Proceedings of Machine Learning Research 157, 2021 ACML 2021

Transfer Learning with Adaptive Online TrAdaBoost for Data Streams

Ocean Wu HWU344@AUCKLANDUNI.AC.NZ
Yun Sing Koh YKOH@CS.AUCKLAND.AC.NZ
Gillian Dobbie G.DOBBIE@AUCKLAND.AC.NZ
Thomas Lacombe THOMAS.LACOMBE@AUCKLAND.AC.NZ
School of Computer Science, The University of Auckland, New Zealand

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

In many real-world applications, data are often produced in the form of streams. Consider, for example, data produced by sensors. In data streams there can be concept drift where the distribution of the data changes. When we deal with multiple streams from the same domain, concepts that have occurred in one stream may occur in another. Therefore, being able to reuse knowledge across multiple streams can help models recover from concept drifts more quickly. A major challenge is that these data streams may be only partially identical and a direct adoption would not suffice. In this paper, we propose a novel framework to transfer both identical and partially identical concepts across different streams. In particular, we propose a new technique called Adaptive Online TrAdaBoost that tunes weight adjustments during boosting based on model performance. The experiments on synthetic data verify the desired properties of the proposed method, and the experiments on real-world data show the better performance of the method for data stream mining compared with its baselines.

Keywords: Transfer Learning; Online Boosting; Concept Drift; Data Stream

1. Introduction

Transfer learning aims to enhance the learning performance of a model in a target domain by transferring knowledge from source domains (Zhuang et al., 2020). Most existing studies on transfer learning focus on stationary environments, assuming that the data in each domain are generated from the same distribution. Addressing target tasks that have sparse or no labelled data has been well studied. Yet, this assumption may not hold where the environments are dynamic, *e.g.*, financial data inferences, energy demand predictions, and climate data analysis. In a non-stationary environment, the distributions of the generated data may evolve, thus rendering models designed using a previous distribution obsolete. This is known as concept drift. Some research efforts exist on transfer learning under non-stationary environments (Yang et al., 2021; Wu et al., 2017). Yet, very little work explicitly exploit all of the continuous, infinite, and concept-recurring characteristics of data streams. Every data stream can act as either the source, the target, or even both, to share and apply knowledge among each other.

Unlike most transfer learning techniques in the area, we focus on online transfer for evolving data streams. Current approaches either assume that there is a similarity between the source and target domains, or only use the underlying trends within the data streams without the assumption of similarity of concepts. However, we believe that there is a middle ground whereby partial model reuse is possible, given the models may be closely related.

The main contributions are summarised as follows. First, we propose a transfer learning framework for evolving data streams that implements transfer between identical domains. Secondly, we propose a novel Adaptive Online TrAdaBoost (AOTrAdaBoost) technique that tunes the sensitivity of weighting during the boosting process such that our framework can benefit from partially identical domains where noise and dissimilarities in data distributions may exist.

The remainder of this paper is organised as follows. In Section 2 we provide an overview of work related to transfer learning under data streams, followed by preliminaries to our work in Section 3. In Section 4 we present the details of our transfer learning framework and theoretical analysis of runtime and memory complexities. We then evaluate the performance of our techniques on both synthetic and real-world datasets in Section 5. Finally, Section 6 concludes our work and proposes future perspectives.

2. Related Work

This section discusses research related to transfer learning and the boosting techniques for data streams.

Transfer Learning for Data Streams. Numerous researches have been conducted on the topic of Transfer Learning in either the online or the offline settings, but few have focused on the data stream environments, especially for instance-incremental methods. Minku (2019) discusses the similarities between transfer learning and learning in non-stationary environments. Both Melanie (Du et al., 2019) and MARLINE (Du et al., 2020) are ensemble techniques that maintain a pool of ensembles for multiple data streams, whose predictions are based on the weighted majority vote among the predictions of the pool of ensembles. Higher weights are assigned to ensembles that are most appropriate to the current concept. In particular, Melanie learns from multiple data sources with the assumption that both source and target domains share similar concepts, and MARLINE performs transfer learning using multiple data sources that may have different concepts from the target. Wen et al. (2019) proposed a variant of Hoeffding Tree to adapt to online transfer learning, and a framework embedding such variant to select the optimal source model to do multi-sources online transfer learning by a measure of local similarity. Yet, the proposed framework does not handle concept drifts. BOTL (Mckay et al., 2020) enables bi-directional transfer learning between a single source and a single target domain under the regression setting.

Boosting Techniques. AdaBoost is a boosting algorithm where each iteration a model is trained on the instances with updated weights, based on a previous model. The weight is incremented if a previous model makes a wrong prediction, so that incorrect classifications are given more attention. TrAdaBoost (Dai et al., 2007) extends AdaBoost to the transfer learning scenario to deal with the negative effects of the instances coming from a different distribution. Different weight update strategies are applied based on the data distributions. Weights are incremented (*i.e.* AdaBoost) for same-distribution data (*i.e.* target domain), and decremented on the diff-distribution data (*i.e.* source domain) to reduce the impacts. TransferBoost (Eaton and desJardins, 2011) is a multiple source transfer learning algorithm based on TrAdaBoost which performs boosting at both the instance level and the task level. It selectively chooses the source knowledge to transfer to the target task based on task transferability. All the discussed boosting techniques have been adapted to instance-incremental learning methods for data streams (Oza and Russell, 2001; Wang and Pineau, 2015).

In particular, we consider transfer learning under homogeneous environments under the classification setting, while utilising the recurring models for instance transfer when concept drifts are detected in data streams. Our particular setting distinguishes our framework from the existing methodologies.

3. Preliminaries

A **data stream** is potentially an infinite sequence of data which arrives sequentially, $S = (s_1, s_2, \dots, s_t, \dots)$. An instance of the data stream S at time t is defined as a tuple $s_t = (x_t, y_t)$ where x is a feature vector used to predict the target variable y. The learning objective at a particular time t is to predict the corresponding y_t for a given feature vector x_t by a model H_t that was generated at the time stamp t, which results in a prediction $\hat{y}_{t+1} = H_t(x_{t+1})$. The real label y_{t+1} is compared with the prediction $\hat{y}_{t+1}$.

A **concept drift** (Gama and Kosina, 2014) occurs when the distribution from the feature x and target y changes between the two time steps t_0 and t_1, *i.e.*, $\exists x : P_{t_0}(x, y) \neq P_{t_1}(x, y)$. The joint distribution can also be written as $P_t(x, y) = P_t(x)P_t(y|x)$, where $P_t(x)$ is the distribution of the features and $P_t(y|x)$ is the posterior probability of the classes.

3.1. Problem Definition

Let $\mathcal{S} = \{S_0, S_1, \dots, S_{N-1}\}$ be N evolving data streams. Given the feature space $\mathcal{X}_i$ and the label space $\mathcal{Y}_i$, each $i \in \mathcal{S}$ is associated with a domain $D_i = \{\mathcal{X}_i, p_i(x)\}$ and a task $T_i = \{\mathcal{Y}_i, p_i(y|x)\}$, where $x \in \mathcal{X}$, $y \in \mathcal{Y}$, $p_i(x)$ is the marginal probability distribution, $p_i(y|x)$ is the posterior probability distribution. Each of the streams can either act as the source or the target data stream. Concept drifts may occur for all i in $\mathcal{S}$. For each concept drift we use c_i^j to denote the jth concept occurred in i. Our objective is to learn a function f that minimizes the risk for the upcoming c_i^j in the target data stream by model reuse. Specifically, we investigate our methodology under the inductive transfer learning setting (*i.e.* $\forall i, [\mathcal{Y}_i \neq \mathcal{Y}' \lor p_i(y|x) \neq p'(y|x)]$) in the context of evolving data streams.

3.2. Online AdaBoost and TrAdaBoost

Given $D_m(n)$ as the weight distribution of the nth sample (x_n, y_n), and ϵ_m as the weighted error of the mth boosting iteration with the mth base model h_m. AdaBoost (Freund and Schapire, 1997) updates the sample weight for the following base model by

$$D_{m+1}(n) = \frac{D_m(n)}{Z_m} \times \begin{cases} e^{-\alpha_m} & h_m(x_n) = y_n \\ e^{\alpha_m} & h_m(x_n) \neq y_n \end{cases}$$

where Z_m is a normalization factor such that D_{m+1} becomes a distribution, and $\alpha_m = \frac{1}{2}\ln(\frac{1-\epsilon_m}{\epsilon_m})$. Online AdaBoost (Wang and Pineau, 2015) algorithm use the normalization factor incrementally so that the weight of each new arriving instance can be appropriately updated for each base model. Oza et al. noted that the weight update step of AdaBoost can be reformulated as:

$$D_{m+1}(n) = D_m(n) \times \begin{cases} \frac{1}{2(1-\epsilon_m)} & h_m(x_n) = y_n \\ \frac{1}{2\epsilon_m} & h_m(x_n) \neq y_n \end{cases}$$

Online AdaBoost algorithm tracks the sum of correctly classified (λ^{SC}) and misclassified (λ^{SW}) instances respectively. The error of each base model can be approximated by $\epsilon = \frac{\lambda^{SW}}{\lambda^{SC}+\lambda^{SW}}$. When a new instance (x_n, y_n) arrives, its weight λ is set to 1 for the first base model. For the next iteration it is updated by $\lambda \leftarrow \frac{\lambda}{2(1-\epsilon)}$ if it is classified correctly, and $\lambda \leftarrow \frac{\lambda}{2(\epsilon)}$ otherwise. Each base model is repeatedly updated k times using (x_n, y_n),where $k \approx Poisson(\lambda)$.

In online TrAdaBoost (Wang and Pineau, 2015), consider the following $\lambda^{SC}_{T,m}$, $\lambda^{SW}_{T,m}$, $\lambda^{SC}_{S,m}$, $\lambda^{SW}_{S,m}$ use to track the sum of weights of correctly classified (SC) and misclassified (SW) samples from target domain (T) and source domain (S) by the mth base model during the online learning process. Here

$$\epsilon_{S,m} = \frac{\lambda^{SW}_{S,m}}{\lambda^{SC}_{T,m} + \lambda^{SW}_{T,m} + \lambda^{SC}_{S,m} + \lambda^{SW}_{S,m}},$$

$$\epsilon_{T,m} = \frac{\lambda^{SW}_{T,m}}{\lambda^{SC}_{T,m} + \lambda^{SW}_{T,m} + \lambda^{SC}_{S,m} + \lambda^{SW}_{S,m}},$$

and

$$D_{T,m} = \frac{\lambda^{SC}_{T,m} + \lambda^{SW}_{T,m}}{\lambda^{SC}_{T,m} + \lambda^{SW}_{T,m} + \lambda^{SC}_{S,m} + \lambda^{SW}_{S,m}}.$$

For the following base models, referring to the weight update step of batch TrAdaBoost, the weight can be updated by

$$\lambda = \begin{cases} \frac{\lambda}{1+D_{T,m}-(1-\beta)\epsilon_{S,m}-2\epsilon_{T,m}} & h_m(x_n) = y_n \\ \frac{\beta\lambda}{1+D_{T,m}-(1-\beta)\epsilon_{S,m}-2\epsilon_{T,m}} & h_m(x_n) \neq y_n \end{cases}$$

for a sample from source domain, and

$$\lambda = \begin{cases} \frac{\lambda}{1+D_{T,m}-(1-\beta)\epsilon_{S,m}-2\epsilon_{T,m}} & h_m(x_n) = y_n \\ \frac{\beta\lambda}{\epsilon_{T,m}(1+D_{T,m}-(1-\beta)\epsilon_{S,m}-2\epsilon_{T,m})} & h_m(x_n) \neq y_n \end{cases}$$

for a sample from target domain. Here $\beta = \frac{1}{1+\sqrt{2\ln|N_S|/M}}$ where N_S is the total number of samples from the source domain and M is the number of base models.

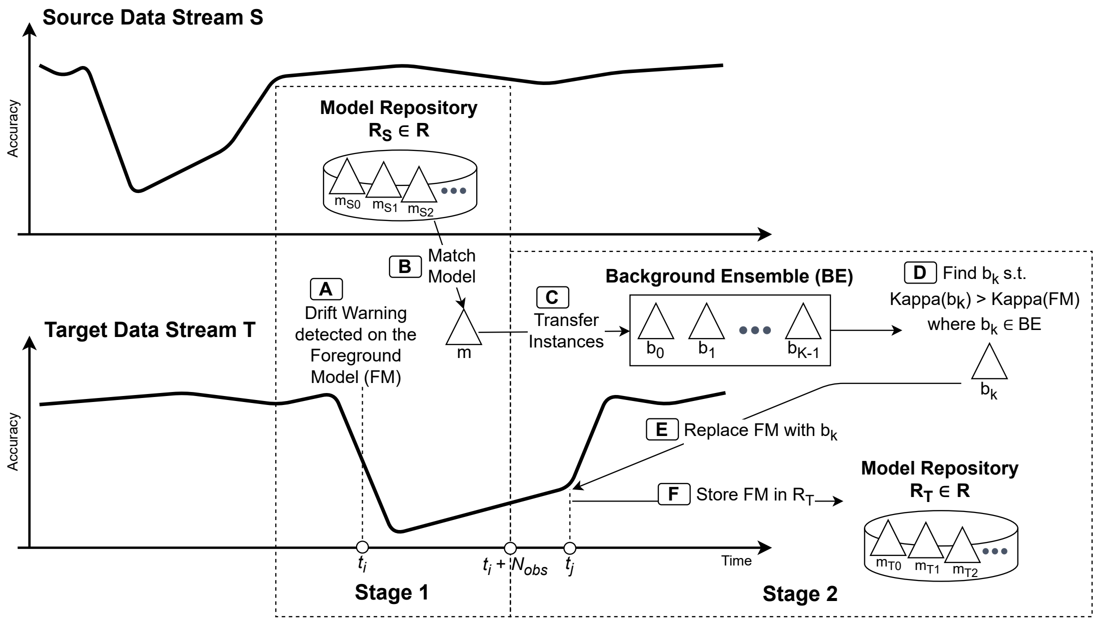

Figure 1: An overview of the primary transfer learning task in our proposed framework. The execution order of the main workflow is labeled alphabetically.

4. Our Transfer Learning Framework

In this section, we present our proposed transfer learning framework and the AOTrAdaBoost algorithm which is based on the online TrAdaBoost (Wang and Pineau, 2015). Our approach is able to transfer knowledge from a source data stream and reuse old concepts in the target data stream at the same time. This is achieved by using an online model and an instance transfer strategy. Both source and target data streams may suffer concept drift.

Our framework maintains a set of **model repository** $R = \{R_0, R_1, \ldots, R_{N-1}\}$ for each of the data streams $\mathcal{S} = \{S_0, S_1, \ldots, S_{N-1}\}$. Each repository in R stores all the models for each of the observed concepts in its corresponding data stream. In addition, each model caches a set of training instances that were used to build the model. For each data stream $i \in \mathcal{S}$, the framework maintains a corresponding **foreground model** for training and making predictions. The foreground model is equipped with two drift detectors, one for detecting drift warnings and the other for detecting actual drifts. When the drift warning is detected on the foreground model, the framework starts training both a **background ensemble** for the transfer learning task, and a **background model** to handle new concepts that have not been observed in $\mathcal{S}$. Fig. 1 provides an overview of the primary transfer learning workflow of the framework. Our proposed framework consists of two stages. Stage 1 prepares the system for transfer learning by observing the new distribution and then selects a model in R which is the closest match. Stage 2 carries out transfer learning by performing boosting on the instances provided by the matched model.

We initiate Stage 1 at either the very beginning of the target data stream (*i.e.*, at timestamp $t = 0$) or when a drift warning is detected on a foreground model at time t_i.

At this point we initialise a **background ensemble** of fixed size K to perform boosting on upcoming data instances in the target data stream. The background ensemble is a set of models training in the background that may be useful in the future, however these background models do not take part in the current prediction process. During Stage 1 (*i.e.* $[t_i, t_i + N_{obs}]$ where N_{obs} is a user-defined window size of instances called grace period), the background ensemble performs boosting on its models by increasing weights on wrongly predicted data instances, which is equivalent to AdaBoost or OzaBoost.

At the end of the grace period, we carry out a model matching process whereby we match a model from the model repositories R, using the instances observed in the target data stream during the grace period. There are many ways to perform model matching, such as comparing the distributions of the matched model's training instances and the observed instances in target data stream, clustering and RCD (Gonçalves Jr and De Barros, 2013). For simplicity, we perform matching based on model performance on data instances during the grace period. That is, the model with the highest kappa statistics on predicting the grace period instances is matched. The procedure of Stage 1 corresponds to Algorithm 1 in Lines 1 to 16.

Stage 2 uses the matched model's cached training instances to train the background ensemble using our boosting strategy. For each instance received from the target data stream, we train the background ensemble on both the received instance and N_{xfer} number of instances provided by the matched model. Precisely, at each time stamp $t \geq t_i + N_{obs}$, the background ensemble (1) performs boosting for the instance arriving at t in the target data stream, by increasing instance weight on correct predictions (*i.e.* AdaBoost or OzaBoost); (2) performs boosting for N_{xfer} instances provided by the matched model, by decreasing weights on wrong predictions. The details of this weighting mechanism is explained in Section 4.1. There are two benefits with this configuration. Firstly, this configuration does not block training/predicting in the target data stream, and secondly the data instances from the target data stream can enhance the background ensemble on desired distribution and thus improve the effectiveness of transfer learning continuously. This process is detailed in Algorithm 1 in Line 11 and Lines 18 to 21.

During Stage 2, we continuously monitor the performance of the models in the background ensemble by evaluating their prediction accuracy on each arriving instance in the target data stream. If a model in the background ensemble has obtained higher performance than the foreground model on the target instances by a threshold of κ, it then replaces the foreground model. The background ensemble is then removed, and the foreground model is added to the target stream's concept repository. In Algorithm 1, Lines 11 and Lines 18 to 29 describe the whole procedure for Stage 2.

In order to handle new concepts that have not been observed in any data streams, the framework also starts training a **background model** when the drift warning was detected, and then use the background model to replace the foreground model when the actual drift is detected. This is similar to the base models in the Adaptive Random Forest (Gomes et al., 2017). Another benefit of having a background model under our transfer learning framework is that it can act as a fail-safe measure for negative transfers. If the framework fails to match a correct model which produces inferior members in the background ensemble, the background model can replace the drifted foreground model instead.

Algorithm 1: The Transfer Learning Framework

```
Data: An instance $(x, y)$ from the target data stream $T$ arriving at time $t$
Input : Grace period $N_{obs}$, number of instances to transfer $N_{xfer}$, Size of Background Ensemble $K$,
        model replacement threshold $\kappa \in (0, 1)$, $\gamma$ for AOTrAdaBoost, model repositories for all data
        streams $R$, model repository for target data stream $R_T$
if $drift\ warning\ detected \vee t = 0$ then
    $BackgroundEnsemble \leftarrow \{b_0, b_1, \dots b_{K-1}\}$;
    Init $BackgroundModel$;
    $t_{start} \leftarrow t$;
end
if $actual\ drift\ detected$ then
    $R_T$.append($ForegroundModel$);
    $ForegroundModel \leftarrow BackgroundModel$;
end
if $BackgroundEnsemble \neq \emptyset$ then
    AOTrAdaBoost($BackgroundEnsemble$, $(x, y)$, IsFromTarget, 1);
    if $m = null$ then
        if $t = t_{start} + N_{obs}$ then
            Match model $m$ from $R$;
            $f \leftarrow tanh(\gamma \cdot Kappa(m))$;
        end
    else
        for $i \leftarrow [1, N_{xfer}]$ do
            Obtain an instance $(x_{src}, y_{src})$ from $m$'s cached training instances;
            AOTrAdaBoost($BackgroundEnsemble$, $(x_{src}, y_{src})$, IsFromSource, $f$);
        end
    end
    $CandidateModel \leftarrow$ a member model in BackgroundEnsemble with the highest Kappa statistics
      on instances from $T$;
    if $Kappa(CandidateModel) - Kappa(ForegroundModel) \geq \kappa$ then
        $R_T$.append($ForegroundModel$);
        $ForegroundModel \leftarrow CandidateModel$;
        $BackgroundEnsemble \leftarrow \emptyset$;
        $m \leftarrow null$;
    end
end
```

4.1. Adaptive Online TrAdaBoost (AOTrAdaBoost)

The base model receives a sample either from the source domain or the target domain. For inter-stream instances, we propose a weighted-based subspace alignment method by re-weighting the source samples based on how close they are to the target subspace. We can use a distance measure α to determine the closeness of the source space to the target space. We run a subsample of the instances from the target stream on the models in the source stream. In our framework, we used the kappa statistics to measure the model's performance. The intuition behind this approach is that if the data used to build the source model is similar to the target stream, then running the subsample of the instances from the target stream should yield high kappa statistics.

We use a weight assignment strategy for assigning larger weights to the source samples that are closer to the target domain. If the instances are from the target data stream, we boost the instances by adopting a similar strategy in TrAdaBoost. The weight adjustments for transferred instances during the boosting process is based on a weight factor f calculated as such:

$$f = \begin{cases} 1 & (x_n, y_n) \in T \\ \tanh(\gamma \cdot \alpha) & (x_n, y_n) \in S \end{cases}$$

where $\gamma \in (0, \inf)$ is a user-defined value and α is the matched model's performance (*i.e.* kappa statistics) on instances during grace period in target stream. S and T denotes the source and target data streams, respectively. Fig. 2 illustrates the effects of γ. The factor f is then applied on to λ during the weight update process:

$$\lambda = \begin{cases} f \cdot \frac{\lambda}{1+D_{T,m}-(1-\beta)\epsilon_{S,m}-2\epsilon_{T,m}} & h_m(x_n) = y_n \\ f \cdot \frac{\beta\lambda}{1+D_{T,m}-(1-\beta)\epsilon_{S,m}-2\epsilon_{T,m}} & h_m(x_n) \neq y_n \end{cases}$$

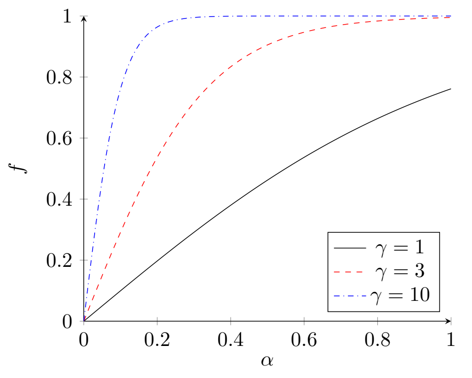

Figure 2: Effects of γ on weight adjustments during boosting.

The implication of the weight factor f is that, the boosted weights on data instances coming from the matched model, becomes proportional to the matched model's performance on the target data stream. This is equivalent to utilising the quality of the matched model to control the confidence of the boosting process. If the applied weight lowers the λ value, then even when a model in the background ensemble has made a correct prediction, the growth of λ value becomes more conservative. Essentially this factor reduces our transfer strategy to a partial model transfer, if the matched model has low performance on the target data stream. As a result, it may reduce the possibility of false positives, which is especially crucial under unstable environments such as the data streams.

4.2. Theoretical Analysis

4.2.1. Memory Complexity

The main memory consumers in the transfer learning framework are the model repositories across the data streams. Suppose the base model's memory complexity (including cached training instances) is Q, and we have N data streams with a total of N model repositories of size L each to transfer concepts across. The memory complexity of the framework is thus $\mathcal{O}(Q \cdot N \cdot L)$.

4.2.2. Time Complexity

Referring to Algorithm 1, two processes have main impacts on the time complexity, that is, the model matching process in Stage 1 and the AOTrAdaBoost process on transferred instances in Stage 2. Let the base model's time complexity for training be P, and the size of the background ensemble be K. AOTrAdaBoost is based on the instance-incremental version of TrAdaBoost (Wang and Pineau, 2015). The time complexity of this version of TrAdaBoost is based on the user-defined λ for online boosting. The weight adjustments of AOTrAdaBoost only impose constant time complexity during the boosting process. In addition, our transfer learning framework performs AOTrAdaBoost on N_{xfer} transferred instances per arriving instance in the target data stream. Therefore the total time complexity of AOTrAdaBoost is $\mathcal{O}(\lambda \cdot P \cdot K \cdot N_{xfer})$.

The time complexity of the model matching process in Stage 1 depends on the choice of matching strategy. Suppose the classification complexity of the base model is I and the average grace period is of length J, our simplified model matching technique iterates through all the models across the repository thus producing a total time complexity of $\mathcal{O}(I \cdot J \cdot N \cdot L)$. But there are other approaches to reduce the time complexity of this process as mentioned in Section 4.

5. Experimental Evaluation

In this section, we evaluate the performance of our transfer learning framework and AOTrAdaBoost. The classification performance is based on prequential evaluation, where each instance from the data stream is first used for testing, and then for training. The experiments were performed on the NVIDIA DGX-2 Station (Version 4.0.7, 40 cores, Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz, 252 GiB memory). Our code, synthetic dataset generators and test scripts are available online[1] for reproducible results.

Baseline Techniques. In the experiments we chose to use Hoeffding Tree as the base model. In order to evaluate the effectiveness of the different components in the transfer learning framework, we chose to compare against the following baselines: (1) HT: a variant of Hoeffding Tree where a background tree starts growing upon drift warning detection, and replace the foreground model upon actual concept drift detection. This is similar to the base model by Gomes et al. (2017); (2) w/o Boost: this baseline replaces the background ensemble in our transfer learning framework with a single model, to evaluate the effectiveness of using boosting for transfer learning in data streams; (3) OzaBoost (Oza and Russell, 2001); and (4) (instance-incremental) TrAdaBoost (Wang and Pineau, 2015). We plugged the baselines in our transfer learning framework in place of our AOTrAdaBoost algorithm. This design allows us to compare the benefits of the different boosting components.

We developed the above baselines on top of the Hoeffding Tree in StreamDM[2]. The default parameters of the base model are used for experimentations. We used grid search to tune the parameters in Algorithm 1 for every baselines on each of the datasets. The ranges of the parameters considered are in the format of start:step:end, inclusive: $N_{obs} = 100 : 100 : 400$, $N_{xfer} = 100 : 100 : 300$, $K = 10 : 10 : 50$, $\kappa = 0.0 : 0.1 : 0.4$, and $\gamma = 1 : 1 : 10$.

Evaluation Measures. We evaluate the performance of the techniques using standard performance measures for data streams (accuracy and kappa statistics (Bifet et al., 2013)) and runtime. We also use the cumulative accuracy gain (CAG) against HT as the baseline technique over the entire stream, specifically $\sum((\text{accuracy(B)} - \text{accuracy(HT)})$,where B is the model we are evaluating. The cumulative accuracy gain is calculated as follows:

1. https://github.com/ingako/AOTrAdaBoost
2. https://github.com/huawei-noah/streamDM-Cpp

$$\text{CAG} = \sum_{i=0}^{n}(M_B(x_i) - y_i) - \sum_{i=0}^{n}(M_{HT}((x_i) - y_i)$$

where $M_B(.)$ is the prediction from the model we are evaluating, and $M_{HT}(.)$ is the prediction from the HT model. The function above is performed over the lifetime of the data stream. The cumulative kappa gain (CKG) is calculated in the same way, with the kappa metric instead of accuracy. This measure is adapted from Wu et al. (2020). This performance measure tracks the working characteristics over the course of the stream, which is especially important for evaluating new solutions to dynamic data streams (Krawczyk et al., 2017). It also takes into account the performance decay caused by model overfitting at the drift points. We believe this is a better adapted measure to continuously monitor the utility of our transfer learning framework. The sample frequency parameter is set to 100 data instances.

Datasets. We use both synthetic and real-world datasets in our experiments. For generating synthetic datasets, we used the Agrawal generator (Agrawal et al., 1993) and the Random Tree generator (Domingos and Hulten, 2000). The Agrawal generator generates classification functions with different complexities. The Random Tree generator builds a decision tree by splitting on features and assigning class on leaves at random. The synthetic datasets include abrupt and gradual drifts. The abrupt and gradual drift widths are set to 1 and 500 data instances, respectively. The stable period is set to 8000 data instances and we generate a total of 15000 data instances for each of the data streams so that each dataset contains one concept drift. The real-world dataset is obtained from Du et al. (2020). The task is to classify whether rental bikes are in low or high demand at different times in different cities. We use the configuration of weekdays from Washington D.C as the source and weekends in London as the target, whereby both source and target datasets contain concept drifts.

5.1. Baseline Technique Comparisons on Synthetic Datasets

We study the two scenarios that our transfer learning framework with AOTrAdaBoost mainly address. In the first scenario transfer learning is conducted on data streams with different noise levels. In the second scenario transfer is conducted on streams with partial concept drifts, whereby only part of the concept changes but a portion of the relationship in the concept still holds. We also conducted a case study to demonstrate the effectiveness of our technique on long-running data streams with multiple concept drifts and different lengths of drift intervals (*i.e.* the intervals between every consecutive concept drifts).

We ran all experiments 10 times with varying seeds for every synthetic dataset configuration. We selected Functions 1 and 8 according to Agrawal et al. (1993) to generate concept drifts in the Agrawal datasets. The details of the functions are available in the supplementary materials (See footnote 1). With the Random Tree generator, we generate concept drifts by building decision trees with different maximum tree depths of 5 and 9. Two datasets are generated for source and target data streams for each of the experimentation. We only measure performance on the target data streams, since the results on the source data streams are the same.

We performed Friedman tests with a level of significance of 0.05 across all the experimentation results. The results showing statistically significant differences among all the baseline techniques are in bold in Tables 1 to 3. Overall, our framework with AOTrAdaBoost leads to higher accuracy, kappa statistics, cumulative accuracy gain and cumulative kappa gain compared with the baselines. In the following sections we provide details of the individual experiments.

Table 1: Results on Synthetic Datasets for Different Noise Levels

		Agrawal with Abrupt Drifts					Agrawal with Gradual Drifts				
	Benchmark	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)
0% - 0%	HT	92.28 ± 17.52	84.56 ± 35.05	-	-	0.59 ± 0.02	82.20 ± 30.43	64.40 ± 60.86	-	-	0.58 ± 0.01
	w/o Boost	95.68 ± 10.57	91.35 ± 21.15	509 ± 281	1019 ± 562	0.99 ± 0.14	89.89 ± 19.58	79.79 ± 39.17	1154 ± 1996	2308 ± 3992	1.03 ± 0.14
	OzaBoost	95.65 ± 10.64	91.29 ± 21.29	505 ± 260	1010 ± 520	2.39 ± 0.35	90.63 ± 14.89	81.25 ± 29.78	1264 ± 1646	2528 ± 3294	1.26 ± 0.18
	TrAdaBoost	95.98 ± 11.15	91.96 ± 22.30	555 ± 261	1111 ± 521	2.28 ± 0.29	94.10 ± 13.15	88.20 ± 26.28	1785 ± 1678	3571 ± 3358	1.87 ± 0.17
	AOTrAdaBoost	**96.09 ± 11.14**	**92.17 ± 22.28**	**571 ± 246**	**1142 ± 491**	2.63 ± 0.37	**94.34 ± 12.99**	**88.69 ± 25.97**	**1822 ± 1696**	**3644 ± 3394**	1.75 ± 0.30
0% - 10%	HT	84.56 ± 20.57	69.11 ± 41.14	-	-	0.58 ± 0.01	70.19 ± 36.42	40.37 ± 72.84	-	-	0.59 ± 0.02
	w/o Boost	90.09 ± 10.39	80.17 ± 20.78	830 ± 393	1659 ± 786	1.09 ± 0.10	85.88 ± 16.44	71.76 ± 32.87	2354 ± 2318	4708 ± 4636	1.05 ± 0.17
	OzaBoost	89.27 ± 10.65	78.53 ± 21.31	707 ± 505	1413 ± 1010	1.44 ± 0.23	87.61 ± 11.60	75.21 ± 23.21	2613 ± 2043	5226 ± 4086	2.64 ± 0.58
	TrAdaBoost	90.61 ± 10.70	81.22 ± 21.41	908 ± 389	1816 ± 779	2.19 ± 0.23	89.72 ± 11.68	79.45 ± 23.35	2931 ± 2084	5862 ± 4169	2.51 ± 0.41
	AOTrAdaBoost	90.61 ± 10.70	81.22 ± 21.41	908 ± 389	1816 ± 779	2.64 ± 0.25	**89.83 ± 11.44**	**79.66 ± 22.88**	**2946 ± 2100**	**5893 ± 4201**	2.60 ± 0.47
10% - 0%	HT	92.28 ± 17.52	84.56 ± 35.05	-	-	0.59 ± 0.03	82.20 ± 30.43	64.40 ± 60.86	-	-	0.58 ± 0.01
	w/o Boost	94.67 ± 10.92	89.34 ± 21.84	359 ± 247	718 ± 494	1.00 ± 0.22	89.18 ± 19.98	78.36 ± 39.96	1048 ± 1996	2095 ± 3993	1.03 ± 0.21
	OzaBoost	93.12 ± 12.91	86.24 ± 25.83	126 ± 303	253 ± 606	1.52 ± 0.30	90.79 ± 17.31	81.59 ± 34.61	1289 ± 2127	2578 ± 4255	1.95 ± 0.52
	TrAdaBoost	94.72 ± 10.78	89.44 ± 21.56	366 ± 261	732 ± 523	2.37 ± 0.09	91.53 ± 16.23	83.06 ± 32.45	1400 ± 2029	2799 ± 4059	1.76 ± 0.42
	AOTrAdaBoost	**95.06 ± 10.82**	**90.11 ± 21.64**	**416 ± 226**	**833 ± 451**	2.69 ± 0.15	**92.63 ± 15.80**	**85.26 ± 31.59**	**1565 ± 1972**	**3129 ± 3944**	1.78 ± 0.23
10% - 20%	HT	75.47 ± 24.24	50.94 ± 48.48	-	-	0.58 ± 0.02	69.40 ± 32.01	38.80 ± 64.03	-	-	0.58 ± 0.01
	w/o Boost	84.78 ± 11.61	69.57 ± 23.22	1397 ± 697	2795 ± 1394	1.07 ± 0.17	80.14 ± 18.43	60.27 ± 36.86	1610 ± 2049	3221 ± 4099	0.99 ± 0.18
	OzaBoost	84.12 ± 11.70	68.24 ± 23.40	1298 ± 745	2596 ± 1490	2.10 ± 0.29	79.66 ± 15.32	59.33 ± 30.63	1539 ± 1967	3080 ± 3934	1.04 ± 0.22
	TrAdaBoost	85.17 ± 11.92	70.34 ± 23.85	1455 ± 685	2910 ± 1370	1.90 ± 0.20	83.40 ± 13.49	66.81 ± 26.98	2100 ± 2001	4202 ± 4003	1.86 ± 0.24
	AOTrAdaBoost	**85.21 ± 11.88**	**70.41 ± 23.77**	**1460 ± 671**	**2921 ± 1343**	2.04 ± 0.18	**84.59 ± 11.56**	**69.18 ± 23.11**	**2278 ± 2042**	**4557 ± 4084**	2.17 ± 0.24
20% - 10%	HT	84.56 ± 20.57	69.11 ± 41.14	-	-	0.66 ± 0.01	70.19 ± 36.42	40.37 ± 72.84	-	-	0.58 ± 0.01
	w/o Boost	89.33 ± 11.29	78.66 ± 22.58	716 ± 497	1433 ± 994	1.09 ± 0.25	84.48 ± 20.18	68.96 ± 40.36	2144 ± 2353	4288 ± 4706	0.93 ± 0.19
	OzaBoost	89.75 ± 11.13	79.51 ± 22.26	779 ± 386	1559 ± 772	1.69 ± 0.34	86.28 ± 16.38	72.56 ± 32.75	2414 ± 2214	4828 ± 4429	1.39 ± 0.36
	TrAdaBoost	90.41 ± 10.96	80.81 ± 21.93	877 ± 437	1755 ± 874	1.50 ± 0.29	87.01 ± 15.98	74.03 ± 31.95	2524 ± 2174	5049 ± 4348	1.52 ± 0.24
	AOTrAdaBoost	**90.90 ± 10.76**	**81.80 ± 21.53**	**951 ± 401**	**1903 ± 802**	1.42 ± 0.21	**87.87 ± 15.72**	**75.74 ± 31.43**	**2653 ± 2193**	**5306 ± 4387**	1.36 ± 0.39

		Random Tree with Abrupt Drifts					Random Tree with Gradual Drifts				
	Benchmark	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)
0% - 0%	HT	57.96 ± 7.42	7.56 ± 13.43	-	-	0.75 ± 0.01	57.97 ± 7.17	7.58 ± 12.95	-	-	0.76 ± 0.01
	w/o Boost	63.74 ± 10.64	22.76 ± 17.63	867 ± 2	2280 ± 7	1.33 ± 0.33	63.52 ± 10.61	22.29 ± 17.52	832 ± 23	2207 ± 39	1.53 ± 0.26
	OzaBoost	59.19 ± 9.01	11.19 ± 12.60	185 ± 21	544 ± 36	3.33 ± 1.53	60.64 ± 8.91	14.69 ± 14.31	400 ± 63	1066 ± 143	3.03 ± 0.48
	TrAdaBoost	63.04 ± 10.10	20.52 ± 16.45	763 ± 12	1943 ± 33	3.81 ± 0.37	60.71 ± 8.41	12.45 ± 12.40	411 ± 56	730 ± 121	2.47 ± 0.26
	AOTrAdaBoost	**65.44 ± 10.69**	**26.60 ± 18.48**	**1123 ± 87**	**2856 ± 184**	3.80 ± 0.47	**64.35 ± 10.77**	**23.18 ± 17.81**	**956 ± 49**	**2339 ± 95**	2.40 ± 0.26
0% - 10%	HT	55.59 ± 5.90	4.12 ± 9.13	-	-	0.76 ± 0.01	55.58 ± 5.94	4.22 ± 9.07	-	-	0.76 ± 0.01
	w/o Boost	58.93 ± 9.97	14.54 ± 16.69	501 ± 5	1563 ± 5	1.89 ± 0.21	60.53 ± 8.52	17.79 ± 14.23	743 ± 27	2036 ± 45	1.53 ± 0.11
	OzaBoost	55.45 ± 6.36	4.22 ± 9.79	−21 ± 3	14 ± 24	1.64 ± 0.28	55.70 ± 5.98	5.68 ± 10.44	18 ± 32	219 ± 27	1.57 ± 0.23
	TrAdaBoost	59.67 ± 8.61	16.61 ± 14.72	611 ± 12	1873 ± 47	3.99 ± 0.51	60.90 ± 8.73	17.80 ± 14.62	797 ± 70	2037 ± 116	2.89 ± 0.25
	AOTrAdaBoost	**61.97 ± 8.36**	**19.59 ± 14.29**	**956 ± 21**	**2321 ± 41**	4.12 ± 0.19	**61.51 ± 8.84**	**18.77 ± 14.68**	**889 ± 99**	**2183 ± 186**	3.19 ± 0.18
10% - 0%	HT	57.96 ± 7.42	7.56 ± 13.43	-	-	0.77 ± 0.02	57.97 ± 7.17	7.58 ± 12.95	-	-	0.76 ± 0.01
	w/o Boost	62.44 ± 10.49	19.44 ± 16.96	672 ± 4	1781 ± 4	1.58 ± 0.24	63.05 ± 9.92	20.52 ± 16.11	762 ± 91	1941 ± 183	1.51 ± 0.16
	OzaBoost	60.33 ± 8.34	16.37 ± 13.94	356 ± 54	1321 ± 111	2.92 ± 0.33	57.67 ± 7.39	11.33 ± 12.12	−45 ± 37	562 ± 234	3.37 ± 0.28
	TrAdaBoost	62.83 ± 9.37	18.91 ± 14.56	731 ± 2	1702 ± 5	2.07 ± 0.30	60.50 ± 8.40	14.91 ± 12.53	378 ± 65	1099 ± 163	2.35 ± 0.38
	AOTrAdaBoost	**64.50 ± 9.78**	**23.75 ± 16.16**	**981 ± 121**	**2428 ± 194**	1.95 ± 0.38	**63.48 ± 10.47**	**22.05 ± 17.22**	**826 ± 67**	**2169 ± 121**	2.30 ± 0.44
10% - 20%	HT	53.96 ± 5.73	1.71 ± 6.98	-	-	0.76 ± 0.01	53.96 ± 5.77	1.56 ± 6.90	-	-	0.76 ± 0.01
	w/o Boost	56.72 ± 7.62	11.26 ± 13.98	414 ± 5	1433 ± 12	1.49 ± 0.17	57.29 ± 7.13	12.05 ± 13.07	500 ± 65	1573 ± 136	1.59 ± 0.10
	OzaBoost	54.29 ± 5.73	0.87 ± 5.50	50 ± 25	−127 ± 56	2.57 ± 0.94	54.49 ± 5.93	5.54 ± 9.87	79 ± 34	596 ± 74	3.99 ± 0.69
	TrAdaBoost	55.68 ± 6.04	6.15 ± 8.39	257 ± 8	667 ± 10	2.91 ± 0.25	56.67 ± 6.55	11.13 ± 12.06	407 ± 35	1436 ± 117	2.44 ± 0.40
	AOTrAdaBoost	**57.94 ± 6.84**	**13.23 ± 12.37**	**596 ± 50**	**1728 ± 45**	2.81 ± 0.29	**57.58 ± 7.08**	**12.78 ± 12.75**	**544 ± 49**	**1682 ± 78**	2.48 ± 0.21
20% - 10%	HT	55.59 ± 5.90	4.12 ± 9.13	-	-	0.76 ± 0.01	55.58 ± 5.94	4.22 ± 9.07	-	-	0.76 ± 0.02
	w/o Boost	58.97 ± 7.35	13.59 ± 12.85	507 ± 17	1420 ± 60	1.69 ± 0.10	58.69 ± 7.30	12.74 ± 12.11	467 ± 58	1278 ± 117	1.62 ± 0.18
	OzaBoost	55.73 ± 6.10	4.46 ± 9.66	20 ± 14	51 ± 16	1.52 ± 0.26	55.56 ± 6.10	4.43 ± 9.05	−3 ± 28	31 ± 48	1.55 ± 0.17
	TrAdaBoost	57.58 ± 6.94	12.34 ± 11.93	299 ± 18	1232 ± 30	2.53 ± 0.34	57.08 ± 7.65	10.92 ± 13.19	225 ± 109	1005 ± 187	2.47 ± 0.30
	AOTrAdaBoost	**60.44 ± 8.07**	**16.30 ± 13.60**	**728 ± 45**	**1827 ± 69**	2.56 ± 0.07	**59.90 ± 8.28**	**15.42 ± 14.14**	**648 ± 73**	**1679 ± 138**	2.56 ± 0.20

NOTE: The left-most column are the specific noise settings in the format of (noise percentage in source data stream) - (noise percentage in target data stream). Statistically significant results for Acc./Kappa and CAG/CKG are in bold.

5.1.1. Transfer under Different Noise Levels

We generate three different noise level settings, $0\%, 10\%, 20\%$, on the datasets to evaluate the effects of noise on our techniques. In particular, we study the following settings: (1) transfer between data streams without any noise; (2) transfer between noise-free and noisy data streams and vice versa; (3) transfer between data streams with noise in different levels. The left-most column in Table 1 lists the specific noise settings in the format of (noise percentage in source data stream) - (noise percentage in target data stream). For example, the noise setting of 10% - 0%, indicates that there is a 10% noise in the source data stream and 0% noise in the target stream. The results in Table 1 show that our framework with AOTrAdaBoost is the most robust technique against noise compared to other baselines. One exception is in the Agrawal datasets with gradual drifts and a noise setting of 10% - 0%, where the cumulative gains of the baselines are the most unstable and the standard deviations are higher than the means. This is because gradual drifts exhibit similar properties to noise, therefore gradual drifts with noise greatly increase the actual noise in total. As a result, when transferring from a very noisy environment to noise-free environment, boosting may cause negative transfers. In that case, it may be better to use the background model instead of the boosted models from the background ensemble. A similar observation was made on the Random Tree datasets with the same configuration, where the standard deviations on cumulative gains are higher compared to other Random Tree dataset configurations. However, our framework is still able to obtain much higher positive gains on this configuration. Potentially, this is due to the different complexities of the concept drifts between the two data generators. Another observation is that our transfer learning framework with AOTrAdaBoost presents more gain on the scenarios in which both source and target data streams are noisy, especially when the source contains more noise than the target. More noise in the source data stream means the matched model tend to have lower performance. Therefore AOTrAdaBoost is able to adjust the boosting process to reduce the effect of noise (*i.e.* false positives).

5.1.2. Partial Concept Transfer

Transferring from a complex concept to a simpler concept exhibit similar properties to transferring from a noisy environment to a less noisy one. Therefore for partial concept transfer we evaluate a specific scenario where a simpler concept is transferred to a more complex one. For the Agrawal generator, we added two new functions based on Functions 1 and 8 by adding a logical disjunction of the proposition (car maker ≤ 2) for classifying Group A. Function 1 and 8 were used to generate concept drifts in the source data stream, and the new functions were used in the target data stream, since the new functions are more complex due to the added propositions. For the Random Tree generator, we generate trees with more depth 6 and 10. These configurations are used for generating the datasets acting as the target data streams. Table 2 shows the effectiveness of our transfer learning framework with AOTrAdaBoost on transferring partial concepts.

5.1.3. Case Study

We also performed a case study with the Random Tree generator to produce a larger dataset with 150000 of instances. Concept drifts are generated based on a random tree

Table 2: Results on Synthetic Datasets for Partial Concept Transfer

	Agrawal with Abrupt Drifts					Agrawal with Gradual Drifts				
Benchmark	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)
HT	71.26 ± 35.66	42.52 ± 71.33	-	-	0.57 ± 0.01	68.78 ± 36.35	37.56 ± 72.69	-	-	0.58 ± 0.02
w/o Boost	92.76 ± 10.98	85.51 ± 21.97	3225 ± 600	6449 ± 1199	0.96 ± 0.24	91.16 ± 11.44	82.33 ± 22.88	3358 ± 527	6716 ± 1053	1.01 ± 0.20
OzaBoost	92.15 ± 12.12	84.31 ± 24.23	3134 ± 433	6268 ± 867	1.15 ± 0.17	90.01 ± 13.88	80.01 ± 27.77	3184 ± 670	6368 ± 1340	1.49 ± 0.33
TrAdaBoost	93.89 ± 10.66	87.78 ± 21.32	3394 ± 490	6789 ± 980	2.07 ± 0.26	92.44 ± 11.46	84.89 ± 22.92	3550 ± 678	7099 ± 1355	2.21 ± 0.23
AOTrAdaBoost	$\mathbf{94.26 \pm 10.44}$	$\mathbf{88.52 \pm 20.87}$	$\mathbf{3450 \pm 535}$	$\mathbf{6901 \pm 1069}$	2.27 ± 0.31	$\mathbf{93.33 \pm 10.79}$	$\mathbf{86.66 \pm 21.58}$	$\mathbf{3683 \pm 529}$	$\mathbf{7365 \pm 1057}$	2.34 ± 0.46

	Random Tree with Abrupt Drifts					Random Tree with Gradual Drifts				
Benchmark	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)
HT	57.96 ± 7.42	7.56 ± 13.43	-	-	$\mathbf{0.76 \pm 0.01}$	57.97 ± 7.17	7.58 ± 12.95	-	-	$\mathbf{0.76 \pm 0.01}$
w/o Boost	63.74 ± 10.64	22.76 ± 17.63	867 ± 2	2280 ± 7	1.44 ± 0.25	62.76 ± 10.25	21.28 ± 17.10	718 ± 10	2055 ± 25	1.46 ± 0.23
OzaBoost	59.19 ± 9.01	11.19 ± 12.60	185 ± 21	544 ± 36	3.21 ± 1.77	58.93 ± 9.10	10.54 ± 12.91	143 ± 28	443 ± 43	1.73 ± 0.22
TrAdaBoost	63.04 ± 10.10	20.52 ± 16.45	763 ± 12	1943 ± 33	3.51 ± 0.44	62.97 ± 10.07	19.99 ± 16.27	749 ± 68	1861 ± 232	3.26 ± 0.38
AOTrAdaBoost	$\mathbf{65.44 \pm 10.69}$	$\mathbf{26.60 \pm 18.48}$	$\mathbf{1123 \pm 87}$	$\mathbf{2856 \pm 184}$	3.88 ± 0.61	$\mathbf{64.42 \pm 11.01}$	$\mathbf{23.73 \pm 19.17}$	$\mathbf{968 \pm 127}$	$\mathbf{2422 \pm 257}$	3.87 ± 0.46

depths sequence of $[2, 4, 6, 8, 10, 12]$, with random drift intervals generated between $1000 + 200 \cdot D[0, 20) + \text{uniform}[0, 100)$, where D is a Poisson distribution $P(X = x) = \frac{\lambda^x e^{-\lambda}}{x!}$ and $\lambda = 3$. Both source and target data streams contain no noise. While the AOTrAdaBoost configuration has more advantages over other baselines when the source and target data streams are more dissimilar, it is still able to obtain higher performance continuously as demonstrated in Table 3.

Table 3: Results for Case Study with 0% Noise in Both Source and Target Stream

Benchmark	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)
HT	59.95 ± 8.62	13.75 ± 15.12	-	-	14.47 ± 0.38
w/o Boost	61.45 ± 9.00	17.96 ± 16.10	2247 ± 1065	6323 ± 3210	13.55 ± 1.70
OzaBoost	60.08 ± 8.72	13.97 ± 15.08	197 ± 611	334 ± 2118	19.33 ± 2.87
TrAdaBoost	61.00 ± 9.39	17.37 ± 16.64	1579 ± 2109	5430 ± 4623	22.96 ± 4.14
AOTrAdaBoost	$\mathbf{61.77 \pm 9.25}$	$\mathbf{18.87 \pm 16.55}$	$\mathbf{2724 \pm 1155}$	$\mathbf{7684 \pm 3371}$	22.33 ± 4.56

5.2. Parameter Sensitivity Analysis

Referring to Algorithm 1, five parameters may affect the performance of our technique. Table 4 shows the impact of the parameters on the Agrawal datasets with abrupt and gradual drifts and a noise setting of 20% - 10%.

According to Table 4, on the datasets with abrupt drifts, higher performances can be achieved with lower N_{obs} values compared to datasets with gradual drifts. For N_{xfer}, both abrupt and gradual drifts prefer higher values, but for the gradual drifts this parameter needs to be fine-tuned. This is because gradual drifts exhibit similar properties to noisy data, and a higher N_{xfer} value may lead to overfitting before the end of the gradual drifts. K needs to be fine-tuned even though a larger size may create more diversity in the background ensemble. This is because having more boosting members can potentially lead to higher instance weights, causing members near the end of the boosting queue to overfit. Higher κ may lead to models in the background ensemble having less probability to swap with the drifted foreground model. Therefore, higher κ may reduce false positives, but for gradual drifts a lower κ value may be preferred since the data is noisy. As for γ, a lower value may be preferred when transferring from noisy environments, since lower γ can reduce AOTrAdaBoost to partial model transfer to avoid false positives.

Table 4: Parameter Sensitivity Evaluation

	Abrupt Drift					Gradual Drift				
N_{obs}	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)
100	90.90 ± 10.76	81.80 ± 21.53	951 ± 401	1903 ± 802	1.42 ± 1.42	84.96 ± 15.44	69.91 ± 30.87	2215 ± 2084	4431 ± 4168	2.02 ± 2.02
200	88.46 ± 14.38	76.92 ± 28.77	586 ± 476	1172 ± 953	1.17 ± 1.17	85.44 ± 17.28	70.89 ± 34.55	2288 ± 1937	4577 ± 3874	1.93 ± 1.93
300	88.15 ± 15.36	76.31 ± 30.72	540 ± 542	1079 ± 1083	1.33 ± 1.33	84.35 ± 19.08	68.71 ± 38.16	2125 ± 2010	4250 ± 4021	1.83 ± 1.83
400	87.39 ± 16.10	74.78 ± 32.21	425 ± 485	850 ± 969	1.26 ± 1.26	87.87 ± 15.72	75.74 ± 31.43	2653 ± 2193	5306 ± 4387	1.36 ± 1.36
N_{xfer}	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)
100	89.24 ± 11.52	78.47 ± 23.04	702 ± 494	1404 ± 988	1.23 ± 1.23	86.28 ± 16.08	72.56 ± 32.15	2414 ± 2164	4829 ± 4329	1.47 ± 1.47
200	89.63 ± 11.35	79.25 ± 22.71	760 ± 479	1521 ± 958	1.29 ± 1.29	87.87 ± 15.72	75.74 ± 31.43	2653 ± 2193	5306 ± 4387	1.36 ± 1.36
300	90.90 ± 10.76	81.80 ± 21.53	951 ± 401	1903 ± 802	1.42 ± 1.42	86.80 ± 15.94	73.61 ± 31.87	2493 ± 2220	4986 ± 4441	1.56 ± 1.56
K	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)
0.0	86.89 ± 15.70	73.77 ± 31.39	350 ± 471	699 ± 942	1.05 ± 1.05	85.80 ± 16.99	71.60 ± 33.99	2342 ± 2197	4685 ± 4394	1.25 ± 1.25
0.1	88.43 ± 12.11	76.86 ± 24.22	581 ± 465	1162 ± 931	1.08 ± 1.08	86.78 ± 15.88	73.56 ± 31.76	2489 ± 2191	4978 ± 4382	1.34 ± 1.34
0.2	90.21 ± 11.01	80.41 ± 22.03	848 ± 480	1695 ± 959	1.31 ± 1.31	87.87 ± 15.72	75.74 ± 31.43	2653 ± 2193	5306 ± 4387	1.36 ± 1.36
0.3	90.90 ± 10.76	81.80 ± 21.53	951 ± 401	1903 ± 802	1.42 ± 1.42	87.75 ± 15.67	75.50 ± 31.32	2634 ± 2220	5269 ± 4441	1.50 ± 1.50
0.4	90.74 ± 11.09	81.48 ± 22.18	928 ± 387	1856 ± 774	1.56 ± 1.56	87.40 ± 15.85	74.81 ± 31.69	2582 ± 2170	5166 ± 4340	1.62 ± 1.62
κ	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)
10	90.90 ± 10.76	81.80 ± 21.53	951 ± 401	1903 ± 802	1.42 ± 1.42	87.22 ± 15.61	74.44 ± 31.21	2555 ± 2111	5110 ± 4223	1.24 ± 1.24
20	90.90 ± 10.74	81.79 ± 21.48	951 ± 421	1902 ± 842	1.77 ± 1.77	86.28 ± 16.38	72.56 ± 32.75	2414 ± 1971	4828 ± 3943	1.32 ± 1.32
30	90.63 ± 10.86	81.25 ± 21.72	910 ± 451	1821 ± 902	2.03 ± 2.03	87.87 ± 15.72	75.74 ± 31.43	2653 ± 2193	5306 ± 4387	1.36 ± 1.36
40	89.76 ± 11.34	79.52 ± 22.68	781 ± 369	1562 ± 738	2.30 ± 2.30	86.12 ± 16.96	72.24 ± 33.92	2389 ± 2279	4779 ± 4560	1.82 ± 1.82
50	90.19 ± 11.01	80.38 ± 22.02	845 ± 337	1690 ± 673	2.54 ± 2.54	85.65 ± 17.47	71.31 ± 34.93	2320 ± 1944	4641 ± 3890	2.12 ± 2.12
γ	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)
2.0	90.26 ± 10.95	80.51 ± 21.91	855 ± 396	1710 ± 791	1.38 ± 1.38	87.09 ± 15.89	74.18 ± 31.77	2535 ± 2174	5071 ± 4348	1.37 ± 1.37
4.0	90.90 ± 10.76	81.80 ± 21.53	951 ± 401	1903 ± 802	1.42 ± 1.42	86.58 ± 16.07	73.15 ± 32.14	2458 ± 2219	4917 ± 4438	1.61 ± 1.61
6.0	90.79 ± 10.80	81.58 ± 21.61	935 ± 426	1870 ± 853	1.36 ± 1.36	87.87 ± 15.72	75.74 ± 31.43	2653 ± 2193	5306 ± 4387	1.36 ± 1.36
8.0	90.12 ± 11.10	80.24 ± 22.20	835 ± 385	1670 ± 770	1.44 ± 1.44	86.57 ± 16.03	73.14 ± 32.04	2457 ± 2169	4915 ± 4339	1.49 ± 1.49

5.3. Experiments on Real World Dataset

Table 5 shows our framework with AOTrAdaBoost is able to obtain the highest accuracy gains compared to other baselines. Fig. 3 shows that the AOTradaBoost configuration is more stable than the other baselines throughout the course of the stream. We notice that at the beginning AOTradaBoost and HT w/o Boost were the top two performers, while later AOTrAdaBoost and TrAdaBoost were the top two performers. We observe that our technique adapts to this stream as it evolves and manages to keep a more stable and higher accuracy.

Table 5: Results on Real-World Dataset - Bike

Benchmark	Acc. (%)	Kappa (%)	CAG (%)	CKG (%)	Runtime (sec)
HT	72.49 ± 12.02	29.10 ± 29.66	-	-	0.39
w/o Boost	72.02 ± 13.01	29.90 ± 28.94	-23	39	0.31
OzaBoost	70.96 ± 10.42	25.98 ± 25.22	-75	-153	0.53
TrAdaBoost	76.22 ± 8.02	38.15 ± 26.81	183	443	1.39
AOTrAdaBoost	77.37 ± 7.27	42.28 ± 24.59	239	646	1.54

6. Conclusion

We presented a novel transfer learning framework for evolving data streams. In particular, we proposed a novel AOTrAdaBoost technique that tunes the sensitivity of weighting during the boosting process, such that our framework is more robust against noisy source domains

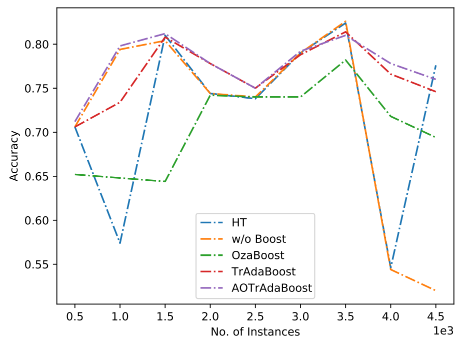

Figure 3: Accuracy results for the bike dataset (target domain).

and benefit from source domains containing partially similar concepts. We empirically evaluated our technique on both synthetic and real-world datasets. Our AOTrAdaBoost has shown to obtain statistically significant performance gains over baseline techniques.

In future work, we will investigate the problem of negative transfers in noisy to noise-free environments. We plan to adaptively swap between the transfer learning framework and the background model. We will also adapt the user-defined parameters dynamically, based on the characteristics of the data streams such as stream volatility.

Acknowledgments

This work was funded in part by the Office of Naval Research Global grant (N62909-19-1-2042).

References

R. Agrawal, T. Imielinski, and A. Swami. Database mining: a performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, Dec 1993.

Albert Bifet, Jesse Read, Indrė Žliobaitė, Bernhard Pfahringer, and Geoff Holmes. Pitfalls in benchmarking data stream classification and how to avoid them. In *Machine Learning and Knowledge Discovery in Databases*, pages 465–479, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-40988-2.

Wenyuan Dai, Qiang Yang, Gui-Rong Xue, and Yong Yu. Boosting for transfer learning. In *ICML*, page 193–200, New York, NY, USA, 2007.

Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *SIGKDD*, page 71–80, New York, NY, USA, 2000.

H. Du, L. L. Minku, and H. Zhou. Multi-source transfer learning for non-stationary environments. In *IJCNN*, pages 1–8, 2019.

H. Du, L. L. Minku, and H. Zhou. Marline: Multi-source mapping transfer learning for non-stationary environments. In *ICDM*, pages 122–131, 2020.

Eric Eaton and Marie desJardins. Selective transfer between learning tasks using task-based boosting. *AAAI*, 25(1), 2011.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. ISSN 0022-0000.

João Gama and Petr Kosina. Recurrent concepts in data streams classification. *Knowledge and Information Systems*, 40(3):489–507, 2014.

Heitor M Gomes, Albert Bifet, Jesse Read, Jean Paul Barddal, Fabrício Enembreck, Bernhard Pfharinger, Geoff Holmes, and Talel Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9-10):1469–1495, 2017.

Paulo Mauricio Gonçalves Jr and Roberto Souto Maior De Barros. Rcd: A recurring concept drift framework. *Pattern Recognition Letters*, 34(9):1018–1025, 2013.

Bartosz Krawczyk, Leandro L. Minku, João Gama, Jerzy Stefanowski, and Michał Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132 – 156, 2017.

Helen Mckay, Nathan Griffiths, Phillip Taylor, Theo Damoulas, and Zhou Xu. Bi-directional online transfer learning: a framework. *Annals of Telecommunications*, 75(9-10):523–547, 2020.

Leandro L. Minku. *Transfer Learning in Non-stationary Environments*, pages 13–37. Springer International Publishing, Cham, 2019.

Nikunj Oza and Stuart Russell. Online bagging and boosting. *Proceedings of Artificial Intelligence and Statistics*, 2001.

Boyu Wang and Joelle Pineau. Online boosting algorithms for anytime transfer and multitask learning. In *AAAI*, page 3038–3044, 2015.

Yimin Wen, Yixiu Qin, Keke Qin, Xiaoxia Lu, and Pingshan Liu. Online transfer learning with multiple decision trees. *IJMLC*, 10(10):2941–2962, 2019.

Ocean Wu, Yun Sing Koh, Gillian Dobbie, and Thomas Lacombe. Pearl: Probabilistic exact adaptive random forest with lossy counting for data streams. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 17–30. Springer, 2020.

Q. Wu, H. Wu, X. Zhou, M. Tan, Y. Xu, Y. Yan, and T. Hao. Online transfer learning with multiple homogeneous or heterogeneous sources. *IEEE Transactions on Knowledge and Data Engineering*, 29(7):1494–1507, 2017.

C. Yang, Y. M. Cheung, J. Ding, and K. C. Tan. Concept drift-tolerant transfer learning in dynamic environments. *TNNLS*, pages 1–15, 2021.

Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.