Online Appendix to: A Survey on Concept Drift Adaptation

JOÃO GAMA, University of Porto, Portugal
INDRĖ ŽLIOBAITĖ, Aalto University and HIIT, Finland
ALBERT BIFET, Yahoo! Research Barcelona, Spain
MYKOLA PECHENIZKIY, Eindhoven University of Technology, the Netherlands
ABDELHAMID BOUCHACHIA, Bournemouth University, UK

A. PSEUDOCODE OF CONCEPT DRIFT ALGORITHMS

ALGORITHM 1: Page-Hinkley Algorithm

input: Admissible change δ, Drift threshold λ, Loss at example t: e_t ;
output: $drift \in \{TRUE, FALSE\}$;
/* Initialize the error estimators */ ;
$SR(0) \leftarrow 0$; $m_T(0) \leftarrow 0$; $M_T \leftarrow 1$;
/* Page Hinkley test */ ;
Let $SR(t) \leftarrow SR(t-1) + R(t)$;
Let $m_T(t) \leftarrow m_T(t-1) + R(t) - \frac{SR(t)}{t} - \delta$;
Let $M_T \leftarrow min(M_T, m_T(t))$;
if $m_T(t) - M_T \geq \lambda$ **then**
  $drift \leftarrow TRUE$;
else
  $drift \leftarrow FALSE$;
end

ALGORITHM 2: The `ADWIN` change detection algorithm

begin
  Initialize Window W;
  foreach $(t) > 0$ **do**
    $W \leftarrow W \cup \{x_t\}$ (i.e., add x_t to the head of W);
    repeat
      Drop elements from W
    until $|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| < \epsilon_{cut}$ *holds for every split of* W *into* $W = W_0 \cup W_1$;
  end
  Output $\hat{\mu}_W$
end

DOI: http://dx.doi.org/10.1145/2523813

ALGORITHM 3: The SPC change detection algorithm

input: Current decision model $\mathcal{L}$; X, y^T: Training Set with class $y \in Y$;
α and β parameters
begin
  Let X_t, y_t be the current instance and $\hat{y}_t \leftarrow \mathcal{L}(X_t)$ be the prediction;
  Let $error_t \leftarrow E(\hat{y}_t, y_t)$, compute mean p_t and variance σ_t of with $error_t$;
  if $p_t + \sigma_t < p_{min} + \sigma_{min}$ **then**
    $p_{min} \leftarrow p_t$ and $\sigma_{min} \leftarrow s_t$;
  end
  if $p_t + \sigma_t < p_{min} + \beta \times \sigma_{min}$ **then**
    `In-control`: Warning? $\leftarrow False$; update $\mathcal{L}$ with the instance X_t, y_t;
  else
    if $p_t + \sigma_t < p_{min} + \alpha \times \sigma_{min}$ **then**
      if *NOT Warning* **then**
        `Warning`: Warning? $\leftarrow True$; start buffer $\leftarrow \{X_t, y_t\}$;
      else
        `Warning`: buffer $\leftarrow$ buffer $\cup\{X_j, y_j\}$;
      end
    else
      `Out-control`: train a new decision model using the instances in the buffer;
      Warning? $\leftarrow False$; re-start p_{min} and σ_{min};
    end
  end
end

ALGORITHM 4: The CVFDT algorithm

intput: S is a sequence of examples,
X is a set of symbolic attributes,
$G(.)$ is a split evaluation function,
δ is one minus the desired probability of choosing the correct attribute at any given node,
τ is a user-supplied tie threshold,
w is the size of the window,
n_{min} is the # examples between checks for growth,
f is the # examples between checks for drift.
output: HT is a decision tree.

Procedure CVFDT($S, X, G, \delta, \tau, w, n_{min}$)
/* Initialize */
Let HT be a tree with a single leaf l_1 (the root).
Let $ALT(l_1)$ be an initially empty set of alternate tree for l_1.
Let W be the window of examples, initially empty.
Initialize sufficient statistics to compute G.
/* Process the examples */
foreach *example* (x, y) *in* S **do**
  Sort (x, y) into a set of leaves L using HT and all trees in ALT of any node (x, y) passes through.
  Let ID be the maximum id of the leaves in L.
  Add $((x, y), ID)$ to the beginning of W.
  if $|W| > w$ **then**
    Let $((x_w, y_w), ID_w)$ be the last element of W
    ForgetExamples($HT, n, (x_w, y_w), ID_w$)
    Let $W = W$ with $((x_w, y_w), ID_w)$ removed
  end
  CVFDTGrow($HT, n, G, (x, y), \delta, n_{min}, \tau$)
  if *there have been* f *examples since the last checking of alternate trees* **then**
    CheckSplitValidity(HT, n, δ)
  end
end
return HT.

ALGORITHM 5: Algorithm for Dynamic Weighted Majority (DWM)

Input:
$\{\overrightarrow{x}, y\}_n^1$: training data, feature vector and class label; $c \in \mathbb{N}^*$: number of classes $c \geq 2$; β: factor for decreasing weights, $0 \leq \beta < 1$; θ : threshold for deleting experts; p: period between expert removal, creation, and weight update; $\{e, w\}_m^1$: set of experts and their weights; $\Lambda, \lambda \in \{1, ..., c\}$: global and local predictions; $\overrightarrow{\sigma} \in \mathbb{R}^c$: sum of weighted predictions for each class.

begin
  $m \leftarrow 1$
  $e_m \leftarrow$ Create-New-Expert()
  $w_m \leftarrow 1$
  foreach $i \leftarrow 1, ..., n$ **do**
    $\overrightarrow{\sigma} \leftarrow 0$
    foreach $i \leftarrow 1, ..., m$ **do**
      $\lambda \leftarrow$ Classify(e_j, $\overrightarrow{x_i}$)
      if $\lambda \neq y_i \wedge i \mod p = 0$ **then**
        $w_j \leftarrow \beta w_j$
      end
      $\sigma_\lambda \leftarrow \sigma_\lambda + w_j$
    end
    $\Lambda \leftarrow argmax_j \sigma_j$
    if $i \mod p = 0$ **then**
      $w \leftarrow$ Normalize-Weights (w)
      $\{e, w\} \leftarrow$ Remove-Experts ($\{e, w\}, \theta$)
      if $\Lambda \neq y_i$ **then**
        $m \leftarrow m + 1$
        $e_m \leftarrow$ Create-New-Expert()
        $w_m \leftarrow 1$
      end
    end
    foreach $j \leftarrow 1, \ldots, m$ **do**
      $e_j \leftarrow$Train(e_j, $\overrightarrow{x_i}$, y_i)
    end
    return Λ
  end
end

B. DATASETS FOR CONCEPT DRIFT

B.1. Synthetic

Synthetic data have several benefits: they are easy to reproduce and bear low cost of storage and transmission and, most importantly, provide an advantage of knowing the ground truth. For instance, we can know where exactly concept drift happens, what is the type of drift, and what are the best classification accuracies achievable on each concept. The main limitation of synthetic data is uncertainty about whether corresponding drifts happen in reality.

Next, we present seven popular models for generating synthetic data with concept drift; their characteristics are summarized in Table VI.

SEA Concepts Generator. Street and Kim [2001] model abrupt (real) concept drifts. There are three independent real valued attributes in $[0, 10]$; only the first two attributes are relevant for prediction. The original data model can produce four different concepts. The class decision boundary is defined as $x_1 + x_2 \leq \theta$, where

ALGORITHM 6: The `AddExp` algorithm for discrete classes

input: X, y^T A Training Set with class $y \in Y$
$\beta \in [0, 1]$: factor for decreasing weights; $\tau \in [0, 1]$: loss required to add a new expert
begin
  Set the initial number of experts: $N_1 \longleftarrow 1$;
  Set the initial expert weight: $w_{1,1} \longleftarrow 1$;
  for $t \leftarrow 1$ **to** T **do**
    Get expert predictions: $\epsilon_{t,1}, \ldots, \epsilon_{t,N_t} \in Y$;
    Compute prediction: $\hat{y}_t = argmax_{c \in Y} \sum_{i=1}^{N_t} w_{t,i}[c = \epsilon_{t,i}]$;
    Update experts weights: $w_{t+1,i} \longleftarrow w_{t,i}\beta^{[y_t \neq \epsilon_{t,i}]}$;
    if $\hat{y}_t \neq y_t$ **then**
      Add a New Expert;
      $N_{t+1} \longleftarrow N_t + 1$;
      $w_{t+1,N_{t+1}} \longleftarrow \gamma \sum_{i=1}^{N_t} w_{t,i}$;
    end
    Train each expert on instance X_t, y_t;
  end
end

ALGORITHM 7: The `AddExp` algorithm for continuous classes

input: X, y^T A Training Set with class $y \in [0 : 1]$; $\beta \in [0, 1]$: factor for decreasing weights
$\gamma \in [0, 1]$: factor for new expert weight; $\tau \in [0, 1]$: loss required to add a new expert
begin
  Set the initial number of experts: $N_1 \longleftarrow 1$;
  Set the initial expert weight: $w_{1,1} \longleftarrow 1$;
  for $t \leftarrow 1$ **to** T **do**
    Get expert predictions: $\epsilon_{t,1}, \ldots, \epsilon_{t,N_t} \in [0, 1]$;
    Compute prediction: $\hat{y}_t = \frac{\sum_{i=1}^{N_t} w_{t,i}\epsilon_{t,i}}{\sum_{i=1}^{N_t} w_{t,i}}$;
    Suffer loss $\|\hat{y}_t - y_t\|$;
    Update experts weights: $w_{t+1,i} \longleftarrow w_{t,i}\beta^{\|\epsilon_{t,i} - y_t\|}$;
    if $\|\hat{y}_t - y_t\| \geq \tau$ **then**
      Add a New Expert;
      $N_{t+1} \longleftarrow N_t + 1$;
      $w_{t+1,N_{t+1}} \longleftarrow \gamma \sum_{i=1}^{N_t} w_{t,i}\|\epsilon_{t,i} - y_t\|$;
    end
    Train each expert on instance X_t, y_t;
  end
end

Table VI. Models for Synthetic Data Generation

Name	# of concepts	Task	Type of drift: Real CD	Type of drift: Virtual CD	Priors
SEA	4	classification (2)	✓	–	✓
STAGGER	3	classification (2)	✓	–	✓
Rotating hyperplane	any	classification (2)	✓	–	–
RBF generator	any	classification (any)	✓	✓	✓
Function generator	10	classification (2)	✓	–	✓
LED generator	–	classification (10)	–	✓	–
Waveform generator	–	classification (3)	–	✓	–

x_1 and x_2 are the first two attributes and θ is a threshold value different for each concept: (1) $\theta = 9$, (2) $\theta = 8$, (3) $\theta = 7$, and (4) $\theta = 9.5$.

STAGGER Concepts Generator. Schlimmer and Granger [1986] model abrupt (real) concept drifts. There are three independent categorical attributes: size $\in$ small, medium, large; color $\in$ red, green, blue; and shape $\in$ square, circular, triangular. A binary classification task is defined by a disjunct of conjuncts. There are three concepts: (1) positive class *if* size = small *and* color = red, (2) color = green *or* shape = circular, and (3) size = medium *or* size = large.

Rotating Hyperplane was first used to test CVFDT against VFDT in Hulten et al. [2001]. A generic model is as follows. Data is generated uniformly in a hyperplane d-dimensional real space. The decision boundary is defined as $\sum_{i=1}^{d} w_i x_i = w_0 = \sum_{i=1}^{d} w_i$, where x_i is the i^{th} attribute. Examples for which $\sum_{i=1}^{d} w_i x_i \geq w_0$ are labeled positive, and examples for which $\sum_{i=1}^{d} w_i x_i < w_0$ are labeled negative. Concept changes are introduced by modifying the weights w_i, which need to satisfy the constraint $w_0 = \sum_{i=1}^{d} w_i$ to keep the prior probabilities fixed. Hyperplanes are very flexible. They can be used for simulating various kinds of concept drifts (e.g., abrupt, gradual, reoccurring) by changing the orientation and position of the hyperplane via weights w_i. Noise can be added by randomly swapping class labels.

Random RBF Generator [Bifet et al. 2009] was devised to offer an alternate complex concept type that is not straightforward to approximate with a decision tree model. A fixed number of random centroids are generated in d-dimensional real space. Each center is characterized by a random position, a single standard deviation, class label, and prior probability (weight). New examples are generated by picking a center at random with a given prior probability. The direction of an offset is chosen at random from the centroid following the normal distribution with zero mean and a given standard deviation. The example is labeled the same as the centroid. Concept drift is introduced by moving the centroids around over time.

Function Generator [Agrawal et al. 1992] used to be a popular data model for early work on scaling up decision tree learners [Agrawal et al. 1993; Mehta et al. 1996; Shafer et al. 1996; Gehrke et al. 2000]. The generator produces a stream containing nine attributes, six numeric and three categorical. Although not explicitly stated by the authors, a sensible conclusion is that these attributes describe hypothetical loan applications. There are 10 functions defined for generating binary class labels from the attributes, presumably meaning approval of the loan. The original data model has no drift. Concept drift may be introduced by switching between labeling functions over time.

LED Generator originates from the CART book [Breiman et al. 1984] and an implementation in C is available from the UCI repository [Bache and Lichman 2013]. The goal is to predict the digit displayed on a seven-segment LED display, where each attribute has a 10% chance of being inverted. The original data model has no drift. Drift may be introduced by swapping the positions of attributes.

Waveform Generator originates from the CART book [Breiman et al. 1984] and is available from the UCI repository [Bache and Lichman 2013]. The classification task is to distinguish between three classes of a waveform, each of which is generated from a combination of two or three base waves. There are two versions of the problem: the first one has 21 numeric attributes, all of which include noise; the second one has the same 21 attributes and, in addition, 19 irrelevant attributes. The original data model has no drift. Drift may be introduced by swapping the positions of attributes.

Implementations of these data models are available in MOA [Bifet et al. 2011].

B.2. Real-World Data

Nowadays, it is easier than in the past to find large real-world datasets for public benchmarking with concept change. The UCI machine learning repository [Bache and Lichman 2013] contains some real-world benchmark data for evaluating machine learning techniques; however, they are not large datasets.

The main real data domains of the large datasets mentioned in the *Concept Drift* page of Wikipedia[7] are the following ones.

Text Mining. Documents of text that contain words or combinations of words as the features to use to process the data. A collection of text mining datasets with concept drift is maintained by I. Katakis.[8] Data recollected from Twitter may also be considered for text mining.

Electricity. A widely used dataset is the Electricity Market Dataset introduced in Harries [1999]. This time-series-based data was collected from the Australian New South Wales Electricity Market, available from J. Gama.[9] In this market, the prices are not fixed and are affected by demand and supply of the market. The prices in this market are set every 5 minutes. The *ELEC2* dataset contains 45, 312 instances. Each example of the dataset refers to a period of 30 minutes (i.e., there are 48 instances for each time period of 1 day). The class label identifies the change of the price related to a moving average of the last 24 hours. The class level only reflects deviations of the price on a 1-day average and removes the impact of longer-term price trends.

Email Spam. Datasets of email messages are used to predict if they are unsolicited messages or not. For example, *ECUE Spam* datasets are two datasets each consisting of more than 10,000 emails collected over a period of approximately 2 years referring to one user compiled by S.J. Delany.[10]

Business Oriented. Datasets contain data used in decision management systems in companies. The *PAKDD'09 competition* dataset[11] is used for a credit evaluation task. It is collected over a 5-year period; unfortunately, the true labels are released only for the first part of the data. Another dataset called *Airline* contains approximately 116 million flight arrival and departure records (cleaned and sorted) and is compiled by E. Ikonomovska.[12]

Games. Datasets are obtained from online or nononline games: for example, a dataset *Chess.com* (online games) compiled by I. Žliobaitė.[13]

C. EVALUATION EXAMPLE

We give an example on evaluation of data stream classification using the MOA software framework [Bifet et al. 2011]. MOA is an open-source framework for dealing with massive evolving data streams. MOA is related to WEKA [Hall et al. 2009], the Waikato Environment for Knowledge Analysis, which is an award-winning open-source workbench containing implementations of a wide range of batch machine-learning methods.

MOA enables evaluation of data stream classification algorithms on large streams, in the order of tens of millions of instances under explicit memory limits. Any less than this does not actually test data stream algorithms in a realistically challenging setting.

[7]http://en.wikipedia.org/wiki/Concept_drift retrieved 12/11/2012.
[8]http://mlkd.csd.auth.gr/concept_drift.html.
[9]http://www.liaad.up.pt/kdus/products/datasets-for-concept-drift.
[10]http://www.comp.dit.ie/aigroup/?page_id=729.
[11]http://sede.neurotech.com.br:443/PAKDD2009/arquivo.do?method=load.
[12]http://kt.ijs.si/elena_ikonomovska/data.html.
[13]https://sites.google.com/site/zliobaite/resources-1.

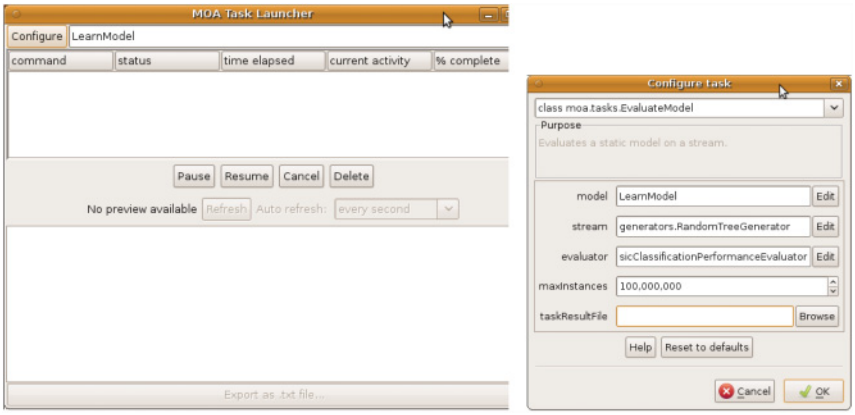

Fig. 10. MOA graphical user interface.

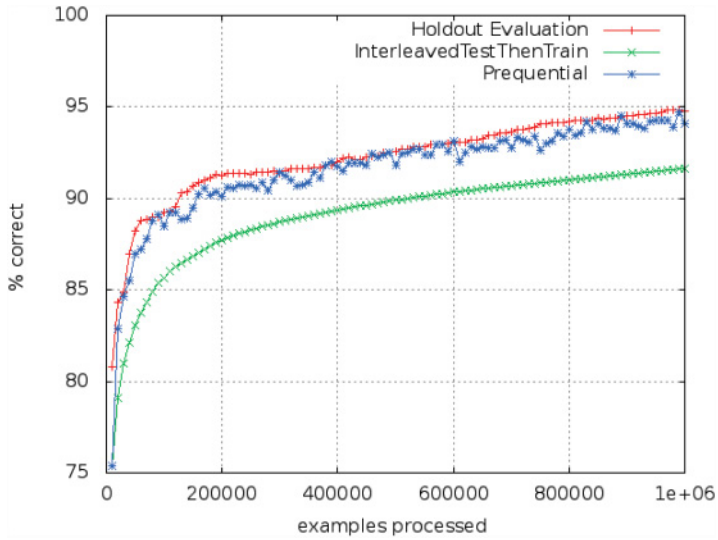

Fig. 11. Evaluation for a stream of a million of instances comparing holdout, prequential with landmark window, and prequential with sliding window.

MOA is written in Java. The main benefits of Java are portability, where applications can be run on any platform with an appropriate Java virtual machine, and the strong and well-developed support libraries. Use of the language is widespread, and features such as automatic garbage collection help to reduce programmer burden and error.

MOA contains stream generators, classifiers, and evaluation methods. Figure 10 shows the MOA graphical user interface. A command line interface is also available.

Considering data streams as data generated from pure distributions, MOA models a concept drift as a weighted combination of two pure distributions that characterizes the target concepts before and after the drift. Within the framework, it is possible to define the probability that instances of the stream belong to the new concept after the drift using the sigmoid function, as an elegant and practical solution [Bifet et al. 2009].

MOA contains the popular data generators described in online Appendix B. MOA streams can be built using generators, reading ARFF files, joining several streams, or

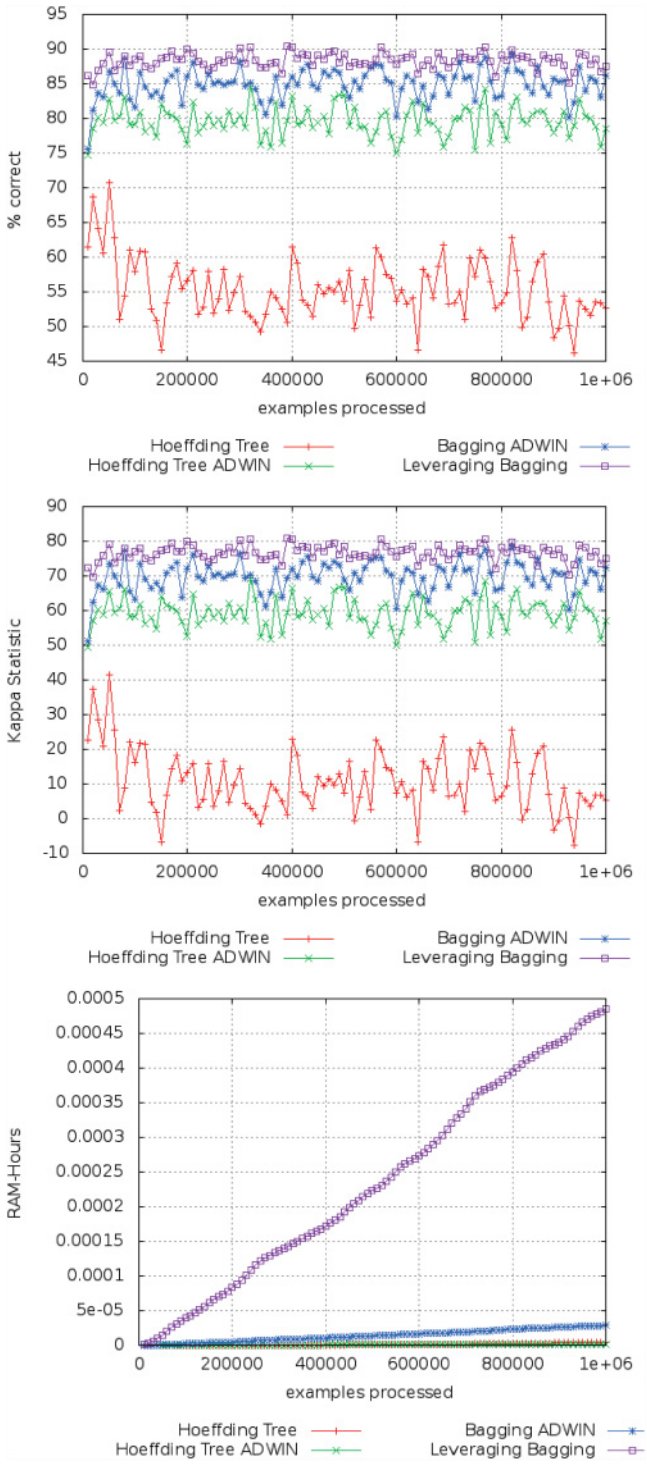

Fig. 12. Prequential evaluation for an RBF stream of a million of instances where the centers are moving with a speed of 10^{-4}.

filtering streams. They allow for the simulation of a potentially infinite sequence of data. The following generators are currently available: Random Tree Generator, SEA Concepts Generator, STAGGER Concepts Generator, Rotating Hyperplane, Random RBF Generator, LED Generator, Waveform Generator, and Function Generator.

MOA contains several classifier methods such as Naive Bayes, Decision Stump, Hoeffding Tree, Hoeffding Option Tree, Adaptive Hoeffding Tree, Bagging, Boosting, Bagging using ADWIN [Bifet et al. 2009], and Leveraging Bagging [Bifet et al. 2010].

Figure 11 shows a comparison between a holdout evaluation, a prequential evaluation using a landmark window (*InterleavedTestThenTrain*), and a prequential evaluation using a sliding window of size 1,000. We observe that the prequential evaluation using a sliding window is a good approximation to the holdout evaluation.

We run also on MOA the following experiment simulating a concept drift scenario: a prequential evaluation using a sliding window of size 1,000 for a stream of a million of instances generated by the Random RBF Generator, with the following learners: Hoeffding Tree, Adaptive Hoeffding Tree, and ADWIN Bagging and Leveraging Bagging. The stream is evolving and the centroids are moving with constant speed of 10^{-4}: this speed is defined as the distance moved when each new instance arrives and it is initialized by a drift parameter.

Figure 12 shows accuracy, kappa-statistic, and RAM-hours for this experiment. We observe that the Hoeffding Tree is the method with a lower capacity of adaptation. Ensemble methods perform better than single classifiers, but they have a higher cost in RAM-hours. Leveraging Bagging is the method with higher accuracy and kappa-statistic, but the number of resources that it needs is considerably larger. Data stream evaluation is a two-dimensional process with a tradeoff between accuracy results and resource costs.