Annals of Telecommunications
https://doi.org/10.1007/s12243-020-00776-1

Bi-directional online transfer learning: a framework

Helen McKay[1] · **Nathan Griffiths**[1] · **Phillip Taylor**[1] · **Theo Damoulas**[1,2] · **Zhou Xu**[3]

Received: 5 November 2019 / Accepted: 1 June 2020

Abstract

Transfer learning uses knowledge learnt in source domains to aid predictions in a target domain. When source and target domains are online, they are susceptible to concept drift, which may alter the mapping of knowledge between them. Drifts in online environments can make additional information available in each domain, necessitating continuing knowledge transfer both from source to target and vice versa. To address this, we introduce the Bi-directional Online Transfer Learning (BOTL) framework, which uses knowledge learnt in each online domain to aid predictions in others. We introduce two variants of BOTL that incorporate model culling to minimise negative transfer in frameworks with high volumes of model transfer. We consider the theoretical loss of BOTL, which indicates that BOTL achieves a loss no worse than the underlying concept drift detection algorithm. We evaluate BOTL using two existing concept drift detection algorithms: RePro and ADWIN. Additionally, we present a concept drift detection algorithm, Adaptive Windowing with Proactive drift detection (AWPro), which reduces the computation and communication demands of BOTL. Empirical results are presented using two data stream generators: the drifting hyperplane emulator and the smart home heating simulator, and real-world data predicting Time To Collision (TTC) from vehicle telemetry. The evaluation shows BOTL and its variants outperform the concept drift detection strategies and the existing state-of-the-art online transfer learning technique.

Keywords Online learning · Transfer learning · Concept drift

1 Introduction

Online learning (OL) is an important field of machine learning research which allows supervised learning to be conducted on data streams [9, 30]. Learning from data streams can be challenging, particularly in environments that are non-stationary in their nature, which can cause concept drift [9]. Concept drift occurs when the underlying concept changes over time, causing changes to the distribution of data, and requires predictive models to be updated or discarded to maintain effective predictions. To build accurate models, many real-world applications require large amounts of training data, which is often limited if concept drifts occur [24].

Transfer learning (TL) is another prominent field of machine learning research, which allows models to be learnt in domains where training data is readily available, and used where it is limited to build more effective predictors [24]. TL has typically been conducted offline, limiting its use in real-world online environments [36]. It may be desirable to use on-device learning to personalise the functionalities of user facing applications, where a rich history of data may not be available locally due to memory limitations, and drifts may be encountered frequently. Predictive performances could be enhanced using TL in an online setting by using knowledge learnt from other data streams to aid the target predictor.

The Online Transfer Learning (OTL) framework, developed by Zhao et al. [36], was proposed to enable TL to be used within an online setting. Current versions of OTL assume the source is in an offline environment [10, 12, 32], ignoring the possibility of concept drift occurring in a source domain.

In this paper, we propose the Bi-directional Online Transfer Learning (BOTL) framework,[1] which considers source

✉ Helen McKay
H.McKay@warwick.ac.uk

[1] Department of Computer Science, University of Warwick, Coventry, UK

[2] Department of Statistics, University of Warwick, Coventry, UK

[3] Jaguar Land Rover Research, Coventry, UK

[1] Available here: https://github.com/hmckay/BOTL

Published online: 06 October 2020

and target domains to be online. This has three benefits over existing approaches. Firstly, individual concepts are learnt in a source domain, using concept drift detection strategies, and transferred to other domains to improve their predictive performance [9].

Secondly, as new concepts are encountered in a source domain, additional knowledge of the new concept is transferred to other domains. Thirdly, knowledge can be transferred bi-directionally, enabling more effective predictions to be made in both source and target domains. Specifically, we:

- Introduce the BOTL framework, enabling each domain to benefit from online TL in a regression setting,
- Consider the theoretical loss of BOTL, showing predictions made by BOTL are no worse than the underlying concept drift detection algorithm,
- Introduce a novel drift detector, AWPro, which has combined benefits of RePro [34] and ADWIN [3], and
- Show the performance of BOTL exceeds an existing state-of-the-art online transfer learning technique and existing concept drift detection algorithms with no knowledge transfer using a variety of datasets.

We evaluate BOTL in a regression setting using two synthetic datasets and one real-world dataset containing both sudden and gradual drifts. We use BOTL in conjunction with three concept drift detection strategies to identify the underlying drifts occurring locally in each domain, namely RePro [34] and ADWIN [3], and a novel drift detection algorithm, Adaptive Windowing with Proactive drift detection (AWPro). AWPro combines key characteristics exhibited by ADWIN and RePro that are beneficial to the BOTL framework when used within applications that have computational and communication limitations.

We compare BOTL with a state-of-the-art online TL framework, the Generalised Online Transfer Learning (GOTL) framework, which assumes the source is offline [12].

The remainder of this paper is organised as follows. Section 2 outlines related work. Section 3 formulates the setting in which BOTL is used. Section 4 presents the proposed framework, and the theoretical loss of BOTL is presented in Section 5. Section 6 specifies the datasets used to investigate the applicability of underlying concept drift detection strategies, and evaluate the BOTL framework. Section 7 outlines the three concept drift detection strategies and discusses their limitations for use in BOTL. Section 8 presents empirical results of BOTL using a variety of datasets, highlighting the beneficial characteristics of concept drift detection strategies, robustness to noise, and applicability to real-world data. Finally, Section 9 concludes the paper.

2 Related work

Online TL combines OL and TL. The aim of TL is to use knowledge learnt for a predictive task in one domain, referred to as the source, to improve the effectiveness of predictions in another domain, referred to as the target [23]. There are three distinct types of TL: inductive, transductive, and unsupervised [1, 5, 24]. Inductive TL is used when source and target predictive tasks are different. Knowledge is transferred from the source to induce a supervised predictive function in the target [5]. Typically, large amounts of labelled target data are required to create a mapping between domains [24]. Unsupervised TL is applied in a similar way, but to unsupervised learning tasks, such as clustering [24]. Transductive TL is used when the source and target predictive tasks are the same, transferring knowledge to improve the predictive performance in a target domain where no labelled data is available [1]. TL can be further categorised as homogeneous, where the domains of source and target are the same, or heterogeneous, where they differ [36]. In this paper, we consider a homogeneous setting, and use inductive TL to improve the predictive performances within both source and target domains.

It is desirable for many modern applications, such as smart home heating systems, to predict future events from historical data. However, applications are often limited by memory constraints, preventing a complete history of data being retained [11]. Additionally, due to the dynamic and non-stationary environment of data streams, the underlying concept may evolve or drift over time [18]. Concept drift is a change in the distribution of the observed data, or a change in the mapping between observations and response variables [15]. If the underlying concept changes, the previously built model may no longer make effective predictions, requiring the model to be modified or re-learnt [9].

To maintain effective predictions, concept drift detection algorithms are frequently used in OL. Concept drift detection algorithms typically use a sliding window to maintain a subset of recent instances, usually used to update or rebuild the predictive model. Strategies to update a model include ensemble learning approaches, where the window of recent instances is used to create a new model and combined with previously learnt models to improve the predictive performance. Model predictions are aggregated; for example, Dynamic Weighted Majority (DWM) uses a mean weighted by the models' estimated performance [19].

Alternatively, concept drift detection algorithms such as ADWIN [3] use the window of data to detect concept drifts, and once a drift has been detected, a new predictive model can be learnt that represents the current concept independently of previous concepts. A challenge associated with these concept drift detection strategies is that every time a concept is encountered, a new model must be learnt,

and data must be collected to build each model, even if that concept has previously been encountered. RePro [33] uses an approach similar to ADWIN, but retains a history of concepts and concept transitions to prevent learning new models for recurring concepts [34]. This prevents the need to collect new data each time a recurring concept is encountered; however, data must still be collected to build models for new concepts. For many real-world applications, particularly those that are user facing, knowledge obtained from other data streams could enhance predictions when new concepts are encountered through the use of online TL.

Existing online TL frameworks aim to transfer knowledge learnt from an offline source to an online target for classification tasks. OTL [36] combines the offline source model with the online target model using a weighting mechanism that is updated with respect to the performance of the source and target models on a sliding window of data in the target domain. Other online TL frameworks [10, 17, 31] use similar strategies to combine transferred models specifically for classification tasks, and cannot easily be adapted to regressive settings. GOTL [12] extends the OTL weighting mechanism such that online TL can be used for both classification and regression. The weighting mechanism used by GOTL incrementally updates in steps to obtain weightings for source and target models. If the step size, Δ, used to modify the weights is small enough, the ensemble of source and target models approximates the optimal weight combination [11]. However, if the step size is too small, it may take substantial time for the weights to update to their desired values, making predictions unreliable during this period.

The field of online TL relates to Online Multi-task Learning (OMTL) [22, 25, 26], and Multistream Regression (MSR) [14]. MSR can be seen as a special case, where the source and target data streams are drawn from the same underlying distribution, and all concepts encountered in the target domain have previously been encountered in the source [6]. This means the models transferred from the source can be used to make predictions in the target without requiring a target learner. This is unrealistic for many real-world applications as although source and target domains may be similar, it is unlikely the data streams are drawn from the same distribution. The goal of OMTL is to minimise the cumulative global loss across all domains [21], whereas online TL aims to minimise the predictive losses within each individual domain. Considering loss in this way is beneficial when applied to tasks such as application personalisation, where each domain represents a different user, and prediction errors should be minimised for that specific individual.

Although online TL has been actively studied [10, 12, 31, 32, 35, 36], existing approaches assume the source is offline. We propose BOTL, which considers both source and target in online environments, as might be expected in real-world applications such as smart home heating, or vehicle personalisation such as Adaptive Cruise Control (ACC).

3 Problem formulation

Let domain D consist of a feature space χ, where $x_t \in \mathbb{R}^m$ is the instance observed at time t such that $x_t = \{x_{t_1}, \ldots, x_{t_m}\} \in \chi$. Given domain D, a task consists of the target response variable, $y \in Y$, where $y \in \mathbb{R}$, and a regression function, $f : \chi \rightarrow Y$, which is learnt to map observed data to the target concept [24]. The knowledge learnt in a source domain, D^S, can be transferred to the target domain, D^T, and used to enhance predictions [30].

Online TL aims to learn the target predictive function, f^T, that effectively predicts the response variable, $y_t^T \in Y^T$, for each instance, $x_t^T \in \chi^T$, observed in the target data stream, such that $\hat{y}_t^{\prime T_i} = f_i^T(x_t^T)$. Model transfer is used to enhance the target predictor by combining knowledge learnt in the local domain with knowledge learnt from other domains. For example, if we consider the scenario of application personalisation, where each domain represents an individual user, each instance, x_t, may describe the user's current environmental setting. If we wish to personalise application functionality by predicting some unknown value, y_t, we may be able to utilise knowledge learnt from another user, f_j^S, to enhance the predictive performance of the target learner. Identifying concept drift in the source, S, allows models to be transferred, f_j^S where $j = 1 \ldots k$, for each of the k concepts encountered in S.

BOTL aims to minimise the predictive error in the target domain by combining knowledge learnt from the target data stream with models previously learnt in a source domain. Focusing on minimising the loss with respect to the local, or target, domain makes BOTL highly applicable to the task of application personalisation, where predictions are made to benefit a specific individual. To achieve this, if we have a source domain, D^S, that has previously learnt models $f_1^S, \ldots, f_j^S$, and a target domain, D^T, that has previously learnt models $f_1^T, \ldots, f_i^T$, at time t, then models $f_1^S, \ldots, f_j^S$, should be made available to the target domain such that the target learner can benefit from the knowledge learnt in the source domain, D^S. As both domains are online, and knowledge transfer is bi-directional, the models $f_1^T, \ldots, f_i^T$ should also be made available to the source domain, D^S, such that the source learner can benefit from the knowledge learnt in the target domain, D^T.

In this paper, the source and target domains are considered to be homogeneous, such that they share the same underlying feature space, $\chi^S = \chi^T$, and $Y^S = Y^T$. Although the domains are homogeneous, the underlying concepts to be learnt within source and target domains may not be equivalent; therefore, models from a source domain

may not be relevant to the current target concept. BOTL provides a mechanism to combine models and maximise the impact of transferred models on the target. In presenting BOTL, we use the notation detailed in Table 1.

4 Bi-directional Online Transfer Learning

To utilise knowledge of distinct concepts, BOTL hinges upon a sliding window-based concept drift detection algorithm. BOTL uses drift detection strategies that employ batch learners to create base models, f_i, from a window of data within a domain. With small windows, batch learners are susceptible to overfitting; however, larger window sizes can cause a reduction in sensitivity to gradual drifts [9]. Alternatively incremental, or online, learners can be used. However, during periods of gradual drifts, data belonging to a new concept may be used to incrementally update the base learner, preventing a drift from being detected. This is problematic for the BOTL framework as a pair of consecutive concepts present in one domain may not exist in another domain, meaning transferred models may be less effective than if they were learnt using data from individual concepts. In this paper, we use three concept drift detection algorithms, RePro [34], ADWIN [3], and a novel drift detection algorithm, AWPro, detailed in Section 7.

Table 1 Notation

	Definition
D^{α}	Domain α: target, T, or source, S
χ^{α}	Data stream α
Y^{α}	Response variables of α
$x_t \in \chi^{\alpha}$	The t^{th} observed instance in χ^{α}
$y_t \in Y^{\alpha}$	The response variable of instance x_t
M	Knowledge base of models
$f^{\alpha}_{\beta} : \chi^{\alpha} \rightarrow Y^{\alpha}$	Model β learnt in domain α
$F^{M} : \chi^{T} \rightarrow Y^{T}$	Meta-model of M
$\hat{y}_t$	Prediction using $F^{M}(x_t)$
$\hat{y}_t^{\prime\alpha\beta}$	Prediction using $f^{\alpha}_{\beta}(x_t)$
W	Sliding window of instances
W_{max}	Maximum window size (RePro)
W_{min}	Minimum window size (ADWIN, AWPro)
err_t	Predictive error of instance x_t
err_W	Predictive error across W
λ_l	Loss threshold (RePro)
λ_d	Drift threshold (RePro)
λ_r	Recurrence threshold (AWPro)
δ	Confidence value (ADWIN, AWPro)
λc_{perf}	Performance culling threshold
λc_{MI}	Mutual Information culling threshold

Although BOTL uses knowledge learnt from other domains to improve the predictive performance of the target learner, concept drift detection is conducted solely using the locally learnt model. Conducting drift detection independently of any knowledge transfer is necessary as the use of transferred knowledge may enhance the predictive performance across the current window of target data, hindering drift detection.

A common challenge encountered by TL frameworks is negative transfer [24], which occurs when an ineffective model is transferred between domains. To address this, BOTL adopts the notion of model stability, introduced by Yang et al. [33], to determine if a locally learnt model should be transferred to other domains. Yang et al. deem a model to be stable if it has been learnt across $2W_{max}$ instances; however, this does not guarantee that the model is able to make good predictions in the local domain. Therefore, BOTL only considers a model to be stable if it has been used to make predictions across $2W_{max}$ instances without a drift being detected. Unstable models are not transferred, preventing them from negatively impacting the target predictor. Defining model stability in this way prevents BOTL from transferring models that have been learnt from short, noisy periods of data, for example, during drifting periods as one concept changes to another. Once a model is considered to be stable, it is transferred to other domains to aid their respective predictors, as shown in Algorithm 1. This means that the models transferred by BOTL are limited to those that have successfully learnt a concept in their local domain.

Algorithm 1 BOTL: transferring models.

1: **Input:** W_{max}, χ^{T}, M.
2: **for** $t = 1, 2, \ldots$ **do**
3: **if** f^{S}_{j+1} available **then**
4: Receive f^{S}_{j+1} from source, add to M
5: Receive $x_t \in \chi^{T}$
6: Update $W = \{x_{t-W_{max}}, \ldots, x_t\}$
7: Learn f_i^{T} and detect drifts using Alg. 3, 5 or 6, add to M
8: $x'_t = \langle f^{S}_{1}(x_t), \ldots, f^{S}_{k}(x_t), f_i^{T}(x_t)\rangle$
9: $\hat{y}_t = F^{M}(x'_t)$ (see Eq. 1)
10: Receive y_t
11: **if** f_i^{T} is a new stable model **then**
12: Send f_i^{T} to all other domains in framework

Knowledge transfer is achieved in BOTL by communicating models across domains. When model f_j^{S} is received from a source domain, it is added to the set of transferred models, M, and combined with the target predictor, f_i^{T}, to enhance the overall predictive performance. Our instantiation of BOTL uses an Ordinary Least Squares (OLS) regressor

as a meta learner to combine the available models such that the squared error of the predicted values, $\hat{y}$, across W is minimised. Other regression meta-learners that are less prone to overfitting on small windows of data, such as Ridge Regression [4], could be used in place of OLS. In this paper, we have chosen to use OLS as the meta-learner as it does not require additional parameters which would have to be determined from domain expertise or parameter tuning prior to learning in each data stream.

Each transferred model, $f_j^S \in M$, and the current target model, f_i^T, are used to generate a new window of data. Each sample x'_t in the newly generated window of data is of the form $x'_t = \{\hat{y}_t'^{S_1}, \ldots, \hat{y}_t'^{S_k}, \hat{y}_t'^{T_i}\}$, where $\hat{y}_t'^{S_j}$ for all $j = 1, \ldots, k$ is the predicted value of source model f_j^S on instance x_t from the original window of target data, and $\hat{y}_t'^{T_i}$ is the predicted value of the locally learnt target model, f_i^T, learnt using the underlying concept drift detection algorithm for the current concept, c_i. This window of model predictions is used by the OLS meta-learner to obtain the overarching predictive function:

$$\begin{aligned}\hat{y}_t &= F^M(x'_t) \\ &= w_0 + \left(\sum_{j=1}^{k} w_j f_j^S(x_t)\right) + w_{(k+1)} f_i^T(x_t). \end{aligned} \tag{1}$$

As the OLS meta-learner is prone to overfitting when the window size is small and the number of base learners is large, the BOTL framework only uses the current target model, f_i^T as input to the meta-learner. Other historical models learnt within the data stream are excluded from the meta-learning process as the underlying concept drift detection strategy deems the current target model, f_i^T, to be the most relevant with respect to the current concept.

4.1 Bi-directional transfer

BOTL considers the scenario where all domains are online, therefore distinctions between source and target can be disregarded. In this paper, BOTL conducts peer-to-peer model transfer, allowing knowledge transfer to enhance the predictive performances of all domains. When a newly learnt model is stable, it is transferred to all other domains in the framework, and each domain updates its model set, M, when a concept drift is encountered.

Real-world applications, such as smart home heating system personalisations, may be comprised of a large number of domains, rapidly increasing the number of models to be transferred as the number of domains grow. Such applications can suffer in predictive performance due to the curse of dimensionality, where the number of input features to the OLS meta learner becomes large in comparison with the window size [8]. To combat this, we introduce culling to BOTL, referred to as BOTL-C.

4.2 Model culling

Culling transferred models from the model set, M, helps prevent the OLS meta-learner overfitting when a large number of models have been transferred and only a small window of data is available. This could be achieved by limiting the maximum number of models used by the meta-learner. However, due to the dynamic nature of the online environment, the maximum number of models that will prevent the meta-learner overfitting cannot be known in advance. Therefore, a conservative estimate would have to be made, requiring additional domain expertise. Using a conservative estimate may prevent beneficial transferred knowledge from being used to aid the target predictor.

Alternatively, transferred models can be evaluated on the current window of data in order to discard transferred models that are considered to be the least beneficial to the target learner. We achieve this by introducing two variants of BOTL-C. Firstly, BOTL-C.I reduces the number of models available to the OLS meta-learner by temporarily removing transferred models from the model set, M, when their R^2 performance across the current window of data drops below a threshold, λc_{perf}. These models can be considered to be the least beneficial to the target learner as they achieve poor predictive performance on the current window of data. Culled models are re-added to M when a concept drift is encountered to enhance predictions of future concepts in the target domain. Although this method of culling is naïve, it can reduce the impact of negative transfer.

Algorithm 2 BOTL-C.II: model culling.

```
Input: W, λc_perf, λc_MI, M, f_i^T
for f_j^S ∈ M do
    if R²(f_j^S, W) ≤ λc_perf then
        Remove f_j^S from M
    for f_k^S ∈ M do
        if MI(f_j^S, f_k^S, W) ≥ λc_MI then
            if R²(f_j^S, W) ≥ R²(f_k^S, W) then
                Remove f_k^S from M
            else
                Remove f_j^S from M
```

In scenarios with high volumes of model transfer, BOTL-C.I requires a high λc_{perf} to sufficiently reduce the number of models to prevent the OLS meta-learner overfitting. This can be detrimental as a high proportion of the transferred models containing useful information are culled and no longer available to enhance the predictive performance of

the target learner. To overcome this, BOTL-C.II, outlined in Algorithm 2, evaluates transferred models based on both performance and diversity, metrics commonly used in ensemble pruning [37]. Initially, BOTL-C.II reduces the impact of negative transfer by culling models that achieve an R^2 performance less than λc_{perf}, on W. A low λc_{perf} value is preferred, ensuring transferred models containing some useful information are retained. Using a low threshold may not sufficiently reduce the model set, M, to prevent overfitting; therefore, a second round of culling is performed based on model diversity. BOTL-C.II measures the diversity between transferred models using Mutual Information (MI). MI allows models that obtain similar predictions on the current window of data to be identified [7]. If two transferred models have a high MI, using both models in the meta-learning process will provide little benefit to the target learner as a high MI indicates the predictions of the two models on the current window of data are highly correlated. Therefore, no additional knowledge is provided to the target learner by keeping both in the model set. If two transferred models have a MI greater than λc_{MI}, BOTL-C.II culls the model that performs worse. This enables redundant models to be removed from the model set, helping to prevent the OLS meta-learner overfitting. A high λc_{MI} should be selected as the window of locally available data is often small; therefore if a complex concept is to be learnt, the target learner may benefit from utilising knowledge transferred from similar concepts. However, if this threshold is too high, the model set, M, will not be reduced sufficiently to prevent overfitting.

Culling thresholds, λc_{perf} and λc_{MI}, could be updated as the data stream progresses using cross-validation, allowing alternative culling parameter values to be compared during the meta-learning process. However, due to the online nature of the data streams, instances within a window cannot be considered independent; therefore, the i.i.d. assumption cannot be made [13], as three consecutive instances in the window, x_{t-1}, x_t, and x_{t+1}, are likely to be dependent. Therefore, any validation set created from the window has some dependence on the training set. This can cause cross-validation to provide an overestimate of the performance of culling parameters. Additionally, using cross-validation for this purpose would require $p * k$ models to be trained and validated every time the meta-learner is updated, where p is the number of culling parameter values compared, and k is the number of folds. As the BOTL framework is to be used in domains with concept drifts, the meta-learner must be updated regularly. Therefore, the use of cross-validation would significantly increase the computation and storage requirements of the BOTL framework, while overestimating the performance of the culling parameters considered. This would limit the use of BOTL in applications that require on-device learning, or have limited computational resources. Within this paper, naïve culling approaches are used where the values of culling parameters are chosen in advance, mitigating the need for cross-validation.

4.3 Initialisation

For any underlying concept drift detection algorithm, an initial window of data, W, is required to create the first predictive model, f_1^T. Prior to obtaining this data, no predictions can be made as no local knowledge has been learnt. BOTL allows models transferred from other domains to be used to make predictions during this period. Models transferred are initially weighted equally to obtain:

$$\hat{y}_t = \frac{1}{|M|} \sum_{j=1}^{|M|} f_j^S(x_t). \tag{2}$$

Before the first target model, f_1^T, has been learnt and only a small amount of data has been observed, the OLS regressor can create a model, F^M, using only source models, f_j^S. This approach is prone to overfitting due to the small amount of data available but may be preferred over making no predictions or using Eq. 2 over the entire initial window of data.

The BOTL-C variants help reduce overfitting within this initial period; however, as the amount of available data is small, all transferred models may have R^2 performances below the culling threshold. In this scenario, both BOTL-C variants select the best k transferred models, where $k < |W|$, regardless of the culling threshold. In this paper, we select $k = 3$.

5 BOTL loss

Theorem 1 *BOTL has a squared loss less than or equal to the model learnt locally using a concept drift detection algorithm with no knowledge transfer:*

$$\mathscr{L}(f_i^T) \geq \mathscr{L}(F^M), \tag{3}$$

where $\mathscr{L}(f_i^T)$ denotes the squared loss of the local model, f_i^T, created using a concept drift detection algorithm, and $\mathscr{L}(F^M)$ is the squared loss of the OLS meta-learner, F^M, created using the set of k models transferred from the source, $\{f_1^S, \ldots, f_k^S\}$ and the current target model, f_i^T.

Proof We measure loss over the local window of data, W, using the mean squared error of predictions:

$$\mathscr{L}(\cdot) = \frac{1}{|W|} \sum_{t=1}^{|W|} \left(y_t - \hat{y}_t\right)^2, \tag{4}$$

where y_t is the response variable for instance x_t, and $\hat{y}_t$ is the predicted value. If no transfer is used, the local model, f_i^T, is used to predict $\hat{y}_t$ for each instance x_t such that $\hat{y}_t = f_i^T(x_t)$.

BOTL uses the set of models, M, to obtain predictions $\hat{y}_t'^{T_i}$ and all $\hat{y}_t'^{S_j}$ for instance x_t, using the locally learnt model, f_i^T, and each of the j transferred model, $f_j^S \in \{f_1^S, \ldots, f_k^S\}$, respectively.

Predictions are used to create a meta-instance, x_t', which the OLS meta-learner, F^M, uses to obtain an overarching prediction:

$$\begin{aligned}\hat{y}_t &= F^M\left(x_t'\right) \\ &= F^M\left(\langle f_1^S(x_t), \ldots, f_k^S(x_t), f_i^T(x_t)\rangle\right) \\ &= F^M\left(\langle \hat{y}_t'^{S_1}, \ldots, \hat{y}_t'^{S_k}, \hat{y}_t'^{T_i}\rangle\right),\end{aligned} \tag{5}$$

where

$$F^M\left(x_t'\right) = w_0 + \sum_{j=1}^{j=k} w_j \hat{y}_t'^{S_j} + w_{(k+1)}\hat{y}_t'^{T_i}. \tag{6}$$

Weights $w_0, \ldots, w_{(k+1)}$ are assigned to each prediction, $\hat{y}_t'^{n}$, for each model n in M, where $|M| = (k+1)$, to obtain an ensemble prediction, $\hat{y}_t$, for instance x_t by solving the optimisation problem that minimises the squared error of F^M:

$$\min_{w_0,\ldots,w_{(k+1)}} \sum_{t=1}^{|W|}\left(y_t - \left(w_0 + \sum_{j=1}^{j=k} w_j \hat{y}_t'^{S_j} + w_{(k+1)}\hat{y}_t'^{T_i}\right)\right)^2. \tag{7}$$

F^M is used to make predictions, $\hat{y}_t$, for instance x_t, using Eq. 6. Using Eq. 4, we can rewrite the loss of F^M as:

$$\mathscr{L}(F^M) = \frac{1}{|W|}\sum_{t=1}^{|W|}\left(y_t - \left(w_0 + \sum_{j=1}^{j=k} w_j \hat{y}_t'^{S_j} + w_{(k+1)}\hat{y}_t'^{T_i}\right)\right)^2. \tag{8}$$

If we constrain the optimisation problem in Eq. 7 to obtain the meta-learner F^{M^*} by fixing the weights, w_a, such that the weight associated with the locally learnt model f_i^T is 1, while all others are 0, we obtain a meta-model of the form:

$$F^{M^*}\left(x_t'\right) = \left(0 + \sum_{j=1}^{j=k} 0\hat{y}_t'^{S_j} + 1\hat{y}_t'^{T_i}\right), \tag{9}$$

giving the loss function:

$$\mathscr{L}(F^{M^*}) = \frac{1}{|W|}\sum_{t=1}^{|W|}\left(y_t - \hat{y}_t'^{T}\right)^2, \tag{10}$$

equivalent to only using the locally learnt model, $\mathscr{L}(F^{M^*}) = \mathscr{L}(f_i^T)$. As the optimisation problem in Eq. 7 is convex:

$$\mathscr{L}(F^{M^*}) \geq \mathscr{L}(F^M). \tag{11}$$

Finally, as the constrained optimisation problem in Eq. 9 is equivalent to using only the locally learnt model, f_i^T, the loss of BOTL is less than or equal to the loss of the locally learnt model. □

6 Experimental set-up

Many benchmark datasets have been created to evaluate concept drift detection algorithms [16, 27, 28]; however, most are categorically labelled. In order to evaluate BOTL in a regression setting, we present a modification to the benchmark drifting hyperplane dataset [20]. Additionally, a simulation of a smart home heating system was created using data from a UK weather station to derive desired heating temperatures for a user. The use of such data enables BOTL to be evaluated on data streams containing drifts that are typical within real-world environments. Finally, we evaluate the performance of BOTL using a following distance dataset,[2] created from vehicular data, and used to predict the Time To Collision (TTC). An overview of the dataset characteristics is shown in Table 2.

6.1 Drifting hyperplane

For this benchmark data generator, an instance at time t, x_t, is a vector, $x_t = \{x_{t_1}, x_{t_2}, \ldots, x_{t_n}\}$, containing n randomly generated, uniformly distributed, variables, $x_{t_n} \in [0, 1]$. For each instance, x_t, a response variable, $y_t \in [0, 1]$, is created using the function $y_t = (x_{t_p} + x_{t_q} + x_{t_r})/3$, where p, q, and r reference three of the n variables of instance x_t. This function represents the underlying concept, c_a to be learnt and predicted. Concept drifts are introduced by modifying which features are used to create y. For example, an alternative concept, c_b, may be represented by function $y_t = (x_{t_u} + x_{t_v} + x_{t_w})/3$, where $\{p, q, r\} \neq \{u, v, w\}$ such that $c_a \neq c_b$. We introduce uniform noise, ± 0.05, by modifying y_t for each instance x_t with probability 0.2.

A variety of drift types have been synthesised in this generator including sudden drift, gradual drift and recurring drifts. A sudden drift from concept c_a to concept c_b is encountered immediately between time steps t and $t+1$ by changing the underlying function used to create y_t and y_{t+1}. A gradual drift from concept c_a to c_b occurs between time steps t and $t+m$, where m instances of data are observed during the drift. Instances of data created between t and $t+m$ use one of the underlying concept functions to determine their response variable. The probability of an instance belonging to concept c_a decreases proportionally to the number of instances seen after time t while the

[2] Data generators, sample vehicle data, and reproducibility documentation are available at: https://github.com/hmckay/BOTL

Table 2 Dataset characteristics

Dataset	Characteristic	Dataset type	Drift type Sudden/gradual	# streams	Avg. #x_t per stream	Drifts per stream
SuddenA	Uniform noise	Synthetic	✓/–	6	10,000	20
SuddenB	Sensor failure	Synthetic	✓/–	6	10,000	20
SuddenC	Intermittent sensor failure	Synthetic	✓/–	6	10,000	20
SuddenD	Sensor deterioration	Synthetic	✓/✓	6	10,000	20
GradualA	Uniform noise	Synthetic	–/✓	6	11,900	20
GradualB	Sensor failure	Synthetic	✓/✓	6	11,900	20
GradualC	Intermittent sensor failure	Synthetic	✓/✓	6	11,900	20
GradualD	Sensor deterioration	Synthetic	–/✓	6	11,900	20
Heating	Weather data	Hybrid	✓/✓	5	17,664	NA
Following	Vehicular data	Real-world	✓/✓	17	1909	NA

probability of it belonging to c_b increases as we approach $t+m$. Recurring drifts are created by introducing a concept c_c that reuses the underlying function defined by a previous concept, c_a, such that we achieve conceptual equivalence, $c_c = c_a$. Datasets generated in this way, containing uniform noise, are denoted by SuddenA and GradualA for sudden and gradual drifting data streams respectively.

We also create variations of the drifting hyperplane datasets that introduce problems that may be encountered when using BOTL in real-world environments. The first variation simulates sensor failure. In this scenario, a feature vector, i, is set to 0 from time t for the remainder of the data stream with probability 0.001, such that $x_{t_i} = 0$. In the scenario where feature i is used to create the response variable y, we modify two other feature vectors, j and k, such that $x_{t_j} = x_{t_i}/4$ and $x_{t_k} = 3x_{t_i}/4$. This ensures that the underlying concept can still be learnt from the data. We denote datasets generated in this way as SuddenB and GradualB for sudden and gradual drifting data streams.

The second variation simulates intermittent sensor failure, where once the feature vector i has been selected to fail, the feature value at time step t is set to 0 such that $x_{t_i} = 0$ with probability 0.3. Datasets generated using this scenario are denoted as SuddenC and GradualC for sudden and gradual drifting data streams respectively.

The third variation introduced emulates the deterioration of a sensor. Sensor deterioration is captured by including noise depending on the time step t such that $x_{t_i} = x_{t_i} \pm (0.2(t/|\chi|))$, where 0.2 is the maximum amount of noise added to instance x_{t_i} and $|\chi|$ is the number of instances in the dataset. This means that as the data stream progresses, more noise is added to an individual feature, simulating the gradual deterioration in accuracy of a sensor over time. Additionally, the probability of a sensor deteriorating increases as the data stream progresses, such that the probability of a feature being selected for deterioration at time t is $0.001(t/|\chi|)$. Datasets generated in this way are denoted as SuddenD and GradualD for sudden and gradual drifting data streams respectively.

6.2 Heating simulation

A simulation of a smart home heating system was created, deriving the desired room temperature of a user. Heating temperatures were derived using weather data collected from a weather station in Birmingham, UK, from 2014 to 2016. This dataset contained rainfall, temperature and sunrise patterns, which were combined with a schedule, obtained from sampling an individual's pattern of life, to determine when the heating system should be engaged. The schedule was synthesised to vary the desired heating temperature based on time of day, day of week and external weather conditions, creating complex concepts. To create multiple domains, weather data was sampled from overlapping time periods and used as input to the synthesised schedule to determine the desired heating temperatures. Due to the dependencies on weather data, each stream was subject to large amounts of noise. Concept drifts were introduced manually by changing the schedule; however, drifts also occurred naturally due to changing weather conditions. By sampling weather data from overlapping time periods, and due to seasonality, data streams follow similar trends, ensuring predictive performance can benefit from knowledge transfer. By using complex concepts, dependent on noisy data, the evaluation of BOTL on this data is more indicative of what is achievable when used in real-world environments.

6.3 Following distance

This dataset uses a vehicle's following distance and speed to calculate TTC when following another vehicle. Vehicle

telemetry data such as speed, gear position, brake pressure, throttle position and indicator status, alongside sensory data that infer external conditions, such as temperature, headlight status and windscreen wiper status, were recorded at a sample rate of 1 Hz. Additionally, some signals such as vehicle speed, brake pressure and throttle position were averaged over a window of 5 seconds to capture a recent history of vehicle state. Vehicle telemetry and environmental data can be used to predict TTC and used to personalise vehicle functionalities such as ACC by identifying the preferred following distance, reflecting current driving conditions. Data was collected from 4 drivers for 17 journeys which varied in duration, collection time and route. Each journey is considered to be an independent domain and BOTL enables knowledge to be learnt and transferred across journeys and between drivers. Each data stream is subject to concept drifts that occur naturally due to changes in the surrounding environment such as road types and traffic conditions.

7 Drift detection strategies

BOTL relies on a concept drift detection algorithm, local to each domain. Although any sliding window-based drift detection strategy can be used, it is desirable for the chosen strategy to learn as few models as possible to represent concepts in the data stream. Limiting the number of models learnt in each domain reduces the number of input features to the OLS meta-learner, helping to prevent overfitting caused by the curse of dimensionality [8]. Additionally, reducing the number of models needing to be transferred across domains reduces the communication and computational overhead of combining knowledge, which may impact the feasibility of using BOTL in real-world applications that require on-device learning.

Drift detection strategies that employ single model-based approaches, instead of ensemble techniques, reduce the number of models used to represent a single concept. If an ensemble-based drift detector was used, such as DWM [19], or Adaptive Windowing Online Ensemble (AWOE) [29], the knowledge learnt to represent a single concept may be encompassed across multiple models in the ensemble. Therefore, all models, and their ensemble weights, would need to be transferred across domains. Additionally, the number of models learnt in each domain can be reduced by allowing the reuse of previously learnt models when concepts reoccur [33]. To achieve this, a history of models can be retained to prevent redundant models being learnt locally, and transferred across domains.

We consider three concept drift detection techniques to underpin BOTL. Firstly, we adapt RePro [34] to a regression setting; secondly we apply ADWIN [3]; and thirdly we propose a new concept drift detection algorithm, Adaptive Windowing with Proactive drift detection (AWPro) which combines elements of RePro and ADWIN. Each of these drift detection strategies are dependent on user-defined parameters, which require domain expertise to select appropriate values. Within this section, we discuss the impact of user-defined parameters on each concept drift detection strategy, and how this effects their applicability as the drift detection algorithm within domains for BOTL. To highlight this, we evaluate RePro, ADWIN and AWPro on synthetic drifting hyperplane datasets, generated with uniform noise, containing sudden and gradual drifts, simulated smart home heating data, and real-world vehicle following distance data.

7.1 RePro

We consider an adaptation of RePro [34] for regression as an underlying drift detector. Although RePro requires domain expertise to select appropriate parameter values, including window size, W_{max}, drift threshold, λ_d, and loss threshold, λ_l, it encapsulates key characteristics that allow few models to be learnt in each domain. RePro is a sliding window-based detection algorithm that learns a single model for the current concept [33]. Additionally, RePro prioritises the reuse of existing models over learning new models by retaining a history of previously learnt models, H^T, and concept transitions, TM^T, to proactively determine which concept is likely to occur next [34].

RePro was initially developed specifically for classification tasks; therefore, modifications are required for regression settings, shown in Algorithm 3. The original RePro algorithm detects drifts by measuring the target models' classification accuracy across the sliding window, W. When the classification accuracy drops below an error threshold a drift is detected [34]. If the window is full, $|W| = W_{max}$, but the classification accuracy does not drop below the error threshold, the sliding window is maintained by discarding one incorrectly classified instance, and all subsequent correctly classified instances. To apply RePro to regression, the sliding window must be maintained; however, the notion of a correctly classified instance must be altered as small inaccuracies are inevitable in regression settings due to noise. To overcome this, ε-insensitivity can be used, allowing for a small margin of error between the prediction and response variable. To maintain a sliding window (lines 11–14), we introduce a loss threshold, λ_l that allows instance $x_{(t-|W|)}$ to be discarded from the window if the predicted value, $\hat{y}'_{(t-|W|)}$ satisfies:

$$|\hat{y}'_{(t-|W|)} - y_{(t-|W|)}| \leq \lambda_l.$$

The R^2 performance of the target model, f_i^T, across W is used to detect drifts (line 8). A drift is said to have occurred when the performance of the target model drops below a predefined drift threshold, λ_d, akin to observing the classification accuracy dropping below an error threshold.

Algorithm 3 Adapted RePro for regression.

1: **Input:** W_{max}, λ_l, λ_d, χ^T, $H^T = \emptyset$ (historical concepts), $TM^T = \emptyset$ (transition matrix).
2: Learn f_1^T using $x_1 \ldots x_{W_{max}}$, add to H^T
3: **for** $t = W_{max} + 1, W_{max} + 2, \ldots$ **do**
4: Receive x_t and predict $\hat{y}'_t = f_i^T(x_t)$
5: Receive y_t, add $\langle x_t, \hat{y}'_t, y_t \rangle$ to W
6: **if** f_i^T is new and stable **then**
7: Add f_i^T to H^T and $f_{i-1}^T \rightarrow f_i^T$ to TM^T
8: **if** $R^2(f_i^T, W) < \lambda_d$ **then**
9: f_{i+1}^T = getNextModel(H^T, TM^T, W, λ_d) using Alg. 4
10: $W = \{\}$
11: **else if** $|W| \geq W_{max}$ **then**
12: Remove $x_{(t-|W|)}$ from W
13: **while** $|f_i^T(x_{(t-|W|)}) - y_{(t-|W|)}| \leq \lambda_l$ **do**
14: Remove $x_{(t-|W|)}$ from W

Algorithm 4 Model selection and creation for RePro. For additional details see [34].

1: **Function:** getNextModel
2: **Input:** H^T, TM^T, W, λ_d
3: f_{i+1}^T = next model in TM^T
4: **if** $R^2(f_{i+1}^T, W) \geq \lambda_d$ **then**
5: **return** f_{i+1}^T
6: f_{i+1}^T = best performing model in H^T
7: **if** $R^2(f_{i+1}^T, W) \geq \lambda_d$ **then**
8: **return** f_{i+1}^T
9: Learn f_{i+1}^T using W
10: **return** f_{i+1}^T

The original formulation of RePro used the notion of a stable learning size, specifying how much data is required to learn a stable model. This was necessary for the simulated classification tasks presented by Yang et al. [34] as small window sizes were required to allow drifts to be detected quickly. However, this meant that insufficient instances were available in the window to learn a model that adequately represented the current concept [33]. Yang et al. suggest a stable learning size of $3W_{max}$ [34]. As real-world environments are often considerably more noisy than simulated or synthetic environments, using a small window size can cause drifts to be falsely detected; therefore, a larger window size is necessary [3]. Increasing the window size also increases the stable learning size; however, if the stable learning size is increased, the data used to create a model may encapsulate multiple underlying concepts. To overcome this challenge, our adaptation of RePro for regression defines a stable model to be one that is learnt from W_{max} instances and is used to make predictions over $2W_{max}$ instances without a drift being detected.

To proactively determine future concepts, RePro maintains a transition matrix, TM^T, to determine the likelihood of encountering a recurring concept. To prevent the reuse of unstable models that make poor predictions, only those that are considered to be stable are added to the transition matrix. If the transition matrix indicates that it is equally likely that two or more concepts may be encountered next, RePro evaluates the performance of each model on the current window of data and selects the model with the highest accuracy. If the transition matrix does not indicate a likely successor concept, each historical model is considered for reuse. A new model is only learnt when all historical models perform worse than the drift threshold λ_d, as shown in Algorithm 4.

7.1.1 Parameter selection

The characteristics of RePro as a drift detection strategy are desirable for BOTL; however, the selection of parameter values, W_{max}, λ_d and λ_l, may not be intuitive. Given an online data stream, trade-offs must be considered for each parameter. For example, selecting a large window size, W_{max}, allows more data to be retained to build local target models, increasing their accuracy and stability [2]. However, a window size that is too large may cause RePro to react slowly to concept drifts, and may retain data from multiple concepts, preventing a model from being created to represent each concept independently. Alternatively, using a small window size may allow RePro to react quickly to drifts as a smaller window size may encapsulate a sample of data that is more representative of the current distribution of the data stream. However, selecting a window size that is too small may prevent a representative sample being retained to build a model that effectively represents the current concept, reducing the overall performance of RePro.

The drift and loss thresholds, λ_d and λ_l, determine RePro's sensitivity to concept drift. Small drift thresholds and large loss thresholds decrease RePro's sensitivity to concept drifts as small λ_d values allow the performance of a model to greatly decrease before a drift is detected, while large λ_l values allow instances to be removed from the sliding window while a model's predictive error is high.

Large λ_d and small λ_l values increase RePro's sensitivity to drifts. As RePro becomes more sensitive to concept drifts,

it also becomes more likely that the window of data, W, used to build a model for the newly encountered concept contains instances belonging to the previous concept. Models built using data belonging to both the previous, and new, concepts exhibit high predictive errors. As RePro monitors the model performance to detect drifts, this can cause RePro to repeatedly detect drifts and create unstable models immediately after a concept drift, and during periods of gradual drift. The repeated creation of unstable models increases computation; however, these models are not added to the transition matrix, TM^T, or the model history, H^T, and therefore do not greatly impact the overarching performance of RePro across the data stream and do not impact the communicational overhead of knowledge transfer. Due to this, selecting values for λ_d that are too large, and values for λ_l that are too small, may prevent RePro from creating stable models that can be added to the model history where drifts are falsely detected in the presence of noise.

7.2 ADWIN

ADWIN, presented by Bifet et al. [3], detects drifts by monitoring changes in the distribution of a data stream. For use in a regression setting, the distribution of predictive error is monitored across a sliding window. Instead of using a fixed length sliding window, the size of the window is determined according to the rate of change observed in the online data stream [3].

Algorithm 5 ADWIN. For full implementation details see [3].

1: **Input:** W_{min}, χ^T, δ (confidence value), driftFlag = False.
2: Learn f_1^T using $x_1 \ldots x_{W_{min}}$
3: adwin = new ADWIN(δ)
4: **for** each $t = W_{min} + 1, W_{min} + 2, \ldots$ **do**
5: Receive x_t and predict $\hat{y}'_t = f_i^T(x_t)$
6: Receive y_t, and calculate diff=$(\hat{y}'_t - y_t)$
7: Add $\langle x_t, \hat{y}'_t, y_t, \text{diff}\rangle$ to W
8: **if not** driftFlag **then**
9: driftFlag, $W_0, W_1 =$ adwin.detectDrift(W) (using Eqs. 12 & 13)
10: **if** driftFlag **then**
11: $W = W_1$
12: **if** driftFlag and $|W| \geq W_{min}$ **then**
13: Learn f_{i+1}^T using $W = \{x_{t-W_{min}}, \ldots, x_t\}$
14: driftFlag = False
15: adwin = new ADWIN(δ)

ADWIN operates on the principal that if two large enough sub-windows have distinct enough means, the expected values within each sub-window will differ [3, 9]. A drift is said to be detected when:

$$|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| \geq \varepsilon_{\text{cut}}, \tag{12}$$

where $\hat{\mu}_{W_0}$ and $\hat{\mu}_{W_1}$ are the means of sub-windows W_0 and W_1, and ε_{cut} is defined by the Hoeffding bound:

$$\varepsilon_{\text{cut}} = \sqrt{\frac{1}{2m} \cdot \ln \frac{4|W|}{\delta}}, \tag{13}$$

where m is the harmonic mean of the sub-windows, $m = \frac{2}{1/|W_0| + 1/|W_1|}$, and δ is a confidence value, defined by the user, which determines the sensitivity of drift detection [3, 9].

ADWIN can be used by BOTL, as presented in Algorithm 5, to detect drifts within the data stream by monitoring the distribution of the predictive error of the locally learnt model, f_i^T:

$$|f_i^T(x_t) - y_t|.$$

Once a drift is detected, a new model is learnt locally, f_{i+1}^T, using the second sub-window such that $W = W_1$ (lines 9–15). Monitoring the distribution of predictive error allows drifts to be detected rapidly. However, if two consecutive concepts are dissimilar, drifts are frequently detected when only a small number of instances from the new concept have been observed; therefore, the data contained in the sliding window, W, after a drift is detected is unlikely to be representative of the new underlying concept. To address this, our implementation of ADWIN for BOTL only creates a new model, f_{i+1}^T, once W_{min} instances belonging to the new concept have been observed, such that $|W| = W_{min}$ (lines 12–15). Until sufficient data has been observed to build a model that adequately represent the new concept, the previously learnt model, f_i^T, must continue to be used to make predictions.

7.2.1 Parameter selection

Using ADWIN in this way requires two user-defined parameters, the minimum window size, W_{min}, and the confidence value, δ. Similarly to RePro, domain expertise is required to select these parameter values.

As ADWIN uses a dynamic sliding window, the minimum window size parameter, W_{min}, does not directly impact ADWIN's ability to detect concept drifts. Instead W_{min} is only used to determine how much data should be retained to build a model that adequately represents the current concept [2]. Similar to RePro's W_{max} parameter, large values of W_{min} allow more data to be made available to the local target learner, f_i^T, creating an accurate and stable model [2]. However, if W_{min} is too large, the data contained within the window may encapsulate data from multiple concepts, preventing individual models being learnt for each

concept. Small values of W_{min} ensure the data used to build a model is representative of the current distribution of data.

ADWIN detects drifts by monitoring the distribution of predictive error; therefore, W_{min} can indirectly effect ADWIN's ability to correctly detect concept drifts and overarching performance. If W_{min} is too small, it may cause a model to be learnt that overfits and has high predictive error. As ADWIN monitors the change in distribution of predictive error, using a model that initially has a high predictive error may prevent or delay the detection of a concept drift as no significant change in the distribution of predictive error is observed during periods of drift. However, large W_{min} values increase the number of instances observed before a new model can be learnt, prolonging the use of the previously learnt model, decreasing the overarching predictive performance of ADWIN across the data stream.

The confidence value, δ, is used to determine ADWIN's sensitivity to concept drifts through the ε_{cut} threshold (Eqs. 12 and 13). High values of δ increases drift sensitivity; however, in noisy data streams, this can cause drifts to be falsely detected due to the increased variability of sub-window means, $\hat{\mu}_{W_0}$ and $\hat{\mu}_{W_1}$. To overcome this, lower values of δ can be chosen; however, this may prevent drifts from being detected in domains containing similar consecutive concepts, or slow gradual drifts, where the sub-window means do not change greatly.

7.3 AWPro

The use of RePro as a concept drift detection algorithm can be computationally demanding due to the creation of high volumes of unstable models, caused by its inability to detect the precise point of drift within the sliding window. ADWIN allows this point to be identified by splitting the sliding window into two sub-windows where the first sub-window contains instances belonging to the old concept, which can be discarded, while the second sub-window contains instances belonging to the new concept. However, if the number of remaining instances in the second sub-window is small, ADWIN must wait until sufficient instances have been observed before a new model can be learnt, negatively impacting the performance of ADWIN. Additionally, ADWIN does not reuse previously learnt models; therefore, models must be re-learnt for recurring concepts. This increases the number of models transferred between domains when ADWIN is used as the underlying concept drift detection algorithm for BOTL, and prevents previously learnt models from being used to make predictions when few instances from a recurring concept have been observed.

Algorithm 6 AWPro.

1: **Input:** W_{min}, χ^T, δ (confidence value), driftFlag = False, λ_r, $H^T = \emptyset$ (historical concepts), $TM^T = \emptyset$ (transition matrix).
2: driftFlag, tempModel = False
3: Learn f_1^T using $x_1 \ldots x_{W_{min}}$, add to H^T
4: adwin = new ADWIN(δ)
5: **for** each $t = W_{min} + 1, W_{min} + 2, \ldots$ **do**
6: Receive x_t and predict $\hat{y}'_t = f_i^T(x_t)$
7: Receive y_t, and calculate diff=$(\hat{y}'_t - y_t)$
8: Add $\langle x_t, \hat{y}'_t, y_t, \text{diff} \rangle$ to W
9: **if not** driftFlag **then**
10: driftFlag, $_{W_0, W_1}$= adwin.detectDrift(W) (using Eqs. 12 & 13)
11: **if** driftFlag **then**
12: $W = W_1$
13: **if** driftFlag and $|W| < \frac{1}{2} W_{min}$ and **not** tempModel **then**
14: tempModel = True
15: Learn f_{temp}^T using W
16: **else if** driftFlag and $|W| \geq \frac{1}{2} W_{min}$ **then**
17: **if** $R^2(\text{getNextModel}(H^T, TM^T, W, \lambda_r, W_{min}), W) \geq \lambda_r$ **then**
18: f_{i+1}^T = getNextModel($H^T, TM^T, W, \lambda_r, W_{min}$) using Alg. 7
19: driftFlag,tempModel = False
20: adwin = new ADWIN(δ)
21: **else if** driftFlag and $|W| \geq W_{min}$ **then**
22: f_{i+1}^T = getNextModel($H^T, TM^T, W, \lambda_r, W_{min}$) using Alg. 7
23: driftFlag,tempModel = False
24: adwin = new ADWIN(δ)
25: **if** f_i^T is new and stable **then**
26: Add f_i^T to H^T and $f_{i-1}^T \rightarrow f_i^T$ to TM^T

To reduce computation from creating unstable models, while also preventing duplicate models being learnt for recurring concepts, we introduce an alternative concept drift detection strategy, AWPro, presented in Algorithm 6, which combines desirable characteristics from ADWIN and RePro that better suit the BOTL framework.

AWPro uses ADWIN to monitor the change in distribution of predictive error, allowing the drift detection strategy to partition instances belonging to different concepts within a dynamic sliding window. Concept drifts are identified using Eqs. 12 and 13, which use a confidence value, δ, to determine the sensitivity to changes in the distribution of the predictive error (line 10). Once a model is learnt, RePro is used to identify stable models, which are retained in the model history, H^T, and the transition between concepts is added to the transition matrix, TM^T (lines 25–26). A stable model is one that is used to make predictions over $2W_{min}$ instances without a drift being detected.

When a concept drift is encountered, AWPro drops the first sub-window of instances, W_0, such that all instances in the window belong to the new concept, $W = W_1$. If the remaining data in the window is less than $\frac{1}{2} W_{min}$ instances, a temporary model is created (lines 13–15). Although

these temporary models are akin to unstable models learnt using RePro, the window used to build these models only contains instances belonging to the new concept. Having few instances available increases the likelihood of learning a model that is not representative of the entire concept; however, this may be preferable to ADWIN's approach of continuing to use a model that represents the previous concept, or RePro's approach where a model may be learnt from data belonging to both concepts. Once a temporary model has been learnt, incoming instances continue to be added to the window.

Algorithm 7 Model selection and creation for AWPro.

1: **Function:** getNextModel
2: **Input:** H^T, TM^T, W, λ_r, W_{min}
3: f^T_{i+1} = next model in TM^T
4: **if** $R^2(f^T_{i+1}, W) \geq \lambda_r$ **then**
5: **return** f^T_{i+1}
6: f^T_{i+1} = best performing model in H^T
7: **if** $R^2(f^T_{i+1}, W) \geq \lambda_r$ **then**
8: **return** f^T_{i+1}
9: **if** $|W| \geq W_{min}$ **then**
10: Learn f^T_{i+1} using $W = \{x_{t-W_{min}}, \ldots, x_t\}$
11: **return** f^T_{i+1}

If $\frac{1}{2}W_{min}$ or more instances have been observed after a drift, the proactive nature of RePro is used by AWPro to determine if an existing model can be reused to represent the current concept (lines 16–20). This allows AWPro to identify an existing model, using Algorithm 7, that has already been learnt to be used for predictions prior to a full window of instances being observed. AWPro uses a recurrence threshold that determines if an existing model can be reused, λ_r, which acts in that same way as the drift threshold, λ_d, defined by RePro, when considering the reuse of existing models. If a model's R^2 performance is greater than the recurrence threshold, λ_r, it is reused, and the process of detecting concept drifts through monitoring changes to the distribution of predictive error is resumed. However, if no model exists, the use of the temporary model continues until W_{min} instances of the new concept have been observed.

Finally, if W_{min} instances have been observed after a concept drift, and a temporary model is still being used to make predictions, AWPro uses the transition matrix and historical models to identify existing models that could be reused now a more representative sample of data is contained within the window. If no existing model exceeds the recurrence threshold, λ_r, a new model is learnt using the W_{min} most recently observed instances.

7.3.1 Parameter selection

AWPro relies on three user-defined parameters: the confidence value, δ, the window size, W_{min}, and the recurrence threshold, λ_r. As AWPro adopts ADWIN's approach to detecting concept drifts, many of the challenges of parameter selection specified with respect to ADWIN for the confidence value, δ, and the window size, W_{min}, are also applicable to AWPro. However, instead of waiting for W_{min} instances to be observed after a drift is detected, AWPro uses the recurrence threshold, λ_r, to determine if an existing model can be reused.

Parameter λ_r effects AWPro in two ways. If large values are chosen for λ_r, the likelihood of reusing a historical model decreases, as a historical model must exhibit low predictive error in order to be selected for reuse. Therefore, high λ_r values will increase the number of models learnt by AWPro. To increase the reuse of models in the presence of recurring concepts smaller values should be chosen for λ_r. However, if λ_r is too small, an existing model may be reused for a concept that has not previously been encountered, lowering the overarching predictive performance of AWPro. This may hinder the detection of concept drifts as the predictive error across the sliding window of data may initially be high, therefore identifying concept drifts through monitoring changes to the distribution of predictive error becomes challenging.

7.4 Impact of parameter values

In order to investigate how BOTL is impacted by parameters defined by each drift detection strategy, we consider the performance of the underlying drift detectors in addition to the number of both stable and unstable models created.[3] Parameter values should be chosen with the aim of maximising the performance of the underlying drift detector, while reducing both the number of stable and unstable models, therefore minimising unnecessary computation, and reducing communication overheads.

The first parameter considered was the loss threshold, λ_l, used by RePro, which determines how close a prediction must be to the response variable in order for it to be discarded from the sliding window. As the hyperplane datasets are synthetic, and response variables are in the range [0,1], we used $\lambda_l = 0.01$, allowing for a 1% error in predictions. The heating simulation and following distance datasets do not have a definitive range for their respective response variables; therefore, a percentage of error could not be used. As these are examples of BOTL being used

[3] Further analysis of the impact of user defined parameters is available at: https://github.com/hmckay/BOTL

by a user facing application, λ_l was selected by considering errors that would not be noticeable to a user. For the heating simulation datasets, we used $\lambda_l = 0.5$ as a prediction error of 0.5 °C would not be detectable by an individual. Similarly, we used $\lambda_l = 0.1$ for the following distance datasets, allowing for a predictive error of 0.1 s. These loss thresholds are used by RePro throughout the remainder of this paper.

Drift sensitivities and window sizes must also be chosen for RePro, ADWIN and AWPro. To consider how predictive performance and the number of models created are effected by parameter values, we varied both drift sensitivity and window size for each drift detection strategy.

Figure 1 displays results of varying drift sensitivities for each drift detection strategy. These results used a fixed window size for each dataset type, representative of the results presented in Fig. 2, which were obtained by varying window size. Figure 1 uses a window size of 30 instances for hyperplane datasets, 480 instances for heating simulation datasets and 90 instances for following distance datasets.

Figure 1 indicates higher drift sensitivity values typically obtained a higher performance across all drift detectors; however, the number of unstable models was also larger. Lowering the drift sensitivity introduced a slight decrease in performance but significantly reduced the number of unstable models, particularly in the case of RePro, shown in Fig. 1a; therefore, a trade-off between performance and the number of models created is necessary.

If we are concerned solely with the performance of the underlying drift detector, then RePro obtained the best performance overall. RePro, shown in Fig. 1a, outperformed ADWIN, Fig. 1b, and AWPro, Fig. 1c, due to its drift detection mechanism. Unlike ADWIN and AWPro, RePro monitors the predictive performance of the current model, and detects drifts when its performance drops below the drift threshold, λ_d. This means poorly performing models are replaced as new instances of data are observed, until a model that achieves a performance greater than λ_d is learnt. ADWIN and AWPro only monitor the distribution of predictive error, regardless of how poorly the model performs; therefore, a poorly performing model will only be replaced when a change in the distribution of predictive error is observed.

Although RePro uses a sliding window to capture the most recent instances, it does not detect the precise point a drift occurs within the window. This means that RePro frequently builds models from windows containing instances belonging to both the previous, and new concept, causing unstable models to be learnt. As RePro monitors the performance of each model, unstable models are often created in quick succession during gradual drifts, or immediately after sudden drifts. This is highlighted in the annotations in Fig. 1, which show the percentage of models that were considered stable and useful for knowledge transfer. RePro created significantly more unstable models, wasting computation that may mean it is infeasible in environments with limited computational resources. ADWIN and AWPro may be more applicable in these environments as they allow the precise point of drift to be identified using a dynamic sliding window, therefore reducing the number of unstable models.

Although ADWIN created fewer unstable models, it cannot reuse existing models in the presence of recurring concepts; therefore, a larger number of stable models were learnt. This is detrimental to BOTL as it increases the communication required for knowledge transfer. Additionally, BOTL relies upon a meta-learner to combine the knowledge transferred across domains. Transferring redundant, or duplicate models, across domains can negatively impact the overarching performance of BOTL as it increases the number of input features to the meta-learner, increasing the likelihood of overfitting caused by the curse of dimensionality [8].

AWPro combines ADWIN's drift detection strategy with RePro's ability to prioritise the reuse of existing models. This combats communication overheads and reduces the risk of overfitting introduced by transferring redundant models. Although the percentage of models that were considered stable was higher on average for ADWIN, AWPro created both fewer stable and unstable models due to the prioritisation of reusing existing models when drifts are encountered. This makes AWPro more applicable to environments that require minimised computation and communication overheads.

By considering performance and the number of stable and unstable models learnt, we selected a drift sensitivity value $\lambda_d = 0.5$ for RePro as the performance obtained across all datasets remained high, and the percentage of models learnt that are considered stable increased drastically compared with using drift sensitivity values $\lambda_d = 0.6$ and $\lambda_d = 0.7$.

The percentage of models learnt by ADWIN that are considered stable varied little across drift sensitivity values within the hyperplane datasets; however, a drop in performance was observed when $\delta > 0.02$ and $\delta > 0.002$ for sudden and gradual drifting datasets respectively. Additionally, an increase in the percent of models that were considered stable was observed in the heating simulation datasets at $\delta = 0.02$, while no significant change in performance or number of stable or unstable models was observable in the following distance datasets. Therefore, a drift sensitivity value $\delta = 0.02$ was selected for ADWIN.

As AWPro is based upon the drift detection mechanism used by ADWIN, observations in changes to performance with varying drift sensitivities were similar. To enable fair comparisons, we also used $\delta = 0.02$ for AWPro.

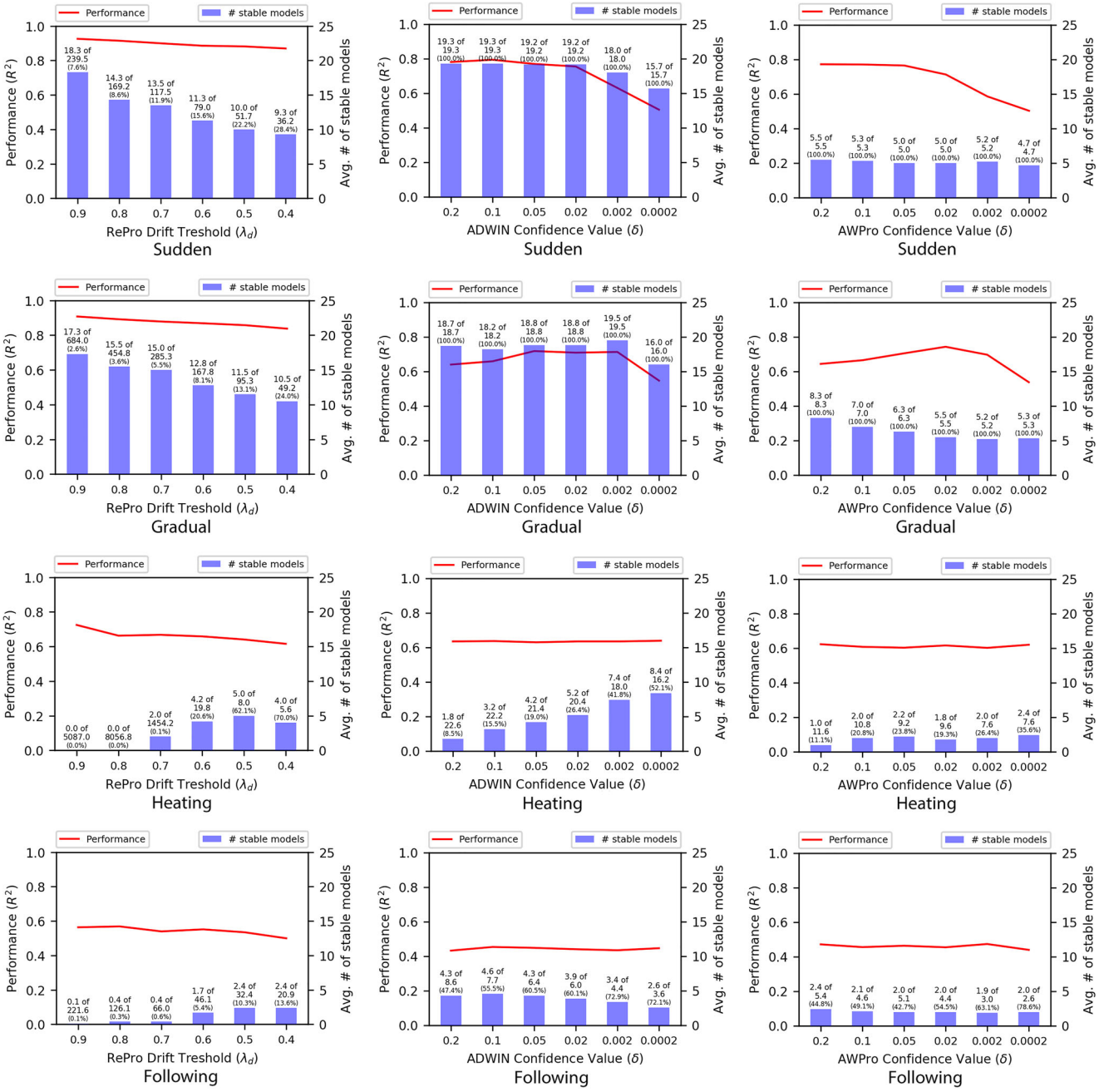
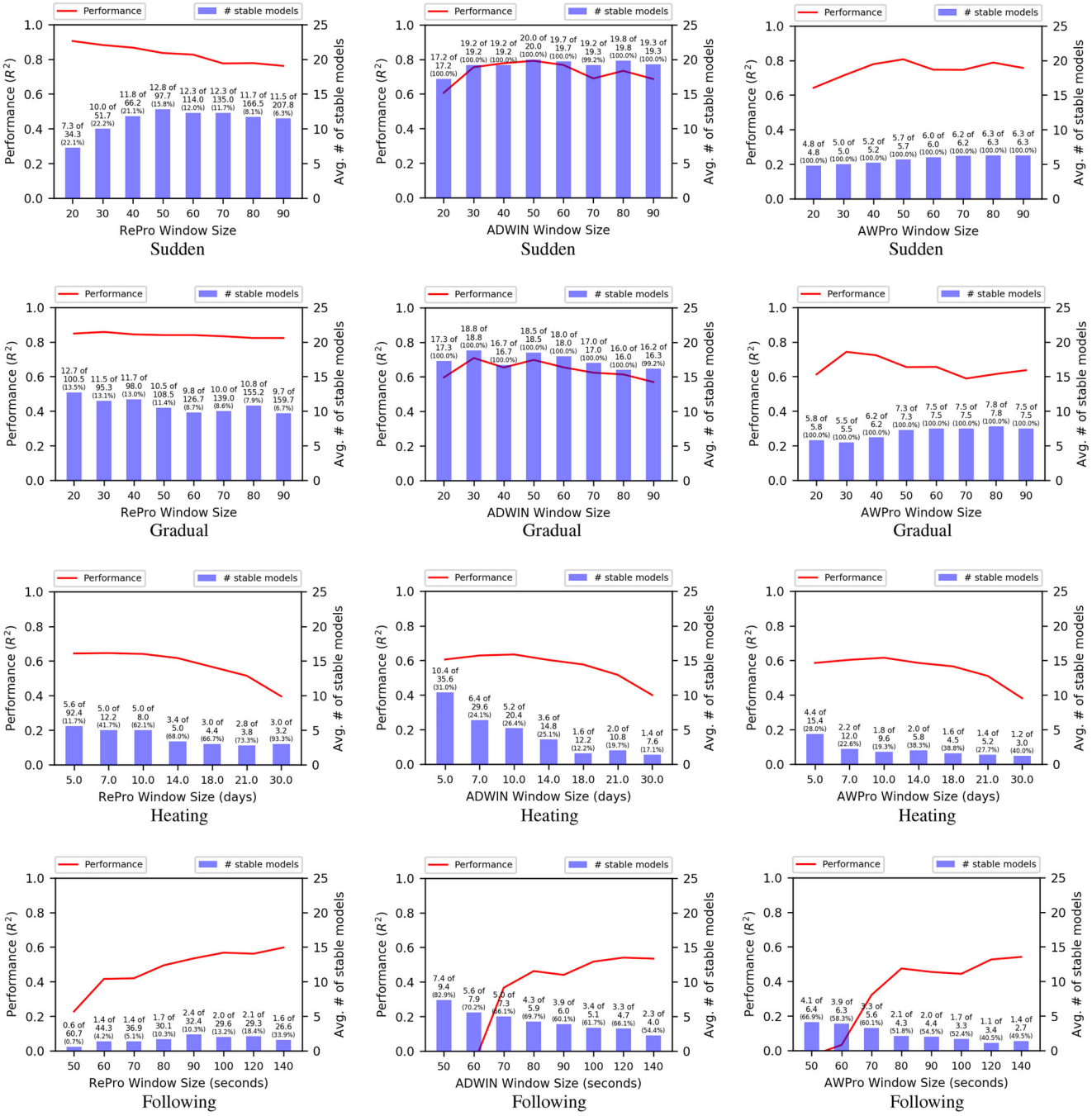


Fig. 1 Performance and number of stable models created by RePro, ADWIN and AWPro with varying drift sensitivities (drift threshold, λ_d, confidence value, δ, and confidence value, δ, respectively). Annotated with the total number of models learnt and percent that are considered stable. Window sizes of 30 are used for Sudden and Gradual drifting hyperplane datasets, 480 for heating simulation datasets (capturing instances across a period of 10 days), and 90 for following distance datasets

In addition to the impact of drift sensitivity, we considered the selection of an appropriate window size. Figure 2 presents the results obtained when varying window size for each drift detection strategy. This parameter determines how much data is made available to learn a new model in the presence of concept drift, and how much data is retained in order to detect drifts. The results presented in Fig. 2 used a fixed drift sensitivity value for each drift detector, representative of the results presented in Fig. 1, and displayed in Table 3, to enable fair comparison. Across synthetic sudden and gradual drifting hyperplane datasets, RePro, ADWIN and AWPro (Fig. 2a, b and c respectively)



(a) RePro with varying W_{max} (b) ADWIN with varying W_{min} (c) AWPro with varying W_{min}

Fig. 2 Performance and number of stable models created by RePro, ADWIN and AWPro with varying window sizes, W_{max} for RePro and W_{min} for ADWIN and AWPro. Annotated with the total number of models learnt and percent that are considered stable. Drift sensitivities of $\lambda_d = 0.6$, $\delta = 0.02$ and $\delta = 0.02$ have been used for RePro, ADWIN and AWPro respectively across all datasets

maintained similar ratios of stable to unstable models, regardless of window size; however, the performance of the drift detection strategy typically decreased as the window size increased. This phenomenon is observed due to the presence of simple underlying concepts in the synthetic data streams; therefore, effective predictive models can be learnt from little data. Increasing the window size decreased predictive performance in these datasets as it

Table 3 Window size and drift sensitivity parameters used by RePro, ADWIN and AWPro to obtain results presented in Section 8

	RePro			ADWIN		AWPro		
	W_{max}	λ_d	λ_l	W_{min}	δ	W_{min}	δ	λ_r
SuddenA	30	0.5	0.01	30	0.02	30	0.02	0.5
GradualA	30	0.5	0.01	30	0.02	30	0.02	0.5
Heating	10 days	0.5	0.5 °C	10 days	0.02	10 days	0.02	0.5
Following	90 s	0.5	0.1 s	90 s	0.02	90 s	0.02	0.5

delayed drift detection. However, a significant increase in performance is observed as the window size increased for each drift detection strategy for the real-world following distance data streams as the concepts to be learnt are more complex, and therefore require more data to be made available to the target learner in order to build a predictive model that effectively represents the current concept [2]. Figure 2 highlights that the performance and ratio of stable to unstable models are significantly affected by the window size. However, all drift detection strategies perform similarly, indicating that the window size is dependent on the data stream to be learnt from, rather than the drift detection strategy. From the results presented in Fig. 2, we selected window sizes of 30 instances for sudden and gradual drifting hyperplane datasets, 480 instances for smart home heating simulation datasets which encapsulates 10 days of observations and 90 instances for following distance datasets which encapsulates 90 s of observations.

AWPro has an additional parameter, λ_r, which determines if a historical model can be reused. As its functionality is similar to how λ_d is used by RePro to identify recurring concepts, this parameter value for AWPro has been selected based on the analysis of RePro in Figs. 1 and 2 to allow fair comparisons between the drift detection strategies.

To ensure fair comparisons, we selected a single window size per data stream to be used by all concept drift detection strategies, and a single drift sensitivity value per drift detection strategy to be used across different data streams. The parameter values displayed in Table 3 were used to obtain the results presented in Section 8. Overall, parameter values were selected such that the window size is small to allow swift drift detection, but ensures sufficient data is retained to build an effective predictive model. This was inferred by considering the percentage of models that were considered stable. Drift sensitivity parameters were selected that not only prioritised high performance, but also took into account communicational and computational overheads. This was inferred by considering the number of both stable and unstable models.

8 Experimental results

We compared BOTL, using RePro, ADWIN and AWPro as the underlying concept drift detectors, against each of the concept drift detection strategies with no knowledge transfer, and an existing state-of-the-art online transfer learning (GOTL) framework [12], using the drifting hyperplane, heating simulation, and following distance datasets. BOTL is model agnostic; however, in order to make comparisons between BOTL and existing techniques, all implementations used ε-insensitive Support Vector Regressors (SVRs) as base learners.

The underlying concept drift detection strategies, RePro, ADWIN and AWPro, are used to determine a baseline performance threshold, obtained when no knowledge is transferred [33]. For each of these drift detection strategies, parameter values were chosen based on the discussion outlined in Section 7.4 such that each drift detection strategy aims to balance the trade-off between performance and computational and communication overheads.

GOTL was designed to learn from an offline source; however, as we are considering the implications of both domains being online, we used the underlying concept drift detection strategies to detect individual concepts in the source domain. This is necessary as many online applications cannot retain an entire history of data, preventing a single model from being learnt across the entire data stream. We used the drift detection strategies to identify the model that had been used in the source for the largest proportion of the data stream, and therefore is considered to be the most stable. GOTL transferred this model from the source domain to the target to enhance the effectiveness of the target predictor. A small step size, $\Delta = 0.025$, was chosen, as suggested by Grubinger et al. [11], which slowly modified the weights used to combine source and target models.

When evaluating GOTL, experiments were conducted such that each data stream was paired with every other data stream as source and target domains respectively. Due to only transferring the most stable model when using GOTL, learning in the target domain only commenced once learning in the source domain had completed such that the most stable source model could be identified and transferred to the target domain. Additionally, the performance of GOTL presented in this section takes into account both the performance of the source and target domains, as GOTL requires learning in the source, without knowledge transfer prior to learning in the target, whereas BOTL allows both domains to benefit from knowledge transfer simultaneously.

BOTL combines knowledge via the OLS meta-learner and therefore no additional parameters are required; however, the BOTL-C culling parameters must be defined.

We set $\lambda c_{perf} = 0$ for BOTL-C.I, thereby discarding models that performed worse than the average predictor ($R^2 < 0$). To ensure BOTL-C.II used a more aggressive approach to model culling, we increased λc_{perf} to 0.2. Additionally, as small window sizes were used to enable swift drift detection, we used $\lambda c_{MI} = 0.95$ to allow knowledge of similar concepts to be retained by the meta-learner to aid predictions of complex concepts. However, models with extremely high mutual information were not both retained, as little to no beneficial knowledge would be provided to the meta-learner if both remained in the model set, M.

When evaluating BOTL and BOTL-C variants, all data streams for a given experiment were used as source domains with bi-directional transfer. Repeat experiments were conducted by randomising the ordering and interval between the commencement of learning in each domain. For the baseline concept drift detection strategies without knowledge transfer, all data streams were learnt from independently.

8.1 Drifting hyperplane

We considered the effectiveness of BOTL on synthetic data created using the drifting hyperplane data generator containing two types of drift: sudden and gradual. We conducted four experiments on each type of drift using the drifting hyperplane data generator, investigating the impact of different types of noise that may be encountered when using BOTL in real-world environments.

Table 4 Drifting hyperplanes: average performance (R^2, PMCC^2, RMSE) and number of models used by the meta learner ($|M|$) to make predictions using no knowledge transfer, GOTL, BOTL and BOTL-C variants for six sudden drifting domains, where * indicates $p < 0.01$ in comparison with RePro and GOTL, and italicised values indicate the highest R^2 performance

	RePro				ADWIN				AWPro			
	R^2	PMCC^2	RMSE	$\|M\|$	R^2	PMCC^2	RMSE	$\|M\|$	R^2	PMCC^2	RMSE	$\|M\|$
(a) SuddenA: sudden drifting hyperplanes with uniform noise												
noTransfer	0.830 (± 0.003)	0.859	0.076	1	0.755 (± 0.009)	0.761	0.091	1	0.698 (± 0.015)	0.703	0.100	1
GOTL	0.814 (± 0.002)	0.844	0.079	1.5	0.745 (± 0.004)	0.752	0.092	1.5	0.685 (± 0.007)	0.690	0.102	1.5
BOTL	**0.903* (± 0.002)	0.904	0.057	29.9	**0.888* (± 0.003)	0.889	0.061	46.9	**0.887* (± 0.003)	0.888	0.062	19.0
BOTL-C.I	*0.894 (± 0.002)	0.895	0.060	10.1	*0.881 (± 0.003)	0.882	0.063	14.0	*0.861 (± 0.004)	0.862	0.068	5.7
BOTL-C.II	*0.888 (± 0.002)	0.888	0.061	3.8	*0.867 (± 0.003)	0.868	0.067	2.2	*0.842 (± 0.007)	0.843	0.072	2.0
(b) SuddenB: sudden drifting hyperplanes with single sensor failure												
noTransfer	0.825 (± 0.002)	0.849	0.074	1	0.723 (± 0.016)	0.732	0.092	1	0.661 (± 0.023)	0.677	0.101	1
GOTL	0.811 (± 0.002)	0.836	0.077	1.5	0.718 (± 0.006)	0.725	0.093	1.5	0.664 (± 0.010)	0.676	0.101	1.5
BOTL	−2e+20 (± 8e+19)	0.606	9e+8	30.9	−4e+19 (± 1*e*+19)	0.598	4e+8	47.1	−7e+20 (± 2e+20)	0.586	1e+9	20.4
BOTL-C.I	−3e+20 (± 1e+20)	0.817	8e+8	10.0	−9e+19 (± 2e+19)	0.663	5e+8	13.7	**0.862* (± 0.006)	0.863	0.065	5.9
BOTL-C.II	**0.890* (± 0.002)	0.890	0.059	4.1	**0.870* (± 0.002)	0.871	0.064	2.6	*0.845 (± 0.007)	0.846	0.069	2.1
(c) SuddenC: sudden drifting hyperplanes with intermittent single sensor failure												
noTransfer	0.833 (± 0.004)	0.859	0.074	1	0.778 (± 0.004)	0.787	0.085	1	0.765 (± 0.005)	0.774	0.087	1
GOTL	0.817 (± 0.002)	0.843	0.077	1.5	0.770 (± 0.002)	0.780	0.087	1.5	0.752 (± 0.003)	0.762	0.090	1.5
BOTL	**0.907* (± 0.002)	0.908	0.055	30.7	**0.892* (± 0.002)	0.894	0.059	48.9	**0.893* (± 0.002)	0.895	0.059	19.9
BOTL-C.I	*0.900 (± 0.002)	0.900	0.057	10.7	*0.885 (± 0.002)	0.886	0.061	16.0	*0.876 (± 0.002)	0.878	0.064	6.6
BOTL-C.II	*0.891 (± 0.003)	0.891	0.060	4.4	*0.869 (± 0.002)	0.870	0.066	2.3	*0.859 (± 0.003)	0.860	0.068	1.9
(d) SuddenD: sudden drifting hyperplanes with gradual sensor deterioration												
noTransfer	0.827 (± 0.004)	0.854	0.074	1	0.770 (± 0.001)	0.778	0.085	1	0.647 (± 0.031)	0.664	0.103	1
GOTL	0.809 (± 0.002)	0.838	0.078	1.5	0.764 (± 0.001)	0.773	0.087	1.5	0.649 (± 0.014)	0.662	0.104	1.5
BOTL	−3e+20 (± 6e+19)	0.304	2e+9	31.8	−4e+20 (± 1e+20)	0.299	1e+9	48.9	−7e+21 (± 2e+21)	0.299	7e+9	19.7
BOTL-C.I	−1e+21 (± 4e+20)	0.312	3e+9	10.0	−3e+19 (± 5e+18)	0.312	6e+8	13.4	−1e+19 (± 6e+18)	0.790	1e+8	5.7
BOTL-C.II	**0.891* (± 0.003)	0.891	0.059	3.9	**0.868* (± 0.003)	0.869	0.065	2.6	**0.843* (± 0.004)	0.845	0.070	2.2

Firstly, we used drifting hyperplane datasets containing uniform noise, denoted by datasets *SuddenA* and *GradualA* for the sudden drifting, and gradual drifting hyperplane datasets respectively. Secondly, we considered the impact of single sensor failure. Datasets of this nature are denoted as *SuddenB* and *GradualB*. Thirdly, we introduced the scenario of intermittent single sensor failure, denoted by *SuddenC* and *GradualC*. These datasets allowed us to investigate the use of BOTL within unreliable environments. Finally, we emulated single sensor deterioration by increasing the amount of noise associated with a feature vector throughout the data stream. Datasets containing this variant of sensor failure are denoted by *SuddenD* and *GradualD* for sudden and gradual drifting data streams.

For each variant of experiments, six data streams were created for each drift type. Each data stream contained five concepts, occurring four times throughout the data stream, with drifts encountered every 500 time steps. Sudden drifts occurred immediately, and gradual drifts occurred over a period of 100 time steps. Each data stream shared at most three concepts with another domain, ensuring some models transferred were useful to the target learner, while others were not. Data streams were separated such that transfer occurred only between domains of the same drift and noise type.

Tables 4 and 5 present the results obtained by the concept drift detection algorithms, with no knowledge transfer, GOTL and BOTL variants, for the sudden and gradual

Table 5 Drifting hyperplanes: average performance (R^2, PMCC^2, RMSE) and number of models used by the meta learner ($|M|$) to make predictions using no knowledge transfer, GOTL, BOTL and BOTL-C variants for six gradual drifting domains, where * indicates $p < 0.01$ in comparison with RePro and GOTL, and italicised values indicate the highest R^2 performance

	RePro				ADWIN				AWPro			
	R^2	PMCC^2	RMSE	$\|M\|$	R^2	PMCC^2	RMSE	$\|M\|$	R^2	PMCC^2	RMSE	$\|M\|$
(a) GradualA: gradual drifting hyperplanes with uniform noise												
noTransfer	0.797 (± 0.001)	0.822	0.078	1	0.711 (± 0.010)	0.714	0.093	1	0.710 (± 0.010)	0.715	0.093	1
GOTL	0.792 (± 0.001)	0.820	0.079	1.5	0.712 (± 0.005)	0.717	0.093	1.5	0.710 (± 0.005)	0.718	0.093	1.5
BOTL	**0.904* (± 0.001)	0.905	0.054	34.2	**0.901* (± 0.002)	0.902	0.054	45.5	**0.893* (± 0.003)	0.894	0.057	19.0
BOTL-C.I	*0.897 (± 0.002)	0.897	0.056	15.1	*0.894 (± 0.002)	0.895	0.056	19.4	*0.876 (± 0.004)	0.877	0.061	7.9
BOTL-C.II	*0.881 (± 0.003)	0.882	0.060	7.1	*0.872 (± 0.004)	0.872	0.062	4.0	*0.861 (± 0.005)	0.862	0.064	3.4
(b) GradualB: gradual drifting hyperplanes with single sensor failure												
noTransfer	0.797 (± 0.001)	0.821	0.081	1	0.703 (± 0.019)	0.711	0.097	1	0.639 (± 0.021)	0.646	0.107	1
GOTL	0.784 (± 0.001)	0.808	0.084	1.5	0.706 (± 0.008)	0.715	0.097	1.5	0.641 (± 0.010)	0.649	0.107	1.5
BOTL	−2e+19 (± 7e+18)	0.451	2e+8	39.6	−2e+17 (± 6e+16)	0.447	4e+7	46.1	−8e+18 (± 1e+18)	0.445	2e+8	20.7
BOTL-C.I	−1e+21 (± 3e+20)	0.557	1e+9	15.3	−2e+18 (± 7e+17)	0.630	1e+8	16.2	−2e+18 (± 8e+17)	0.826	5e+7	7.6
BOTL-C.II	**0.876* (± 0.003)	0.876	0.063	6.0	**0.852* (± 0.006)	0.852	0.069	3.0	**0.823* (± 0.007)	0.824	0.075	2.7
(c) GradualC: gradual drifting hyperplanes with intermittent single sensor failure												
noTransfer	0.794 (± 0.003)	0.818	0.078	1	0.695 (± 0.015)	0.698	0.094	1	0.698 (± 0.015)	0.704	0.093	1
GOTL	0.778 (± 0.002)	0.802	0.080	1.5	0.695 (± 0.007)	0.700	0.094	1.5	0.697 (± 0.007)	0.704	0.094	1.5
BOTL	**0.899* (± 0.003)	0.900	0.054	34.0	**0.876* (± 0.008)	0.877	0.060	44.9	**0.875* (± 0.008)	0.876	0.060	21.3
BOTL-C.I	*0.894 (± 0.003)	0.895	0.056	14.8	*0.861 (± 0.010)	0.862	0.063	18.5	*0.857 (± 0.009)	0.858	0.064	8.8
BOTL-C.II	*0.882 (± 0.003)	0.883	0.059	6.7	*0.842 (± 0.009)	0.843	0.067	4.5	*0.841 (± 0.009)	0.841	0.068	3.4
(d) GradualD: gradual drifting hyperplanes with gradual sensor deterioration												
noTransfer	0.806 (± 0.003)	0.828	0.090	1	0.691 (± 0.007)	0.694	0.114	1	0.640 (± 0.010)	0.643	0.122	1
GOTL	0.791 (± 0.002)	0.813	0.093	1.5	0.685 (± 0.004)	0.691	0.114	1.5	0.640 (± 0.005)	0.643	0.122	1.5
BOTL	−9e+19 (± 1e+19)	0.148	1e+9	36.0	−3e+20 (± 1e+20)	0.149	1e+9	44.3	−8e+20 (± 1e+20)	0.148	3e+9	22.3
BOTL-C.I	−5e+20 (± 2e+20)	0.393	1e+9	12.6	−3e+19 (± 1e+19)	0.565	3e+8	13.3	−4e+19 (± 1e+19)	0.601	3e+8	6.8
BOTL-C.II	**0.890* (± 0.003)	0.890	0.068	5.7	**0.869* (± 0.007)	0.870	0.073	3.8	**0.845* (± 0.007)	0.846	0.080	2.8

drifting hyperplane datasets respectively. These results indicate that GOTL obtained slightly poorer performances in comparison with the concept drift detectors without knowledge transfer, despite the most stable source model being transferred to the target domain. Although knowledge transfer was not beneficial to GOTL, at least one of the BOTL variants was able to outperform RePro, ADWIN, AWPro and GOTL with statistical t tests achieving p values < 0.01, highlighting the importance of transferring knowledge of multiple concepts bi-directionally.

The performance increase of BOTL over GOTL ($p < 0.01$) on datasets containing uniform noise (*SuddenA*, *GradualA*), and intermittent sensor failure (*SuddenC*, *GradualC*), can be attributed to the availability of all source models in the target domain. Additionally, GOTL's stepwise weighting mechanism prevents the influence of a model changing drastically over a small period of time. This means a large amount of data must be observed after each drift to converge on an approximation of the optimal weights. To overcome this, a larger step size could be used; however, this may prevent or hinder convergence. BOTL overcomes this by using the OLS meta learner to minimise the squared error of the combined predictor with instantaneous effect.

The performances of BOTL-C variants were also significantly better than the underlying drift detection algorithms and GOTL on these data streams, obtaining t test values of $p < 0.01$; however, they performed slightly worse than BOTL. Figure 3 highlights the aggressive nature of the culling techniques used by BOTL-C.I and BOTL-C.II on a sudden drifting hyperplane data stream with uniform noise. It shows BOTL used at least four times more models than BOTL-C variants and highlights correlations between the number of models used and performance. When the number of models used was small, the predictive performance of BOTL-C variants decreased. This performance decrease can be attributed to the aggressive nature of these culling mechanisms. Culling based on model performance alone prohibited the inclusion of a diverse set of models, reducing the overall predictive performance of the meta learner. When BOTL-C variants retained a larger proportion of the transferred models, a performance similar to BOTL was achieved.

However, BOTL and BOTL-C.I were not able to outperform RePro, ADWIN, AWPro or GOTL in the data streams containing single sensor failure (SuddenB, GradualB) and gradual sensor deterioration (SuddenD, GradualD). Although BOTL and BOTL-C.I obtained significantly lower R^2 performances, their PMCC^2 performance values were impacted less significantly. This indicates that the poor performance of BOTL and BOTL-C.I on these data streams can be attributed to the transfer of high volumes of models, causing the OLS meta learner to overfit. To overcome this, the performance culling threshold, λc_{perf}, could be increased for BOTL-C.I, further restricting the number of models used as input to the meta learner. The more aggressive nature of the culling technique used by BOTL-C.II meant it outperformed the concept drift detection strategies with no knowledge transfer and GOTL.

Overall, RePro was able to achieve a better performance across all drifting hyperplane data streams compared with ADWIN and AWPro. This was caused by RePro monitoring the performance of a model to detect concept drifts, whereas ADWIN and AWPro monitor the distribution

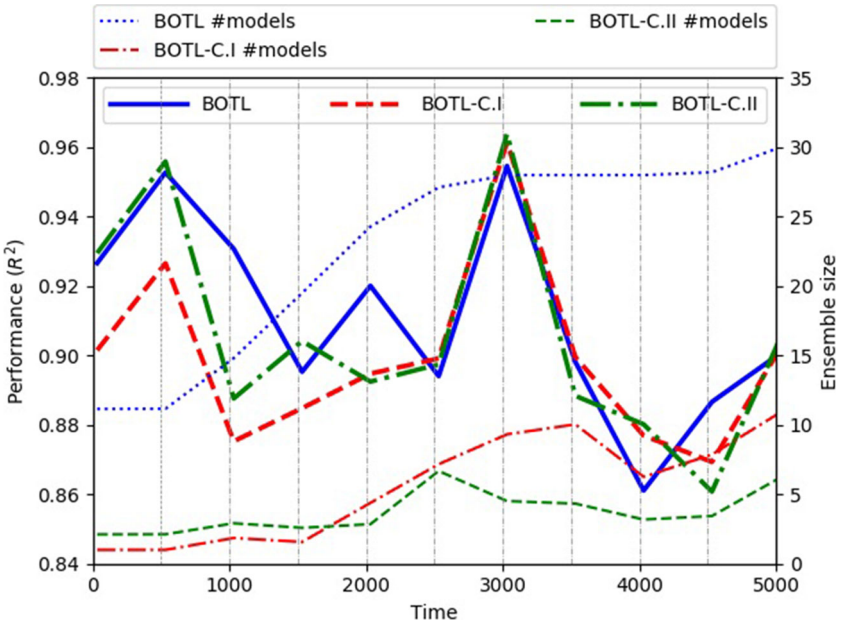


Fig. 3 Sudden drifting hyperplanes: R^2 performance and number of models used by BOTL and BOTL-C variants using two SuddenA data streams where vertical lines indicate concept drifts

Table 6 Heating simulations: average performance (R^2, PMCC^2, RMSE) and number of models used by the meta learner ($|M|$) to predict desired heating temperatures across five domains using no knowledge transfer, GOTL, BOTL and BOTL-C variants, where * indicates $p < 0.01$ in comparison with the concept drift detection algorithm and GOTL, and italicised values indicate the highest R^2 performance

	RePro				ADWIN				AWPro			
	R^2	PMCC^2	RMSE	$\|M\|$	R^2	PMCC^2	RMSE	$\|M\|$	R^2	PMCC^2	RMSE	$\|M\|$
noTransfer	0.633 (± 0.002)	0.651	2.521	1	0.635 (± 0.013)	0.655	2.509	1	0.612 (± 0.009)	0.633	2.590	1
GOTL	0.666 (± 0.005)	0.677	2.397	1.5	0.656 (± 0.005)	0.670	2.434	1.5	0.618 (± 0.006)	0.635	2.567	1.5
BOTL	**0.744* (± 0.006)	0.747	2.104	8.6	**0.746* (± 0.007)	0.749	2.097	7.4	**0.746* (± 0.007)	0.749	2.093	8.2
BOTL-C.I	0.664 (± 0.028)	0.693	2.377	5.6	*0.739 (± 0.006)	0.742	2.126	4.8	*0.737 (± 0.006)	0.740	2.134	5.3
BOTL-C.II	0.653 (± 0.028)	0.683	2.415	1.7	*0.702 (± 0.011)	0.710	2.264	1.2	*0.727 (± 0.005)	0.730	2.175	1.4

of predictive error. Detecting drifts in this way was beneficial to RePro, as unstable models were repeatedly learnt and discarded until the sliding window of data contained instances that were representative of the current concept. Although the use of RePro as the underlying drift detector outperformed ADWIN and AWPro, the difference in performance across these drift detection strategies was not statistically significant; therefore, the computational overhead of repeatedly learning unstable models may impact RePro's applicability as the underlying drift detector for BOTL in real-world environments.

8.2 Heating simulation

Lower performances were observed across the heating simulation datasets due to containing more complex concepts, and additional noise, in comparison with the drifting hyperplane datasets. The addition of knowledge transfer, using GOTL and BOTL, provided an increase in performance in comparison with using the concept drift detection strategies, with no knowledge transfer, as shown in Table 6. GOTL, BOTL and BOTL-C variants, using each drift detector, performed better than the drift detection strategy alone, with GOTL achieving statistical t test p values of $p < 0.01$ over RePro with no knowledge transfer.

The use of GOTL in this setting highlighted the advantage of knowledge transfer when concepts were more complex, preventing the underlying concept drift detectors from building effective models on the window of available data. This meant the knowledge transferred helped enhance the performance of the target predictor, even when only a single model was transferred using GOTL. Whereas using GOTL in environments that have simple concepts to be learnt, such as those present in the hyperplane datasets, provided little to no benefit. Transferring multiple models provided a significant benefit as all BOTL variants performed better than GOTL with a t test p value < 0.01, for all concept drift detection strategies.

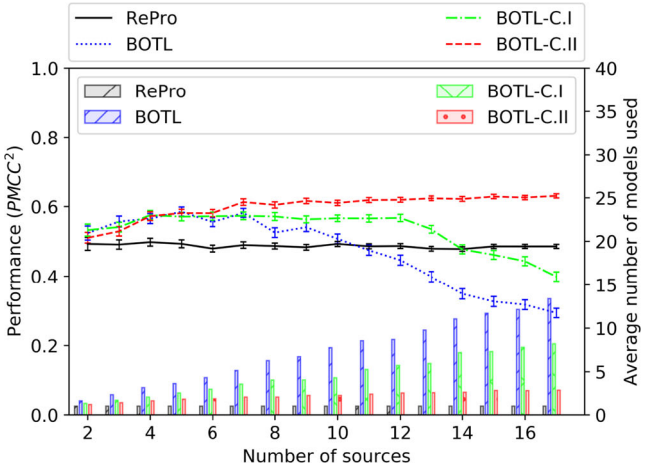
8.3 Following distance

Finally, we evaluated BOTL on real-world data using the following distance dataset, where the task was to predict TTC. Due to the real-world nature of this data, concept drifts occurred frequently and data streams were noisy.

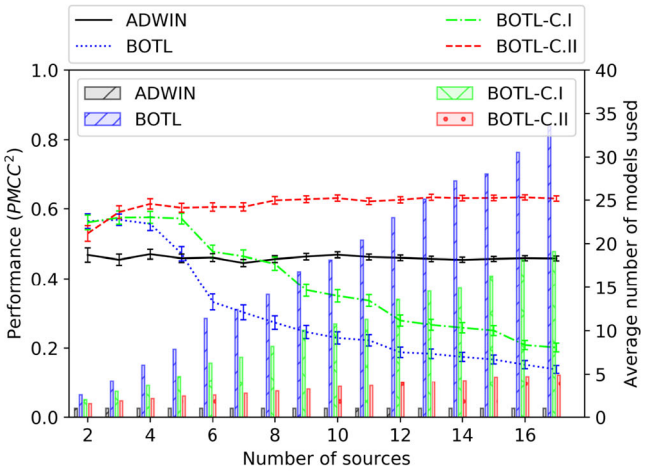
Table 7 shows the performance of drift detectors, GOTL and BOTL variants across seven data streams. These results highlight GOTL was less suitable when the relationship between source and target concepts were unknown. Variants of BOTL and BOTL-C that used RePro or AWPro as drift

Table 7 Following distances: average performance (R^2, PMCC^2, RMSE) and number of models used by the meta learner ($|M|$) to predict TTC across seven domains using no knowledge transfer, GOTL, BOTL and BOTL-C variants, where * indicates $p < 0.01$ in comparison to the concept drift detection algorithm and GOTL, and italicised values indicate the highest R^2 performance

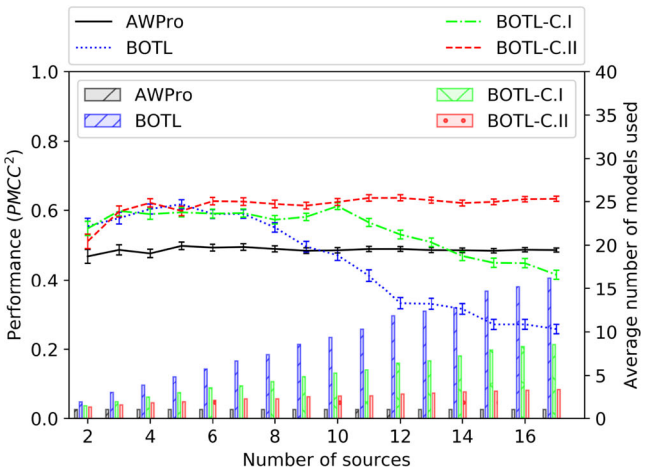
	RePro				ADWIN				AWPro			
	R^2	PMCC^2	RMSE	$\|M\|$	R^2	PMCC^2	RMSE	$\|M\|$	R^2	PMCC^2	RMSE	$\|M\|$
noTransfer	0.547 (± 0.002)	0.572	0.600	1	0.441 (± 0.022)	0.502	0.676	1	0.430 (± 0.025)	0.514	0.681	1
GOTL	0.602 (± 0.007)	0.657	0.558	1.5	0.419 (± 0.011)	0.560	0.676	1.5	0.487 (± 0.013)	0.603	0.634	1.5
BOTL	0.636 (± 0.011)	0.674	0.546	5.2	−2e+15 (± 3e+14)	0.406	1e+7	10.1	*0.653 (± 0.013)	0.680	0.537	5.0
BOTL-C.I	0.655 (± 0.012)	0.685	0.532	3.2	−1e+14 (± 4e13)	0.604	1e+6	5.1	*0.660 (± 0.009)	0.680	0.530	3.0
BOTL-C.II	**0.662* (± 0.011)	0.686	0.524	1.1	**0.641* (± 0.017)	0.662	0.542	1.6	**0.691* (± 0.009)	0.701	0.507	1.3



(a) RePro.

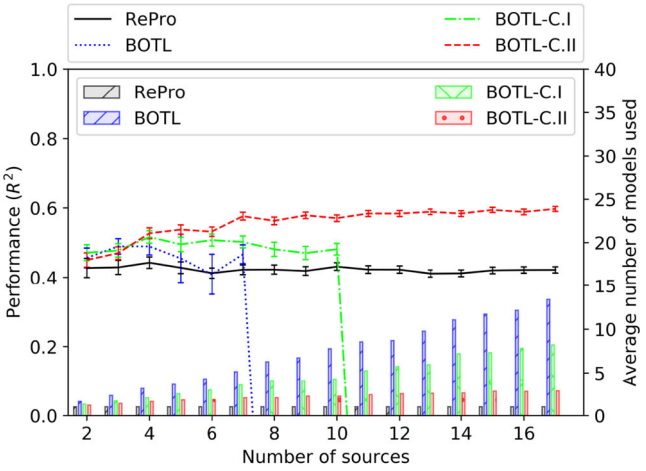


(b) ADWIN.

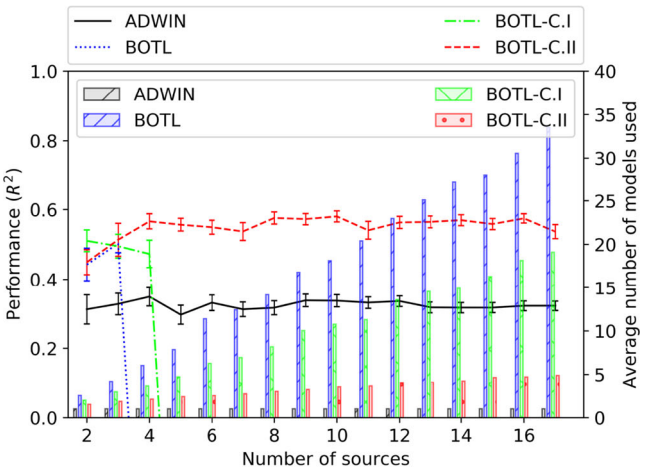


(c) AWPro.

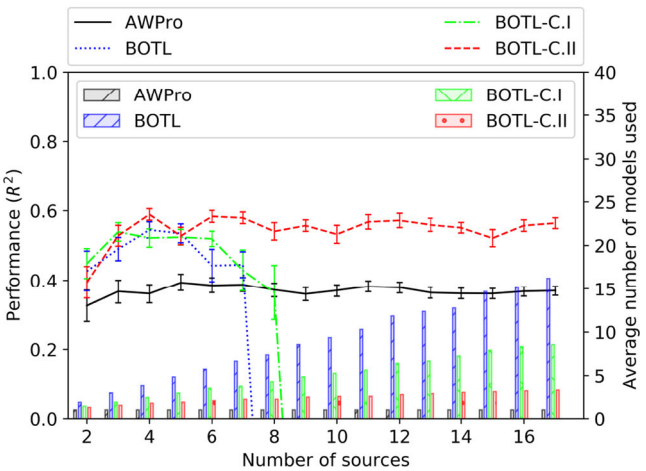
Fig. 4 Following distances: $PMCC^2$ performance (with standard error) and number of models used by concept drift detection strategies with no knowledge transfer, BOTL and BOTL-C variants as the number of domains increase



(a) RePro.



(b) ADWIN.



(c) AWPro.

Fig. 5 Following distances: R^2 performance (with standard error) and number of models used by the concept drift detection strategies with no knowledge transfer, BOTL and BOTL-C variants as the number of domains increase

detectors performed better than their respective baseline drift detector and GOTL, achieving statistical t test p values of $p < 0.01$. However, the BOTL and BOTL-C.I implementations that used ADWIN as the underlying drift detector were not able to outperform the drift detector with no knowledge transfer, or GOTL. Although these variants of BOTL performed poorly, BOTL-C.II using ADWIN as the underlying drift detector outperformed ADWIN and GOTL, achieving statistical t test p values of $p < 0.01$. This highlights the importance of preventing the transfer of redundant models when using BOTL in real-world environments, as the large numbers of transferred models likely caused the OLS meta learner to overfit the local window of data. This observation was also supported by BOTL-C.II achieving the best performance in comparison with other BOTL variants.

To investigate scalability, Figs. 4 and 5 display the average $PMCC^2$ and R^2 performance respectively per domain, and the number of models used by the OLS meta learner to make predictions as the number of domains in the framework increased. For settings with a small number of domains, BOTL and BOTL-C variants achieved similar $PMCC^2$ and R^2 performances, outperforming their respective baseline concept drift detection algorithms. However, as the number of domains expanded, and the number of models transferred increased, the $PMCC^2$ performance of BOTL dropped below the performance of the concept drift detection algorithm with no knowledge transfer. Although $PMCC^2$ gradually decreased as the number of sources increased, by considering the R^2 performance in Fig. 5, we observe that the average R^2 performance decreased drastically. This occurred due to the nature of these performance metrics. As $PMCC^2$ ranges between [0, 1], when one domain performed poorly, it did not greatly impact the average $PMCC^2$ across all domains, whereas R^2 ranges between $(-\infty, 1]$; therefore, when one domain performed poorly, the average R^2 performance was greatly impacted. The difference between performance metrics, shown in Figs. 4 and 5, indicates that BOTL and BOTL-C.I suffered from the OLS meta learner overfitting the small window of local data when the number of models transferred was large. Culling using the performance of transferred models alone (BOTL-C.I) enabled a larger number of domains to be used in the framework, however cannot be considered scalable as the performance of BOTL-C.I decreased below that of the drift detector when more domains were added. BOTL-C.II culled more aggressively, using diversity alongside performance, ensuring enough beneficial knowledge was retained to enhance the target learners' performance, while minimising negative transfer and preventing the OLS meta learner overfitting the small window of locally available data.

Additionally, Figs. 4b and 5b show the $PMCC^2$ and R^2 performance when ADWIN was used as the underlying concept drift detector. Compared with Figs. 4a and 5a, and 4c and 5c, which used RePro and AWPro respectively, the $PMCC^2$ and R^2 performances obtained by ADWIN reduced quickly, even when a small number of domains were included. This again highlights the importance of selecting a concept drift detection strategy that reuses existing models in the presence of recurring concepts, instead of relearning and transferring duplicate models.

Overall, these results indicate that the ability to consider both source and target domains to be online is beneficial. In doing so, the number of transferred models greatly increases, requiring culling mechanisms, particularly when used in noisy real-world data streams, to retain the benefit of transferring knowledge between domains.

9 Conclusion

Online domains that must learn complex models often have limited data availability, and are hindered by the presence of concept drift. We have presented the BOTL framework, and two BOTL-C variants, that enable knowledge to be transferred across online domains. We enhanced predictive performance by combining knowledge transferred from other online domains using an OLS meta learner, enabling additional knowledge to be used to minimise the error of the overarching prediction.

Using RePro as the underlying concept drift detection strategy ensured effective models were learnt from the available data; however, RePro may not be appropriate for use in applications that have computational limitations due to frequently creating unstable models during, or immediately after, periods of drift. Applications that have computational limitations may need to trade-off performance with computation and use a drift detection strategy such as ADWIN or AWPro. ADWIN does not reuse previously learnt models when drifts are detected; therefore, a number of models transferred between domains increases when recurring concepts are encountered. This can degrade the performance of BOTL when the number of domains in the framework, or volume of models transferred, is large.

Instead, AWPro can be used in settings where recurring concepts are likely to be encountered, or computational and communicational resources are limited.

In this paper, we chose RePro, ADWIN and AWPro as the underlying concept drift detection algorithms. Although each of these requires some domain expertise to identify appropriate parameter values, their ability to retain a history of models to prevent relearning recurring concepts is a more influential factor to consider when selecting an

underlying concept drift detector for BOTL. RePro and AWPro helped to reduce the number of models transferred between domains and therefore allowed more domains to be included in the framework before the OLS meta learner suffered from overfitting.

However, in real-world environments with many domains, the number of models transferred may need to be reduced further. BOTL-C variants achieved this using common ensemble pruning strategies. These pruning strategies also required culling parameter values to be specified. To overcome the need to specify these additional parameters, we will investigate the use of task relatedness to identify similar concepts across domains without requiring parameterised thresholds in future work. This will reduce the dependency on domain expertise and will allow BOTL to be used for applications that require scalability to larger numbers of domains.

Funding This work was supported by the UK EPSRC and Jaguar Land Rover under the iCASE scheme.

References

1. Arnold A, Nallapati R, Cohen WW (2007) A comparative study of methods for transductive transfer learning. In: Seventh IEEE international conference on data mining workshops (ICDMW 2007), pp 77–82
2. Bifet A (2009) Adaptive learning and mining for data streams and frequent patterns. SIGKDD Explor Newsl 11(1):55–56. https://doi.org/10.1145/1656274.1656287
3. Bifet A, Gavalda R (2007) Learning from time-changing data with adaptive windowing. In: Proceedings of the 2007 SIAM international conference on data mining. SIAM, pp 443–448, https://doi.org/10.1137/1.9781611972771.42
4. Budka M, Gabrys B (2010) Ridge regression ensemble for toxicity prediction. Procedia Computer Science 1(1):193–201. https://doi.org/10.1016/j.procs.2010.04.022. ICCS 2010
5. Daume III H, Marcu D (2006) Domain adaptation for statistical classifiers. J Artif Intell Res 26:101–126. https://doi.org/10.1613/JAIR.1872
6. Dong B, Li Y, Gao Y, Haque A, Khan L, Masud MM (2017) Multistream regression with asynchronous concept drift detection. In: 2017 IEEE international conference on big data, pp 596–605, https://doi.org/10.1109/BIGDATA.2017.8257975
7. Dutta H (2009) Measuring diversity in regression ensembles. In: IICAI, vol 9, p 17. Citeseer
8. Friedman JH (1997) On bias, variance, 0/1—loss, and the curse-of-dimensionality. Data Min Knowl Disc 1(1):55–77. https://doi.org/10.1023/A:1009778005914
9. Gama J, Žliobaitė I, Bifet A, Pechenizkiy M, Bouchachia A (2014) A survey on concept drift adaptation. ACM Comput Surv 46(4):44:1–44:37. https://doi.org/10.1145/2523813
10. Ge L, Gao J, Zhang A (2013) Oms-tl: a framework of online multiple source transfer learning. In: Proceedings of the 22nd ACM international conference on information & knowledge management, CIKM '13. ACM, pp 2423–2428, https://doi.org/10.1145/2505515.2505603
11. Grubinger T, Chasparis G, Natschläger T (2016) Online transfer learning for climate control in residential buildings. In: Proceedings of the 5th annual european control conference (ECC 2016), pp 1183–1188, https://doi.org/10.1109/ECC.2016.7810450
12. Grubinger T, Chasparis G, Natschläger T (2017) Generalized online transfer learning for climate control in residential buildings. Energy and Buildings 139:63–71. https://doi.org/10.1016/J.ENBUILD.2016.12.074
13. Hammerla NY, Plötz T (2015) Let's (not) stick together: pairwise similarity biases cross-validation in activity recognition. In: Proceedings of the 2015 ACM international joint conference on pervasive and ubiquitous computing, UbiComp '15. Association for Computing Machinery, pp 1041–1051, https://doi.org/10.1145/2750858.2807551
14. Haque A, Tao H, Chandra S, Liu J, Khan L (2018) A framework for multistream regression with direct density ratio estimation. In: Thirty-second AAAI conference on artificial intelligence
15. Hoens TR, Polikar R, Chawla NV (2012) Learning from streaming data with concept drift and imbalance: an overview. Progress in Artificial Intelligence 1(1):89–101. https://doi.org/10.1007/S13748-011-0008-0
16. Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining, KDD '01. ACM, pp 97–106, https://doi.org/10.1145/502512.502529
17. Kang Z, Yang B, Li Z, Wang P (2019) Otlamc: an online transfer learning algorithm for multi-class classification. Knowl-Based Syst 176:133–146. https://doi.org/10.1016/j.knosys.2019.03.024
18. Kelly MG, Hand DJ, Adams NM (1999) The impact of changing populations on classifier performance. In: Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining, KDD '99. ACM, pp 367–371, https://doi.org/10.1145/312129.312285
19. Kolter JZ, Maloof MA (2003) Dynamic weighted majority: a new ensemble method for tracking concept drift. In: Third IEEE international conference on data mining, pp 123–130, https://doi.org/10.1109/ICDM.2003.1250911
20. Kolter JZ, Maloof MA (2005) Using additive expert ensembles to cope with concept drift. In: Proceedings of the 22nd international conference on machine learning, ICML '05. ACM, pp 449–456, https://doi.org/10.1145/1102351.1102408
21. Li G, Hoi SC, Chang K, Liu W, Jain R (2014) Collaborative online multitask learning. IEEE Trans Knowl Data Eng 26(8):1866–1876. https://doi.org/10.1109/TKDE.2013.139
22. Murugesan K, Carbonell J (2017) Multi-task multiple kernel relationship learning. In: Proceedings of the 2017 SIAM international conference on data mining. SIAM, pp 687–695
23. Pan J, Hu X, Li P, Li H, He W, Zhang Y, Lin Y (2016) Domain adaptation via multi-layer transfer learning. Neurocomputing 190:10–24. https://doi.org/10.1016/J.NEUCOM.2015.12.097

24. Pan SJ, Yang Q (2010) A survey on transfer learning. IEEE Trans Knowl Data Eng 22(10):1345–1359. https://doi.org/10.1109/TKDE.2009.191
25. Ruvolo P, Eaton E (2013) Active task selection for lifelong machine learning. In: AAAI
26. Saha A, Rai P, Daumã H, Venkatasubramanian S (2011) Online learning of multiple tasks and their relationships. In: Proceedings of the 14th international conference on artificial intelligence and statistics, pp 643–651
27. Schlimmer JC, Granger RH (1986) Incremental learning from noisy data. Mach Learn 1(3):317–354. https://doi.org/10.1007/BF00116895
28. Street WN, Kim Y (2001) 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. ACM, pp 377–382, https://doi.org/10.1145/502512.502568
29. Sun Y, Wang Z, Liu H, Du C, Yuan J (2016) Online ensemble using adaptive windowing for data streams with concept drift. International Journal of Distributed Sensor Networks 12(5):4218,973
30. Tsymbal A (2004) The problem of concept drift: definitions and related work. Computer Science Department, Trinity College Dublin 106
31. Wu Q, Wu H, Zhou X, Tan M, Xu Y, Yan Y, Hao T (2017) Online transfer learning with multiple homogeneous or heterogeneous sources. IEEE Trans Knowl Data Eng 29(7):1494–1507. https://doi.org/10.1109/TKDE.2017.2685597
32. Yan Y, Wu Q, Tan M, Ng MK, Min H, Tsang IW (2018) Online heterogeneous transfer by hedge ensemble of offline and online decisions. IEEE Transactions on Neural Networks and Learning Systems 29(7):3252–3263. https://doi.org/10.1109/TNNLS.2017.2751102
33. Yang Y, Wu X, Zhu X (2005) Combining proactive and reactive predictions for data streams. In: Proceedings of the eleventh ACM SIGKDD international conference on knowledge discovery in data mining, KDD '05. ACM, pp 710–715, https://doi.org/10.1145/1081870.1081961
34. Yang Y, Wu X, Zhu X (2006) Mining in anticipation for concept change: proactive-reactive prediction in data streams. Data Min Knowl Disc 13(3):261–289. https://doi.org/10.1145/1081870.1081961
35. Yin H, Yang YA (2017) Online transfer learning with extreme learning machine. In: AIP conference proceedings, vol 1839. AIP Publishing, p 020199, https://doi.org/10.1063/1.4982564
36. Zhao P, Hoi SC (2010) Otl: a framework of online transfer learning. In: Proceedings of the 27th international conference on machine learning (ICML-10), pp 1231–1238
37. Zhou ZH (2012) Ensemble methods: foundations and algorithms. Chapman and Hall/CRC