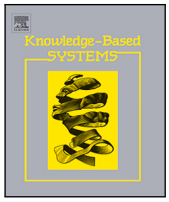
Knowledge-Based Systems 293 (2024) 111610

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys



Cost-aware retraining for machine learning

Ananth Mahadevan *, Michael Mathioudakis

Department of Computer Science, Helsinki, Finland

ARTICLE INFO

Keywords:
Machine learning
Resource-aware computing

ABSTRACT

Retraining a machine learning (ML) model is essential for maintaining its performance as the data change over time. However, retraining is also costly, as it typically requires re-processing the entire dataset. As a result, a trade-off arises: on the one hand, retraining an ML model too frequently incurs unnecessary computing costs; on the other hand, not retraining frequently enough leads to stale ML models and incurs a cost in loss of accuracy. To resolve this trade-off, we envision ML systems that make automated and cost-optimal decisions about when to retrain an ML model.

In this work, we study the decision problem of whether to retrain or keep an existing ML model based on the data, the model, and the predictive queries answered by the model. Crucially, we consider the costs associated with each decision and aim to optimize the trade-off. Our main contribution is a Cost-Aware Retraining Algorithm, Cara, which optimizes the trade-off over streams of data and queries. To explore the performance of Cara, we first analyze synthetic datasets and demonstrate that Cara can adapt to different data drifts and retraining costs while performing similarly to an optimal retrospective algorithm. Subsequently, we experiment with real-world datasets and demonstrate that Cara has better accuracy than drift detection baselines while making fewer retraining decisions, thus incurring lower total costs.

1. Introduction

Retraining a machine learning (ML) model is essential in the presence of *data drift* [1], i.e., continuous changes in the data due to factors such as system modifications, seasonality, or changes in user preferences. As a data drift occurs, the performance of an ML model trained on old data typically decreases — or, more generally, is not as high as it could be if it took advantage of the new data. As a result, maintaining the performance of an ML model calls for an online decision on whether to `Retrain` or `Keep` the existing ML model. In what follows, we refer to any algorithm that makes such a decision as a *retraining algorithm*.

Several methods exist to detect data drift [2–4] by monitoring the errors of an ML model on a stream of labeled samples, i.e., *data*. These methods decide to retrain the ML model whenever a drift is detected. However, note that (i) ML models are typically used to predict a stream of unlabeled samples [5], i.e., *queries*, and (ii) (re)training is associated with *costs*, such as the monetary cost for (re)training a model on a cloud service or the energy cost for executing the training algorithm [6]. Considering queries and costs in addition to data is essential in deciding whether to retrain a model. To see why, consider a real estate firm that uses an ML model to predict housing prices. Suppose, the data used to train the ML model represents the price distribution of all houses in the market. However, the customer queries concern only a niche of this market, for example, mansions. In such a case, if new data suggests prices have changed for studio apartments but not for mansions, retraining the model will not improve its performance on queries significantly enough to justify the retraining cost. By contrast, if the new data indicates that prices for mansions have changed significantly, then the potential drop in performance on queries will warrant a `Retrain` decision. Unfortunately, many existing drift detection methods do not consider queries or retraining costs, which may lead to suboptimal decisions for scenarios like the above.

The above discussion gives rise to two costs when deciding whether to retrain. The first is a *model staleness cost*, i.e., the performance loss due to keeping an ML model trained on old data. Typically, when a data drift occurs, a model trained on old data will have lower performance compared to an ML model trained on fresher data. Note that while we use the notion of "cost" in a general rather than strictly monetary sense, a performance loss often does translate into a monetary cost. For example, in the scenario of a real estate firm described earlier, a performance loss would lead to a loss in customer satisfaction and, eventually, a monetary cost.

The second type of cost is the *model retraining cost*, i.e., the resources spent to train an ML model on new data. Again, this cost may be monetary (e.g., renting a machine from an online cloud service to

* Corresponding author.
E-mail address: ananth.mahadevan@helsinki.fi (A. Mahadevan).

https://doi.org/10.1016/j.knosys.2024.111610
Received 6 October 2023; Received in revised form 9 February 2024; Accepted 4 March 2024
Available online 18 March 2024

execute the training) or of other kinds (e.g., energy consumption). This cost depends on the class of ML models used, the type of data, and the training algorithm.

There is a direct trade-off between these two costs of staleness and retraining. While retraining frequently results in a fresher ML model, the performance increase may be marginal when compared to the cost of retraining. On the other hand, infrequent retraining reduces the retraining cost but results in a stale model with performance loss.

Our contributions In this paper, we formalize and study the trade-off between staleness and retraining costs. Our main contribution is an online Cost-Aware Retraining Algorithm (Cara) that optimizes the trade-off between the two costs. Cara is defined as a function that makes `Retrain` or `Keep` decisions based on a choice of retraining parameters that consider both data and queries, along with the costs associated with each decision. We present three variants of Cara for different such parameters. In addition, we present a retrospective optimal algorithm Oracle, which we use as a baseline in our experimental comparisons to provide an upper bound on the performance of online retraining algorithms. In more detail, in synthetic experiments, we vary both data and query distribution and showcase how Cara captures and adapts to different data drifts, achieving similar performance to an optimal algorithm. Moreover, using real-world datasets, we compare with standard drift detection baselines from the literature and demonstrate that Cara exhibits better accuracy, but with fewer `Retrain` decisions, and thus lower total costs.

2. Related work

Gama et al. [1] provide a taxonomy of methods that handle data drift based on (1) whether the method uses single or multiple models, (2) whether the method adapts to data drift blindly or uses informed metrics, and (3) whether the method retrains or incrementally updates the model. Based on this taxonomy, our proposed Cara algorithm is a **single-model**, **informed** model-independent **retraining** method. Furthermore, we identify the following categories of existing work in the literature related to our contributions.

Multiple Model Methods These methods have multiple pre-existing trained models which are reused in the future. They monitor the data stream and select the best model from the pre-existing set. Pesaranghader et al. [7] use an error, memory and runtime (EMR) measure which they balance to select the best model. Mallick et al. [8] propose a method that can handle both covariate and concept drifts while picking the best model. While such methods explore the trade-off between different costs, they do not consider the decision to retrain, but rather choose from a fixed set of pre-existing models. By contrast, Cara addresses the problem of retraining ML models.

Informed Model-Independent Methods These methods use model-independent change detectors to identify data drifts. When a drift is detected, these methods adapt by making a `Retrain` (or update) decision. These change detectors monitor a model's error over a window and use statistical principles such as Page–Hinkley [9], Kolmogorov–Smirnov [4], Hoeffding bounds [10] or PAC learning [2,3] to detect a data drift. Other methods such as [11] train a neural network to predict a concept drift and avoid statistical hypothesis tests. However, unlike Cara, these methods only detect concept drift and ignore the covariate drifts in both data and query streams. Furthermore, unlike Cara, they do not balance the trade-off between the cost of retraining and make several unnecessary `Retrain` decisions due to false positive detections.

Model-Specific Incremental Methods These methods use change detectors that are tightly integrated with the model and update the model incrementally instead of making `Retrain` decision. Bifet and Gavaldà [12] propose the Hoeffding Adaptive Tree (HAT), which extends the standard Hoeffding Tree [13] by using the ADWIN drift detector [3] in each node of the decision tree to monitor its performance. Upon a signal from the drift detector, an alternate branch is trained in parallel and then swapped out when the performance degrades further. Similarly, Gomes et al. [14] propose an adaptive random forest which also uses ADWIN to monitor errors inside the nodes of each tree. Such model-specific methods typically make a decision to update the model after every new batch of data. Therefore, when using such model-specific methods, there is no ability to control when update decisions are made and thus offer no control over the trade-off between the cost of the update decision and the model's performance. In contrast, Cara offers fine-grained control over the trade-off between retraining cost and model staleness. Furthermore, Cara is model-independent and works with any generic ML model.

Data-Aware Retraining These methods use data-aware metrics to inform their `Retrain` decisions. Žliobaitė et al. [15] propose a Return on Investment (ROI) metric to decide when a `Retrain` decision is useful. They show that when the gain in performance is larger than the resource cost a model should be updated. However, unlike our work, they only perform an offline analysis and suggest monitoring the ROI during online evaluation. Jelenčič et al. [16] combine a drift detector metric with KL-divergence metric to assess the optimal time to retrain a model. Their experiments show that retraining when both metrics indicate a data drift results in better model performance. However, unlike our work, they do not consider the resource costs of retraining the ML model while making decisions.

3. Preliminaries

In this section, we introduce the terms and notations that are necessary for the presentation of our technical contribution.

Data Each data entry (x, y) consists of a point $x \in \mathbb{R}^d$ and a target y. We use $\mathbf{X}$ and $\mathbf{y}$ to refer to a set of points and their respective targets. In what follows, we assume a stream setting in which the data $D = (\mathbf{X}, \mathbf{y})$ arrive over time in batches, with batch t denoted by $D_t = (\mathbf{X}_t, \mathbf{y}_t)$.

Data drift occurs when the arriving data change over time. There are two main types of data drift as defined by Gama et al. [1]. The first is **covariate drift** (or virtual drift), when only the distribution of the points $p(x_t)$ changes with time. This occurs when points in different batches come from different regions of the feature space. The second is **concept drift**, when the conditional distribution of the targets $p(y_t|x_t)$ changes with time. This occurs when the underlying relationship between the target and the points is different across different data batches. Furthermore, based on the frequency and duration of the change, concept drift can be categorized according to different patterns such as sudden, incremental, recurring, etc.

Queries An individual query is a point in a d-dimensional space, $q \in \mathbb{R}^d$, without a ground-truth target. In the streaming setting queries also arrive at every batch t denoted by $q_t \in Q_t$. As queries do not have any ground-truth target y, query streams may only exhibit a covariate drift when the query distribution $p(q_t)$ changes with time.

Model We use the term model to refer to a function $M : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} \in \mathbb{R}^d$ and $\mathcal{Y}$ is the set of targets for the problem. Models are trained in a supervised manner using data D that contain both points x and ground-truth targets y. Once trained, a model is used to predict the target incoming query points $\hat{y} := M(q)$. In the stream setting, M_t denotes a model trained using the data from batch t i.e., D_t.

Retraining Algorithm A retraining algorithm decides whether to retrain the ML model or not. Formally, it consists of a decision function $\mathcal{R}$ with parameters θ. At each batch t, the decision function $\mathcal{R}$ receives the data D_t, queries Q_t and existing model $M_{t'}$ (trained at batch $t' < t$). $\mathcal{R}$ makes either a `Retrain` or `Keep` decision based on its input and parameters θ, where θ typically consists of cost-related thresholds. A `Retrain` decision indicates that the model must be retrained, while the `Keep` decision indicates that the existing model will be kept without retraining. In what follows, we denote decisions of retraining algorithms as $\mathcal{R}(t, t', \theta) \in \{\texttt{Keep}, \texttt{Retrain}\}$.

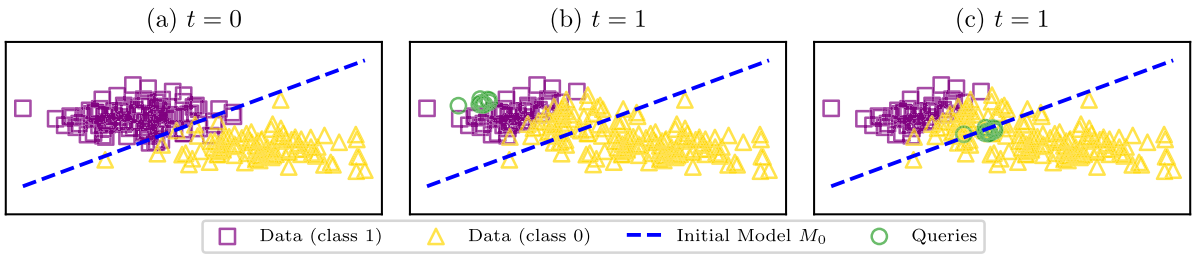


Fig. 1. First scenario. (a) Initial data D_0 and model M_0. Concept drift occurs at $t = 1$. (b) Queries are far from misclassifications. (c) Queries are close to misclassifications.

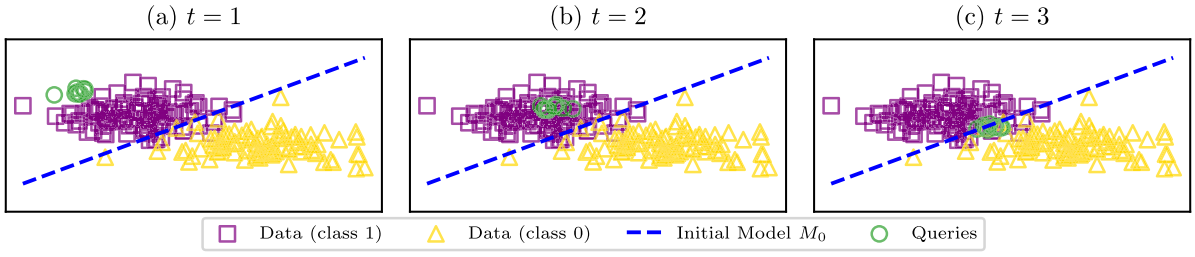


Fig. 2. Second scenario. (a)–(c) Data has no concept or covariate drift in batches $t = 1$ till $t = 3$. Queries show covariate shift, moving from being far from the decision boundary in (a) to being closer to the misclassifications in (c).

4. Problem formulation

In this section, we define the costs considered when making a `Retrain` or `Keep` decision and formalize the cost-optimization problem we address. First, in Section 4.1, we motivate and define the cost of model staleness in terms of expected query performance. Second, in Section 4.2, we define the model retraining cost which represents the resources required to retrain a model on a batch of new data. Third, in Section 4.3, we define a cost matrix that is computed offline retrospectively, given all the data and query batches within a given time interval, and which represents the possible costs for all the different possible retraining decisions in that time interval. Based on this cost matrix, we define the cost of any retraining strategy, i.e., any sequence of `Retrain` & `Keep` decisions. Lastly, in Section 4.4, and based on the aforementioned definitions, we define the problem statement.

4.1. Staleness cost

The performance of a model potentially changes as the data undergoes a data drift. In this paper, we aim to quantify the cost of keeping a model trained on old data, referred to as the "staleness cost", in terms of the model's query performance. Let us 'build' the formal definition of this cost as we discuss two exemplary scenarios that demonstrate how the performance of a model may change for different cases of data drift and queries.

In the first scenario, shown in Fig. 1, we consider a linear classification model trained on the initial 2D data at batch $t = 0$ in Fig. 1(a). Then, in the following batch $t = 1$, a concept drift occurs as seen in Fig. 1(b) and (c) changing the distribution of class labels. This leads to the stale model M_0 misclassifying some new data points. Typical concept drift methods will detect the drift in the data and make a `Retrain` decision ignoring the query distribution. However, the performance of the model does depend on the query distribution of the queries on which it is called to make a prediction, and therefore so does the cost associated with a potential performance loss. Specifically, if the query region is far from the misclassification region, as in Fig. 1(b), then a `Retrain` decision will yield only small improvements in query performance, and so the staleness cost of a decision to `Keep` the current model is small in this case. Conversely, if the query region is closer to the misclassification region, as in Fig. 1(c), then a `Retrain` decision may improve query performance significantly, and so the staleness cost of a decision to `Keep` the current model is large in this case.

To distinguish between these two cases, we consider the model's misclassifications in the vicinity of the queries. Towards this end, we define the model staleness cost for a single query point q given a model M and data D as

$$\psi(q, D, M) = \frac{1}{|D|} \sum_{(x,y)\in D} \mathrm{sim}(q, x) \cdot \ell(M, x, y), \tag{1}$$

where $\mathrm{sim}(q, x)$ is the similarity between the query and data point and $\ell(M, x, y)$ is the loss of the model on the data point x and labels y. The $\mathrm{sim}(\cdot, \cdot)$ function captures the probability that a query's label is similar to a data point and is typically defined as a function of their distance. Hence, the staleness cost as defined in Eq. (1) measures the expected loss of a single query in the region of the data given a model. Back to the scenario of Fig. 1, the staleness cost of the queries of Fig. 1(b) would be small, because the misclassifications (large ℓ) occur in areas away from the queries (small sim). Conversely, the staleness cost of the queries of Fig. 1(c) would be large, because the misclassifications (large ℓ) occur in areas near the queries (large sim).

Building upon Eq. (1), we define the absolute staleness cost of all queries Q_t in batch t given data D_t and model $M_{t'}$ trained at batch $t' \leq t$ as the sum of all the individual ψ costs and is defined as

$$\Psi\left(Q_t, D_t, M_{t'}\right) = \sum_{q\in Q_t} \psi\left(q, D_t, M_{t'}\right). \tag{2}$$

More generally, we use Ψ_{t_1,t_2,t_3} to denote $\Psi\left(Q_{t_1}, D_{t_2}, M_{t_3}\right)$ — the absolute staleness cost of using model from batch t_3 for the queries in batch t_1 given the data from batch t_2.

In the second scenario, shown in Fig. 2, the data stream exhibits no drift from $t = 1$ to $t = 3$, but the query stream does drift over three batches. Specifically, in each batch, the query distribution moves towards the decision boundary of the linear classifier M_0. Because of this, the absolute staleness cost Ψ, as defined in Eq. (2), increases from $t = 1$ to $t = 3$, because the number of data misclassifications increase as the queries move closer to the decision boundary. However, if the data distribution is static, retraining would not improve the query performance: as the data remain the same, the model that would be trained on them would also be the same. To identify such a scenario and prevent unnecessary `Retrain` decisions, we measure the relative *increase* in the staleness cost for the queries between two batches of

data. Towards this end, given the current batch t and the batch when the model was trained $t' \leq t$, we define a *relative staleness cost* as

$$\overline{\Psi}_{t,t'} := \Psi_{t,t,t'} - \Psi_{t,t',t'}. \tag{3}$$

Here, $\overline{\Psi}_{t,t'}$ measures the relative increase in staleness cost of using model $M_{t'}$ for the queries Q_t from $D_{t'}$ (data used to train the model) to D_t (data from current batch).

4.2. Retraining cost

Resources such as time and monetary budget are spent when retraining an ML model. The amount of resources spent depends on various factors, such as the class of the ML model and the dimensionality of the dataset. For example, assume it costs A\$ per hour on AWS SageMaker[1] to train a standard model configuration. However, not retraining results in using a stale model with decreased accuracy causing business losses of B\$ per hour. Depending on the values of A and B, it may be worth retraining more or less frequently. Therefore, there is a clear trade-off between the performance costs from model staleness and the resource costs from retraining the model.

We define retraining cost denoted by κ_t as the relative value of the resources spent in the common units of the model staleness cost at batch t. For example, assume a model takes 5 min to train, and the staleness cost measures the expected number of query misclassifications. Then, $\kappa_t = 150$ indicates that 5 min is equivalent to 150 query misclassifications i.e., one is willing to trade 1 min of time for 30 query misclassifications. Therefore, larger values of κ_t signifies resources are more important than model staleness cost and vice-versa. In the real world, the κ_t is dependent on the use-case and business requirements, and is set by the administrator of the ML system.

4.3. Cost matrix and strategy cost

Having defined the staleness and retraining costs, let us now collect in one matrix C the costs associated with the possible decisions at different times t to `Keep` or `Retrain` a model that was trained at time t'. Specifically, given a set of queries and data D_t and Q_t where $0 \leq t \leq T$ and models $M_{t'}$ where $t' \leq t$ we define an upper-triangular cost matrix C as follows

$$C[t', t] = \begin{cases} \overline{\Psi}_{t,t'} & \text{if } t' < t \\ \kappa_t & \text{if } t' = t \\ \infty & \text{otherwise,} \end{cases} \tag{4}$$

As defined, the upper-diagonal entries correspond to batches t for which a decision was made to `Keep` a model trained from a previous batch $t' < t$, incurring a staleness cost $\overline{\Psi}_{t,t'}$ but no retraining cost. Moreover, the diagonal entries correspond to batches $t = t'$ for which a decision was made to `Retrain` a model, incurring a retraining cost κ_t but no staleness cost, as the new model is based on the latest data (i.e., $\overline{\Psi}_{t,t} = 0$).

A *retraining strategy* is a sequence of the decisions made at each batch as seen in Algorithm 1. Formally, we define a strategy as the sequence of the model used at every batch $t \in [0, T]$

$$S = \left(s_0, s_1, \ldots, s_t, \ldots, s_T\right), \tag{5}$$

where $s_t \in [0, t]$ denotes that the model M_{s_t} was used at batch t. Hence, if $s_t = t$ then a `Retrain` decision was made at batch t otherwise a `Keep` decision was made. The cost of a given retraining strategy is the sum of the cost for every batch defined as

$$\text{cost}(S) = \sum_{t=0}^{T} C[s_t, t]. \tag{6}$$

[1] See https://aws.amazon.com/sagemaker/pricing/ for actual pricing details.

Algorithm 1 Retraining Algorithm

Input: Data D, Queries Q, Decision Function $\mathcal{R}$, parameters θ, T_{start}, T_{end}
Output: Retraining Strategy S
Start evaluation with $t' \leftarrow T_{\text{start}}$
$S \leftarrow \emptyset$
Train model $M_{t'}$ with $D_{t'}$
for $t \leftarrow T_{\text{start}}$ **to** T_{end} **do**
  $decision \leftarrow \mathcal{R}(t, t', \theta)$
  if $decision =$ `Retrain` **then**
    Retrain model with D_t and $t' \leftarrow t$
  end if
  $S \leftarrow S \cup \{t'\}$
end for
Return S

4.4. Problem statement

For a retraining algorithm, we assume two phases, an **offline optimization phase** and an **online evaluation phase**.

In the offline phase historical streaming data and queries are available. Typically, these data and queries would be collected and stored during a prior run of the system. Let the data D_t and queries Q_t corresponding to batches $0 \leq t \leq T_{\text{offline}}$ be available in the offline phase. During this phase, the complete offline cost matrix can be computed using Eq. (4) and analyzed. The goal during this phase is to learn the patterns in the streams and optimize the parameters of the retraining algorithm.

In the online phase batches $T_{\text{offline}} < t \leq T_{\text{online}}$ arrive sequentially. The retraining algorithm is evaluated using Algorithm 1 with $T_{\text{start}} = T_{\text{offline}} + 1$ and $T_{\text{end}} = T_{\text{online}}$. At each batch t the decision function $\mathcal{R}$ is called with its, possibly optimized, parameters. If a `Retrain` decision is made then a new model is trained otherwise the existing model is kept. At the end of the evaluation, we obtain the online retraining strategy and its corresponding online strategy cost.

Problem 1. Learn a decision function $\mathcal{R}$ in the offline phase which minimizes the online retraining strategy cost.

5. CARA: Cost-Aware Retraining Algorithm

In this section, we present three cost-aware decision functions $\mathcal{R}$, and their parameters θ. Each decision function gives rise to a variant of the retraining algorithm described in Algorithm 1. We collectively refer to the three variants as Cost Aware Retraining Algorithm, CARA. The variants are cost-aware because their parameters are chosen during the offline phase to optimize the trade-off between model staleness and retraining costs.

During the offline phase each retraining algorithm finds parameters θ^* that minimize the offline strategy cost as follows:

$$\theta^* = \arg\min_{\theta} \text{cost}(S), \tag{7}$$

where S is the retraining strategy from Algorithm 1 with $T_{\text{start}} = 0$ and $T_{\text{end}} = T_{\text{offline}}$. This optimization is possible because all the data and queries in the offline phase are available in advance, as described in Section 4.4. These optimal parameters θ^* balance the trade-off between the staleness and retraining costs and are then used with the decision function for the online evaluation. In practice, we perform the optimization using techniques such as grid search over the space of all parameters.

5.1. CARA-T: *Threshold variant*

The CARA-T variant has a threshold parameter τ and its decision function $\mathcal{R}$ is defined in Eq. (8).

$$\mathcal{R}(t,t',\{\tau\}) = \begin{cases} \texttt{Keep} & \overline{\Psi}_{t,t'} < \tau \\ \texttt{Retrain} & \text{otherwise.} \end{cases} \tag{8}$$

During the online phase, if the current model staleness cost is larger than the threshold a `Retrain` decision is made.

5.2. CARA-CT: *Cumulative threshold variant*

The CARA-CT variant inspired by [9] has a cumulative threshold parameter τ_{cum} and its decision function $\mathcal{R}$ is defined in Eq. (9).

$$\mathcal{R}(t,t',\{\tau_{\text{cum}}\}) = \begin{cases} \texttt{Keep} & \sum_{j=t'+1}^{t} \overline{\Psi}_{j,t'} < \tau_{\text{cum}} \\ \texttt{Retrain} & \text{otherwise} \end{cases} \tag{9}$$

During the online phase a variable tracks the cumulative cost, adding the current staleness cost at every batch from the last `Retrain` decision i.e., t' until the current time t. If this cumulative cost is larger than the parameter, then a `Retrain` decision is made and the cumulative cost is reset to 0.

5.3. CARA-P: *Periodic variant*

The CARA-P variant has a periodicity ϕ and initial offset a. During the online phase, the algorithm makes a `Retrain` decision every ϕ batch after an initial a number of batches. The decision function is defined in Eq. (10).

$$\mathcal{R}(t,t',\{\phi,a\}) = \begin{cases} \texttt{Retrain} & (t-a)\%\phi = 0 \\ \texttt{Keep} & \text{otherwise} \end{cases} \tag{10}$$

CARA-P is a generalized version of a static periodic adaptation policy described in Žliobaitė et al. [15] also used in many real-world frameworks such as TensorFlow Extended [17]. The difference is that CARA-P optimizes for the trade-off between the model staleness and retraining costs for a given dataset to find the optimal periodicity instead of using a pre-defined periodicity.

5.4. *Complexity analysis*

Each CARA variant computes the model staleness cost $\overline{\Psi}$ at each batch t during online evaluation. For analysis purposes, assume that the size of a batch of data and queries are B_D and B_Q, respectively. Then, the time complexity is $\mathcal{O}(B_Q B_D)$, dominated by the pairwise similarity computation, and the memory complexity is $\mathcal{O}(B_Q + 2B_D)$ for storing the data and queries.

6. The ORACLE Algorithm

In this section we present the ORACLE algorithm which returns the optimal retraining strategy retrospectively. The algorithm requires complete knowledge of all future batches of queries and data. Given this knowledge, the algorithm finds the retraining strategy which has the lowest strategy cost. In our experiments, we use the ORACLE algorithm in the online phase to provide a lower bound for the strategy cost of any retraining algorithm.

Assume an oracle provides the optimal strategy as a set of the batches when a `Retrain` decision should occur, denoted by O. Then the decision function follows the oracle as seen in Eq. (11).

$$\mathcal{R}\left(t,t',\{O\}\right) = \begin{cases} \texttt{Retrain} & \text{if } t \in O \\ \texttt{Keep} & \text{otherwise} \end{cases} \tag{11}$$

Algorithm 2 Memoize DP Table

Input: Cost Matrix C, number of batches T
Output: Memoized DP table V
Initialize V as $(T+1)\times(T+1)$ matrix filled with ∞
for $t = 0$ **to** T **do**
  $V[t,0] \leftarrow \sum_{t'=0}^{t} C[0,t']$ {fill first row}
end for
for $t \leftarrow 1$ **to** T **do**
  for $p \leftarrow 1$ **to** t **do**
    if $t = p$ **then**
      $V[t,p] \leftarrow C[t,t] + \min_{t'} V[t-1,t']$
    else
      $V[t,p] \leftarrow C[p,t] + V[t-1,p]$
    end if
  end for
end for
Return: V

Algorithm 3 Oracle Retrains

Input: DP table V, number of batches T
Output: Oracle `Retrain` decision batches O
$p \leftarrow \arg\min_{p'} V[T,p']$
$O \leftarrow \{p\}$
while $p > 0$ **do**
  $p \leftarrow \arg\min_{p'} V[p-1,p']$
  $O \leftarrow O \cup \{p\}$
end while
Return O

To find the optimal `Retrain` decision set O, we first define the optimal strategy as

$$S^* = \arg\min_{S} \text{cost}(S). \tag{12}$$

Then, we formulate a dynamic programming (DP) problem to find the optimal strategy cost and subsequently the optimal strategy.

Let $v(t,p)$ be the strategy cost at batch t using a model trained at batch p. Then, the optimal strategy cost at t is

$$v^*(t) = \min_{p\in[0,t]} v(t,p), \tag{13}$$

where the term $v(t,p)$ term is defined as

$$v(t,p) = \sum_{t'=p}^{t} C[p,t'] + v^*(p-1). \tag{14}$$

The first term in Eq. (14) is the total cost of using the model trained at batch p in the batches from p to t. The second term in Eq. (14) is the optimal strategy cost prior to the last model retraining. This sets up the recursive formulation for the DP problem. Therefore, solving for $v^*(T)$ yields the optimal strategy cost at the end of batch T.

Algorithm 2 describes a top-down method to memoize the strategy costs $v(\cdot,\cdot)$ in the form of a DP table V using the complete cost matrix C. Then, Algorithm 3 uses the computed V and returns the oracle set O. Note, O is a partial strategy consisting of only the `Retrain` decision batches. The optimal strategy S^* can be obtained by expanding O to include the `Keep` decisions.

The ORACLE algorithm requires $\mathcal{O}(T^2)$ memory to materialize the memoized table V and the cost matrix C, where T is the number of batches. The algorithm has three stages, namely computing C, memoizing V and returning the oracle retrains O. Therefore, the overall running time complexity is $\mathcal{O}(T^2 B_D B_Q + T^2 + T) = \mathcal{O}(T^2 B_D B_Q)$, where B_D and B_Q are the size of a batch of data and query.

Table 1
Dataset statistics. N is number of points, d is number of dimensions, T_{offline} and T_{online} indicate the offline and online batches retrospectively.

Name	N	d	T_{offline}	T_{online}
Gauss	100 000	2	25	100
Circle	100 000	2	25	100
CovCon	100 000	2	25	100
Electricity	45 312	6	25	100
Airlines	539 383	7198[a]	25	100
Covertype	581 000	54	25	100

[a] 5 nominal attributes into 7196 one-hot and 2 numeric features.

7. Experimental setup

We evaluate our Cara algorithm against the Oracle algorithm and other baselines on several synthetic and real-world datasets shown in Table 1. In our experiments, we have fixed batch sizes and therefore consider a static retraining cost, i.e., $\kappa_t = \kappa$. We focus on the task of binary classification, i.e., $\mathcal{Y} = \{0, 1\}$. Hence, to compute the $\overline{\Psi}$ cost, we use the radial basis function (RBF) kernel for similarity and the 0-1 loss function defined as follows

$$\text{sim}(q, x) = \exp\left(-\gamma \|q - x\|^2\right), \tag{15}$$

$$\ell(M, x, y) = \mathcal{I}(M(x) \neq y) \tag{16}$$

where γ is the inverse of the variance and $\mathcal{I}$ is the indicator function.

7.1. Synthetic datasets

We create several synthetic datasets with different concept and covariate drifts. In each synthetic dataset, we generate 100 batches and in each batch we create 1000 data points and labels.

Gauss A 2D synthetic dataset that has a gradual covariate shift. Data points are sampled from a Gaussian whose centers move at each batch introducing covariate drift. These points are labeled using a parabolic classification boundary. At each batch t, points (x_1, x_2) are sampled from a Gaussian distribution with centers at $(c, 0.5 - c)$, where $c = ((t + 1)\%15)/30$. The decision boundary for each point (x_1, x_2) is fixed and given by $x_2 > 4(x_1 - 0.5)^2$.

Circle [7] A 2D dataset that has a gradual concept drift. Each data point has two features (x_1, x_2) which are drawn from a uniform distribution. The classification is decided by the circle equation $(x_1 - c_1)^2 + (x_2 - c_2)^2 - r^2 > 0$, where (c_1, c_2) is the circle center and r is the radius. The centers and radius of this circle are changed over time resulting in a gradual concept drift.

CovCon [8] A 2D dataset having both covariate and concept drift. In each batch t, data points (x_1, x_2) are drawn from a Gaussian distribution with mean $((t + 1)\%7)/10$ and a fixed standard deviation of 0.1 which introduces covariate drift. The decision boundary of a data point is given by $\alpha * \sin(\pi x_1) > x_2$. Every 10 batches, the inequality of the decision boundary shifts from $>$ to $<$ and vice versa introducing an abrupt periodic concept drift.

7.2. Real-world datasets

We experiment with three real-world datasets Electricity, Covertype and Airlines which have unknown concept and covariate drifts. For each dataset we split the data points into 100 batches and use 25 batches for the offline phase.

Electricity [18] A binary classification dataset with the task to predict rise or fall of electricity prices in New South Wales, Australia. The data has concept drift due to seasonal changes in consumption patterns.

Covertype [19] Dataset containing 54 cartographic variables of wilderness in the forests of Colorado and labels are the forest cover type. We use binary version of the dataset from the LibSVM library [20].

Airlines [14] Dataset with five nominal and 2 numerical features describing the airlines, flight number, duration, etc. of various flights. The classification task is to predict if a particular flight will be delayed or not. We use the version from the Sklearn Multiflow library [21].

7.3. Query distributions

For each synthetic dataset, we generate 100 queries in each batch t from two different query streams.

In the first query stream, the queries are sampled from the data stream. We randomly sample 10% of the data entries in every batch and assign them as queries. Here, the points $x \in \mathbf{X}_t$ are used as queries q in the experiment while the labels $y \in \mathbf{y}_t$ are used during evaluation. We use the suffix "-D" for datasets with the first query stream. For example, CovCon-D is the CovCon dataset with queries sampled from data stream.

In the second query stream, queries come from a static Gaussian distribution centered at $(0.5, 0.5)$ with a standard deviation of 0.015. Here, we use the known concept to generate ground-truth labels y for each query which are used during evaluation. We use the suffix "-S" for datasets with the second query stream. For example, CovCon-S is the CovCon dataset with static queries. These different query streams will highlight the adaptability of the retraining algorithms to different query distributions.

For real-world datasets, we only consider the first stream of queries sampled from the data stream.

7.4. Baselines

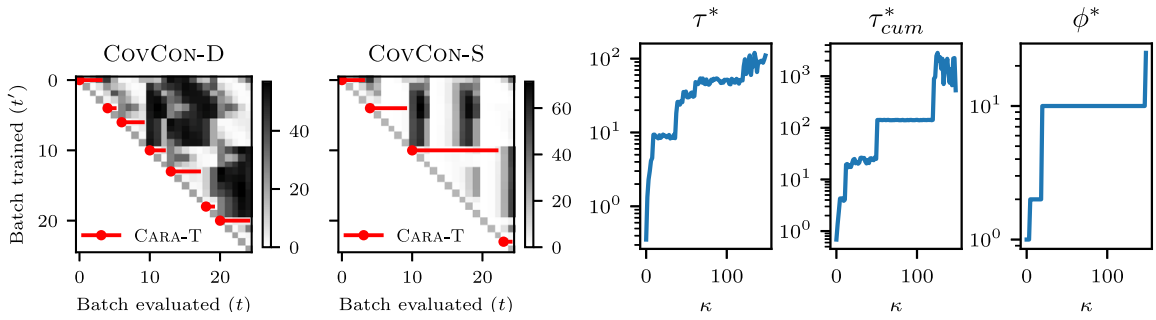
The Cara variants are categorized as single-model, informed, model-independent retraining methods as per the taxonomy of Gama et al. [1] discussed in Section 2. Therefore, we compare Cara against four baselines which also take a model-independent approach to make retraining decisions for a single model, described below.

ADWIN This method uses the signal from the model-independent drift detector proposed by Bifet and Gavaldà [3] to make `Retrain` decisions. In detail, this method maintains statistics of the model's error rate over two sliding windows corresponding to old and new data. When these window statistics differ beyond a specific threshold, a data drift is detected, and a signal is raised. If a signal is raised, the method makes a `Retrain` decision at the end of the corresponding batch.

DDM This method uses the signal from the drift detector proposed by Gama et al. [2] to make `Retrain` decisions. The method monitors the online error rate of the current model on the data stream. When this online error rate increases beyond a certain threshold, the detector assumes that a drift has occurred in the data stream and raises a signal. If a signal is raised, the method makes a `Retrain` decision at the end of the corresponding batch.

Markov This method makes `Retrain` decisions based only on the current model staleness cost $\overline{\Psi}(t, t')$ and retraining cost κ_t. The decision function of the method is described below in Eq. (17).

$$\mathcal{R}(t, t') = \begin{cases} \texttt{Keep} & \overline{\Psi}(t, t') < \kappa_t \\ \texttt{Retrain} & \text{otherwise} \end{cases} \tag{17}$$



(a) Offline cost matrices for CovCon datasets at $\kappa = 20$ (b) Optimal parameters for Cara variants for CovCon-D

Fig. 3. (a) Varying query distribution. The red lines indicate the offline optimal Cara-T strategy (markers and solid lines correspond to `Retrain` and `Keep` decisions respectively). (b) Varying the retraining cost.

This baseline always makes Markovian decisions and is similar to an uncalibrated Cara-T variant where the threshold is always the model retraining cost κ_t.

Oracle This method uses the retrospectively optimal retraining strategy to make `Retrain` decisions. Specifically, we compute the online cost matrix and use the Oracle algorithm (see Section 6) to obtain the optimal retraining strategy. This optimal retraining strategy has the lowest strategy cost, and therefore, the Oracle baseline is a lower bound on the strategy cost for all retraining algorithms.

In addition to the above baselines, we also compare against the **NR** (Never-Retrain) baseline, which is a trivial **blind** method that always makes a `Keep` decision and never retrains the model. Therefore, the decision function of this baseline is $\mathcal{R}(t, t') = \texttt{Keep}$. The retraining strategy from this baseline is to always use the oldest trained model in a stream. Moreover, as the retraining cost increases, all other retraining algorithms should ideally devolve into the NR strategy.

7.5. Evaluation metrics

For each retraining strategy from a retraining algorithm we report the strategy cost and number of `Retrain` decisions along with the following two metrics.

Query Accuracy We perform a *prequential* [1] i.e., test-then-train evaluation. For example, if a `Retrain` decision is made in batch t to update the model $M_{t'}$, the model is used first to answer queries Q_t and then retrained with data D_t to produce the model M_t. In this manner we compute the query accuracy at every batch and then report the average query accuracy. To achieve this, we stagger the strategy of a retraining algorithm by one batch to determine the model to be used in answering the queries. Higher values are good and indicate that the retraining algorithm is able to answer queries effectively.

Strategy Cost Percentage Error (SCPE) We compute the relative difference between a retraining algorithm's strategy cost and the optimal strategy cost from the Oracle baseline. Percentage error is defined as $\text{PE} = 100 \times |(b - a)|/|b|$, where b and a are the theoretical and experimental values respectively. Lower values are good and indicate that the retraining algorithm is closer to the optimal Oracle algorithm.

7.6. Implementation details

For experiments, we use a Random Forest (RF) classifier from the Scikit-Learn library [22] as the model. For each experiment we use five random seeds and average the results over the seeds. The ADWIN and DDM baselines are implemented using Scikit-Multiflow [21].

For all Cara variants we select the length scale γ of the RBF kernel based on the number of features d. During the offline phase, we construct the offline cost matrix as described in Eq. (4). Then for the Cara-T and Cara-CT variants, we use the dual annealing optimizer from SciPy [23] to find the optimal parameters τ^* and τ^*_{cum}. For the Cara-P variant we perform a linear search over $\phi \in [1, T_{\text{offline}}]$ to find the optimal parameter ϕ^*. We use these optimal parameters to evaluate Cara variants during the online phase.

All experiments were performed on a Linux machine with 32 cores and 50 GB RAM. The data and code used in our experiments is available at https://version.helsinki.fi/ads/cost-aware-retraining-algorithms.

8. Experimental results

In this section, we independently vary the staleness and retraining costs and analyze their trade-offs.

8.1. Varying staleness cost

We study the effect of the model staleness cost on the decisions of a retraining algorithm. Towards this, for a given dataset, we keep the retraining cost κ fixed and vary the query distribution to change the model staleness cost.

Fig. 3(a) presents the offline cost matrices at $\kappa = 20$ for the CovCon-D and CovCon-S datasets, which have non-static and static query distributions, respectively. We also show the retraining strategies of the Cara-T variant after finding the optimal threshold τ^* from the cost matrices. These retraining strategy are shown in the figure using a set of solid lines and markers which correspond to the `Keep` and `Retrain` decisions, respectively.

There are three main takeaways. First, the staleness cost captures both covariate and concept drift, seen by the gray regions and black checkerboard regions in the cost matrix of CovCon-D. Second, the staleness cost is query-aware, i.e., cost increases only when the queries are affected by concept or covariate drift. We see this query-awareness in CovCon-S, where fewer regions have a high staleness cost due to the static query distribution. Thirdly, for the given retraining cost, Cara-T adapts to CovCon-S making far fewer `Retrain` decisions than in CovCon-D.

8.2. Varying retraining cost

We study the effect of the retraining cost on the retraining decision by fixing the query distribution and varying κ for the CovCon-D dataset.

First, for a range of κ, we find the optimal parameters θ^* for each Cara variant using the offline cost matrix for every retraining cost as seen in Fig. 3(b). These optimal parameters implicitly control the decisions the retraining algorithm makes in the online phase. For example, when $\kappa \leq 50$, Cara-P makes a `Retrain` decision every other batch because the optimal periodicity $\phi^* = 2$. However, when $50 \leq \kappa \leq 125$, fewer `Retrain` decisions will be made because $\phi^* = 10$.

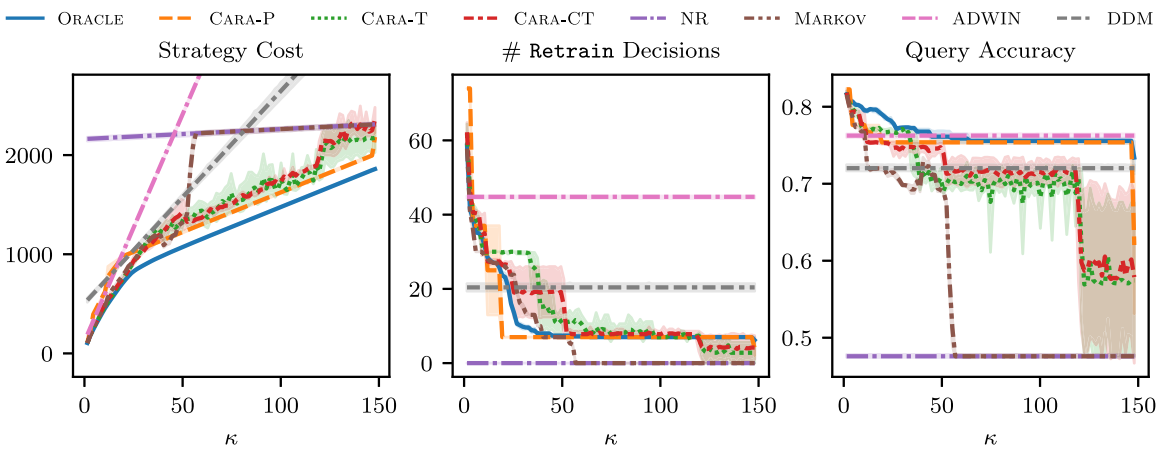


Fig. 4. Strategy cost, number of retrains and query accuracy as a function of retraining cost for the CovCon-D dataset.

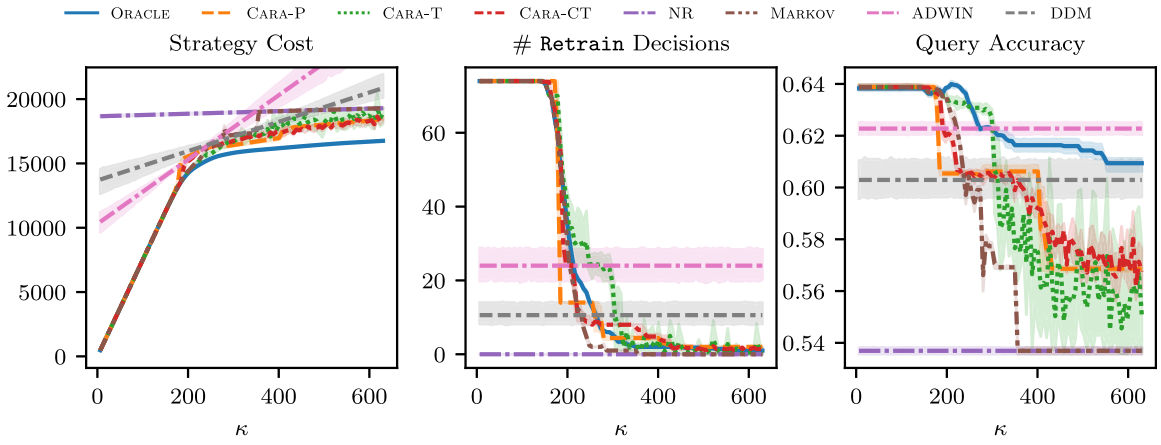


Fig. 5. Strategy cost, number of retrains and query accuracy as a function of retraining cost for the Airlines dataset.

Next, we evaluate every retraining algorithm in the online phase using the optimal parameters for a range of κ (see Appendix B). In Fig. 4, we present the evaluation metrics for the CovCon-D dataset as a function of the retraining cost κ. There are three main takeaways. First, Cara variants have strategy costs close to the optimal Oracle baseline, effectively balancing the cost-trade-off. Second, as the retraining cost increases, Cara implicitly reduces the number of `Retrain` decisions by minimizing for strategy cost. Third, the query accuracy of Cara is comparable to ADWIN and DDM baselines, except in regions of very high retraining cost, where it has high variance.

8.3. Real-world data analysis

We analyze the online performance of the Cara variants in the three real-world datasets for a range of retraining costs κ. Figs. 5–7 present detailed online evaluation metrics as a function of κ for the Airlines, Electricity and Covertype datasets respectively. There are three main observations.

First, Cara variants learn to retrain more frequently in regions of low retraining cost, similar to the Oracle baseline. This retraining results in higher query accuracy compared to the drift detection baselines. Second, based on the dataset, Cara variants adapt to retrain less frequently in regions of moderate retraining cost. For example, when $\kappa > 2.5$, all Cara variants (and the Oracle baseline) choose to retrain fewer than ten times in the Electricity dataset. Whereas, in the Covertype dataset, Cara variants drop to fewer than ten retrains only when the retraining cost is relatively higher. Third, Cara variants are very conservative in regions of high retraining cost and retrain infrequently. Here, we also observe a gap between the Oracle baseline and Cara variants in strategy cost and query accuracy while the number of `Retrain` decisions remains similar. This performance gap indicates Cara variants make few suboptimal `Retrain` decisions that in high-cost regions. Hence, there is scope for improvement through designing more complex variants of the Cara algorithm.

9. Discussion

This section presents the aggregated results for all retraining algorithms and datasets and discusses the key takeaways. First, Section 9.1 compares each retraining algorithm against the Oracle baseline and reports the average percentage error in strategy cost. Second, Section 9.2 compares the average query accuracy against the number of `Retrain` decisions taken for each algorithm. Lastly, Section 9.3 compares and discusses the different Cara variants' performance.

9.1. Strategy cost results

We present the mean SCPE over the retraining costs in Table 2. There are three main takeaways from these results. First, Cara variants have the lowest SCPE amongst all algorithms in almost all datasets. Even in the Covertype dataset, where the Markov baseline has the lowest SCPE, the Cara-T variant is close to the Markov baseline. Second, Cara-T has the least SCPE amongst all Cara variants consistently. In the CovCon-D and Airlines datasets, Cara-P has a lower SCPE, indicating an intrinsic periodicity in the underlying data or query distribution. However, in other datasets that do not present periodicity, such as CovCon-S

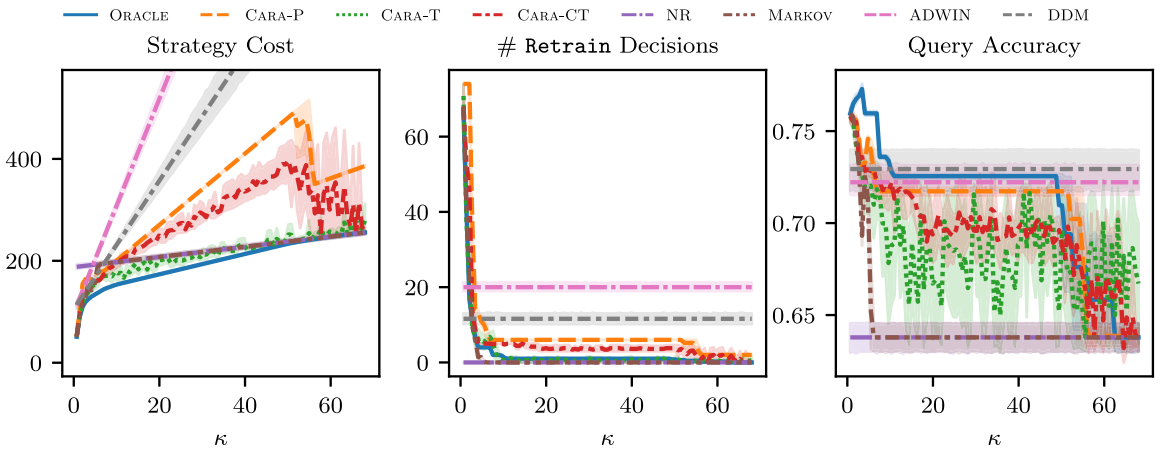


Fig. 6. Strategy cost, number of retrains and query accuracy as a function of retraining cost for the Electricity dataset.

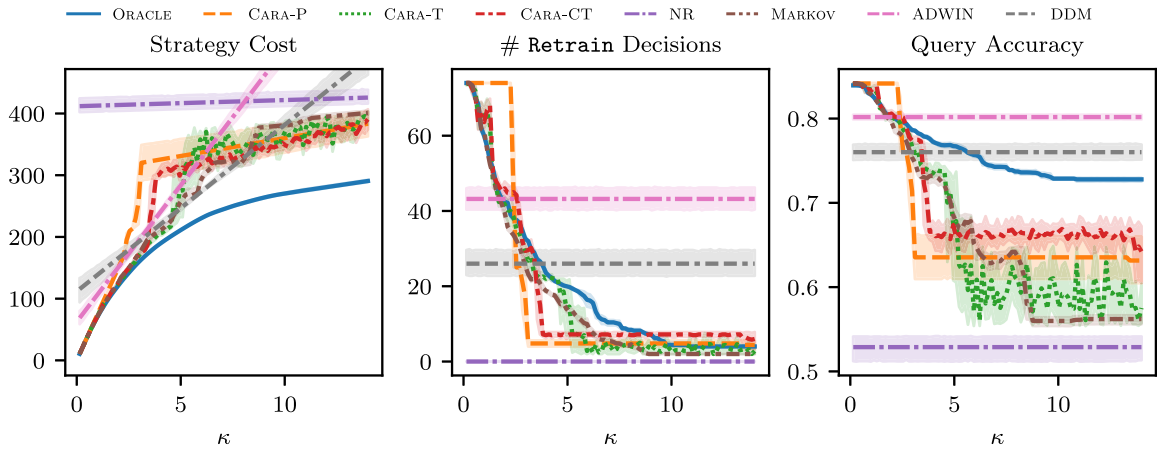


Fig. 7. Strategy cost, number of retrains and query accuracy as a function of retraining cost for the Covertype dataset.

Table 2
Average SCPE for all algorithms and datasets. Lower is better.

Dataset	Cara-T	Cara-CT	Cara-P	NR	Markov	ADWIN	DDM
CovCon-D	17.72	19.26	**16.3**	141.12	40.85	163.05	71.81
CovCon-S	**15.58**	22.94	104.79	191.38	25.17	305.52	106.03
Circle-D	**45.61**	47.29	84.96	350.75	185.16	108.59	115.83
Circle-S	**33.89**	45.15	79.34	305.75	193.38	132.77	109.38
Gauss-D	**8.63**	59.93	135.95	17.05	9.67	468.27	306.49
Gauss-S	**66.62**	96.38	202.09	169.52	105.51	628.08	433.44
Electricity	**8.32**	40.32	64.07	17.08	11.01	285.78	156.55
Airlines	**6.11**	5.48	5.45	141.13	8.99	86.41	99.95
Covertype	26.39	27.92	38.37	188.85	**25.61**	73.63	57.12

Table 3
Average query accuracy for all algorithms and datasets. Higher is better.

Dataset	Cara-T	Cara-CT	Cara-P	Oracle	NR	Markov	ADWIN	DDM
CovCon-D	0.7	0.71	**0.76**	**0.76**	0.48	0.56	**0.76**	0.72
CovCon-S	0.79	0.79	0.61	0.8	0.47	0.66	**0.85**	0.81
Circle-D	0.96	**0.97**	0.95	**0.97**	0.85	0.89	**0.97**	0.94
Circle-S	0.93	0.93	0.91	**0.96**	0.91	0.92	0.95	0.92
Gauss-D	0.9	0.88	0.8	**0.92**	0.9	0.9	0.88	0.9
Gauss-S	0.99	0.99	0.98	**1.0**	**1.0**	**1.0**	0.93	0.85
Electricity	0.68	0.69	0.7	0.71	0.64	0.64	0.72	**0.73**
Airlines	0.6	0.6	0.6	**0.62**	0.54	0.58	**0.62**	0.6
Covertype	0.66	0.7	0.67	0.76	0.53	0.66	**0.8**	0.76

Table 4
Average number of `Retrain` decisions for all algorithms and datasets.

Dataset	Cara-T	Cara-CT	Cara-P	Oracle	NR	Markov	ADWIN	DDM
CovCon-D	13.89	13.33	10.75	11.2	0	7.18	44.8	20.4
CovCon-S	10.05	10.22	9.68	8.21	0	5.99	44.8	20.4
Circle-D	5.67	7.49	5.41	3.89	0	2.13	11.0	4.2
Circle-S	3.56	6.06	5.41	3.09	0	1.42	11.0	4.2
Gauss-D	2.62	6.29	7.29	2.23	0	1.88	31.8	22.2
Gauss-S	2.57	5.04	5.85	1.92	0	0.92	31.8	22.2
Electricity	2.39	5.08	7.83	2.36	0	1.53	20.0	11.6
Airlines	27.69	25.29	24.43	25.23	0	23.12	24.0	10.6
Covertype	16.62	17.9	17.06	19.42	0	15.66	43.2	26.0

and Gauss-S, the Cara-P algorithm performs worse than other Cara variants due to unnecessary `Retrain` decisions. Third, the ADWIN and DDM baselines always have a high mean SCPE due to their inability to adapt to the change in query distribution and retraining costs.

9.2. Query Accuracy & Number of `Retrain` Decisions

Tables 3 and 4 present the mean query accuracy and number of `Retrain` decisions, respectively. There are three takeaways. First, the Oracle baseline has better or similar query accuracy than the drift detection baselines ADWIN and DDM, which retrain more frequently. Second, the Cara variants have query accuracy close to the Oracle baseline. Lastly, looking at the average number of `Retrain` decisions, we see that Cara variants are very similar and only slightly higher

compared to the Oracle baseline. Furthermore, in datasets where the NR baseline has high query accuracy, such as Circle-S and Gauss-S, the Cara variants learn to make fewer `Retrain` decisions compared to the ADWIN and DDM baselines.

9.3. Discussion on different Cara variants

Each variant of Cara exhibits specific properties valuable in different use cases.

First, as defined in Eq. (8), Cara-T relies on a single threshold and, therefore, is highly sensitive to changes in the model staleness cost. The variance of the optimal threshold τ^* found during the offline phase (seen in Fig. 3(b)), especially in regions of high retraining cost, is reflected in the variance of the metrics such as strategy cost and query accuracy (seen in Fig. 4). Nevertheless, due to this sensitivity, Cara-T has the lowest strategy cost amongst all the Cara variants across datasets and query distributions, as discussed in Section 9.1.

Second, Cara-CT makes a `Retrain` decision when the cumulative staleness cost crosses a threshold, as defined in Eq. (9). Therefore, Cara-CT is conservative by design, making fewer `Retrain` decisions than Cara-T while adapting to changing model staleness costs. We see this conservative nature in the low variance of strategy cost and query accuracy in Figs. 4 and 5. However, this performance stability comes with a slightly higher strategy cost, as seen in Table 2.

Third, Cara-P is a standard periodic adaptation policy. As seen in Section 8.1, Cara-P performs well and has a low strategy cost when the data (or queries) follow a periodic pattern. However, when the data (or query) distribution does not exhibit an apparent periodicity, Cara-P makes several unnecessary `Retrain` decisions with a higher strategy cost.

10. Conclusion

In our paper we studied the trade-off between the model staleness and retraining costs. Towards this, we motivated and defined the model staleness cost as the increase in the model's misclassifications in the region of the query. Further, we measured the retraining cost in the same metric which allowed us to compute the strategy cost of a retraining algorithm. We presented Cara, our cost-aware retraining algorithm which optimizes for the trade-off by minimizing the strategy cost. Through our analysis on synthetic data we demonstrated that Cara variants are able to adapt to both concept and covariate drifts in data and query streams. Furthermore, we show that Cara variants have lower number of `Retrain` decisions as a consequence of minimizing for strategy cost while having query accuracy comparable to drift detection baselines such as ADWIN and DDM. We also developed a retrospective optimal Oracle algorithm which we used as a baseline in our experiments as the lower bound the strategy cost of any retraining algorithm. We observed that amongst all Cara variants, Cara-T had the lowest strategy cost percentage error with respect to the Oracle baseline across both real-world and synthetic datasets. Lastly, in real-world datasets, Cara-T had query accuracy comparable to the DDM and ADWIN baselines while making fewer `Retrain` decisions.

Our current work has several limitations and potentials for future research. First, the current implementation of the staleness cost computes pairwise similarities which has a considerable computational cost when the number of data points in batch are large. Therefore, exploring the use of k-dimensional (KD) trees or core-sets will increase efficiency and improve scalability to larger datasets. Second, in regions of high retraining cost there is a gap between the performance of Cara variants and the Oracle baseline. Hence, using Reinforcement Learning (RL) based approaches to learn more complex decision functions during the offline optimization phase will lead to more robust retraining algorithms with better performance. Lastly, in our experiments the retraining cost κ was static, and the Cara variants were optimized in the offline phase for a given retraining cost. A future research direction is to study and develop retraining algorithms for a non-static retraining cost i.e., retraining cost which can change during each batch.

CRediT authorship contribution statement

Ananth Mahadevan: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Michael Mathioudakis:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Michael Mathioudakis reports financial support was provided by Academy of Finland.

Data availability

Link to code is provided in the manuscript.

Acknowledgments

Ananth Mahadevan would like to thank Arpit Merchant and Sachith Pai for their useful suggestions regarding the Oracle baseline. Michael Mathioudakis is supported by University of Helsinki and Academy of Finland projects MLDB (322046) and HPC-HD (347747).

All authors approved the version of the manuscript to be published.

Appendix A. Methodology

A.1. Scenario 1

In Fig. A.8, we present an extended version of the scenarios discussed in Section 4. Here, the scenario here spans across four batches, the initial data at batch $t = 0$ shown in Fig. 1(a) and the final batch at $t = 3$ shown in Fig. 1(b) and (d). At $t = 3$, we see the data has clearly drifted away with more class 0 data points being on the incorrect side of the decision boundary of initial model M_0. In Fig. 1(b), the queries are far away from the decision boundary and would therefore be classified correctly by M_0. We see in Fig. 1(c) where the staleness cost for the first row corresponding to model M_0 is low. The Oracle retraining strategy indicated by the red line also suggests that the initial model need not be updated in this scenario. In Fig. 1(d), the queries are much closer to the decision boundary of M_0 and the data misclassifications. Therefore, we see in Fig. 1(e) the staleness cost increases steadily in the first row. The Oracle retraining strategy is to retrain the model at $t = 2$ to reduce the staleness cost of keeping M_0 at batches $t = 2$ and $t = 3$.

A.2. Scenario 2

In Fig. A.9, we provide an example of the second scenario discussed in Section 4. In Fig. A.9(a)–(d) we see the query distribution moving towards the classification boundary of the initial model M_0. As expected, the simple staleness cost Ψ defined in Eq. (1) increases with incoming batches. However, in this scenario making a `Retrain` decision will not improve query performance as the data points and labels are static. The staleness cost $\overline{\Psi}$ defined in Eq. (3) mitigates this flaw. We see that the $\overline{\Psi}$ cost in Fig. A.9 is 0 indicating there is no cost to keeping the initial model M_0.

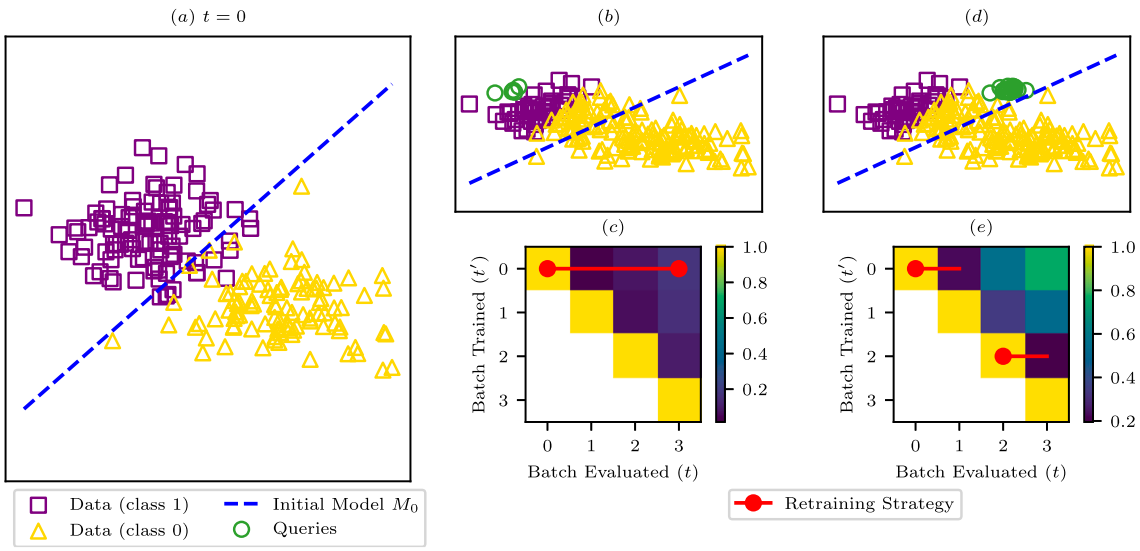


Fig. A.8. Extended example scenario from Fig. 1. (a) Initial data and model M_0. (b) Queries are far from misclassification. (d) Queries are closer to misclassifications. (c) and (e): Cost matrix and ORACLE retraining strategies with retraining cost fixed to $\kappa = 1$ for (b) and (d) respectively.

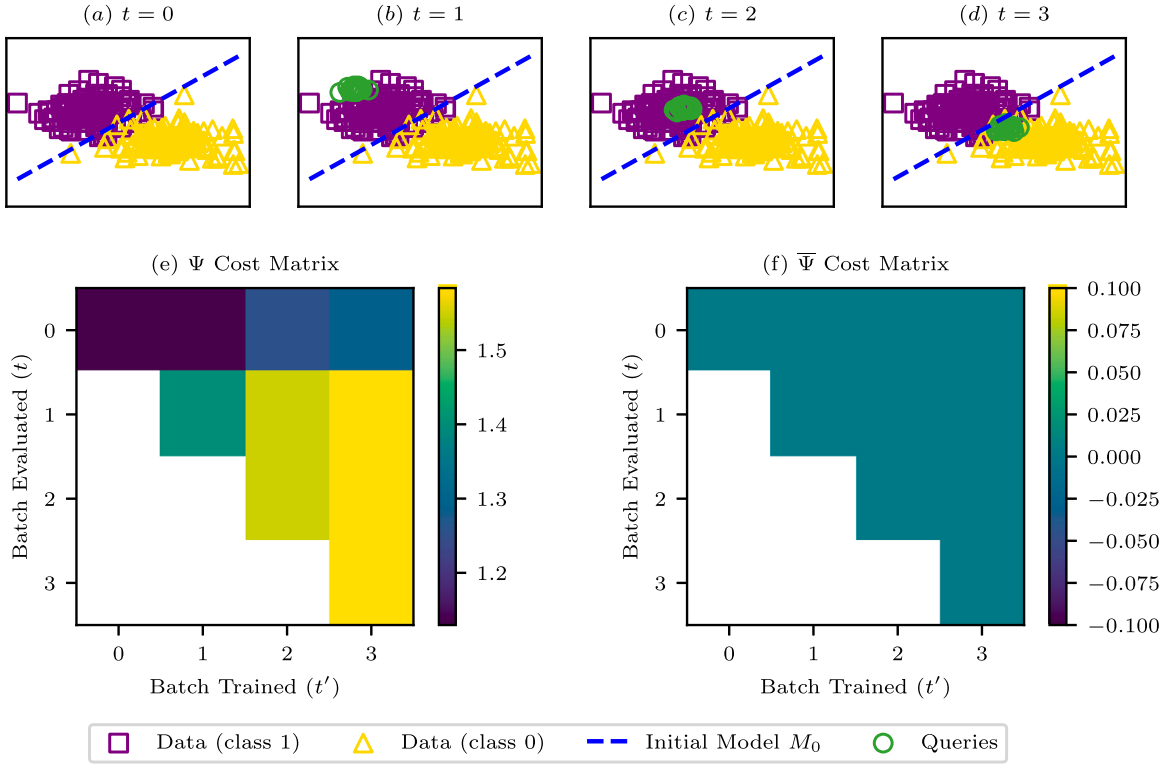


Fig. A.9. Extended scenario 2 from Fig. 2.(a)–(d) different batches of static data with moving query distribution. (e) Cost matrix computed using simple staleness Ψ (Eq. (2)). (f) Cost matrix computed using staleness cost $\overline{\Psi}$ (Eq. (3)).

Appendix B. Online evaluation

We evaluate every retraining algorithm in the online phase using the optimal parameters for a range of κ. In Fig. B.10, we present the online evaluation of all retraining algorithms for the COVCON-D dataset at $\kappa = 46$, where the cumulative strategy cost is on the y-axis and the online batches are on the x-axis. There are three takeaways. First, the NR algorithm has the highest cost as it does not retrain, accumulating staleness cost. Conversely, the ORACLE algorithm has the lowest cost as it is optimal. Second, the CARA-P variant is the closest to the ORACLE strategy cost, performing similarly to the MARKOV baseline. Third, although the ADWIN makes several `Retrain` decisions, it has a strategy cost similar to the NR baseline. This is because the ADWIN baseline is not cost-aware and accumulates retraining costs from making excessive `Retrain` decisions.

Appendix C. Synthetic dataset results

In this section, we present the results of all the synthetic datasets described in Table 1.

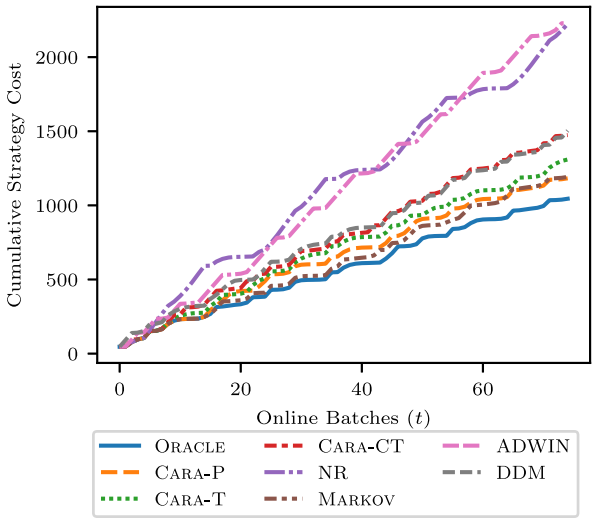


Fig. B.10. Online evaluation of all retraining algorithms for the CovCon-D dataset with $\kappa = 46$. The plot shows the cumulative strategy cost as a function of the online batches.

Fig. B.11 shows the results for the CovCon-S dataset which has a static query distribution. Here, we see the ineffectiveness of the Cara-P variant show in the significantly larger strategy cost. This large strategy cost is due to unnecessary `Retrain` decisions being made even when the query distribution is static. These decisions lead to accumulated retraining costs which add to the total cost of the Cara-P retraining strategy. Furthermore, the `Retrain` decisions are also poorly timed as we see the query accuracy of Cara-P is lower than other Cara variants. From these results, we conclude that periodic retraining strategies cannot adapt well to aperiodic data and query changes.

In Fig. C.12, we present the results for the Circle dataset. We observe that the number of `Retrain` decisions required is lower compared to the CovCon dataset. Furthermore, the variance of the Cara-T variant is seen in the query accuracy of Circle-S dataset, shown in Fig. C.12(b). This variance is due to the sensitivity of the Cara-T variant discussed in Section 9.

In Fig. C.13, we present the results for the Gauss dataset. We again see that Cara-P performs poorly in both Gauss-D and Gauss-S, as seen in Figs. C.13(a) and C.13(b). Interestingly, in Fig. C.13(b), we observe that ADWIN and DDM baselines make frequent `Retrain` decisions, however, have slightly lower query accuracy than other retraining algorithms. On the other hand, the Oracle baseline and the Cara variants make much fewer `Retrain` decisions, resulting in lower strategy cost, and have better query accuracy. Therefore, from these results, we conclude that the Oracle and Cara variants can adapt better to drift in data and query distributions while optimizing the trade-off between retraining costs and model staleness.

Appendix D. Results with logistic regression model

This section studies the impact of the choice of model class on the experiments and discussions from Sections 8 and 9. Towards this, we change the model M from a Random Forest classifier to a Logistic Regression (LR) classifier. We use the `SGDClassifier` from scikit-learn [22] with loss parameter set to "log". This loss parameter results in a LR classification model. Note, we do not perform an extensive grid-search and used the standard hyperparameters for the model. Therefore, the resulting trained LR model might perform poorly for different data and query distributions. We share these result in Tables D.5 to D.7.

Comparing the SCPE in Tables D.5 to 2, we see that strategy cost errors for all retraining algorithms are higher when using LR classifier. Furthermore, comparing Tables D.6 to 3, the query accuracies are also lower for all algorithms. For example, the query accuracy for the Circle-S dataset, is 0.09 across algorithms indicating the LR model trained on the data was unable to answer the static queries accurately. These results indicate that the LR classifier is unable to effectively model the non-linearity in the data, especially for Circle-S and Electricity datasets. Next, Cara variants make far fewer `Retrain` decisions compared to ADWIN and DDM baselines, as seen in Table D.7. Nevertheless, Cara variants have good query accuracy and low strategy cost. Finally, the overall trends discussed in Section 9 still hold, i.e., Cara-T and Cara-CT have low strategy cost and their query accuracy is comparable to the Oracle baseline.

Appendix E. Results with incremental algorithms

In this paper, we study retraining algorithms that decide whether to `Retrain` or `Keep` an existing ML model in the presence of data drifts. When a `Retrain` decision is made, the prior model is discarded, and a new model is trained from scratch using the new data. Incremental algorithms offer an alternative decision called `Update` to handle data drifts. When an `Update` decision is made, the prior model is retained, and the new data is used to update the prior model parameters incrementally. In this section, we discuss experiments with incremental algorithms.

Incremental algorithms are fundamentally different from the retraining algorithms described in Section 3 for two main reasons. First, incremental algorithms make different decisions (i.e., `Update` decisions instead of `Retrain` decisions). Moreover, incremental algorithms specifically require models that support incremental learning to make `Update` decisions. On the other hand, retraining algorithms are model-agnostic, as every ML model can be retrained from scratch. Second, information from prior data batches is present in the model parameters after an `Update` decision. In contrast, retraining algorithms only have access to the current data batch when a `Retrain` decision is made because the prior model is discarded. Therefore, due to this access to prior data, incremental algorithms operate outside the constraints of the batch setting for retraining algorithms as defined in Section 3.

Furthermore, these differences vary significantly based on the choice of model and the implementation of incremental learning for the given model, as shown by Chen [24]. For example, incrementally updating a multilayer perceptron (MLP) is fast, and they quickly forget prior data after updating compared to a retrained MLP. On the other hand, decision trees (DT) take longer to update and retain more prior data after updates than a retrained DT.

Regardless of the above differences, an incremental algorithm offers an alternative trade-off between the model performance and resources utilized compared to a retraining algorithm. Therefore, in this section, we study and compare the trade-off offered by incremental and retraining algorithms, specifically between the average query accuracy and the time spent in retraining/updating. Towards this, we choose the Hoeffding Tree Classifier (HT) [13] as the base model M, which is similar to the Random Forest classifier used in our original experiments (see Section 8). The HT model is a decision tree with incremental learning which uses the Hoeffding bound to make optimal choices on splitting nodes in the decision tree after observing new data points. We use the Scikit-Multiflow [21] implementation with retraining and incremental update methods.

In the following experiments, we compare three retraining algorithms (Cara-T-Retrain, ADWIN-Retrain, and Always Retrain) against three incremental algorithms (Always Update, ADWIN-Update, and HAT-Update). The Cara-T-Retrain and ADWIN-Retrain algorithms are the previously introduced Cara-T variant (see Section 5), and ADWIN baselines (see Section 7.4) are renamed to highlight that the algorithms make `Retrain` decisions explicitly. The Always Retrain and Always Update algorithms are uninformed (or blind, as per Gama et al. [1]) algorithms which make decisions without checking for data drifts.

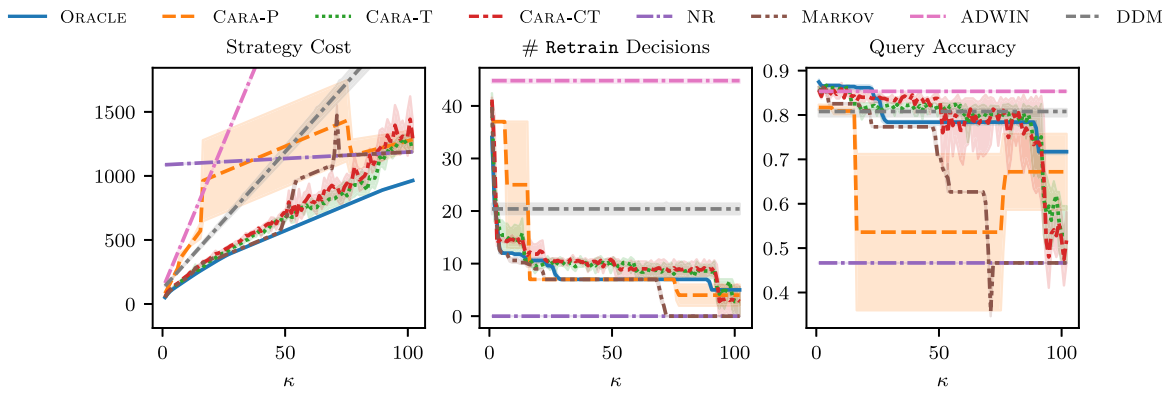


Fig. B.11. Strategy cost, number of retrains and query accuracy a function of retraining cost for the CovCon-S dataset.

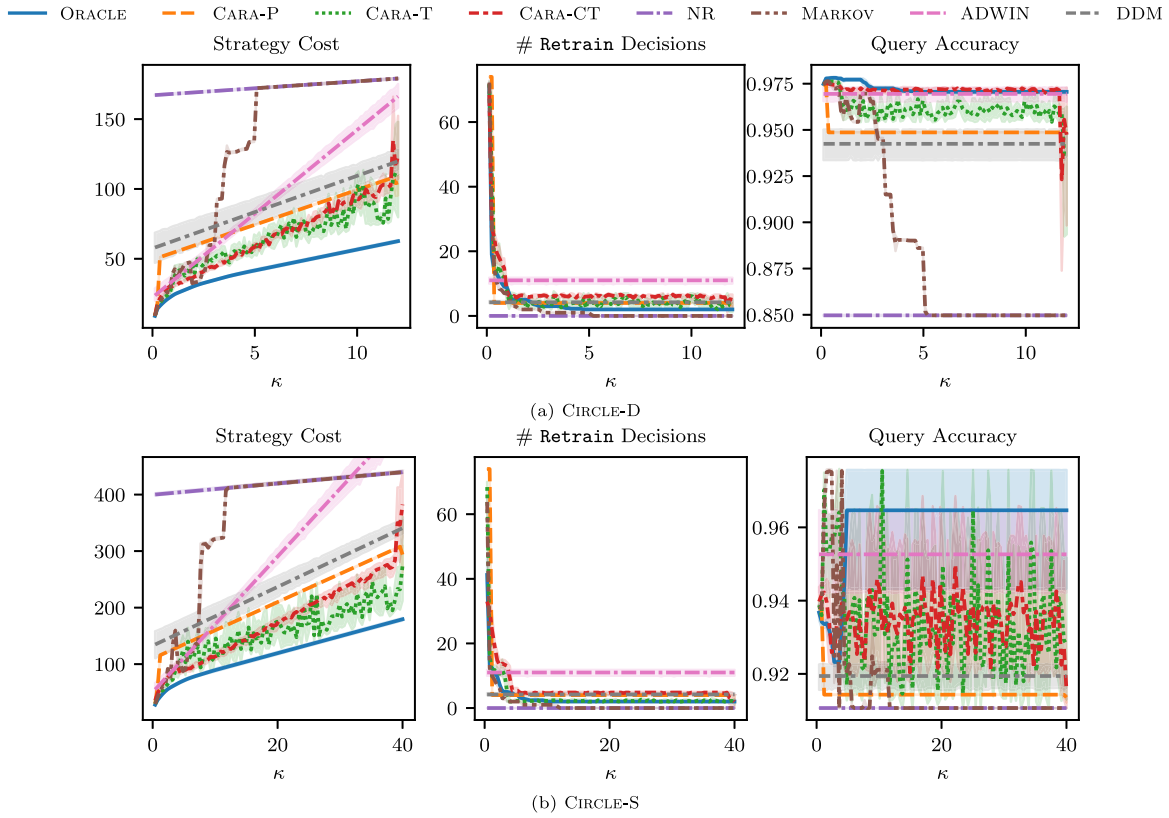


Fig. C.12. Strategy cost, number of retrains and query accuracy as a function of retraining cost for the Circle dataset.

As their name suggests, they always make a `Retrain` or `Update` decision, respectively. Next, the ADWIN-Retrain and ADWIN-Update algorithms use the model-independent ADWIN [12] drift detector and make a `Retrain` or `Update` decision, respectively, when a signal is raised from the detector. Lastly, the HAT-Update algorithm implements the model-specific HT incremental algorithm from Bifet and Gavaldà [12]. This algorithm uses ADWIN as a drift detector **internally** in each node of the decision tree. When a drift in the performance of a node is detected, an alternate tree branch is trained and then swapped out when performance degrades further. Therefore, due to its model-specific implementation, the HAT-Update algorithm makes an `Update` decision for every batch to process the data and internally updates the model. The main difference between the HAT-Update and Always Update algorithms is that the HAT-Update algorithm is specifically designed to handle data drifts for the HT model.

The trade-offs offered by the different retraining and incremental algorithms are shown in Fig. E.14. In each plot the x-axis is the total time spent either retraining or updating the model when a `Retrain` or `Update` decision is made. The y-axis shows the average prequential query accuracy for each algorithm. We analyze these results in detail and observe the following four insights.

First, Only the Cara-T-Retrain algorithm (Fig. E.14 green line) offers a range of trade-offs between query accuracy and retraining time. The fixed trade-offs are because the other algorithms make a fixed set of decisions based on the data drift and do not consider the cost of a `Retrain` or `Update` decision. In contrast, the Cara-T-Retrain algorithm offers different trade-offs based on the retraining cost parameter κ value. For example, in the Covertype dataset (Fig. E.14 bottom-left), as κ reduces, more `Retrain` decisions are made, which take more time. The query accuracy and time spent by the Cara-T-Retrain variant steadily increases till it achieves values similar to the Always Retrain algorithm.

Second, sometimes there is a small dip in the query accuracy of the Cara-T-Retrain variant as the retraining time increases, as seen in

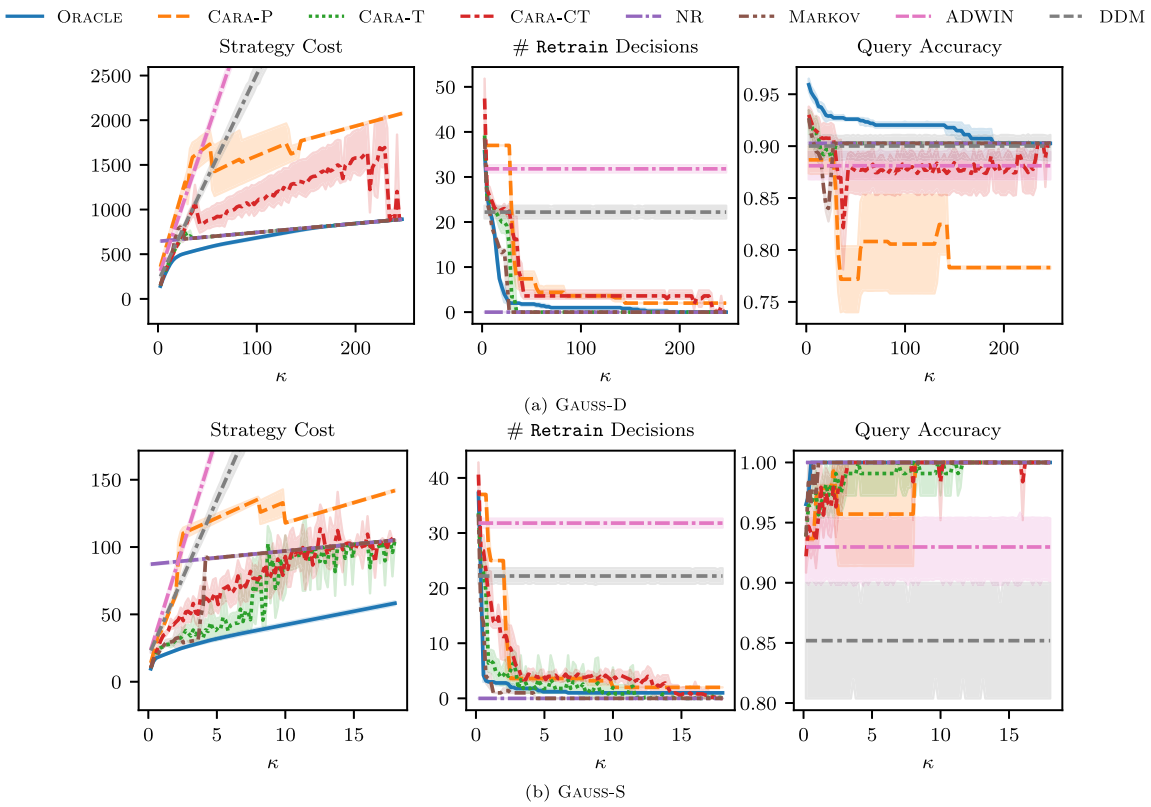


Fig. C.13. Strategy cost, number of retrains and query accuracy as a function of retraining cost for the Gauss dataset.

Table D.5
Average SCPE for all algorithms and datasets using a logistic regression model. Lower is better.

Dataset	Cara-T	Cara-CT	Cara-P	NR	Markov	ADWIN	DDM
CovCon-D	27.03	16.46	13.12	158.27	43.36	210.16	62.41
CovCon-S	55.61	126.68	449.28	256.81	50.05	852.72	230.38
Circle-D	840.88	1067.78	1170.57	2 370.12	2306.45	1879.82	1576.84
Circle-S	1635.49	626.12	1577.3	1 664.42	1645.01	2379.48	1833.82
Gauss-D	148.57	216.2	211.52	101.55	82.58	850.56	548.71
Gauss-S	387.83	275.75	1398.26	564.41	559.1	4490.27	3006.65
Electricity	1031.5	1830.55	6310.27	14 948.71	625.42	5230.54	3174.36
Airlines	11.68	35.89	31.38	25.11	10.38	211.51	90.32
Covertype	28.65	35.69	40.93	129.37	45.52	189.32	75.35

Table D.6
Average query accuracy for all algorithms and datasets using a logistic regression model. Higher is better.

Dataset	Cara-T	Cara-CT	Cara-P	Oracle	NR	Markov	ADWIN	DDM
CovCon-D	0.66	0.72	0.76	0.77	0.47	0.56	0.74	0.71
CovCon-S	0.67	0.56	0.45	0.62	0.47	0.54	0.91	0.9
Circle-D	0.81	0.8	0.81	0.77	0.82	0.82	0.81	0.82
Circle-S	0.09	0.09	0.09	0.09	0.09	0.09	0.09	0.09
Gauss-D	0.85	0.85	0.76	0.9	0.78	0.79	0.84	0.83
Gauss-S	1.0	1.0	0.95	1.0	1.0	1.0	0.7	0.59
Electricity	0.67	0.67	0.64	0.7	0.56	0.67	0.65	0.67
Airlines	0.56	0.57	0.56	0.57	0.55	0.56	0.59	0.56
Covertype	0.63	0.66	0.66	0.72	0.47	0.54	0.75	0.7

Table D.7
Average number of `Retrain` decisions for all algorithms and datasets using a logistic regression model.

Dataset	Cara-T	Cara-CT	Cara-P	Oracle	NR	Markov	ADWIN	DDM
CovCon-D	11.71	11.85	10.2	10.55	0	6.02	46.0	17.0
CovCon-S	7.82	8.29	6.15	4.37	0	2.5	46.0	17.0
Circle-D	2.32	2.22	4.24	3.23	0	0.57	6.8	5.0
Circle-S	0.56	2.06	4.15	2.56	0	0.23	6.8	5.0
Gauss-D	4.72	7.78	5.28	1.32	0	0.57	28.2	19.0
Gauss-S	0.85	1.89	4.61	1.16	0	0.05	28.2	19.0
Electricity	18.48	14.13	10.56	13.57	0	20.46	13.2	10.0
Airlines	6.52	10.06	10.13	5.51	0	4.72	29.4	11.6
Covertype	12.2	12.59	11.11	9.29	0	5.77	43.6	17.0

the Electricity (Fig. E.14 first column, third row) and Gauss dataset (Fig. E.14 third column, first and second rows). Two factors cause this drop in accuracy. The first factor is that the Cara variants optimize for the strategy cost, not the query accuracy. Therefore, Cara variants might make `Retrain` decisions, which reduces not only the strategy cost but also query accuracy. The second factor is that, unlike the Oracle baseline (see Section 6), the Cara variants are not optimal. Therefore, Cara variants might make suboptimal `Retrain` decisions, which may reduce query accuracy.

Third, In the CovCon dataset, the Always Update and ADWIN-Update algorithms have very poor query accuracy, close to 50% (Fig. E.14 first column, first and second rows). This poor query accuracy is because the prior data is present in the model parameters after each `Update` decision. Therefore, due to this prior data and concept in the incrementally updated model, the model misclassifies several queries when the abrupt concept drift (see Section 7.1) occurs in the dataset. On the other hand, HAT-Update performs better than the Always Update algorithm because the model-specific incremental algorithm is designed to swap out poorly performing decision branches when an abrupt concept drift occurs. Similarly, the ADWIN-Retrain and Always Retrain algorithms perform better than the Always Update algorithm because they make `Retrain` decisions that discard the prior model and retrain a new model from scratch. This retraining ensures that only the most recent data and concepts are present in the new model resulting in fewer misclassified queries.

Fourth, across all datasets, the Always Update (cyan marker) and the HAT-Update algorithm (navy blue marker) take more time compared to the Always Retrain algorithm (purple marker) to achieve similar query accuracy. This behavior is because the HT model is a decision tree that

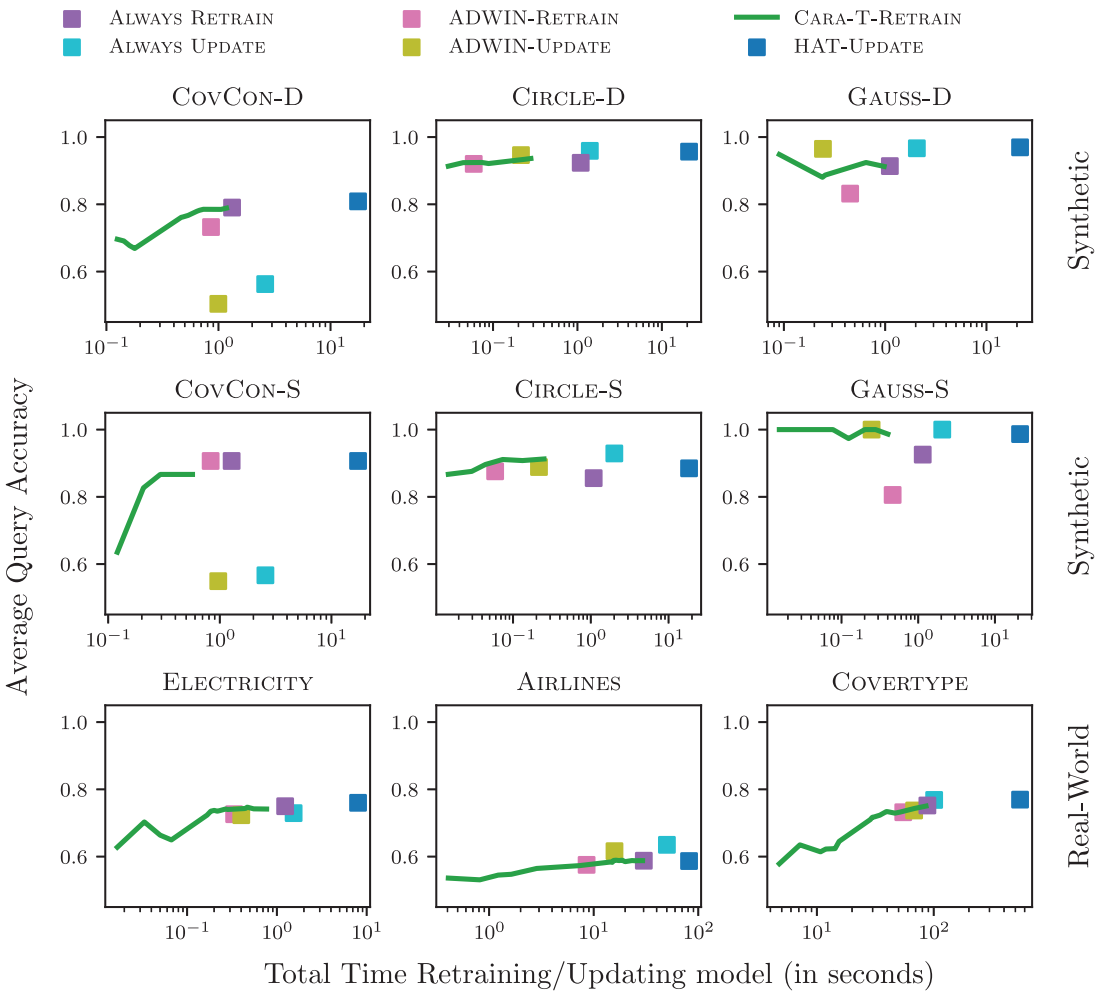


Fig. E.14. Trade-offs offered by incremental and retraining algorithms when using a base Hoeffding Tree Classifier (HT) model. The first two rows correspond to synthetic datasets and the third row corresponds to real-world datasets. In each plot the x-axis is the time spent by each algorithm in either retraining or updating the model and the y-axis is the average prequential accuracy at the end of online evaluation. There are three retraining algorithms (Always Retrain, ADWIN-Retrain and Cara-T-Retrain) which make `Retrain` decisions and three incremental algorithms (Always Update, ADWIN-Update and HAT-Update) which make `Update` decisions. HAT-Update is a model-specific incremental algorithm for the HT model. The Cara-T-Retrain algorithm provides a range of trade-offs shown by a green line by varying the retrain cost parameter κ. The other algorithms offer a fixed trade-off indicated by square marker.

takes more computations to incrementally add more decision branches to an existing tree than creating a new tree from scratch. This overhead to update an existing tree is compounded in the HAT-Update algorithm, where additional model-specific computations are required to handle data drifts.

In conclusion, this section provided an overview of incremental algorithms and their fundamental differences to retraining algorithms. We experimented with a HT model that supports incremental updates and evaluated several incremental and retraining algorithms across different datasets. From these experiments, we observed that incremental algorithms offer a fixed-trade off between query accuracy and update time, and that these trade-offs are heavily dependent on the model implementation and the specific drifts present in the dataset. On the other hand, we observed that the Cara-T-Retrain variant provides a range of trade-offs by varying the retraining cost parameter κ which is robust to different data drifts.

References

[1] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Comput. Surv. 46 (4) (2014) 44:1–44:37, http://dx.doi.org/10.1145/2523813.

[2] J. Gama, P. Medas, G. Castillo, P. Rodrigues, Learning with drift detection, in: A.L.C. Bazzan, S. Labidi (Eds.), Advances in Artificial Intelligence – SBIA 2004, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2004, pp. 286–295, http://dx.doi.org/10.1007/978-3-540-28645-5_29.

[3] A. Bifet, R. Gavaldà, Learning from time-changing data with adaptive windowing, in: Proceedings of the 2007 SIAM International Conference on Data Mining, Society for Industrial and Applied Mathematics, 2007, pp. 443–448, http://dx.doi.org/10.1137/1.9781611972771.42.

[4] C. Raab, M. Heusinger, F.-M. Schleif, Reactive soft prototype computing for concept drift streams, Neurocomputing 416 (2020) 340–351, http://dx.doi.org/10.1016/j.neucom.2019.11.111, URL: https://www.sciencedirect.com/science/article/pii/S0925231220305063.

[5] S. Chandra, A. Haque, L. Khan, C. Aggarwal, An adaptive framework for multistream classification, in: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 1181–1190, http://dx.doi.org/10.1145/2983323.2983842.

[6] E. Strubell, A. Ganesh, A. McCallum, Energy and policy considerations for modern deep learning research, Proc. AAAI Conf. Artif. Intell. 34 (09) (2020) 13693–13696, http://dx.doi.org/10.1609/aaai.v34i09.7123, URL: https://ojs.aaai.org/index.php/AAAI/article/view/7123.

[7] A. Pesaranghader, H.L. Viktor, E. Paquet, A framework for classification in data streams using multi-strategy learning, in: T. Calders, M. Ceci, D. Malerba (Eds.), Discovery Science, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2016, pp. 341–355, http://dx.doi.org/10.1007/978-3-319-46307-0{_}22.

[8] A. Mallick, K. Hsieh, B. Arzani, G. Joshi, Matchmaker: Data drift mitigation in machine learning for large-scale systems, in: D. Marculescu, Y. Chi, C. Wu (Eds.), Proceedings of Machine Learning and Systems, 4, 2022, pp. 77–94, https://proceedings.mlsys.org/paper_files/paper/2022/file/069a002768bcb31509d4901961f23b3c-Paper.pdf.

[9] E.S. Page, Continuous inspection schemes, Biometrika 41 (1/2) (1954) 100–115, URL: http://www.jstor.org/stable/2333009.

[10] I. Frías-Blanco, J.d. Campo-Ávila, G. Ramos-Jiménez, R. Morales-Bueno, A. Ortiz-Díaz, Y. Caballero-Mota, Online and non-parametric drift detection methods based on hoeffding's bounds, IEEE Trans. Knowl. Data Eng. 27 (3) (2015) 810–823, http://dx.doi.org/10.1109/TKDE.2014.2345382.

[11] H. Yu, Q. Zhang, T. Liu, J. Lu, Y. Wen, G. Zhang, Meta-ADD: A meta-learning based pre-trained model for concept drift active detection, Inform. Sci. 608 (2022) 996–1009, http://dx.doi.org/10.1016/j.ins.2022.07.022.

[12] A. Bifet, R. Gavaldà, Adaptive learning from evolving data streams, in: N.M. Adams, C. Robardet, A. Siebes, J.-F. Boulicaut (Eds.), Advances in Intelligent Data Analysis VIII, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 249–260.

[13] J. Gama, R. Fernandes, R. Rocha, Decision trees for mining data streams, Intell. Data Anal. 10 (1) (2006) 23–45, http://dx.doi.org/10.3233/IDA-2006-10103.

[14] H.M. Gomes, A. Bifet, J. Read, J.P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, T. Abdessalem, Adaptive random forests for evolving data stream classification, Mach. Learn. 106 (9) (2017) 1469–1495, http://dx.doi.org/10.1007/s10994-017-5642-8.

[15] I. Žliobaitė, M. Budka, F. Stahl, Towards cost-sensitive adaptation: When is it worth updating your predictive model? Neurocomputing 150 (2015) 240–249, http://dx.doi.org/10.1016/j.neucom.2014.05.084.

[16] J. Jelenčič, J.M. Rožanec, D. Mladenić, KL-ADWIN: Enhanced concept drift detection over multiple time windows, in: Central European Conference on Information and Intelligent Systems, Faculty of Organization and Informatics Varazdin, Varazdin, Croatia, 2022, pp. 49–54, https://www.proquest.com/conference-papers-proceedings/kl-adwin-enhanced-concept-drift-detection-over/docview/2720988240/se-2.

[17] D. Baylor, K. Haas, K. Katsiapis, S. Leong, R. Liu, C. Menwald, H. Miao, N. Polyzotis, M. Trott, M. Zinkevich, Continuous training for production ML in the TensorFlow extended (TFX) platform, in: 2019 USENIX Conference on Operational Machine Learning, OpML 19, USENIX Association, Santa Clara, CA, 2019, pp. 51–53, URL: https://www.usenix.org/conference/opml19/presentation/baylor.

[18] M. Harries, N.S. Wales, SPLICE-2 comparative evaluation: Electricity pricing, 1999.

[19] J.A. Blackard, D.J. Dean, Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables, Comput. Electron. Agric. 24 (3) (1999) 131–151, http://dx.doi.org/10.1016/S0168-1699(99)00046-0, URL: https://www.sciencedirect.com/science/article/pii/S0168169999000460.

[20] C.-C. Chang, C.-J. Lin, LIBSVM: A library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (2011) 27:1–27:27, Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[21] J. Montiel, J. Read, A. Bifet, T. Abdessalem, Scikit-multiflow: A multi-output streaming framework, J. Mach. Learn. Res. 19 (72) (2018) 1–5, URL: http://jmlr.org/papers/v19/18-251.html.

[22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[23] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, İ. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors, Scipy 1.0: Fundamental algorithms for scientific computing in python, Nature Methods 17 (2020) 261–272, http://dx.doi.org/10.1038/s41592-019-0686-2.

[24] T. Chen, All versus one: An empirical comparison on retrained and incremental machine learning for modeling performance of adaptable software, in: 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS, 2019, pp. 157–168, http://dx.doi.org/10.1109/SEAMS.2019.00029.