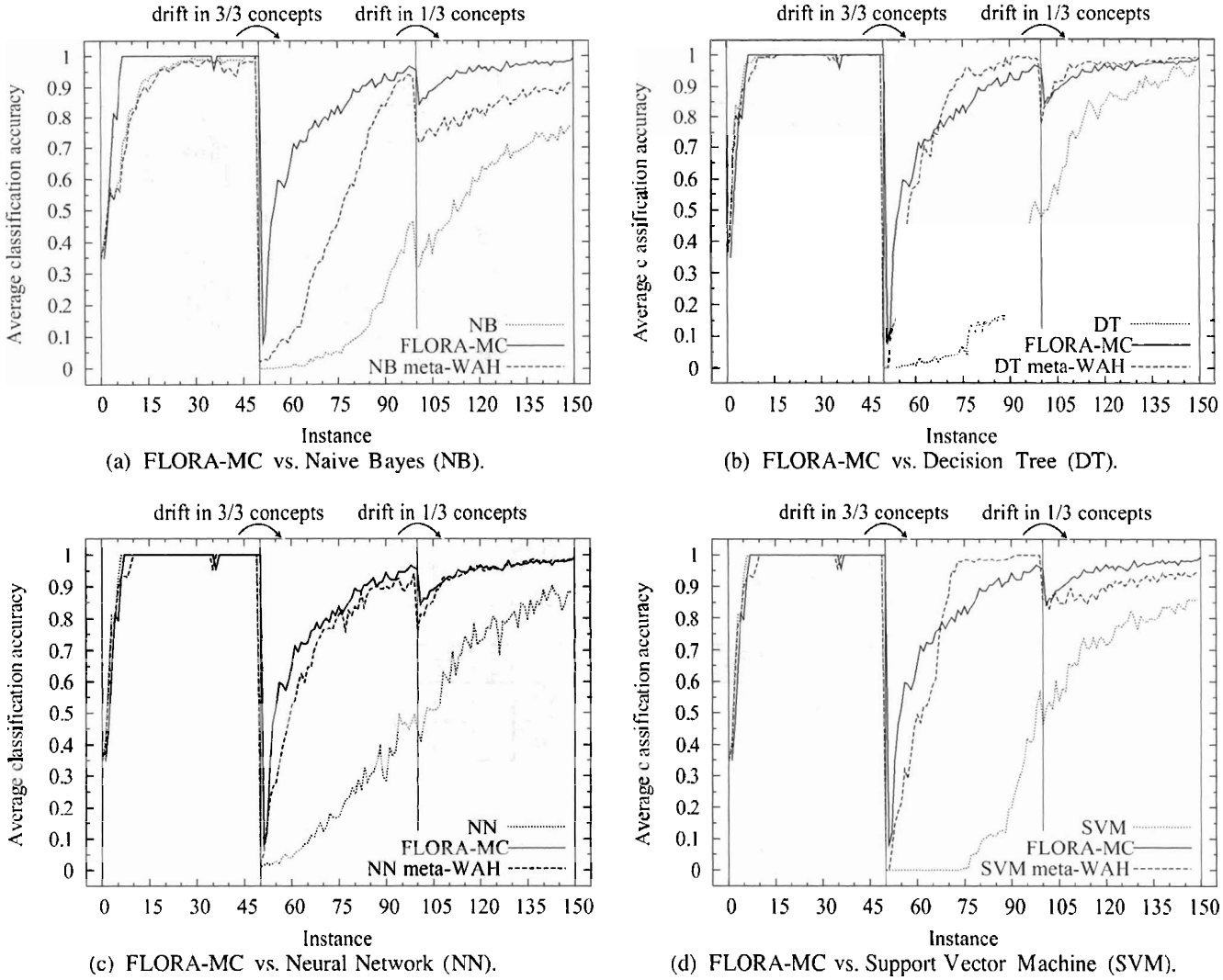
All presented results represent the average values over ten replications per simulation. For the ease of following the discussion, we highlight the relative runtime-performance of the algorithms in Tables 1 and 2 using arrows: ↓ indicates that FLORA-MC performs worse compared with the respective offline learner, while ↑ denotes that FLORA-MC outperforms the candidate offline algorithm. Please note that some of the results for the algorithm-runtime show up to an order of three magnitudes difference between the algorithms; this behavior is obvious though, since we compare our online learner (FLORA-MC) that incrementally adjusts its model with offline learners (NB, DT, NN, SVM) that have to fully rebuild the model 150 times during our test.

4.3.1 FLORA-MC vs. Naive Bayes (NB)

Figures 9a and 10a clearly show that our FLORA-MC approach outperforms NB with respect to classification accuracy for all three phases of the modified STAGGER-concept. During the first phase ($I = 1 \ldots 50$) FLORA-MC quickly adapts to the concepts, because all sets (ADES, PDES, NDES) are initially empty. For the second phase ($I = 51 \ldots 100$) all concepts are exchanged, thus modeling an abrupt concept drift. FLORA-MC's behavior—as shown in Fig. 9a—is characterized by a slightly damped learning process; WAH detects the concept drift and reduces the window size, however, still the outdated concepts need to be vacated.

If we compare the results from Figs. 9a and 10a, it is clearly visible that a moderate reduction of the window (i.e. of 20%, see Fig. 9a) leads to a slow adaptation, while the increase in learning accuracy is much steeper for the aggressive WAH (reduction of 100%, see Fig. 10a). The drift of one concept during the transition into the third phase ($I = 101 \ldots 150$) does not trigger our WAH, nevertheless FLORA-MC shows an excellent performance: the learner moves the vacated concept from an ADES set to a PDES set and creates a novel ADES set describing the new concept and is, thus, able to follow the concept drift swiftly.

Compared with NB, FLORA-MC is characterized by faster convergence times as well as overall higher accuracy of the classification. Especially the ability of FLORA-MC to construct a working set with very few samples



(a) FLORA-MC vs. Naive Bayes (NB).
(b) FLORA-MC vs. Decision Tree (DT).
(c) FLORA-MC vs. Neural Network (NN).
(d) FLORA-MC vs. Support Vector Machine (SVM).

Figure 9 Classification accuracy of FLORA-MC vs. offline learning algorithms NB, DT, NN, and SVM

outperforms NB, which needs a larger number of samples due to its conditional-probability-based nature. NB without meta-WAH struggles to handle the concept drifts of phase 2 appropriately, since the classifier still uses all (now outdated) samples. Only at around $I = 100$, NB again reaches an accuracy of 50%, because the set of samples for both concepts is equal. The same behavior can be observed during phase 3. Our proposed meta-WAH significantly improves NB for all tests. The window reduction of 20% (see Fig. 9a, $I = 51 \ldots 100$) is not able to remove enough conflicting samples and results in a slower adaptation to the novel concepts compared with FLORA-MC. The results in phase 3 are closer to FLORA-MC, because the moderate change of only one concept can be covered by NB's strategy much faster. Table 1 and Table 2 show that NB has excellent runtime-behavior, though. Due to the simple incremental probabilistic nature, the model-building times are very short and faster if compared with FLORA-MC; the classification times are excellent as expected for the simple NB learner.

4.3.2 FLORA-MC vs. Decision Tree (DT)

The original DT algorithm (which bases on NB to determine its branching decision) shows very fast convergence for phase 1 ($I = 1 \ldots 50$). In contrast to NB, the construction of the decision tree allows for immediate classifications of high certainty. For phase 2 and 3, pure DT reaches only mediocre classification accuracy as shown in Fig. 9b, because it still relies on the tree constructed during phase 1, thus mirroring the shortcoming of NB. In combination with our meta-WAH, the accuracy improves significantly and is roughly on par with FLORA-MC (see Fig. 9b and 10b) due to the construction of an entirely novel tree-structure.

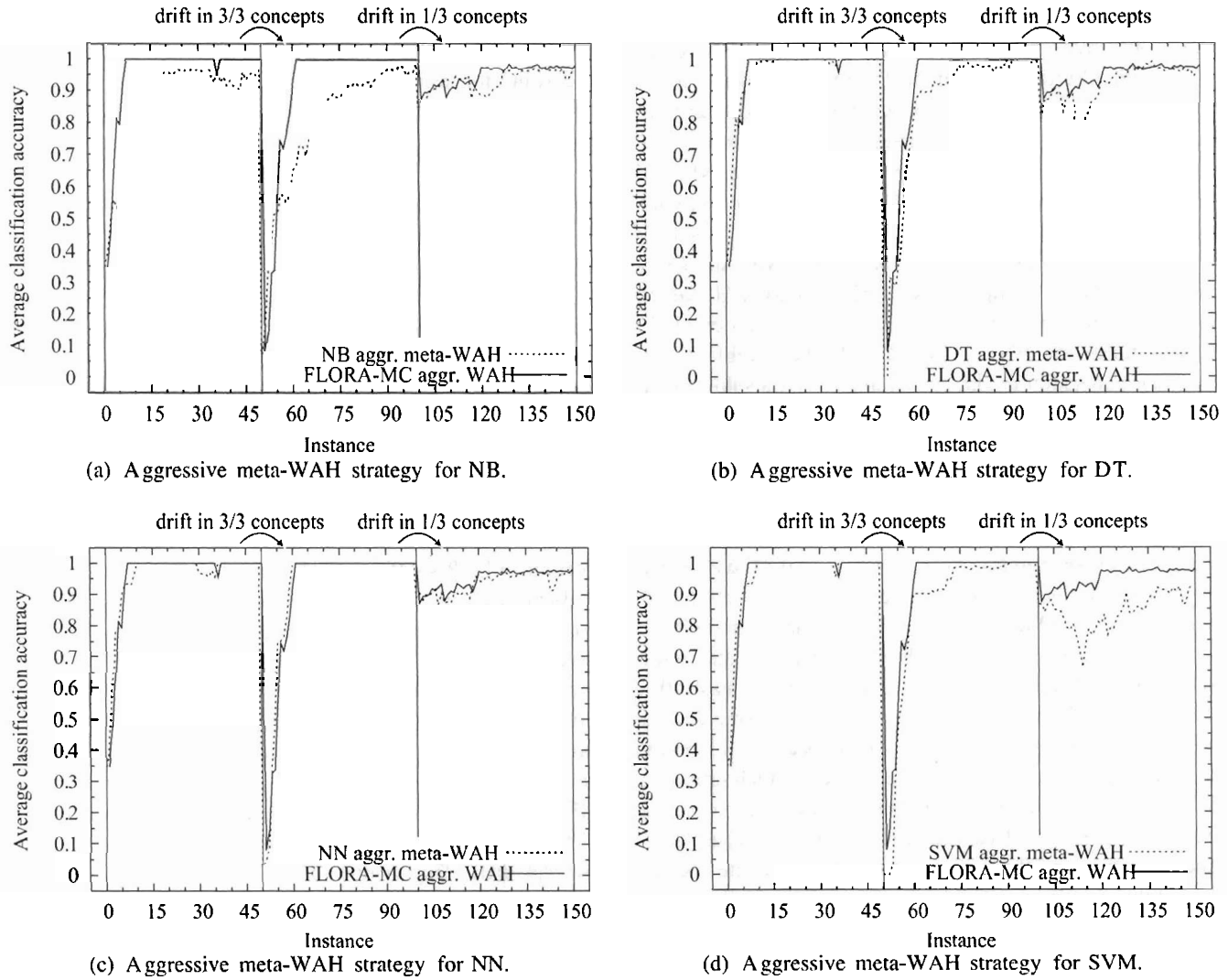
Table 1 Results for runtime of model-building process and classification process (standard algorithms)

Parameter	FLORA-MC	Naive Bayes (NB)		Decision tree (DT)		Neural network (NN)		Support vector machine (SVM)	
	Time [ms]	Time [ms]	↕	Time [ms]	↕	Time [ms]	↕	Time [ms]	↕
Avg. model building	1.048	0.251	4.17 ↓	35.64	0.03 ↑	304	0.003 ↑	721	0.001 ↑
Min. model building	0.513	0.083	6.18 ↓	13.25	0.04 ↑	251	0.002 ↑	694	0.001 ↑
Max. model building	1.666	0.940	1.77 ↓	62.20	0.03 ↑	365	0.004 ↑	744	0.002 ↑
Avg. classification	0.085	0.013	6.53 ↓	0.009	9.44 ↓	0.008	10.63 ↓	0.006	14.17 ↓
Min. classification	0.068	0.009	7.55 ↓	0.003	22.70 ↓	0.005	13.60 ↓	0.004	17.00 ↓
Max. classification	0.101	0.023	4.39 ↓	0.021	4.81 ↓	0.011	9.18 ↓	0.011	9.18 ↓

We denote the relative gain/penalty both as factor and visually using arrows: ↓ indicates that FLORA-MC performs worse compared with the respective learner, while ↑ denotes that FLORA-MC outperforms the respective algorithm

While DT is able to converge even faster than FLORA-MC for the second phase ($I = 51 \ldots 100$) with standard window settings, it is more susceptible to mis-classifications for the aggressive meta-WAH setting (see Fig. 10b).

The runtime-behavior for the model-building process is inferior to the performance of FLORA-MC as shown in Tables 1 and 2; in contrast, the classification process is handled better by DT. The edge in performance of FLORA-



(a) Aggressive meta-WAH strategy for NB.

(b) Aggressive meta-WAH strategy for DT.

(c) Aggressive meta-WAH strategy for NN.

(d) Aggressive meta-WAH strategy for SVM.

Figure 10 Classification accuracy of FLORA-MC vs. offline learning algorithms NB, DT, NN, and SVM. Aggressive meta-WAH strategy

Table 2 Results for runtime of model-building process and classification process (aggressive WAH/meta-WAH)

Parameter	FLORA-MC	Naive Bayes (NB)		Decision tree (DT)		Neural network (NN)		Support vector machine (SVM)	
	Time [ms]	Time [ms]	↕	Time [ms]	↕	Time [ms]	↕	Time [ms]	↕
Avg. model building	0.894	0.126	7.10 ↓	11.55	0.07 ↑	158	0.006 ↑	664	0.001 ↑
Min. model building	0.520	0.010	52.0 ↓	4.99	0.10 ↑	118	0.004 ↑	639	0.001 ↑
Max. model building	1.360	0.520	2.26 ↓	25.09	0.05 ↑	361	0.004 ↑	695	0.002 ↑
Avg. classification	0.067	0.012	5.58 ↓	0.011	6.09 ↓	0.009	7.444 ↓	0.004	16.75 ↓
Min. classification	0.056	0.009	6.22 ↓	0.005	11.20 ↓	0.006	9.333 ↓	0.001	56.00 ↓
Max. classification	0.075	0.018	4.16 ↓	0.017	4.41 ↓	0.014	5.357 ↓	0.010	7.50 ↓

We denote the relative gain/penalty both as factor and visually using arrows: ↓ indicates that FLORA-MC performs worse compared with the respective learner, while ↑ denotes that FLORA-MC outperforms the respective algorithm

MC for the expensive model-building process can be considered the most important distinguishing factor between FLORA-MC and DT; thus, the choice between both algorithms should be based on the corresponding application requirements.

4.3.3 FLORA-MC vs. Neural Network (NN)

Figure 9c shows the classification accuracy of NN: compared with FLORA-MC, again, the original learner is inferior, while the meta-WAH brings NN nearly on par with FLORA-MC. This behavior meets our expectations, because NN is subject to the same limitations as NB and DT. For the aggressive meta-WAH, the quality of classifications of FLORA-MC and NN is equally good as shown in Fig. 10c. The classification runtime-behavior of NN is moderately better compared with FLORA-MC. However, the model-building process is slower in the order of 2–3 magnitudes for NN vs. FLORA-MC (see Table 1 and Table 2). Especially the latter fact makes NN a sub-optimal choice for ambient computing applications.

4.3.4 FLORA-MC vs. Support Vector Machines (SVM)

For static concepts SVM offers an excellent classification accuracy as shown in Fig. 9d for $I = 1 \ldots 50$. However, the original SVM algorithm shows the worst classification performance of all offline learners for abrupt change in multiple concepts (see $I = 51 \ldots 100$ in Fig. 9d), which is due to the construction of the hyperplanes that requires at least 50% of the samples from the new concepts; starting from $I = 75$ this requirement is met and the accuracy starts to improve. The standard meta-WAH is able to very much improve the performance of the SVM, because the sample set is reduced and the SVM can construct maximal hyperplanes much earlier. E.g., the switch from the first to the second phase (abrupt change in all concepts that trigger the meta-WAH) is handled superior compared with all other learners. However, the switch from phase 2 to 3 (abrupt change in one of three concepts, meta-WAH is not triggered) is not handled that well. Like DT, SVM is also susceptible to the more aggressive meta-WAH (see Fig. 10d), showing worse classification accuracy compared with FLORA-MC.

For SVM we observe the highest difference in runtime-behavior between classification and model building. The model-building process is up to three orders of magnitude slower compared with FLORA-MC, but the classification time is extremely fast, outperforming all other learners. Like NN, the performance penalty of SVM excludes this algorithm for tasks that demand fast model building. Additionally, the inconsistency in the prediction quality of SVM can be considered as a drawback of this algorithm.

4.4 Summary

The obtained results are very promising, especially if we keep the intended application scenario in mind. Compared with the tested offline learners, FLORA-MC is able to reduce execution time for the model-building process in average by factors around 13–34 (DT), 175–290 (NN), 690–740 (SVM); NB outperforms FLORA-MC by a factor of 4–7, though. In contrast the classification time of FLORA-MC shows a 5–17-fold increase compared with the offline learners.

From the presented results, we can argue that our algorithm FLORA-MC is well suited for applications in ambient computing. The model-building process that is commonly considered time-critical is reduced significantly (except compared with NB), while the increase in classification time can be tolerated, because it is still very low (in average below 0.1ms per classification with FLORA-MC). The quality of the predictions can be judged as very good for FLORA-MC. Without meta-WAH, none of the offline learners is able to compete with the accuracy of FLORA-MC's classifications, given the workload includes concept drift. This is due to the functioning of FLORA-MC, which is able to adjust its ADES or NDES sets as soon as one conflicting sample is recognized. Extending the offline learners with our meta-WAH brings the accuracy of classification of DT, NN, and SVM on par with FLORA-

MC; NB with meta-WAH still falls significantly behind FLORA-MC showing the worst accuracy of all algorithms, thus rendering the excellent runtime-performance pointless. In conclusion FLORA-MC presents a highly efficient and accurate machine learning algorithm that has been tailored to fit the requirements of ambient computing.

5 Conclusion

The challenging requirements of reasoning mechanisms for ambient computing include ease-of-use, self-learning abilities, and very high prediction accuracy, while the employed techniques need to be able to operate in realtime. To fulfil these requirements, we propose to rely on machine learning algorithms to evaluate and determine the context of users, devices, and objects. We presented the FLORA-MC approach, which is one of the first approaches to introduce realtime model-building capabilities to aid the process of context determination. Based on the online learning approach FLORA, we augmented the crucial features for the intended application area: (1) multiple classification capabilities and (2) support for numerical input values. At the same time, our window adjustment heuristic allows for superior handling of concept drift. We gave a detailed description of the design, implementation, and working of FLORA-MC. Based on our proof-of-concept implementation [19], we validated the functioning of our algorithm and performed an extensive simulation study using representative artificial workload. Our results are very promising with respect to the performance as well as accuracy of our algorithm. FLORA-MC is able to outperform traditional offline learner by orders of magnitude with respect to model building time, which we consider crucial given the possibly very high number of samples to be instantaneously processed in truly ambient computing. At the same time, our scheme is able to follow concept drift with only minimal reaction time while maintaining an excellent quality of prediction, which is on par or better compared with state-of-the-art offline learners. We consider FLORA-MC to open up various avenues of research. In current work, we are collecting input data in real world scenarios to be able to assess the performance of FLORA-MC under realistic settings; promising next steps are to fine-tune our algorithm to reach optimal performance in such realistic scenarios.

Acknowledgements The authors would like to thank the guest editors and the anonymous reviewers for providing valuable comments that aided to improve the quality of the manuscript.

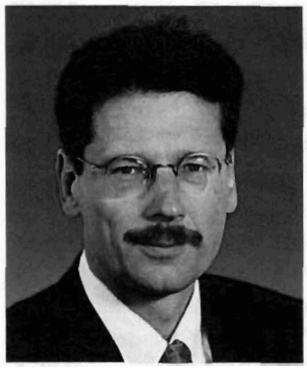
References

1. Tsymbal A (2004) The problem of concept drift: definitions and related work. Department of Computer Science, Trinity College Dublin, Ireland, Technical Report TCD-CS-2004-15. https://www.cs.tcd.ie/publications/tech-reports/reports.04/TCD-CS-2004-15.pdf, April
2. Widmer G (1997) Tracking context changes through meta-learning. Mach Learn 27(3):259–286, June
3. Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23(1):69–101, April
4. The University of Waikato (2006) University of Waikato Weka Data Mining Software. http://www.cs.waikato.ac.nz/~ml/weka/
5. Görtz M (2005) Effiziente Echtzeit-Kommunikationsdienste durch Einbeziehung von Kontexten. Ph D dissertation, Technische Universität Darmstadt
6. Schmitt J, Hollick M, Steinmetz R (2007) Der Assistent im Hintergrund: Adaptives Kommunikationsmanagement durch Lernen vom Nutzer. PIK - Praxis der Informationsverarbeitung und Kommunikation 30(2):100–105, April–June
7. Schilit WN (1995) A system architecture for context-aware mobile computing. Ph D dissertation, Columbia University
8. Dey A, Abowd G, Salber D (1999) A context-based infrastructure for smart environments. In: Proceedings of the 1st international workshop on managing interactions in smart environments (MANSE '99), pp 114–128
9. Dey A (2000) Providing architectural support for building context-aware applications. Ph.D. dissertation, Georgia Institute of Technology
10. Schmidt A, Aidoo KA, Takaluomai A, Tuomelai U, Laerhoven KV, de Velde WV (1999) Advanced interaction in context. In: Lecture notes in computer science 1707
11. Schmidt A (2002) Ubiquitous computing—computing in context. PhD dissertation, Lancaster University
12. Amigo (2008) Amigo—Ambient intelligence for the networked home environment. European Project, Information Society Technologies. http://www.hitech-projects.com/euprojects/amigo/
13. Ambient Networks (2006) Ambient Networks European Project, Information Society Technologies. http://www.ambient-networks.org/
14. Littlestone N (1988) Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm. Mach Learn 2(4):285–318, April
15. Maloof MA, Michalski RS (2000) Selecting examples for partial memory learning. Mach Learn 41(1):27–52
16. Elio R, Watanabe L (1991) An incremental deductive strategy for controlling constructive induction in learning from examples. Mach Learn 7(1):7–44, July
17. Iba W, Woogulis J, Langley P (1988) Trading simplicity and coverage in incremental concept learning. In: Proceedings of the fifth international conference on machine learning. Morgan Kaufmann, San Francisco, pp 73–79
18. Flora-MC - Implementation as WEKA Module (2007) http://www.kom.tu-darmstadt.de/en/downloads/software/flora-mc/
19. Roos C (2006) Adaptation and extension of FLORA for application within a self-learning communication service. Student Thesis, Technische Universität Darmstadt
20. Schlimmer JC, Granger RH (1986) Beyond incremental processing: tracking concept drift. In: Proceedings of fifth national conference on artificial intelligence—AAAI, pp 502–507
21. Kolter JZ, Maloof MA (2003) Dynamic weighted majority: a new ensemble method for tracking concept drift. In: Proceedings of third IEEE international conference on data mining (ICDM 2003), Washington, DC, USA, pp 123–130, November

Johannes Schmitt is a member of the research group Ubiquitous Communications at the Multimedia Communications Lab (KOM) of Technische Universität Darmstadt, Germany. He graduated in Computer Science at Technische Universität Darmstadt in 2004. His research interests are issues of combining network services, sensor-systems, methods for context evaluation and learning algorithms in order to obtain smart communication services and to improve call management in telephony systems.

Christoph Roos graduated from Technische Universität Darmstadt in 2007, where he studied Business Administration, Economics and Law in the area of Computer Science. Nowadays he works as software engineer for business IT solutions at the Campana & Schott Group in Frankfurt am Main, Germany.

Matthias Hollick is the head of the research groups Mobile Networking and Ubiquitous Communications at the Multimedia Communications Lab (KOM) of Technische Universität Darmstadt, Germany, from which he graduated in Electrical Engineering and Information Technology. His research interests are in the area of dependability, security, and quality of service provisioning in heterogeneous communication networks. Dr.-Ing. Hollick is a recipient of the award of the Adolf-Messer Foundation; in 2007 he was a research scholar at the University of Illinois at Urbana-Champaign.



Ralf Steinmetz is the head of the Multimedia Communications Lab (KOM) at Technische Universität Darmstadt. Prof. Dr.-Ing. Steinmetz is Governor of the ICCC and Fellow of the ACM and the IEEE; he received numerous research awards and contributed to more than 400 refereed publications. He is member of the board and advisor of various academic, industrial as well as governmental institutions. His research interest are in the field of communications networks as well as seamless multimedia communications.

5 Simulation

5.1 Simulation model

Round based simulation of P2P network

- each round peers de ermine their behaviour anew
- whether to start qu ries looking for new content
- or just remain idle, eacting on incoming queries with QueryHit messages if possible

5.2 Node states

- Peers' behaviour abstracted as finite state automaton
  - each round, peers behave according to their current state
  - determine state for next round
- main states are self-termined
  - *Idle, SendQuery*
  - transistion controlled by *transition properties*
- triggered states
  - *processQuery, sendQueryHit, processQueryHit*
  - transition triggered by events

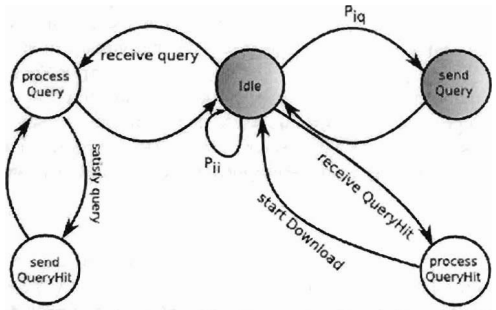


Figure 5.1: Node states

5.3 Simulation components

5.3.1 Random graph creation

6 Evaluation

In this chapter, we analyse the performance our similarity-based overlay system, by the use of the simulation we implemented and described in 5. First we will compare our system to a p2p system using a conventional random-based overlay network. To keep the first comparison simple, our system only uses *contact management* for the generation of the similarity based overlay (see 4.2.3). Later we will include *direct downloads* from neighbouring peers, *probabilistic flooding* and *adaptive TTL* mechanisms in our simulation. Looking at different aspects, we point out the benefits our system brings along. After this first analysis, we go into depth, presenting a detailed perfomance evaluation by running our simulation with different parameters and pointing out their impacts on network performance as a whole.

To start off, we first define a reasonable scenario for a p2p network to simulate, which we will use for our performance tests:

- Network parameters
  - 1000 peers, with 100 files cache size
  - caching policies
    - *LRU* for the random-based overlay network
    - both *LRU* and *FIFO*, interchangeably for performance comparison of the similarity-based overlay network
  - each peer connected by 8 edges with other peers
  - 5000 files, 20 interest categories, with uniform random distribution
  - 20 autonomous systems
  - TTL of 5 hops
- Simulation parameters
  - 3000 simulation rounds, i.e. 300 downloads per peer
  - Mandelbrot-Zipf distribution for file popularity, with 50 ranks, $\alpha = 0.6$, $q = 40$.
    - $ranks = 50$, $\alpha = 0.6$, $q = 40$

The Mandelbrot-Zipf parameters were chosen as appropriate after studying [SH06] (see related work 3.3.2).

6.1 Similarity-Based vs. Random Overlay Network

In this starting section of the evaluation chapter, we show that our similarity-based overlay performs significantly better than a conventional static random overlay network. We examine the following aspects:

Average distance between searching and providing peers

- Over all downloads, we derive ratios of how many requested files are found in 1, 2, 3, etc. hop distance. This is a measure of how far a query is to be forwarded. This is a good measure to show the complexity of content location.

Query forwards

- In our simulations, we counted the forwards caused by the flooding process. Since each peer has a certain number (this simulation: 8) of neighbouring nodes, each peer participating in the floowing process generates this number of additional forwards of a query. Therefore this number is a direct correlation to the number of nodes getting involved in the search process.

Similarity measures

- The more files the peers in the network share over the time, the more other peers with similar interest each peer gets to know. This lets the peers arrange in clusters, expressing a shared interest. This gives the peers the possibility of providng files, they already received, for the peers in they clusters. As time goes by, the network reaches a steady state.
  We depict this by comparing the download behaviour of adjacent nodes. Using the *ASim* formula presented in 3.8, we derive values on how similar the files, neighbouring peers download, are. More detailed, we compute the following values:
  - cache similarity
  - similarity of complete download history
  - similarity of download history over time interval (100 simulation rounds)
  - similarity in respect to the peer's interest category

  We complement these statistics by a diagram showing the development of the network's clustering coefficient.

While some of the graphs show total averages, most others depict a certain development over the course of our simulation. Usually, in these cases, after a starting phase, the graphs level off at a certain value. This is because at the beginning, the peer caches are empty. Once they are full, the caching policies start to work and thus a steady state gets reached.

6.1.1 Complexity of Content Location

In figure 6.1 we show the fractions of files that were found in a certain distance. In the network using the random-based overlay, more than 50 % of all files are found within three or more hops distance. This is caused by the fact that with one or two hops distance, a query does not reach enough peers to have a good chance of receiving hits. At a distance of three hops, enough peers are covered by the flooding process, to have a good chance of finding a node with the requested file (that is, in our setting, up to $8^1 + 8^2 + 8^3 = 584$ peers).

On the other hand, in the similarity-based overlay, in more than 65 % of all cases, requested files are found at adjacent peers. This provides possibilities for significant reduction in search complexity. Using the *direct download* mechanism (see 4.2.5), in these cases, no queries need to be send.

We also see that the *Least Recently Used* caching policy performs slighty better than a plain *First In First Out*. The explanation for this is quite simple. *FIFO* always deletes the oldest files in cache first, no matter of how often they were requested by other peers. However, *LRU* makes sure that often requested files are kept in cache, rather than files that are once downloaded and never provided for other peers.

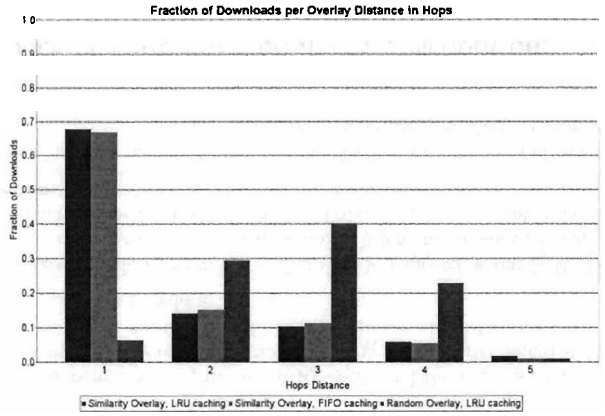


Figure 6.1: Average distance between source and downloading peers in the overlay

We know, that in the flooding process, a huge number of nodes is involved. If an intermediate node is able to provide a requested file, this node has no need for further forwarding the query and thus stops the flooding on its own behalf. Of course, in other parts of the network, flooding continues as usual, until the maximum hop count is reached. This is

caused by the earlier content location (in terms of hop distance to the searching peer) in the semantic overlay network. Figure 6.2 illustrates how costly flooding a single query is, by measuring the average number of forwards generated per query, at regular time intervals. For an item, that eventually is found at only a few peers, nearly thousand forwards of the original query message are sent. After 1000 simulation rounds, the peer chaches reach to their maximum capacity and the caching policies start to work. At this point, the number of forwards settle down at a certain level. As explained above, every eight forwards correspond to one peer involved in flooding a query, the reduced number of generated forwards in the simulations using the similarity overlay, relate to the peers discovered earlier due to the interest-based locality. Although the benefit turns out to be rather small, when using *direct downloads*, the savings will be much more signigicant, as we will see later. Lastly, a slight better performance of *LRU* caching can be seen again.

6.1.2 Comparing Neighbouring Nodes

Figures 6.3 and 6.4 depict the developement of the peers' neighbourhood. In the first plot, adjacent peers are compared in respect to the files in their caches. While in the random-based overlay network, neighbouring peers have little or no relationship, in the similarity-based overlay similarity values of about 25 % achieved.

Comparing the overall download behaviour, a steady rising can be seen, since in all cases, peers have at least a few common downloaded files. Of course this is far greater in the

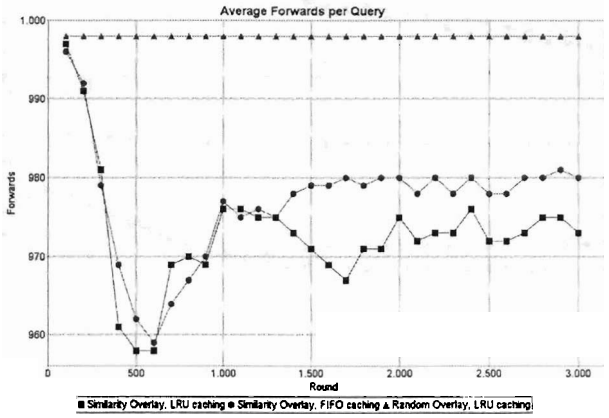


Figure 6.2: Average forwards per query

semantic overlay, since adjacent peers have common interests. The difference between *LRU* and *FIFO* is caused by the policy after which files are chosen for deletion from cache. IN *LRU* the most popular files stay much longer in cache or even get never deleted und therefore form a common basis for nodes with same interests. Since the download history grows with no upper bound, the related interests cause the similarity values to increase steadily.

In the third plot, the download behaviour over a time interval is compared. The difference between the graphs representing the similarity-based and the random-based overlay network, is caused be the same reasons as before. The higher similarity values at the start are caused by the peers first downloading the high-popularity files, which are fewer in number than the files with lower popularity.

The category-based similarity values show that after levelling off, about 70 % of each peer's neighbours, in the similarity-based overlay network, belong to the same category if interest, while in the random-based network the value of about 0.5 % is a result from the random overlay creation.

The last plot illustrates the development of the clustering coefficient and thus shows how nodes arrange in clusters with peers with similar interests, providing files for each other.

6.2 Impact of Different Popularity Distributions

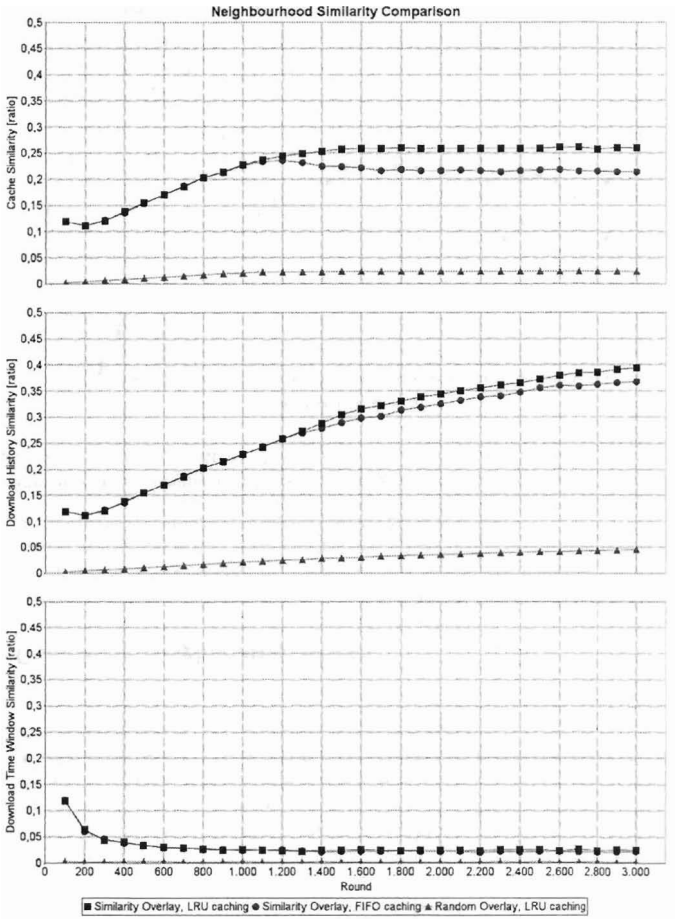


Figure 6.3: Similarity of adjacent peers

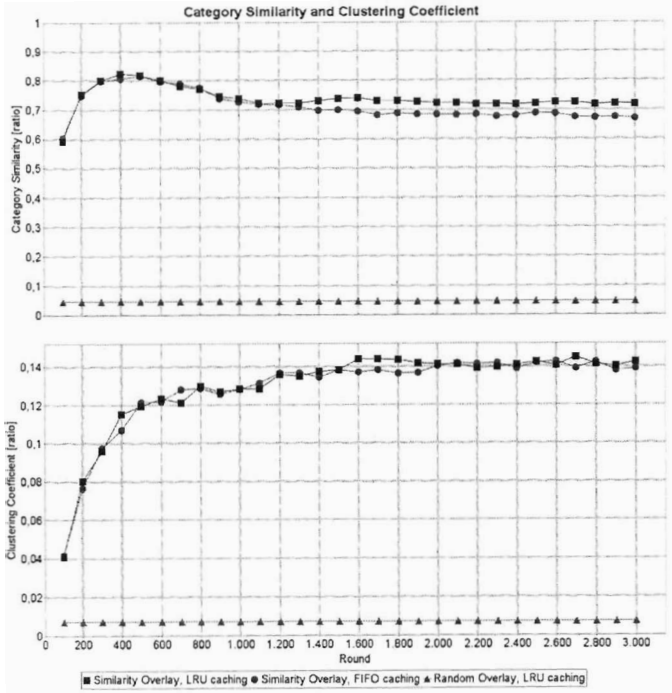


Figure 6.4: Similarity of adjacent peers

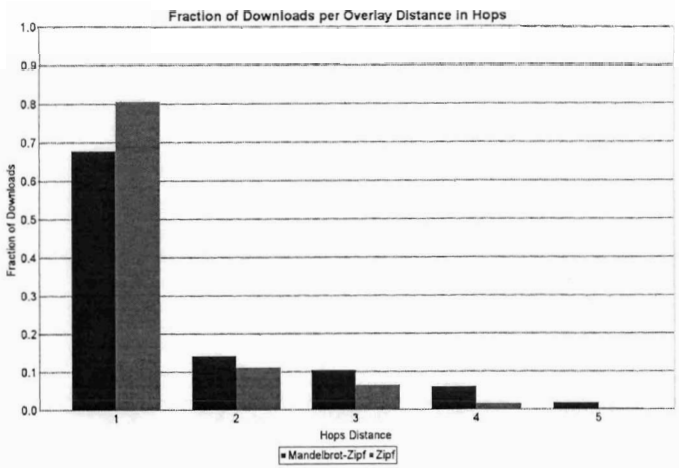


Figure 6.5: Mandelbrot-Zipf vs. Zipf popularity: Average distance between searching and providing peer

7 Conclusion

Bibliography

[BCF+] L. Breslau, Pei Cao, Li Fan, G. Phillips, and S. Shenker. Web caching and zipf-like distributions: evidence and implications. [BCF+], pages 126–134.

[FP] Justin Frankel and Tom Pepper.

[GDS+03] Krishna P. Gummadi, Richard J. Dunn, Stefan Saroiu, Steven D. Gribble, Henry M. Levy, and John Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *19th Symp. Operating Systems Principles (SOSP)*, pages 314–329, October 2003.

[GEvS05] P. Garbacki, D. H. J. Epema, and M. van Steen. A two-level semantic caching scheme for super-peer networks. In *Tenth IEEE International Workshop on Web Content Caching and Distribution (WCW2005)*, pages 47–55. IEEE Computer Society Press, 2005.

[GEvS07] Pawel Garbacki, Dick Epema, and Maarten van Steen. Optimizing peer relationships in a super-peer network. In *27th IEEE International Conference on Distributed Computing Systems (27th ICDCS'2007)*, Toronto, Ont., Can., June 2007. IEEE Computer Society.

[KLVW04] Alexander Klemm, Christoph Lindemann, Mary K. Vernon, and Oliver P. Waldhorst. Characterizing the query behavior in peer-to-peer file sharing systems. In Alfio Lombardo and James F. Kurose, editors, *Internet Measurment Conference*, pages 55–67. ACM, 2004.

[KRP05] Thomas Karagiannis, Pablo Rodriguez, and Konstantina Papagiannaki. Should internet service providers fear peer-assisted content distribution? In *Internet Measurment Conference*, pages 63–76. USENIX Association, 2005.

[KTOI07] H. KOBAYASHI, H. TAKIZAWA, T. OKAWA, and T. INABA. An Efficient Control Mechanism for Self-Organizing Overlay Networks of Large-Scale P2P Systems. *Interdisciplinary Information Sciences*, 13(2):227–237, 2007.

[LCX04] Jiangchuan Liu, Xiaowen Chu, and Jianliang Xu. Proxy cache management for fine-grained scalable video streaming. In *INFOCOM*, 2004.

[MIB08a] Loubna Mekouar, Youssef Iraqi, and Raouf Boutaba. Personalized recommendations in peer-to-peer systems. 2008.

[MIB08b] Loubna Mekouar, Youssef Iraqi, and Raouf Boutaba. A recommender scheme for peer-to-peer systems. *Applications and the Internet, IEEE/IPSJ International Symposium on*, 0:197–200, 2008.

[MJ06] Mary Meeker and David Joseph. The state of the internet, part 3: The world's information is getting organized and monetized, November 2006.

[PB03] Podlipnig and Boszormenyi. A survey of web cache replacement strategies. *CSURV: Computing Surveys*, 35, 2003.

[SH06] Osama Saleh and Mohamed Hefeeda. Modeling and caching of peer-to-peer traffic. In *ICNP*, pages 249–258. IEEE Computer Society, 2006.

[SKKR01] Badrul M. Sarwar, George Karypis, Joseph A. Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *WWW*, pages 285–295, 2001.

[SMZ03] Kunwadee Sripanidkulchai, Bruce M. Maggs, and Hui Zhang. Efficient content location using interest-based locality in peer-to-peer systems. In *INFOCOM*, 2003.

[TK05] Saurabh Tewari and Leonard Kleinrock. On fairness, optimal download performance and proportional replication in peer-to-peer networks. In Raouf Boutaba, Kevin C. Almeroth, Ramón Puigjaner, Sherman X. Shen, and James P. Black, editors, *NETWORKING*, volume 3462 of *Lecture Notes in Computer Science*, pages 709–717. Springer, 2005.

[TK06] Saurabh Tewari and Leonard Kleinrock. Proportional replication in peer-to-peer networks. In *INFOCOM*. IEEE, 2006.

Appendix

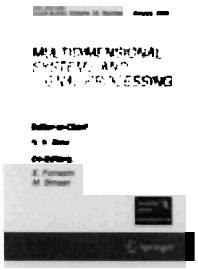
Index

Numbers written in italic refer to the page where the corresponding entry is described; numbers underlined refer to the definition; numbers in roman refer to the pages where the entry is used.

SHRS 08

Springer
the language of science

springer.com



Multidimensional Systems and Signal Processing
An International Journal
Editor-in-Chief: Nirmal K. Bose
ISSN: 0923-6082 (print version)
ISSN: 1573-0824 (electronic version)
Journal no. 11045
Springer US
Online version available
Online First articles available

Description

Multidimensional Systems and Signal Processing publishes surveys and research papers ranging from the fundamentals to important new findings. The journal responds to and provides a solution to the widely scattered nature of publications in this area, offering unity of theme, reduced duplication of effort, and greatly enhanced communication among researchers and practitioners in the field.

A partial list of topics addressed in the journal includes blurred and noisy image processing; multidimensional signal reconstruction from partial or incomplete observations and projections; signal modeling; spectral analysis and transform techniques; array processing; linear and nonlinear prediction and filtering of multidimensional processes; multidimensional spectrum estimation; multivariate approximation; multidimensional realization theory; and multivariate polynomial and matrix factorization schemes.

Abstracted/Indexed in:
ACM Guide To Computing Literature, Compendex, Computer Science Index, Current Contents/Engineering, Computing and Technology, Inspec, Mathematical Reviews, Science Citation Index, Science Citation Index Expanded (SciSearch), SCOPUS, Zentralblatt Math

Zeitschriftenbeitrag

Adapting the User Context in Realtime: Tailoring Online Machine Learning Algorithms to Ambient Computing

Zeitschrift	Mobile Networks and Applications
Verlag	Springer Netherlands
ISSN	1383-469X (Print) 1572-8153 (Online)
Heft	Volume 13, Number 6 / Dezember 2008
DOI	10.1007/s11036-008-0095-8
Seiten	583-598
Fachgebiete	Informatik
SpringerLink Date	**Samstag,** 4. Oktober 2008

PDF (1,1 MB) **HTML**

Johannes Schmitt[1], Matthias Hollick[1] ✉, Christoph Roos[1] and Ralf Steinmetz[1]

(1) Multimedia Communications Lab (KOM), Department of Electrical Engineering and Information Technology, Technische Universität Darmstadt, Merckstr. 25, 64283 Darmstadt, Germany

Received: 4 December 2007 **Accepted:** 30 June 2008 **Published online:** 3 October 2008

Abstract Ambient systems weave computing and communication aspects into everyday life. To provide self-adaptive services, it is necessary to acquire context information using sensors and to leverage the collected information for reasoning and classification of situations. To enable self-learning systems, we propose to depart from static rule-based decisions and first-order logic to define situations from basic context, but to build on machine-learning techniques. However, existing learning algorithms show substantial weaknesses if applied in highly dynamic environments, where we expect accurate decisions in realtime while the user is in-the-loop to give feedback to the system's recommendations. To address ambient and pervasive computing environments, we propose the FLORA—multiple classification (FLORA-MC) online learning algorithm. In particular, we enhance the FLORA algorithm to allow for (1) multiple classification and (2) numerical input values, while improving its concept drift handling capabilities; thus, making it an excellent choice for use in the area of ambient computing. The multiple classification allows context-aware systems to differentiate between multiple categories instead of taking binary decisions. Support for numerical input values enables the processing of arbitrary sensor inputs beyond nominal data. To provide the capability of concept drift handling, we propose the use of an advanced window adjustment heuristic, which allows FLORA-MC to continuously adapt to the user's behavior, even if her/his preferences change abruptly over time. In combination with the inherent characteristics of online learning algorithms, our scheme is very well suited for realtime application in the area of ambient and pervasive computing. We describe the design and implementation of FLORA-MC and evaluate its performance vs. state-of-the-art learning algorithms. We are able to show the superior performance of our algorithm with respect to reaction time and concept drift handling, while maintaining an excellent accuracy. Our implementation is available to the research community as a WEKA module.

Keywords ambient computing - pervasive computing - context-aware computing - machine learning

✉ **Matthias Hollick**
Email: matthias.hollick@kom.tu-darmstadt.de

Johannes Schmitt is a member of the research group Ubiquitous Communications at the Multimedia Communications