Imperial College of Science, Technology and Medicine
Department of Computing

Learning in Mobile Context-aware Applications

Jeremiah Smith

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Computing of Imperial College, August 2015

Abstract

This thesis explores and proposes solutions to the challenges in deploying context-aware systems that make decisions or take actions based on the predictions of a machine learner over long periods of time.

In particular, this work focuses on *mobile* context-aware applications which are intrinsically personal, requiring a specific solution for each individual that takes into account user preferences and changes in user behaviour as time passes.

While there is an abundance of research on mobile context-aware applications which employ machine learning, most does not address the three core challenges required to be deployable over indefinite periods of time. Namely, (1) user-friendly and longitudinal collection and labelling of data, (2) measuring a user's experienced performance and (3) adaptation to changes in a user's behaviour, also known as concept drift.

This thesis addresses these challenges by introducing (1) an *infer-and-confirm* data collection strategy which passively collects data and infers data labels using the user's natural response to target events, (2) a weighted accuracy measure A_w as the objective function for underlying machine learners in mobile context-aware applications and (3) two training instance selection algorithms, *Training Grid* and *Training Clusters* which only forget data points in areas of the data space where newer evidence is available, moving away from the traditional time window based techniques. We also propose a new way of measuring concept drift indicating which type of concept drift adaption strategy is likely to be beneficial for any given dataset.

This thesis also shows the extent to which the requirements posed by the use of machine learning in deployable mobile context-aware applications influences its overall design by evaluating a mobile context-aware application prototype called RingLearn, which was developed to mitigate disruptive incoming calls. Finally, we benchmark our training instance selection algorithms over 8 data corpuses including the RingLearn corpus collected over 16 weeks and the Device Analyzer corpus which logs several years of smartphone usage for a large set of users. Results show that our algorithms perform at least as well as state-of-the-art solutions and many times significantly better with performance delta ranging from -0.2% to +11.3% compared to the best existing solutions over our experiments.

Acknowledgements

I would like to express my profound gratitude to my supervisor Dr. Naranker Dulay for giving the opportunity to do research in a subject I am passionate about and guiding me throughout the PhD. I would also like to thank Dr. Luke Dickens for his precious feedback on the content of this thesis and the EU for financing the iCareNet FP7 project which I was part of.

This thesis is dedicated to my mother — Panda, my father — Bennett, my sister — Rebi and my grandmother — Nagyi.

"In HCI it's not the number of mistakes [a mobile context-aware system makes] that counts, it's about the acceptance of them by the user. If a smartphone app fails to react in the way I want it to in exceptional situations I might tolerate the mistakes, while if it repeatedly fails on a common task I might feel it doesn't work"

Paul Lukowicz, PerCom 2014

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Using machine learning in context-aware systems has the potential to overcome the tedious, and often impossible, task of entirely predefining a system's behaviour before deployment. Given the dynamic nature of human behaviour, machine learning is a technique to complement, or even replace, static system behaviour rules set by the user or an expert. This makes the overall system adaptive and convenient for long-term use. For instance, the context-aware Luna mattress allows users to set the mattress's temperature and also promises to 'learn and program itself to create a personalized schedule for bed temperature adjustments' [Sle15]. While both context-aware systems and machine learning are well researched topics in isolation, the challenges in combining them to produce a system that's *actually deployable* are largely unaddressed. These challenges arise from (1) the machine learning component requiring labelled data to infer desired system behaviour, (2) the system itself which must offer a user-friendly and user-tailored experience and (3) the user who might change his behaviour as time passes — also known as concept drift — requiring the system to be adaptive. In this thesis, we are specifically interested in *mobile* context-aware applications which we define as context-aware systems that interact with a single user through a mobile device such as a smartphone or wearable device.

1.1 Application Domains

Three applications domains will be used to help illustrate various concepts throughout the thesis. Depending on design and implementation decisions, the applications may use context that is gathered using sensors outside a mobile device and could have their computational cycles offloaded to a remote server, yet, we consider them as mobile context-aware applications because mobile devices are key to interaction with the user and external sensing and offloading are orthogonal to the problems this thesis is concerned with.

Activities of Daily Living Recognition

Activity Recognition applications make use of the presence or lack of certain activities in a user's daily routine to provide a given service or take an action. For example, Activities of Daily Living (ADL) are important in Healthcare as they provide an indication of a person's wellbeing [FNV10, KA76], one could think of an application that attempts to recognise ADLs and provide summaries for nurses and doctors to help track their patient's health over time. Collecting and labelling the ADLs must be done in a practical way so that the application can be deployed over an indefinite period of time. This could be done by showing the patient a summary of their ADLs at each day's end, allowing them to correct the application's predictions if need be which we term an *infer-and-confirm* strategy (Chapter 4). Some ADLs might be more important to recognise than others for the patient's safety. For instance, attaining a perfect true positive recognition rate for falls might be considered a priority compared to watching TV for a given application. This in turn necessitates optimised learning using user-specific and class-specific performance measures (Chapter 3). A patient's ADLs and the way they are performed is also bound to change as time passes leading to concept drift (Chapter 3).

Thermostat Auto-scheduling

Smart homes are a challenging application domain for context-aware systems because their success and acceptance greatly depends on their usability and how well underlying systems can adapt to users. Thermostat auto-scheduling implemented in the Nest system [Nes15] and the Luna Mattress are interesting use cases for our work as they are commercially available systems that learn user behaviours. Thermostat auto-scheduling is a simple task that consists in learning a user's preferred temperature setting on the different days of the week subject to exceptional and seasonal variations. We are interested in the mechanism to gather ground truth for the user's preferred temperature setting, the fact that users might prefer 'too hot' to 'too cold' or vice versa and, finally, that regardless of any normal seasonal variation in temperature, heating preference and personal schedules might change over time.

It is interesting to note that Nest published a whitepaper in November 2014 reaching a conclusion very similar to our infer-and-confirm approach to collect and label data: 'every interaction is treated as a way for the user to communicate with the device about his or her preferences for a particular temperature at a particular time and day of the week. In addition to considering active interactions, we also consider lack of interactions (indicating satisfaction with the current temperature), as well as the room temperature and whether the user is home or away. This provides a more holistic view of user preference than was considered previously' [Nes14].

Disruptive Smartphone Notifications

Smartphone notifications are an ideal use case for mobile context-aware systems as smartphones are the most widespread mobile device with an estimated 1.75 billion owners worldwide in 2014 [eMa15]. A solution to alleviate disruptive smartphone notifications would thus benefit a large number of people and could be implemented on existing systems. A context-aware application that could predict when an incoming push notification or incoming calls is disruptive given a user's context could automatically change the delivery method for the notification to lessen disruptiveness by using a silent alert instead of an audible alert for instance.

1.2 Research Statement

1.2.1 Scope

This work focuses on mobile context-aware applications that can be modelled as supervised binary class sequential learning tasks. In supervised learning, labels are available for all data points. This contrasts with semi-supervised learning where only some of the data is labelled. While there are machine learning techniques that leverage unlabelled data points as well as labelled points to increase prediction performance compared to supervised learning (e.g. co-training [RKSM14]), these are performance optimisation methods which do not solve the core machine learning challenges this thesis aims to address and are thus beyond the scope of it.

In binary class tasks, data points can belong to one of two classes, for instance, labels for incoming calls could be *disruptive* or *non-disruptive*. We chose binary problems as a simplification step, the methods proposed in this thesis are straightforwardly application to multi-class problems which is important as many context-aware applications such as ADL recognition can be multi-class learning problems. Finally, learners are assumed to be asked for predictions online, one point at a time, and subsequently receive the ground truth for the prediction — which we refer to as (supervised) sequential learning. This model of learning in context-aware systems is realistic in many context-aware systems where data is generated by a single user. Consequently, computational and space complexities of algorithms are only a minor concern which is reflected in the thesis. Instead, we focus on fast adaptation of mobile-context aware systems to cope with the unpredictable nature of human beings. Specifically, we do not consider context-aware applications that learn from data streams which are characterised by large volumes of partially labelled data. We argue many mobile context-aware applications do not generate streams of data because the rate at which a single person is able to change their context, and thus generate data points, is lower than any modern system's processing capacity and more importantly, the rate at which a person is able to label data is even lower. Because human-labelled data is usually expensive, we thus assume relatively small, fully labelled datasets.

1.2.2 Research Questions

Our research questions are concerned with concept drift and data collection issues due to the use of machine learning in mobile context-aware applications. Concept drift, which is a change in input data distribution is typically caused by a change in a user's existing behaviour patterns or the creation of new behaviour patterns, is difficult to recognise. Currently, concept drift on real datasets is *assumed* because of the way a dataset was generated (as is the case of real datasets where a specific event is thought to cause concept drift), or symptomatically observed due to a decrease in performance of learners across a dataset. Our interest in data collection methodologies comes from the fact most research tackles either only the learning side of intelligent mobile context-aware applications, not taking into account that the data used cannot be obtained in a practical setting, or the human computer interaction side of intelligent mobile-context aware applications, proposing approaches that work well on fixed sized datasets but do not account for potential changes in data distribution and thus also limit their applicability in practice. We take a more comprehensive approach to intelligent mobile context-aware systems driven by the following research questions:

Research Question 1. Can we devise a learner-independent measure for concept drift on an arbitrary dataset that enables us to quantify concept drift?

Research Question 2. Does the presence of concept drift always imply the need to forget existing data as suggested by the large majority of existing work? In some cases, for instance a new behaviour pattern emerging from a user, is it sometimes more beneficial to integrate new data with current data rather than automatically discarding some of the current data. If so, can we qualify and quantify these cases?

Research Question 3. Is it necessary to make a distinction between sudden, gradual, incremental and reoccurring concept drifts? Can we derive algorithms to cope with concept drift that are agnostic to the type of drift and perform at least as well as current state of the art solutions?

The following two questions are concerned with the usability issues in using machine learning in context-aware applications.

Research Question 4. Can we incorporate user preferences and cope with unbalanced datasets, both of which are important concerns when dealing with real data?

Research Question 5. Because mobile context-aware applications usually aim to simplify or better one's daily routine, how can we collect and label data from users in a convenient and practical way in applications deployed over indefinite periods of time?

1.3 Publications and Statement of Originality

The research in this thesis was conducted as part of an FP7 European project called iCareNet. iCareNet focused on the areas of healthcare, wellness, and assisted living applications. The goal of iCareNet was to introduce and make use of context-awareness in pervasive healthcare, to develop telehealth and preventive medicine. One aspect of iCareNet was to use technology to reduce the burden of 'non care' tasks done by medical staff by automating aspects of these. Our work contributes to this goal by developing a comprehensive approach to deploying mobile context-aware systems which use machine learning, addressing the cornerstone requirements to successfully implement intelligent applications in the environment of non technical users in a convenient way.

The following publications arose from the work carried out during the course of this PhD and I declare that this thesis was written by myself, and that the work it presents is my own, unless stated otherwise.

[SDT$^+$13] Jeremiah Smith, Naranker Dulay, Mate Attila Toth, Oliver Amft, Yanxia Zhang: **Exploring concept drift using interactive simulations.** *PerCom Workshops 2013: 49-54*
This paper presents a tool enabling the creation of interactive simulations with customisable concept drift events. These are used to test the suitability of learners when faced with single or concurrent concept drift occurring in models of human-centric tasks like activity recognition. This initial work suggests that hybrid, self-calibrating learners that are also able to incorporate ground truth might be appropriate for robust body-worn sensor activity recognition carried out over extended periods of time where concurrent concept drifts are likely to occur.

[JSS+13] Shahram Jalaliniya, Jeremiah Smith, Miguel Sousa, Lars Buthe, Thomas Pederson: **Touch-less interaction with medical images using hand & foot gestures.** *UbiComp (Adjunct Publication) 2013: 1265-1274*

This paper presents a practical system that uses an inertial body-worn sensor combined with a capacitive floor sensor for touch-less interaction with displays in a surgery context. It was designed to recognise 12 different gestures (6 hand gestures and 6 foot gestures) for interaction. A practical study showed it was possible to attain high accuracy with limited amount of training.

[SD14] Jeremiah Smith, Naranker Dulay: **RingLearn: Long-term mitigation of disruptive smartphone interruptions.** *PerCom Workshops 2014: 27-35*

This paper presents a new approach to smartphone interruptions that maintains the quality of mitigation under concept drift considering long-term usability. The approach uses online machine learning and gathers labels for interruptive events using implicit experience sampling without requiring extra cognitive load on the user's behalf. An Android application, RingLearn, was developed for handling incoming phone calls and tried by 10 participants using their own phones over 2 months. The gathered data, together with a post-hoc survey, confirmed the feasibility of the approach as well as highlighted the strengths and weaknesses of 3 different types of learners in 3 different modes for this application

[SLM+14] Jeremiah Smith, Anna Lavygina, Jiefei Ma, Alessandra Russo, Naranker Dulay: **Learning to recognise disruptive smartphone notifications.** *Mobile HCI 2014: 121-124*

This paper shows that dependent on the relative importance a user associates with detecting disruptive incoming calls versus non-disruptive calls, machine-learners can be used to lessen the occurrence of disruptive calls (for example, by silencing the phone's ringer when such a call is detected) without fully removing incoming call notifications, as putting the phone in a silent mode would. Out of the 6 benchmarked learners, an association rule classifier, not only outperformed all other benchmarked techniques but also outputs human readable rules which could be used to inform a user why a (non-)disruptive call prediction was made, an important HCI consideration.

[SRLD14] Jeremiah Smith, Anna Lavygina, Alessandra Russo, Naranker Dulay: **When did

your smartphone bother you last? *UbiComp Adjunct 2014: 409-414*
This paper attempts to answer the following questions: ‘Do users revise what they perceive as disruptive incoming calls as time goes by?’, ‘How do different types of machine-learners (lazy, eager, evolutionary, ensemble) perform on this task?’ and ‘Can we restrict the initial amount of data and/or the number of features we need to make predictions without degrading performance?’. These questions were answered on the Cambridge University’s Device Analyzer dataset regrouping over 17000 mobile phone usage traces. Out of the 10 selected datasets we found that the majority showed that users did revise what they considered disruptive calls, that online and windowed learning performed similarly indicating that the benefit of forgetting all old data does not markedly outweigh the cost of losing certain points that remain relevant for predictions and that a limited set of features performed just as well as using the initial selected set of features.

[Amf15] iCareNet group: **Context-Aware Systems: Methods and Applications.** *Springer, in production*
Each chapter of this book was written by a different researcher or group of researchers from the iCareNet EU project on a topic related to context-aware systems. In the chapter produced by Frank Bolton and myself, we combined and synthesised his work on interruption theory and modelling, and used my research on disruptive smartphone notifications as a use case for the chapter.

1.4 Thesis Structure

Chapter 2 gives a broad overview of the concept drift and mobile context-aware application topics, laying out fundamental concepts and seminal works, while the state of the art in each topic is explored in detail in the contribution Chapters 3 and 4.

Chapter 3 is concerned with concept drift due to changes in user behaviour. While most existing work addresses concept drift using either a form of windowed learning or ensemble learning, we explore the hypothesis that in context-aware applications, where data comes from human actions, changes in data distributions are likely to be local. This hypothetical concept drift — referred to as **local concept drift** — means that some parts of the input data space will stay stable throughout the dataset while others will require the use of adaptive learning techniques loosely linked to new habits appearing, disappearing or changing independently from each other and calling for more refined strategies than forgetting all 'old' data (typical windowed learning) or training learners on time-continuous chunks of data (typical ensemble learning). The chapter thus proposes a new way of quantifying concept drift on datasets and decouples concept drift and the *need to forget* which is not always implied by the former as we will show. It also proposes two learner-independent dynamic training set formation algorithms, Training Grid and Training Clusters, that only forget data when newer data is available to replace it in order to tackle the problem of local concept drift while still being able to handle cases where concept drift affects the entire dataset. Because data in mobile context-aware applications is likely to be unbalanced, that is to say the majority of points in a dataset belong to a single class, this makes traditionally used performance measures such as learner accuracy uninformative. Further, users might have class recognition preferences as in the case of spam recognition where false positives are typically perceived as more costly than false negatives, our algorithms are benchmarked using a proposed weighted accuracy performance measure A_w.

Chapter 4 closes the loop by addressing the practical data collection challenge in using machine learning as part of a mobile context-aware application. The challenge is to collect and label data in a way that the perceived benefit of using the application outweighs the increase in the cognitive load caused by the data gathering mechanism when deployed in the wild over an indefinite length of time. To address the said challenge, we propose to couple *implicit* experience sampling that infers data labels using a user's response to naturally occurring events and an confirmation step that informs them of the inferred label, with the option to change it. Finally, we test our contribution in its entirety over 6 real incoming call datasets collected using either our own RingLearn application or the Cambridge Device Analyzer application.

Chapter 5 summarises the findings of thesis and discusses their implications pointing to future work.

Chapter 2

Background

Machine learning is a set of methods used to determine a mapping between input and output values of a dataset or patterns in a dataset, automatically. It is used to predict the output values of unseen data or uncover patterns to better understand the underlying process that generated a dataset. The type of machine learning used in context-aware systems is normally the predictive kind, for example, where a machine learner is used to determine the best action for the system to take in a particular situation given a previous history of situation-action pairs.

A **context-aware system** is the combination of a computer program, a set of sensors (physical or virtual) and optionally a set of controllers, that provides functionality that depends on the sensed state of its environment and/or its user(s). We are specifically interested in *mobile* context-aware systems where the key sensing and/or interaction points between the system and its user(s) is a mobile device. A **mobile device** is any of the following: a smartphone, phablet, tablet, or any wearable device that can accept user input such as a head mounted device or smart watch.

We assume each instance of a mobile context-aware system has a single user as mobile devices are usually personal. Further, we focus on **intelligent** mobile context-aware applications — that is to say applications that use machine learning to map the sensed state of the environment and/or user to actions. These are the applications that have the highest barriers for adoption, highlighted in the challenges below, yet potentially offer the most impact in users' daily lives.

2.1 Challenges

Throughout the thesis we refer to these three main challenges for intelligent mobile context-aware systems:

Challenge 1. Collecting and labelling data

Using machine learning in a context-aware system requires labelled data i.e. situation-action pairs where the user's desired action is known. A system thus has to gather action labels in a way that the user's perceived benefit in using the system dominates the increase in cognitive load due to gathering action labels. This is an important point as machine learning performance is highly linked to the amount of data available to learn on, so that performance can be increased by gathering more labels but could make a system impractical.

A typical example of this is using *experience sampling* which is a common way to gather labels. Experience sampling queries the user for labels in different situations to gather situation-action pairs. In Activities of Daily Living recognition a context-aware system could query the user several times per day asking them to label their current activity. This could be acceptable for a small amount of time but in the long-run a user might prefer not to use the system at all rather than be interrupted too often.

The challenge is thus to collect labels in a user friendly way adapted for deployment over an indefinite period of time.

Challenge 2. Measuring Learner Performance

A context-aware system's performance can be measured in different ways, for instance, execution speed, memory requirements or battery usage. However, in the scope of this thesis, we are interested in how successful a system is in predicting the correct action in different situations which is not implementation dependent. It is directly linked to the underlying learner used in the system.

Learner accuracy, as defined in Appendix A.1, is the most widely used performance measure in machine learning. It tell us the percentage of times the learner was right in making predictions.

While it has a clear intuitive meaning it is unclear how accuracy translates into user *perceived* performance.

This can be illustrated using the fire alarm example. If we discretise a year's time into second slots with a fire alarm queried every second to know whether there is a new fire or not, we would get a certain number of points p, about $3.1 * 10^7$. If within that year there are no fires but the fire alarm goes off twice, the accuracy would be $\frac{p-2}{p} > 99.99\%$ and might be acceptable to a user. If within the next year there is a fire but the fire alarm fails to go off, the accuracy would be higher $\frac{p-1}{p}$ yet the alarm would probably be considered useless by the user. This highlights the problem of **asymmetric costs in making wrong predictions** across actions (in this case, fail to signal no fire correctly and fail to signal fire correctly) and **unbalanced datasets** which are exaggerated in the fire alarm example yet essential concerns in context-aware applications.

The challenge is thus to take into account user preferences when it comes to making wrong predictions and account for unbalanced datasets when measuring learner performance.

Challenge 3. Coping with Changes in User Behaviour

Coping with changes in user behaviour presents the most difficult challenge when deploying a context-aware application over an indefinite period of time yet is crucial given that human behaviour is variable and the environment we live in is highly dynamic. In practice, this means that a learner can face the following cases: new situation-action pairs appear in the data where the situation was not known beforehand, this might happen if a user goes to a new location for instance; situation-action pairs appear in the data where the action contradicts the previously observed action for the same situation, this might happen if the user changes his schedule for instance.

Formally, these events fall under the notion of concept drift in machine learning which means a change in the statistical properties of a dataset. Because experiments can only be conducted on a finite set of data, many works in the literature draw conclusions from experiments where the learners have no mechanism to cope with the said concept drift. This leads to a situation where a system might perform well on the specific time chunk of the experiment but will eventually degrade its performance if deployed for a long enough period of time.

The challenge is to design machine learning methods that can be used in a context-aware system to be deployed over indefinite periods of time without deteriorating their predictive performance if the user changes his behaviour or the environment changes its properties.

2.2 Machine Learning

The situation-action pairs mentioned in previous sections can be generalised into an input value $\vec{x} = (x_1, ..., x_n), x_i \in S_i$ and an output value, or label, $y \in S'$ pairs, where S_i and S' are sets. In the case of context-aware systems, S_i can be a continuous or discrete set. The n different x_i values, usually called features, can come from a large continuous set such as $\mathbb{R}$ if we are measuring GPS longitude for instance, or a small discrete sets such as $\{screen_{on}, screen_{off}\}$ if we choose to incorporate a mobile's device's screen state as part of a context-aware application. Similarly, S' can be continuous, in which case the learning task is called regression, or discrete in which case the learning task is referred to as prediction. The following definitions and explanations are based on [Bis06] unless specified otherwise.

Learning tasks can further be segmented into 5 different categories:

- **Supervised learning**

  In supervised learning, a set of $\vec{x}, y$ pairs are given (the *training* set) and are used by a learner to predict unknown y' values for previously unseen $\vec{x}'$ values (the *testing* set).

- **Unsupervised learning**

  In unsupervised learning, only a set of $\vec{x}$ is given and is used by a learner to find structure in the data to get insights on the underlying process that generated it. It can also be used to devise the existence of different labels without knowing what the different labels correspond to.

- **Semi-supervised learning**

Semi-supervised learning is similar to supervised learning except that a set of $\vec{x}''$ without the corresponding labels is also given which a earner can use to better predict y' labels.

- **Reinforcement learning**

  In reinforcement learning, instead of being given $\vec{x}, y$ pairs, we are given an initial $\vec{x}_0$ and a set of actions that a learner can take to obtain a $\vec{x}_1, y_1$ pair. y_1 is called the reward from transitioning from $\vec{x}_0$ to $\vec{x}_1$ using the action we chose. From $\vec{x}_1$ we can once again take an action to generate further data. Reinforcement learning is used to predict the best action to take for any $\vec{x}$.

- **Logic based learning**

  In logic based learning, $\vec{x}, y$ pairs are encoded as facts (background knowledge), and a learner derives a set of rules which entail (informally, best explains) the $\vec{x_i} \to y_i$ mappings found in the training set. Logic based learning has an advantage in that it produces human-interpretable rules, which is often times not the case for learners in the other categories.

2.2.1 Modelling the Learning Flow in Mobile Context-aware Systems

Learning in mobile context-aware systems deployed over indefinite periods of time is sometimes referred to as *lifelong learning* [KH07]. The scope of the thesis is systems where the underlying learning task can be modelled as a binary class supervised sequential learning problem. Our assumptions differ from stream learning where data is assumed to be abundant and only partially unlabelled so that learner computation speed and data storage requirements are of concern. Sequential learning [Zli10], illustrated in Figure 2.1, describes the order in which learners receive labelled data and update their internal model or prediction heuristic. Initially, learners are given a chunk of data to train (**training phase**), in mobile context-aware systems this could be considered as part of the initial system configuration and depending on the actual system it could come from the user filling in a questionnaire or using the system in a data collection mode for some time and then manually labelling the data for instance. Next, each time the system's target event is detected, the learner is asked to make a prediction based on the sensed situation (context), resulting in the system potentially taking an action (**prediction phase**). Finally, the ground truth label for the previous prediction is given to the learner so that it can update its internal model or predictions heuristic (**update phase**). This process goes on indefinitely.

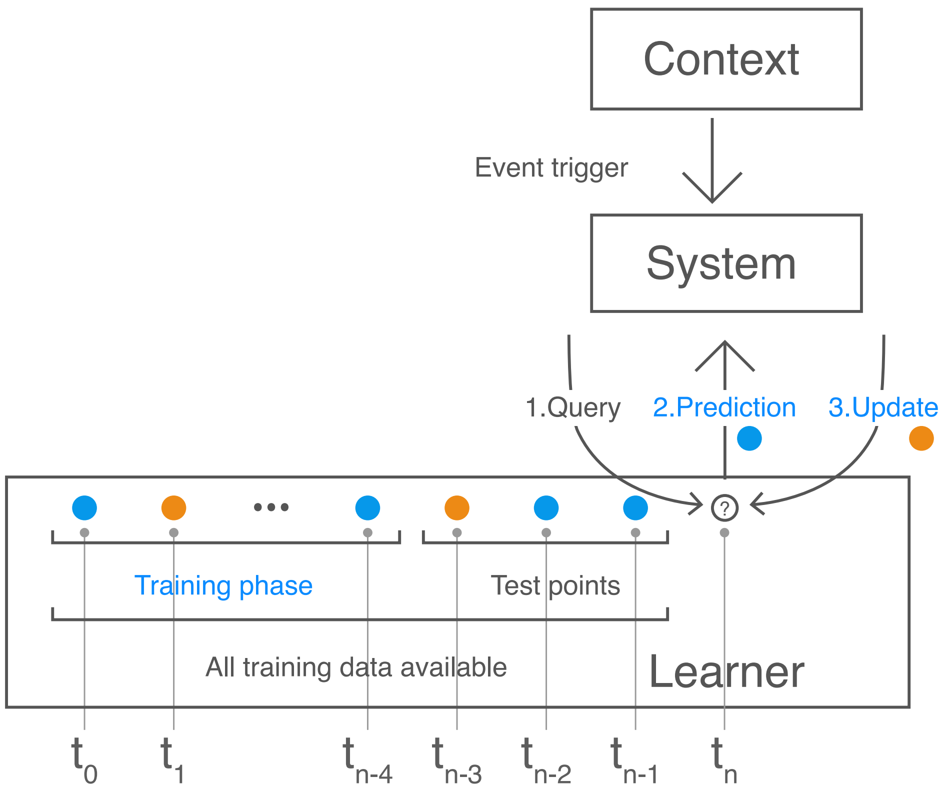


Figure 2.1: Illustration of the training, prediction and update phases in sequential learning as a function of time t

2.2.2 Machine Learners

Buliding upon the core notions of Appendix A.2 machine learning is centred around the following concepts: in the supervised learning, we define **a learner** as an algorithm $L(\vec{x}_a) \rightarrow y_a$ that maps an input value $\vec{x}_a \in S_1 \times ... \times S_n$ to a value $y_a \in S'$ i.e. a method to map arbitrary input points to a known set of labels.

Learners can be [NJ02]:

- **Direct discriminative maps**, in which case it maps $S_1 \times ... \times S_n \to S'$ without probabilities playing a direct role. A simple example is a linear classifier that uses a line or plane $\vec{b}^T\vec{x} + c = 0$ mapping $L(\vec{x}_a) \to 0$ if $\vec{b}^T\vec{x}_a + c < 0$ and 1 otherwise for $S' = \{0, 1\}$.

- **Probabilistic discriminative maps**, these implicitly or explicitly estimate $p(y_i|\vec{x_a})$ $\forall y_i \in S'$ to predict the label of $\vec{x}_a$, generally the label with highest posterior class probability. If the learner models $p(y_i|\vec{x})$ using a model with a fixed number of parameters it is called *parametric* and the learning procedure can be summarised as choosing a model and finding the parameters that make the assumed model best fit the training data, as in the case of Logistic Regression. Conversely, if the learner does not assume a specific model for $p(y_i|\vec{x})$ it is called *non-parametric* and the learning procedure varies from learner to learner. Example are SVMs, Decision Trees and nearest neighbour classifiers. Probabilistic discriminative maps have an advantage over direct discriminative maps as probabilities can used to assess the confidence of predictions using the posterior class probabilities values.

- **Probabilistic generative maps**, instead of estimating $p(y_i|\vec{x})$, generative maps estimate the joint probability distribution $p(\vec{x}, y)$ and use the probability product rule to compute $p(y_i|\vec{x}_a)$ and make predictions (as in probabilistic discriminative maps). The difference between discriminative and generative maps is that the latter can be used to artificially generate data using $p(\vec{x}, y)$. Generative maps can also be parametric, as in the case of fitting a Normal distribution to a dataset or non-parametric as in the case of Hierarchical Dirichlet processes. Context-aware applications usually do not need to generate data although parametric generative maps are sometimes used to make predictions, Hierarchical hidden Markov Models in Activity Recognition for instance.

Finally, learners also have hyperparameters, whether they are parametric or not.

Hyperparameter

Hyperparameters are learner parameters set by the practitioner before learning takes place. They are different from the parameters a parametric learner fits to the model it uses to make predictions as they remain fixed during training. They can be thought of as corresponding to a particular setting for a learner, for instance the heuristic for creating internal nodes in a Decision Tree.

The main assumption of learning as presented in this section is that an $L(X) \to Y$ map devised from training data can be used to make predictions on testing data. Specifically, the assumption is that the underlying data probability distributions of the training and testing sets are the same. In context-aware systems this assumption is optimistic as a person might change their habits (informally, a change in $p(x,y)$) or create new habits (informally, a change in $p_X(x)$) which can mean a fixed $L(X) \to Y$ mapping learned during training is not valid over the entire testing set, this is called concept drift which we now expand upon.

2.2.3 Concept Drift

The most common definition of concept drift is given by Kelly et al. [KHA99]

Kelly's Definition of Concept Drift
Given a set of time ordered instances $\vec{x} \in X, X \in \mathbb{R}^d$ where $d \in \mathbb{Z}_{>0}$ each belonging to one of n classes c_i $i = 1, 2...n$. Concept drift is said to occur if one or more of the following event happen as time goes by:

- i) The marginal distribution $p_X(X)$ changes
- ii) The marginal distribution $p_C(X)$ for two or more classes changes
- iii) The conditional distribution $p(X|C)$ for two or more classes changes
- iv) The conditional distribution $p(C|X)$ for two or more classes changes

The immediate question that arises from this definition is what is mean by 'a distribution changing'. In practice, we only every observe samples from an underlying distribution so that one must choose a data property (for example one of the features' moments) to observe the change over and account for sampling uncertainty in order to apply this definition. These two points are addressed later on in our concept drift measure algorithm. We will also later refine this definition to suit the needs of context-aware systems, and we note that because of Bayes' rule, in general, concept drift will affect $p(C|X)$ which is what we are interested in. Intuitively, in a mobile context-aware system, if a person changes their behaviour $p_X(X)$ is likely to change as features might take on new values (visiting new a location for instance) or the relative frequency of feature values might be altered (change in schedule for instance). Similarly, $p(C|X)$ and $p(X|C)$ are affected if a user changes the desired action to be taken by the context-aware system for a given situation (context), which in turn can affect $p_C(C)$.

Types of concept drift

Historically, concept drift is usually characterised as being of 4 different types as illustrated in Figure: 2.2 [GvB+14]. In the definitions below, a core distribution refers to $p_X(X)$, $p_C(X)$ or $p(X|C)$ — $p(C|X)$ being directly dependent on those three distributions.

- **Sudden drift:** is when a core distribution changes abruptly. In Activities of Daily Living Recognition this could happen if a user suddenly stops performing a certain activity.
- **Gradual drift:** is similar to a sudden concept drift except that the core distribution is interleaved between an old and new distribution until the new distribution gradually dominates the old one completely. For Disruptive Smartphone Notifications a user might work one Saturday a month for some time, affecting what the user perceives as being a disruptive smartphone notification once a month compared to usual weekends, and might then start working increasingly often on Saturdays, gradually replacing what they perceive as being a disruptive notification for that day.
- **Incremental drift:** happens when there is a progressive shift in a core distribution. In Thermostat Auto-scheduling this would correspond to the user starting to prefer increasingly high temperature settings as winter comes.
- **Reoccurring drift:** is similar to sudden concept drift except that the same old core distribution can become valid again after some time. In Activity Recognition, this could

happens if a user starts performing an activity he stopped doing again.

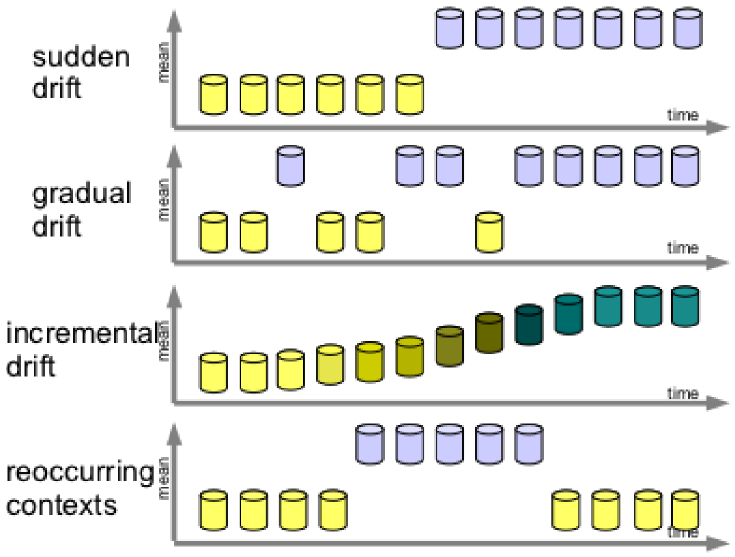


Figure 2.2: The different types of concept drift and how they might affect the mean of a random variable (taken from [Zli10])

In practice, classifying concept drift as belonging to any of these 4 categories is subjective and an ideal solution should cope with concept drift regardless of its type as further discussed in Chapter 3. The key to cope with concept drift lies in the learner employed in a system. We now review the main types of adaptive learners.

Adaptive learning

Adaptive learners are learners that continue to refine their internal prediction model after the initial training phase by integrating newly available labelled training points, they fall into the online learner category. We use the definition of offline and online learning given by Fern et al [FG00]

Online and Offline Learning

Offline learning algorithms take as input a set of training instances and output a hypothesis. In contrast, online learning algorithms take as input a single labelled training instance as well as a hypothesis and output an updated hypothesis. Thus, given a sequence of training instances an online algorithm will produce a sequence of hypotheses.

In the above definition, an algorithm is what we call a learner L and a hypothesis is what we call a $L(X) \rightarrow Y$ map, where X is the set of possible inputs (contexts) and Y the set of the possible outputs (classes for contexts). When we say a learner is *not adaptive* we mean it learns *offline*. In offline learning, learners are trained on the training set and are used to make predictions on the testing set without revising their internal model or prediction heuristic throughout. The simplest way a learner can be made adaptive is by **retraining** it as new data labels become available during testing i.e. periodically growing the training set with new labels and repeating the initial training step of learning every so often. This is a form of *online* learning that can be achieved with any learner and which we refer to as *online retraining.* Some learners are specifically designed for online learning having the ability to integrate each new labelled datapoint without having to go through an entire retraining step as in the case of Multilayer Perceptrons which can update their internal weights with each new point [WM03].

Given the accepted definition of concept drift, it is possible to have a dataset where online learning achieved through retraining is enough to handle the concept drift present on the dataset, a specific example of this will be shown on the Ž-luxem dataset in Section 3.4.3. However, research in the field of concept drift is focused on problems where this type of online learning is not the best adaptive learning technique — which is the case in general.

The seminal paper in concept drift was published by Widmer et al. and is concerned with concept learning [WK93] (classifying objects into different classes given their description in terms of discrete features) and there are five recent doctoral theses on using adaptive learning to handle concept drift. The closest work to ours was published by Žliobaite [Zli10] investigating learner adaptivity through dynamic training set selection . Similarly to this thesis, she focuses on real world dataset, some of which we reuse in Chapter 3, and proposes the FISH dynamic training set selection algorithms which we also benchmark in our experiments. Garnett [Gar10] is concerned with concept drift in large streams of data. He proposes a Dynamic Logistic Regressor (DLR) and a Gaussian Processes for Global Optimization (GPGO) algorithm to cope with missing and corrupted data labels. Our work differs from his because we assume small and fully labelled datasets and propose learner independent methods to cope with concept drift. Minku [Min11] researches ensemble learning for handling concept drift. He proposes a Diversity for Dealing with Drifts (DDD) method to improve learning accuracy in a base learner and drift agnostic way. While his concern about the popular categorisation of concept drift mentioned in Section 2.2.3 and his focus on drift agnostic learner adaptivity concords with our work, his DDD method is a symptomatic method as it explicitly detects concept drift to trigger a response within the ensemble. His work also differs from ours as ensemble learning, as we later show in Chapters 3 and 4, is best fitted to stream learning applications which contrasts with the class of mobile context-aware systems we are interested in, characterised by relatively small fully labelled datasets. Conca [Con12] also focuses on ensemble learning but in a semi-supervised setting. While we could only access the thesis abstract, it seems that his work is concerned with applications where unlabelled data exists and is plentiful which is again opposite to the types of problems we are interested in. Finally, Lindstrom [Lin13] presents 2 adaptive learning methods for learning from semi-supervised data streams. The proposed Decision Value Sampling (DVS) and Confidence Distribution Batch Detection (CDBD) algorithms assume that learners have a mechanism to obtain labels from a data stream and aim to minimise the number of queries needed for a learner to adapt to concept drift, which is not relevant for the class of mobile context-aware systems considered in this thesis.

Concept Drift Learner Adaptivity Overview

To give the reader a bird's-eye view on concept drift adaptation techniques we divide approaches into 3 different categories and highlight representative examples in each case. For further reference we also recommend a recent concept drift literature review by Gama et al. [GvB$^+$14] which uses a different categorisation of adaptive learning techniques. An in-depth literature survey concerned solely on the type of concept drift relevant to mobile context-aware system is given in Chapter 3.

In the case of concept drift, learner adaptivity can be categorised into 3 paradigms:

- **Base Learner Adaptivity**

  In base learner adaptivity the drift adaptation process is part of the learner. It is thus *not* *learner independent* and is made up of custom algorithms. Decision trees that dynamically reconfigure their nodes when faced with concept drift are the most popular example of this [HSD01, GFR06, IGD11]. A well known example is Concept-adapting Very Fast Decision Trees [HSD01]. In VFDT, a decision tree is learned online by recursively splitting leaf nodes as new data is made available to the learner. The Concept-adapting extension of VFDT works by growing an alternate decision tree in the background using a sliding window of the most recent training instances, once the alternate subtree becomes more accurate on the current sliding window data than the current tree, it replaces it and another alternate decision tree starts to grow in the background. Garnett's DLR and GPGO mentioned above also fall into this category. The trade off in using custom algorithms rather than learner independent algorithms is performance versus flexibility. The former might perform very well in specific applications but because their base learners are immutable, system designers have to replace the entire concept drift adaptation mechanism if they wish to take advantage of another type of base learner. For instance, a system designer might want to start using a logic based learner able to output human readable rules to make a system more user-friendly.

- **Adaptive Training Set Formation**

  Adaptive training set formation can be achieved by either selecting a subset of available training instances (dynamic training set formation) or weighting training instances to bias the learner's internal model (instance weighting). Dynamic training set formation is thus a learner independent paradigm while instance weighting is indirectly learner dependent as not all learners are able to take instance weights into account for training. Most dynamic training set formation strategies are based on a training set sliding window of the most recent instances [BG07, KZ09, ZK09, BvdAvP11, DSJ14, LZL14] . For instance the WR* algorithm [BG07] tries to detect when a concept drift event occurs in the data and estimates the most likely training instance at which this happened to adjust the training window size and retrain the base learner accordingly. Žliobaite's FISH algorithms cited above also fall into this category although they do not use a time continuous window of instances, rather, they dynamically form training windows by considering the age and spatial proximity of available training instances compared to the current testing instance. An example of instance weighting is given in [Sen14] where instance weights are exponentially decayed to train a Naive Bayes classifier according to instance age. Adaptive training set formation is the most flexible concept drift adaptation technique as base learners are interchangeable and adaptation is learner independent with a potential performance trade off as base learners will react differently to a given training set.

- **Ensemble Learning**

  In ensemble learning, a collection of base learners are combined to make predictions by voting on testing instance labels. The concept drift adaptation mechanism lies in the ensemble's voting strategy and/or the ensemble's strategy to add and delete base learners. These approaches are thus learner independent. Examples include dynamically weighted voting strategies [EP11, TPCP06a, NYO05] and responding to concept drift by varying the amount of base learners in the ensemble [MWY10]. The Adaptive Classifiers-ensemble is a typical example [NYO05] of this. The ACE algorithm defines a batch size and either trains a new base learner on every new batch or when it detects that concept drift has occurred. To detect a concept drift event, ACE computes lower and upper bounds on the accuracy of a learner on the previous batch of instances, if the maximal base learner accuracy falls outside this range concept drift is assumed. The performance of learners on the previous batch of instances are used as weights when performing weighted majority voting. The DDD method mentioned above falls into this category. Ensemble learning is more flexible than base learner adaptivity as base learners can be changed according to the system designer's will, although it is constrained by its voting process making it less versatile than adaptive training set formation. Specifically, for ensemble learning to perform well, large amounts of data are required so that ensemble learning is generally more suitable for stream learning than sequential learning.

2.3 Mobile Context-aware Systems

2.3.1 Context

We use the popular definition of context given by Dey [ADB+99, Dey01].

Context

'Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves'

In mobile context-aware systems, context is gathered using either physical sensors such as proximity sensors and microphones or virtual sensors such as device time or IP address that are gathered from software applications. Context can be classified into 6 categories [CK00] which aim to answer the questions who?, what?, where? and when?:

Time context such as time, time-zone or season

Location context such as GPS coordinates or currently used cell tower id for GSM/CDMA capable devices

Environment context such as light condition or atmospheric pressure

Motion context such as accelerometer or gyroscope data

Computing context such as calendar events or the list of running applications

Computing activity context such as tracking when the user opens an app or the last time they used the mobile device

The context variables a system uses can be sensed directly through the device it is deployed on, for instance a system might try to recognise whether the user is walking, running, cycling, or training using an onboard accelerometer [CCH+08, GSB14, SFH+11], through a server in charge of aggregating and propagating context as in the case of the AwarePhone which gathers location and duty information for surrounding hospital workers in an smart hospital setting [BH10] or dynamically through localised beacons for instance at a restaurant where the menu can be provided to systems that are nearby [PA03].

2.3.2 Mobile Context-aware System

Context-aware systems in general leverage context to change the information they record, the information they display and/or the actions they take. There are two types of context-aware systems, **passive context-aware systems** which simply record and/or display information such as smart tourist guides which show information relevant to the user's current location [HCS05, HMM06] and **active context-aware systems** which are able to take actions [MN13]. This thesis focuses on mobile context-aware systems. A **mobile context-aware system** (application) is a single computer program running on the user's mobile device, or a set of collaborating computer programs one of which is running on a user's mobile device, which provides a functionality or service to the user that is dependent on context. A mobile context-aware system (application) can be active or passive.

The simplest active mobile context-aware systems are what we would define as **static** systems which have a pre-programmed context $\rightarrow$ action map (or rules), a device that changes its display orientation based on gyroscope data or a smartphone application that alerts the user when battery charge is computed to be insufficient to reach the next charging point [RSHI08] are two examples of this. On the contrary, in **intelligent** systems, the context $\rightarrow$ action map is inferred using machine learning — these are the types of systems we are specifically interested in and are further discussed in the following section.

The first deployed mobile context-aware application was called *birddog* and ran on the ParcTab PDA [SAG$^+$93]. The application enabled users to see where colleagues were located in a smart office environment building upon the Active Badge system (wearable beacons) [WHFG92]. Shortly after, the ParcTab also enabled the execution of arbitrary UNIX commands based on a user's location and status (arriving, departing, settled-in,missing or attention) using 'Context-triggered Actions' [SAW94] (pre-programmed context $\rightarrow$ action map) making it an active static mobile context-aware system according to our definitions. The ParcTab is representative of initial research in mobile context-aware systems which was focused on prototype development. Since then, the field has branched out into several areas of research topics which build upon each other. The 6 key research areas in mobile context-aware systems are illustrated in Figure 2.3. We briefly review and give examples for each layer defining the scope of this thesis within mobile-context aware systems and how it relates to the other key research topics of the field.

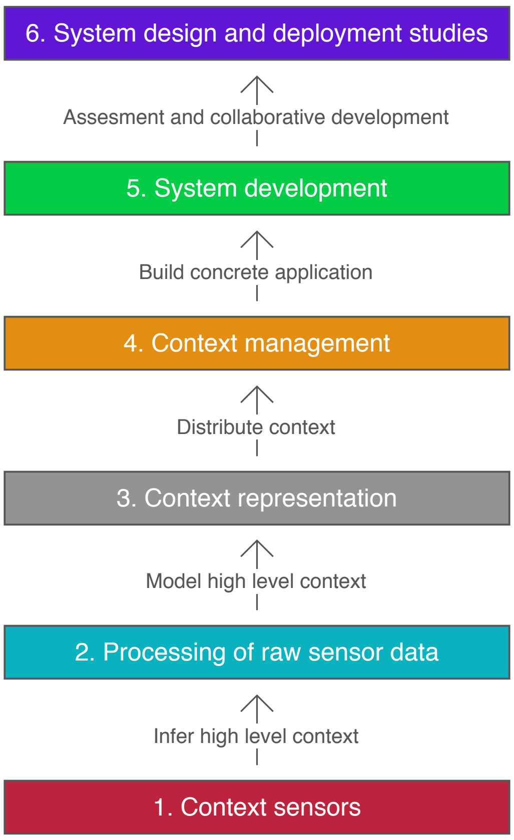

6. System design and deployment studies
Assesment and collaborative development
5. System development
Build concrete application
4. Context management
Distribute context
3. Context representation
Model high level context
2. Processing of raw sensor data
Infer high level context
1. Context sensors

Figure 2.3: They 6 key research areas in Mobile context-aware systems and how they relate to each other

The first area of research is concerned with the development of physical sensors and sensor platforms (embedded systems capable of outputting digital signals) to gather context. For instance, we might wish to know the high level physical activity of a user at any point in time which requires hardware sensors to translate user motion into discrete context variables. The main challenges in the area are *sensor selection*, *sensor precision*, *energy consumption* and *form-factor* [BVP$^+$14], examples of available commercial products are the Fitbit line of activity trackers [Inc15] or Shimmer wearable sensors [Sen15], custom research solutions include the ETHOS [HAW+10] and HedgeHog wristbands [fTD15].

The second layer is concerned with gathering high level context that is directly exploitable by mobile context-aware systems. High level context can be extracted from low level sensor signals by using *signal processing* such as thresholding and machine learning in which case it is called context inference. Going back to our previous example, accelerometer and gyroscope signals can be used to infer a user's physical activity [EPC08, SZL+10, HTM11, RB11] with a recent in depth tutorial available in [BBS13]. High level context can also be directly gathered from virtual sensors (from a device's API or external APIs for instance), in both cases, high level context is subject to noise and uncertainty which are the main challenges in this field [STA+12].

The third layer treats about representing, or modelling, high level context. Many times this is trivial, the simplest form of context modelling is using a Key-Value model which essentially makes context available to applications as environment variables but can also be more sophisticated as in the case of *ontologies* which can be used to precisely describe the environment a system is deployed in, specifying relationships between context variables and enabling reasoning over these [SLP04]. For instance COBRA-ONT [CFJ03] based on the OWL ontology [MVH+04] was made to describe locations, agents, events and their associated properties in a smart office setting, as an example, if a user's phone is detected via a bluetooth scan in a specific room the COBRA-ONT is used to deduce that the user is also in that room, that he is in the corresponding office building and that he is at work that day.

The fourth layer is concerned with how to manage context across multiple mobile context-aware systems so that systems do not have to implement their own context sensing mechanism. Context management is implemented as context-aware development frameworks (middlewares) which centralise context information and distribute it to requiring applications [LSNJ11, BCFF12]. For instance, in the Android OS the recent cODA open-source framework [FDN+14] uses 2 types of components to enable easy development of context-aware applications. *Observers* take care of the 'Context sensors' layer and extract high level context from smartphone physical and virtual sensors while *Deciders* use high level context to notify applications of context events they have subscribed to. Applications can then decide whether or not to act upon these. Practitioners can thus reuse existing Observers and Deciders to make an application or implement their own if further context is required which can then be reused by others.

The fifth layer is concerned with mobile context-aware system development. Literature in this area explains why and how actual systems are developed specific to a target use case and encompass different elements such as *interface graphic design* [HM06], *context choice*, *hardware choice*, *interaction choice* and *energy efficiency considerations* [LKLZ10]. Examples include tourist guides [HMM06, MRGS14], hospital smart devices [Bar04, BHMS06, BH10, DDL15] and fitness monitors [CCH$^+$08, RB11, RLGB$^+$14].

The sixth layer is concerned with mobile context-aware system design principles [HCS05], system development methodologies such as *participatory design* [BHMS06] and *user centred design* [Gou95, DDL15] and user studies of systems deployed in the wild [BH10]. This is especially relevant in sensitive application areas where a badly designed system can have harmful consequences such as a hospital. The AWARE architecture is perhaps the most in depth participatory design example available [BHMS06, BH10]. The architecture was developed over 4 years using detailed field studies to understand the requirements of clinicians, future workshops [HLP97] (a three phase brainstorming process where participants formulate their current work problems, envision solutions and then reflect on the steps needed for a possible implementation) to derive potential solutions, paper and video prototyping to concretise context-aware system behaviour, and 'a series of design workshops in which different versions of the systems were evaluated by the clinicians'. This led to the elaboration of the previously mentioned AwarePhone and the AwareMedia public display which gives an overview of the location and activity of clinicians in a ward (not a mobile context-aware system) as well a set of application-specific insights on the workflow of clinicians and design principles context-aware systems must take into consideration if they are to be deployed in a hospital setting.

We point the reader to the main reviews in the field for further reading [SAT$^+$99, CK00, MN13, Sch14, YLS$^+$14]. In our case, the scope of this thesis are the last two layers, namely system development, specifically addressing disruptive incoming call management and usability issues in gathering labelled data for *intelligent* mobile context-aware system.

2.3.3 Intelligent Mobile Context-aware Systems

Intelligent mobile context-aware systems, which we sometimes abbreviate to intelligent mobile systems or intelligent applications, refer to systems where the context $\rightarrow$ action map is inferred and adapted using machine learning. Expanding the introduction of Chapter 1, this offers three potentially compelling benefits compared to systems with static context $\rightarrow$ action maps. First, it might be laborious, or even impossible, to define a system's behaviour before deployment. Machine learning removes the task of explicitly defining the system's behaviour instead only requiring desired system behaviour examples (labelled training data) to generalise a context $\rightarrow$ action map for arbitrary contexts. Second, pre-programming the system's behaviour with a static map requires the system designer to make assumptions about the user and their use of the system. While this might be perfectly acceptable in some cases, changing a device's display orientation to match the user's for instance, in more complex applications such as Thermostat Auto-scheduling, there might not exist a system behaviour adapted to all users as they might each have a different way of using the system and different expectations when it comes to its behaviour. Machine learning is a scalable way to tailor systems to specific users by training them on individual users' desired system behaviour examples. Finally, even when it is possible to exhaustively define a system's behaviour using a static context $\rightarrow$ action map, there is no guarantee the user won't change their mind about the desired system behaviour. Even for simple examples such as automatic device display orientation, some users will prefer to disable the functionality after trying it out for some time — the problem of concept drift. An **intelligent mobile context-aware system** is a mobile context-aware system where the system's context $\rightarrow$ action map is partially or entirely defined using machine learning.

We note that this definition does not apply to the case where machine learning is used in the 'Processing of raw sensor data' layer only. For instance, if a smartphone application uses learning to convert accelerometer data into high level activity context but uses a static map to decide which action to choose, we do not consider it as an intelligent mobile context-aware application. This is because the application's *intended* behaviour is static and does not depend on how noisy the context inference component is — although the application's perceived performance will. In this thesis we assume that high level context is readily available focusing on methods to leverage machine learning in real world scenarios where the context $\rightarrow$ action maps are likely to change as time goes by.

While intelligent mobile context-aware systems might be the right solution in some cases, it is not the panacea for all application scenarios. Especially in the case of *active* mobile context-aware systems, the uncertainty brought by the use of machine learning could have dramatic consequences in critical application areas such as healthcare or the military where a mismatch between the system's action and the ground truth action can be very harmful. Intelligent mobile context-aware systems are thus usually researched in cases where incorrect actions on the system's behalf have benign consequences.

Examples of such systems (applications) include: Recommender systems [BKL+11, HMB12, PTG14] such as InCarMusic which recommends tracks based on traffic state [BKL+11], disruptive smartphone notification management applications [MCRL11, RDV11, OTN14] our use case for Chapter 4 or even digital behaviour intervention applications where, for instance, signs of depression are recognised and different actions (e.g. displaying links to theatre tickets) are explored by the applications until the right 'therapy' is learned [PM14a]. In practice however, very few intelligent context-aware systems go past the prototype stage where they are tested over shorts periods of time in a controlled environment. As captured by our research challenges, we argue this is mainly due to the need for systems to adapt to changes in user behaviour, a recognised but seldom addressed issue [MN13], and the collateral requirement for labelled data throughout deployment, the topic of the following two chapters. Chapter 3 focuses on quantifying concept drift on arbitrary datasets and the adaptivity mechanisms for learners to be able to cope with the said drift. Chapter 4 then investigates a disruptive smartphone notification use case which assembles and evaluates the proposed solutions to the challenges presented in Section 2.1 of intelligent mobile context-aware applications.

2.4 Summary

This chapter starts by presenting the 3 challenges that guide the work in this thesis. These are in the areas of data collection and labelling, measuring learner performance and coping with changes in user behaviour in mobile context-aware systems. The core concepts are then described in the contribution Chapters 3 and 4.

The background section on machine learning gives an overview of the existing types of machine learning and machine learners, justifying our choice to model the mobile context-aware systems that we are interested in as binary class supervised sequential learning problems. The focus of this background section within machine learning is on concept drift, the phenomenon linked to changing user behaviours as time passes and the different adaptive learning techniques to cope with it.

The chapter concludes with related work on mobile context-aware systems. It classifies the research into 6 areas that build upon each other, situating the work in this thesis in both the system development and system design and deployment studies areas.

Chapter 3

Measuring Concept Drift and Forgetting Strategies

This chapter addresses two challenges in deploying intelligent mobile context-aware applications over long periods of time (Challenges 2 and 3). Measuring the performance of machine learners in human-centric applications, for which we propose to use a weighted accuracy measure, and adapting to changes in user behaviour when deploying intelligent mobile context-aware applications over long periods of time, for which we propose to use forgetting strategies. Changes in user behaviour represents by far the largest challenge because they pose the problem of knowing how to integrate new data and how to *forget* old data that corresponds to obsolete behaviour patterns. These *forgetting strategies* are only ever implicitly mentioned in the literature and while it is easy to construct cases where existing forgetting strategies fail, this chapter not only proposes new forgetting strategies but also introduces a new approach to determine whether forgetting strategies are likely to increase performance on a particular dataset compared to offline learning and online retraining. In addition, we benchmark existing and proposed forgetting strategies on 8 datasets with two in depth case studies to demonstrate their behaviour as well as their relative performance.

3.1 Measuring Concept Drift

Before taking action to manage concept drift, we first need a means to measure its presence on an arbitrary set of time ordered instances. We give a definition of concept drift adapted to the context-aware systems we are interested in based on the definition given by Kelly et al. in Section 2.2.3.

Concept Drift

In a sequential learning framework as defined in 2.2.1, given a set of time ordered instances $\vec{x} \in X = (X_1, ...X_n)$ with each X_i being a discrete or continuous random variable and $\vec{x}$ belonging to class c^+ or c^- associated with the discrete random variable Y, concept drift is said to occur if the posterior distribution $p(Y|X)$ changes as time goes by. This implies one or more of the following:

- The marginal probability distribution $p_Y(Y)$ has changed
- The marginal probability distribution $p_X(X)$ has changed
- The class conditional probability distribution $p(X|Y)$ has changed

This restates the Kelly's definition by defining concept drift as a change over time in a dataset which affects the posterior distribution $p(Y|X)$ — precisely the distribution we are approximating when making predictions, it also extends the notion of concept drift to mixed discrete and continuous spaces for X which is likely to be the case in context-aware systems . Intuitively, concept drift represents an event that might affect the validity of the model or heuristic we are currently using to make future predictions.

A possible solution to measure concept drift is to track changes in statistical characteristics of instances throughout a dataset. For example, a change in moving window feature mean across a dataset indicates the presence and type (none, sudden, gradual, incremental or reoccurring) of concept drift [BvdAvP11, RMM$^+$13]. One can also use statistical tests to inspect consecutive data segments and find statistically significant differences between segments, also a sign of concept drift [AB13, KBDG04]. The issue with this type of approach is that it does not tell us what to do about the drift. For example, in the case of sudden concept drift, if we observe a jump in a feature's mean value at some time t, it might be the case that class boundaries change in a way that does not produce future misclassifications by keeping the current learnt class boundaries or simply requires retraining the learner using all newly available training instances to refine previously learned boundaries. Further, in practical cases such as mobile context-aware applications, detecting the presence of concept drift in a dataset is irrelevant if it does not affect the performance measure by which the application will be judged by the designer or end-user.

Another way of approaching the problem is thus to have a performance based measure of concept drift, assuming a relative increase or drop in learner performance is due to concept drift. This technique is used in trigger based approaches to counter concept drift where a learner will adapt by changing the size of its training window [ZK09, DM14, GvB+14] or adding base learners to an ensemble [BS14, BWPL14] when performance falls below a certain threshold. The problem with this type of approach is that the measurement of the drift is dependent on the learner whose performance is being tracked. Further, in practice, the problem with this type of approach is that it is *symptomatic*. This means the effect of the concept drift will only be measured or recognised once it hits a certain threshold, depending on the type of drift, the drift might have started, and thus might have been acted upon, well before it was detected. This is especially a problem if we have small datasets as can often happen in mobile context-aware applications.

We thus *decouple* the problem of measuring concept drift and measuring the need to forget data instances, ultimately offering a hands-on approach to detecting and managing concept drift suitable for mobile context-aware applications. This is because we are not actually concerned about the presence of *any* type of concept drift in a dataset, rather, *we are concerned about the presence of drifts that change class boundaries in a way that contradicts previous instance classifications*. Throughout a dataset, if the class boundaries change in a way that does not affect previous instance classifications, online retraining will suffice to adapt to the concept drift, while in the other cases, some data will have to be discarded to learn the correct boundary as illustrated in Figure 3.1.

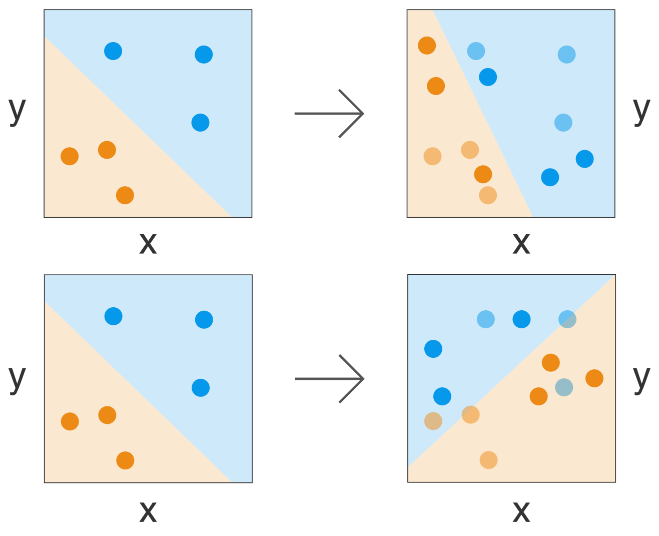


Figure 3.1: Two archetypical scenarios of how concept drift might affect a dataset as time passes (initial state on the left, later state on the right with points from the initial state faded out). The figures shows what the class boundaries might look like for each state using a discriminative learner. On top, an example where retraining a learner with all newly available training data will suffice to remedy concept drift. On the bottom, a case where class boundaries have changed in a way that affects previous instance classifications and where a forgetting strategy is potentially needed to learn the new boundary.

3.1.1 Drift Events

We thus propose to measure the amount of concept drift and the need to forget as two separate quantities. For this, we segment a dataset into $\boldsymbol{k}$ equally sized chunks which we call **epochs** and look for the occurrence of two types of events across successive epochs.

- **New data event:** an area (or neighbourhood) in the dataspace that is not empty in the current epoch but was empty in the previous epoch. Specifically, we say an area is empty if it contains less than $\boldsymbol{\theta_{empty}} * 100$ percent of the total number of datapoints in the set.
- **Class flip event:** an area in the dataspace that is not empty and had a class balance that noticeably changed compared to the previous epoch where it was also not empty. Specifically, we define the class balance $b^t_{c_i}$ of class c_i at epoch e_t as $b^t_{c_i} = \frac{|c^t_i|}{\sum_j |c^t_j|}$ where $|c^t_i|$ is the number of instances of class c_i in e_t. We then say an area's class balance noticeably changed if there is a class c_i such that the difference between class balances at epochs e_t and e_{t-1} $|b^t_{c_i} - b^{t-1}_{c_i}|$ is larger than $\boldsymbol{\theta_{change}}$ and the area was non-empty in e_t and e_{t-1}.

Class Flip Measure

We define the class flip measure as being the percentage of points in a dataset which have been affected by a class flip event defined by a specific θ_{change} over consecutive epochs. The number of points affected is obtained by summing the number of points in each affected area belonging to the latest epoch over consecutive epochs. The algorithm to compute the class flip measure of dataset is shown in Algorithm 1.

Concept Drift Measure

We define the concept drift measure as the sum of the class flip measure and the new data measure. The new data measure is the percentage of points in a dataset which are affected by a new data event defined by a specific θ_{empty} over consecutive epochs. The number of points affected is obtained by summing the number of points in each affected area belonging to the latest epoch over consecutive epochs. The algorithm to compute the concept drift measure of dataset is shown in Algorithm 1.

These drift events have a convenient interpretation in mobile context-aware systems. A new data event shows evidence of a new context being sensed, which could be due to a new behaviour pattern on the part of the user the data was collected over, for instance, a user receiving calls from a new contact in a disruptive smartphone notification use case scenario. A class flip event shows evidence of a change in an existing behaviour pattern as the class distribution changes in a specific area loosely corresponding to a context, for instance, a change in the preferred temperature for a given day and time in Thermostat Auto-scheduling use case scenario. Because we only consider class flip events over non-empty areas, the two events are mutually exclusive.

The measure of the prevalence of our two events depends on how we define an **area**. A straightforward way to do so, and the one we use in this thesis, is to partition the dataspace using a regular n-dimensional grid (where n is the number of features in the dataset) so that each grid element is an area. To make the grid, we partition each feature into $\boldsymbol{f}$ segments, resulting in a grid where each element is of equal n-dimensional volume. The number of grid elements, or areas, new data and class flip events are considered over for each epoch is thus f^n. A 3 dimensional example of this partitioning with $f = 3$ is shown in figure 3.2. In general, areas need not all have the same n-dimensional volume, if they are defined using clusters for instance.

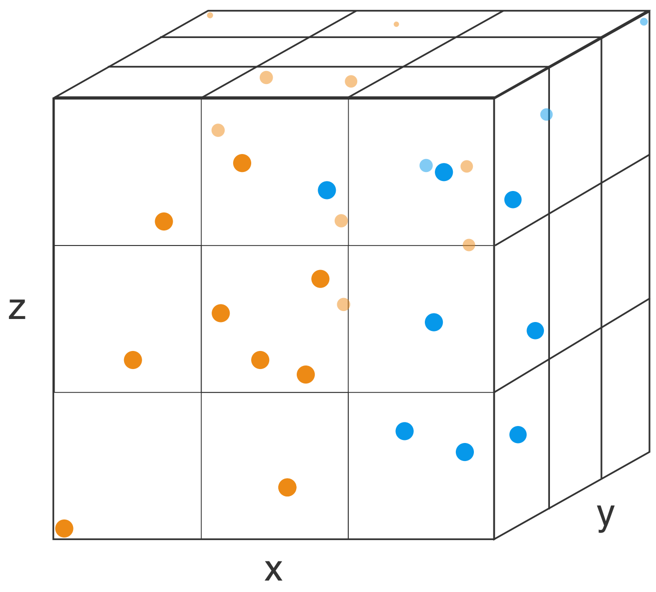


Figure 3.2: An illustration of a 3-dimensional dataset being partitioned into areas using a grid with $f = 3$, each feature is segmented into 3 equal parts yielding 9 different areas in which to measure new data and class flip events.

3.1.2 Concept Drift Analysis Algorithm (CDA)

We now present our proposed learner-independent concept drift measurement method. It quantifies the amount of concept drift in dataset d by assessing the prevalence of new data events and class flip events using parameters k (number of epochs), θ_{empty} (threshold for an area to be considered empty), θ_{change} (threshold for a change in an area's class balance to be considered a class flip) and f (feature partition parameter). Specifically our algorithm measures the amount of concept drift by estimating the percentage of datapoints affected by new data and class flip events as these correspond to changes in $p_X(X)$ and $p(X|C)$ respectively. The algorithm also outputs the percentage of datapoints affected by class flip events only, as these suggest the need to use a forgetting strategy in addition to online learning to maximise learner prediction performance. A toy example of the CDA algorithm is shown in Figure 3.3.

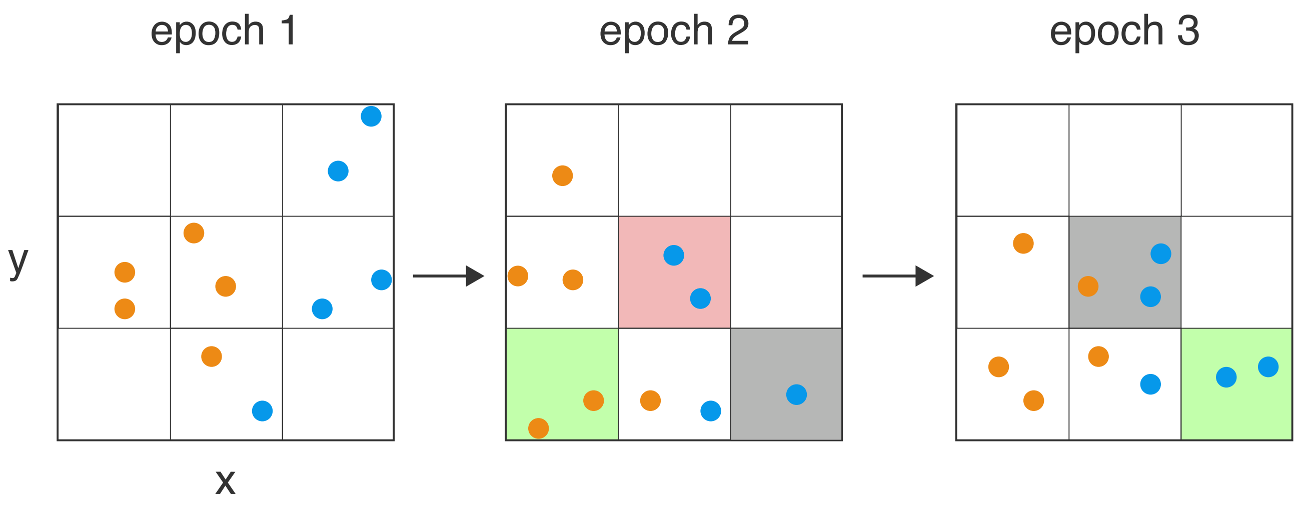


Figure 3.3: A toy example of the CDA algorithm over 3 epochs ($k = 3$) on a 30 point 2-dimensional dataset with $\theta_{empty} = 0.05$, $\theta_{change} = 0.5$ and feature partition parameter $f = 3$. CDA considers new data and class flip events between epoch 1 and 2, and epochs 2 and 3. At epoch 2, the class balance changes by 100% in the red area in favour of the blue class so that a class flip event is counted and the class flip measure is initialised to $\frac{2}{30}$, similarly the green area goes from empty to non-empty as it holds $\frac{2}{30} > \theta_{empty} = 0.05$ points in epoch 2 so that a new data event is counted and the new data measure initialised to $\frac{2}{30}$. On the contrary, in the grey area no new data event is counted as it did not hold any points in epoch 1 and is still considered empty in epoch 2 as it only holds 1 point and $\frac{1}{30} \leq \theta_{empty} = 0.05$. Between epoch 2 and 3, in the green area, a new data event is counted as it now holds $\frac{2}{30} > \theta_{empty} = 0.05$ points while it was considered empty in the previous epoch. The new data measure is then incremented by $\frac{2}{30}$. The grey area is not considered to have had a class flip event occur in it as the class balance went from 0:1 (orange class ratio : blue class ratio) to $\frac{1}{3}$:$\frac{2}{3}$ so that the change in class balance remains below θ_{change}. The output of CDA is the concept drift measure which is the sum of new data and class flip measures $3 * \frac{2}{30} = 20\%$ in this case, and the class flip measure $\frac{2}{30} \sim= 6.6\%$.

Algorithm 1 Top level concept drift analysis to measure concept drift. Returns the estimated percentage of points in d affected by concept drift and the percentage of points affected by class flips.

```
1: procedure ConceptDriftAnalysis($d$,$k$, $\theta_{empty}, \theta_{change}, f$)
2:     $q_{ND} \leftarrow 0$                                  ▷ Number of points linked to new data events
3:     $q_{CF} \leftarrow 0$                                  ▷ Number of points affected by class flip events
4:     $grid, centres \leftarrow$ MakeGrid($d, f$)            ▷ Initialise grid
5:     $l_{epoch} \leftarrow \left\lfloor \frac{|d|}{k} \right\rfloor$                ▷ We only consider full length epochs
6:     $d_0 \leftarrow d[0 : l_{epoch}]$                       ▷ Data for first epoch
7:     $grid \leftarrow$ Insert($grid, d_0, centres$)
8:     for $i = 1..k-1$ do                                    ▷ $i$ is the epoch index
9:         $grid_{previous} \leftarrow grid$                  ▷ Deep copy of current grid
10:        clear($grid$)                                      ▷ Clears data points held in the grid
11:        $d_i \leftarrow d[i * l_{epoch}, (i+1) * l_{epoch}]$   ▷ Data for the new grid
12:        $grid \leftarrow$ Insert($grid, d_i, centres$)
13:        $temp_{ND}, temp_{CF} \leftarrow$ CompareGrids($d, centres, grid, grid_{previous}, \theta_{empty}, \theta_{change}$)
14:        $q_{ND} \leftarrow temp_{ND} + q_{ND}$
15:        $q_{CF} \leftarrow temp_{CF} + q_{CF}$
16:    Return $\frac{q_{ND}+q_{CF}}{|d|}, \frac{q_{CF}}{|d|}$
```

Algorithm 2 Returns an regular n-dimensional grid and the set of grid element centres, where n is the number of features in data d

```
1: procedure MakeGrid($d, f$)
2:     Initialise $\vec{bounds}_{lower} \leftarrow$ (Min($d[:,0]$), Min($d[:,1]$), ..., Min($d[:,n]$))
3:                                                        ▷ A vector with minimum values of each feature
4:     Initialise $\vec{bounds}_{upper} \leftarrow$ (Max($d[:,0]$), Max($d[:,1]$), ..., Max($d[:,n]$))
5:                                                        ▷ A vector with maximum values of each feature
6:     $\vec{range} \leftarrow \vec{bounds}_{upper} - \vec{bounds}_{lower}$
7:     $\vec{l}_{partitions} \leftarrow \vec{range}/f$         ▷ element wise division by $f$
8:     $centres \leftarrow$ MakeGridHelper($n, f, \vec{bounds}_{lower}, \vec{l}_{partitions}$, [ [ ] ])
9:     for $\vec{centre}$ in $centres$ do
10:        Initialise $grid[\vec{centre}] \leftarrow$ [ ]
11:    Return $grid, centres$
```

Algorithm 3 Tail-recursive helper function that returns a set of grid element centres $\vec{centres}$ based on the minimum values of each feature $\vec{bounds}_{lower}$, feature partition lengths $\vec{l}_{partitions}$ and the number of partitions per feature f

```
1: procedure MakeGridHelper($n, f, \vec{bounds}_{lower}, \vec{l}_{partitions}, centres$)
2:     if $n = 0$ then                                  ▷ base case: we have gone through all features
3:         return $centres$
4:     $centres_{new} \leftarrow [\ ]$                                                   ▷ Initialise
5:     $temp_{coordinates} \leftarrow [\ ]$                                               ▷ Initialise
6:     for $j = 0..f-1$ do
7:         $temp_{coordinates} \leftarrow$ Append($temp_{coordinates}, \vec{bounds}_{lower}[0] + \frac{2j+1}{2} * \vec{l}_{partitions}[0]$)
8:                                          ▷ Use the centre of mass of grid elements as centres
9:     for $\vec{centre}$ in $centres$ do
10:        $temp_{centres} \leftarrow$ list($\vec{centre}$.append(coordinate)) for coordinate in $temp_{coordinates}$
11:                                    ▷ Each $\vec{centre}$ branches into $|temp_{coordinates}|$ new centres
12:        $centres_{new} \leftarrow centres_{new}$.append($temp_{centres}$)        ▷ $centres_{new}$ is a list of vectors
13:    $\vec{bounds}_{lower} \leftarrow \vec{bounds}_{lower}[1 : end]$                ▷ Discard the feature we just processed
14:    $\vec{l}_{partitions} \leftarrow \vec{l}_{partitions}[1 : end]$                ▷ Discard the feature we just processed
15:    Return MakeGridHelper($n-1, \vec{bounds}_{lower}, \vec{l}_{partitions}, centres_{new}$)
```

Algorithm 4 Inserts data d into $grid$ defined by $centres$

```
1: procedure Insert($grid, d, centres$)
2:     for $\vec{datum}$ in $d$ do
3:         $\vec{target} \leftarrow argmin_{centre}$Distance($\vec{datum}, centre$)
4:                                      ▷ Distance measure is arbitrary, we use Euclidian
5:         $grid[\vec{target}]$.Append($\vec{datum}$)
6:     Return $grid$
```

Algorithm 5 Returns the number of points linked to new data events q_{ND} and points affected by class flips events q_{CF}, in between the epochs corresponding to $grid_{previous}$ and $grid$, according to θ_{empty} and θ_{change} threshold parameters as defined in Section 3.1.1

```
1: procedure compareGrids($d, centres, grid, grid_{previous}, \theta_{empty}, \theta_{change}$)
2:     $q_{ND} \leftarrow 0$
3:     $q_{CF} \leftarrow 0$
4:     for $\vec{centre}$ in $centres$ do
5:         $q^{t-1} \leftarrow |grid_{previous}[\vec{centre}]|$     ▷ Number of points in grid element in previous epoch
6:         $q^{t} \leftarrow |grid[\vec{centre}]|$
7:         if $q^{t-1}/|d| < \theta_{empty}$ and $q^{t}/|d| \geq \theta_{empty}$ then
8:             $q_{ND} \leftarrow q^{t} + q_{ND}$
9:         else if $q^{t} \geq \theta_{empty}$ then
10:            $\vec{b^{t-1}} \leftarrow$ GetBalance($grid_{previous}[\vec{centre}]$)
11:                          ▷ Class balance $\vec{b^{t-1}}$ is $(\frac{|c_1^{t-1}|}{|c_1^{t-1}|+...+|c_n^{t-1}|}, ..., \frac{|c_n^{t-1}|}{|c_1^{t-1}|+...+|c_n^{t-1}|})$
12:            $\vec{b^{t}} \leftarrow$ GetBalance($grid[\vec{centre}]$)
13:            if $|b^{t}_{c_i} - b^{t-1}_{c_i}| > \theta_{change}$ for some class $c_i$ then
14:                $q_{CF} \leftarrow q^{t} + q_{CF}$
15:    Return $q_{ND}, q_{CF}$
```

Dataset name	Set Size	Num. features	Class balance	f param.	Grid Size	Drift measure	Class flip measure
Iris	150	4	0.33/0.67	4	256	47.3%	0.0%
Mushroom	8124	4	0.48/0.52	9	6561	13.4%	2.5%
SEA	60000	3	0.37/0.63	6	216	13.6%	13.6%
STAGGER	120	3	0.44/0.56	6	216	43.3%	29.1%
Hyperplane	1000	2	0.51/0.49	3	9	8.0%	8.0%

Table 3.1: 5 datasets we used our concept drift analysis algorithm on with results in the second half of the table

3.1.3 Example use of the CDA Algorithm

In real datasets, concept drift can only ever be suspected, as the underlying distribution, or data-generating process, is often unknown. In artificial datasets however, where concept drift is purposefully introduced, we have access to the concept drift ground truth so that it is highly likely that using a forgetting strategy will be advantageous compared to online retraining and offline learning. We report the results of our concept drift analysis algorithms on 5 different well-known datasets using $k = 3$ epochs which can be thought of as segmenting datasets into training, validation and testing sets and analysing the changes that occur throughout these learning phases. The dataset characteristics, computed concept drift measures (Drift Measure in the table) and class flip measures are summarised in Table 3.1 and are discussed in detail below.

- **Iris:** the famous Iris flower dataset introduced by Fisher [Fis36, fRAF36], often used in Hello World machine learning examples. It consists of $3 * 50$ instances that belong to one of 3 different types of Iris flower: Setosa, Virginica and Versicolor. For our experiments we merged the Virginica and Versicolor sets to have a binary class learning problem. The data has 4 features, sepal length, sepal width, petal length and petal width. This set is interesting because it presents a simple learning problem where maximum accuracy can easily be attained using offline learning i.e. the concept drift present in the set does not affect performance. Intuitively this is because Iris flower species are a stable concept, in a given location it is highly likely that all Irises have similar physical characteristics. While the drift measure is the highest (47.3%) of all considered datasets the class flip measure is the lowest (0.0%) confirming the expected outcome. This type of result occurs when each epoch is an incomplete representation of the full dataset but the full dataset does not have changing class boundaries that eventually contradict previous instance classifications. This situation is prone to happen in small datasets such as Iris.

- **Mushroom:** a popular UCI repository dataset [tNAM81]. It consists of 8124 instances that correspond to hypothetical samples of 23 different species of gilled mushrooms categorised into poisonous or edible. We restricted the original dataset to 4 features, odor, spore-print-color, population and habitat as these are sufficient to discriminate between the two classes with over 99% accuracy [Sch87]. As in the Iris dataset, the characteristics of the different types of mushrooms and their poisonousness is assumed to have stayed the same while the dataset was made, meaning any potential concept drift in the set should not affect performance. Our results show that, as expected, both the drift and class flip measures are low (13.4% and 2.5% respectively). We suspect the non-zero class flip measure is due to a high data density which also required the use of an unusually fine grid (6561 elements) before the values output by CDA converged.

- **SEA:** this is one of the most cited datasets in the concept drift literature [SK01, Pec06]. It is an artificial datasets with 60,000 3-dimensional randomly generated instances in $[0, 10]^3$. The instances are labelled as belonging to class 1 if the value of $feature1 + feature2 < \theta$ and class 0 otherwise, where θ is an arbitrary threshold. Concept drift is created by segmenting the data into 4 consecutive blocks (which we call epochs) and changing the threshold value θ between consecutive data blocks. Interestingly, the value of the drift measure and the class flip measure are the same (13.6%) meaning that there were no 'new' datapoints after the first epoch and that all concept drift was due to class boundaries contradicting previous point classifications. This makes intuitive sense as the datapoints were uniformly randomly generated, thus covering the whole space and it is only class assignments that changed throughout the data.

- **STAGGER:** this is another emblematic dataset in the concept drift literature [SG86, WK96, Kun07]. It is an artificial dataset with 120 instances generated over a 3-dimensional discrete feature space $\{small, medium, large\} \times \{red, green, blue\} \times \{square, circular, triangular\}$. Similarly to the SEA dataset, concept drift is introduced by generating instances in the space and changing the label of datapoints as time passes (every 40 instances). For example, points will be labelled as coming from class 1 if they carry the features *small* and *red* in the first 40 instances and 0 otherwise, while in the next 40 instances, points will be classified as coming from class 1 if they carry the features *green* or *circular*. The values of the drift and class flip measures are high (43.3% and 29.1% respectively) which conforms to our expectations as the epoch segmentation we use in our concept drift analysis matches the epochs used in the data generation process and classification rules after each sudden drift are very disconnected (most class 1 instances from the first epoch will change their class each 40 instances).

- **Hyperplane** this widely used concept drift dataset is made by uniformly randomly generating and labelling data in $[-1, 1]^2$ depending on which side of a hyperplane passing through the origin they fall [Kun07]. Concept drift is generated by gradually rotating the hyperplane about the origin, thereby changing the class of points close to the hyperplane. This datasets is in the same situation as the SEA dataset where the drift measure and class flip measure are the same (8.0%). The absolute value of the measures are relatively low, probably due to the class balance per grid element being either below or well above θ_{change} when considered at the epoch level. We predict a number of epochs much larger than 3, would show the class flip events even more clearly.

These experiments show that datasets crafted to have concept drift that affects learner performance have a higher class flip measure than the Iris and Mushroom datasets which are popularly considered as 'concept drift free' (in the sense that near perfect accuracy can be obtained with an offline learner). This epitomises the problem with the term concept drift and shows the importance of decoupling the notions of concept drift and the need to forget. In all datasets we observe a change in the underlying data distributions throughout the 3 epochs we consider (leading to non-zero drift measures) but only SEA, STAGGER and Hyperplane benefit from employing forgetting strategies. As our experiments later show, it is not the drift measure which is correlated with the need to forget but the class flip measure. We further confirm this link in Section 3.3 on the 8 datasets used to benchmark forgetting strategies.

3.1.4 Discussion and limitations

This section presents a simple way of measuring concept drift and estimating whether a forgetting strategy, such as windowed learning, is likely to improve performance compared to offline learning and online retraining on an arbitrary dataset. The proposed concept drift analysis algorithm CDA quantifies concept drift and the need to forget in a learner-independent way and with an intuitive interpretation, we thus believe it presents an essential tool for practitioners in the field of mobile context-aware applications as it provides a way to inspect data relating to an application and know which type learning setting might be appropriate for it.

The proposed methods has, however, certain limitations and areas in which it could be improved which we list below:

- **The method is not parameter free.** In practice, 4 parameters need to be set. k, the number of epochs we segment a dataset into; θ_{empty}, the threshold below which we consider a grid element to be empty; θ_{change}, the threshold over which we consider a class balance change to occur; and f, the number of segments we split features into. Setting different values for k, θ_{empty} and θ_{change} will change the estimated percentage of points considered to be affected by concept drift and class flips events. To best compare values across datasets, a solution would be to keep CDA parameters fixed and to normalise datasets so that each grid area is of equal n-dimensional volume. For all our experiments, we used $\theta_{empty} = 0.01$ and $\theta_{change} = 0.08$ as we worked with relatively small datasets, and chose f large enough so that the output for f and $f + 1$ converged to the same drift and class flip measure.
- **Time and space complexity**. The time and space complexities of CDA are both $O(f^n)$ where n is the number of features in a dataset and f the number of partitions each feature is segmented into. This exponential runtime restricts the use of our method to datasets

with a small number of features which we assume to be the case for the type of mobile context-aware applications we are interested in. That being said, the algorithm is meant to be a pre-processing or data analysis step and thus only needs to be done once for each dataset and can be done offline. Further, for large and sparse datasets, a different definition of the areas used by CDA could make CDA scale better.

- **Refinements** Our approach is novel in that it decouples concept drift and the need to forget (discard) some parts of a dataset as we argued the former does not always imply the latter. It makes the assumption that class flip events indicate that the learnt class boundary no longer corresponds to the optimal one and that in those cases a forgetting strategy needs to be employed to learn the new boundary. Datasets where class flip events occurs without the need to forget any data can easily be constructed but we have found that on our initial test datasets our measure could reliably be used to assess whether a forgetting strategy should be employed.

  It would be interesting to compare our proposed approach with other ways of measuring changes in data distributio. For instance, the Earth Mover's Distance [LB01] could be used instead of our CompareGrids() function (line 13 in Algorithm 1) if the grid was flattened into normalised 1-dimensional bins. The amount of concept drift could then be estimated by taking the EMD over all data points between two epochs and the need to forget by taking the weighted sum of EMDs for each class. This would contrast with our current approach as it would compute grids level statistic instead of computing statistics for each element in isolation and combining them.

3.2 Forgetting Strategies

Being able to detect the need to forget certain instances in a sequential learning setting naturally leads to the question of finding the *right forgetting strategy* for a given dataset.

Forgetting Strategy

A forgetting strategy is a dynamic training set formation algorithm achieved by selecting subsets of training instances for online learning. The datapoints from the overall available training set which are not used as part of the current training subset to make predictions are said to be forgotten. The most common forgetting strategy is windowed learning where the most recent n instances corresponding to the window size are selected regardless of the current testing point.

Intelligent systems have one fundamental property that must be considered when choosing a forgetting strategy. Not only must an application be able to adapt to sudden changes in user behaviour affecting a large number of instances thereby radically changing current class boundaries, but it must also be capable of handling isolated changes of existing habits and creation of new habits that only affect class boundaries *locally*. Going back to the Disruptive Smartphone Notification application, if a user changes his schedule, rules for the affected days will have to be re-learned (and also potentially other days collaterally) while visiting a new location might only affect future instances connected to that location without changing the current underlying rules.

We thus need forgetting strategies that do not assume anything about the nature of concept drifts and that are able to handle **local concept drift**. We also add the requirement that the forgetting strategies be *learner independent* as an application might require a specific type of learner, a computationally efficient black-box learner or a logic based learner that outputs human readable rules for instance.

Local concept drift is a very little studied field, notable exceptions being [DCTC05, TPCP06b, TPCP08, Zli09, Zli10]. It is usually only recognised implicitly when instance based learning is shown to better cope with concept drift than learners that use training windows. Email spam filtering is the prototypical example of this where new types of spam such as 'Win tickets to the Champions League' will suddenly appear and disappear independently of online casino spam, that is constantly being disseminated.

Local concept drift was first formally defined in Tsymbal et al. in [TPCP08] and our definition of it is a rephrasing of the original, relaxing the constraint that t and t' are consecutive time points.

Local Concept Drift

Local concept drift occurs between two time points t and t' if classification boundaries between the two times remain stable except in one area where the class boundary of t' contradicts previous instance classifications.

The notion of local concept drift is important because by and large most learner independent concept drift adaptation techniques (discussed in Section 2.2.3) assume that concept drift equally affects the entire data space. This translates into discarding of all old data points in the case of adaptive-size windowed learning and all outdated or under performing base learners in ensemble learning approaches. Omitting the possibility of local drift thus means infrequent patterns will periodically be unlearned as old data is indiscriminately discarded either directly or indirectly. In mobile context-aware applications this is *especially undesirable* as lesser frequent patterns might be the ones that have the highest cost when misclassified. In a Disruptive Smartphone Notifications application, if a user receives disruptive PPI spam[1] related text messages every 3 months they will want these correctly classified as such regardless of how many new contexts they experience in the mean time for instance.

To the best of our knowledge the only learner independent adaptation techniques that were designed to handle local concept drift are Dynamic Integration of Classifiers (DIOC) [TPCP06b, TPCP08] and the uniFied Instance Selection algoritHm (FISH) [Zli09, Zli10] which we now present before proposing our own strategies.

3.2.1 Existing strategies

The notion of forgetting strategy is never explicitly used in the literature although it forms the basis of most concept drift adaptation algorithms. We believe this way of approaching concept drift is useful in mobile context-aware applications as it intuitively links to the need to forget no-longer-accurate user behaviour patterns and also clarifies the amalgam between concept drift and the need to forget as explained in Section 3.1. While some forgetting strategies are straightforward: discard data older than the n-th most recent instance in the case of windowed

[1]Payment Protection Insurance

learning, some are more indirect: ignore data used to train a base learner not participating in the current prediction in the case of an ensemble.

Dynamic Integration of Classifiers (DIOC)

Dynamic integration of classifiers in an ensemble means that the voting weights of base classifiers are reevaluated for each test instance $\vec{x}_i$. In DIOC, the weights depend on the local performance of base classifiers on the k nearest neighbours of $\vec{x}_i$ in a validation set; it fits into the Fusion rules of the ensembles category of the taxonomy discussed in Section 2.2.3. In its prediction phase (Algorithm 7) local concept drift is thus accounted for by only using those base classifiers, and hence those training instances, which are thought to be locally optimal to predict $\vec{x}_i$. Algorithm 6 and 8 show what DIOC does during the training and update phases of sequential learning and all three algorithms are discussed below.

Algorithm 6 Dynamic Integration of Classifiers algorithm training algorithm

```
1: procedure DIOCTRAIN($trainingSet, chunkSize, baseClassifier$)
2:     Initialise $ensemble \leftarrow \{\}$
3:     $validationSet \leftarrow$ Queue($chunkSize$)          ▷ An empty queue of size $chunkSize$
4:     Divide $trainingSet$ into $chunks$ of size $chunkSize$          ▷ $chunks$ is a list of lists
5:     for $chunk$ in $chunks$ do
6:         $ensemble \leftarrow$ APPEND($ensemble, baseClassifier$.train($chunk$))
7:                                    ▷ Add new $baseClassifier$ instance to ensemble
8:     $validationSet \leftarrow chunks[end]$          ▷ the last chunk of data
```

Algorithm 7 Dynamic Integration of Classifiers algorithm prediction algorithm. Returns the predicted label for input $\vec{x}$

```
1: procedure DIOCPREDICT($\vec{x}, k$,NEARESTNEIGHBOURS(),GETPERFORMANCES(),VOTE())
2:     $neighbours \leftarrow$ NEARESTNEIGHBOURS($validationSet, \vec{x}, k$)
3:                                    ▷ $validationSet$ from DIOCTRAIN()
4:     $bCperfs \leftarrow$ GETPERFORMANCES($ensemble, neighbours$)
5:                                    ▷ $ensemble$ from DIOCTRAIN()
6:     $label \leftarrow$ VOTE($\vec{x}, bCperfs$)
7:     Return $label$
```

The DIOC training algorithm takes 3 parameters: $trainingSet$, the training set; $chunkSize$ the size of each base learner's training set; $baseClassifier$, a base classifier class. This algorithm is similar to any ensemble learner's training phase with the exception of the $validationSet$ which is used to find a test instance's nearest neighbours later on.

Algorithm 8 Dynamic Integration of Classifiers algorithm update algorithm called after a prediction is made

```
1: procedure DIOCUPDATE($\vec{x}$, $groundTruth$, $maxEnsembleSize$, DISCARDWEAKEST())
2:     $validationSet \leftarrow$ ADD($validationsSet$, $\vec{x}$, $groundTruth$)
3:                                         ▷ $validationSet$ from DIOCTRAIN()
4:     every $chunkSize$ iterations do
5:         $ensemble \leftarrow$ ADD($ensemble$, $baseClassifier$.train($validationSet$))
6:                                         ▷ $baseClassifier$ defined in DIOCTRAIN()
7:     if SIZE($ensemble$) $>$ $maxEnsembleSize$ then
8:         $ensemble \leftarrow$ DISCARDWEAKEST($ensemble$)
```

The DIOC prediction algorithm is where the approach differentiates itself from other ensemble learning techniques. It takes 5 parameters: $\vec{x}$ the test input to predict; k, the number of neighbours we assess base classifiers' performance on; nearestNeighbours(*point*,*k*), a function to be called on a queue returning the k nearest neighbours of *point* in it; getPerformances(*points*) a function to be called on an ensemble returning $bCPerfs$ a list of its base classifiers and their performance on *points*; vote(*point*,$bCPerfs$) a function that returns the predicted label of *point* based on $bCPerfs$ and the voting strategy it implements. The 3 crucial choices for a practitioner are:

- **A distance measure** for NEARESTNEIGHBOURS(). In mobile context-aware applications data is likely to have a mix of categorical and numerical features which makes the notion of distance between two instances unclear. In [TPCP08], Tsymbal et al. propose to use the Heterogeneous Euclidean/Overlap Metric (HEOM) from [WM97].

$$d_{HEOM}(\vec{a},\vec{b}) = \sqrt{\Sigma_{i=1}^{n}\ HEOM_i(\vec{a},\vec{b})^2} \tag{3.1}$$

$$HEOM_i(\vec{a},\vec{b}) = \begin{cases} 0 \text{ if feature } i \text{ is categorical and } a_i = b_i \\ 1 \text{ if feature } i \text{ is categorical and } a_i \neq b_i \\ \frac{|a_i - b_i|}{range_i} \text{ else} \end{cases} \tag{3.2}$$

where n is the number of features and $\vec{a_i}$ denotes the values of the i-th feature of $\vec{a}$

- **A performance measure** for GetPerformances(). The performance measure proposed by the authors to compute the weight w of a base learner over the k-neighbourhood of $\vec{x_i}$ is:

$$w(\vec{x_i}) = \Sigma_{j=1}^{k}\sigma(\vec{x_i},\vec{x_j})c(\vec{x_j})/\Sigma_{j=1}^{k}\sigma(\vec{x_i},\vec{x_j}) \tag{3.3}$$

$$c(\vec{x_j}) = \begin{cases} 1 \text{ if the learner's prediction was correct for } \vec{x_j} \\ -1 \text{ else} \end{cases}$$

$$\sigma(\vec{x_i},\vec{x_j}) = 1/d_{HEOM(\vec{x_i},\vec{x_j})} \tag{3.4}$$

  w thus takes into account the number of correct and incorrect prediction for a learner close to $\vec{x_i}$ with the $c(\vec{x_j})$ function and advantages learners that perform well in the immediate neighbourhood of $\vec{x_i}$.

- **A voting strategy** for Vote(). Tsymbal et al. compare 3 voting strategies. *Dynamic Selection:* select the base classifier with highest w to predict $\vec{x_i}$'s label. *Dynamic Voting:* sum the weights of base classifiers for each predicted class and predict the class with largest sum of weights. *Dynamic Voting with Selection*: do the same as Dynamic Voting but only take into account the weights that are over the median weight. We note that the most recent base classifier in an ensemble was trained with much of the *validationSet* used to compute it's local performance which might lead to a bias in favour of it.

The DIOC update algorithm takes 4 parameters: $\vec{x}$, a datapoint; *groundTruth* its label, *maxEnsembleSize* the maximum size of the ensemble; DiscardWeakest() a function to be called on an ensemble removing one of its base classifier according to some criteria. Tsymbal et al. discard the classifier with smallest w according on the current *validationSet*.

The authors report that Dynamic Selection performs the best, with DVS coming in close second, on the datasets they consider: SEA, Hyperplane (see Section 3.1.3) and a private antibiotic resistance dataset. In the rest of this thesis, when we refer to DIOC we speak of the above mentioned algorithm with Dynamic Selection as the VOTE() function in Algorithm 7.

Unified Instance Selection Algorithm (FISH)

The uniFied Instance Selection algoritHm is not a forgetting strategy per se as it never permanently discards data, rather, it dynamical forgets and 'unforgets' subsets of training instances to maximise prediction performance for each test instance $\vec{x_i}$. FISH has 3 variants FISH1, FISH2 and FISH3 and fits into the adaptive training set formation category of the taxonomy presented in Section 2.2.3. The idea behind FISH is to dynamically construct a fixed-sized training window to make a prediction for each $\vec{x_i}$. The training window is made by combining the training instances that are closest in time and closest in space to $\vec{x_i}$, thereby addressing the problem of local concept drift. Instances are selected based on the distance measure shown in equation 3.5 where α_t and α_s are used to trade off contemporariness d_t and spacial proximity d_s of points. For instance, if α_t and α_s are equal, points in the training window will be selected if they are somewhat contemporary with $\vec{x_i}$, if they are somewhat close in space according to a chosen distance function or if they are very close in time regardless of how far they are spatially and very close in space regardless of how old they are compared to $\vec{x_i}$.

$$D(\vec{a},\vec{b}) = \alpha_t d_t(\vec{a},\vec{b}) + \alpha_s d_s(\vec{a},\vec{b}),\ \alpha_t, \alpha_s \in [0,1],\ \alpha_t + \alpha_s = 1 \tag{3.5}$$

The variants of FISH differentiate themselves in their prediction function. The one for FISH1 is shown in Algorithm 9. FISH2 is identical to FISH1 except for an extra step where multiple training window sizes are tried so that lines 7 to 8 from Algorithm 10 become part of a loop over window size candidates. The window size with highest cross-validation performance is kept to make the final prediction. FISH3 extends FISH2 by adding yet another for loop where a range of α_t and α_s are tried equivalent to FISH2 becoming part of a loop over all weight candidates.

Because we choose not to recompute hyperparameters beyond an initial validation step for other

Algorithm 9 FISH1 training phase

```
1: procedure FISH1TRAIN($trainingSet$)
2:     $allTrainingData \leftarrow trainingSet$                              ▷ Initialise
```

Algorithm 10 FISH1 prediction phase. Returns the predicted label for input $\vec{x}$

```
1: procedure FISH1PREDICT($\vec{x}, winSize, \alpha_t, \alpha_s, baseClassifier$, DISTANCE$_t$(), DISTANCE$_s$())
2:     $distances \leftarrow [\,]$
3:     for $\vec{point}$ in $allTrainingData$ do
4:         $temp_{distance} \leftarrow \alpha_t$DISTANCE$_t(\vec{x}, \vec{point}) + \alpha_s$DISTANCE$_s(\vec{x}, \vec{point})$      ▷ Equation 3.5
5:         $distances \leftarrow$ APPEND$(distances, temp_{distance})$
6:     SORT($distances$)                        ▷ assume ordered by increasing distances
7:     $trainingData \leftarrow distances[0, winSize]$
8:     $baseClassifier$.train($trainingSet$)
9:     $label \leftarrow baseClassifier$.predict($\vec{x}$)
10:    Return $label$
```

forgetting strategies in our benchmarks of Section 3.4, we select FISH1 as the FISH forgetting strategy representative which we refer to as FISH from now on.

The FISH training algorithm takes 1 parameter: *trainingSet*, the training set which becomes the initial pool of points used to construct training windows.

The FISH prediction algorithm takes 5 parameters: $\vec{x}$ the test input to predict; *winSize*, the size of the dynamically formed training window; α_t the time distance weight from Equation 3.5; α_s the spacial distance weight from Equation 3.5; *baseClassifier* the classifier used to make predictions; DISTANCE$_t$($point_a$,$point_b$) a function that returns the time distance between 2 points; DISTANCE$_s$($point_a$,$point_b$) a function that returns the spacial distance between 2 points; In [Zli09, Zli10] Žliobaitė uses DISTANCE$_t = |i - j|$ as the distance in time between the i-th and j-th instances and chooses the euclidian distance for DISTANCE$_s$. We note that all distances are normalised to fall in $[0, 1]$ and make α_t and α_s comparable across datasets.

The FISH update algorithm takes 2 parameters: $\vec{x}$, a datapoint; *groundTruth* its label. The function simply adds these to the pool of available training data.

Algorithm 11 FISH1 update phase called after a prediction is made

```
1: procedure FISH1UPDATE($\vec{x}, groundTruth, maxEnsembleSize$,DISCARDWEAKEST())
2:     $allTrainingData \leftarrow$ ADD$(allTrainingData, \vec{x}, groundTruth)$
3:                                         ▷ $allTrainingData$ from FISH1TRAIN()
```

3.2.2 Proposed Strategies

Historically, most forgetting strategies are based on a single training window which forgets points in a time-linear FIFO fashion. We view these as **eager** forgetting strategies as they do not consider the possibility that some 'old' data might still be useful for future predictions and are indiscriminately discarded. The DIOC algorithm is interesting because it introduces the idea that distinct training windows might be optimal for specific areas of the input space, albeit indirectly, showing different base classifiers are optimal in different areas of the data space. FISH continues in this direction by introducing the idea that not only do different areas in the data space require different training instances but that these need not all have the same age i.e. 'old' instances can be relevant for future predictions if they are close enough in space to a test point. To the best of our knowledge, FISH was the first algorithm to explicitly combine spacial and time distances to dynamically form a unique training window for each test input.

While the edge cases of FISH have an intuitive interpretation: a heavy weight α_t for the time component makes FISH a classic windowed learning strategy, conversely, a heavy weight α_s for the spacial distance component turns FISH into a nearest-neighbour Case Based Reasoning strategy, the intermediate cases are more difficult to rationalise. In general, a FISH training window will include instances that are close in time but far away in input space from the current test point $\vec{x_i}$, instances that are close in space but of a very different age and points that are somewhat close in time and and somewhat close in space. In Žliobaitė' [Zli10], only 1 (the Vote dataset) out of the 6 datasets used to benchmark FISH had α weights that weren't heavily skewed either toward space or time, hinting there might be better ways to leverage both dimensions. We thus propose to combine time and space using the idea of **local forgetting**.

The idea of local forgetting is to only discard data if it can be replaced by newer instances covering the same concept. We propose 2 algorithms implementing this strategy, **Training Grid** and **Training Clusters**, which both spatially segment the data space and hold a fixed sized training window for each area. As new training points are made available to the learner in the update phase of sequential learning, they get assigned to a training window which might cause the oldest point from the window to be discarded if the window is at capacity. This allows for local changes in patterns while preserving the knowledge of less frequently updated areas and also accommodates total concept drift, in which case all areas of the data space would be updated within a short time interval.

The Training Grid and Training Clusters algorithms are part of the Adaptive training set formation category of the taxonomy presented in Section 2.2.3. They are blind, learner-independent adaptation techniques [GvB+14], meaning they are always active and always build training sets in the same way regardless of concept drift type or intensity, and can be used with any learner, addressing the requirements set in Section 3.2

Training Grid Algorithm (TG)

Our Training Grid algorithm segments the data space in the same way as our concept drift analysis algorithm from Section 3.1.2. It takes a parameter f to specify the number of segments each feature should be split into and uses the same MakeGrid() function to initialise the grid used to keep the local training windows. The core idea is to keep independent local training windows that get populated as new training instances are made available in the update phase of the sequential learning process. Each new training point is assigned to the unique grid element that has its centre closest to it, so that each training window contains increasingly recent points although these will not come from a time continuous chunk of the overall training data in general. The forgetting strategy is illustrated in Figure 3.4. The TG prediction phase shown in Algorithm 14 can be used in *two different ways*. TG can either be used to train a base classifier using the window belonging to the grid element the current test point $\vec{x_i}$ belongs to or using the union of points across all training windows. The former could be called local training reminiscent of Case Based Reasoning, while the latter can be used in inductive learning where a classifier generalises from locally updated training instances.

Algorithm 12 Training Grid training phase

```
1: procedure TGTrain($trainingSet, f, winSize$, Distance())
2:     $grid, centres \leftarrow$ makeGrid($trainingSet, f$)
3:                                  ▷ Initialise using Algorithm 2 from Section 3.1.2
4:     $grid \leftarrow$ TGInsert($grid, trainingSet, centres, winSize$, Distance())
```

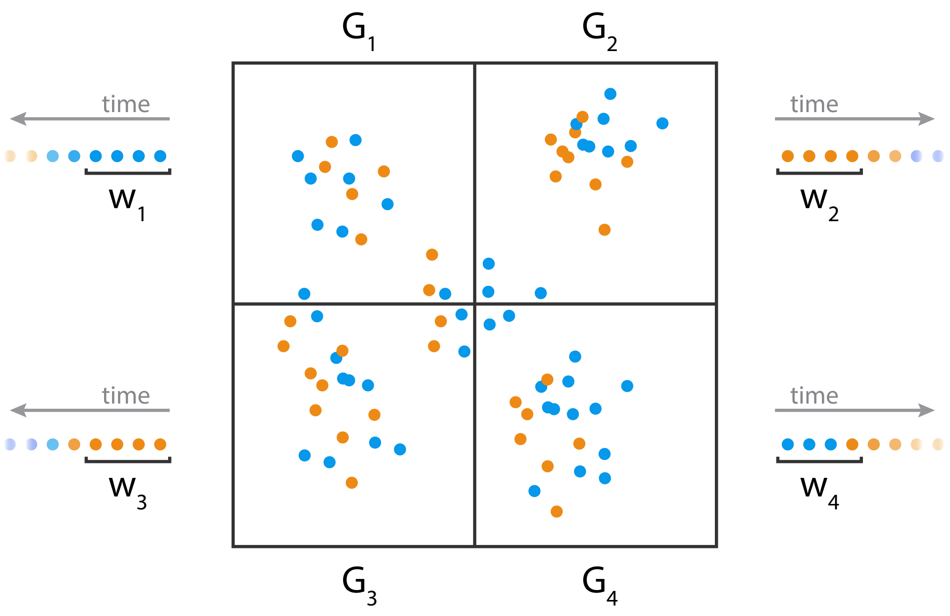


Figure 3.4: A 2-dimensional example of the Training Grid adaptive training set formation algorithm. G_i denotes grid elements that segment the data space and w_i denotes the corresponding training window each of size 4.

Algorithm 13 Inserts data d into the $grid$ the function was called upon

```
1: procedure TGINSERT($grid, d, centres, winSize,$ DISTANCE())
2:     for $\vec{datum}$ in $d$ do
3:         $\vec{target} \leftarrow argmin_{centre}$DISTANCE$(\vec{datum}, centre)$   ▷ Find closest grid element centre
4:         $grid[\vec{target}] \leftarrow$ APPEND$(grid[\vec{target}], \vec{datum})$
5:         if SIZE$(grid[\vec{target}]) > winSize$ then
6:             $grid[\vec{target}] \leftarrow$ REMOVEHEAD$(grid[\vec{target}])$
7:                                         ▷ Remove oldest element from local training window
    Return $grid$
```

Algorithm 14 Training Grid prediction phase. Returns the predicted label for input $\vec{x}$

```
1: procedure TGPREDICT($\vec{x}, trainLocal, baseClassifier$)
2:     if trainLocal then
3:         $\vec{target} \leftarrow argmin_{centre}$DISTANCE$(\vec{x}, centre)$
4:                 ▷ Find closest grid element centre, $centres$ and DISTANCE() from TGTRAIN()
5:         $trainingSet \leftarrow grid[\vec{target}]$
6:     else
7:         $trainingSet \leftarrow \cup_{centre} grid[\vec{centre}]$
8:                                         ▷ Gather all grid points, $centres$ from TGTRAIN()
9:     $baseClassifier$.train$(trainingSet)$
10:    $label \leftarrow baseClassifier$.predict$(\vec{x})$
11:    Return $label$
```

Algorithm 15 Training Grid update phase called after a prediction is made

```
1: procedure TGUpdate($\vec{x}$, $groundTruth$)
2:     $grid \leftarrow$ TGInsert($grid$, [$\vec{x}$, $groundTruth$], $centres$, $winSize$, Distance())
3:                                        ▷ $centres$, $winSize$ and Distance() from TGTrain()
```

The TG training algorithm TGTrain() takes 4 parameters: *trainingSet*, the training set ; f, the number of segments we split features into; *winSize*, the size of each grid element's training window; DISTANCE($point_a, point_b$), a distance function to compute distances from training and testing points to grid centres. This algorithm sets up the actual training grid and populates local training windows with the initial *trainingSet*. In our case we use the Euclidian distance.

The TG prediction TGPredict() algorithm takes 3 parameters: $\vec{x}$ the test input to predict; *trainLocal*, a boolean specifying whether the training set will be made of a single training window — the one attached to the grid element $\vec{x_i}$ belongs to — or the union of all training windows in case it is set to False; *baseClassifier* the classifier used to make predictions;

The TG update TGUpdate() algorithm takes 2 parameter: $\vec{x}$, a datapoint; *groundTruth* its label. The function simply inserts these into the training grid.

Training Clusters (TC) Algorithm

Our Training Clusters algorithm resembles the TG algorithm except in the way areas are defined in data space. Instead of having regularly spaced areas of identical n-dimensional volume, TC associates training windows to an arbitrary number of density-based clusters. The f parameter from TG is thus replaced by ϵ and θ_{pts}, the maximum cluster point distance threshold and the minimum cluster size of the DBSCAN algorithm [EKSX96] used to form clusters. The use of clusters is a natural solution to the data sparsity and fused pattern problems TG can face. If data is sparse, many of the grid elements TG creates, and later on loops over, will remain empty throughout learning. Using density-based clustering ensures we only focus on areas which are populated with data and do not create useless training windows. Further, it can be the case that two patterns find themselves in the the same grid element when using TG. This makes it impossible to update them independently from each other, while if the two patterns can be separated into instance clusters, TC overcomes this issue. The TC forgetting strategy is illustrated in Figure 3.5, and just like TG, the prediction phase of TC shown in Algorithm 18 can be done using a single local training window or the union of points available across all training windows.

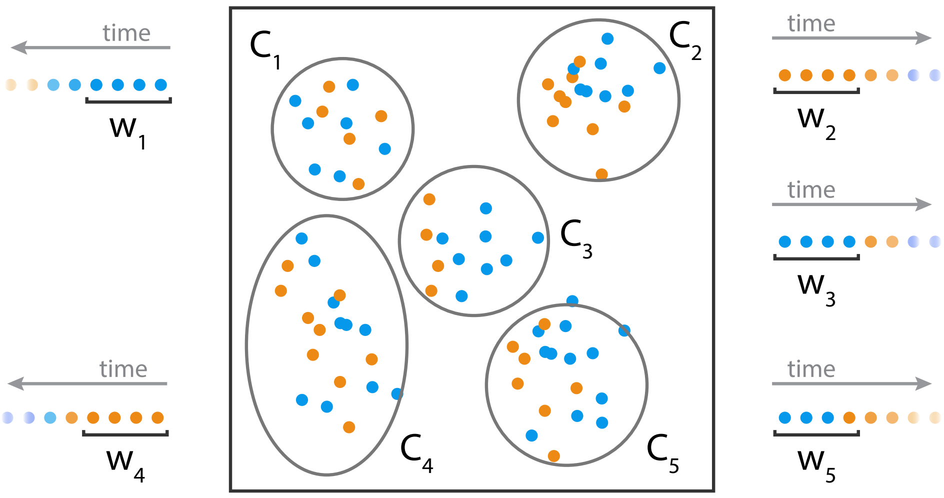


Figure 3.5: A 2-dimensional example of the Training Clusters adaptive training set formation algorithm. C_i denotes clusters that segment the data space and w_i denotes the corresponding training window.

Algorithm 16 Training Clusters training phase

```
1: procedure TCTrain($trainingSet, \epsilon, \theta_{pts}, winSize$, Distance())
2:     $centroids \leftarrow \{\}$                                        ▷ Initialise
3:     $clusters \leftarrow [\,]$                                         ▷ Initialise
4:     $trainingSetClustering \leftarrow$ DBSCAN($trainingSet, \epsilon, \theta_{pts}$)    ▷ Initialise
5:     for $clusterPts$ in $trainingSetClustering$ do
6:                                  ▷ $clusterPts$ holds the set of points for each cluster
7:         $temp_{centroid} \leftarrow$ Mean($clusterPts$)             ▷ Compute the cluster's centroid
8:         $centroids \leftarrow$ Add($centroids, temp_{centroid}$)
9:     $clusters \leftarrow$ TCInsert($clusters, trainingSet, centroids, winSize$, Distance())
```

The TC training algorithm TCTrain() takes 5 parameters: *trainingSet*, the training set ; ϵ, the maximum cluster point distance threshold for DBSCAN; θ_{pts}, the minimum cluster size for DBSCAN; *winSize*, the size of each cluster's training window; distance($point_a, point_b$), a distance function to compute distances from training and testing points to cluster centroids. In our case we use the Euclidian distance. This algorithm thus uses DBSCAN to compute clusters

Algorithm 17 Inserts data d into the training windows attached to the *clusters* the function was called upon

```
1: procedure TCInsert($clusters, d, centroids, winSize$, Distance())
2:     for $\vec{datum}$ in $d$ do
3:         $\vec{target} \leftarrow argmin_{centroid}$DISTANCE$(\vec{datum}, centroid)$
4:                                                          ▷ Find closest cluster centroid
5:         $clusters[\vec{target}] \leftarrow$ Append$(clusters[\vec{target}], \vec{datum})$
6:         if Size$(clusters[\vec{target}]) > winSize$ then
7:             $clusters[\vec{target}] \leftarrow$ RemoveHead$(clusters[\vec{target}])$
8:                                         ▷ Remove oldest element from local training window
9:     Return $clusters$
```

Algorithm 18 Training Clusters prediction phase. Returns the predicted label for input $\vec{x}$

```
1: procedure TCpredict($\vec{x}, trainLocal, baseClassifier$)
2:     if trainLocal then
3:         $\vec{target} \leftarrow argmin_{centroid}$DISTANCE$(\vec{x}, centroid)$
4:                 ▷ Find closest cluster centroid, $centroids$ and Distance() from TCtrain()
5:         $trainingSet \leftarrow clusters[\vec{target}]$
6:     else
7:         $trainingSet \leftarrow \cup_{centre} clusters[\vec{centre}]$
8:                                         ▷ gather all cluster points, $centres$ from TCtrain()
9:     $baseClassifier$.train$(trainingSet)$
10:    $label \leftarrow baseClassifier$.predict$(\vec{x})$
11:    Return $label$
```

Algorithm 19 Training Clusters update phase called after a prediction is made

```
1: procedure TCupdate($\vec{x}, groundTruth$)
2:     $clusters \leftarrow$ TCInsert$(clusters, [\vec{x}, groundTruth], centroids, winSize,$ Distance())
3:                                 ▷ $centroids, winSize$ and Distance() from TGtrain()
```

on the *trainingSet*. Although DBSCAN is not a centroid based algorithm, for simplicity, we assume minimum distance to a cluster's centre is equivalent to cluster membership which need not always be the case in practice.

The TC prediction TCpredict() algorithm takes 3 parameters: $\vec{x}$ the test input to predict; *trainLocal*, a boolean specifying whether the training set will be made of a single training window associated with the cluster $\vec{x_i}$ belongs to or the union of all training windows in case it is set to False; $baseClassifier$ the classifier used to make predictions;

The TG update TCupdate() algorithm takes 2 parameter: $\vec{x}$, a datapoint; *groundTruth* its label. The function simply inserts these into the correct cluster training window.

3.2.3 Discussion and Limitations

This section gathers 4 forgetting strategies which each have their own intuition. From a training set formation perspective, DIOC makes predictions either by gathering a chunk of time continuous instances leading to the the highest validation performance in the neighbourhood of the current test point $\vec{x_i}$ or by weighting different sets of time continuous instances based on their associated local performance, depending on the chosen ensemble voting strategy. FISH gathers training instances that are contemporary with and close in space to $\vec{x_i}$ while TG and TC can be used in two different ways. Either locally, by making predictions based on the most recent instances in the neighbourhood of $\vec{x_i}$, or globally, by gathering the most recent training instances in all areas of the data space, the only difference being in the way areas are defined in TC and TG.

The limitations of DIOC come from the fact it is an ensemble learning method. It is computationally expensive and data-hungry so that it is more adapted to stream learning. Its underlying forgetting strategy is also less flexible than the other 3 strategies as the time continuous chunks of training data it uses to train base classifiers are mutually exclusive. If only half the instances of a training chunk are 'outdated' and need to be forgotten (discarded), the other 'healthy' half will be forgotten at the same time when a base classifier is replaced.

FISH offers a more flexible approach as it considers instances one-by-one when making a training set. Further, by setting hyperparameters accordingly, FISH can be shifted toward acting as a classic training window forgetting strategy or a nearest neighbour CBR algorithm. It is important that more complex forgetting strategies also include simple existing strategies as a special case of their expression as these are sometimes optimal as we show in Section 3.4. The limitation of FISH appeared in our experiments where many times the best optimal hyperparameters were the ones that made it into a classic windowed learner. FISH also never permanently discards datapoints so that points that become noisy during the testing phase might still be used to make predictions if they are close enough to the current test point $\vec{x_i}$.

The TG and TC algorithms were designed to keep the flexibility of FISH but change the way time and distance are combined and allow local predictions like DIOC. They also have the desirable property that online retraining, window learning and nearest neighbour CBR are special instances of their expression. Online retraining is achieved by using local or global training with infinite size training windows. Windowed learning is achieved using a single area (grid element or cluster) regardless of local or global training and nearest neighbour CBR is achieved using local training.

The limitations of TG and TC are computational, each time a new training instance is available the base learner must be fully retrained, and a large grid size in TG or number of clusters in TC also leads to slow computation times. In the case of mobile context-aware applications, this is less of a problem as we aim to be data-efficient rather than computation-efficient because of our assumption that new ground truth data is only infrequently available due to the high cost of labelling. In practice, because of the extra parameter needed by TC to segment data space into clusters validation will take longer than for TG. This is then made up for during learning where TG can be slower due to the number of empty grid elements it has to loop over that do not exist in the cluster based data partition.

3.2.4 Potential Improvements and Future Directions

The proposed TG and TC algorithms are the simplest version of an entire family of algorithms leveraging the idea of local forgetting. Similar to the relationship between FISH1, FISH2 and FISH3, TG and TC could be refined by allowing periodic retuning of hyperparameters. For instance, the local training window sizes could be recomputed every so often by validation on recently acquired training data. In fact, training windows need not all be of equal size they could also be proportional to the number of points in the associated area.

While one can think of refining the TG and TC forgetting strategies, other forgetting strategies could also be developed. For instance, given a function that computes the area an instance covers, a forgetting strategy could be to formulate a set cover instance problem and use the cover with the least aggregate instance age as a training set.

3.3 Evaluating the Performance of Learners in Mobile Context-aware Applications

nThe final step before we can compare forgetting strategies is to choose a performance measure pertinent to mobile context-aware applications. Accuracy and the related recall, precision and f-measures are perhaps the most popular performance measures in the context-aware systems literature but we argue that they are not adapted to the field. In mobile context-aware applications, there is often a dominant class leading to unbalanced datasets. In the case of Disruptive Smartphone Notifications we've experimentally confirmed that a person typically receives many more wanted calls than unwanted calls for instance. The problem with using recall and precision related measures in an unbalanced dataset is that high values can be achieved by always predicting the class recall and precision are considered over or always predicting the majority class in the case of accuracy.

Performance measures that alleviate this problem such as the the Matthews correlation coefficient [Mat75] have been developed to this end, but most do no take into account the asymmetric costs of misclassifying instances of different classes. Continuing with our example of disruptive smartphone notifications, it has been shown that users are dissimilar in the way they rate their interruptibility [FS11, FHA+05, KS06, KC05, RDV11] so that, for instance, some users will associate a high cost for being disturbed by an unwanted incoming call even though it is the minority class [KH07].

Dataset unbalance and asymmetric cost of misclassification thus have to be accounted for in the target performance measure. While it is tempting to use the Area Under the Receiver Operating Characteristic curve (AUC) for this, it has recently been shown to be an incoherent performance measure because its value depends on the classifier itself and thus cannot be soundly compared across learners [Han09]. The proposed coherent replacement, the H measure, requires class misclassification costs to be specified via a beta distribution, which makes it difficult to use from an HCI perspective. On the grounds of simplicity, we propose to use a weighted accuracy measure A_w which encompasses class imbalance and user preferences in a single parameter w.

3.3.1 Weighted Accuracy Measure

In binary classification problems, our weighted accuracy measure A_w gives a way to express the preference, or importance, of the positive class c^+ being correctly predicted compared to the negative class c^- based on the parameter w. Formally A_w is defined in Equation 3.6

$$w = \frac{c^+ \text{ weight}}{c^- \text{ weight}} \in [0, \infty)$$

$$\begin{aligned} A_w &= \frac{w * TP + TN}{w * (TP + FN) + TN + FP} \\ &= \frac{w * recall * |c^+| + specificity * |c^-|}{w * |c^+| + |c^-|} \end{aligned} \tag{3.6}$$

Where TP, TN are the number of correctly predicted c^+, c^- instances respectively and FN, FP

the number of incorrectly predicted c^+, c^- instances respectively. We constrain the c^+ weight to be in $[0, \infty)$ and the c^- weight to be in $(0, \infty)$. $|c^+|$ and $|c^-|$ denote the number of instances in each class. Intuitively, A_w artificially grows or diminishes the number of instances from the positive class c^+ proportionally to w while keeping the recall, or performance on class c^+, fixed before computing the standard accuracy measure thereby emphasising or playing down the importance of correctly predicting the c^+ class. Going back to the example of disruptive smartphone notifications from Section 1.1, setting c^+ to be non-disruptive notifications and c^- to be disruptive ones, $w > 1$ would apply to individuals that prefer to be interrupted by some unimportant notifications to avoid missing important ones while $w < 1$ indicates that a user prefers to miss some important notifications instead of accidentally being interrupted by unimportant ones. The weighted accuracy measure has the nice property that setting $w = 1$ yields the well-known accuracy measure that weights performance on both classes equally.

3.3.2 Discussion

The immediate consequence of using a cost sensitive performance measure is the need to use cost sensitive learners so that their internal objective function is coherent with the function their performance is being judged by. While some learners can naturally be made cost sensitive (such as SVMs by setting the misclassification penalty for each class accordingly), in others, cost-sensitivity is less straightforward like in a Logistic Regression classifier. In the latter case, undersampling of a low weight majority class or oversampling of high weight minority class can be used to make any classifier cost sensitive.

While weighted accuracy gives us a mechanism to take into account different user preferences, an important question remains: how can we know a w for a specific user? This is an HCI issue and somewhat outside the scope of this thesis. However, it poses an even greater one: is it at all possible to reduce an individual's experienced performance into a single number i.e. what does 80% accuracy feel like, is it enough? Paul Lukowicz once commented that "in HCI it's not the number of mistakes that counts, it's about the acceptance of them by the user. If a smartphone app fails to react in the way I want it to in exceptional situations I might tolerate the mistakes, while if it repeatedly fails on a common task I might feel it doesn't work"[2]. One can argue, the ultimate true, but indirect, performance measure for a mobile context-aware application is the percentage of users that keep on using the application after being introduced to it.

3.4 Experiments

In the previous sections we presented a method to quantify a dataset's concept drift and need to forget ageing instances, a set of 4 forgetting strategies to tackle the problem of local concept drift and defined weighted accuracy as a suitable performance measure for such applications. We are now ready to run experiments and benchmark the strategies.

To verify the hypothesis that local forgetting is necessary in mobile context-aware applications we must move away from artificial datasets that are usually used in concept drift research. Longitudinal real datasets are rare because of the time it takes to gather the data and the cost of associated with manually labelling it — this issue is further discussed in Chapter 4. The 8 datasets we run experiments on in this chapter were chosen either because they are related to mobile context-aware applications, present interesting concept drift properties or because they have previously been used in concept drift research with published results we can easily compare our methods to.

This section studies the evolution of a single A_w performance measure for each dataset using a moving average performance window to understand the relative strengths and weaknesses of forgetting strategies when faced with concept drift as time goes by. Later, Chapter 4 studies each strategy's performance in different user preference scenarios by benchmarking them over a range of w parameters for a given dataset.

[2]Private communication

Set name	Set Size	Time span	Num. feats.	Class balance	Data type	f param.	Grid Size	Drift measure	Class flip measure
Olympics	8619	12y	1	0.56/0.44	real	10	10	6.4%	6.4%
Artificial6	400	N/A	2	0.50/0.50	artif.	9	81	47.5%	31.7%
Ž-elec	2956	3m	7	0.57/0.43	real	4	16389	51.2%	26.3%
Ž-luxem	1901	5y	9	0.51/0.49	real	2	512	5.8%	0.0%
Twitter-activ	2735	7y3m	2	0.57/0.43	real	9	81	57.1%	57.1%
Twitter-times	4722	6y4m	4	0.43/0.57	real	5	625	64.3%	6.3%
Sleep-dur	289	9.5m	3	0.51/0.49	real	8	512	64.7 %	14.1%
Sleep-time	867	9.5m	3	0.27/0.73	real	8	512	64.1%	13.2%

Table 3.2: Summary of the 8 datasets used in our experiments. Concept drift analysis (right-hand side of table) was made as described in Section 3.1

3.4.1 Datasets

We now present and explain our choice of datasets used to initially benchmark forgetting strategies. Their properties are summarised in Table 3.2.

- **Olympics:** this is the first dataset we use to present and discuss our experiments. The dataset is assembled from real raw data and will be used to exemplify how local concept drift might manifest itself in certain real datasets. It has 8619 single dimensional instances representing the age of athletes in all Olympic games from 2000 to 2012, it is freely available [Ref15]. The aim is to predict whether an athlete's age is under (class c^+) or over (class c^-) the mean age of athletes for that year's Olympic game. Concept drift is known to occur because the year of the competition is not included in the feature set and is the hidden context that produces the concept drift. The dataset was chosen as it is a longitudinal real dataset, known to have concept drift that requires certain instances to be discarded at various times during learning.

- **Artificial6:** this is the second dataset we use for a detailed discussion of experiment results. The dataset is artificial (the only artificial dataset we use in this thesis) and has 400 2−dimensional instances. The aim is to predict whether points belong to class c^+ (coloured in orange) or class c^- (coloured in blue) in $[-2, 2]^2$. Concept drift is known to occur as we had total control over the data generation process. We created this set to show the effects of patterns locally and independently drifting in a dataset and highlight the shortcoming of existing forgetting strategies in such cases.

- **Ž-elec:** this is the same dataset used by Žliobaite in her thesis. It has 2956 instances gathered between May and July 1997 (subset of a larger set of 45,312 instances gathered over 2 years [HW99]). The aim is to predict whether eletricity prices have gone up (class c^-) or down (class c^+) compared to the average price for the last 24 hours in a region of Australia. There are 6 features: *day of the week, time, demand in region, price in neighbouring region, demand in neighbouring region* and *scheduled energy transfer between regions*. Concept drift is suspected to occur because electricity prices tend to change as time goes by. The dataset was chosen because it has been used in approximately 40 other published works and has recently shown to have a very simple optimal forgetting strategy: an A_1 of $\sim$85% can be obtained by using a training window of size 2 because consecutive instances are autocorrelated [Zli13].

- **Ž-luxem:** this is also a set used by Žliobaite in her thesis. It has 1901 instances gathered between 2002 and 2007 gathered from a European Social Survey [JtCCT07, Zli09]. The aim is to predict whether a person is a heavy (class c^+) or low (class c^-) internet user. Originally there are 20 features, but we restrict ourselves to 9 features that can still be used to achieve an A_1 close to 100% (for computation time reasons): *country, watches TV?, average num. hours TV watched per week, reads newspapers?, average num. hours newspapers read per week, trust in the European Parliament, trust in the United Nations, signed a petition in the last 12 months?* and *average number of social meetings with friends per week*. Concept drift is suspected to occur because Internet penetration increased between 2002 and 2007 and usage patterns are thus also suspected to have changed. The dataset was chosen because our concept drift analysis shows that there is indeed concept drift but no class flip events, meaning that no forgetting strategy should be needed and online retraining should be sufficient to reach high performance.

- **Twitter-activ:** this set was made using the Twitter feed, available through the Twitter API, of Noah Glass one of the Twitter founders [Gla15]. It has 2735 instances and spans over 7 years. Each instance corresponds to a day and the aim is to predict whether a given day will be an active tweet day (class c^-) or not (class c^+), we define an active tweet day to be a day where a user created a number of tweets greater than the daily average for the month it belongs to. There are 2 features: *month* and *day*. Concept drift is suspected to occur because users can potentially change their Twitter habits throughout the years. The dataset was chosen because it is a recorded human behaviour over a long period of time and Twitter activity could easily be part of a mobile context-aware application.

- **Twitter-times:** this set was gathered in the same way as Twitter-activ but over a different Twitter founder, Christopher Stone [Sto15]. It has 4722 instances made by splitting each day in the 6+ years it spans into morning (before 12pm) and afternoon (12pm onwards) periods. The aim is to predict whether the user will tweet (class c^-) or not (class c^+) for a given day and period. There are 4 features: *year, month, day* and *period*. Concept drift is suspected to occur for the same reason as in Twitter-activ. The set was chosen because it is an example of total concept drift so that windowed learning should be the best forgetting strategy.

- **Sleep-dur:** this set was made using the HedgeHog open sensor platform [fTD15] in the context of sleep tracking research. The data was gathered over a period of 9 and half months using a watch case for the sensor that was constantly worn by a subject [BVL12, ds15]. In this learning task, the data is separated into days yielding 289 instances and the aim is to predict whether a person will sleep in (class c^-) or not (class c^+) (sleep longer than their mean sleep time over the whole dataset or not). There are 3 features: *month, day of the week* and *bedtime for that day*. Concept drift is suspected to occur because of the natural variability in a person's sleep patterns. This set was chosen because it could easily be part of a sleep tracking mobile context-aware application.

- **Sleep-time:** this set was made from the same raw data as Sleep-dur. It has 867 instances made by splitting each day into 3 time blocks. The aim is to predict whether a person is sleeping (class c^+) or awake (class c^-) at the time corresponding to the instance's time block: 9pm, 10pm or 11pm, i.e. while in Sleep-dur we try to predict if a person will sleep a lot on a given day, in Sleep-time we try to predict when they will fall asleep. There are 3 features: *month, day of the week* and *time block*. Concept drift is suspected to occur for the same reason as in Sleep-dur and the set was chosen for the same reason.

We now go through the implementation and experimental setup details, with an in-depth explanation of the Olympics and Artificial6 experiments to familiarise the reader with the experimental approach used for all the remaining experiments in this thesis.

3.4.2 Experimental Setup and Implementation Details

As most datasets we use are reasonably well balanced, forgetting strategies were benchmarked using A_1, except Sleep-time, which was benchmarked using A_2 because of its class balance ratio $\frac{0.73}{0.27} \sim 2.7$. In order to compare forgetting strategies equitably, they were implemented as meta-learners that all used the same base classifier, a cost-sensitive Logistic Regression classifier. Unless specified otherwise, learners were trained on the first third of datasets and tested on the two remaining thirds using the sequential learning process described in Section 2.2.1 where, after an initial call to the train function, the predict and update function of strategies are called one after another for all test points. Performance is then visualised by plotting the A_w performance's moving average over the last n points showing their evolution over time.

For each experiment we benchmark 6 forgetting strategies, if present, hyperparameters were set experimentally by iterating over a range of candidates and keeping the best ones:

- **Offline:** this is equivalent to the constant forgetting strategy where regardless of any new data available past the training set, no updates are made to the learner. This is the control for our experiments showing what would happen if we took no adaptive action at all.
- **Online retraining:** this is equivalent to the no-forgetting strategy where all new data available is used by the base learner. It shows the necessity, and sometimes the lack of necessity, in using a forgetting strategy when faced with concept drift.

- **Window:** this is the age-based forgetting strategy where all new data is discarded except the n most recent instances. It is, or forms the basis of, the default answer to concept drift in the literature and is used as a performance baseline.

- **DIOC:** this the local-performance-based forgetting strategy of Section 3.2.1. Its $chunkSize$ and k parameters are noted **CS** and **k** in figures.

- **FISH:** this the local age-based forgetting strategy of Section 3.2.1. The α_t and $winsize$ parameters are noted **a_t** and **WS** in figures, by property $\alpha_s = 1 - \alpha_t$.

- **TG:** this is the local age-based forgetting strategy of Section 3.2.2. The f, $winSize$ and $trainLocal$ parameters are noted **f**, **WS** and **TL** (T for True F for False) in figures.

- **TC:** this is the local age-based forgetting strategy of Section 3.2.2. The ϵ, minimum cluster size θ_{pts}, $winSize$ and $trainLocal$ parameters are noted **e**, **MCS**, **WS** and **TL** (T for True F for False) in figures.

Each forgetting strategy was made into its own class and experiments were coded in Python making use of the open source sci-kit learn package [sklc15a] when possible, namely for the base learner implementations and the DBSCAN algorithm. Cost sensitivity was achieved by oversampling class c^+ when $w > 1$ until the number of class c^+ instances was w times the number of class c^- instance, and when $w < 1$ by oversampling class c^- until the number of class c^- instances was $\frac{1}{w}$ times the number of class c^+ instances. Plots were generated using matplotlib [mdt15] and all experiment code and datasets are available from the thesis author's website [Smi15].

The Olympics Dataset

The Olympics dataset presents a seemingly simple task, given an athlete's age, label them as younger or older than the average athlete age of the Olympic game they participated in. The difficulty comes from the hidden (unavailable) context feature of the Olympic game year. Assuming an athlete is classified as belonging to class c^+ if the number of full years he has lived

Year	Season	Avg. athlete age
2000	Summer	26.2
2002	Winter	27.1
2004	Summer	26.3
2006	Winter	27.4
2008	Summer	26.2
2010	Winter	26.8
2012	Summer	26.1

Table 3.3: The average age in years of athletes for Olympic games between 2000 and 2012. From 2002 to 2006 Winter Olympics participants are on average about 1 year older than Summer games participants, while starting 2008 the average is below 27 years for all games.

is *less than or equal* to the average athlete age of the Olympic game he participated in and class c^- otherwise. Table 3.3 shows why local drift will occur at the age value of 27. Because instances are presented to learners in time increasing order, initially all athletes aged 27 years old belong to class c^- (for the year 2000), then, as instances from the 2002 Olympics games are presented to the learners, the ground truth for athletes aged 27 years old changes to class c^+ with all other class assignments remaining valid, hence the *local* concept drift. Referring to Table 3.2 we see that the amount of concept drift is equal to the amount of class flip events. This is because all age input values are present in the initial training set, so that all drift is due to the input value 27 changing class throughout the different Olympic games, hinting at the necessity to use a forgetting strategy to attain and maintain high performance.

Figure 3.6 shows how each forgetting strategy reacts to the drift and performance is summarised in Table 3.4. Note that the initial training set is composed of the instances corresponding to the 2000 Olympic games only (1436 instances). The moving average A_1 is taken over a window of size 200.

The best hyperparameters for TC are $\epsilon = 0.05$, a minimum cluster size of 2, a training window size of 750 and using training windows locally. These hyperparameters yielded 29 clusters, meaning the data space is segmented into 29 areas and each area holds the latest 750 training instances corresponding to it. Predictions are thus made based on the 750 latest training instances of the area a test point belongs to. The best hyperparameters for TG are $f = 40$, a training window size of 1 and using training windows locally. This means the data space is segmented into 40 grid elements and each element contains the latest corresponding instance. Predictions are thus made based on the class of that single element, similar to a Case Based Reasoning approach. The best hyperparameters for FISH are $\alpha_s = 1$, $\alpha_t = 0$ and a training window size of 100. This means predictions are made based on the 100 closest points in space in the overall training set, again similar to a Case Based Reasoning approach. The best hyperparameters for DIOC are $k = 3$ and a training chunk size of 250 for each base learner. Predictions are thus made by choosing the learner, and corresponding 250 training instances, which performs best on the 3 nearest neighbours of a test point in the validation set. The best window size for baseline windowed learning was found to be 200.

Intuitively, the difficulty for learners in this dataset arises in years 2002, 2004 and 2006 leading to the dips in performance observed in Figure 3.6. This is because in the initial training set the age value 27 belongs to class c^-. Due to concept drift in the year 2002, it changes to class c^+. In 2004, it returns back to its original class asignment. It flips to c^+ in 2006 before finally returning to c^- from 2008 onwards. The class assignment for value 27 thus has to be relearned throughout the datasets and the results show that TG and TC in their local training mode perform best (with A_1 over 0.99) followed closely by FISH (with $A_1 = 0.9733$). This is due to each technique quickly replacing obsolete labels for input value 27 as explained above, and shows that a local forgetting strategy is optimal for the Olympics dataset. DIOC comes in fourth (with $A_1 = 0.8691$) because it is unable to do instance selection and has to select instance chunks instead. Because it has lesser fine grain control over training window formation it is affected by inconsistent training chunks that have some obsolete labels for input value 27 and some current, valid, labels. As expected, windowed learning performs better than online retraining (with $A_1 = 0.8023$) because it also integrates newly available training points but discards older ones, which in this case resolves the inconsistencies over the label of input value 27. Online retraining on the other hand, never discards instances so that it is capable of *some* adaptation but eventually gets confused by contradictory labels for input value 27. Last, offline learning performs the worse as it keeps the $27 \rightarrow$ class c^- mapping that it learnt from the training set throughout testing.

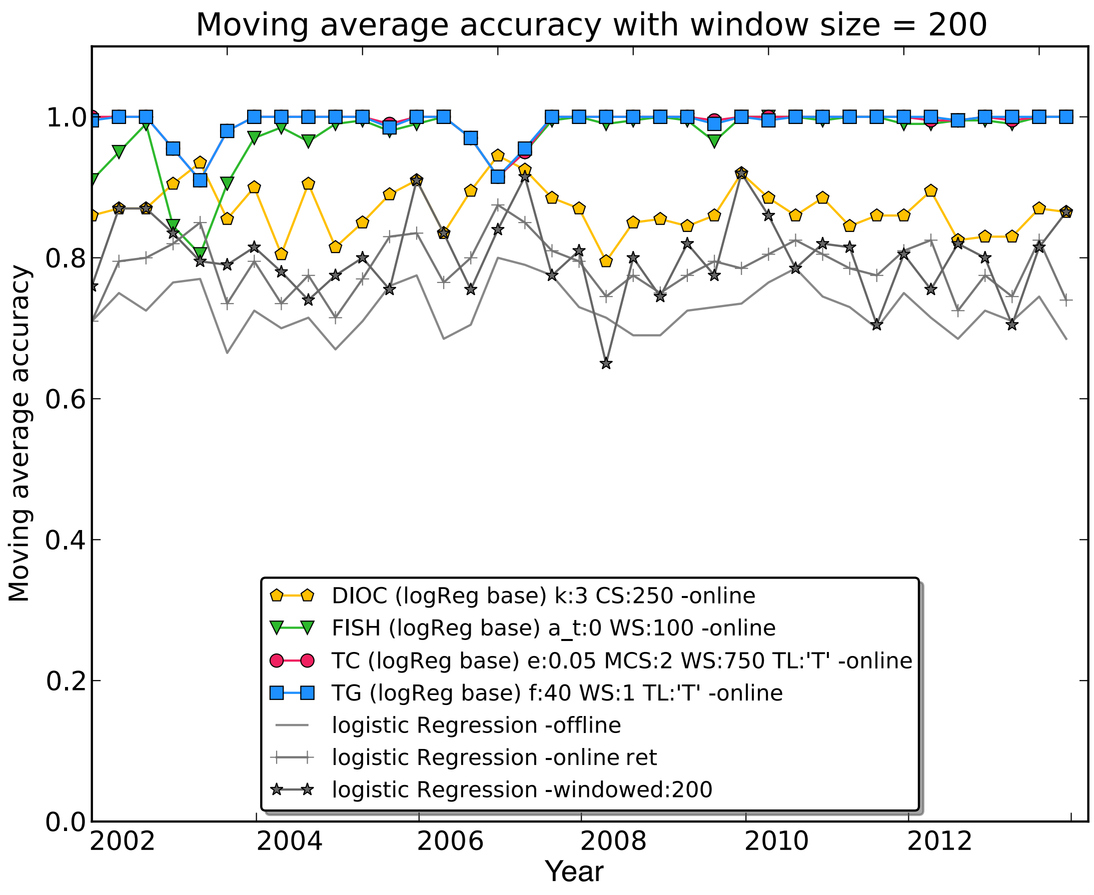


Figure 3.6: Evolution of forgetting strategy performance throughout the Olympics testing set. Moving average performance computed using a window of size 200.

Strategy	Recall c^+	Precision c^+	Recall c^-	Precision c^-	A_1
TC	0.9846	0.9983	0.9979	0.9814	0.9906
TG	0.9846	0.9980	0.9976	0.9814	0.9904
FISH	0.9699	0.9814	0.9775	0.9636	0.9733
DIOC	0.8991	0.8681	0.8324	0.8706	0.8691
Windowed	0.8952	0.7790	0.6885	0.8426	0.8023
Online ret.	0.8991	0.7591	0.6500	0.8401	0.7872
Offline	0.8991	0.6963	0.5191	0.8075	0.7284

Table 3.4: Summary of forgetting strategy performance on the Olympics testing set in decreasing order of weighted accuracy.

The Artificial6 Dataset

The Artificial6 dataset presents various learning tasks throughout 5 epochs taking place over a 2-dimensional square data space centred around the origin.

Figure 3.7 shows the test dataset made up of 4 epochs and the cumulative training dataset available to learners to make predictions at each epoch. Concept drift is once again local as the data space was segmented into 4 equal area squares and points from the different classes where uniformly generated in disjoint squares for each epoch. It is easily seen that some training points will have to be discarded as time goes by as the cumulative training set contains contradictory patterns (bottom row of Figure 3.7). Drift events were purposefully exaggerated in this dataset as reflected by the relatively high class flip measure of 31.7% shown in Table 3.2.

Epoch 0 covers half of the data space and is the initial training set. Epoch 1 corresponds to a class flip event where points are in the same area of the data space as in epoch 0 but are of opposite classes. This epoch is challenging for a learner using online retraining which will get confused by the contradicting patterns. Epoch 2 represents a concept drift event without class flips: a new area in the data space starts to be populated when it was previously empty, it requires adaptation but does not require forgetting. Epoch 3 corresponds to old patterns becoming relevant again with the top left square having the same pattern as in epoch 1. This epoch is tricky for windowed learning as a choice has to made between a small window to adapt fast which will discard the points from epoch 1 or a large window that will keep the pattern of epoch 1 but carry contradictory patterns from epoch 0. Epoch 4 emphasises this problem again as it contains the patterns from epochs 1 to 3.

Figure 3.8 shows how each forgetting strategy reacts to the drift and performance is summarised in Table 3.5. Note that the initial training set is composed of the instances corresponding to epoch 0 only (80 instances). The vertical red bars of Figure 3.8 show the different epochs and the moving average A_1 is taken over a window of size 30.

The best hyperparameters for TC are $\epsilon = 3$, a minimum cluster size of 2 and a training window size of 10, these hyperparameters yield a single cluster so the local training parameter's value

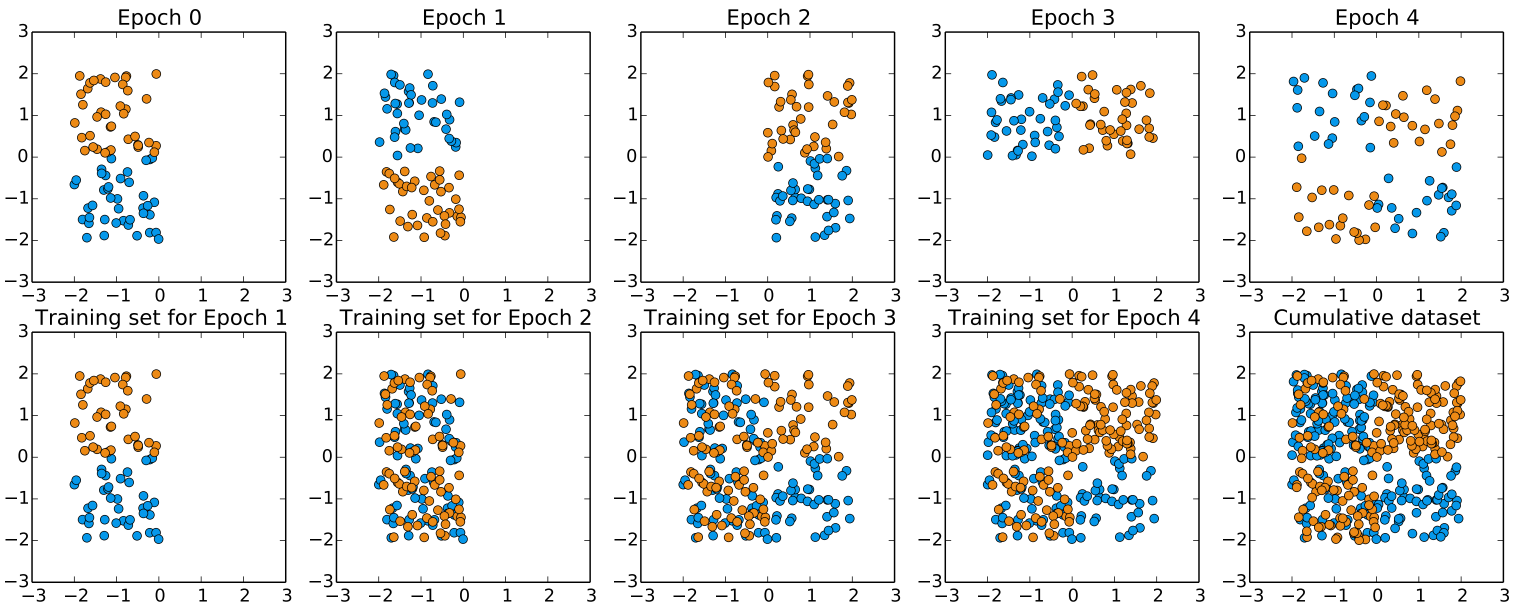


Figure 3.7: Visualisation of the Artificial6 dataset. The top row shows the data for each epoch. Epoch 0 is used as the initial training set and the subsequent epochs are all part of the testing set. The second row shows the cumulative training data available to learners before each test epoch.

is insignificant and means TC converges to the same forgetting strategy as having a training window of size 10. The best hyperparameters for TG are $f = 2$, a training window size of 3 and training using the union of all local training windows. Predictions are thus made based on 12 points, 3 points from each area, each coming from potentially different epochs. The best hyperparameters for FISH are $\alpha_s = 0.1$, $\alpha_t = 0.9$ and a training window size of 10. Because of the high α_t value, this is almost a training window strategy but also takes into account spacial distance. The best hyperparameters for DIOC are $k = 5$ and a training chunk size of 30 for each base learner. Predictions are thus made by choosing the learner, and corresponding 30 training instances, which performs best on the 5 nearest neighbours of a test point in the validation set. The best window size for baseline windowed learning was found to be 10.

The results show that TG performs best ($A_1 = 0.9844$), this is because it only forgets data if there are newer instances covering the same areas, or pattern, of the dataspace. In this case, we note that the way in which TG segments the data space closely matches the data generation process—which is unlikely to happen in practice. FISH comes in second ($A_1 = 0.9094$) and performs better than TC and windowed learning which are the same strategy in this particular case ($A_1 = 0.8875$). The advantage of FISH over the windowed strategy comes from $\alpha_s = 0.1$ and $\alpha_t = 0.9$ (the three strategies use the same window size) as it essentially behaves like a simple windowed strategy but α_s also includes some points that are very close to the current test point and enables for 'old but still valid' patterns to be present in its dynamically formed training windows. This is a double-edged sword as some obsolete instances from epoch 0 might then also be be included in later epochs' training windows if they are close enough to a test point. Further, unlike TC, FISH does not depend on an initial segmentation of the data space which is where TC shows its weakness. Because epoch 0 covers only half the space, clusters aren't made for the other half and points from epochs 1 and beyond get merged in a single inconsistent training window so that the best strategy is then to keep only the most recent points. DIOC lags behind ($A_1 = 0.7781$) as it is data inefficient and Artificial6 is a relatively small dataset (320 test instances). Last, online and offline learning perform no better than a naive learner (A_1 around 0.5) as offline learning is unable to cope with new patterns and online gets confused by opposite class instances very close in space showing the need to apply

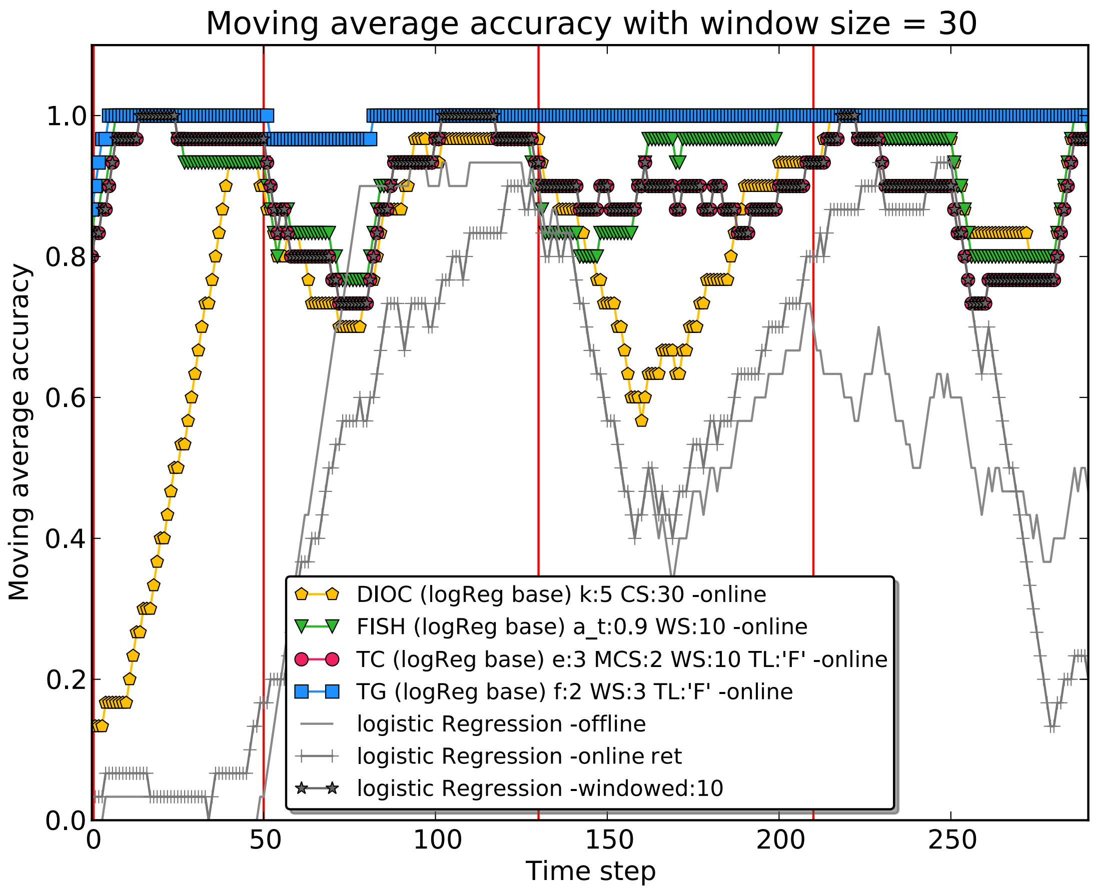


Figure 3.8: Evolution of forgetting strategy performance throughout the Artificial6 testing set. Moving average performance computed using a window of size 30.

a forgetting strategy in this dataset as well.

Strategy	Recall c^+	Precision c^+	Recall c^-	Precision c^-	A_1
TG	0.9875	0.9814	0.9812	0.9874	0.9844
FISH	0.9187	0.9018	0.9000	0.9172	0.9094
TC	0.8875	0.8875	0.8875	0.8875	0.8875
Windowed	0.8875	0.8875	0.8875	0.8875	0.8875
DIOC	0.7562	0.7908	0.8000	0.7665	0.7781
Online ret.	0.3438	0.5000	0.6562	0.5000	0.5000
Offline	0.3625	0.4957	0.6312	0.4975	0.4969

Table 3.5: Summary of forgetting strategy performance on the Artificial6 testing set in decreasing order of weighted accuracy.

3.4.3 Results Beyond Simple and Synthetic Cases

Each of the 6 real datasets that we used to further benchmark the forgetting strategies show interesting properties.

Ž-elec Results

Ž-elec shown in Figure 3.9 with performance summarised in Table 3.6, shows a case where a very simple forgetting strategy is optimal. Making predictions based on the two most recent training points yields higher performance than any other forgetting strategy, confirming the findings of [Zli13] that show data from the closest previous two periods can be used to predict current electricity prices with high accuracy. In this case, online retraining and offline learning perform worse than the optimal forgetting strategy (15.5% and 24.8% respectively). This is reflected in the high value for the class flip measure reported in Table 3.2, 26.3%, which signalled that a forgetting strategy will be perform better than no or constant forgetting.

Strategy	Recall c^+	Precision c^+	Recall c^-	Precision c^-	A_1
TC	0.8876	0.8836	0.8682	0.8727	0.8785
TG	0.8876	0.8836	0.8682	0.8727	0.8785
FISH	0.8876	0.8836	0.8682	0.8727	0.8785
Windowed	0.8876	0.8836	0.8682	0.8727	0.8785
DIOC	0.8776	0.8352	0.8049	0.8537	0.8434
Online ret.	0.9755	0.6951	0.5179	0.9494	0.7603
Offline	0.9918	0.6428	0.3790	0.9763	0.7037

Table 3.6: Summary of forgetting strategy performance on the Ž-elec testing set in decreasing order of weighted accuracy.

Ž-luxem Results

On the other hand, Ž-luxem shown in Figure 3.10 with performance summarised in Table 3.7, has a class flip measure of 0% foreshadowing that no forgetting strategy is necessary — all historical survey data is beneficial when making predictions. This is proven to be correct as

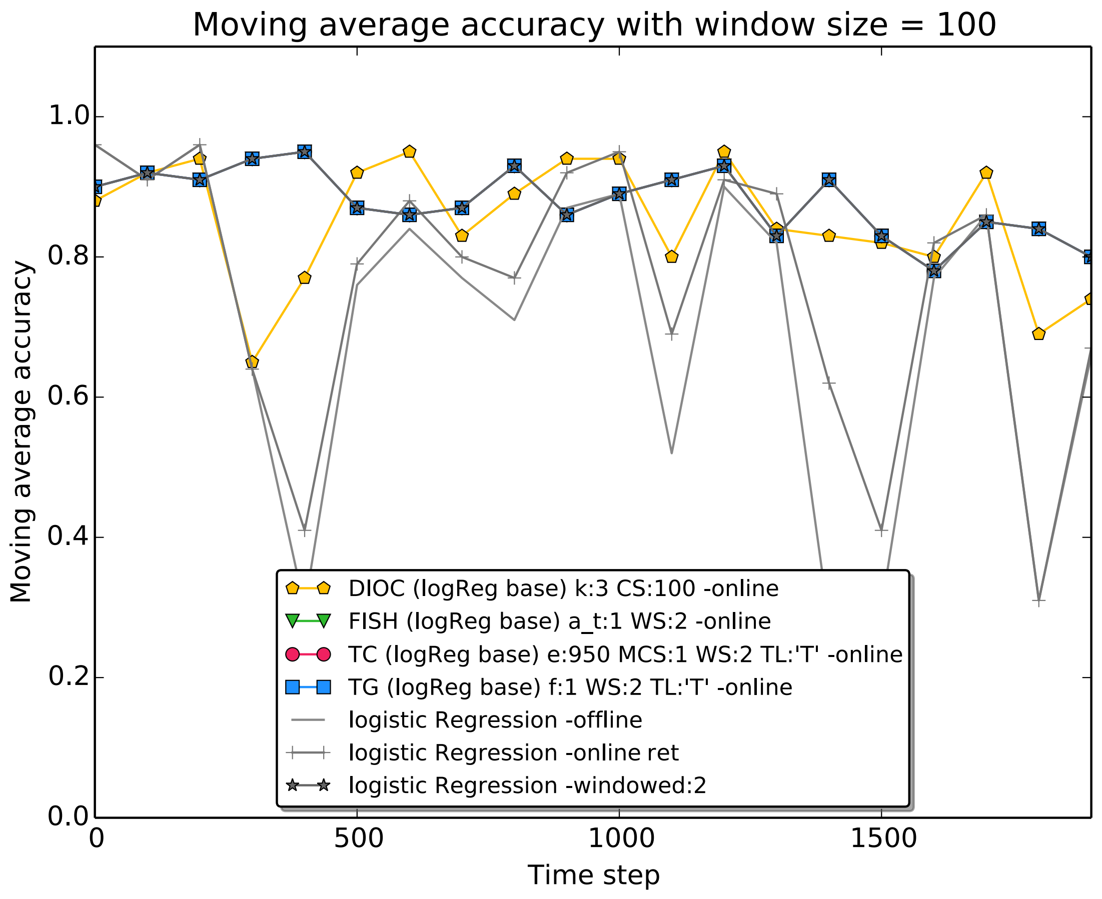


Figure 3.9: Evolution of forgetting strategy performance throughout the Ž-elec testing set. Moving average of the A_1 weighted accuracy computed using a window of size 100.

online retraining, the no-forgetting strategy, yields a near perfect A_1 measure. Offline, the constant forgetting strategy, performs 13.9% worse than online which could also be suspected as the 5.8% drift measure value for the dataset indicates that some adaptiveness is needed nonetheless; supporting our claim that concept drift and the need to forget need to be decoupled.

Twitter-active Results

Twitter-active shown in Figure 3.11 with performance summarised in Table 3.8, has three interesting patterns. TC, TG and DIOC perform best with TC and TG forgetting points locally but making predictions globally (their *trainLocal* parameter is False, TC uses 6 clusters, i.e. local training windows, and TG uses 4). This means that the user in question has tweet

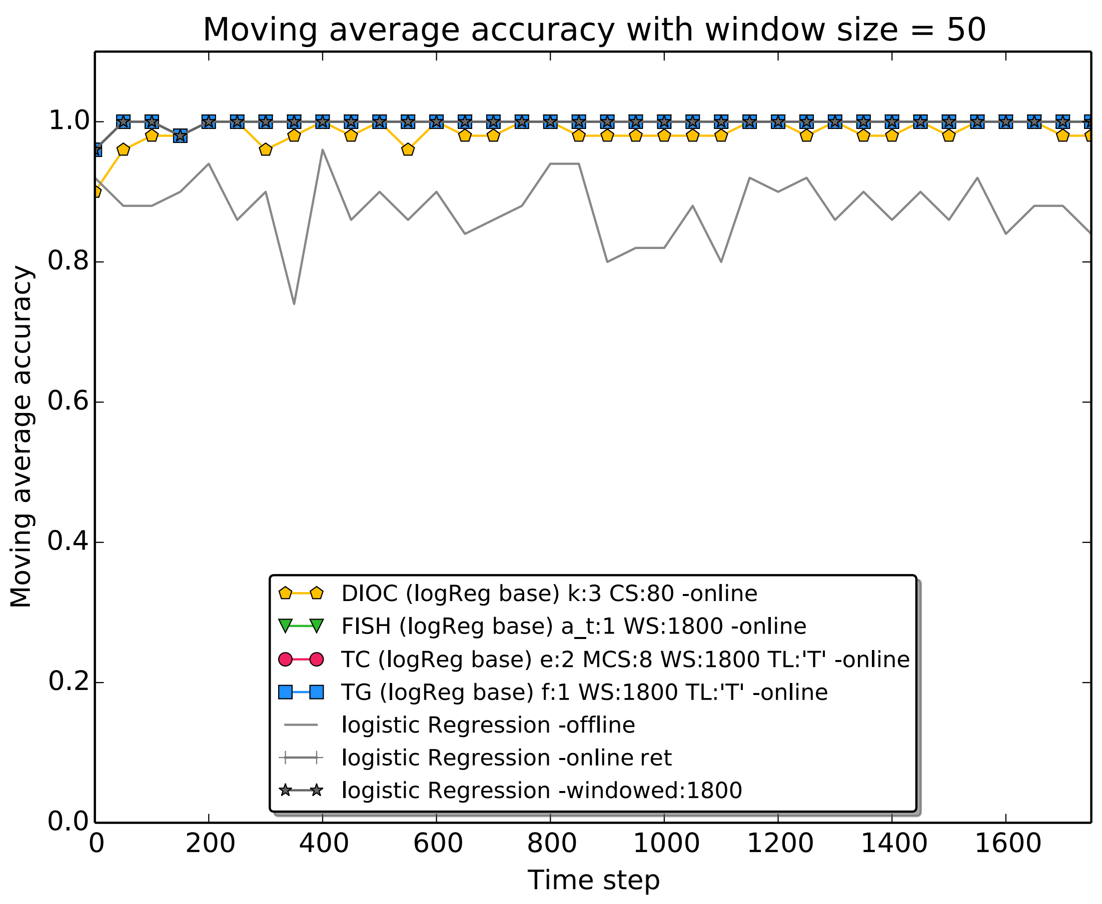


Figure 3.10: Evolution of forgetting strategy performance throughout the Ž-luxem testing set. Moving average of the A_1 weighted accuracy computed using a window of size 50.

Strategy	Recall c^+	Precision c^+	Recall c^-	Precision c^-	A_1
TC	0.9989	0.9978	0.9977	0.9989	0.9983
TG	0.9989	0.9978	0.9977	0.9989	0.9983
Online ret.	0.9989	0.9978	0.9977	0.9989	0.9983
Windowed	0.9989	0.9978	0.9977	0.9989	0.9983
FISH	0.9989	0.9978	0.9977	0.9989	0.9983
DIOC	0.9859	0.9816	0.9808	0.9853	0.9834
Offline	0.9978	0.8053	0.7494	0.9970	0.8760

Table 3.7: Summary of forgetting strategy performance on the Ž-luxem testing set in decreasing order of weighted accuracy.

patterns which change independently from each other and must be updated as such. Twitter-active is also the only case where FISH performs considerably worse than DIOC (12.5%) with performance similar to the windowed strategy. Last, as predicted by the 57.1% class flip measure, online and offline perform markedly worse than the optimal forgetting strategy (68.0% and 93.8% respectively). It is also interesting to note that although both have similar A_1 values, the test points they correctly predict are quite distinct as shown on the corresponding figure.

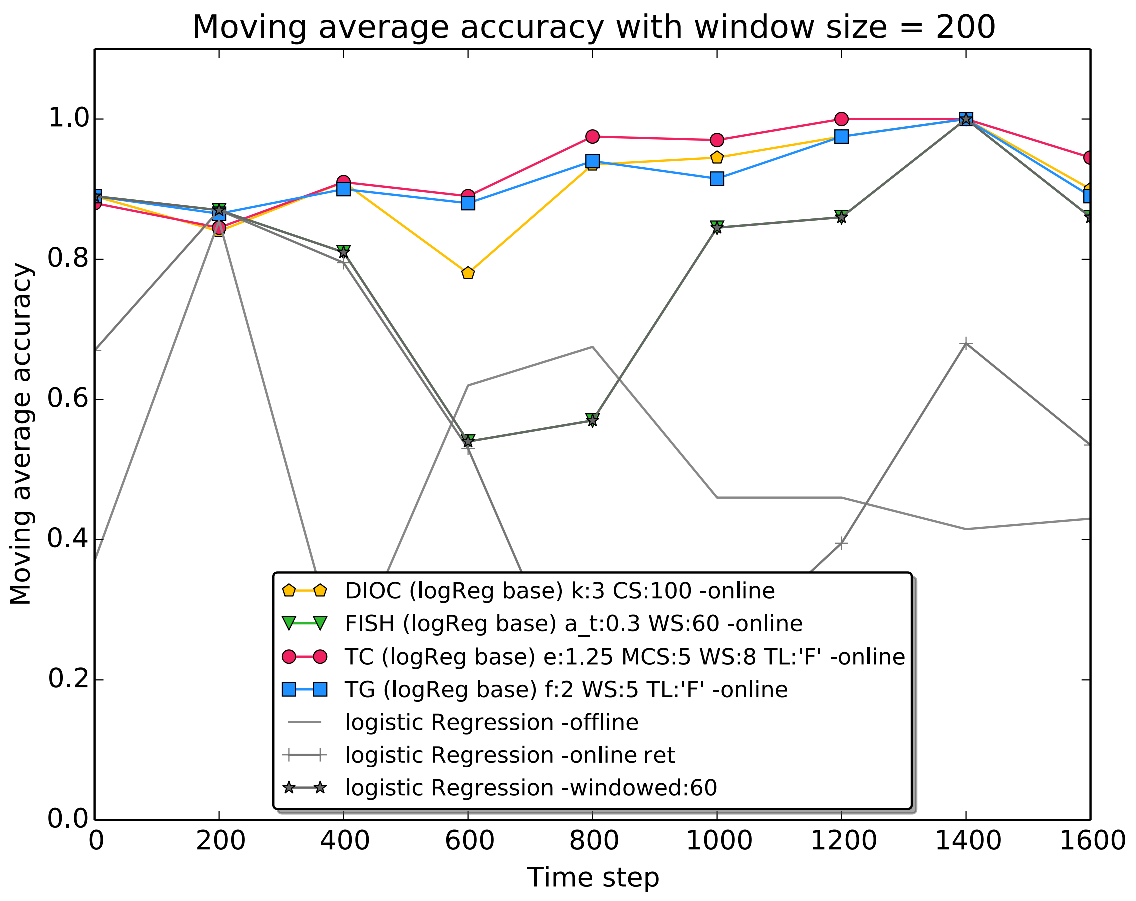


Figure 3.11: Evolution of forgetting strategy performance throughout the Twitter-active testing set. Moving average of the A_1 weighted accuracy computed using a window of size 200.

Strategy	Recall c^+	Precision c^+	Recall c^-	Precision c^-	A_1
TC	0.9274	0.9695	0.9512	0.8869	0.9363
TG	0.9007	0.9651	0.9456	0.8507	0.9175
DIOC	0.9032	0.9525	0.9247	0.8511	0.9112
FISH	0.7304	0.9552	0.9428	0.6767	0.8099
Windowed	0.7304	0.9552	0.9428	0.6767	0.8094
Online ret.	0.3656	0.8327	0.8773	0.4528	0.5572
Offline	0.3932	0.6417	0.6332	0.3844	0.4830

Table 3.8: Summary of forgetting strategy performance on the Twitter-active testing set in decreasing order of weighted accuracy.

Twitter-times Results

Twitter-times shown in Figure 3.12 with performance summarised in Table 3.9, shows the classic concept drift scenario. Total concept drift occurs, so that a windowed strategy performs best, i.e. old data is useless when making new predictions. This means the user's underlying Twitter behaviour constantly evolves during the 6+ years the data was collected over. The relatively low 6.4% class flip measure, yet high 64.3% concept drift measure explains the performance gap between online and offline strategies which perform 23.0% and 97.3% worse than the optimal forgetting strategy.

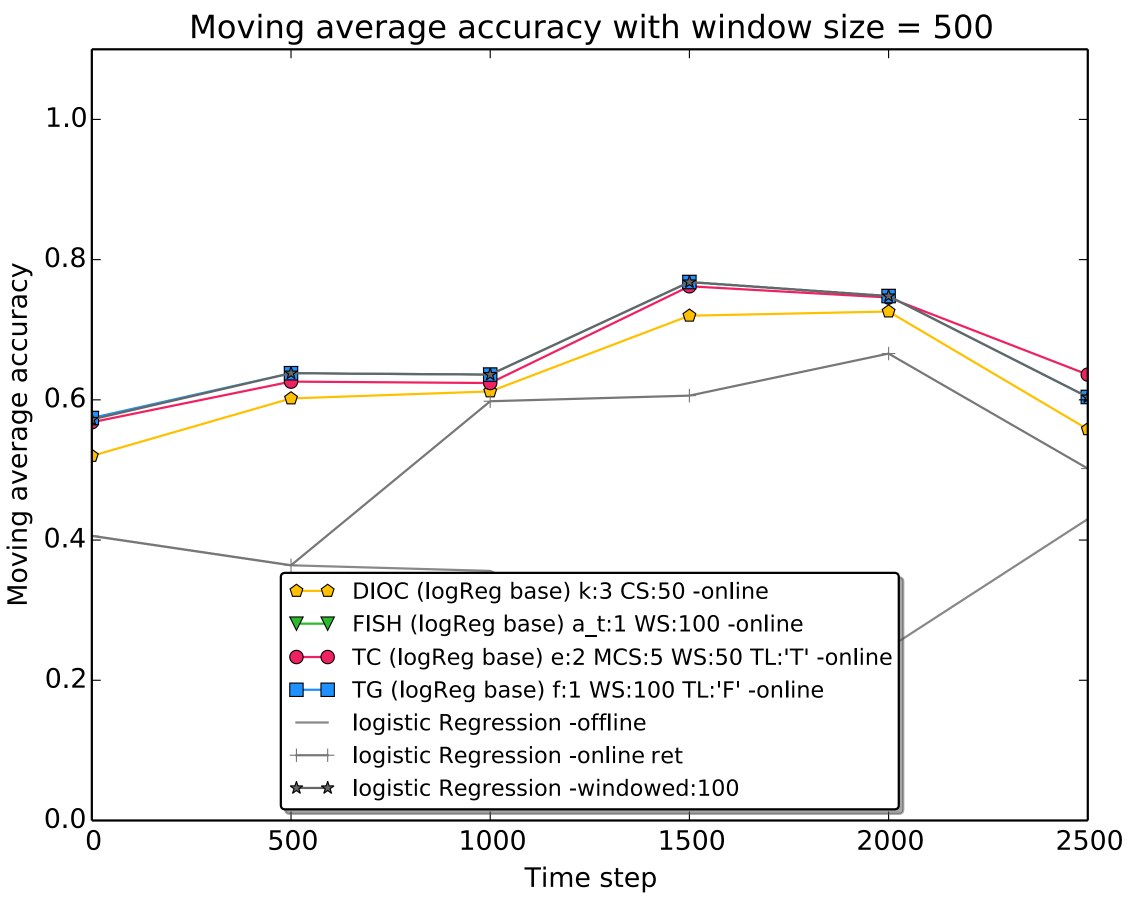


Figure 3.12: Evolution of forgetting strategy performance throughout the Twitter-times testing set. Moving average of the A_1 weighted accuracy computed using a window of size 500.

Strategy	Recall c^+	Precision c^+	Recall c^-	Precision c^-	A_1
TG	0.2613	0.4949	0.8653	0.6986	0.6625
FISH	0.2613	0.4949	0.8653	0.6986	0.6625
Windowed	0.2613	0.4949	0.8653	0.6986	0.6625
TC	0.2757	0.4920	0.8562	0.7006	0.6613
DIOC	0.3631	0.4283	0.7551	0.7012	0.6235
Online ret.	0.5297	0.3693	0.5430	0.6956	0.5386
Offline	1.0000	0.3357	0.0000	0.0000	0.3357

Table 3.9: Summary of forgetting strategy performance on the Twitter-times testing set in decreasing order of weighted accuracy.

Sleep-dur Results

Sleep-dur shown in Figure 3.13 with performance summarised in Table 3.10, shows a case where TC and TG have a large performance gap, TC is the best performing strategy and TG comes in 4th performing 17.2% worse. Interestingly, TC achieves this result using only 6 clusters while TG uses a much finer segmentation of the data space — 125 grid elements. Further, the low performance of the windowed strategy shows the importance of discarding data locally in this case, this corresponds to certain sleep patterns happening less frequently than others, yet remaining valid throughout portions of the dataset. The class flip measure of 14.1% and high concept drift measure of 64.7% once again foreshadow the performance of online and offline strategies which perform 25.5% and 56.6% worse than TC.

Strategy	Recall c^+	Precision c^+	Recall c^-	Precision c^-	A_1
TC	0.7345	0.7034	0.7287	0.7581	0.7314
FISH	0.6283	0.6339	0.6822	0.6769	0.6570
DIOC	0.6814	0.6063	0.6124	0.6870	0.6446
TG	0.5133	0.6170	0.7209	0.6284	0.6240
Windowed	0.4956	0.5714	0.6744	0.6042	0.5909
Online ret.	0.5752	0.5508	0.5891	0.6129	0.5826
Offline	1.0000	0.4669	0.0000	0.0000	0.4669

Table 3.10: Summary of forgetting strategy performance on the Sleep-dur testing set in decreasing order of weighted accuracy.

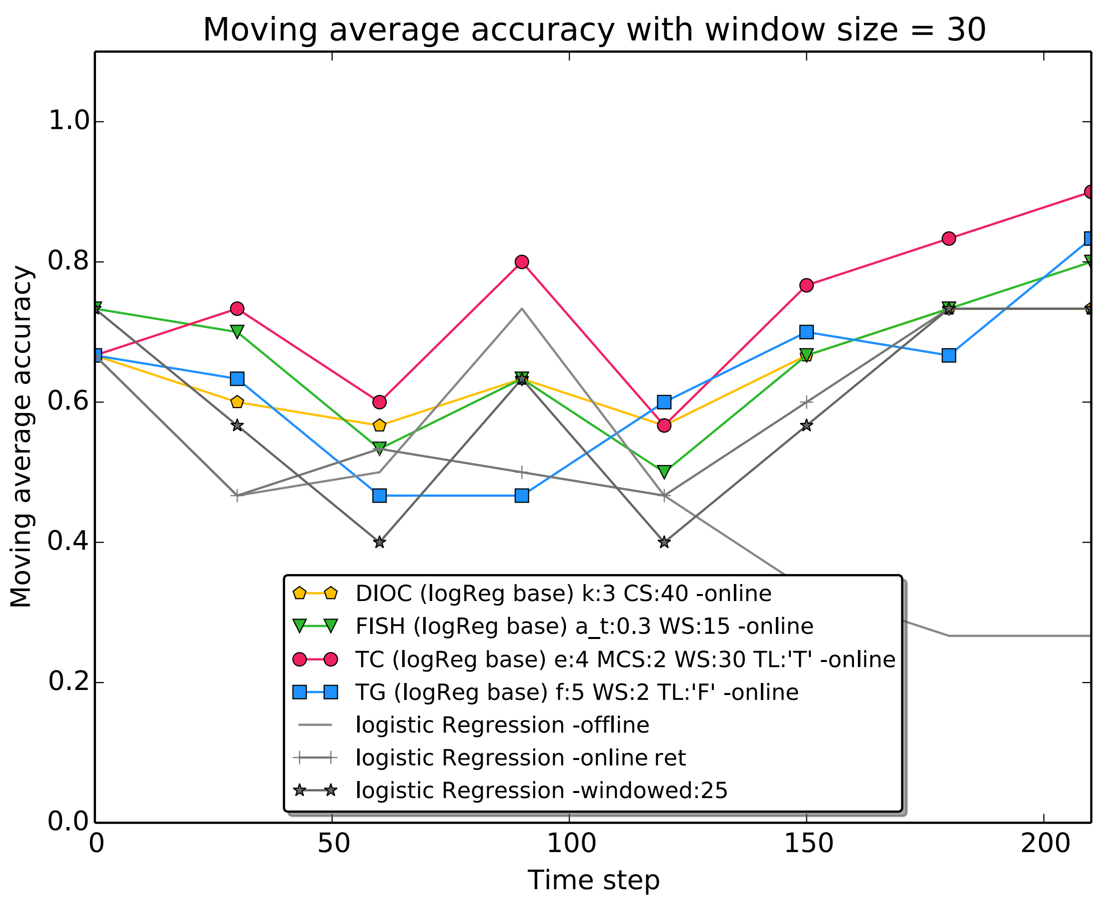


Figure 3.13: Evolution of forgetting strategy performance throughout the Sleep-dur testing set. Moving average of the A_1 weighted accuracy computed using a window of size 30.

Sleep-time Results

Sleep-time shown in Figure 3.14 with performance summarised in Table 3.11, shows a case where TC and TG converge to the same forgetting strategy and perform significantly better than other forgetting strategies. Similarly to Sleep-dur, local changes in sleep patterns are not well handled by other forgetting strategies and the class flip measure of 13.2% and concept drift measure of 64.1% explain the low performance of offline and online which perform 49.8% and 57.8% worse than TC and TG.

In terms of overall performance, TC is the most versatile forgetting strategy. It either converges to the best found other forgetting strategy: window of size 2 for Ž-elec, online for Ž-Luxem. and a window of size 100 for Twitter-times, or performs the best on datasets where simple strategies

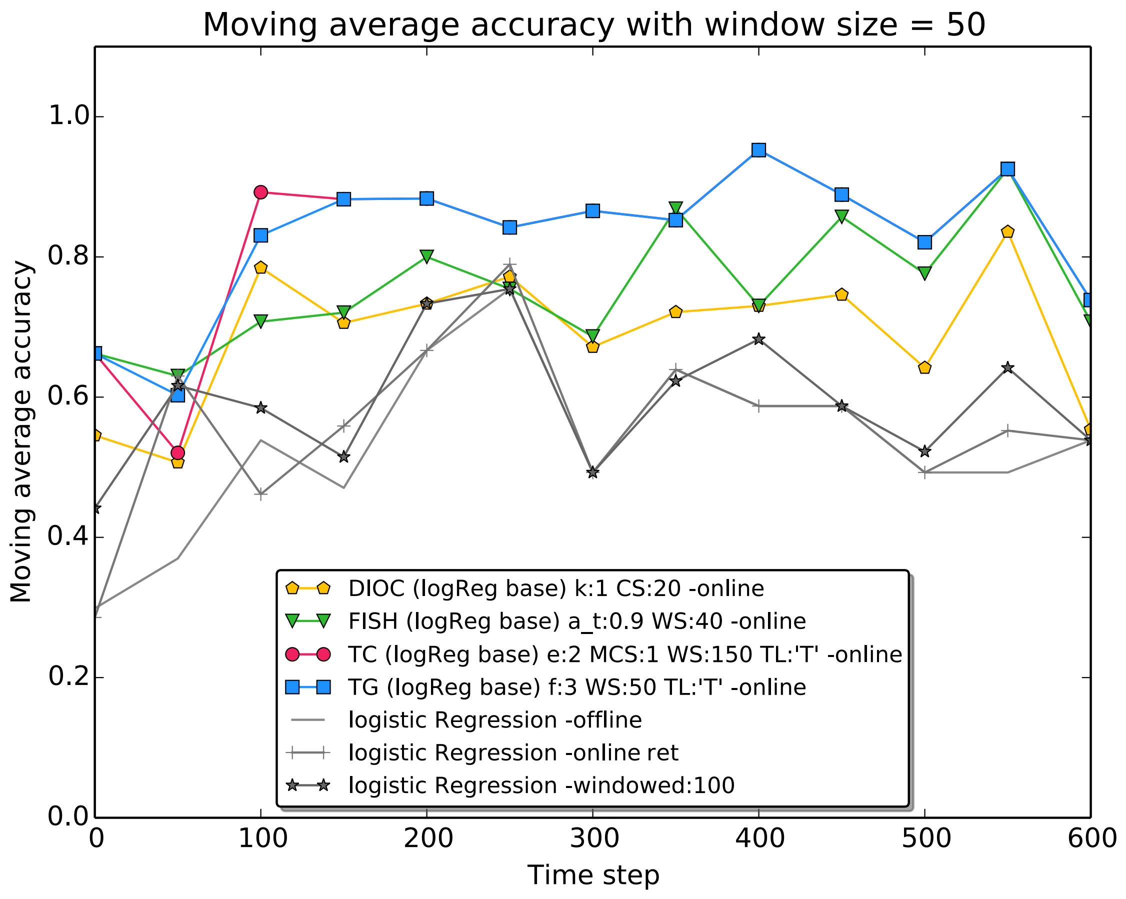


Figure 3.14: Evolution of forgetting strategy performance throughout the Sleep-time testing set. Moving average of the A_2 weighted accuracy computed using a window of size 50.

Strategy	Recall c^+	Precision c^+	Recall c^-	Precision c^-	A_2
TG	0.8018	0.7045	0.8470	0.9038	0.8255
TC	0.8018	0.6988	0.8428	0.9034	0.8233
FISH	0.6912	0.6410	0.8239	0.8543	0.7607
DIOC	0.5806	0.5526	0.7862	0.8047	0.6883
Windowed	0.3041	0.4925	0.8574	0.7304	0.5939
Online ret.	0.3272	0.3777	0.7547	0.7115	0.5510
Offline	0.0000	0.0000	1.0000	0.6873	0.5236

Table 3.11: Summary of forgetting strategy performance on the Sleep-time testing set in decreasing order of weighted accuracy.

are weak: $A_1 = 0.9363$ for Twitter-active (runner-up DIOC with $A_1 = 0.9112$), $A_1 = 0.7314$ for Sleep-dur (runner-up FISH with $A_1 = 0.6570$) and $A_2 = 0.8255$ for Sleep-time (runner-up FISH with $A_2 = 0.7607$). TG is also capable of converging to simple forgetting strategies in the same cases as TC and performs well on Twitter-active ($A_1 = 0.9175$, 2.0% worse) and

Sleep-time ($A_2 = 0.8233$, 0.02% worse). The only case where it performs clearly worse than TC is Sleep-dur ($A_1 = 0.6570$, 17.2% worse).

FISH is also capable of converging to simple forgetting strategies on Ž-elec, Ž-luxem and Twitter-times, but performs worse than TC on Twitter-active ($A_1 = 0.8099$, 15.6% worse), Sleep-dur ($A_1 = 0.6570$, 11.3% worse) and Sleep-time ($A_2 = 0.7607$, 8.5% worse) only performing better than TG on Sleep-dur by 5.2%. We hypothesise this is due to way it combines spacial distance and time when forming training sets which uses potentially obsolete points if they are close enough in space to a test point. DIOC, on the other hand, does not manage to converge to simple forgetting strategies on Ž-elec, Ž-luxem and Twitter-times although it only performs slightly worse than the optimal forgetting strategy in all cases 4.1%, 1.5% and 6.5% respectively. On Twitter-active, Sleep-dur and Sleep-time it performs 2.6%, 13.4% and 19.9% worse than TC respectively. We hypothesise this is due to lack of flexibility in the instances it is able to select as this ensemble techniques is constrained to use chunks of time-linear data when making predictions.

3.4.4 Discussion and limitations

Throughout the 8 experiments shown above, the way in which TC and TG combine space and time to form training datasets has proved to perform at least as well as any other forgetting strategy and many times better, with performance delta ranging from 0 to 11.3% compared to the best of FISH and DIOC on each dataset. The second best performing strategy was FISH which, similarly to TG and TC, was able to converge to simple windowed or online strategies when optimal, something DIOC did not manage.

TC is the best performing strategy on all datasets except Artificial6 where it shows one of its two weaknesses. Because testing data appears in new areas as compared to the training data, it is not properly covered by any of TC's clusters and leads to clusters with contradictory patterns yielding low performance. As mentioned in Section 3.2.4, a solution would be to periodically recompute optimal cluster and window parameters using the most recent n training points as a validation set. The second weakness of both TC and TG is that they require a retraining of the associated base learner which adds a number of steps proportional to the number of areas when creating a training set.

Finally, we further confirmed the results of Section 3.1 by showing that the concept drift measure was linked with the need to use an adaptive learning strategy but does not always imply a forgetting strategy is appropriate, in the case of Ž-luxem, online learning is optimal which is hinted by the null class flip measure returned by the concept drift analysis algorithm.

3.5 Summary

In this chapter we addressed the problem of knowing if concept drift is present in a dataset. We proposed to decouple the notions of concept drift, simply meaning a change in a dataset's underlying data distribution, and the need to forget, where patterns in a dataset change in a way that contradicts previous patterns. Our proposed concept drift analysis algorithm (CDA) outputs two quantities, a concept drift measure and a class flip measure. We experimentally showed a link between the need to use adaptive learning and the concept drift measure and between the class flip measure and the need to forget certain instances while learning online using 5 datasets in the Section 3.1 and 8 further datasets in Section 3.4.

We then reviewed the two existing learner independent forgetting strategies able to cope with local concept drift DIOC and FISH and proposes two new strategies TC and TG. The TC and TG forgetting strategies offer a new way of dealing with concept drift based on the principle of local forgetting: instances are discarded only if they can be replaced by newer ones covering the same area, the two methods differ only in the way they define areas for a given dataset, clusters for TC and grid areas for TG. In data from mobile context-aware applications, each area is assumed to loosely correspond to a different user context so that the way in which TC and TG discard instances allows for a change in a specific behaviour independently of others while also allowing for changes in multiple behaviours at once, putting minimal requirements on the type and intensity of concept drift tolerated in an application. Similarly to FISH and TG, these forgetting strategies thus take into consideration time and distance when forming training sets to make predictions. Further, they have the same advantage as FISH as they encompass online retraining, windowed learning and a form of nearest-neighbour Case Base Reasoning as a special case of their expression.

In order to compare these different strategies, the chapter also proposed to use a weighted accuracy measure A_w as a performance measure. This measure allows to account for user preferences in terms of which class is most important to predict correctly as a function of w. This is relevant for mobile context-aware applications as datasets can be unbalanced and it can be the case that the minority class labels an event with more severe consequences than the majority class i.e. the relative cost of unsignalled alarms versus false alarms taking into account the frequency of the alarm causing event. This relative costs is encoded in the w parameter of the weighted accuracy measure.

Finally, we benchmarked seven forgetting strategies using the weighted accuracy measure A_w, the 4 mentioned above and 3 baseline strategies, windowed learning, online retraining and offline learning on eight different datasets. The results shows that TC is the most versatile and overall best performing strategy, able to converge to simple strategies when these are optimal and perform best in more complex cases. Although TG follows closely, the finer grained control given by considering instance clusters instead of identical instance grid areas gives an advantage to TC in most cases, also, TC usually requires many less clusters than grid elements saving on computation time.

Chapter 4

Integrating Machine Learning into Mobile Context-aware Applications

The previous chapter treated about how the user requirements for intelligent mobile context-aware applications influence the machine learning aspects of it. We now show how the requirements stemming from the use of machine learning influences the application itself through a detailed practical use case. Specifically, this chapter addresses the data collection and labelling challenge in intelligent mobile context-aware applications (Challenge 1). Through the practical example of smartphone notifications, we show the extent to which an application's internal use of machine learning can affect its user-facing component. We focus on disruptive incoming calls and propose an *infer-and-confirm* data gathering strategy which silently collects data and infers labels from the user's natural response to target events while letting them amend inferred labels, if necessary. We finish by applying the CDA algorithm and forgetting strategies from Chapter 3 to real datasets gathered using our own RingLearn Android application and Cambridge's Device Analyzer Android application, applying and assessing the solutions proposed in this thesis in their entirety.

4.0.1 Why use Machine Learning?

The concept of ubiquitous computing is approximately 25 years old [Wei91] and while some aspects of it are now part of our daily lives, such as the widespread use of wireless devices, other aspects are still mostly at the research and prototype stage. This is the case of ambient intelligence [GZW11] which intelligent mobile context-aware applications are a part of.

While there is a limited number of mobile context-aware applications available to the general public, with the exception of the Nest and Luna Thermostat auto-schedulers, applications are *not intelligent*. That is to say, their underlying context-to-action mapping has to be pre-programmed by an expert or the user. Here are notable examples

- **Medical asset tracking** at Tallahassee Memorial HealthCare where equipment is tagged so that nurses can see assets on a floor map and access them quickly when needed [Wor15].
- The **Walksafe** Android application which uses computer vision to alert users talking on the phone if a vehicle is approaching when they are crossing the road [CoB15].
- The **Launch Here** iOS application which launches user-specified applications when a given location is visited [app15]

Some approaches, such as Walksafe, do use machine learning (for computer vision) but only to help sense the context. Using machine learning to learn a user's context to action map offers the possibility to effortlessly complement or even replace pre-programmed static rules. Pre-programmed static rules have two main weaknesses. First, they are difficult to exhaustively specify, it can be tedious, or often times impossible, to list all contexts and all corresponding actions an application might face during its deployment. Second, due to the dynamic nature of human behaviour, new rules might be needed and existing rules can become incorrect as time passes — the topic of the Chapter 3. Machine learning alleviates this problem by using known context to action examples to generalise context to action mappings (rules) with minimal need for human input. Finding the right way to use machine learning in mobile-context aware applications is thus a cornerstone to fulfil the promise of ubiquitous computing to change our daily lives.

4.0.2 Why Focus on Disruptive Incoming Calls as a Use Case?

Smartphone notifications have gained a lot of interest lately as apps are increasingly competing for user attention using push notifications [Mid07] and can be disruptive [FGB11, KS06, LBGK12]. A disruptive notification is a notification that interrupts the recipient such that the disturbance of receiving the notification outweighs the perceived benefit [SLM$^+$14]. For instance, for a conference paper submission, a call from a paper co-author at midnight may be disruptive or not depending on the how close the paper submission deadline is and the current state of the submission. Disruptive notifications thus concern virtually every smartphone user. To decide which type of notification to focus on, we performed an initial survey with 65 participants [SD14], with highlights shown in Figure 4.0.2[1]. The survey reported that while SMS was found to be the most frequently used communication method for 41% of participants, 51% of participants answered that they have already been bothered by incoming phone call ringtones, versus only 28% for an audible incoming SMS alert.

Further, we found that for a typical week, the number of incoming calls per surveyed participant was less than 7 for 18% of them, between 7 and 14 for 30% of them, between 15 and 21 for 27% of them, and more than 21 for 25% of them. Moreover, 43% rejected between 1 and 3 incoming calls per week, 32% rejected between 4 and 7, 17% rejected less than 1 and 8% rejected more than 8, indicating that disruptive incoming calls are not only potentially the most disruptive notification but are an everyday occurrence, justifying our choice.

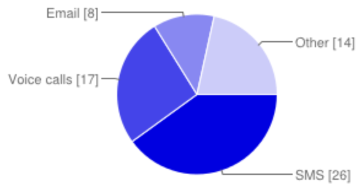
4.1 Background

4.1.1 Interruptions

An interruption can be defined as an event disrupting a person's continuity of cognitive focus while performing a task [FGB11]. This definition assumes that human behaviour can be

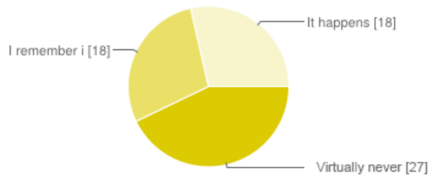
[1]The full survey results can be found at `https://docs.google.com/forms/d/1OFe19KoE0FqgMltL4_5fUfE9C_VCb8OjMYrda3WwXbg/viewanalytics`

Which is the communication method you use most frequently on your mobile?



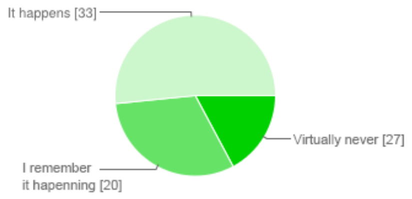
SMS	**26**	40%
Voice calls	**17**	26%
Email	**8**	12%
Other	**14**	22%

How often are you in a situation where an audible incoming SMS alert bothers you?



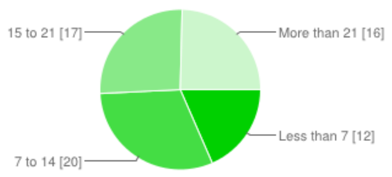
Virtually never	**27**	42%
I remember it happening in the past but it's not a real concern	**18**	28%
It happens	**18**	28%

How often are you in a situation where an incoming call ringtone bothers you?



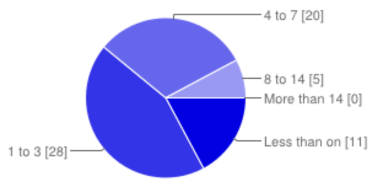
Virtually never	**11**	17%
I remember it happening in the past but it's not a real concern	**20**	31%
It happens	**33**	51%

How many incoming calls do you receive on a typical week?



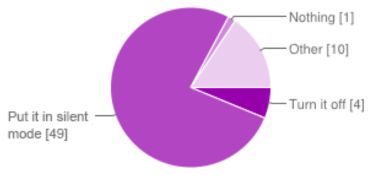
Less than 7	**12**	18%
7 to 14	**20**	31%
15 to 21	**17**	26%
More than 21	**16**	25%

Typically, how many times per week do you reject incoming calls?



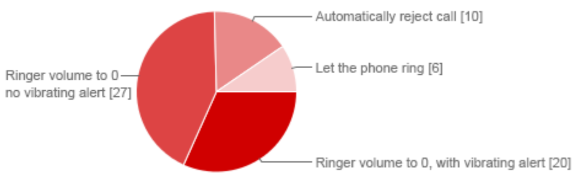
Less than once	**11**	17%
1 to 3	**28**	43%
4 to 7	**20**	31%
8 to 14	**5**	8%
More than 14	**0**	0%

What do you do when you do not want to be disturbed by your phone?



Turn it off	**4**	6%
Put it in silent mode	**49**	75%
Nothing	**1**	2%
Other	**10**	15%

Imagine there was an mobile app that could magically know whether an incoming call will disturb you and take an action, would you prefer?



For the app to put the ringer volume to 0 but still have a vibrating alert for the call	**20**	31%
For the app to put the ringer volume to 0 and do not have a vibrating alert for the call	**27**	42%
For the app to automatically reject the call i.e. as if you pressed the hang up button before the call started ringing	**10**	15%
For the app to do nothing, let the phone ring unless it is in a silent mode	**6**	9%

Figure 4.1: Summary of the call behaviour questions and answers in our mobile phone usage survey conducted on 65 participants.

modelled by a sequence of non-overlapping tasks, also called activities, which can be further broken down into phases. For instance, making a call from a mobile could be broken down into: retrieving the mobile, finding the contact to call, calling the contact, getting a response for the call, speaking, terminating the call and replacing the mobile. In any of its phases, the *primary* task can be interrupted by an event that becomes a *secondary* task if it initiates a new activity for this person.

Interruptions have been shown to have quantifiable consequences on the performance and completion of primary tasks usually dependent on the phase they occur in [FHA+05]. For instance, [CCH00] shows that for a task consisting of a web-search followed by an analysis of the results, an interruption happening in the analysis phase of the task yields longer completion times than in other phases. Conversely, interruptions can sometimes be beneficial. This and other studies [FYB+10, LBGK12, MGH05, TM13, GS09, WWR+10, PdOKO14, AH06], show that interruptions unrelated to the ongoing task take longer to attend to than interruptions useful to task completion, so that the disruptive value of an interruption not only depends on when it occurs but also its content with respect to the recipient's current context and task.

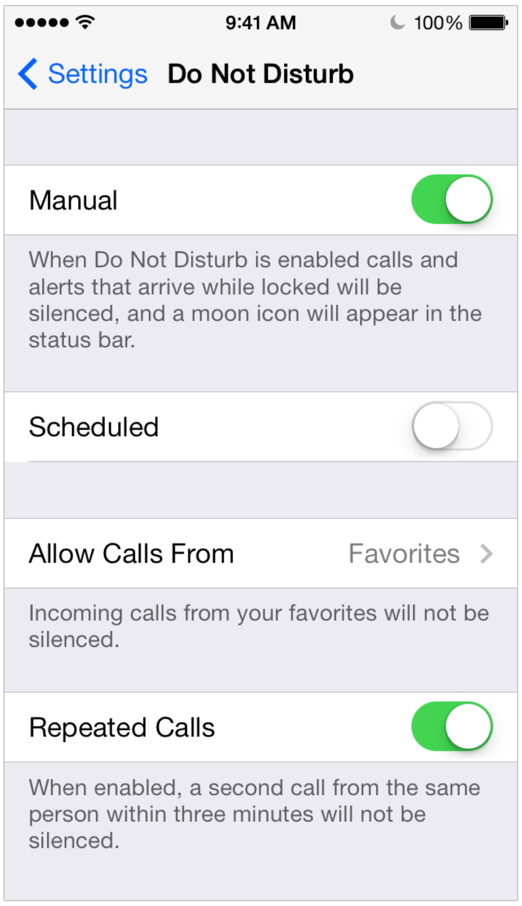
Interruptions themselves have been characterised not only in terms of performance costs such as the time to attend the interruption, the time spent dealing with the interruption or the time to resume the interrupted primary task [AB04, LBGK12] but also in terms of cognitive, emotional and even physical effects they cause, such as changes in emotional state and cognitive load [ML02, PCdO14]. Empirical studies have shown that mobile-originating interruptions can have greater impact than face-to-face interruptions [Glu08] and that interruptibility preferences are different across users [KS06, KC05].

4.1.2 Disruptive Smartphone Notifications

In our work, we focus on approaches to handle notification signalling so that they are socially and situationally appropriate. In consumer products, the current solution to handling disruptive notifications is manual profile configuration as detailed below.

Commercial Solutions

The traditional approach to manage disruptive notifications is to manually change a phone's profile to a silent mode. For example in Apple's iOS 8 the 'Do Not Disturb' function shown in Figure 4.2 can be activated for this purpose. When on, it silences all notifications (incoming calls, messages and app push notifications) except for incoming calls from a customisable whitelist set by the user. The function can be scheduled to be activated by manually setting a schedule, and notifications from a particular number can also be entirely blocked by creating a blacklist. An interesting function is the 'Repeated Calls' option, that will not apply the 'Do Not Disturb' policy when activated if a person calls twice within 3 minutes. This is an indirect form of *availability sharing* where a user is willing to be interrupted if the content of the notification is judged important by the interrupter.


9:41 AM 100%

Settings Do Not Disturb

Manual

When Do Not Disturb is enabled calls and alerts that arrive while locked will be silenced, and a moon icon will appear in the status bar.

Scheduled

Allow Calls From Favorites

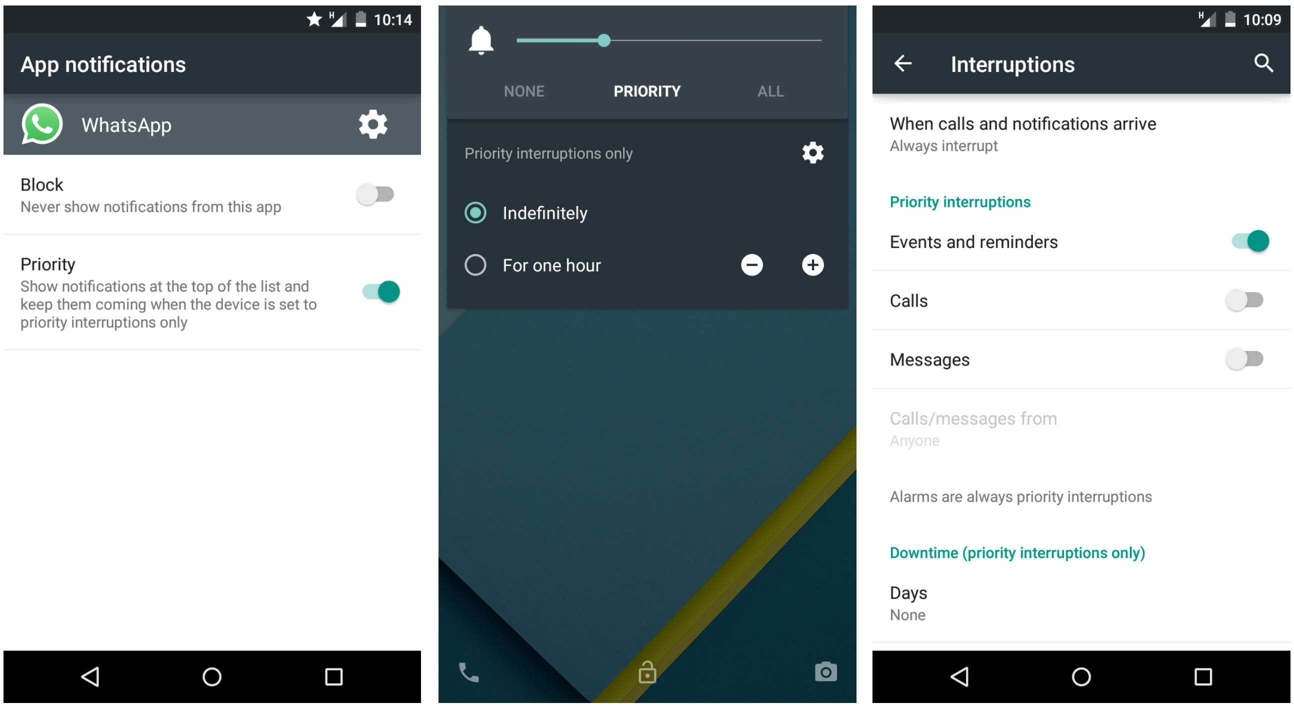
Incoming calls from your favorites will not be silenced.

Repeated Calls

When enabled, a second call from the same person within three minutes will not be silenced.

Figure 4.2: iOS 'Do Not Disturb' setting screen

The latest Android operating system (version 5) introduced a very fine grained notification management system which is based on 3 levels of notification: 'None', 'Priority' and 'All'. Each installed app, including incoming phone calls, messages and calendar events can be assigned to one of these levels (first screen capture on Figure 4.3). Then, depending on which level the user sets for the device itself (second screen capture on Figure 4.3), all notifications will be blocked if 'None' is selected, only notifications from apps that are on the priority list will be delivered if 'Priority' is selected and all notifications will be delivered if 'All' is selected (with the option to blacklist specific apps so that their notifications are never delivered regardless of the device's notification level). A device's notification levels can also be scheduled (third screen capture on Figure 4.3) and it is also possible to block individual contacts as in iOS 8. On top of this, the signal of notifications, including, incoming phone calls, messages and calendar events, can be set to vibrate or a certain level of loudness for the ringtone (second screen capture on Figure 4.3).


10:14

App notifications

WhatsApp

Block
Never show notifications from this app

Priority
Show notifications at the top of the list and keep them coming when the device is set to priority interruptions only

NONE PRIORITY ALL

Priority interruptions only

Indefinitely

For one hour

10:09

Interruptions

When calls and notifications arrive
Always interrupt

Priority interruptions

Events and reminders

Calls

Messages

Calls/messages from
Anyone

Alarms are always priority interruptions

Downtime (priority interruptions only)

Days
None

Figure 4.3: Screen captures from the Android notification management system

Although there are no *intelligent* mobile notification management applications publicly available, on Android, some basic form of context-awareness for notifications does exists in apps such as Smart Call Manager [Man15] where notification volume is dynamically set based on the ambient sound level of the recipient. In summary, Android 5 is more flexible than iOS 8 in terms of notification signalling settings, showing a good example of what a pre-programmed static rule based system might look like in practice. The drawback is that it is also much more complicated, so that it is quite possible that due to the tediousness and required time to set up the mentioned Android lists and rules, common users will keep on using only the notification volume switch already known to them to control notification signalling.

Research Solutions

Intelligent interruptions management stems from human-computer interaction and machine learning research. They can be divided into *scheduling* and *mitigation* approaches. Scheduling avoids interruptions altogether, with the user at risk of missing an important notification, while mitigation strategies maintain interruptions but try to reduce their disruptiveness or adverse consequences. We are interested in approaches that are deployable in an everyday setting over long periods of time and thus omit approaches that require complex sensing. This applies to solutions like the Finger Ring system [MS05] which allows members of a collocated group to anonymously veto a potentially disruptive incoming phone call to any of its members by pressing a button on a finger ring, or approaches like gaze detection used to infer whether a face-to-face conversion is taking place during an incoming call event [VDS02].

Scheduling Interruptions at the Recipient

One approach to alleviate disruptive smartphone notifications is to intercept potential disruptive events and postpone their delivery until a later moment, when interruptions are thought to be less disruptive. In [FGB11] the authors formulate (5) hypotheses to test whether break points in mobile interaction are appropriate for delivery. To test these, a natural-setting experiment was carried out during 2 weeks over 18 participants using an Android application. The application works by interrupting participants roughly 6 times per day with an SMS-style notification that is equally likely to happen at a random time or right after a call or SMS episode, asking the user to do one of three predefined tasks. Analysis of task acceptance, decision, and completion times lead the authors to the conclusion that scheduling interruptions at the recipient leads to faster interruption acceptance and decision times but that factors such as the social context and cognitive demands of the interruptions play an important role in how these are handled. Moreover, the gathered data reveals that breakpoints in primary activities and endings of mobile episodes do not always collocate — one could receive a call and have to make one right after, for instance. This approach thus entails a further requirement which is to recognise mobile episode breakpoints, a potentially challenging task.

Similarly, in [PM14b] and [OTN14] the authors propose Android publish-subscribe middlewares that aim to predict opportune moments to deliver push notifications. In [PM14b] the authors run an experience sampling experiment for two weeks over 20 subjects where 8 notifications are sent out each day asking the subjects about their sentiment (level of happiness, sadness and boredom) and their interruptibility (not at all, a little, somewhat and very much). From this experiment they show they can often predict the presence of reaction to a notification and their sentiment towards it based on the context (time, accelerometer data, location, presence of company, high-level activity and emotional state) they received it in. It is interesting to note they could not verify the hypothesis formulated in [HI05] that physical activity boundaries are opportune moments to deliver notifications as they could not detect physical activity boundaries using the sensors available on the phone. The authors then test their approach in the wild by running the same experience sampling experiment using a restricted number of context features over a month and 10 subjects, alternating days where samples are gathered using opportune moments as predicted by their proposed InterruptMe library and random sample times. They find that the median user sentiment towards InterruptMe notifications is ‘somewhat’ suitable, whereas it is only ‘a little’ suitable for notifications delivered at random moments. A comparable approach is taken in [OTN14] with a focus on short-term notification scheduling, i.e. finding breakpoints in smartphone usage while a user is actively using their smartphone. The aim is thus to only slightly delay notifications so that they do not appear when a user is writing a Tweet for instance, but right after. The sensed context is thus different from the approach in [PM14b] and revolves around user interface events (e.g. taps and scrolls) and runtime events (eg number of threads running and application switch events). An in the wild study conducted by the authors on 30 participants shows that for a subset of participants cognitive load from notifications was reduced by 33% compared to random notification times as measured by a NASA-TLX questionnaire.

Correctly scheduling notifications to opportune moments has a double benefit. On the side of the recipient, disruptiveness of notifications and increase in cognitive load can be reduced, while on the side of the initiator, i.e. a caller or SMS sender, a faster response can be obtained if their interruption is timed right. This hints at the other interruption scheduling approach, namely, leaving the scheduling of interruptions up to the initiator.

Scheduling Interruptions at the Initiator

Another scheduling strategy consists of providing some of the recipient's contextual information to a potential interruption initiator before they decide to start a mobile interaction episode. This way, the initiator can estimate the disruptiveness of his potential interaction and schedule it for later if need be — this is sometimes referred to as availability sharing.

The experimental studies [AGHK07] (78 participants one-off simulation) and [DGSB07] (13 participants 4 week diary study) confirm that giving simple context information to the initiator, such as coarse location, presence of others nearby and ringer status, can greatly reduce the number of interruptions. A live study [tH07], using a Windows PDA application that asked participants (10 participants over 7 days) questions about their *location* and *presence of company*, combines this context information with machine learning by predicting availability of the recipient. In this example, using machine learning not only removes the interruption initiator's need to extrapolate the recipient's availablity but is is also more respectful of the recipient's privacy as no direct context is shared. A practical example of an availability sharing system, also called InterruptMe, is given in [HRVM11]. It takes into account privacy issues by allowing a user to set which availability features are visible to potential interruptors and enables the user to make groups such as friends, collegues or default, to fine tune privacy settings to different types of people.

This last point underlines the two drawbacks of this type of scheduling. First, initiators are left with the responsibility of predicting the recipient's availability based on potentially incomplete context information. This might be difficult to do, especially if the initiator is not well acquainted with the recipient; in fact, the initiator could choose to ignore the context information entirely for instance in the case of telemarketers. Second, there are serious privacy implications in being able to access someone's context without them knowing about it.

Mitigating Interruptions via Explicit Experience Sampling

Unlike scheduling, mitigation strategies always maintain interruptions but try to reduce their disruptiveness by changing the intensity or mode of notifications (e.g. quieter rings, vibration mode, flashing LED, forwarding notifications to a third party). The challenge in these strategies is learning the context to notification mode mapping for a particular user.

The method commonly known as 'experience sampling' [RDV11, FS11, KC05], which we refer to as **explicit experience sampling** from now on, consists of intentionally interrupting individuals to gather their sentiment or label their current situation or actions. The rationale behind experience sampling is that humans have difficulty predicting context-dependent preferences in advance and are more easily queried online, when experiencing different contexts, to approximate these preferences. We later contrast it with *implicit* sampling approaches.

In [RDV11] the authors propose a disruptive incoming call mitigation approach that consists of automatically silencing a mobile's ringer. They use machine learning to activate their mitigation approach based on context and previous contact interactions. The degrees of freedom in explicit experience sampling lies in the sampling strategy. The authors compare 3 sampling techniques: random, uncertainty-based and their decison-theoretic approach. The advantage of their proposed solution is that they take into account the cost of interruption due to sampling so as to only ask the user to label their interruptibility if the cost of an interruption at that point is time would be more costly than the interruption of asking the label.

Another approach using experience sampling is presented in [FS11] with the difference that the authors also use passive observations, or as we call them *implicit samples*, for the same mitigation approach. The implicit samples they gather are ringer settings (corresponding to the top left switch's state on the iPhone brand smartphone the data is collected on) as well as acceptance or rejection of incoming calls. They take advantage of these to decrease or cancel the number of samples needed for learning. They refer to their technique as density-weighted uncertainty sampling which will query the user, not only based on the level of uncertainty of the current context but also how many additional contexts a sample covers, maximising the utility of interrupting the user.

We argue mitigation offers a broader solution to the disruptive notification problem than scheduling as incoming voice calls cannot be rescheduled, so that incoming call notifications will inevitably require their own approach if scheduling is used for other notifications. That being said, *learning to mitigate disruptive notification signals using explicit experience sampling presents a paradox in that the method creates further interruptions, and is thus susceptible to increase the number of interruptions the user experiences.* Further, for approaches that are meant to be deployed over long periods of time, it does not present a sustainable way of learning as it leads to *experimental fatigue* and can be frustrating for users [KH08].

Mitigating Interruptions via Implicit Experience Sampling

In an ideal world, a context-aware application would know which notifications are disruptive to the user at any given time without any extra effort on their part such as configuring a set of rules or periodically answering a question as in the case of explicit experience sampling. While data must be collected to train the underlying learner of an intelligent context-aware application, implicit experience sampling aims to do so passively, without interrupting the user artificially. It can be seen as an implicit Human-computer Interaction paradigm as described by Schmidt [Sch05].

Implicit Experience Sampling

Implicit experience sampling is the gathering or inferring of labelled context data through the response of a user to naturally occurring events without interrupting them.

The use of implicit experience sampling to mitigate disruptive incoming calls has been explored in [Mar11, MCRL11] where the author uses static logic rules that can be set at any time by the user, combined with inferred rules from past responses to incoming calls to predict whether calls should be answered or rejected. This approach uses a form of inductive logic programming which differentiates itself from previously mentioned approaches as it produces human readable rules that can give insight into the recipient's mobile answering policy. The authors propose their ULearn system as an Android application that combines the user defined rules and learned rules to automatically accept or reject incoming calls – their mitigation strategy. Unfortunately, a survey carried out by the author shows that 79% of the 60 surveyed participants would not use such an application, presumably due to very high cost of false negatives (automatically rejecting a call when it should not have been) and the inconvenience of automatically answering calls.

We follow the lead of the ULearn approach by proposing the RingLearn intelligent context-aware application prototype which aims to mitigate disruptive incoming calls by passively observing the user's past reactions to incoming call events.

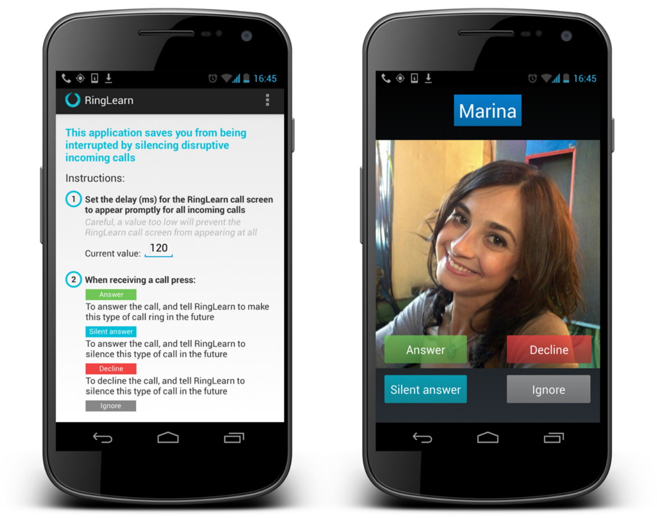
4.2 RingLearn

RingLearn is an intelligent Android application prototype aiming to mitigate disruptive incoming calls in a long-term fashion. A first version was developed to gather context data and incoming call events to test two hypotheses. First, that it is possible to predict when a user will accept or reject incoming calls using data collected via implicit experience sampling. Second, that users change their incoming call acceptance behaviour in a way that makes the use of the forgetting strategies of Chapter 3 beneficial. While we did not iterate over the initial implementation to develop a second version, a post-usage survey of RingLearn over 11 participants highlighted the changes that needed to be made in order to fully achieve the application's intended goal while addressing user experience needs and machine learning requirements.

4.2.1 RingLearn Initial Design

The initial challenge of RingLearn revolves around collecting labelled incoming call events in a practical way for the deployment of the application over an indefinite period of time. We categorise incoming calls into two classes, non-disruptive (class c^+) and disruptive (class c^-). Disruptive calls are the ones where a mitigation strategy would be applied. Currently, when a user receives an incoming call they can choose to pick up, reject or ignore the call by either taking no action or silencing the phone ringer by the press of a button.

It is difficult, perhaps impossible, to differentiate cases where a user consciously ignores an incoming call by taking no action and cases where the user is unaware of the incoming call so that no conclusions were drawn using missed calls and they were discarded from the datasets used in our experiments. Incoming calls that were picked up or rejected where labelled in-situ as part of the application's custom incoming call screen shown in Figure 4.4 which is RingLearn's data collection and labelling mechanism. This custom incoming call screen adds a fourth button to the usual 3: 'Answer', 'Decline' (reject) and 'Ignore' (silence the ringtone but let the call ring through). The incoming call event tree from Figure 4.5 shows the purpose of the new 'Silent answer' button. Answering a call using the 'Answer' button labels it as non-disruptive, signalling the application that these types of calls should not be mitigated in the current context. Declining or ignoring an incoming call using the 'Decline' or 'Ignore' buttons mark the call as disruptive, signalling the application that the notification for these calls should be mitigated in the current context. The 'Silent answer' button covers the case where a disruptive calls is received but a recipient still wishes to answer it without labelling it at non-disruptive. For instance, imagine a situation where someone is at work and their partner calls them, the person might want to step out and answer the call while still labelling the call as disruptive so calls from their partner at work keep on being mitigated in the future without removing their ability to answer such calls. For simplicity, we assumed that the converse case, where someone rejects a call but wants similar calls not to be mitigated in the future, do not happen, and thus did not provide the corresponding button in RingLearn's user interface as shown in the event tree of Figure 4.5 although such cases do exist. These could be described



16:45

RingLearn

This application saves you from being interrupted by silencing disruptive incoming calls

Instructions:

1 Set the delay (ms) for the RingLearn call screen to appear promptly for all incoming calls

Careful, a value too low will prevent the RingLearn call screen from appearing at all

Current value: 120

2 When receiving a call press:

Answer

To answer the call, and tell RingLearn to make this type of call ring in the future

Silent answer

To answer the call, and tell RingLearn to silence this type of call in the future

Decline

To decline the call, and tell RingLearn to silence this type of call in the future

Ignore

Figure 4.4: On the left, we show RingLearn's configuration screen which is launched when the application is opened from the application menu and on the right, we show RingLearn's custom incoming call screen

as *exceptional rejections* such as being in the middle of a task and not wanting to pick up the call at that very moment in time without signalling to the application that the notification for these types of calls should be mitigated.

4.2.2 RingLearn Implementation Details

RingLearn is as an Android 4 application[2]. It works by catching the Android incoming call ring event and overlaying its custom incoming call screen on top of the default one that ships with the phone. Figure 4.4 shows the single parameter to set in the application, namely the overlay delay for the application's custom screen; if the delay is too low, the custom screen will not always successfully overlay itself as an incoming call is signalled by an unordered broadcast event in the Android operating system; if the delay is too high, there will be a noticeable delay in the overlay such that the user might see the default incoming call screen for a few instants and witness the custom call screen overlaying itself. The data, stored in the Android application's local database, was remotely retrieved via the application main screen which offered an upload to server button that uploads the contents of the local database to our server. We note that an option to delete all data is also available to palliate any privacy concerns from users.

[2]The application can be obtained at the following url `https://www.doc.ic.ac.uk/~jeremiah/ringlearn`

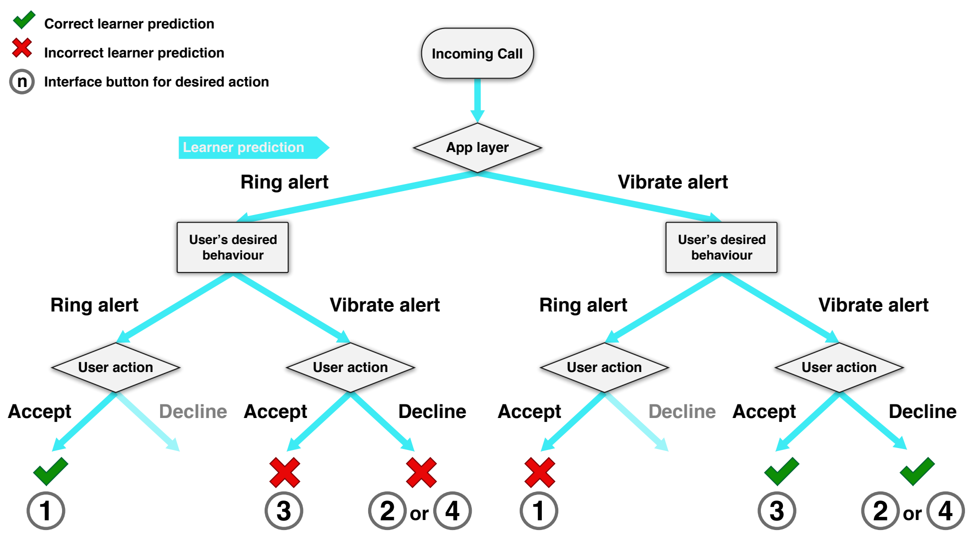


Figure 4.5: A flowchart representing possible events and user actions on an incoming call event; the ‘Answer’ button is labelled 1, the ‘Decline’ button is labelled 2, the ’Silent answer’ button is labelled 3 and the ’Ignore’ button is labelled 4

RingLearn collected features

To choose the features collected by RingLearn we asked participants about the factors contributing to them rejecting a call, shown in Figure 4.6, as part of our initial survey. We restricted ourselves to 7 features that were straightforwardly obtained by using a smartphone’s sensors, namely: the time and day to cover temporal factors; the contact’s number (or lack of) and SSID, cell tower id and GPS information to cover location factors. We omitted features that could not be instantly retrieved so that the decision to silence the ringer can be made fast enough to implement the desired application behaviour without requiring a background process to continuously poll the state of the phone sensors on the phone. We note that 70% of surveyed participants reported using a calendar app on their mobile. In previous work, calendar information was used to determine interruptibility [KC05, RDV11] but was also shown to be an unreliable sensor [LOIP10]. For simplicity reasons we decided not to collect this feature in the application. In this initial experiment, the application was set in data collection mode so that it did not actually silence calls; the participants were given the instruction to carefully choose the button that best corresponded to the type of incoming call they were receiving. After collecting the data, it became clear that GPS was also an unreliable sensor as most users had the functionality turned off as part of their normal smartphone usage, leading to many missing values for those features. That final 6 features used for learning where:

- The current month
- The current day of the week
- The current time
- The incoming call number
- The cell tower id the recipient is connected to
- The WIFI SSID the recipient is connected to

RingLearn Deployment and Experimental Findings

To keep the data collection experiment as close as possible to a real use case where an unknown user can download and decide to use or not a disruptive incoming call mitigation app, we asked a small group of students to install RingLearn without setting an end time for the experiment. This left them the choice to uninstall the app at any given time. After 16 weeks participants were contacted and asked whether the app was still installed and to take a post-usage survey. The only technical difficulty encountered was on HTC branded devices where it was impossible to programatically launch the custom incoming screen making RingLean useless on those devices. We ended up gathering data from 11 participants over 16 weeks. The collected data showed that many of the participants received very few incoming calls during the 16 week period (294 was the maximum) so that we only kept the 3 largest datasets for the data analysis and learner benchmarking of Section 4.3.

The post usage survey of the application[3] over the 11 participants confirmed that our data collection and labelling method was indeed user friendly as illustrated in Figure 4.7. 7 out of 11 (63.6%) participants stated they would use a version of the RingLearn app that would actually

[3]Link to the RingLearn post usage survey: `https://docs.google.com/forms/d/1zsob1VPAPRh-_1er4ud9vRNeWsbt0TRSefBaMLNfTo0/viewanalytics`

The day of the week	**8**	12.3%
The time of the day	**30**	46.2%
The battery level of your mobile	**9**	13.8%
Your mobile is charging	**1**	1.5%
Your location	**42**	64.6%
The physically activity you are doing when receiving the call	**45**	69.2%
The mobile app you are using when receiving the call	**7**	10.8%
You do not wish to speak to the caller, ever	**27**	41.5%
You are next to a person that you do not want to answer the call in front of	**37**	56.9%
You are at a location at which you do not want to answer the call from	**44**	67.7%
It is a specific time and/or date for which you do not wish to speak to the caller	**22**	33.8%
The call is from a private number	**24**	36.9%

Which of these factors have had an influence on your decision to reject a call

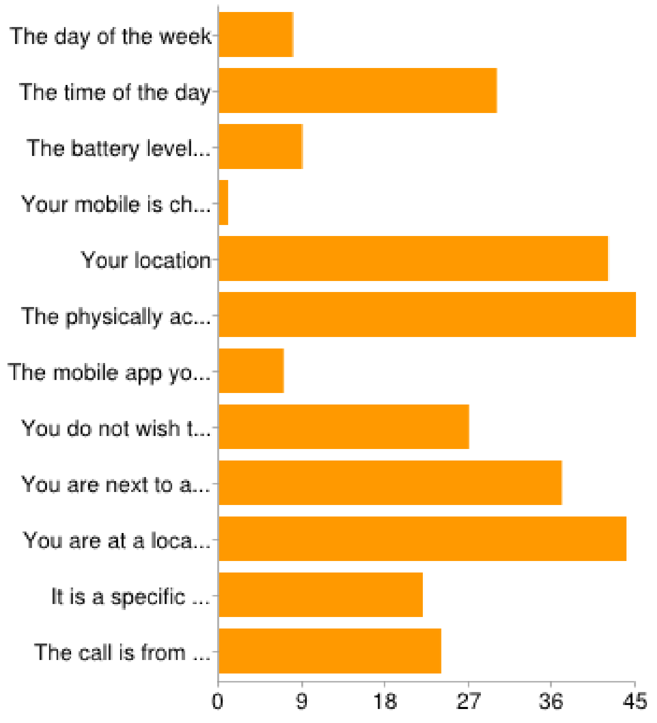


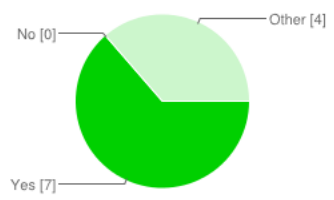
Figure 4.6: Summary of the factors leading to incoming call rejection question and answers in our mobile phone usage survey conducted over 65 participants.

take actions when a disruptive call is predicted, while no participants declared they would not use such an app, the 4 remaining participants asked for modifications to the interface which we discuss below. 9 out of 11 (81.8%) stated that they understood, and were not confused by, the presence of the new 'Silent answer' button on the incoming call screen interface. Finally, 9 out of 11 (81.8%) participants did not find the application difficult to configure and 7 out of 11 (63.6%) found that RingLearn did not add any noticeable extra cognitive effort compared with the default incoming call screen. The most important point that came out of the survey was that it is not always possible for a user to know whether an incoming call is disruptive before answering it, sometimes, factors such as hidden or unknown numbers can change the ground truth once the call is over.

Taking into account user feedback from the post usage survey and iterating over our initial approach, we found that the best solution to perform practical disruptive incoming call mitigation is the **infer-and-confirm strategy** shown in Figure 4.8. The infer-and-confirm approach relies on the same principle as the initially designed solution, namely inferring a label from the user's action after a notification is received (after the 'user responds to notification step'), but it also temporarily informs the user of the collected label once the user has stopped interacting with the notification content (when a user puts down the phone in case of incoming calls or quits a messaging app in case of message-based communication) user can then amend ('display inferred label to user' and 'user relabels if needed' steps). This last extra step will mostly be passive if a system is able to predict disruptive notifications requiring no extra action on the part of the user compared to our initial design, yet allowing for a user to correct their initial label in-situ if needed.

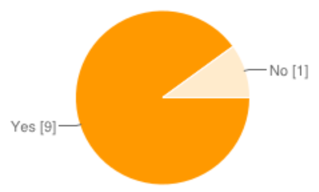
This shows the extent to which the collection and labelling of data required to use machine learning might influence the user interface of a context-aware application and shows well that the HCI component and learning component of an application cannot be dissociated as they exert mutual requirements on each other. While we propose the infer-and-confirm approach for incoming calls, the approach can be applied to other notifications. Taking the example of push notifications, their disruptiveness could be inferred by looking at how long a user takes to attend to them and which action they take when acting on them: swiping away or opening

With the same interface you have been using so far, if the RingLearn application would actually take actions i.e. silence a call's ringtone when recognised as disruptive, would you use it?



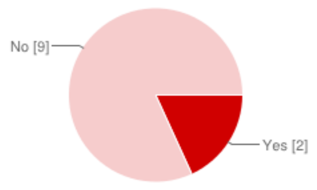
Yes	**7**	63.6%
No	**0**	0%
Other	**4**	36.4%

Did you understand the purpose of the (turquoise) 'silent answer' button on the incoming call screen?



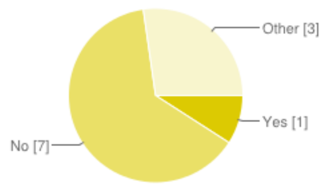
Yes	**9**	81.8%
No	**1**	9.1%

Did you find the presence of the 'silent answer' button confusing or bothering?



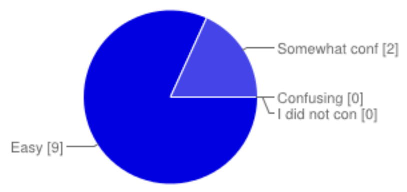
Yes	**2**	18.2%
No	**9**	81.8%

Except for the configuration step, did the RingLearn application require any noticeable extra cognitive effort vs using the default incoming call screen?



Yes	**1**	9.1%
No	**7**	63.6%
Other	**3**	27.3%

How difficult was the application to configure (adjust the custom call screen overlay delay)?



Easy	**9**	81.8%
Somewhat confusing	**2**	18.2%
Confusing	**0**	0%
I did not configure it	**0**	0%

Figure 4.7: Summary of the RingLearn post usage survey questions and answers in our mobile phone usage survey conducted over 65 participants.

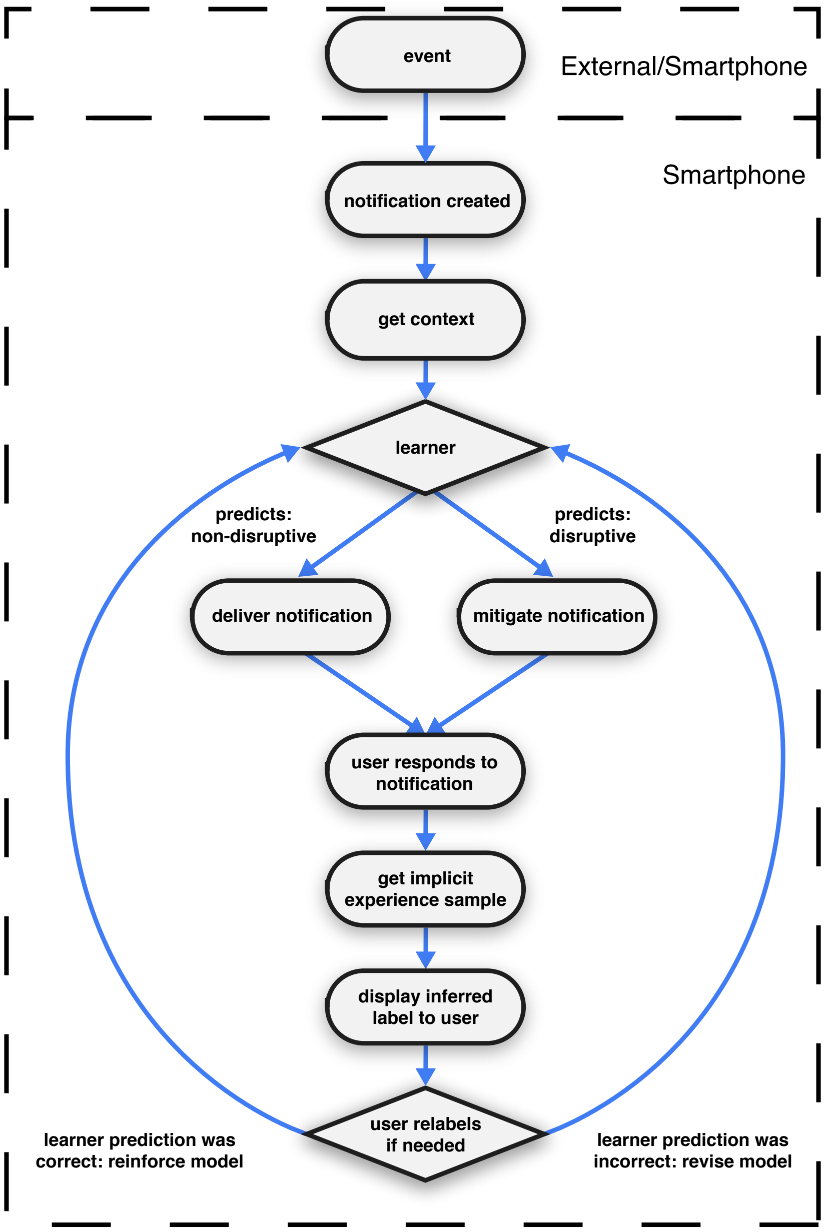

event
External/Smartphone
Smartphone
notification created
get context
learner
predicts: non-disruptive
predicts: disruptive
deliver notification
mitigate notification
user responds to notification
get implicit experience sample
display inferred label to user
user relabels if needed
learner prediction was correct: reinforce model
learner prediction was incorrect: revise model

Figure 4.8: An illustration of our incoming call mitigation strategy.

the corresponding app. If they open the corresponding app, one could consider the amount of time they stay in it and the further action(s) the user takes. These could be used to infer a label using a set of preprogrammed rules and once a label is inferred the user can be notified via a dialog, allowing him to amend it if necessary as part of the label confirmation step.

4.3 Experiments

We now detail the experiments we ran to test the two hypotheses presented in Section 4.2: that it is possible to predict whether a user accepts or rejects incoming calls using data collected via implicit experience sampling and that users' call acceptance behaviours change as time passes. We selected 6 incoming call datasets as detailed below and conversely to Section 3.4, for a given set, we benchmarked forgetting strategies over multiple A_w measures. This means we are interested in each learner's performance over the testing set under different user preferences parameters w, instead of showing the evolution of each strategy's performance over time under a fixed w. This is because we cannot know an individual's w parameter a priori, an optimal forgetting strategy should perform well on average over any w parameter.

In the previous section, we presented the RingLearn Android application that was used to gather data over 11 participants during 16 weeks. The relatively small sizes of the gathered datasets forced us to keep only the three largest ones and we chose to complement them with 3 further datasets coming from the Cambridge Device Analyzer corpus.

4.3.1 The Cambridge Device Analyzer Corpus

The Cambridge Device Analyzer corpus [WRB14] consists of anonymised smartphone usage traces from more than 23000 users. The data is collected through a downloadable Android app [fA15] that silently logs over 300 Android system events[4] for each user (the logged events can be restricted by the user). The corpus was chosen because it is has many more datasets

[4]Full list available here `http://deviceanalyzer.cl.cam.ac.uk/keyValuePairs.htm`

than the RingLearn corpus and many of them span much longer periods of time — some over 2 years — which presents a clear advantage over RingLearn. That being said, there were two drawbacks when using Device Analyzer datasets for our purpose. First, because the datasets are raw event logs, context cannot directly be gathered from the data as in RingLearn datasets and has to be reconstituted from the said logs which are several gigabytes in size for datasets of interest. Second, there are no labels for incoming calls, it is thus impossible to know the ground truth when trying to categorise unintentionally missed or ignored calls versus explicitly declined (rejected) calls. Assumptions thus have to be made to infer data labels which potentially lead to noisy labels, a problem which could be offset if the Device Analyzer application used the infer-and-confirm strategy presented in Section 4.2.

To convert raw Device Analyzer data into standard feature and label format, we thus reconstituted context across time by linearly going through a dataset's event logs from top (earliest recorded event) to bottom (latest recorded events), each line corresponding to a single system event, reconstructing the context corresponding to every incoming call event. The events we collected were from the *phone, screen, conn* and *wifi* keys yielding 11 features for each incoming phone call with an extra feature not present during learning used to label incoming calls (incoming call ring time). The 11 initial features are similar to RingLearn's features augmented with phone usage markers such as the amount of time a phone hasn't been used as measured by the amount of time the screen has been off, as these have shown to be good predictors for notification disruptiveness [Pie14, PdOKO14]. For each call, we thus initially extracted 11 features as follows:

- The current month
- The current day of the week
- The current time
- The incoming call number
- The cell tower Id the recipient is connected to

- The cell tower's location area code
- The WIFI network name (if connected)
- Is the phone's screen on or off when the call was received
- The amount of time since the phone's screen was last on (0 if already on)
- If the screen is on, the number of seconds it has been on for (0 if it's off)
- The number of seconds since the user's previous call

Labelling Device Analyser Datasets

Having selected the context, the next step was to label the data. Using the logged Android system events *phone|OFFHOOK*, *phone|RINGING* and *phone|IDLE*, we labelled datapoints as follows: answered calls were labelled as *non-disruptive* while declined calls where the user consciously chose not to answer the call by pressing the decline button, were labelled as *disruptive*. Answered calls could be recognised due to a *phone|OFFHOOK* event taking place in between a *phone|RINGING* and *phone|IDLE* event. On the other hand, missed calls, when the user either ignored or was unaware of the call happening, were initially indistinguishable from declined calls as they were both characterised by a *phone|RINGING* followed by a *phone|IDLE* event. To separate the two cases, we modelled the distribution of ringing times for both answered and unanswered calls using a mixture of respectively a one and two Gaussian component distribution, an example is shown in Figure 4.9.

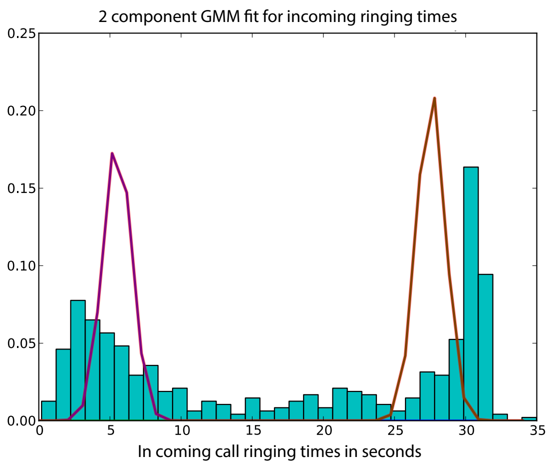
We then labelled unanswered calls as disruptive if they fell within 2 standard deviations of the answered call ringing time distribution mean, making the assumption that the user would take the same amount of time to press the answer button as the decline button and ignoring caller abandons (the caller drops a call before either reaching the user or their voicemail). We note that unanswered calls could have been assumed to come from a 3 component gaussian distribution: declined, missed and caller abandon (where the caller hangs up after x number of rings) classes but found that the Bayesian Information Criterion was always higher when using 3 classes.

After converting a number of Device Analyser datasets to feature and label format we selected 3 that had a high number of incoming call events. The datasets are anonymised and we refer to them using the 4 first hexadecimal characters of their name 784c, b9ae and c260 and appending a -DA marker to differentiate them from RingLearn dataset. We were then interested in knowing which features where the most discriminanting so we applied Recursive Feature Elimination with a random forest as described in [GFG06]. On these 3 datasets, the 5 features listed below, ranked in term of discriminative power, gave virtually the same performance as using the original 11 features and are the ones we used in our experiments:

- The number of seconds since the user's previous call
- The current time
- The amount of time since the phone's screen was last on (0 if aready on)
- The incoming call number
- The cell tower Id the recipient is connected to

Datasets

The properties of the 3 RingLearn and 3 Device Analyzer datasets used to benchmark forgetting strategies over a range of w parameters are summarised in Table 4.1. Due to a shorter data collection period, RingLearn dataset are much smaller than Device Analyzer datasets — from 193 to 294 points. The concept drift analysis algorithm from Section 3.1 reveals that the class flip measure are moderate on these datasets, between 6.3% and 18.3%, indicating that users did somewhat change what they considered to be disruptive incoming calls in a way that significantly contradicted preceding instances. The relatively high concept drift measure reveals that the 3 epochs used to analyse datasets had disruptive incoming call patterns that



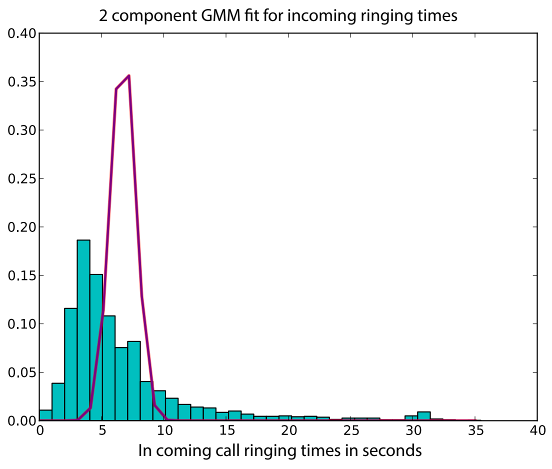


Figure 4.9: For dataset c260-DA: the figure on the left shows the histogram of incoming call ring times and a 2 component Gaussian distribution fit for modelling answered versus missed calls ringing times. The figure on the right, shows a 1 component Gaussian distribution modelling answered incoming calls times only and is used in the data labelling procedure for Device Analyzer datasets.

Set name	Set Size	Time span	Num. feats.	Class balance	Data type	f param.	Grid Size	Drift measure	Class flip measure
L-RingLearn	193	16w	6	0.24/0.76	real	3	729	61.6%	8.8%
F-RingLearn	236	16w	6	0.13/0.87	real	3	729	59.7%	6.3%
J-RingLearn	294	16w	6	0.27/0.73	real	3	729	63.2%	18.3%
784c-DA	871	19w4d	5	0.31/0.69	real	4	1024	40.1%	3.8%
b9ae-DA	1256	1y7m9d	5	0.26/0.74	real	4	1024	42.9%	36.8%
c260-DA	2047	1y5d	5	0.30/0.70	real	4	1024	37.3%	35.5%

Table 4.1: Summary of the 6 incoming call datasets used in our experiments. Concept drift analysis (right-hand side of table) was made as described in Section 3.1

were unique to them, which can be explained by the small size of the datasets. The first Device Analyzer dataset, 784c-DA, while being larger than RingLearn datasets (871 instances), also has the lowest class flip measure: 3.8%, foreshadowing that forgetting strategies will perform close to online retraining. The remaining two Device Analyzer datasets are not only larger (1256 and 2047 instances respectively) but span a longer period of time, over 1 year each, leading to higher class flip measures (36.8% and 35.5% respectively) as expected. These two datasets provide an ideal real use case to benchmark our proposed forgetting strategies and support the hypothesis that in some cases there is a need to forget a subset of old patterns in mobile context-aware applications deployed over long periods of time.

4.3.2 Experimental Setup and Implementation Details

We use the same experimental setup as in Section in 3.4, with three exceptions. First, instead of tracking the evolution of learner performance over time using a fixed w user preference parameter, we are interested in how well learners perform under different user preference scenarios. We thus benchmark learners over 6 w parameters: $w = 0$, $w = 0.5$, $w = 1$, $w = 2$, $w = 3$ and $w = 4$, corresponding to cases where the cost of failing to correctly predict disruptive calls is increasingly higher than for non-disruptive calls, and compute learners' mean performances and standard errors over all w; the learner with highest mean weighted accuracy A_w is considered the best as it can be seen as the most versatile, a default choice for cases where a user's w is unknown — as it is likely to be the case in practice. Second, the training set ratio was set to 20% of the total dataset for RingLearn datasets and 33% for Device Analyzer datasets. Finally, when two learners perform similarly, we use McNemar tests [Alp10] with $\alpha = 0.5$ to assess whether they perform statistically significantly differently or not.

McNemar Test

Given a training set, from the predictions of two learners l_1 and l_2 on the same testing set, the following contingency table can be constructed:

a : number of test instances misclassified by l_1 **and** l_2	b : number of test instances correctly predicted by l_2 **but not by** l_1
c : number of test instances correctly predicted by l_1 **but not by** l_2	d : number of test instances correctly predicted by l_2 **and** l_1

The McNemar test is a statistical test that assumes the following null hypothesis: l_1 and l_2 have the same error rate at significance level α if $\chi_1^2 = \frac{(|b-c|-1)^2}{b+c} \leq \chi_{\alpha,1}^2$ where $\chi_{\alpha,1}^2$ is the chi-square statistic with degree of freedom 1. Informally, the McNemar test makes the hypothesis that l_1 and l_2 have similar error rate i.e. that they perform equally well, and that the observed difference in performance for l_1 and l_2 on a particular experiment is only due to learners' natural variability in predictions over specific datasets. If the computed χ_1^2 is greater than $\chi_{\alpha,1}^2$ the test suggests that it is unlikely that the hypothesis is valid so that the opposite hypothesis, that l_1 and l_2 do not perform equally well in general, is assumed valid by default. In the experiments below we use $\chi^2_{0.05,1} = 3.481$ and apply the test for the sequence of predictions over all 6 w parameter values we consider.

Concept drift analysis was again taken over 3 epochs, although larger grid sizes than in Chapter 3 had to be used for the measures to converge due to denser datasets. On each Device Analyzer dataset, we also ran an extra experiment using a different base learner: the Classification and Regression Trees provided by the sci-kit learn module, a non-parametric learner very similar to the popular C4.5 algorithm [sklc15b]. The extra experiments were used to investigate how different base learners might affect the performance of forgetting strategies and confirm that our proposed forgetting strategies are indeed learner independent.

4.3.3 Results

The first four datasets: L-RingLearn, F-RingLearn, J-RingLearn and 784c-DA, have relatively high concept drift measures and moderate to low class flip measures (61.6%, 59.7%, 63.2%, 40.1%, and 8.8%, 6.3%, 18.3%, 3.8% respectively) as shown in Table 4.1. Thus, we might expect that offline learning will perform significantly worse than online retraining and that forgetting strategies will provide some benefit over online retraining as indicated by the class flip measure. In practice we find slightly different patterns.

L-RingLearn Results

L-RingLearn shown in Figure 4.10 with performances summarised in Table 4.2 is the smallest of all considered datasets. Surprisingly, mean A_w for offline and online retraining only have a 0.5% performance gap between them (mean $A_w = 0.7304$ for offline and mean $A_w = 0.7342$ for online retraining) although the dataset's concept drift measure is 61.6%. The McNemar test reveals an interesting fact, although offline and online retraining have very similar performance, the actual instances they misclassify are significantly different. Further, with the exception of

DIOC, the McNemar tests show that forgetting strategies perform similarly to online retraining with a performance difference range of -1.9% for DIOC to 2.3% for TC (mean $A_w = 0.7203$ and mean $A_w = 0.7516$ respectively) which is surprising given the dataset's 8.8% class flip measure. We attribute these unexpected patterns to the small dataset size and/or inconsistencies of incoming call rejection patterns in the data. As a result, learning is rather unsuccessful as a whole, with a maximal mean learner performance gain of only 8.3% compared to making naive c^- predictions i.e. always predicting the non-disruptive call class regardless of input. This is can further be seen by the proximity of learner and naive performance curves in Figure 4.10.

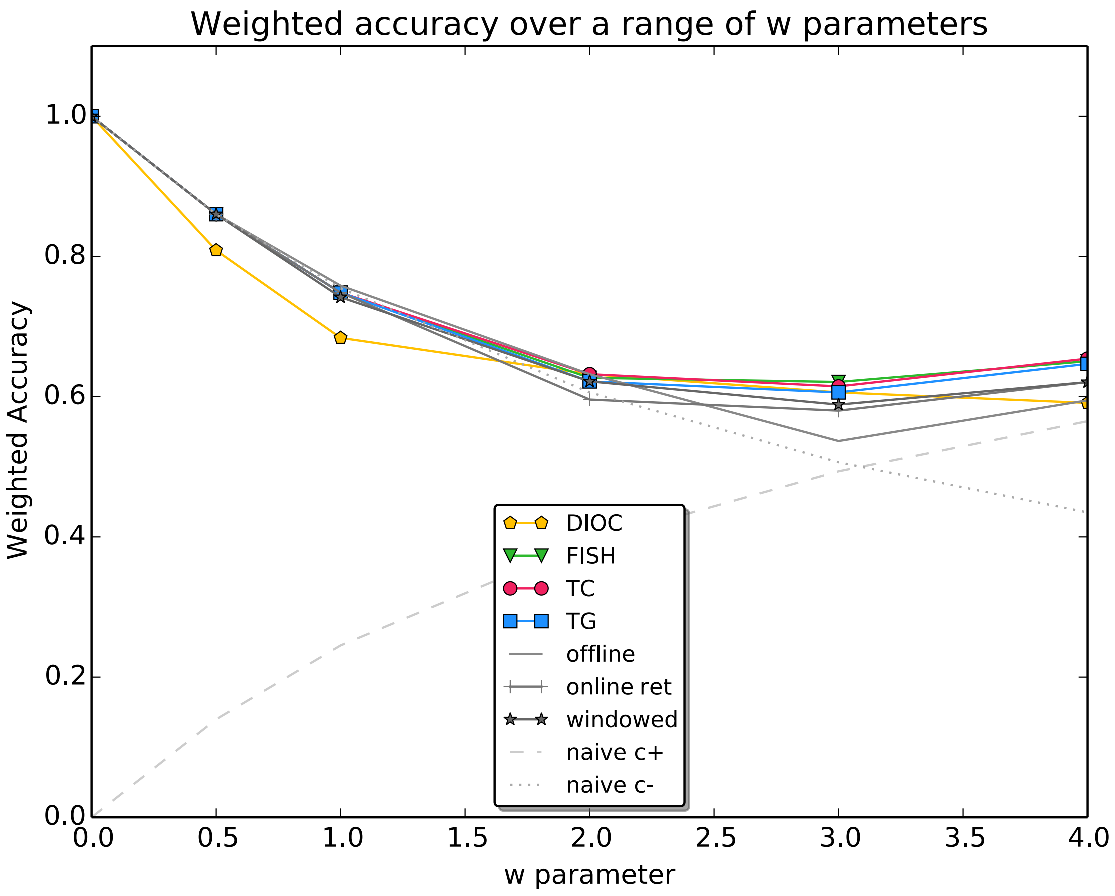


Figure 4.10: Plot of forgetting strategy performances A_w on the L-RingLearn testing set for different values of the user preference parameter $w = \frac{c^+ \text{ weight}}{c^- \text{ weight}}$. c^+ denotes the disruptive call class and c^- the non-disruptive call class. The baseline performances naive c^+ and naive c^- indicate the performance attained when always predicting the corresponding classes. Individual forgetting strategy hyperparameters for each w are reported in Appendix A.3

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4	mean A_w	SE
TC	1	0.8603	0.7484	0.6321	0.6147	0.6543	0.7516	0.062
FISH	1	0.8603	0.7484	0.6269	0.6210	0.6506	0.7512	0.062
TG	1	0.8603	0.7484	0.6218	0.6061	0.6468	0.7472	0.063
Windowed	1	0.8603	0.7419	0.6218	0.5887	0.6208	0.7389	0.066
Online ret.	1	0.8603	0.7484	0.5959	0.5801	0.6208	0.7342	0.068
Offline	1	0.8603	0.7584	0.6321	0.5368	0.5948	0.7304	0.072
DIOC	1	0.8088	0.6839	0.6321	0.6061	0.5911	0.7203	0.064
Naive c^-	1	0.8603	0.7548	0.6062	0.5065	0.4349	0.6937	0.088
Naive c^+	0	0.1397	0.2452	0.3938	0.4935	0.5651	0.3062	0.088

Table 4.2: Summary of forgetting strategy performances on the *L-RingLearn* testing set for a range of w parameters. Strategies are ordered in decreasing order of mean weighted accuracy.

F-RingLearn Results

F-RingLearn shown in Figure 4.11 with performances summarised in Table 4.3 has similar characteristics to L-Ringlearn. Online retraining performs slightly better than offline learning, with a mean A_w gap of 3.7% (mean $A_w = 0.7712$ for offline and mean $A_w = 0.8011$ for online retraining) and the McNemar test results again show that the difference is significant. The McNemar tests show that forgetting strategies perform similarly to online retraining with performances gains ranging from -2.3% for DIOC to 4.2% for TC (mean $A_w = 0.7829$ and mean $A_w = 0.8350$ respectively) which is unexpected given its 6.3% class flip measure. It is interesting to note that F-RingLean is the only dataset where a naive learner outperforms online retraining, naive c^- which has mean $A_w = 0.8243$ performing 2.8% better than online retraining. This means predicting every call to be non-disruptive will yield higher performance than making predictions using online retraining. The maximal mean performance gain compared to naive c^- is 1.2%, again indicating that learning in general was not significantly beneficial for the same potential reasons as in L-RingLearn.

J-RingLearn Results

J-RingLearn shown in Figure 4.12 with performances summarised in Table 4.4 is the the largest RingLearn dataset. It has a concept drift measure of 63.2% and a mean A_w gap of 8.1% between online retraining and offline learning (mean $A_w = 0.6649$ for offline and mean $A_w = 0.7190$

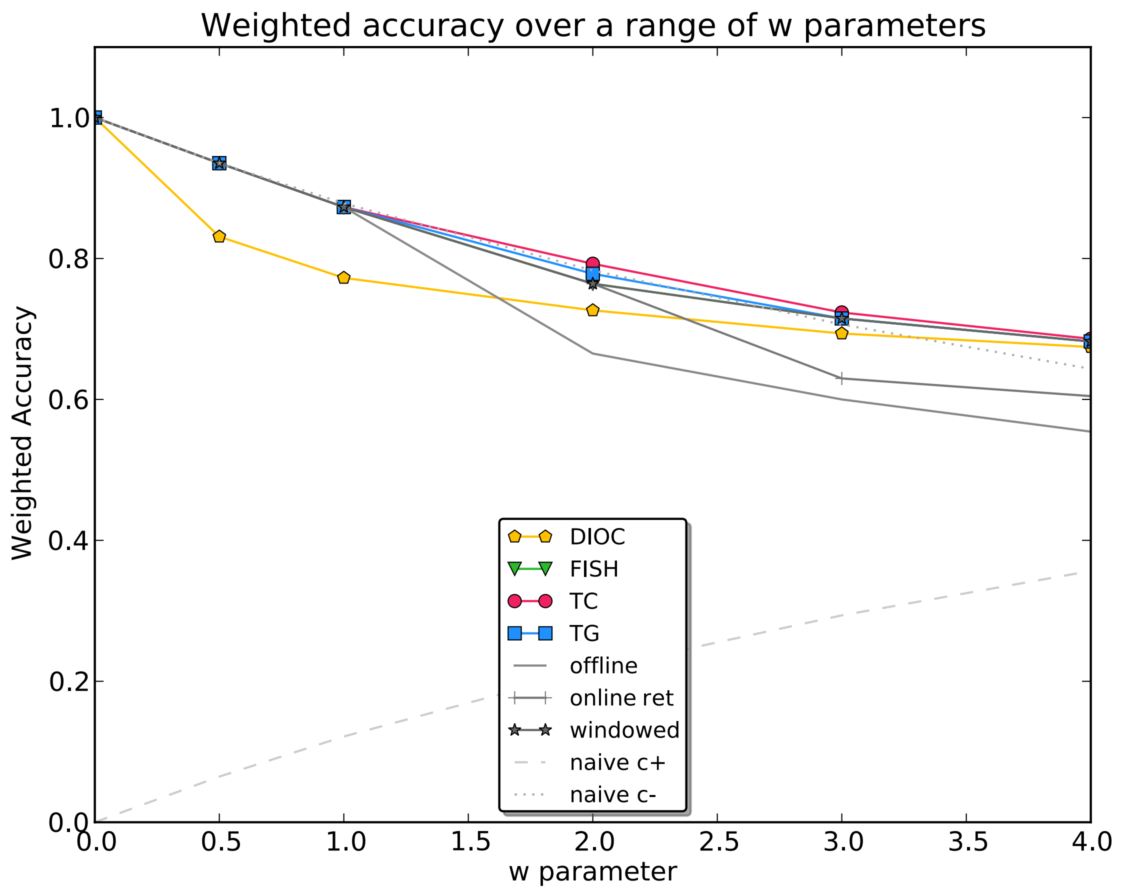


Figure 4.11: Plot of forgetting strategy performances A_w on the F-RingLearn testing set for different values of the user preference parameter $w = \frac{c^+ \text{ weight}}{c^- \text{ weight}}$. c^+ denotes the disruptive call class and c^- the non-disruptive call class. The baseline performances naive c^+ and naive c^- indicate the performance attained when always predicting the corresponding classes. Individual forgetting strategy hyperparameters for each w are reported in Appendix A.3

for online retraining). The McNemar test confirms online retraining is a significantly better strategy than offline learning. Further McNemar tests comparing forgetting strategies to online retraining show that forgetting strategies perform significantly better, with performance gaps higher than for previous datasets ranging from 2.6% for DIOC to 6.7% for FISH (mean $A_w = 0.7379$ and mean $A_w = 0.7674$ respectively) which is inline with our expectations due to the dataset's 18.3% class flip measure. The maximal mean performance gain over naive predictions is 12.6% with naive c^- mean $A_w = 0.6814$ which shows using machine learning is beneficial. That being said, offline learning performs 2.5% worse than naive c^- underlining the importance of using adaptive learning and choosing the right forgetting strategy.

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4	mean A_w	SE
TC	1	0.9352	0.8730	0.7925	0.7234	0.6860	0.8350	0.050
TG	1	0.9352	0.8730	0.7783	0.7149	0.6822	0.8306	0.051
FISH	1	0.9352	0.8730	0.7642	0.7149	0.6822	0.8282	0.052
Windowed	1	0.9352	0.8730	0.7642	0.7149	0.6822	0.8282	0.052
Naive c^-	1	0.9352	0.8730	0.7830	0.7064	0.6434	0.8243	0.055
Online ret.	1	0.9352	0.8730	0.7642	0.6298	0.6047	0.8011	0.066
DIOC	1	0.8310	0.7725	0.7264	0.6936	0.6744	0.7829	0.049
Offline	1	0.9352	0.8730	0.6651	0.6000	0.5543	0.7712	0.076
Naive c^+	0	0.0648	0.1217	0.217	0.2936	0.3566	0.1756	0.056

Table 4.3: Summary of forgetting strategy performances on the *F-RingLearn* testing set for a range of w parameters. Strategies are ordered in decreasing order of mean weighted accuracy.

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4	mean A_w	SE
FISH	1	0.8491	0.7585	0.7003	0.6620	0.6348	0.7674	0.055
TC	1	0.8443	0.7542	0.6734	0.6732	0.6539	0.7665	0.055
TG	1	0.8540	0.7500	0.6667	0.6257	0.6277	0.7540	0.060
Windowed	1	0.8345	0.7373	0.6667	0.6257	0.6038	0.7446	0.061
DIOC	1	0.8273	0.7034	0.6397	0.6145	0.6425	0.7379	0.061
Online ret.	1	0.8345	0.7458	0.6027	0.5587	0.5728	0.7190	0.071
Naive c^-	1	0.8516	0.7415	0.5892	0.4888	0.4177	0.6814	0.091
Offline	1	0.8540	0.6186	0.4714	0.4944	0.5513	0.6649	0.087
Naive c^+	0	0.1484	0.2585	0.4108	0.5112	0.5823	0.3185	0.091

Table 4.4: Summary of forgetting strategy performances on the *J-RingLearn* testing set for a range of w parameters. Strategies are ordered in decreasing order of mean weighted accuracy.

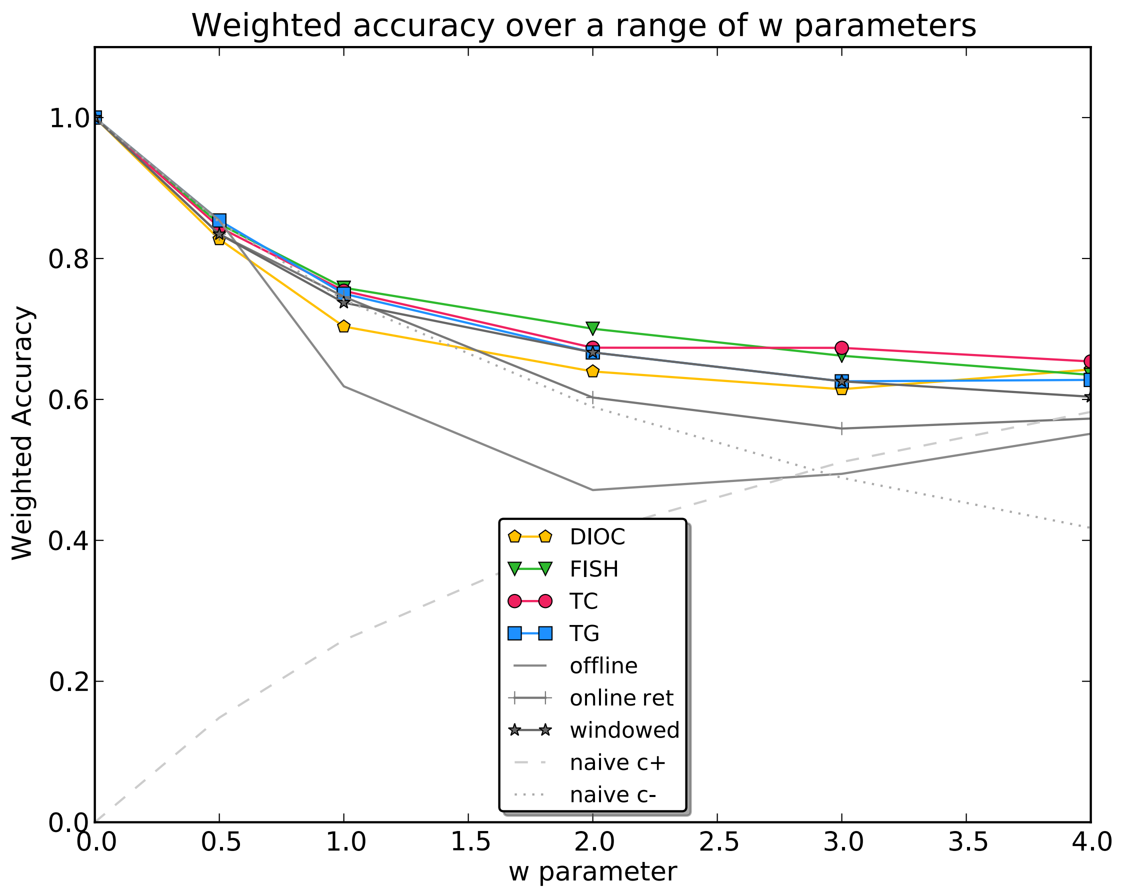


Figure 4.12: Plot of forgetting strategy performances A_w on the J-RingLearn testing set for different values of the user preference parameter $w = \frac{c^+ \text{ weight}}{c^- \text{ weight}}$. c^+ denotes the disruptive call class and c^- the non-disruptive call class. The baseline performances naive c^+ and naive c^- indicate the performance attained when always predicting the corresponding classes. Individual forgetting strategy hyperparameters for each w are reported in Appendix A.3

784c-DA Results

784c-DA shown in Figures 4.13 and 4.16 with performances summarised in Tables 4.5 and 4.8 for Logistic Regression and Decision Tree base learners respectively, is the first Device Analyzer dataset we consider. For Device Analyser datasets we are also interested in seeing which patterns stay true across learners by comparing experimental results across the two base learners. 784c-DA is larger than the RingLearn datasets, 871 points versus 193, 236 and 294 points which might explain the larger maximal mean performance gain of learners compared to naive predictions. The performance increase is consistent across base learners: 21.7% for TG

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4	mean A_w	SE
TG	1	0.8173	0.8173	0.6207	0.6365	0.6749	0.7433	0.059
TC	1	0.8132	0.7055	0.6232	0.6303	0.6757	0.7413	0.058
FISH	1	0.8091	0.7072	0.6079	0.6324	0.6511	0.7346	0.060
Windowed	1	0.8091	0.7038	0.6105	0.6110	0.6655	0.7333	0.061
DIOC	1	0.8122	0.7072	0.6041	0.5967	0.6757	0.7326	0.062
Online ret.	1	0.8050	0.6969	0.5977	0.6090	0.6613	0.7283	0.062
Offline	1	0.7920	0.6712	0.5734	0.5448	0.6224	0.7005	0.069
Naive c^-	1	0.7946	0.6592	0.4917	0.3912	0.3260	0.6106	0.105
Naive c^+	0	0.2054	0.3408	0.5083	0.6079	0.6740	0.3894	0.104

Table 4.5: Summary of forgetting strategy performances on the *784c-DA* testing set for a range of w parameters. Strategies are ordered in decreasing order of mean weighted accuracy.

using Logistic Regression and 18.8% for FISH using a Decision Tree (mean $A_w = 0.7433$ and mean $A_w = 0.7260$ respectively compared to naive c^- mean $A_w = 0.6106$. On the contrary, the gap between online retraining and offline learning is not. 784c-DA's concept drift measure is 40.1% and while using Logistic Regression produces a mean A_w gap of 3.9% (mean $A_w = 0.7005$ for offline and mean $A_w = 0.7283$ for online retraining) the gap is of 10.2% when using a Decision Tree base learner (mean $A_w = 0.6426$ for offline and mean $A_w = 0.7086$ for online retraining). In both cases, McNemar tests reveal once again that online retraining is significantly better than offline learning. However, the performance gap between online retraining and forgetting strategies is consistent, it is low, foreshadowed by the dataset's low class flip measure of 3.8%. When using Logistic Regression as a base learner, it ranges from 0.5% for DIOC to 2.0% for TG (with mean $A_w = 0.7326$ and mean $A_w = 0.7433$ respectively) and when using Decision Trees it ranges from 0.8% for windowed learning to 2.6% for FISH (with mean $A_w = 0.7145$ and mean $A_w = 0.7260$ respectively). McNemar tests show that the null hypothesis cannot be rejected when comparing online retraining to forgetting strategies on these two sets i.e. there is no significant effect in using forgetting strategies on this particular dataset so that the optimal forgetting strategy is to not forget anything. This is inline with the low class flip measure. It is also interesting to note the overall weaker performance of strategies using Decision Tree base learners as they are usually thought of as a more sophisticated base learner than Logistic Regression.

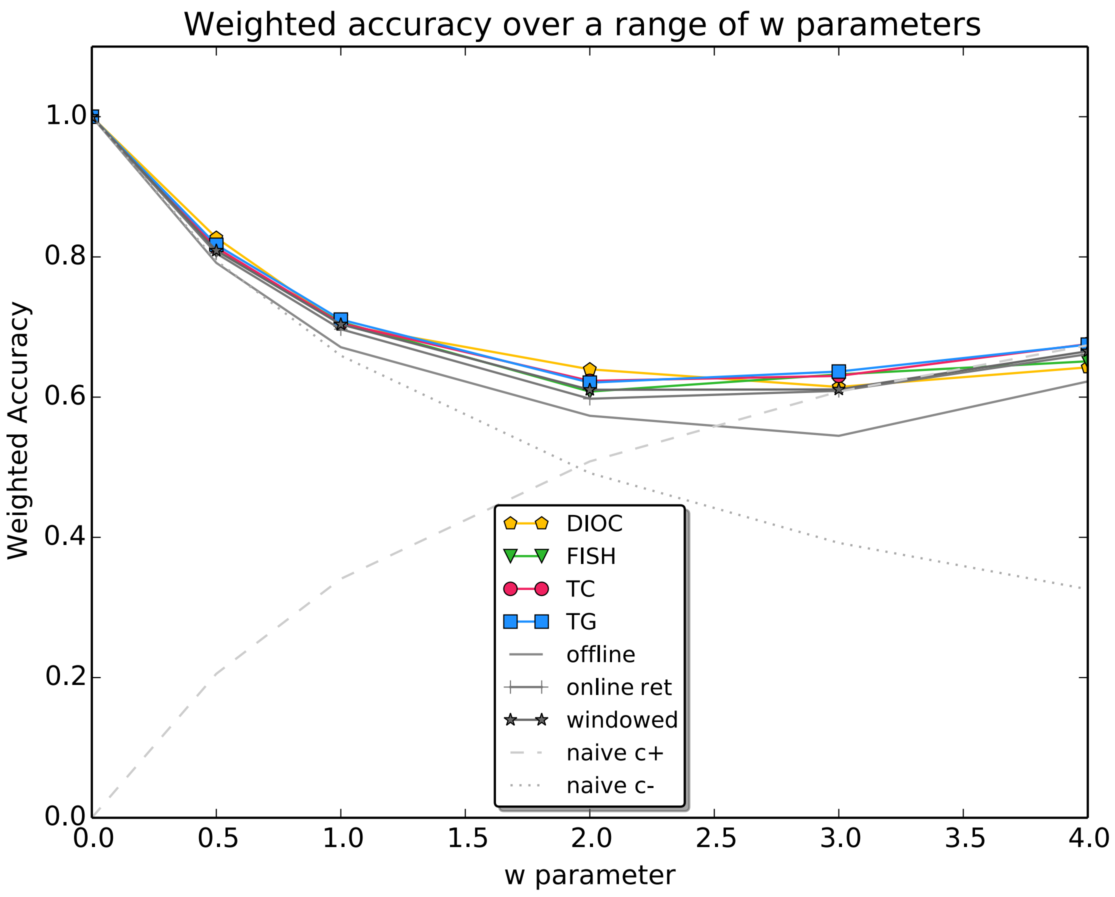


Figure 4.13: Plot of forgetting strategy performances A_w on the 784c-DA testing set for different values of the user preference parameter $w = \frac{c^+ \text{ weight}}{c^- \text{ weight}}$. c^+ denotes the disruptive call class and c^- the non-disruptive call class. The baseline performances naive c^+ and naive c^- indicate the performance attained when always predicting the corresponding classes. Individual forgetting strategy hyperparameters for each w are reported in Appendix A.3

b9ae-DA Results

b9ae-DA shown in Figures 4.14 and 4.17 with performances summarised in Tables 4.6 and 4.9 for Logistic Regression and Decision Tree base learners respectively, is the dataset that spans the longest period of time, over 1 year and 7 months for 1256 points. This leads to a consistent benefit in using machine learning across base learners: a 25.4% increase for TC using Logistic Regression and 17.4% for TG using a Decision Tree (mean $A_w = 0.8057$ and mean $A_w = 0.7545$ respectively compared to naive c^- mean $A_w = 0.6422$). b9ae-DA's concept drift measure is 42.9% and the performance gap between online retraining and offline learning is

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4	mean A_w	SE
TC	1	0.8307	0.7696	0.7338	0.7470	0.7537	0.8057	0.041
FISH	1	0.8300	0.7482	0.7165	0.7241	0.7411	0.7933	0.044
TG	1	0.8286	0.7447	0.6964	0.7034	0.7362	0.7848	0.047
Windowed	1	0.8286	0.7447	0.6964	0.7034	0.7362	0.7848	0.047
DIOC	1	0.8286	0.7411	0.6700	0.6538	0.6521	0.7576	0.055
Online ret.	1	0.8265	0.7233	0.6335	0.6405	0.6889	0.7521	0.057
Offline	1	0.8188	0.7043	0.5843	0.5555	0.5233	0.6977	0.075
Naive c^-	1	0.8216	0.6971	0.5351	0.4342	0.3653	0.6422	0.099
Naive c^+	0	0.1784	0.3029	0.4649	0.5658	0.6347	0.3577	0.099

Table 4.6: Summary of forgetting strategy performances on the *b9ae-DA* testing set for a range of w parameters. Strategies are ordered in decreasing order of mean weighted accuracy.

different across base learners although the McNemar tests show that offline and online retraining misclassify different instances. Using Logistic Regression yields a mean A_w gap of 7.7% (mean $A_w = 0.6977$ for offline and mean $A_w = 0.7521$ for online) and the gap is of 12.3% when using a Decision Tree base learner (mean $A_w = 0.6218$ for offline and mean $A_w = 0.6988$ for online). The performance gap between online retraining and forgetting strategies is consistent, it is higher than on previous datasets as anticipated due to the dataset's higher class flip measure of 36.8%. When using Logistic Regression as a base learner, it ranges from 0.7% for DIOC to 7.1% for TC (mean $A_w = 0.7576$ and mean $A_w = 0.8057$ respectively) and when using Decision Trees it ranges from 3.7% for windowed learning to 7.9% for TG (with mean $A_w = 0.7248$ and mean $A_w = 0.7545$ respectively. This is a typical case of a large dataset where forgetting strategies perform better than online retraining due to the presence of class flips throughout the datasets, this is confirmed by McNemar test which finds the difference in performance statistically significant. Finally, we once again note the overall weaker performance when using Decision Trees as a base learner.

c260-DA Results

c260-DA shown in Figures 4.15 and 4.18 with performances summarised in Tables 4.7 and 4.10 for Logistic Regression and Decision Tree base learners respectively, is the largest dataset we consider with 2047 points and spans over 1 year. This again leads to a marked benefit in using

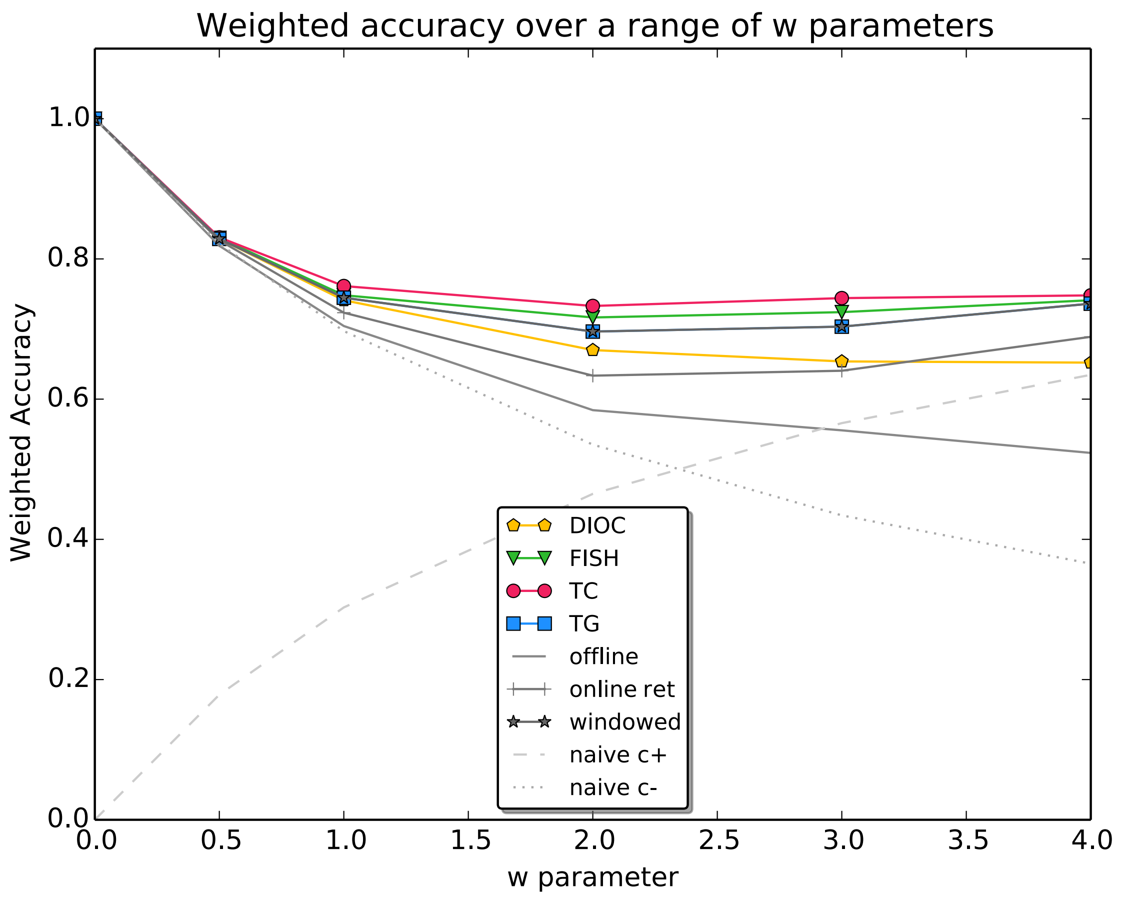


Figure 4.14: Plot of forgetting strategy performances A_w on the b9ae-DA testing set for different values of the user preference parameter $w = \frac{c^+ \text{ weight}}{c^- \text{ weight}}$. c^+ denotes the disruptive call class and c^- the non-disruptive call class. The baseline performances naive c^+ and naive c^- indicate the performance attained when always predicting the corresponding classes. Individual forgetting strategy hyperparameters for each w are reported in Appendix A.3

machine learning across base learners: a 19.7% increase for TC using Logistic Regression and 29.0% for TC using a Decision Tree (mean $A_w = 0.7545$ and mean $A_w = 0.8542$ respectively compared to naive c^- mean $A_w = 0.6621$). c260-DA's concept drift measure is 35.5% with a slight but significant performance gap between online retraining and offline learning as confirmed by McNemar tests. Using Logistic Regression yields a mean A_w gap of 3.5% (mean $A_w = 0.7421$ for offline and mean $A_w = 0.7176$ for online) and the gap is of 1.2% when using a Decision Tree base learner (mean $A_w = 0.7792$ for offline and mean $A_w = 0.7886$ for online). The performance gap between online retraining and forgetting strategies is consistent, and significant according to McNemar tests, with a similar pattern and class flip measure to b9ae-DA.

When using Logistic Regression as a base learner, it ranges from 2.0% for DIOC to 6.8% for TC (mean $A_w = 0.7576$ and mean $A_w = 0.7931$ respectively) and when using Decision Trees it ranges from 1.9% for windowed learning to 8.3% for TG (with mean $A_w = 0.8042$ and mean $A_w = 0.8542$ respectively). It is interesting to note that, opposite to 784c-DA and b9ae-DA, using a Decision trees yields higher performance than using Logistic Regression on this dataset.

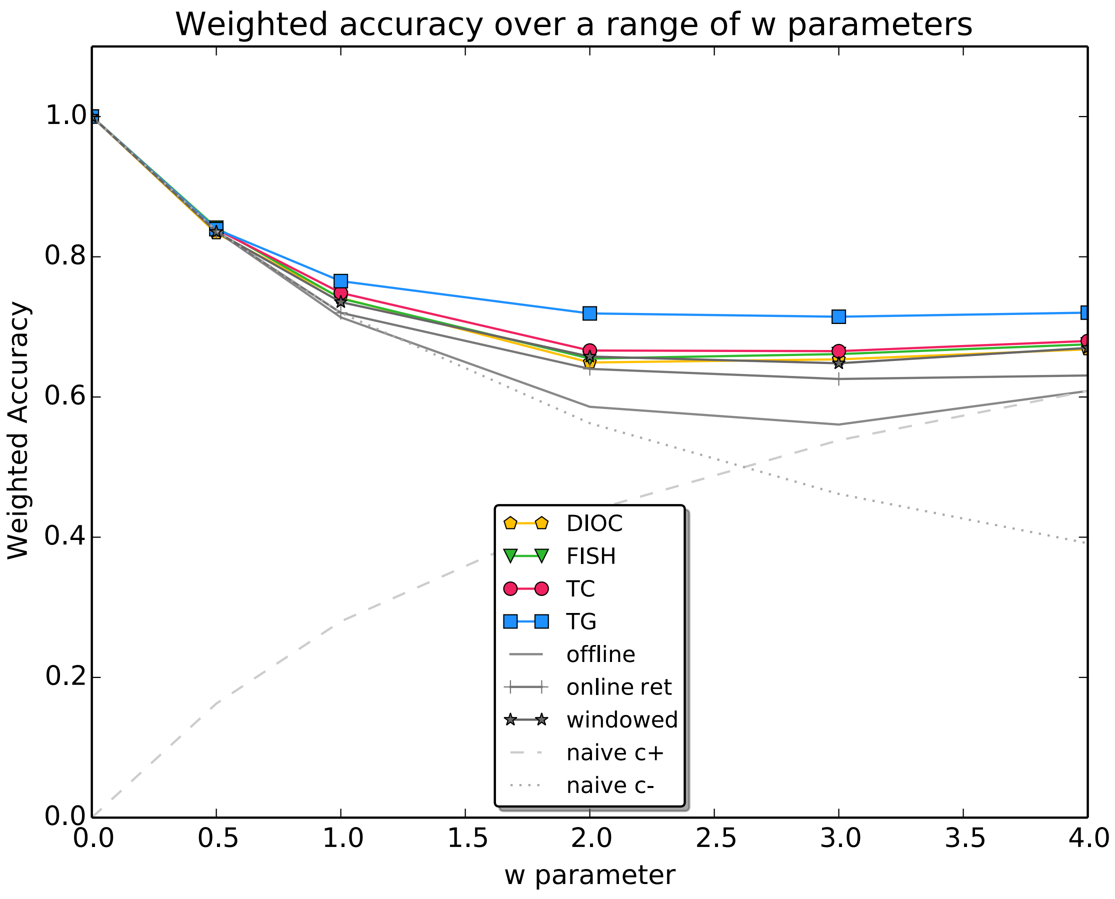


Figure 4.15: Plot of forgetting strategy performances A_w on the c260-DA testing set for different values of the user preference parameter $w = \frac{c^+ \text{ weight}}{c^- \text{ weight}}$. c^+ denotes the disruptive call class and c^- the non-disruptive call class. The baseline performances naive c^+ and naive c^- indicate the performance attained when always predicting the corresponding classes. Individual forgetting strategy hyperparameters for each w are reported in Appendix A.3

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4	mean A_w	SE
TG	1	0.8394	0.7653	0.7192	0.7145	0.7203	0.7931	0.045
TC	1	0.8394	0.7485	0.6663	0.6654	0.6799	0.7665	0.054
FISH	1	0.8415	0.7405	0.6549	0.6612	0.6751	0.7622	0.055
Windowed	1	0.8356	0.7354	0.6577	0.6481	0.6700	0.7578	0.056
DIOC	1	0.8335	0.7413	0.6492	0.6537	0.6680	0.7576	0.056
Online ret.	1	0.8360	0.7201	0.6401	0.6257	0.6307	0.7421	0.061
Offline	1	0.8369	0.7136	0.5860	0.5607	0.6086	0.7176	0.070
Naive c^-	1	0.8373	0.7201	0.5626	0.4617	0.3914	0.6621	0.095
Naive c^+	0	0.1627	0.2799	0.4374	0.5383	0.6086	0.3378	0.095

Table 4.7: Summary of forgetting strategy performances on the *c260-DA* testing set for a range of w parameters. Strategies are ordered in decreasing order of mean weighted accuracy.

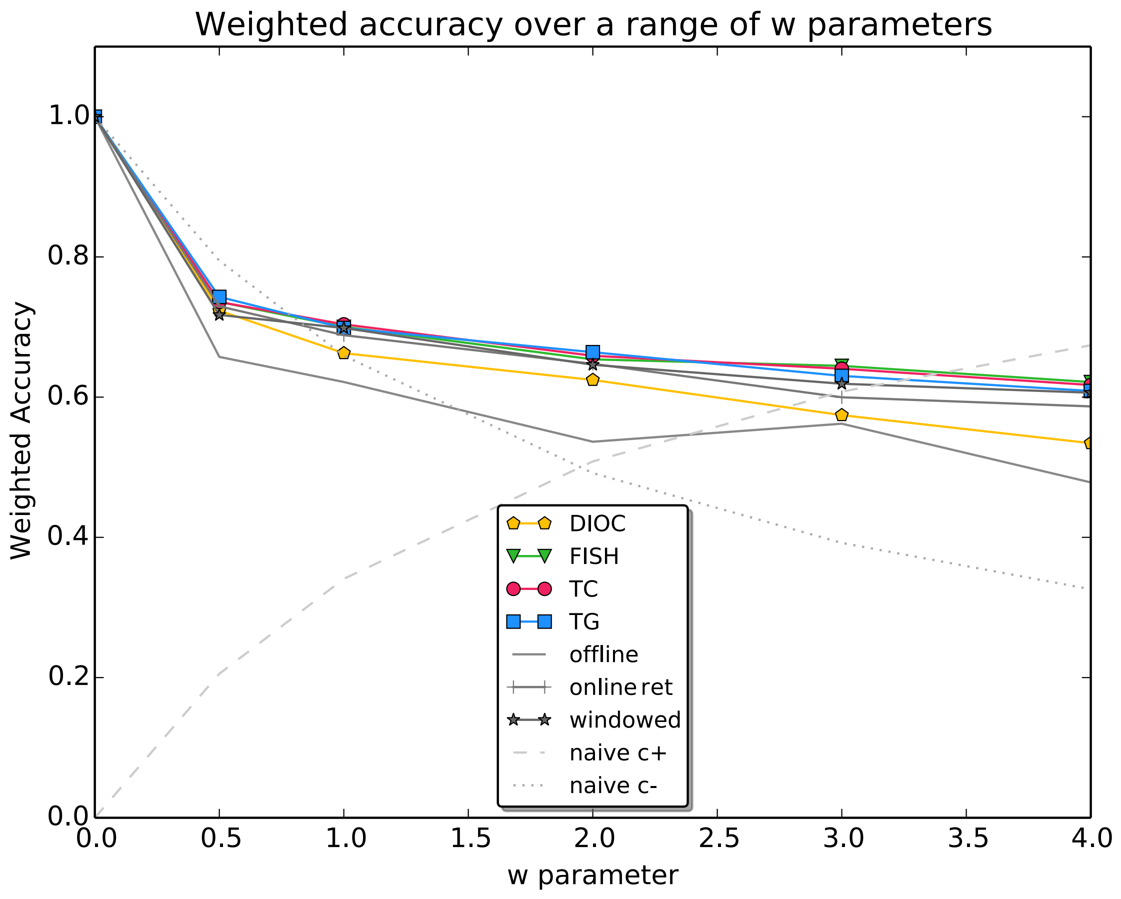


Figure 4.16: Plot of forgetting strategy performances A_w on the 784c-DA testing set using a Decision Tree base learner for different values of the user preference parameter $w = \frac{c^+ \text{ weight}}{c^- \text{ weight}}$. c^+ denotes the disruptive call class and c^- the non-disruptive call class. The baseline performances naive c^+ and naive c^- indicate the performance attained when always predicting the corresponding classes. Individual forgetting strategy hyperparameters for each w are reported in Appendix A.3

4.3.4 Discussion and Limitations

Due to the weak overall performance of learning i.e learners converging to naive strategies on L-RingLearn and F-RingLearn, we do not base our conclusions on those experiments, they were kept as an example of the possible outcome of data collection in the wild practitioners might face. The experiments over J-RingLearn, 784c-DA, b9ae-DA and c260-DA confirm the patterns observed in Chapter 3. The concept drift measure, which was high across the four datasets, ranging from 37.3% to 63.2%, is consistently correlated with adaptive learning achieving higher performance than offline learning. Interestingly, McNemar tests reveal that online retraining

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4	mean A_w	SE
FISH	1	0.7358	0.7003	0.6539	0.6446	0.6215	0.7260	0.057
TC	1	0.7358	0.7038	0.6590	0.6405	0.6173	0.7260	0.057
TG	1	0.7430	0.6986	0.6641	0.6303	0.6088	0.7241	0.058
Windowed	1	0.7173	0.6986	0.6462	0.6191	0.6063	0.7145	0.059
Online ret.	1	0.7296	0.6884	0.6475	0.5998	0.5868	0.7086	0.062
DIOC	1	0.7234	0.6627	0.6245	0.5743	0.5343	0.6865	0.068
Offline	1	0.6574	0.6216	0.5364	0.5621	0.4784	0.6426	0.075
Naive c^-	1	0.7946	0.6592	0.4917	0.3912	0.3260	0.6106	0.105
Naive c^+	0	0.2054	0.3408	0.5083	0.6079	0.6740	0.3894	0.104

Table 4.8: Summary of forgetting strategy performances on the *784c-DA* testing set for a range of w parameters with a Decision Tree base learner. Strategies are ordered in decreasing order of mean weighted accuracy.

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4	mean A_w	SE
TG	1	0.7600	0.7197	0.6791	0.6797	0.6889	0.7545	0.050
TC	1	0.7600	0.7292	0.6855	0.6553	0.6490	0.7465	0.053
DIOC	1	0.7649	0.7162	0.6563	0.6080	0.5905	0.7339	0.061
FISH	1	0.7586	0.7162	0.6673	0.6383	0.6304	0.7351	0.056
Windowed	1	0.7397	0.7209	0.6563	0.6294	0.6030	0.7248	0.059
Online ret.	1	0.7348	0.6960	0.6153	0.5865	0.5607	0.6988	0.066
Naive c^-	1	0.8216	0.6971	0.5351	0.4342	0.3653	0.6422	0.099
Offline	1	0.7110	0.6485	0.5242	0.4430	0.4045	0.6218	0.089
Naive c^+	0	0.1784	0.3029	0.4649	0.5658	0.6347	0.3577	0.099

Table 4.9: Summary of forgetting strategy performances on the *b9ae-DA* testing set for a range of w parameters with a Decision Tree base learner. Strategies are ordered in decreasing order of mean weighted accuracy.

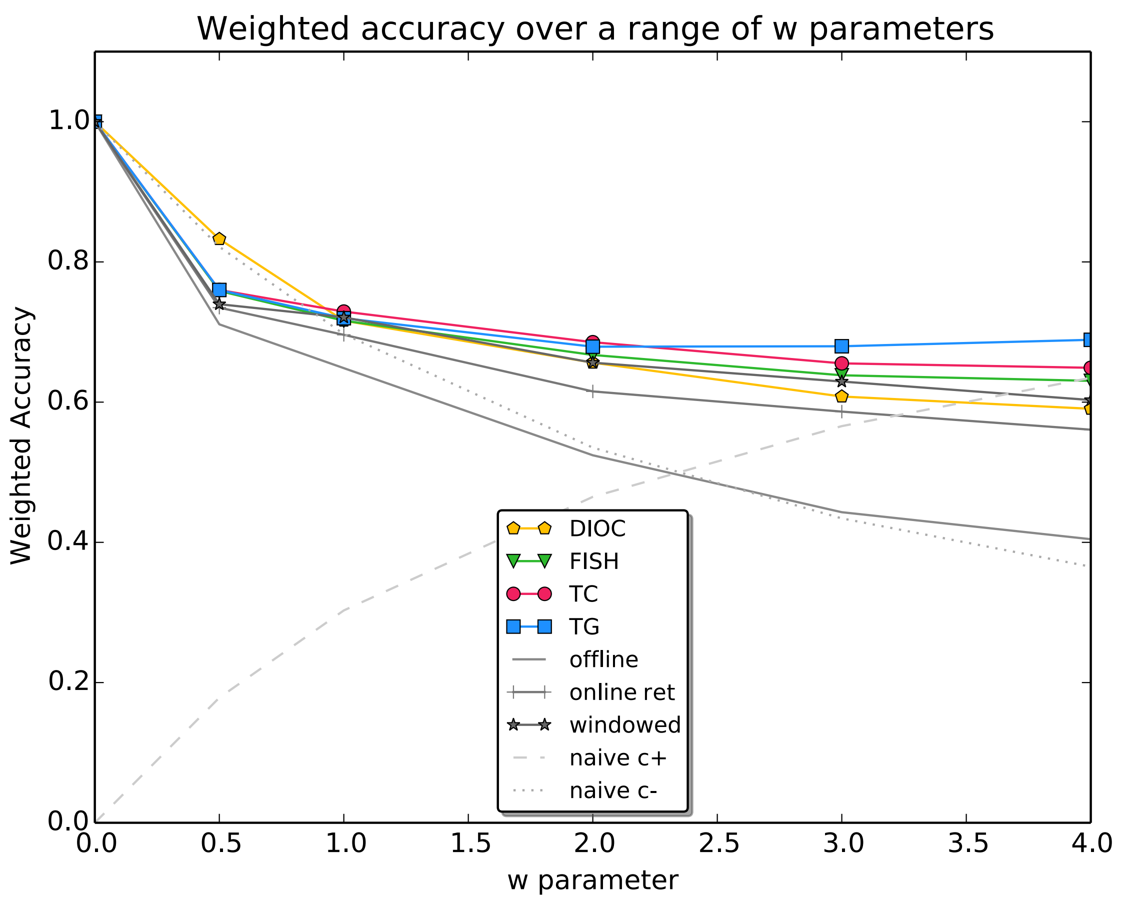


Figure 4.17: Plot of forgetting strategy performances A_w on the b9ae-DA testing set using a Decision Tree base learner for different values of the user preference parameter $w = \frac{c^+ \text{ weight}}{c^- \text{ weight}}$. c^+ denotes the disruptive call class and c^- the non-disruptive call class. The baseline performances naive c^+ and naive c^- indicate the performance attained when always predicting the corresponding classes. Individual forgetting strategy hyperparameters for each w are reported in Appendix A.3

makes predictions which are significantly different from offline learning, the mean A_w performance gap between the two was often small and inconsistent across base learners similar to 784c-DA where it is 3.9% using Logistic Regression and 10.2% for Decision Trees, on b9ae-DA these values where 7.7% and 12.3%. The McNemar tests show that although offline and online retraining can be close in performance, the actual instances they misclassify are different. This means that integrating new knowledge is beneficial as online retraining will correctly classify some test instances offline learning won't, but that integrating new knowledge *indiscriminately* yields misclassifications that would not have happened if it was ignored (offline learning) — confirming the intuitive benefit of forgetting strategies.

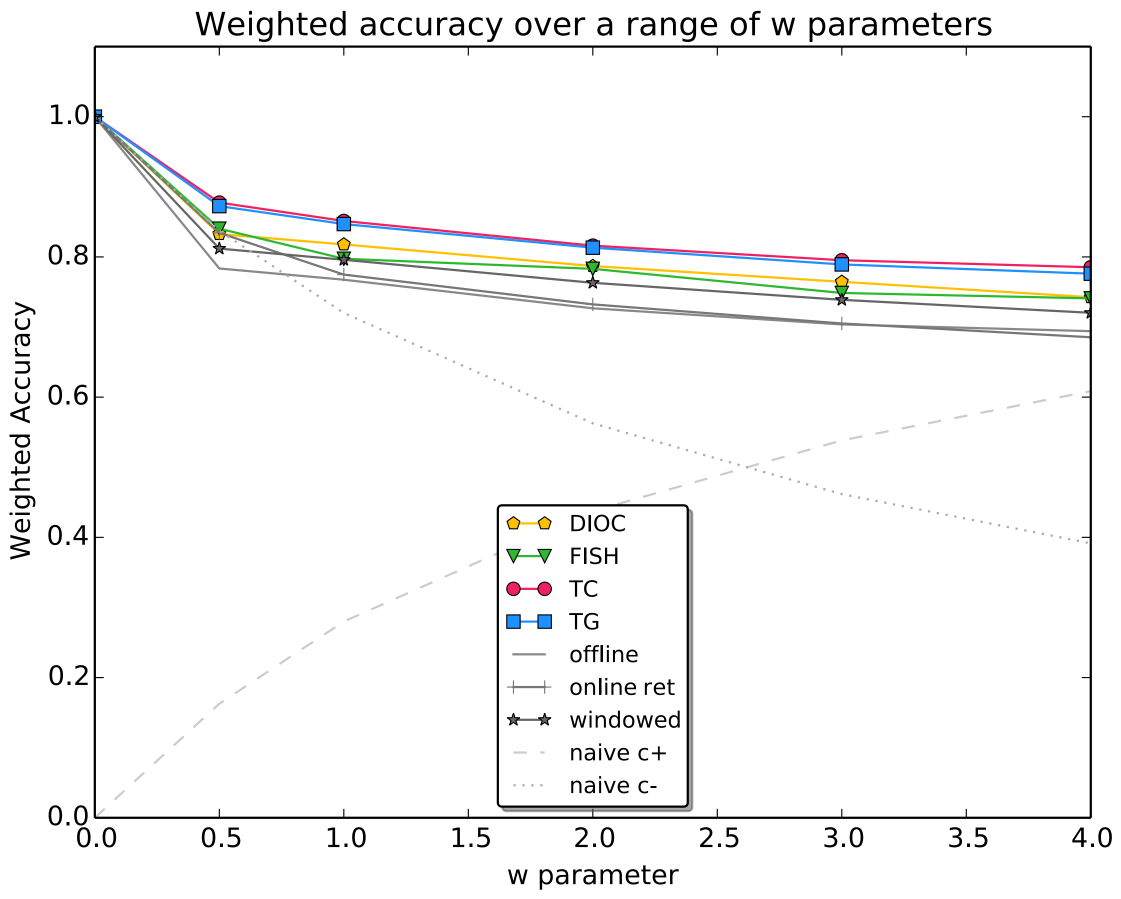


Figure 4.18: Plot of forgetting strategy performances A_w on the c260-DA testing set using a Decision Tree base learner for different values of the user preference parameter $w = \frac{c^+ \text{ weight}}{c^- \text{ weight}}$. c^+ denotes the disruptive call class and c^- the non-disruptive call class. The baseline performances naive c^+ and naive c^- indicate the performance attained when always predicting the corresponding classes. Individual forgetting strategy hyperparameters for each w are reported in Appendix A.3

Larger class flip measures were consistently linked across base learners with larger gap in performance between online retraining and one of the four benchmarked forgetting strategies. The maximal mean A_w performance improvement on J-RingLearn is 6.7% (Logistic Regression based FISH) for a 18.3% class flip measure, 2.0% (Logistic Regression based TG) and 2.6% (Decision Tree based FISH) on 783c-DA for a 3.8% class flip measure, 7.1% (Logistic Regression based TC) and 7.9% (Decision Tree based TG) on b9ae-DA for a 36.8% class flip measure, and 6.8% (Logistic Regression based TC) and 8.3% (Decision Tree based TC) on c260-DA. We note that the actual class flip measure value cannot be used to predict the performance gain range, it does not give bounds on performance increase, rather it is a indicator that a dataset is likely to

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4	mean A_w	SE
TC	1	0.8775	0.8513	0.8161	0.7953	0.7853	0.8542	0.032
TG	1	0.8725	0.8469	0.8132	0.7893	0.7761	0.8496	0.033
DIOC	1	0.8326	0.8326	0.7870	0.7645	0.7425	0.8240	0.037
FISH	1	0.8403	0.7974	0.7830	0.7486	0.7409	0.8183	0.039
Windowed	1	0.8119	0.8309	0.7893	0.7388	0.7203	0.8050	0.040
Online ret.	1	0.8343	0.7748	0.7323	0.7051	0.6854	0.7886	0.047
Offline	1	0.7835	0.7675	0.7267	0.7037	0.7037	0.7792	0.045
Naive c^-	1	0.8373	0.7201	0.5626	0.4617	0.3914	0.6621	0.095
Naive c^+	0	0.1627	0.2799	0.4374	0.5383	0.6086	0.3378	0.095

Table 4.10: Summary of forgetting strategy performances on the *c260-DA* testing set for a range of w parameters with a Decision Tree base learner. Strategies are ordered in decreasing order of mean weighted accuracy.

have contradictory class boundaries across epochs which in turn make the use of a forgetting strategy beneficial.

Regarding which strategy to choose, the experiments shows that DIOC performs the worse, its mean A_w being usually very close to that of windowed learning, online retraining or offline learning and predictions not significantly different from these according to McNemar tests. FISH and TC are usually the best learners with similar performances but significantly different predictions patterns according to McNemar tests (except on 784c-DA where the null hypothesis cannot be rejected regardless of the base learner) giving only a slight advantage to TC: $A_w = 0.7674$ versus $A_w = 0.7665$ On J-RingLearn, $A_w = 0.7933$ versus $A_w = 0.8057$ on b9ae-DA using Logistic Regression, $A_w = 0.7351$ versus $A_w = 0.7465$ on b9ae-DA using Decision Trees, $A_w = 0.7622$ versus $A_w = 0.7665$ on c260-DA using Logistic Regression and $A_w = 0.8183$ versus $A_w = 0.8542$ on c260-DA using Decision Trees. Finally, interestingly, we note that cost sensitivity for low values of w when using a Decision Tree base learner is less well achieved than when using Logistic Regression as can be seen on Figures 4.16, 4.17 and 4.18, although we did not find any related conclusions in the literature this would be an interesting point to investigate in future work.

4.4 Summary

This chapter started by motivating our choice of using smartphone notifications as a case study and justifies our decision to focus on disruptive incoming calls.

It covered background literature in interruptions, targeting disruptive smartphone notifications specifically. In commercial devices, disruptive smartphone notification handling is done by manually setting notification signalling rules for applications and enabling or disabling the enforcement of these rules by choosing an overall smartphone signalling setting such as priority only or 'Do Not Disturb'. In research, solutions aim to leverage user context and machine learning to avoid manually setting rules for applications and the overall phone signalling setting. The methods can be categorised into scheduling notifications to predicted opportune moments and context-dependent mitigation of notification signalling. We then argued that scheduling is not straightforwardly applicable to incoming calls while most of the work in notification mitigation employs explicit experience sampling which is not fit for long term deployment — an essential property for mobile context-aware applications to be useful in practice. Another interesting point that came from a smartphone usage survey conducted on 65 participants showing that mobile communication is increasingly happening through asynchronous communications messaging application such as WhatsApp and Facebook messenger. A comprehensive approach must thus be used to manage disruptive notification again favouring mitigation over scheduling. Different types of notifications thus might have to be mitigated in different ways which is an interesting future work topic.

The chapter proposes a new way of mitigating disruptive notifications using implicit experience sampling. Implicit experience sampling silently infers data labels from users' actions when receiving a notification rather than explicitly requesting labels as in the case of explicit experience sampling. A solution prototype was implemented as an Android application called RingLearn which labels incoming calls as disruptive or non-disruptive based on the button a user chooses to either accept or reject a call on a custom incoming call screen. Data was gathered over 16 weeks on 11 users and a post-usage survey revealed that 7 out of 11 participants would use RingLearn if it went beyond the prototype stage (RingLearn was only collecting data for the duration of the experiment). The remaining 4 participants requested a mechanism to be able to change inferred call labels as it is not always possible to predict the disruptiveness of an incoming call before answering it.

The resulting data collection and labelling mechanism which addresses the corresponding core challenge of intelligent mobile context-aware applications, was obtained by refining our initial implicit experience sampling strategy by combining it with a confirmation step. The resulting infer-and-confirm strategy labels data using implicit experience sampling and then temporarily informs the user each time a new label is inferred, leaving the user a small time window to amend it if necessary. If the user does not amend the label within the time frame, it is considered valid — the optimal outcome, not requiring any extra effort on the part of the user compared to normal device use. This strategy can be used for arbitrary smartphone notifications and beyond, for instance, we retrospectively found it used in a commercial Thermostat auto-scheduling product as mentioned in Section 1.1. An illustration of how our approach can be applied in practice in an arbitrary mobile context-aware application is shown in Figure 4.19.

In Section 4.3 we used the 3 largest datasets collected using RingLearn and 3 datasets from another incoming call dataset, Device Analyzer, to test the chapter's fundamental hypotheses: can we predict disruptive versus non-disruptive incoming calls on data gathered through implicit experience sampling? And, is the use of a forgetting strategy beneficial on the call data i.e. do users change what they consider to be a disruptive call as time goes by?

We constructed 9 experiments over the 6 datasets testing learners over a range of A_w parameters to answer these hypotheses. The results showed that on the RingLearn datasets with 193, 236 and 294 points each, an 8.3%, 1.2% and 12.6% maximal average performance increase was observed over the naive strategies. The implicit experience sampling strategy used over 16 weeks was thus not very effective, especial in the case of the second dataset, in collecting enough data to predict disruptive incoming calls. The three Device Analyzer datasets, chosen for their larger size and lengthier collection time spans, 871 points spanning 19 weeks and 4 days, 1256 points spanning 1 year 7 months and 9 days, and 2047 points spanning 1 year and 5 days, showed maximal average performance increases of 21.7%, 25.5% and 29.0%. *These results* *indicate that human behaviour sensed by simple smartphone sensors is consistent enough so that given sufficient passive observations of incoming call events one is able to predict disruptive incoming calls significantly better than making naive guesses.* That being said, the performance benefits observed over naive predictions are probably not worth the amount of time required to gather the amount of training data needed to achieve them — at least 4 weeks for even the smallest Device-Analyzer dataset. A solution to this is to combine our proposed infer-and-confirm strategy with either an initial period of explicit experience sampling or bootstrapping data with a questionnaire. The way in which to combine data collection techniques, how long do we collect initial data for and how this affects user interaction with the application for instance, is an important future work direction.

The most important finding of the chapter concerns the second hypothesis. First, we have experimentally confirmed that the concept drift measure was consistently correlated with adaptive learning performing significantly better than offline learning (although the performance gaps between offline and online retraining were sometimes small) and that the class flip measure is correlated with forgetting strategies performing significantly better than online and offline learning in line with our findings of Chapter 3. Specifically, 4 cases arise:

- A high concept drift measure is observed with a *low* class flip measure which leads to offline learning's mean A_w performance being *close to* the online retraining and forgetting strategies' mean A_w as in the case of 784c-D.A using Logistic Regression
- A high concept drift measure is observed with a *low* class flip measure which leads to offline learning performing markedly *worse than* online retraining and forgetting strategies as in the case of 784c-D.A using Decision Trees
- A high concept drift measure is observed with a *high* class flip measure which leads to offline learning's mean A_w performance being *close to* the online retraining mean A_w but forgetting strategies performing *better than* online retraining as in the case of c260-DA using Decision Trees
- A high concept drift measure is observed with a *high* class flip measure which leads to

offline learning performing *worse than* online retraining and forgetting strategies performing *better than* online retraining as in the case of J-RingLearn, b9ae-D.A and c260-DA using Logistic Regression.

This once again supports our claim that concept drift is not always informative of any performance patterns other than that some form of adaptive learning will increase performance compared to offline learning and highlights the advantage of thinking in terms of 'necessity to forget' which reliably predicts that forgetting strategies will be beneficial over online retraining *and* offline learning.

In these sets of experiments, we found the best forgetting strategies were TC and FISH with DIOC lagging behind. Interestingly, on 784c-DA using Logistic Regression, c260-DA using Logistic Regression and b9ae-DA using Decision Trees TG performed better than TC, once again revealing the same weakness of TC as on the Artificial6 dataset of Section 3.4. Because clusters are only computed using the training set, if a new cluster appears during testing it will not be covered by existing clusters properly, leading to misclassifications. An immediate piece of future work would thus be to refine the TC algorithm to periodically recompute its density-based clustering as in principle its performance should be at least the same as TG — as a TG segmentation of the data space can be approximated using TC but not vice versa.

Finally, we observed that Decision Trees were worse than Logistic Regression on cost sensitive learning tasks, specifically for $w = 0.5$, where learners using a Decision Tree base classifier often performed worse than naive predictions. This indicates that some base learners might be more suitable or less suitable for cost sensitive learning depending on how they achieve cost-sensitivy. Because cost-sensitivity is so important in mobile context-aware applications it would also be interesting to investigate this further in future work.

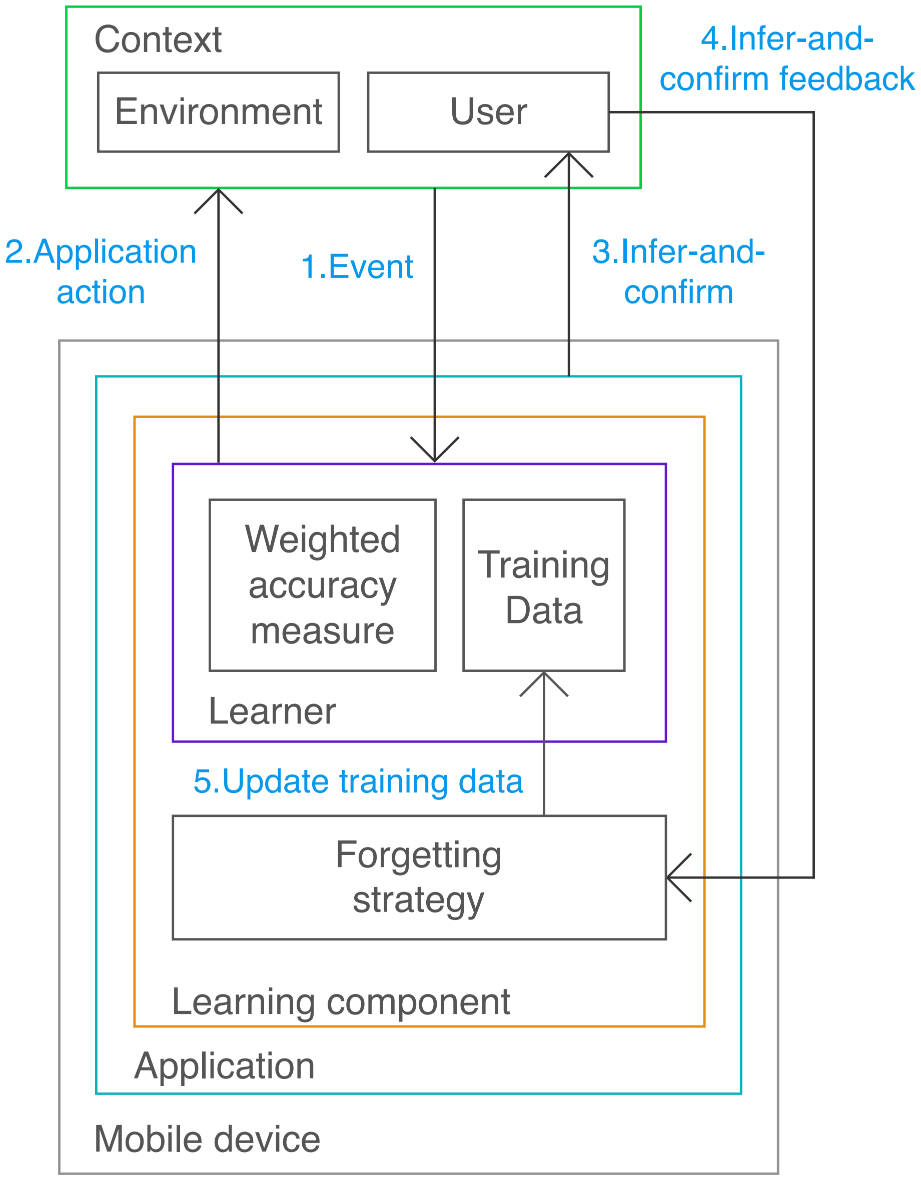


Figure 4.19: An illustration of how forgetting strategies and our weighted accuracy measure from Chapter 3 and the infer-and-confirm strategy proposed in this chapter are meant to be combined in a deployable intelligent mobile context-aware application

Chapter 5

Conclusion

The goal of this thesis is to address the challenges in making mobile context-aware applications deployable over indefinite periods of time. It focuses on *intelligent* mobile context-aware application which use machine learning to infer the correct behaviour to adopt based on sensed context. This type of application presents an advantage over *static* applications which require the user or an expert to entirely pre-program the application's behaviour before deployment and allows for adaptation to new contexts. This subsumes three main challenges. First, finding a way of collecting labelled data to train machine learners that is practical for real world deployment of the application. Second, defining a objective function that matches user preferences for the application's behaviour. Third, devising a mechanism to adapt to changes in user behaviour which might occur during deployment.

This thesis thus differentiates itself from much of the existing literature by proposing a comprehensive approach to intelligent mobile context-aware systems. It proposes an infer-and-confirm user interaction paradigm to collect labelled data while minimising the cognitive load of doing so which was elaborated based on the deployment of a disruptive incoming call manager prototype called RingLearn on the Android platform. It proposes to use a weighted accuracy measure as an objective function to train learners in a way which takes into account user preference and offsets class imbalance. Last but not least, it offers a new way of approaching the problem of concept drift which is the effect of changes in user preferences or behaviour during an application's deployment which in turn might change the expected behaviour of the application. In doing so, the thesis decouples the notion of concept drift and the 'need to forget'. It proposes a new way to quantify both, and introduces the notion of a forgetting strategy. Two new forgetting strategies, Training Grid (TG) and Training Clusters (TC) are developed and compared to both baseline forgetting strategies and state-of-the-art methods using two data corpuses, the data collected via our RingLearn Android application and Device Analyzer, the largest smartphone usage dataset available to date.

5.1 Summary of Thesis Achievements

The main original contributions of this thesis are the following:

A concept drift analysis algorithm which decouples concept drift and the need to forget. The algorithm (Algorithm 1) takes as input a number of epochs to carry out the analysis over and a feature splitting parameter. It then proceeds to segment the dataspace accordingly and analyses local changes in each area of the data space across epochs. The algorithm returns the percentage of points which are new between epochs i.e. not present in anterior epochs, and the number of points affected by what we refer to as class flips. Class flips occur when an area in the dataspace changes its dominant class in between epochs, it is considered a form of concept drift because it indicates that classification boundaries from previous epochs might contradict the boundary of the epoch currently being analysed, the overall percentage of points affected by these two events are said to be affected by concept drift. The number of points only affected by class flips indicate a potential need to forget some data in the corresponding area.

Two learner-independent dynamic training set formation strategies which we refer to as forgetting strategies (Algorithms 14 and 18). These are our proposed solution to learning under concept drift in mobile context-aware applications designed to handle any type of local or total concept drift and perform at least as well, and many times better, than current solutions. The Training Grid and Training Clusters algorithm share the same core idea, segment the input data space and assign a local training window to each area. Data is thus only discarded if it can be replaced by newer instances in the same area building upon the popular concept of windowed learning. The two algorithms differentiate themselves in the way they define areas, Training Grid segments the input space into regular equal n-dimensional volume areas based on a feature splitting parameter, while Training Clusters uses density-based clustering to segment the dataspace.

A use case study of disruptive smartphone notifications, specifically aimed toward disruptive incoming calls, which led to the development of an Android application called RingLearn presented in Section 4.2 to understand user behaviour towards disruptive incoming calls and the gathering of 16 weeks of incoming call data over 11 participants. The conclusion of the study led to the elaboration of a generic infer-and-confirm strategy detailed in Section 4.2.2 to gather labelled data in a user friendly way over indefinite periods of time that can be used in mobile context-aware applications.

Two separate evaluations comparing, naive predictions, offline learning, online retraining, windowed learning, two state-of-the-art forgetting strategies and our proposed forgetting strategies using a weighted accuracy performance measure A_w (presented in Section 3.3). The weighted accuracy performance measure takes into account user preferences as to which class is more important to predict correctly and can be used to offset class imbalance using a single weight parameter w, in the case of disruptive call management for instance a user might consider the correct prediction of disruptive calls to be more important than non-disruptive calls although they might occur less often which corresponds to a large w for the disruptive call class. The set of experiments in Chapter 3 was carried out over 8 datasets coming from 6 different corpuses with a fixed user preference parameter w analysing the performance of learners as time passes, while the 9 experiments in Chapter 4 specifically targeted disruptive smartphone notifications and were carried out over data collected using our RingLearn application and the Device Analyzer application. In Chapter 4, we also compared forgetting strategies using two different base learners, Logistic Regression and Decision Trees to experimentally show that our proposed learning strategies are indeed learner independent. Over the course of the experiments we have confirmed the 4 patterns we had hypothesised. First, that the presence of concept drift does not always imply that a forgetting strategy will be beneficial — it could be the case that online retraining suffices to achieve maximal performance. Second, that the presence of concept drift as computed by our concept drift analysis algorithm implies that online retraining will be significantly better than offline learning which we confirmed by looking at performance gaps between the two across experiments and applying McNemar tests on prediction contingency tables. Third, that the class flip measure indicates the need to forget which implies that forgetting strategies will perform significantly better than online retraining on a dataset, again confirmed by looking at performance gaps and applying McNemar tests on prediction contingency tables. Finally, that our proposed TG and TC algorithms perform at least as well as existing forgetting strategies when these are optimal over a dataset, and many times better. TC usually outperforms TG (10 out of 15 times) and performance delta between the best of TC and TG over existing solutions ranged from -0.2% to 11.3% across the 15 experiments.

5.2 Critical Review and Future Work

This thesis touches upon a variety of topics including machine learning, context-aware systems, HCI and smartphones notifications. As such, some topics were investigated more in depth than others with room for refinement and future work in these.

5.2.1 Machine Learning

The proposed CDA algorithm quantifies and decouples concept drift and the need to forget using the concept drift and class flip measure which seems to be a crucial element in better understanding dataset properties and appropriate learning strategies. That being said, because our proposed method considers chunks of data (the data in a grid element partitioning the dataspace) and does not require equal volume grid elements to compute measures, it can lead to the class flip measure being overestimated which in turn makes it difficult to compare class flip measures across datasets with different data density and feature ranges. An interesting area of future work is thus to find alternative ways to computing the class flip measure, for instance using finer grained grids in areas where class flips are thought to happen and taking steps to normalise datasets before applying CDA.

The proposed Training Grid and Training Clusters algorithms fulfil their intended purpose, to converge to simple forgetting strategies (online retraining and windowed learning) in simple cases and significantly outperform existing state-of-the-art approaches (DIOC and FISH) in most complex scenarios. That being said, further improvements could be explored to potentially increase this performance gap. One of the spotted weaknesses of TC in our experiments is that the dataspace partition it creates during the training phase might not fit new data instances acquired during the testing phase after some time. This could be solved by periodically recomputing dataset partitions. Similarly, both methods could potentially have different sized windows attached to different areas of the dataspace, as a function of their size for instance.

We note that although all our experiments assume binary class problems, our work is directly applicable to multiple class problems. Specifically, the definitions of new data and class flip events of the CDA algorithm from Section 3.1.1 have no restrictions on the number of classes they consider. The weighted accuracy measure we propose in Section 3.3.1 can straightforwardly be extended to multiple class problems by assigning a weight w_i to each class such that $\Sigma_i w_i = 1$ and computing $A_{w_i} = \frac{\Sigma_i w_i * recall_i * |c^i|}{\Sigma_i w_i |c^i|}$ where $recall_i$ is the recall for class c^i and $|c^i|$ is the number of instance in class c^i. Finally the TG and TC forgetting strategies are meta-learners which do not make use of datapoint classes in their adaptive training set formation mechanism so that their support for multiple class learning problems only depends on the base learner chosen by a practitioner.

5.2.2 Mobile Context-aware Systems

In Chapter 4, we presented the RingLearn disruptive incoming call management prototype which implicitly (passively) collected incoming call data during 16 weeks. In the associated experiments, it became clear that 2 out of the 3 collected datasets were too small for machine learning to offer a marked advantage over naive predictions while data collected over longer periods of time did not present this pattern. In practice, 16 weeks seems to be a very long time for a user to wait until a mobile application is able to make predictions that beat naive predictions. A solution to this would be to use an initial data bootstrapping phase using explicit (active) data collection strategies such as explicit experience sampling or a questionnaire and once enough data is collected only rely on passive data collection mechanisms to keep the application updated. The way in which to combine implicit and explicit data collection methods is a worthwhile area of future work.

Our use case focused on disruptive incoming calls as a survey revealed these are potentially the most disruptive. The same survey also revealed that asynchronous messaging applications were the most commonly used communication method, an interesting area of future work is to extend our proposed methodology to cover all types of notifications in a unified way which implies to devise a way to passively recognise whether incoming asynchronous messages are disruptive or not.

5.2.3 Human Computer Interaction

The infer-and-confirm method seems like a natural way to passively collect labelled data from device use, in fact, we found it to be very similar to the data acquisition paradigm used in the Nest thermostat auto-scheduler system retrospectively. The confirm step is a user facing component in which a temporary dialog is shown to the user informing them of the system's predicted label given the current context letting the user correct the label if need be. The confirm step thus has various interface and interaction design questions attached to it such as: what does the dialog look like, where should it be placed to minimise disturbance, how long should it be shown and how does the user interact with it which is a crucial piece of future work to make mobile context-aware systems that can be deployed over indefinite periods of time.

This thesis also proposes a weighted accuracy measure A_w to take into account user preferences and class imbalance when measuring a system's performance. The weighted accuracy measure is parametrised by a class weight parameter w in the case of binary class prediction problems which can be extended to multiple classes by assigning a weight to each class. This implies a fundamental HCI question: how can we model user preferences using numbers, in the specific case of weighted accuracy, how do we find a given user's w? Mending the discrepancy between computed performance and user perceived performance is a cornerstone for learning context-aware applications to become mainstream so that studying how humans perceive a learning (mobile context aware) system's performance is also a critical future work to pursue.

5.3 Final Comments

Mobile context-aware systems are a very important topic as they carry the potential to impact many people's daily lives. Applications for this work are two fold. The first area is pure machine learning, where our proposed concept drift analysis algorithm and forgetting strategies can be used beyond context-aware systems in problems where concept drift is suspected. Second, the overall findings of the thesis in terms of which type of learning to use, which objective function to optimise and how to collect data, can be applied in mobile applications beyond disruptive incoming calls such as smart-home devices, daily activity recognition or recommender systems.

This thesis became what I have had wanted to read when I first started doing research in intelligent context-aware systems: an in depth and unified exploration of the different theoretical and practical challenges in deploying intelligent mobile context-aware systems over indefinite periods of time, allowing me to address the further theoretical and practical challenges of mobile context-aware systems specific to healthcare settings. As I could not find this in existing research, I did my best to carry out the work myself and present it in a format that allows other researchers to build upon it and explore the further theoretical and practical challenges attached to a specialised applications directly.

Bibliography

[AB04] Piotr D. Adamczyk and Brian P. Bailey. If not now, when?: The effects of interruption at different moments within task execution. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 271–278. ACM Press, 2004.

[AB13] Iris Ad and MichaelR. Berthold. Eve: a framework for event detection. *Evolving Systems*, 4(1):61–70, 2013.

[ADB+99] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, HUC '99, pages 304–307, London, UK, UK, 1999. Springer-Verlag.

[AGHK07] D. Avrahami, D. Gergle, S. E. Hudson, and S. Kiesler. Improving the match between callers and receivers: A study on the effect of contextual information on cell phone interruptions. *Behaviour and Information Technology*, 26(3):247–259, May 2007.

[AH06] Daniel Avrahami and Scott E. Hudson. Responsiveness in instant messaging: Predictive models supporting inter-personal communication. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 731–740, New York, NY, USA, 2006. ACM.

[Alp10] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010.

[Amf15] Oliver Amft, editor. *Context-Aware Systems: Methods and Applications*. Lecture Notes in Computer Science. Springer, 2015.

[app15] AWW apps, 2015. Available at `http://launchhere.awwapps.com/`.

[Bar04] Jakob E. Bardram. Applications of context-aware computing in hospital work: Examples and design principles. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, SAC '04, pages 1574–1579, New York, NY, USA, 2004. ACM.

[BBS13] Andreas Bulling, Ulf Blanke, and Bernt Schiele. A tutorial on human activity recognition using body-worn inertial sensors. *ACM Computing Surveys*, 46(3):33:1–33:33, 2013.

[BCFF12] Paolo Bellavista, Antonio Corradi, Mario Fanelli, and Luca Foschini. A survey of context data distribution for mobile ubiquitous systems. *ACM Computing Surveys*, 44(4):24:1–24:45, September 2012.

[BG07] Albert Bifet and Ricard Gavald. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, 2007.

[BH10] Jakob E. Bardram and Thomas Riisgaard Hansen. Context-based workplace awareness - concepts and technologies for supporting distributed awareness in a hospital environment. *Computer Supported Cooperative Work*, 19(2):105–138, 2010.

[BHMS06] Jakob E. Bardram, Thomas R. Hansen, Martin Mogensen, and Mads Soegaard. Experiences from real-world deployment of context-aware technologies in a hospital environment. In *Proceedings of the 8th International Conference on Ubiquitous Computing*, UbiComp'06, pages 369–386, Berlin, Heidelberg, 2006. Springer-Verlag.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[BKL$^+$11] Linas Baltrunas, Marius Kaminskas, Bernd Ludwig, Omar Moling, Francesco Ricci, Aykan Aydin, Karl-Heinz Lke, and Roland Schwaiger. Incarmusic: Context-aware music recommendations in a car. In *E-Commerce and Web Technologies*, volume 85 of *Lecture Notes in Business Information Processing*, pages 89–100. Springer Berlin Heidelberg, 2011.

[BS14] Dariusz Brzezinski and Jerzy Stefanowski. Combining block-based and online methods in learning ensembles from concept drifting data streams. *Information Sciences*, 265:50–67, May 2014.

[BvdAvP11] R. P. Jagadeesh Chandra Bose, Wil M. P. van der Aalst, Indre Žliobaite, and Mykola Pechenizkiy. Handling concept drift in process mining. In *Proceedings of the 23rd international conference on Advanced information systems engineering*, CAiSE'11, pages 391–405, Berlin, Heidelberg, 2011. Springer-Verlag.

[BVL12] Marko Borazio and Kristof Van Laerhoven. Combining wearable and environmental sensing into an unobtrusive tool for long-term sleep studies. In *2nd ACM SIGHIT International Health Informatics Symposium*, Miami, Florida, USA, Jan 2012. ACM Press.

[BVP$^+$14] Lars Büthe, Christian Vogt, Luisa Petti, Niko Münzenrieder, Christoph Zysset, Giovanni Antonio Salvatore, and Gerhard Tröster. A mechanically flexible tilt switch on kapton foil with microspheres as a pendulum. In *Proceedings of Sensors and Measuring Systems*, pages 1–4, June 2014.

[BWPL14] Ayne. Beyene, Tewelle Welemariam, Marie Persson, and Niklas Lavesson. Improved concept drift handling in surgery prediction and other applications. *Knowledge and Information Systems*, pages 1–20, 2014.

[CCH00] Edward B. Cutrell, Mary Czerwinski, and Eric Horvitz. Effects of instant messaging interruptions on computing tasks. In *Proceedings of Extended Abstracts on Human Factors in Computing Systems*, CHI EA '00, pages 99–100, New York, NY, USA, 2000.

[CCH+08] T. Choudhury, S. Consolvo, B. Harrison, J. Hightower, A. Lamarca, L. Legrand, A. Rahimi, A. Rea, G. Bordello, B. Hemingway, P. Klasnja, K. Koscher, J.A. Landay, J. Lester, D. Wyatt, and D. Haehnel. The mobile sensing platform: An embedded activity recognition system. *Pervasive Computing, IEEE*, 7(2):32–41, April 2008.

[CFJ03] Harry Chen, Tim Finin, and Anupam Joshi. An ontology for context-aware pervasive computing environments. *The Knowledge Engineering Review*, 18:197–207, 9 2003.

[CK00] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report, Dartmouth College, Hanover, NH, USA, 2000.

[CoB15] Dartmouth College and University of Bologna, 2015. Available at `https://play.google.com/store/apps/details?id=edu.dartmouth.cs.walksafe&hl=en_GB`.

[Con12] Piero Conca. *An adaptive framework for classification of concept drift with limited supervision.* PhD thesis, University of York, 2012.

[DCTC05] Sarah Jane Delany, Pádraig Cunningham, Alexey Tsymbal, and Lorcan Coyle. A case-based technique for tracking concept drift in spam filtering. *Knowledge-Based Systems*, 18(4-5):187–195, August 2005.

[DDL15] Mitra Dirin, Amir Dirin, and Teemu H. Laine. User-centered design of a context-aware nurse assistant (cana) at finnish elderly houses. In *Proceedings of the 9th International Conference on Ubiquitous Information Management and Communication*, IMCOM '15, pages 39:1–39:8, New York, NY, USA, 2015. ACM.

[Dey01] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1):4–7, January 2001.

[DGSB07] Edward S. De Guzman, Moushumi Sharmin, and Brian P. Bailey. Should i call now? understanding what context is considered when deciding whether to initiate remote communication via mobile devices. In *Proceedings of Graphics Interface*, GI '07, pages 143–150, New York, NY, USA, 2007. ACM.

[DM14] P.B. Dongre and L.G. Malik. A review on real time data stream classification and adapting to various concept drift scenarios. In *Proceedings of Advance Computing Conference*, pages 533–537, Feb 2014.

[ds15] Sleep data set, 2015. Available at `http://www.ess.tu-darmstadt.de/hhg_logs/0089`.

[DSJ14] Lei Du, Qinbao Song, and Xiaolin Jia. Detecting concept drift: An information entropy based method using an adaptive sliding window. *Intelligent Data Analysis*, 18(3):337–364, May 2014.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jrg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, 1996.

[eMa15] eMarketer, 2015. Available at `http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536`.

[EP11] R. Elwell and R. Polikar. Incremental learning of concept drift in nonstationary environments. *IEEE Transactions on Neural Networks*, 22(10):1517–1531, Oct 2011.

[EPC08] M. Ermes, J. Parkka, and L. Cluitmans. Advancing from offline to online activity recognition with wearable sensors. In *Engineering in Medicine and Biology Society, 30th Annual International Conference of the IEEE*, pages 4451–4454, Aug 2008.

[fA15] Device Analyzer for Android, 2015. Available at `https://play.google.com/store/apps/details?id=uk.ac.cam.deviceanalyzer&hl=en_GB`.

[FDN$^+$14] M. Ferroni, A. Damiani, A.A. Nacci, D. Sciuto, and M.D. Santambrogio. coda: An open-source framework to easily design context-aware android apps. In *12th IEEE International Conference on Embedded and Ubiquitous Computing*, pages 33–38, Aug 2014.

[FG00] Alan Fern and Robert Givan. Online ensemble learning: An empirical study. In *In Proceedings of the Seventeenth International Conference on Machine Learning*, pages 279–286. Morgan Kaufmann, 2000.

[FGB11] Joel E. Fischer, Chris Greenhalgh, and Steve Benford. Investigating episodes of mobile phone activity as indicators of opportune moments to deliver notifications. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, pages 181–190, New York, NY, USA, 2011. ACM.

[FHA$^+$05] James Fogarty, Scott E. Hudson, Christopher G. Atkeson, Daniel Avrahami, Jodi Forlizzi, Sara Kiesler, Johnny C. Lee, and Jie Yang. Predicting human interruptibility with sensors. *ACM Transactions in Computer-Human Interaction*, 12(1):119–146, March 2005.

[Fis36] R. A. Fisher. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(7):179–188, 1936.

[FNV10] A. Fleury, N. Noury, and M. Vacher. Introducing knowledge in the process of supervised classification of activities of daily living in health smart homes. In *Proceedings of the 12th IEEE International conference on e-Health Networking Applications and Services*, pages 322–329, July 2010.

[fRAF36] The Iris Dataset from R. A. Fisher, 1936. Available at `https://archive.ics.uci.edu/ml/datasets/Iris`.

[FS11] Robert Fisher and Reid Simmons. Smartphone interruptibility using density-weighted uncertainty sampling with reinforcement learning. In *Proceedings of the*

10th International Conference on Machine Learning and Applications, ICMLA '11, pages 436–441, Washington, DC, USA, 2011. IEEE Computer Society.

[fTD15] HedgeHog from TU Darmstadt, 2015. See `http://www.ess.tu-darmstadt.de/hedgehog`.

[FYB$^+$10] Joel E. Fischer, Nick Yee, Victoria Bellotti, Nathan Good, Steve Benford, and Chris Greenhalgh. Effects of content and time of delivery on receptivity to mobile interruptions. In *Proceedings of the 12th international conference on Human computer interaction with mobile devices and services*, MobileHCI '10, pages 103–112, New York, NY, USA, 2010. ACM.

[Gar10] Roman Garnett. *Learning from data streams with concept drift.* PhD thesis, Oxford University, 2010.

[GFG06] Pablo Granitto, Cesare Furlanello, and Flavia Gasperi. Recursive feature elimination with random forest for ptr-ms analysis of agroindustrial products. *Chemometrics and Intelligent Laboratory Systems*, 2006.

[GFR06] João Gama, Ricardo Fernandes, and Ricardo Rocha. Decision trees for mining data streams. *Intelligent Data Analysis*, 10(1):23–45, June 2006.

[Gla15] Noah Glass, 2015. See `https://twitter.com/noah`.

[Glu08] Jason M. Glushakow. The detrimental effect of phone interruptions on complex task performance. poster, 2008.

[Gou95] John D. Gould. Collection. In *Human-computer Interaction*, chapter How to Design Usable Systems, pages 93–121. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[GS09] Tobias Grundgeiger and Penelope Sanderson. Interruptions in healthcare: theoretical views. *International journal of medical informatics*, 78(5):293–307, 2009.

[GSB14] Dawud Gordon, Markus Scholz, and Michael Beigl. Group activity recognition using belief propagation for wearable devices. In *Proceedings of the 2014 ACM International Symposium on Wearable Computers*, pages 3–10. ACM, 2014.

[GvB+14] João Gama, Indrė Žliobaitė, Albert Bifet, Mykola Pechenizkiy, and Abdelhamid Bouchachia. A survey on concept drift adaptation. *ACM Computing Surveys*, 46(4):44:1–44:37, March 2014.

[GZW11] Bin Guo, Daqing Zhang, and Zhu Wang. Living with internet of things: The emergence of embedded intelligence. In *Proceedings of 4th International Conference on Cyber, Physical and Social Computing*, pages 297–304, Oct 2011.

[Han09] David J. Hand. Measuring classifier performance: a coherent alternative to the area under the roc curve. *Machine Learning*, 77(1):103–123, 2009.

[HAW+10] Holger Harms, Oliver Amft, Rene Winkler, Johannes Schumm, Martin Kusserow, and Gerhard Tröster. Ethos: Miniature orientation sensor for wearable human motion analysis. In *Proceedings of IEEE Sensors Conference*. IEEE, 2010.

[HCS05] Dan Hong, Dickson K. W. Chiu, and Vincent Y. Shen. Requirements elicitation for the design of context-aware applications in a ubiquitous environment. In *Proceedings of the 7th International Conference on Electronic Commerce*, ICEC '05, pages 590–596, New York, NY, USA, 2005. ACM.

[HI05] Joyce Ho and Stephen S. Intille. Using context-aware computing to reduce the perceived burden of interruptions from mobile devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, pages 909–918, New York, NY, USA, 2005. ACM.

[HLP97] Martin G. Helander, Thomas K. Landauer, and Prasad V. Prabhu, editors. *Handbook of Human-Computer Interaction.* Elsevier Science Inc., New York, NY, USA, 2nd edition, 1997.

[HM06] Jonna Häkkilä and Jani Mäntyjärvi. Developing design guidelines for context-aware mobile applications. In *Proceedings of the 3rd International Conference on*

Mobile Technology, Applications and Systems, Mobility '06, New York, NY, USA, 2006. ACM.

[HMB12] Negar Hariri, Bamshad Mobasher, and Robin Burke. Context-aware music recommendation based on latent topic sequential patterns. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, pages 131–138, New York, NY, USA, 2012. ACM.

[HMM06] Annika Hinze, Petra Malik, and Robi Malik. Interaction design for a mobile context-aware system using discrete event modelling. In *Proceedings of the 29th Australasian Computer Science Conference*, ACSC '06, pages 257–266, Darlinghurst, Australia, 2006. Australian Computer Society, Inc.

[HRVM11] Juan David Hincapié-Ramos, Stephen Voida, and Gloria Mark. A design space analysis of availability-sharing systems. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 85–96. ACM, 2011.

[HSD01] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '01, pages 97–106, New York, NY, USA, 2001. ACM.

[HTM11] Apiwat Henpraserttae, Surapa Thiemjarus, and Sanparith Marukatat. Accurate activity recognition using a mobile phone regardless of device orientation and location. In *Body Sensor Networks*, pages 41–46. IEEE, 2011.

[HW99] Michael Harries and New South Wales. Splice-2 comparative evaluation: Electricity pricing, 1999.

[IGD11] Elena Ikonomovska, João Gama, and Sašo Džeroski. Learning model trees from evolving data streams. *Data Mining Knowledge Discovery*, 23(1):128–168, July 2011.

[Inc15] Fitbit Inc., 2015. See `http://www.fitbit.com/`.

[JSS$^+$13] Shahram Jalaliniya, Jeremiah Smith, Miguel Sousa, Lars Büthe, and Thomas Pederson. Touch-less interaction with medical images using hand & foot gestures. In *Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication*, pages 1265–1274. ACM, 2013.

[JtCCT07] R. Jowell and the Central Coordinating Team. European social survey 2002/2003; 2004/2005; 2006/2007. Technical report, London: Centre for Comparative Social Surveys, London City University, 2007.

[KA76] Sidney Katz and C Amechi Akpom. A measure of primary sociobiological functions. *International journal of health services*, 6(3):493–508, 1976.

[KBDG04] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *Proceedings of the 30th International Conference on Very Large Data Bases*, VLDB '04, pages 180–191. VLDB Endowment, 2004.

[KC05] Ashraf Khalil and Kay Connelly. Improving cell phone awareness by using calendar information. In *Proceedings of the IFIP TC13 international conference on Human-Computer Interaction*, INTERACT'05, pages 588–600, Berlin, Heidelberg, 2005. Springer-Verlag.

[KH07] Ashish Kapoor and Eric Horvitz. Principles of lifelong learning for predictive user modeling. In *Proceedings of the 11th International Conference on User Modeling*, UM '07, pages 37–46, Berlin, Heidelberg, 2007. Springer-Verlag.

[KH08] Ashish Kapoor and Eric Horvitz. Experience sampling for building predictive user models: A comparative study. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 657–666, New York, NY, USA, 2008. ACM.

[KHA99] Mark G. Kelly, David J. Hand, and Niall M. Adams. The impact of changing populations on classifier performance. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 367–371, New York, NY, USA, 1999. ACM.

[KS06] Nicky Kern and Bernt Schiele. Towards personalized mobile interruptibility estimation. In *Proceedings of the 2nd international conference on Location and Context Awareness*, LoCA'06, pages 134–150, Berlin, Heidelberg, 2006. Springer-Verlag.

[Kun07] Ludmila Kuncheva, 2007. Available at `http://pages.bangor.ac.uk/~mas00a/EPSRC_simulation_framework/changing_environments_stage1a.htm`.

[KZ09] Ludmila Kuncheva and Indre Zliobaite. On the window size for classification in changing environments. *Intelligent Data Analysis*, 13(6):861–872, 2009.

[LB01] E. Levina and P. Bickel. The earth mover's distance is the mallows distance: some insights from statistics. In *Proceedings of the 8th IEEE International Conference on Computer Vision*, volume 2, pages 251–256 vol.2, 2001.

[LBGK12] Luis A. Leiva, Matthias Böhmer, Sven Gehring, and Antonio Krüger. Back to the app: the costs of mobile application interruptions. In *Mobile HCI*, pages 291–294, 2012.

[Lin13] Patrick Lindstrom. *Handling Concept Drift in the Context of Expensive Labels.* PhD thesis, Dublin Institute of Technology, 2013.

[LKLZ10] Kaisen Lin, Aman Kansal, Dimitrios Lymberopoulos, and Feng Zhao. Energy-accuracy trade-off for continuous mobile device location. In *Proceedings of the 8th International Conference on Mobile Systems, Applications, and Services*, MobiSys '10, pages 285–298, New York, NY, USA, 2010. ACM.

[LOIP10] Tom Lovett, Eamonn O'Neill, James Irwin, and David Pollington. The calendar as a sensor: Analysis and improvement using data fusion with social networks and location. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, UbiComp '10, pages 3–12, New York, NY, USA, 2010. ACM.

[LSNJ11] T.H. Laine, E. Sutinen, E. Nygren, and M. Joy. Rapid improvement of technology integration in context-aware learning spaces. In *AFRICON, 2011*, pages 1–6, Sept 2011.

[LZL14] Ning Lu, Guangquan Zhang, and Jie Lu. Concept drift detection via competence models. *Artificial intelligence*, 209:11–28, 2014.

[Man15] Smart Call Manager, 2015. Available at `https://play.google.com/store/apps/details?id=com.rapidsoftsystems.scm&hl=en`.

[Mar11] Andreas Markitanis. Learning mobile user behaviours. Master's thesis, Imperial College London, 2011.

[Mat75] Brian W Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[MCRL11] Andreas Markitanis, Domenico Corapi, Alessandra Russo, and C Emil Lupu. Learning user behaviours in real mobile domains. In *Proceedings of the 21st International Conference on Inductive Logic Programming (ILP 2011)*, 2011.

[mdt15] matplotlib development team, 2015. Available at `http://matplotlib.org/`.

[MGH05] Gloria Mark, Victor M Gonzalez, and Justin Harris. No task left behind?: examining the nature of fragmented work. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 321–330. ACM, 2005.

[Mid07] Catherine A Middleton. Illusions of Balance and Control in an Always-On Environment: A Case Study of BlackBerry Users. *Continuum: Journal of Media & Cultural Studies*, 21(2):165–178, 2007.

[Min11] Leandro Lei Minku. *Online ensemble learning in the presence of concept drift.* PhD thesis, University of Birmingham, 2011.

[ML02] D. C. McFarlane and K. A. Latorella. The scope and importance of human interruption in human-computer interaction design. *Human-Computer Interaction*, 2002.

[MN13] George Wamamu Musumba and Henry O Nyongesa. Context awareness in mobile computing: A review. *International Journal of Machine Learning and Applications*, 2(1):5–pages, 2013.

[MRGS14] Charlotte Magnusson, Kirsten Rassmus-Gröhn, and Delphine Szymczak. Exploring history: A mobile inclusive virtual tourist guide. In *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational*, NordiCHI '14, pages 69–78, New York, NY, USA, 2014. ACM.

[MS05] Stefan Marti and Chris Schmandt. Giving the caller the finger: collaborative responsibility for cellphone interruptions. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '05, pages 1633–1636, New York, NY, USA, 2005. ACM.

[MVH$^+$04] Deborah L McGuinness, Frank Van Harmelen, et al. Owl web ontology language overview. *W3C recommendation*, 10(10):2004, 2004.

[MWY10] Leandro L. Minku, Allan P. White, and Xin Yao. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Transactions on Knowledge and Data Engineering*, 22(5):730–742, 2010.

[Nes14] Nest, 2014. See `https://nest.com/downloads/press/documents/enhanced-auto-schedule-white-paper.pdf`.

[Nes15] Nest, 2015. See `https://nest.com/thermostat/life-with-nest-thermostat/#meet-the-nest-learning-thermostat`.

[NJ02] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 841–848. MIT Press, 2002.

[NYO05] Kyosuke Nishida, Koichiro Yamauchi, and Takashi Omori. Ace: Adaptive classifiers-ensemble system for concept-drifting environments. In *Proceedings of* *the 6th International Conference on Multiple Classifier Systems*, MCS'05, pages 176–185, Berlin, Heidelberg, 2005. Springer-Verlag.

[OTN14] Tadashi Okoshi, Hideyuki Tokuda, and Jin Nakazawa. Attelia: sensing user's attention status on smart phones. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 139–142, 2014.

[PA03] Anders K. Petersen and Agnar Aamodt. Case-based situation assessment in a mobile context-aware system. In *Artificial Intelligence in Mobile Systems (AIMS 2003), Universität des Saarlandes*, pages 41–49, 2003.

[PCdO14] Martin Pielot, Karen Church, and Rodrigo de Oliveira. An in-situ study of mobile phone notifications. In *Proceedings of the 16th International Conference on Human-computer Interaction with Mobile Devices and Services*, MobileHCI '14, pages 233–242, New York, NY, USA, 2014. ACM.

[PdOKO14] Martin Pielot, Rodrigo de Oliveira, Haewoon Kwak, and Nuria Oliver. Didn't you see my message?: Predicting attentiveness to mobile instant messages. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 3319–3328, New York, NY, USA, 2014. ACM.

[Pec06] Mykola Pechenizkiy, 2006. Available at `http://www.win.tue.nl/~mpechen/data/DriftSets/`.

[Pie14] Martin Pielot. Large-scale evaluation of call-availability prediction. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14, pages 933–937, New York, NY, USA, 2014. ACM.

[PM14a] Veljko Pejovic and Mirco Musolesi. Anticipatory Mobile Computing for Behaviour Change Interventions. In *Proceedings of the 3rd ACM Workshop on Mobile Systems for Computational Social Science*, Seattle, WA, USA, September 2014.

[PM14b] Veljko Pejovic and Mirco Musolesi. InterruptMe: Designing Intelligent Prompting Mechanisms for Pervasive Applications. In *Proceedings of the 2014 ACM*

International Joint Conference on Pervasive and Ubiquitous Computing (ACM UbiComp'14), Seattle, WA, USA, September 2014.

[PTG14] Umberto Panniello, Alexander Tuzhilin, and Michele Gorgoglione. Comparing context-aware recommender systems in terms of accuracy and diversity. *User Modeling and User-Adapted Interaction*, 24(1-2):35–65, 2014.

[RB11] Daniele Riboni and Claudio Bettini. Cosar: Hybrid reasoning for context-aware activity recognition. *Personal Ubiquitous Comput.*, 15(3):271–289, March 2011.

[RDV11] Stephanie Rosenthal, Anind K. Dey, and Manuela Veloso. Using decision-theoretic experience sampling to build personalized mobile phone interruption models. In *Proceedings of the 9th international conference on Pervasive computing*, Pervasive'11, pages 170–187, Berlin, Heidelberg, 2011. Springer-Verlag.

[Ref15] Sports Reference, 2015. Available at `http://www.sports-reference.com/`.

[RKSM14] Valentin Radu, Panagiota Katsikouli, Rik Sarkar, and Mahesh K. Marina. A semi-supervised learning approach for robust indoor-outdoor detection with smartphones. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, SenSys '14, pages 280–294, New York, NY, USA, 2014. ACM.

[RLGB$^+$14] Vijay Rajanna, Raniero Lara-Garduno, Dev Jyoti Behera, Karthic Madanagopal, Daniel Goldberg, and Tracy Hammond. Step up life: A context aware health assistant. In *Proceedings of the Third ACM SIGSPATIAL International Workshop on the Use of GIS in Public Health*, HealthGIS '14, pages 21–30, New York, NY, USA, 2014. ACM.

[RMM$^+$13] Leonardo Rocha, Fernando Mouro, Hilton Mota, Thiago Salles, Marcos Andr Gonalves, and Wagner Meira Jr. Temporal contexts: Effective text classification in evolving document collections. *Information Systems*, 38(3):388 – 409, 2013.

[RSHI08] N. Ravi, J. Scott, Lu Han, and L. Iftode. Context-aware battery management for mobile phones. In *Proceedings of the 6th Annual IEEE International Conference on Pervasive Computing and Communications*, pages 224–233, March 2008.

[SAG^{+}93] B.N. Schilit, N. Adams, R. Gold, M.M. Tso, and R. Want. The parctab mobile computing system. In *Proceedings of the 4th Workshop on Workstation Operating Systems*, pages 34–39, Oct 1993.

[SAT^{+}99] Albrecht Schmidt, Kofi Asante Adoo, Antti Takaluoma, Urop Tuomela, Kristof Van Laerhoven, and Walter Van De Velde. Advanced interaction in context. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing*, pages 89–101. Springer Verlag, 1999.

[SAW94] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proceedings of the 1st Workshop on Mobile Computing Systems and Applications*, WMCSA '94, pages 85–90, Washington, DC, USA, 1994. IEEE Computer Society.

[Sch87] Jeffrey Curtis Schlimmer. *Concept Acquisition Through Representational Adjustment*. PhD thesis, University of California, Irvine, 1987. AAI8724747.

[Sch05] Albrecht Schmidt. Interactive context-aware systems interacting with ambient intelligence. *Ambient intelligence*, 159, 2005.

[Sch14] Albrecht Schmidt. *Context-Aware Computing: Context-Awareness, Context-Aware User Interfaces, and Implicit Interaction.* The Encyclopedia of Human-Computer Interaction, 2nd Ed, 2014. See `www.interaction-design.org/encyclopedia/context-aware_computing.html`.

[SD14] Jeremiah Smith and Naranker Dulay. Ringlearn: Long-term mitigation of disruptive smartphone interruptions. In *Proceedings of the IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 27–35. IEEE, 2014.

[SDT^{+}13] Jeremiah Smith, Naranker Dulay, Máté Attila Tóth, Oliver Amft, and Yanxia Zhang. Exploring concept drift using interactive simulations. In *Proceedings of the 2013 IEEE International Conference on Pervasive Computing and Communications Workshops*, pages 49–54. IEEE, 2013.

[Sen14] Sevil Sen. Using instance-weighted naive bayes for adapting concept drift in masquerade detection. *International Journal of Information Security*, 13(6):583–590, 2014.

[Sen15] Shimmer Sensing, 2015. See `http://www.shimmersensing.com/`.

[SFH$^+$11] Edward S Sazonov, George Fulk, James Hill, Yves Schutz, and Raymond Browning. Monitoring of posture allocations and activities by a shoe-based wearable sensor. *Biomedical Engineering, IEEE Transactions on*, 58(4):983–990, 2011.

[SG86] Jeffrey C. Schlimmer and Jr. Granger, Richard H. Incremental learning from noisy data. *Machine Learning*, 1(3):317–354, 1986.

[SK01] W. Nick Street and Yongseog Kim. A streaming ensemble algorithm (sea) for large-scale classification. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '01, pages 377–382, New York, NY, USA, 2001. ACM.

[sklc15a] sci-kit learn contributors, 2015. Available at `http://scikit-learn.org/stable/index.html`.

[sklc15b] sci-kit learn contributors, 2015. See `http://scikit-learn.org/stable/modules/tree.html`.

[Sle15] Luna Sleep, 2015. See `http://lunasleep.com/`.

[SLM$^+$14] Jeremiah Smith, Anna Lavygina, Jiefei Ma, Alessandra Russo, and Naranker Dulay. Learning to recognise disruptive smartphone notifications. In *Proceedings of the 16th International Conference on Human-computer Interaction with Mobile Devices and Services*, MobileHCI '14, pages 121–124, New York, NY, USA, 2014. ACM.

[SLP04] Thomas Strang and Claudia Linnhoff-Popien. A context modeling survey. In *Workshop Proceedings*, 2004.

[Smi15] Jeremiah Smith, 2015. Available at `www.doc.ic.ac.uk/~jeremiah/code`.

[SRLD14] Jeremiah Smith, Alessandra Russo, Anna Lavygina, and Naranker Dulay. When did your smartphone bother you last? In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, UbiComp '14 Adjunct, pages 409–414, New York, NY, USA, 2014. ACM.

[STA$^+$12] Yoshitaka Sakurai, Kouhei Takada, Marco Anisetti, Valerio Bellandi, Paolo Ceravolo, Ernesto Damiani, and Setsuo Tsuruta. Toward sensor-based context aware systems. *Sensors*, 12(1):632–649, 2012.

[Sto15] Christopher Stone, 2015. See `https://twitter.com/biz`.

[SZL$^+$10] Lin Sun, Daqing Zhang, Bin Li, Bin Guo, and Shijian Li. Activity recognition on an accelerometer embedded mobile phone with varying positions and orientations. In *Proceedings of the 7th International Conference on Ubiquitous Intelligence and Computing*, UIC'10, pages 548–562, Berlin, Heidelberg, 2010. Springer-Verlag.

[tH07] G. H. (Henri) ter Hofte. Xensible interruptions from your mobile phone. In *Proceedings of the 9th international conference on Human computer interaction with mobile devices and services*, MobileHCI '07, pages 178–181, New York, NY, USA, 2007. ACM.

[TM13] Robert J Teather and I Scott MacKenzie. Effects of user distraction due to secondary calling and texting tasks. In *Proceedings of the International Conference on Multimedia and Human Computer Interaction*, pages 1–8. International ASET, Inc, 2013.

[tNAM81] The Audubon Society Field Guide to North American Mushrooms, 1981. Available at `https://archive.ics.uci.edu/ml/datasets/Mushroom`.

[TPCP06a] A. Tsymbal, M. Pechenizkiy, P. Cunningham, and S. Puuronen. Handling local concept drift with dynamic integration of classifiers: Domain of antibiotic resistance in nosocomial infections. In *Proceedings of the 19th IEEE International Symposium on Computer-Based Medical Systems*, 2006.

[TPCP06b] Alexey Tsymbal, Mykola Pechenizkiy, Padraig Cunningham, and Seppo Puuronen. Handling local concept drift with dynamic integration of classifiers: Domain of antibiotic resistance in nosocomial infections. In *Proceedings of the 19th IEEE International Symposium on Computer-Based Medical Systems*, pages 679–684, 2006.

[TPCP08] Alexey Tsymbal, Mykola Pechenizkiy, Pádraig Cunningham, and Seppo Puuronen. Dynamic integration of classifiers for handling concept drift. *Information Fusion*, 9(1):56–68, 2008.

[VDS02] Roel Vertegaal, Connor Dickie, and Changuk Sohn. Designing attentive cell phones using wearable eyecontact sensors. In *Proceedings of Extended Abstracts on Human Factors in Computing Systems*, pages 646–647. ACM Press, 2002.

[Wei91] Mark Weiser. The computer for the twenty-first century. *Scientific American*, 9:94–100, September 1991.

[WHFG92] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The active badge location system. *ACM Transactions on Information Systems (TOIS)*, 10(1):91–102, 1992.

[WK93] Gerhard Widmer and Miroslav Kubat. Effective learning in dynamic environments by explicit context tracking. In *Proceedings of the European Conference on Machine Learning*, ECML '93, pages 227–243, London, UK, UK, 1993. Springer-Verlag.

[WK96] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.

[WM97] D. Randall Wilson and Tony R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6(1):1–34, January 1997.

[WM03] D. Randall Wilson and Tony R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

[Wor15] Computer World, 2015. Available at `http://goo.gl/KC99dP`.

[WRB14] Daniel T. Wagner, Andrew Rice, and AlastairR. Beresford. Device analyzer: Understanding smartphone usage. In *Mobile and Ubiquitous Systems: Computing, Networking, and Services*, volume 131 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 195–208. Springer International Publishing, 2014.

[WWR$^+$10] Johanna I. Westbrook, Amanda Woods, Marilyn I. Rob, William Dunsmuir, and Richard O Day. Association of interruptions with an increased risk and severity of medication administration errors. *Archives of Internal medicine*, 170(8):683, 2010.

[YLS$^+$14] O. Yurur, C. Liu, Z. Sheng, V. Leung, W. Moreno, and K. Leung. Context-awareness for mobile sensing: A survey and future directions. *Communications Surveys Tutorials, IEEE*, (99):1–1, 2014.

[ZK09] Indre Zliobaite and Ludmila I. Kuncheva. Determining the training window for small sample size classification with concept drift. In *Proceedings of the 2009 IEEE International Conference on Data Mining Workshops*, ICDMW '09, pages 447–452, Washington, DC, USA, 2009. IEEE Computer Society.

[Zli09] Indre Zliobaite. Combining time and space similarity for small size learning under concept drift. In *Foundations of Intelligent Systems*, volume 5722 of *Lecture Notes in Computer Science*, pages 412–421. Springer Berlin Heidelberg, 2009.

[Zli10] Indre Zliobaite. *Adaptive Training Set Formation.* PhD thesis, Vilnius University, Lithuania, 2010.

[Zli13] Indre Zliobaite. How good is the electricity benchmark for evaluating concept drift adaptation. *Computing Research Repository*, abs/1301.3524, 2013.

Appendices

Appendix A

Appendix

A.1 General notation

Vectors are noted $\vec{x}$ and are assumed to be in column format. Row vectors are represented using the transpose of a column vector $\vec{x}^T$

- tp: true positive(correct classification of a positive instance as positive), tn: true negative(classification of a negative instance as negative), fp: false positive(incorrect classification of a negative instance as positive), fn: false negative(incorrect classification of a positive instance as negative)

  sensitivity or recall $= \frac{tp}{tp+fn}$ i.e. the proportion of positive instances we have classified as such, 'how much of the positive instances we have found'

  specificity $= \frac{tn}{tn+fp}$ i.e. the proportion of the negative instances that have been classified as such, 'how much of the negative instances we have found'

  precision $= \frac{tp}{tp+fp}$ i.e. the proportion of the instances classified as positive that actually are, 'how much we can trust our classifier when it says something is a positive instance'

  accuracy $= \frac{tp+tn}{tp+tn+fp+fn}$ i.e the proportion of correct classifications, 'how much of the time our classifier is right'

area under receiver operating characteristic curve (AUC) gives the tradeoff between true positive rate and false positive rate, for the range of values of a discriminative threshold for a classifier. It tells us how much a certain *sensitivity* performance will cost in terms of false positive rate (proportion of instances classified positive that are actually negative)

f-measure $= 2\frac{precision*recall}{precision+recall}$ i.e a score that weighs equally precision and recall.

A.2 Machine Learning Core Concepts

Definition 1. Random variable

A random variable X is a variable that can take on different values each associated with a probability $P(X = a) \in [0, 1]$ for discrete variables and $P(X \in A) \in [0, 1],\ A \in \mathbb{R}$ for continuous variables. If X is continuous $P(X = a) = 0$ for any $a \in \mathbb{R}$.

Definition 2. Probability mass and Probability density

A Probability mass function $p(x)$ gives the probability that a discrete random variable will take on a certain value $P(X = a) = p(a)$. A probability density function $p(x)$ is used to compute the probability that a continuous random variable falls in the interval A, $P(X \in A) = \int_A p(x)dx$, it is thus the area under the probability density function curve on the interval A.

Definition 3. Joint probability distributions

Without loss of generality, given two random variables X and Y, the joint probability distribution $p(x, y)$ describes the common dynamics of two variables. In the discrete case, $p(X = a, Y = b)$ gives the probability of the values $X = a$ and $Y = b$ occurring together, $p(x, y)$ is called the joint probability mass function. In the continuous case, $p(x, y)$ is the joint probability density function which enables us to compute the probability that $X \in A$ and $Y \in B$ for some intervals A,B. $P(X \in A, Y \in B) = \int_A \int_B p(x, y)dydx$.

Definition 4. Marginal probability distributions

Without loss of generality, given two random variables X and Y, the marginal distribution $p_X(x)$ is the probability distribution of X without reference to Y. In the discrete case, the marginal probability mass function $p_X(x) = \Sigma_y\ p(X = x, Y = y)$ where $p(x, y)$ is the joint probability mass function, it gives us the overall probability that X will take on the value x. In the continuous case, $p_X(x) = \int_y p(x, y)dy$ where $p(x, y)$ is the joint probability density function, it can be used to compute the overall probability that X will fall within a certain range.

Definition 5. Conditional probability distributions

In the type of context-aware system we are interested in, learning tasks have a random variable vector $X = (X_1, ..., X_n)$ describing the situation or context sensed by the system (X_i can be discrete or continuous), and a discrete random variable Y representing the labels based on which the system might take an action. For a specific input $\vec{x}_a$, we want to know $P(Y = y_i|X = \vec{x}_a)$ which is the conditional probability of the label being y_i given the input $\vec{x}_a$. $P(Y = y_i|X = \vec{x}_a) = \frac{p(\vec{x}_a, y_i)}{p_X(\vec{x}_a)}$ where $p(x, y)$ is the joint probability distribution function and $p_X(x)$ the marginal distribution function. In practice though, the conditional distribution $P(Y = y_i|X = \vec{x})$ is usually directly modelled and fitted using the training data or implicitly estimated using a heuristic.

Definition 6. The product rule and Bayes' rule

The probability product rule states that: $p(x, y) = p(y|x)p(x)$ implying that $p(y|x) = \frac{p(x|y)p(y)}{p(x)}$ which is Baye's rule. It applies to discrete, continuous and mixed random variables as well as random variables vectors.

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4
DIOC	10/7	15/1	15/9	5/3	5/3	5/3
TG	64/40/F	64/40/F	64/40/F	729/3/F	1/10/T	1/10/F
TC	15/1/50/F	15/1/50/F	30/5/1/T	10/2/3/F	3/2/5/F	3/2/5/F
FISH	60/0.5	200/0.5	100/0.5	200/0.5	10/1	10/1
Windowed	300	300	300	300	10	10

Table A.1: Hyperparameters for DIOC (chunkSize/k), TG (f/winSize/trainLocal), /3 TC ($\epsilon/\theta_{pts}/winSize/trainLocal$), FISH ($\alpha_t/winSize$) and Windowed ($winSize$) on the F-RingLearn dataset with a Logistic Regression base learner

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4
DIOC	25/11	25/11	25/11	15/7	5/3	5/3
TG	64/40/F	64/40/F	64/3/F	1/100/F	729/3/F	64/20/F
TC	15/1/50/F	15/1/50/F	0.5/1/5/F	20/1/2/F	20/2/2/F	2.5/2/25/F
FISH	60/0.5	100/0.9	60/0.5	60/0.5	20/0	20/0
Windowed	300	300	100	100	300	300

Table A.2: Hyperparameters for DIOC (chunkSize/k), TG (f/winSize/trainLocal), /3 TC ($\epsilon/\theta_{pts}/winSize/trainLocal$), FISH ($\alpha_t/winSize$) and Windowed ($winSize$) on the L-RingLearn dataset with a Logistic Regression base learner

A.3 Individual forgetting strategy hyperparameters for the results of Section 4.3.3

The tables listing parameters for each experiment appear in the same order as they do in Section 4.3.3.

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4
DIOC	10/7	50/3	25/11	5/1	5/3	15/3
TG	64/2/F	64/2/F	64/10/F	1/5/F	1/3/F	729/2/T
TC	30/1/2/F	30/1/2/F	30/1/7/F	20/2/10/T	3/2/7/T	20/2/10/T
FISH	60/0.5	60/0.5	1/0.5	1/0.5	1/0.5	1/0.5
Windowed	300	300	300	5	3	20

Table A.3: Hyperparameters for DIOC (chunkSize/k), TG (f/winSize/trainLocal), /3 TC ($\epsilon/\theta_{pts}/winSize/trainLocal$), FISH ($\alpha_t/winSize$) and Windowed ($winSize$) on the J-RingLearn dataset with a Logistic Regression base learner

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4
DIOC	10/7	50/3	25/11	5/1	5/3	15/3
TG	729/10/F	729/10/F	4096/3/F	4096/2/T	729/5/F	729/10/F
TC	1000/3/150/F	1000/3/150/F	40/3/100/F	40/3/10/F	40/3/200/F	800/2/50/F
FISH	200/0.3	200/0.3	200/0.3	40/0.3	50/0.5	50/0.5
Windowed	300	600	600	300	300	350

Table A.4: Hyperparameters for DIOC (chunkSize/k), TG (f/winSize/trainLocal), /3 TC ($\epsilon/\theta_{pts}/winSize/trainLocal$), FISH ($\alpha_t/winSize$) and Windowed ($winSize$) on the 784c-D.-A. dataset with a Logistic Regression base learner

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4
DIOC	150/7	150/7	100/7	25/1	250/3	250/3
TG	1/150/T/F	1/150/T(F)	1/75/T(F)	1/150/T(F)	1/75/T(F)	1/150/T(F)
TC	40/2/50/F	40/2/50/F	800/11/80/T	875/10/60/T	895/30/100/T	745/5/130/T
FISH	50/0.9	50/0.9	60/1	40/0.5	100/0.3	200/0.5
Windowed	300	150	75	150	75	150

Table A.5: Hyperparameters for DIOC (chunkSize/k), TG (f/winSize/trainLocal), /3 TC ($\epsilon/\theta_{pts}/winSize/trainLocal$), FISH ($\alpha_t/winSize$) and Windowed ($winSize$) on the b9ae-D.-A. dataset with a Logistic Regression base learner

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4
DIOC	400/7	400/7	600/9	150/1	250/1	200/1
TG	32/900/T	32/900/T	32/150/T	4096/450/T	4096/450/T	4096/450/T
TC	40/2/100/F	40/2/100/F	775/28/250/T	745/16/70/T	1900/7/225/T	1650/5/250/T
FISH	100/0	100/0	200/0	100/1	200/0.3	200/0.3
Windowed	1	1	600	100	400	200

Table A.6: Hyperparameters for DIOC (chunkSize/k), TG (f/winSize/trainLocal), /3 TC ($\epsilon/\theta_{pts}/winSize/trainLocal$), FISH ($\alpha_t/winSize$) and Windowed ($winSize$) on the c260-D.-A. dataset with a Logistic Regression base learner

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4
DIOC	250/7	250/7	50/9	75/9	50/9	200/7
TG	1024/585/F	1024/585/F	1/600/T	3125/330/F	1024/135/T	243/495/F
TC	40/3/450/T	40/3/450/T	350/15/550/F	500/3/300/F	1150/2/600/T	200/25/500/F
FISH	7/0	7/0	600/0	500/0.3	500/0.5	500/0.9
Windowed	700	700	600	500	625	625

Table A.7: Hyperparameters for DIOC (chunkSize/k), TG (f/winSize/trainLocal), /3 TC ($\epsilon/\theta_{pts}/winSize/trainLocal$), FISH ($\alpha_t/winSize$) and Windowed ($winSize$) on the 784c-D.-A. dataset with a Decision Tree base learner

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4
DIOC	150/11	150/11	200/1	50/3	150/3	50/3
TG	900/30/200/T	900/30/200/T	3125/210/F	1/210/T	3125/450/T	3125/450/T
TC	900/30/200/T	900/30/200/T	200/3/100/F	300/30/350/T	450/30/200/F	350/25/200/T
FISH	3/0	3/0	200/0.9	200/0.3	200/1	200/1
Windowed	400	400	750	400	400	400

Table A.8: Hyperparameters for DIOC (chunkSize/k), TG (f/winSize/trainLocal), /3 TC ($\epsilon/\theta_{pts}/winSize/trainLocal$), FISH ($\alpha_t/winSize$) and Windowed ($winSize$) on the b9ae-D.-A. dataset with a Decision Tree base learner

Strategy	A_0	$A_{0.5}$	A_1	A_2	A_3	A_4
DIOC	400/1	400//1	400/7	200/9	400/1	200/9
TG	32/330/F	32/330/F	32/345/F	32/600/F	32/150/F	32/510/F
TC	200/6/700/F	200/6/700/F	200/8/650/F	200/6/350/F	600/9/300/F	500/5/200/F
FISH	200/0.5	200/0.5	100/0.5	200/0.9	100/0.3	200/0.5
Windowed	300	600	600	300	300	350

Table A.9: Hyperparameters for DIOC (chunkSize/k), TG (f/winSize/trainLocal), /3 TC ($\epsilon/\theta_{pts}/winSize/trainLocal$), FISH ($\alpha_t/winSize$) and Windowed ($winSize$) on the c260-D.-A. dataset with a Decision Tree base learner