A Survey on Ensemble Learning for Data Stream Classification

HEITOR MURILO GOMES, JEAN PAUL BARDDAL, and FABRÍCIO ENEMBRECK,
Pontícia Universidade Católica do Paraná
ALBERT BIFET, Institut Mines-Télécom, Télécom ParisTech, Université Paris-Saclay

Ensemble-based methods are among the most widely used techniques for data stream classification. Their popularity is attributable to their good performance in comparison to strong single learners while being relatively easy to deploy in real-world applications. Ensemble algorithms are especially useful for data stream learning as they can be integrated with drift detection algorithms and incorporate dynamic updates, such as selective removal or addition of classifiers. This work proposes a taxonomy for data stream ensemble learning as derived from reviewing over 60 algorithms. Important aspects such as combination, diversity, and dynamic updates, are thoroughly discussed. Additional contributions include a listing of popular open-source tools and a discussion about current data stream research challenges and how they relate to ensemble learning (big data streams, concept evolution, feature drifts, temporal dependencies, and others).

CCS Concepts: • **Computing methodologies** → **Ensemble methods**; *Online learning settings*; Supervised learning by classification;

Additional Key Words and Phrases: Ensemble learning, supervised learning, data stream classification

ACM Reference Format:
Heitor Murilo Gomes, Jean Paul Barddal, Fabrício Enembreck, and Albert Bifet. 2017. A survey on ensemble learning for data stream classification. ACM Comput. Surv. 50, 2, Article 23 (March 2017), 36 pages.
DOI: http://dx.doi.org/10.1145/3054925

1. INTRODUCTION

The amount of data generated by smart phones, social networks, and all kinds of sensors has grown tremendously. All these data are only useful if efficiently processed so individuals can make timely decisions based on them. Recently, a lot of progress has been made towards obtaining useful models from massive amounts of rapidly generated data under the research area of data stream mining.

Data streams pose several challenges for learning algorithms, including, but not limited to, concept drifts [Tsymbal 2004], temporal dependencies [Žliobaitė et al. 2015], massive amount of instances, limited labeled instances, novel classes, feature drifts [Barddal et al. 2016], and restricted resources (time and memory) requirements. On top of that, problems found in a batch-learning setting are also present in a data stream context, for example, absent values, overfitting, noise, irrelevant features, class imbalance, and others.

In recent years, ensembles of learners have been widely studied and deployed in real-world problems. Dietterich [2000] provided three reasons that justify using ensembles instead of single learners, that is, statistical, computational, and representational.

Authors' addresses: H. M. Gomes, J. P. Barddal, and F. Enembreck, Imaculada Conceição Street, 1155, Curitiba, Brazil; emails: {hmgomes, jean.barddal, fabricio}@ppgia.pucpr.br; A. Bifet, 46 rue Barrault, Paris, France; email: albert.bifet@telecom-paristech.fr.

DOI: http://dx.doi.org/10.1145/3054925

Another explanation for this preference is the difficulty of obtaining a strong learner, while a set of weak learners is relatively easy to develop and can effectively be boosted into a strong learner [Freund and Schapire 1997], as long as they are trained and combined strategically. Ensemble learners are popular in the data stream setting, because, besides leveraging weak learners, they can be used to handle general machine-learning problems as well as data-stream-specific challenges. For instance, ensemble learners have been applied to address drifting concepts [Kolter et al. 2003; Bifet et al. 2009, 2010a; Gomes and Enembreck 2014; Dongre and Malik 2013; Deckert 2011; Elwell and Polikar 2011], noisy data [Zhu et al. 2004; Zhang et al. 2011b], recurrent concepts [Nishida et al. 2005; Ramamurthy and Bhatnagar 2007; Katakis et al. 2010; Jaber et al. 2013a; Gonçalves Jr and de Barros 2013], novel class detection [Masud et al. 2010; Parker et al. 2012; Parker and Khan 2013, 2015], and feature drift [Nguyen et al. 2012].

There are a number of surveys and books focusing on data stream learning, ranging from works that cover machine learning in general [Aggarwal 2007; Gama and Gaber 2007; Gama 2010; Hoens et al. 2012; Ditzler et al. 2015] to those that are specific to supervised learning [Lemaire et al. 2015], clustering [Silva et al. 2013], or concept drift [Tsymbal 2004; Hoens et al. 2012; Žliobaitė 2010; Gama et al. 2014; Webb et al. 2016]. Many of these works refer to ensemble methods as an option for data stream learning, especially those that exhibit concept drifts. However, these works' focus is on either a more general problem (e.g., learning from data streams) or a specific machine-learning task (e.g., unsupervised learning). Works addressing ensembles for batch-learning environments are also abundant [Kuncheva 2004b; Brown et al. 2005; Polikar 2006; Sewell 2008; Rokach 2009, 2010; Zhou 2012; Woźniak et al. 2014]; however, conclusions in these works are made assuming that learning is performed on static datasets. Finally, surveys on ensemble classifiers do exist for data stream learning [Fern and Givan 2003; Kuncheva 2004a, 2008], but recent findings and developments on the field justify an updated review.

The particularities of the stream setting change how ensemble methods are used and explored. For example, ensemble classifiers are considered the most popular evolving technique for handling concept drift [Žliobaitė 2010], such that very often drift adaptation is achieved by assigning different weights to each of the ensemble members [Street and Kim 2001; Polikar et al. 2001; Wang et al. 2003; Kolter et al. 2003; Kolter and Maloof 2005; Brzeziński and Stefanowski 2011; Zhang et al. 2011b; Deckert 2011]. This "simple" weighting may combine a multitude of stream processing and ensemble techniques, such as a temporally aware weighting function, a voting method that highlights classifiers adapted to the current concept, a training method that maintains diversity, periodic updates that selective reset/remove or add new classifiers, and many others.

In this work, our goal is to clarify the characteristics involving the application of ensemble learners on a data stream context. To achieve this goal, we propose a taxonomy that organizes general techniques, present a classification of over 60 ensemble algorithms according to our taxonomy, and discuss current and future trends for ensemble learning on a stream setting, including big data stream processing. There are many intersections between ensemble learning on static datasets with that of dynamic data streams. Our proposed taxonomy outlines these resemblances and highlights characteristics from ensemble learning that are unique (and very useful) to the data stream learning setting. Our focus is on data stream classification; however, some of the concepts presented may be extended for regression and perhaps other machine-learning tasks where ensemble learning is relevant.

The remainder of this work is organized as follows. In Section 2, we briefly define the challenges, characteristics, and assumptions related to data stream classification. In Section 3, we present our taxonomy and discuss each of its dimensions. Section 4

presents a high-level discussion about the current state of research on ensemble classifiers for data streams and directions for future research. Finally, Section 5 concludes this survey.

2. DATA STREAM CLASSIFICATION

Data stream classification is a variation of the traditional supervised machine-learning task of classification. Both tasks are concerned with the problem of predicting a nominal value of an unlabeled instance represented by a vector of characteristics. The main difference between these tasks is that, in streaming scenarios, instances are not readily available to the classifier as being part of a large static dataset, and, instead, instances are provided sequentially and rapidly over time as a continuous data stream. Therefore, a data stream classifier must be ready to deal with a great number of instances, such that each instance can only be inspected once or stored for only a short period of time.

In this work, we assume that instances from a data stream S appear as a sequence, in intervals of u time units, of unlabeled instances x^t, where x^t represents a vector of attribute values that arrived at time t. Besides our discussion on partially labeled classification in Section 4, hereinafter it is assumed that the true class label y^t of a given instance x^t is available before the arrival of instance x^{t+1}, and thus the classifier can use it for training immediately after it has been used for testing. These assumptions are commonly used when dealing with data stream classification [Oza 2005; Bifet et al. 2009, 2010a; Gomes and Enembreck 2013, 2014; Barddal et al. 2014; Brzezinski and Stefanowski 2014] and on data stream analysis frameworks, for example, on the Massive Online Analysis (MOA) framework [Bifet et al. 2010b].

In a traditional batch setting, learning is performed over a finite dataset where data distribution is unknown yet stationary. Thus, despite the complexity of learning the model, after training is completed there is no need to update the model. A more general setting assumes that the data distribution changes over time, a phenomenon commonly identified as *concept drift* [Tsymbal 2004]. Concept drifts may be caused by variations that are outside the scope of the data presented to the learning algorithm, that is, changes take place in a "hidden" context that encompasses the learning task. Data streams that exhibit concept drifts are referred to as evolving or non-stationary data streams. Most existing data stream classifiers employ some technique to detect and adapt to concept drifts either implicitly or explicitly. An important characteristic of concept drift relates to the rate at which it happens. The rate of a drift can be abrupt, incremental, gradual, or recurring. Noise or outliners ought not be confused with drift, such that the difference between these and drifts is persistence. Abrupt drifts are easily recognizable, because the prediction error and the data distribution vary greatly in a short period of time. Gradual drifts are characterized by a transitioning window where instances from the previous concept are less frequent, while instances from the new concept become predominant. Incremental drifts represent concepts that slowly evolve over time, similarly to an extreme case of gradual drift where the window of change corresponds to the whole stream. Recurring drift happens whenever concepts keep recurring either periodically or erratically. For further details regarding concept drifts and adaptation, we refer readers to Gama et al. [2014] and Webb et al. [2016].

Data streams may exhibit *temporal dependencies* between class labels. Temporal dependence is also encountered in *time-series analysis*, where previous signal values present the main (sometimes, the only) source of predictive information. In data stream learning, the predictive information is represented by an input feature set, and temporal dependencies can help to determine how these features relate with each other over time. In the context of learning from data streams, temporal dependencies were recently studied first in Bifet et al. [2013] and later in Žliobaitė et al. [2015], where it

was advised to consider temporal dependencies while designing and evaluating data stream classifiers. Formally, a temporal dependence occurs whenever the current instance label y^t is influenced by previous instances labels $(y^{t-1}, y^{t-2}, \dots)$. To date, only one method, along with an evaluation metric, that takes into account temporal dependencies for data streams has been proposed, both in Žliobaitė et al. [2015].

3. A TAXONOMY OF ENSEMBLE LEARNING FOR DATA STREAM CLASSIFICATION

An ensemble can be described as a composition of multiple weak learners to form one with (expected) higher predictive performance (strong learner), such that a weak learner is loosely defined as a learner that performs slightly better than random guessing [Freund and Schapire 1997]. One of the main goals that researchers pursue while designing an ensemble is to permit that each of the ensemble members be as unique as possible, particularly with respect to misclassifications [Polikar 2006]. If an ensemble is composed of classifiers that misclassify different instances, complementing each other, then its members are said to be "diverse," and the *combination* of their predictions might achieve performance above any of them individually. *Combination* is a term often referred to in the ensemble literature as a synonym for voting. Furthering this concept, we employ the term differently, such that combination may refer to either the ensemble architecture or the voting method employed. Concretely, the ensemble architecture refers to how classifiers are organized within the ensemble, while the voting method specifies how their predictions are used to form the overall ensemble prediction.

It is not possible to guarantee that enhancing ensemble diversity will boost its classification performance in practice [Kuncheva 2003; Kuncheva and Whitaker 2003]. Efforts to provide consistent theoretical evidence on how diversity and combination methods relate to the overall ensemble accuracy were neither conclusive nor general [Kuncheva et al. 2003; Kuncheva and Rodríguez 2014]. Even though the research community still lacks a general proof on this subject, it has not prevented the field of ensemble learning from becoming one of the most active research topics in machine learning. As a consequence, many taxonomies and classifications [Brown et al. 2005; Kuncheva 2004b; Rokach 2009] have been proposed to organize research around ensemble learning in a batch-learning context. These works tend to revolve around combination and diversity, along with other dimensions, such as classifier dependency and ensemble size.

We propose a taxonomy that is focused on ensemble learning for data streams. Besides arranging ensemble-related techniques based on diversity, base learner, and combination, we discuss characteristics that influence the ensemble composition that are unique to data stream learning, which we refer to as "update dynamics." This part of the taxonomy represents important methods for stream learning, for example, strategies to cope with drifts, how learning is performed, and when to remove or add classifiers. An overview of the proposed taxonomy is depicted in Figure 1. Algorithms may instantiate techniques from several nodes of our taxonomy, thus one shall not expect to classify algorithms as a leaf node in the taxonomy. The taxonomy organizes general aspects related to algorithms in a data stream learning setting. Some of these aspects, when directly mapped as characteristics of an actual algorithm, are better represented as values rather than dimensions, for example, Cardinality corresponds to a dimension, while fixed and dynamic are values. To classify existing algorithms according to our taxonomy, we present Table I with a summarized view of over 60 ensemble algorithms, including classic algorithms (e.g., OzaBag, DWM, Streaming Ensemble Algorithm (SEA)) as well as less widely known or novel methods (e.g., M^3, BLAST (Best Last), HSMiner (Hierarchical Stream Miner), SAE2 (Social Adaptive Ensemble 2).

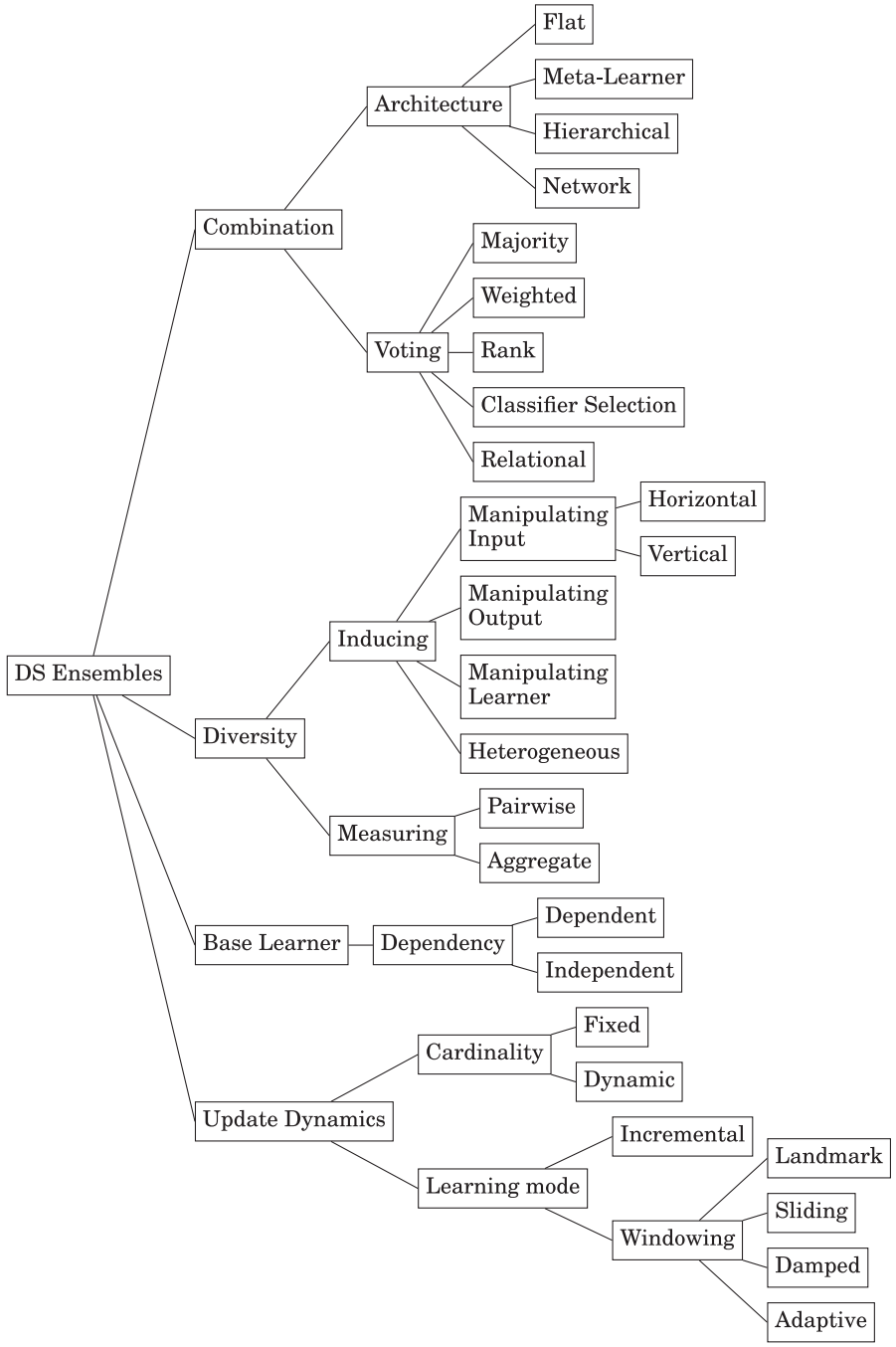


Fig. 1. A taxonomy of data stream ensemble classifiers.

Table I. Data Stream Ensemble Classifiers According to Our Taxonomy

Algorithm	(A)	(V)	(DI)	(B)	(D)	(C)	(L)	Reference
SEA	f	m	t	b	i	m	l	Street and Kim [2001]
Learn^{++}	f	w	v	b	d	f	l	Polikar et al. [2001]
AWE	f	r	t	b	i	f	l	Wang et al. [2003]
CDC	f	w	t	i	i	m	i	Stanley [2003]
FLBoost	f	w	v*	b	d	f	l	Chu and Zaniolo [2004]
CBEA	f	s,m	t	b	i	m	l	Rushing et al. [2004]
AO-DCS	f	s	v	b	i	f	l	Zhu et al. [2004]
OzaBag	f	m	v	i	i	f	i	Oza [2005]
OzaBoost	f	w	v*	i	d	f	i	Oza [2005]
AddExpert	f	w	t	b	i	f	l	Kolter and Maloof [2005]
ACE	f	w	t	i,b	i	d	l,i	Nishida et al. [2005]
FAE	f	w	t,h	i	i	d	i	Wenerstrom and Giraud-Carrier [2006]
DWM	f	w	t	b	i	d	l	Kolter et al. [2003]
BoostDC	f	w	v*	b	d	f	l	Scholz and Klinkenberg [2007]
ICEA	f	w	t	i	i	m	i	Yue et al. [2007]
RDE	f	w	t	b	i	d	l	Ramamurthy and Bhatnagar [2007]
Streaming RF	f	m	h	i	i	f	i	Abdulsalam et al. [2007]
Dynamic SRF	f	m	h	i	i	f	i	Abdulsalam et al. [2008]
MCIK-Ensemble	f	w	v	i,b	i	f	l	Masud et al. [2008]
ASHT Bag	f	w	l	i	i	f	a	Bifet et al. [2009]
ADWIN Bag	f	m	v	i	i	f	a	Bifet et al. [2009]
OVA Trees	f	w	o,v	i	i	f	i	Hashemi et al. [2009]
OCBoost	f	w	v*	i	d	f	i	Pelossof et al. [2009]
Learn^{++}.NC	f	w,re	t	b	i	f	l	Muhlbaier et al. [2009]
FISH	f	s	v	i	d	f	a	Žliobaitė [2009]
LevBag	f	m	v,o[1]	i	i	f	a	Bifet et al. [2010a]
CCP	f	s	v	i,b	i	d[2]	l	Katakis et al. [2010]
Learn^{++}.UDNC	f	w,re	t	i	i	f	l	Ditzler et al. [2010]
DXMiner	f	w	t	i,b	i	f	l	Masud et al. [2010]
ONSBoost	f	w	v*	i	d	f	l	Pocock et al. [2010]
AUE	f	r	v,t	i	i	f	l	Brzeziński and Stefanowski [2011]
AE	f	w	he,v	i,b	i	f	l	Zhang et al. [2011b]
BWE	f	w	t	b	i	f	a	Deckert [2011]
Learn^{++}.NSE	f	w	t	b	i	f	l	Elwell and Polikar [2011]
DDD	m	w	v,l	b	i	f	a	Minku and Yao [2012]
RestrictedHF	m	w	v,h	i	d	f*	a	Bifet et al. [2012]
HEFT-Stream	f	w	he,h,v	i	i	f	l	Nguyen et al. [2012]
AEBC	f	w	v	i	d	f	a	Wankhade et al. [2012]
HSMiner	h	w	he,h,o	i	h	f*	l	Parker et al. [2012]
ChenBoost	f	w	v*	i	d	f	l	Chen et al. [2012]
Woo	f	w	v	i	i	d	l	Ryu et al. [2012]
OOB	f	m	v	i	i	f	i	Wang et al. [2013]
UOB	f	m	v	i	i	f	i	Wang et al. [2013]
SAE	n	m	v,t	i	h	d	l	Gomes and Enembreck [2013]
DACC	f	m	t	b,i	i	f	l	Jaber et al. [2013b]
ADACC	f	m	t	b,i	i	f	a	Jaber et al. [2013a]
SluiceBox	h	w	he,h,o	i	i	f	l	Parker and Khan [2013]
Learn^{++}.CDS	f	w	t	i	i	f	l	Ditzler and Polikar [2013]
Learn^{++}.NIE	f	w	v,t	i	i	f	l	Ditzler and Polikar [2013]
RCD	f	m	t	i	i	m	l	Gonçalves Jr and de Barros [2013]
OAUE	f	r	t	i	i	f	l,i	Brzezinski and Stefanowski [2014]
SFNC	n	w	t	i	i	m	l	Barddal et al. [2014]
SAE2	n	w	v,t	i	h	m	l	Gomes and Enembreck [2014]

(Continued)

Table I. Continued

Algorithm	(A)	(V)	(DI)	(B)	(D)	(C)	(L)	Reference
***M*[3]**	f	w	he[3]	i	i	f	l	Parker et al. [2014]
SE-PLS	h	w	v	i	i	f	i	Sethi et al. [2014]
Fast-AE[6]	f	w	t	i	i	m	l	Ortíz Díaz et al. [2015]
IBEP	f	w	v	i	i	f	l	Zhi et al. [2015]
PA/PP	f	re	v	i	i	f	l	Gomes et al. [2015]
Online BBM.W	f	w	v*	i,b	d	f	i	Beygelzimer et al. [2015]
AdaBoost.OL.W	f	w	v*	i,b	d	f	i	Beygelzimer et al. [2015]
SluiceBox-AM	h	w	he,h,o	i	i	m	i	Parker and Khan [2015]
WEOB	f	w	v	i	i	f	i	Wang et al. [2015]
EDTC	f	m	h	i	i	f	i,s	Li et al. [2015]
BLAST	m	w	he	i	i	f	i	van Rijn et al. [2015]
MOOB	f	r	v	i	i	f	i	Wang et al. [2016]

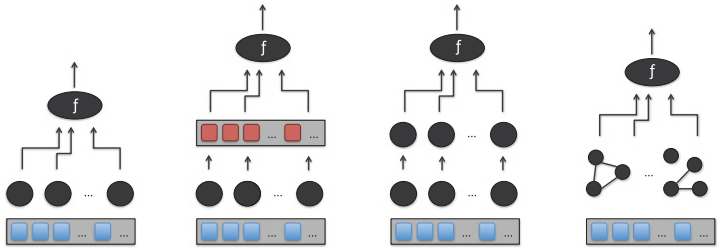
Legends. **(A) Architecture**: *f*: flat, *m*: meta-learner, *h*: hierarchical, *n*: network; **(V) Voting**: *m*: majority, *w*: weighted, *r*: rank, *s*: classifier selection, *re*: relational; **(DI) Diversity Inducer**: *he*: heterogeneous, *l*: learner manipulation, *v*: vertical input, *v**: vertical input with instance weighting, *h*: horizontal input, *o*: output, *t*: time based; **(B) Base Learner**: *b*: batch, *i*: incremental; **(D) Dependency**: *d*: dependent, *i*: independent, *h*: hybrid; **(C) Cardinality**: *f*: fixed, *f**: fixed (derived from feature set), *m*: maximum, *d*: dynamic; **(L) Learning Mode**: *s*: sliding window, *l*: landmark window, *d*: damped window, *a*: adaptive window, *i*: incremental.
[1]It is possible to use output correction codes.
[2]Varies according to the number of clusters found.
[3]Member weights are used for weighted training diversification.

The following sections present each dimension of the taxonomy. Generally, it is difficult to clearly separate different methods into a taxonomy, since some areas are blurred and grouped together by different algorithms. To facilitate the understanding of existing methods and their representatives (i.e., algorithms), in each section we comment on their usage in different algorithms.

3.1. Combination

Combining ensemble members' predictions can either enhance the overall performance or jeopardize it. Through an appropriate combination method, it is expected to correctly predict the class label of difficult-to-classify instances. However, a lot of effort has been dedicated to the generation of diverse sets of classifiers and not so much on methods to combine classifiers outputs [Tulyakov et al. 2008]. For example, the original bagging algorithm [Breiman 1996] emphasizes the construction of a diverse ensemble, while its combination method is a simple majority vote. There are many approaches to handle voting for ensemble classifiers, varying from simple methods (e.g. majority vote) to more complex approaches (e.g., Behavior-Knowledge Space [Huang and Suen 1993]).

In this work, we differentiate between the voting method and how ensemble members are arranged (architecture). Very often ensemble members' arrangement and voting method are intrinsically related, thus while explaining the inner workings of a specific algorithm, it is sometimes much clearer to not separate them into different blocks. However, while analysing multiple ensemble methods, it is reasonable to differentiate between voting and ensemble architecture, since it facilitates the understanding of algorithms in which the ensemble members' arrangement is not trivial. For example, SAE (Social Adaptive Ensemble) [Gomes and Enembreck 2013], SAE2 [Gomes and Enembreck 2014], and SFNC (Scale-Free Network Classifier) [Barddal et al. 2014] arrange the ensemble members into a network (graph) structure, while the ensemble of Restricted Hoeffding trees [Bifet et al. 2012] uses a meta-level combiner trained on the outputs of a set of decision trees trained on the input data. Also, there are situations in which the architecture can be used for more than voting; for example, it can be used to control the ensemble training [Chan and Stolfo 1995] or to enable operators, for example, a redundant classifier removal method [Gomes and Enembreck 2013, 2014].



(a) Flat (b) Meta-Learner (c) Hierarchical (d) Network

Fig. 2. Ensemble structural arrangement (circles = classifiers, squares = instances).

In the following sections, we discuss different architectures and voting methods for ensemble learning in a data stream context.

3.1.1. Architecture. The ensemble architecture defines how classifiers interact with one another. In Jain et al. [2000], the authors organize ensemble methods, for batch learning, into three different architectures: parallel, cascading, and hierarchical. In the parallel architecture, each classifier output is aggregated by a combiner, such that the combiner can be a simple linear function (e.g., weighted vote) or another classifier (e.g., stacking [Wolpert 1992]). Cascading includes architectures in which the output of one classifier is the input of another for multiple levels [Alpaydin and Kaynak 1998; Gama and Brazdil 2000]. Finally, an ensemble in which members are arranged in a treelike structure is classified as hierarchical. Our taxonomy for ensemble architecture differs from that presented in Jain et al. [2000], as we split parallel into two different architectures, combine cascading and hierarchical into one, and include a new architecture. Concretely, we classify a given ensemble structure as flat (parallel with simple combiner), meta-learner (parallel with meta-learners), hierarchical (cascading or treelike structure), or network (graph structure). Figure 2 presents a schematic view of these four different structural arrangements of ensembles.

Flat. The flat structure assumes that base models are trained on the input data and the decision fusion is delegated to a simple combination function (voting scheme) such as majority voting. In comparison to other arrangements this is the most widely used, partly because it is simple, but also because it makes fewer assumptions about individual classifiers. Examples of data stream ensemble classifiers that employ a flat structure includes: Online Bagging [Oza 2005], DWM [Kolter et al. 2003], Leveraging Bagging [Bifet et al. 2010a], and many others.

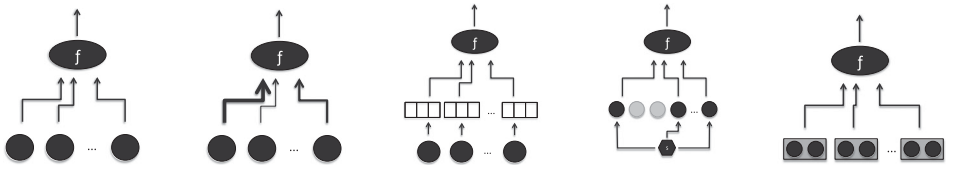
Meta-learner. In a meta-learning structure, the combiner (meta-learner) is trained on meta-data, which may refer to properties of the learning problem [Minku and Yao 2012] or to the outputs of learners trained on the input data [Bifet et al. 2012]. From a high-level perspective, the flat and meta-learner approaches may seem very similar; however, they are effectively differ, since the latter involves creating a meta-dataset and training a meta-learner on it. Despite meta-learning being feasible without an ensemble structure [Brazdil et al. 2008; Gama and Kosina 2009], in this work we focus on meta-learning for ensemble classifiers. A canonical example of meta-learning is the stacking algorithm [Wolpert 1992]. Stacking creates a meta-dataset where every meta-instance corresponds to an instance in the original dataset. This meta-dataset replaces the original instances' inputs by the predictions of each ensemble member, while the class label remains the original. A meta-classifier is induced from the meta-dataset, which during predictions is responsible for combining component classifiers predictions into a final one. An example of stacking for data stream classification is the ensemble of Restricted Hoeffding trees [Bifet et al. 2012], where the first level of learners is

composed of Hoeffding Trees, while the meta-learner level is formed by perceptrons (one per class label).

Hierarchical. A hierarchical ensemble imposes a treelike structure or a strict order (cascading) over its members. Examples of batch ensembles in which the structural arrangement adheres to this definition include Arbiter Trees [Chan and Stolfo 1995], Combiner Trees [Chan and Stolfo 1997], and Hierarchical Misture of Experts [Jordan and Jacobs 1994]. There are ensemble methods for data stream classification that use hierarchical structures, most notably HSMiner [Parker et al. 2012] and SluiceBox [Parker and Khan 2013]. HSMiner is a hierarchical additive weighted voting ensemble that boosts the accuracy of a set of weak learners by decomposing the learning problem. At the top tier of HSMiner's hierarchy stands a multi-class ensemble of k per-class ensembles, where k corresponds to the number of current class labels. Each per-class ensemble is composed of single class ensembles, which are further decomposed into single feature classifiers. Besides the top-tier ensemble, all other classifiers that compose HSMiner's hierarchy of learners are all committed to distinguish one class from all others (i.e., single class learners), thus HSMiner performs a One-Versus-All (OVA) decomposition of multi-class problems. At the bottom of HSMiner's hierarchy, single feature classifiers learn a model that discerns between the class label it represents (positive label) from all others (negative label) using only one feature. To avoid pre-processing, if the feature domain is discrete, then a Naive Bayes classifier is used, otherwise (domain is continuous) an AdaBoost ensemble of threshold learners is used. Finally, in our classification, the main difference between meta-learner and hierarchical structures is that we consider the former to only include one level of learners where the outputs are used to train second-level learners, while the latter may organize learners hierarchically for other purposes, such as decomposing the input data.

Network. The last ensemble structure in our taxonomy includes methods that arrange the ensemble members in a graph. We use the term *network* instead of *graph* to refer to these structures, since they are often dynamic and thus most closely related to complex networks [Albert and Barabási 2002] than to static graphs. In this structure, ensemble members are represented as vertices of a network whose connections are determined according to a specific criterion. In SFNC [Barddal et al. 2014], connections between classifiers are generated according to a Scale-Free Network model, such that classifiers with higher estimated accuracy are more likely to connect to recently added classifiers. During voting, classifiers' weighting is directly proportional to a given centrality metric α, for example, eigenvector, betweenness, degree, and so on. Since highly accurate classifiers usually receive most of the connections, these are expected to have higher influence on the overall decision. In SAE [Gomes and Enembreck 2013] and SAE2 [Gomes and Enembreck 2014], every pair of learners is connected and weighted according to a similarity function. The weighted network formed by all these connections is updated every period to better approximate the current state of learners similarities. This network arrangement is used during predictions, where individual decisions are first combined within subsets of similar classifiers, and afterwards these subsets decisions are combined to obtain the final prediction. This voting strategy is performed to prevent a large amount of similar classifiers (w.r.t. predictions) from dominating predictions. Also, the network in SAE and SAE2 is used to identify redundant classifiers and then remove one of them to optimize resources without drastically affecting overall classification performance.

Our goal with the presented architectures is to be as general as possible while emphasizing distinguishable characteristics of actual algorithms. However, some algorithms, which could be interpreted as ensembles, do not fit into any of these architectures. An example is the Option Hoeffding Trees [Pfahringer et al. 2007], which is basically an algorithm that could be interpreted as either a decision tree or an ensemble. The ensemble architecture, most of the time, is chosen to accomplish a specific voting scheme.



(a) Majority (b) Weighted Majority (c) Rank (d) Classifier Selection (e) Relational

Fig. 3. Ensemble basic voting methods.

3.1.2. Voting. Voting concerns how individual outputs from ensemble members are used during prediction. The ensemble structure influences voting, for example, it is possible that some learners' outputs are only used as input for another level of meta-learners. There are many voting schemes for ensemble learning, which we organize into five categories as follows: majority, weighted majority, rank, classifier selection, and relational. The voting categories can be organized in a hierarchy, such that majority vote is the most basic method, weighted majority extends majority vote by allowing heterogeneous weights, rank is a form of weighted vote where all outputs from all learners are considered, classifier selection sets weights dynamically, and relational voting transform individual votes, through an intricate method, before applying majority or weighted voting. These categories are general enough to represent voting in batch-learning, stationary, and evolving data streams. However, in an evolving data stream scenario, it is common that the voting method also plays an important role with respect to concept drift adaptation. For example, a simple strategy consists of weighting learners based on their age or on their performance restricted to the latest instances [Gomes and Enembreck 2013, 2014; Barddal et al. 2014; Brzezinski and Stefanowski 2014]. These strategies can be build on top of the five voting categories, thus they should not be interpreted as a separate category. Figure 3 depicts our five basic voting categories and the following paragraphs discuss each of them.

Majority vote. Majority vote assumes every classifier has the same weight on the overall ensemble decision. Thus, the final prediction is the class label that most classifiers predicted. To avoid ties in a binary classification setting, it is usual to define an odd number of base learners. In multiclass problems, ties can become a concern, and the default approach is to randomly break them. Examples of data stream ensembles that use majority vote include the following: Online Bagging and Boosting [Oza 2005], Leveraging Bagging [Bifet et al. 2010a], and the MCIK-Ensemble (Minimization of Cluster Impurity Kmeans Ensemble) [Masud et al. 2008].

Weighted majority. It is reasonable to weight classifiers' predictions according to some criteria. For example, it is possible to assign a score to each classifier based on its accuracy on a validation set. A more complex method is the Weighted Majority (WM) algorithm [Littlestone and Warmuth 1994]. WM weights the predictions of classifiers based on their past performance, such that every classifier has a weight β, which is decreased every time it predicts incorrectly. Majority and weighted vote share the characteristic of only considering one prediction per classifier, that is, each classifier chooses one class label. The Dynamic Weighted Majority algorithm [Kolter et al. 2003] as well as other data stream ensemble classifiers rely on some form of weighted majority, to name a few: Accuracy Update Ensemble (AUE) [Brzeziński and Stefanowski 2011], Online Accuracy Update Ensemble (OAUE) [Brzezinski and Stefanowski 2014], Adaptive Classifiers-Ensemble (ACE) [Nishida et al. 2005], and Weighted Ensemble Online Bagging (WEOB) [Wang et al. 2015].

Rank. In situations where the base learner can output more than one class label per prediction, a voting method, which is similar to the weighted majority approach,

can be used to combine all predictions. For example, if the base learner prediction is a sorted list of class labels, then the Borda count method can be used. Borda count [de Borda 1781] is a preferential voting system introduced in 1770 by Jean Charles de Borda. In ensemble learning, the overall decision when using Borda count is the class label with the highest rank sum. An example of batch ensemble that uses rank-based voting (Borda count) is the Nearest Neighbor Ensemble [Domeniconi and Yan 2004].

Classifier selection.[1] Classifier selection concerns selecting the most "appropriate" classifiers for predicting the class label of an unknown instance. The selection can take place during training [Schaffer 1993] or prediction [Merz 1996; Woods et al. 1996], which are commonly known as static classifier selection (SCS) and dynamic classifier selection (DCS), respectively. Usually, DCS involves storing instances used for training each learner and then using a k-nearest-neighbor approach to determine which classifiers should be combined for predicting an unknown instance. This naive DCS approach is infeasible on a data stream setting, as the impact of storing all instances may surpass available memory or simply cause the algorithm to take too long to calculate all needed distances. On top of that, selecting an appropriate distance function and setting k are challenging tasks. DCS has been used in several ensemble methods for data stream classification, such as the Coverage Base Ensemble Algorithm (CBEA) [Rushing et al. 2004], Attribute-Oriented Dynamic Classifier Selection (AO-DCS) [Zhu et al. 2004], and Conceptual Clustering and Prediction (CCP) [Katakis et al. 2010]. In CCP, to avoid storing a large amount of instances to describe each classifier, a clustering algorithm is used to "summarize" the instances' representation, that is, only the instances' centroids are stored.

Relational. Instead of interpreting each ensemble member prediction individually, and literally, these can be translated to reflect the class label that they most likely represent. For example, suppose two classifiers c_i and c_j consistently choose class labels 0 and 1, respectively, whenever the true class label is 2. Then it would be reasonable to "translate" to class label 2 whenever classifier c_i predicts 0 and c_j predicts 1. This is a powerful voting strategy, as it allows a group of learners to indirectly predict the class label of hard to classify instances. This voting strategy is represented by the Behavior-Knowledge Space (BKS) [Huang and Suen 1993], in batch learning, and a similar version appears for online learning as Pairwise Patterns (PP) [Gomes et al. 2015]. The Learn^{++}.NC algorithm [Muhlbaier et al. 2009] uses a voting method, namely the dynamically weighted consult and vote (DW-CAV), in which a classifier consults its peers' decisions to check if its decision is aligned with theirs and with the classes on which it was trained. If the classifier identifies discrepancies, then it reduces its vote or even refrains from voting. Other voting strategies could be classified as "Relational," yet they differ from BKS, PP, and DW-CAV significantly. For example, even though SAE [Gomes and Enembreck 2013], SAE2 [Gomes and Enembreck 2014], and SFNC [Barddal et al. 2014] extract relational data from ensemble members to generate networks, they are more closely related to a multilevel weighted majority vote, since these algorithms do not map original outputs to a different domain. Finally, relational voting can be related to the meta-learner structural arrangement (see Section 3.1.1), where the "vote" translation is delegated to a learner trained on the first layer learners' outputs.

Different voting strategies are biased towards specific assumptions regarding the problem. For example, a voting method that takes into account the class label distribution can outperform another method that does not, especially for imbalanced data streams [Wang et al. 2013, 2015]. However, a simple voting scheme may overcome a

[1]A related term to classifier selection is a *gating network*; this is often used in the Artificial Neural Networks literature [Jacobs et al. 1991].

complicated method that considers a variety of factors, usually because the assumptions on which the latter is based do not hold for the problem at hand. For example, a weighted majority vote strategy will perform poorly if the weighting function fails to represent each classifier's true prediction capabilities. Nevertheless, the ensemble structure and the voting method are useless unless ensemble members are diverse with respect to their outputs. Finally, some authors focus on determining the limits of majority voting [Kuncheva et al. 2003] or on comparing multiple voting methods using a probabilistic framework [Kuncheva and Rodríguez 2014], yet conclusions obtained are often limited to specific cases.

3.2. Diversity

Diversity is often identified as one of the building blocks of ensemble-based classifiers. The motivation for the importance of diversity can be intuitively explained using the anthropomorphic example of a group of individuals, such that their opinions are always homogeneous. This group can safely be replaced by any of its members if its only purpose is decision making.

Although some works are able to show correlations between accuracy and specific diversity measures for some special cases [Kuncheva and Whitaker 2003], theoretical guarantees are more complicated to obtain, and often inconclusive, for the general case. Unfortunately, it is not as simple as "augment diversity measure d and the overall accuracy will improve." This problem is even more complicated, because there is no generally accepted definition of diversity [Kuncheva 2004b].

Only a few studies on ensemble learning for data streams are focused on measuring diversity and on diversity properties [Minku et al. 2010; Minku and Yao 2012] and more recently in Brzezinski and Stefanowski [2016]. The authors in Brzezinski and Stefanowski [2016] present ways of calculating diversity, visualizing them over time, and using them for drift detection as additional information from the stream. Most ensemble learners proposals are accompanied with strategies to induce diversity, even when these are not part of the core proposal. In the following sections, we present different strategies to induce diversity and classical metrics to measure diversity. Appendix C presents experiments that illustrate how diversity can be monitored during stream execution for different ensemble methods.

3.2.1. Inducing Diversity. For our purposes, a set of diverse classifiers is analogous to a set of non-trivial classifiers (i.e., consistent with the training data) that, given the same instance, output different predictions. This definition assumes that learners cannot be random guessers, although it does not consider diverse a set of learners with different internal representations that consistently predict the same class labels.

In this work, we organize methods to induce diversity based on whether they manipulate the input data, the output predictions, or the underlying classifiers or use a set of heterogeneous base learners. A similar classification of diversity-inducing methods is presented in Rokach [2010] with a slightly different nomenclature.

Input manipulation. Methods that manipulate the input are common and include training classifiers in different chunks of data (horizontal partitioning) or with different subsets of features (vertical partitioning). Training classifiers with different instances often include some form of randomization, for example, bagging uses resampling [Breiman 1996]. The problem with resampling in a data stream environment is that it requires multiple passes over data. That is infeasible, since streams are expected to provide an huge amount amount of data. To solve this problem in Oza [2005] authors propose a method that approximates resampling for online processing. Besides sampling, a stream can be partitioned horizontally by "adding classifiers at different points of the stream" [Kolter et al. 2003; Brzezinski and Stefanowski 2014; Barddal

et al. 2014]. For example, classifier c_1 is added at moment t and classifier c_2 is added at moment $t + \delta$, thus c_2 will be only trained with instances provided after $t + \delta$ while c_1 is trained with instances after t. This latter strategy can also be used to indirectly adapt to drifts, but it can potentially jeopardize diversity if there are no concept drifts, because classifiers that have been added not too far apart become very similar after a while. Instead of training classifiers on different subsets of instances it is possible to train them on different subsets of features [Ho 1995; Amit and Geman 1997; Ho 1998b; Breiman 2001]. This strategy is known as the Random Subspace Method (RSM). For a feature space of m dimensions, there are $2^m - 1$ different non-empty subsets of features, and thus it is infeasible to train one learner for each subset given a high-dimensional dataset, especially on a data stream setting [Bifet et al. 2012]. Nevertheless, Ho [1998b] noted, based in the theory of stochastic modeling, that highly accurate ensembles can be obtained far before all possible combinations of subspaces are explored. RSM is usually associated with decision trees; however, in its general form it can be applied to different base learners, such as nearest neighbors [Ho 1998a] or linear classifiers [Skurichina and Duin 2002]. The reason behind the association of decision trees with RSM is attributable to the Random Forests algorithm [Breiman 2001], an ensemble method in which the random feature selection is intrinsically related to how its learners (decision trees) are trained, that is, every node split is based on a (potentially) different random subset of features. Training ensembles using RSM yield several advantages, such as diversity enhancement and efficient training and prediction. The former depends on the base learner's instability (see Section 3.3), while the latter may occur if ensemble member's training is independent, which permits training several learners in parallel. Also, on high-dimensionality problems, such as functional magnetic resonance imaging classification, RSM can be used to ease the impact of the "curse of dimensionality" by using small subsets of features per learner [Kuncheva et al. 2010]. Examples of RSM for data stream classification include the following: Streaming Random Forests [Abdulsalam et al. 2007], Restricted Hoeffding Trees [Bifet et al. 2012], Dynamic Streaming Random Forests [Abdulsalam et al. 2008], and Ensemble Decision Trees for Concept-drifting data streams (EDTC) [Li et al. 2015].

Output manipulation. To manipulate the output of a classification problem, one could decompose the original problem into smaller, potentially easier, problems. Afterwards, each problem can be mapped to a single classifier, and these classifiers would be diverse, since they interpret the hypothesis space differently. One classifier that is capable of differentiating between multiple classes is difficult to achieve, while a set of binary classifiers is relatively easy to obtain. Therefore, to cope with multiclass problems, many ensembles use the One-Versus-All approach, that is, decompose the original problem into $k(k-1)/2$ binary problems and assign a different classifier to each class, such that instances associated with other classes are interpreted as negative examples by the given classifier. Decomposing the problem in tractable smaller problems is the main goal of this strategy, while diversity increase can be viewed only as a by-product. Examples of algorithms that use this strategy for data stream learning include One-versus-All Decision Trees [Hashemi et al. 2009] and HSMiner [Parker et al. 2012]. There are some difficulties when applying this strategy to data stream learning, such as concept drifts, imbalanced class distributions, and the high update cost [Hashemi et al. 2009]. A slightly different manipulation of the output can be achieved by using Error-Correcting Output Codes (ECOC) [Dietterich and Bakiri 1995]. ECOC were inspired by the Error-Correcting Codes presented in Shannon's information theory [Shannon 1948] and were originally used to decompose multiclass problems into binary problems. In Bifet et al. [2010a], the authors experiment with a version of Leveraging Bagging that uses a variation of ECOC, namely random output codes. In random output codes class labels assigned to each example are modified to

create a new binary classification of the data induced by a mapping from all possible labels to $\{0, 1\}$. Effectively, in this setting, every classifier has a different view of the hypothesis space, for example, one classifier may interpret class labels A and B as 0, while C and D as 1. For practical reasons, in Bifet et al. [2010a], the algorithm balances the 0s and 1s for each classifier to prevent them from mapping all original labels to 0 or to 1.

Base learner manipulation. In order to achieve diversity, it is possible to modify characteristics of each base model. For example, one could use multiple neural networks with different topologies or with the same topology but starting with different weights at the first layer [Rokach 2009]. In Bifet et al. [2009], the authors propose the Adaptive Size Hoeffding Trees (ASHT) Bagging algorithm, which is an ensemble of decision trees of varying sizes. ASHT is based on the intuition that smaller trees are able to rapidly adapt to drifts, while bigger trees are useful during stable periods, thus mixing both yield different ensemble members, and may also contribute to drift recovery.

Heterogeneous base learners. Instead of varying parameters of the same base learner, it is possible to use heterogeneous base learners and obtain ensemble members with different biases. Heterogeneous ensembles for data stream learning includes BLAST [van Rijn et al. 2015], HEFT-Stream (Heterogeneous Ensemble with Feature drifT for Data Streams) [Nguyen et al. 2012], and HSMiner [Parker et al. 2012]. BLAST trains several different base learners and, during prediction, selects one of them through a meta-learning approach. HEFT-Stream maintains an ensemble of decision trees and naive Bayes learners, and, in the occurrence of a sudden drift, it adds a new learner, whose base learner matches the current learner with highest weight. HSMiner [Parker et al. 2012] uses two different base learners to avoid preprocessing features, naive Bayes for discrete and threshold learners for continuous; thus, if the feature set is heterogeneous with respect to features' domains, then learners will also be heterogeneous.

Depending on which strategy is employed for inducing diversity into the ensemble, one must be aware that while processing a massive (potentially infinite) data stream, members' models may converge. That is especially true for methods that rely solely on adding (or resetting) models on different moments of the stream. Also, instead of committing to one or another strategy to induce diversity, it is possible to combine two or more strategies. For example, HEFT-Stream trains heterogeneous learners on different samples (online bagging) and subspaces of data.[2] To assess how effective one diversity-inducing strategy is, one could choose to observe the ensemble overall accuracy. However, this analysis is biased, since there may be other factors that influence accuracy. In the next section, we present some diversity-measuring metrics and examples of their use to assess diversity in a data stream setting.

3.2.2. Measuring Diversity. There are a few reasons to measure the diversity among members of an ensemble. The most obvious is based on the intuitive notion that an ensemble of homogeneous classifiers cannot achieve any better than any of its members alone can. Thus, it may seem logical to maximize diversity, since doing so will consequently have a good impact on the overall results. However, before optimizing for diversity, it is necessary to keep in mind that no general correlation between diversity and accuracy has been proven [Kuncheva et al. 2003]. Although high diversity may not directly indicate high accuracy, measuring diversity can be useful to analyze the effectiveness of the diversity-inducing method and even to more specific tasks such as

[2]HEFT-Stream periodically runs a feature selection algorithm (Fast Correlation-Based Filter [Yu and Liu 2003]) and new learners are only trained with the latest subset of features deemed relevant.

Table II. All Possible Outputs of a Pair of Classifiers h_u and h_v

	h_u correct (1)	h_u incorrect (0)
h_v correct (1)	N^{11}	N^{10}
h_v incorrect (0)	N^{01}	N^{00}

pruning ensemble members [Margineantu and Dietterich 1997; Gomes and Enembreck 2013, 2014].

Diversity can be measured in multiple levels, but it is usually calculated in pairs (pairwise) or for the complete ensemble (non-pairwise or aggregated). We now present a few pairwise metrics, but before that we define some concepts that assist in the definition of such metrics [Kuncheva and Whitaker 2003].

Let $X = x_1, \ldots, x_n$ be a labeled data set and $\hat{y}_v = [\hat{y}_v(x_1), \ldots, \hat{y}_v(x_n)]$ an n-dimensional binary vector that represents the output of a classifier h_v, such that $\hat{y}_v(x_j) = 1$, if h_v correctly predicts the class label, for instance, x_j, and 0 otherwise. Table II presents all the possible outcomes for a pair of classifiers h_u and h_v, such that $h_u \neq h_v$, where N^{ab} is the number of instances $x_j \in X$ for which $\hat{y}_u(x_j) = a$ and $\hat{y}_v(x_j) = b$.

The Yule's Q statistic [Yule 1900] (Equation (1)), or simply Q, is a widely used measure of diversity in many fields. Q varies between -1 and 1, such that for statistically independent classifiers the expectation of $Q_{v,u}$ is 0 as intuitively it will happen when the number of equal predictions match the number of divergent predictions, that is, $N^{01} \times N^{10} = N^{11} \times N^{00}$, thus leading to a 0 numerator in Equation (1). Classifiers that tend to correctly predict the same instances yield positive values of Q, while those that commit errors on different instances render negative values,

$$Q_{v,u} = \frac{N^{11}N^{00} - N^{01}N^{10}}{N^{11}N^{00} + N^{01}N^{10}}. \tag{1}$$

Another way of estimating the pairwise diversity is the correlation coefficient $p_{v,u}$ (Equation (2)). For any two classifiers, Q and p have the same sign, and it can be proved that $|p| \leq |Q|$. Since Q is easier to calculate, it is usually preferred,

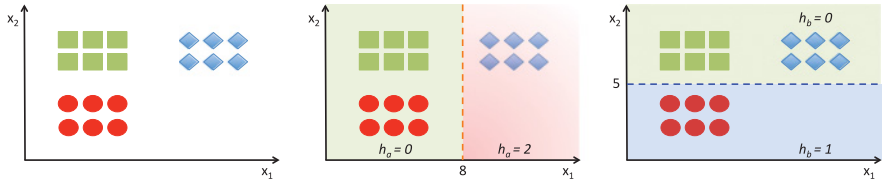
$$p_{v,u} = \frac{N^{11}N^{00} - N^{01}N^{10}}{\sqrt{(N^{11} + N^{10})(N^{01} + N^{00})(N^{11} + N^{01})(N^{10} + N^{00})}}. \tag{2}$$

The disagreement measure $D_{v,u}$ (Equation (3)) is used to characterize diversity between a base classifier h_v and a complementary classifier h_u. This metric is symmetrical and correlated with $Q_{v,u}$ and $p_{v,u}$. $D_{v,u}$ represents the ratio between the number of instances on which one classifier is correct and the other is incorrect with respect to the total number of instances,

$$D_{v,u} = \frac{N^{01} + N^{10}}{N^{11} + N^{00} + N^{01} + N^{10}}. \tag{3}$$

The measures presented so far are all based on concomitant correct or incorrect predictions. For binary classification problems, the way matrix N is calculated (see Table II) is sound, since if classifiers h_v and h_u incorrectly predict instance x class label, then h_v and h_u predictions must be equal; that is, if the correct label was 0, then h_v and h_u both predicted 1. However, for multiclass classification problems, matrix N may fail to measure differences between classifiers that incorrectly predict the same instance using different labels. For example, given a three-class classification problem, two linear classifiers h_a and h_b, such that

$$h_a = \begin{cases} 2, & x_1 \geq 8 \\ 0, & x_1 < 8 \end{cases},$$



(a) Classification space (b) Linear classifier h_a (c) Linear classifier h_b

Fig. 4. Two-dimensional multiclass problem (labels: green squares = 0, blue diamonds = 1, and red circles = 2).

Table III. Correct and Incorrect Counters N^{ab} for h_a and h_b According to Figure 4

	h_a correct (1)	h_a incorrect (0)
h_b correct (1)	6	0
h_b incorrect (0)	0	12

Table IV. All Possible Outputs of a Pair of Classifiers h_v and h_u for a Multiclass Classification Problem with k Possible Labels

	$h_u(x) = 0$	$h_u(x) = 1$	...	$h_u(x) = (k-1)$
$h_v(x) = 0$	C_{00}	C_{01}	...	$C_{0(k-1)}$
$h_v(x) = 1$	C_{10}	C_{11}	...	$C_{1(k-1)}$
...	...	...	...	...
$h_v(x) = (k-1)$	$C_{(k-1)0}$	$C_{(k-1)1}$	...	$C_{(k-1)(k-1)}$

$$h_b = \begin{cases} 0, & x_2 \geq 5 \\ 1, & x_2 < 5 \end{cases},$$

and the distribution of instances according to Figure 4, the classification errors of h_a and h_b will be accounted equally by the measures previously discussed. Table III shows the distribution of correct and incorrect predictions for h_a and h_b. If Q statistic is used to assess h_a and h_b diversity, then we obtain $Q_{a,b} = 1$, which indicates that both classifiers tend to correctly predict the same instances, yet it fails to express their divergences on incorrect predictions.

To precisely capture differences between classifiers in a multiclass problem context, a possible approach is to keep track of the classifiers' exact predictions instead of only the dichotomy correct/incorrect. This can be achieved by constructing a contingency table C_{ij}, such that the value at the intersection of a row i and a column j stores the amount of instances $x \in X$, where $h_v(x) = i$ and $h_u(x) = j$. Table IV shows an example of contingency table C_{ij} for a k-class problem. The diagonal in matrix C_{ij} contains the concomitant decisions of the pair, and thus a naive approach to weight their similarity is to sum its values and divide it by the amount of instances n, as shown in Equation (4),

$$\Theta_1 = \frac{1}{n} \sum_{i=0}^{k} C_{i,i}. \tag{4}$$

As noted in Margineantu and Dietterich [1997] in problems where one class is much more common than others, all classifiers may agree by chance, so all pairs will obtain high values for Θ_1. Thus, it is more appropriate to use the kappa κ statistic,[3] since it

[3]Cohen's kappa statistic [Cohen et al. 1960] measures inter-rater agreement for categorical variables, and it was first used in Margineantu and Dietterich [1997] as a pairwise diversity measure for ensemble learners.

Table V. Contingency Table for N^{ab} for h_a and h_b According to Figure 4

	$h_a(x) = 0$	$h_a(x) = 1$	$h_a(x) = 2$
$h_b(x) = 0$	6	0	6
$h_b(x) = 1$	6	0	0
$h_b(x) = 2$	0	0	0

corrects Θ_1 by considering the probability that two classifiers agree by chance according to the observed values in C_{ij}, namely Θ_2 (see Equation (5)). The kappa κ statistic is shown in Equation (6).

$$\Theta_2 = \sum_{i=0}^{K} \left(\sum_{j=0}^{K} \frac{C_{i,j}}{n} \cdot \sum_{j=0}^{K} \frac{C_{j,i}}{n} \right), \tag{5}$$

$$k = \frac{\Theta_1 - \Theta_2}{1 - \Theta_2}. \tag{6}$$

The interpretation of κ is similar to Q, that is, if h_u and h_v agree on every instance, then $\kappa = 1$, and if their predictions coincide by chance, then $\kappa = 0$. Negative values of κ occur when agreement is weaker than expected by chance, but this rarely occurs in real problems. The kappa statistic has already been used to report diversity for data stream ensemble-based classifiers [Bifet et al. 2009, 2010a; Kuncheva et al. 2010] and is often accompanied by a kappa-error diagram. The kappa-error diagram is a scatterplot where each point corresponds to a pair of classifiers. The x coordinate of the pair corresponds to the κ value, while the y coordinate is the average of the error rates of the two classifiers.

We now analyze our example concerning classifiers h_a and h_b (see Figure 4) from the perspective of the κ statistic. Table V presents the contingency table for h_a and h_b, where the different predictions from h_a and h_b are clearly separated. Classifiers h_a and h_b scores $\kappa = -0.2$, such that the expected agreement by chance is $\Theta_2 = 0.\overline{4}$, while h_a and h_b effective agreement is only $\Theta_1 = 0.\overline{3}$. For this toy problem, κ is able to represent the differences between h_a and h_b more precisely than Q, but still for some problems it may be the case that κ and Q yield very similar results, perhaps because classifiers tend to commit prediction errors consistently.

The semantics behind κ and Q raise at least the following two questions: (1) Does it matter to measure error divergences as in κ? (2) Should κ be the preferred diversity metric in the analysis of every ensemble learner? Reinforcing the statement at the beginning of this section, there is no diversity metric that can be considered the "best," as it depends on the situation; thus, when faced with a multiclass problem, we may choose κ, as it can differentiate between classifiers' errors. However, we may not be concerned whether classifiers commit prediction errors differently; as long as we can identify how often they correctly predict the same instances, then it is reasonable to use Q or a similar metric.

To measure diversity for the whole ensemble, one can either combine average pairwise measurements or use a non-pairwise measurement. Given symmetric diversity metrics, as is the case for κ and Q, we can calculate their average by summing all pairwise measures and dividing it by all possible pairs $2/(L(L-1))$. The main problem of "aggregating" statistics is that potentially interesting information is blurred in the midst of all possible combinations, and thus it might be used and interpreted with caution. Other diversity measures have been studied for ensemble classifiers, including non-pairwise measures, for example, double-fault, entropy, Kohavi-Wolpert, difficulty θ, and others. We refer readers to chapter 8 of Kuncheva [2004b] and other

works [Kuncheva and Whitaker 2003; Banfield et al. 2005; Zhou 2012] for an extensive discussion of diversity measures for ensemble learners.

Diversity is often identified as one of the building blocks of any ensemble-based classifier [Rokach 2010; Zhou 2012]. Developing ensemble learners for data streams it is still considered a very important step to not only induce diversity into the ensemble but also understand its implications in the overall ensemble performance [Minku et al. 2010].

Many techniques are used to induce diversity in a streaming environment, one of the simplest being the online bagging algorithm [Oza 2005]. Online bagging was used in Minku et al. [2010] to conduct diversity-related experiments in evolving data streams for mainly two reasons: (1) "it does not present any specific behaviour to handle concept drift" [Minku et al. 2010], and (2) diversity in online bagging is "controlled" through a single parameter λ.[4] In Minku et al. [2010], experiments with online bagging [Oza 2005] were specifically designed to analyze diversity before, during, and after concept drifts. Based on these experiments, authors found out that before a drift occurs, ensembles with less diversity obtain higher accuracy, but shortly after the drift occurs, highly diverse ensembles yield better accuracy despite the type of drift. Also, when there are no drifts, high diversity becomes less important.

3.3. Base Learner

Most ensemble classifiers for data stream learning were designed to work with any base learner [Bifet et al. 2010a; Oza 2005; Brzezinski and Stefanowski 2014; Gomes and Enembreck 2014] with open-ended constraints (e.g., any incremental base learner). Selecting an appropriate base learner according to the classification problem is an important step for obtaining an accurate ensemble. For example, very often classifiers can naturally deal with only one type of feature domain without resorting to input preprocessing. Thus, one can either select a base learner according to the input features domain, assuming all features have the same domain; use a base learner that deals with both discrete and continuous features, for example, Hoeffding Tree [Domingos and Hulten 2000]; or use an ensemble method that combines heterogeneous base learners coherently with the feature domain, for example, HSMiner [Parker et al. 2012]. This last approach is more flexible, as the ensemble can address problems where new features with different domains appear/disappear over time.

The base learner must match the desired diversity induction strategy. If it is planned to obtain a diverse set of classifiers by training them on different instances, like in Bagging [Breiman 1996], then unstable learners (e.g., decision trees) should be preferred instead of stable learners (e.g., Naive Bayes). Stable learners must be trained on a large set of different instances for their models to differ from one another, while unstable learners tend to yield significantly different models even when trained on subsets of instances that overlap a lot [Zhou 2012]. We illustrate this situation with an experiment where Q_{avg}, κ_{avg} and prequential accuracy (see Section 3.2.2) are reported, every 10,000 instances, for Leveraging Bagging [Bifet et al. 2010a] varying its base learner between a stable learner, Naive Bayes (NB), and an unstable learner, Hoeffding Trees (HT).[5] Figure 5 presents the results of this experiment, which indicates that, by using an unstable learner, the ensemble diversity increases, especially during the

[4]Oza [2005] observe that the probability of each instance to be selected for a given subset is approximated by a Poisson distribution with $\lambda = 1$, and thus it is feasible to "simulate" bagging in an online fashion by training each classifier k times on each instance, such that $k = poisson(\lambda = 1)$. Bifet et al. [2010a] present the Leveraging bagging algorithm, which "enhances" online bagging by using $\lambda = 6$, thus increasing the amount of instances presented to each classifier. Appendix A presents the pseudo code for online bagging.

[5]The datasets and experimental protocol used in these experiments are presented in Appendix B.

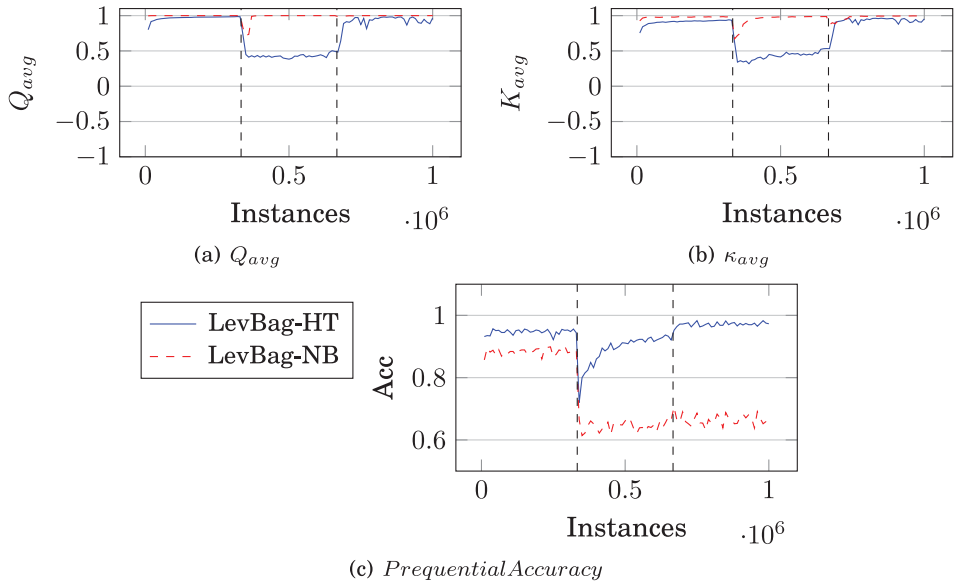


Fig. 5. Q_{avg}, κ_{avg} and Prequential Accuracy for Leveraging Bagging varying its base learner on the dataset AGR (2 Abrupt Drifts: dashed vertical lines). Max diversity is obtained at $\kappa = 0$ and $Q = 0$.

period between concept drifts. It is still possible to achieve a diverse set of classifiers by using stable learners as long as the diversity induction strategy allows it. For example, one could use different subsets of features (see "vertical partitioning" in Section 3.2.1) for training each classifier [Breiman 2001; Abdulsalam et al. 2007; Nguyen et al. 2012].

Decision trees are the most common base learner for ensemble learning in a streaming setting. Specifically, Hoeffding Trees[6] [Domingos and Hulten 2000] or some of its variations explicitly deal with concept drift as Adaptive Hoeffding Trees [Bifet and Gavaldà 2009] and Concept-Adaptive Very Fast Decision Trees [Hulten et al. 2001] or replace majority class predictions by Naive Bayes models at the leaves of the tree [Holmes et al. 2005]. Other base learners often used in ensemble stream learning include Naive Bayes [Kolter et al. 2003], perceptrons [Chen et al. 2012; Parker et al. 2012; Parker and Khan 2013], and multilayer perceptrons [Polikar et al. 2001]. Hoeffding Tree's preference over other base learners is attributable to its characteristic of being not only unstable but also an incremental learner [Domingos and Hulten 2000]. We discuss incremental and batch- (window) based learning in Section 3.4.2; however, while explaining base learners it is important to emphasize that by using non-incremental learners, such as C4.5 [Quinlan 1993], the ensemble must incorporate a parameter to control the window (batch) size used for training its members. Ensemble methods that use incremental base learners may also include a window size parameter, but then it is used to define when the ensemble is updated (e.g., adding new learners or recalculating statistics). This latter approach allows the development of algorithms in which the top-level method (the ensemble) learns at a different rate than its members [Gomes and Enembreck 2013, 2014; Brzezinski and Stefanowski 2014]. Table I can be used for a quick overview of ensemble methods that use incremental or batch-based learners.

[6]Domingos and Hulten [2000] refer to their general method of inducing decision trees for data streams as Very Fast Decision Trees and to their implementation as Hoeffding Trees.

3.3.1. Dependency. The training of one ensemble member may depend on the output of other members. A canonical example of this approach is AdaBoost [Freund et al. 1996] Conversely, classifiers may be trained completely independently of one another, which is the case for Bagging [Breiman 1996] and its variants. Intuitively, training one classifier while considering its peers' mistakes seems reasonable, as it uses more information to guide the training process, for example, which instances to emphasize or which are already correctly mapped by the group. The drawback of this approach is that it may lead to overfitting, and on a stream environment it is not straightforward to train classifiers in this "sequential" way. There are several proposals for adapting boosting for online classification [Oza 2005; Pelossof et al. 2009; Chu and Zaniolo 2004; Scholz and Klinkenberg 2007; Beygelzimer et al. 2015]. Training classifiers independently is often preferred, as it is easier, yields good results [Bifet et al. 2009], and allows training classifiers in parallel. The ability to train classifiers independently of one another is one characteristic that enables ensemble-based methods to cope with big data streams [De Francisci Morales and Bifet 2015]. An example of an algorithm that was developed to allow parallel training is HSMiner [Parker et al. 2012] and its subsequent enhancements presented in Haque et al. [2014] that runs on top of Hadoop [White 2012].

3.4. Update Dynamics

Learning from data streams requires algorithms that are not only accurate but also efficient and able to adapt to changes in data. Many methods have been presented to achieve these goals, which were thoroughly investigated in previous surveys on data stream learning and concept drift adaptation [Tsymbal 2004; Žliobaitė 2010; Hoens et al. 2012; Gama et al. 2014; Lemaire et al. 2015; Ditzler et al. 2015; Barddal et al. 2016; Webb et al. 2016]. In this section, we focus on the update dynamics of ensemble classifiers for data streams, that is, how learning takes place in the ensemble.

3.4.1. Cardinality. Intuitively, it seems that by adding more classifiers it will cause the ensemble to achieve higher accuracy. However, it is not straightforward to exploit this in practice, since as the number of classifiers increases, it becomes difficult to maintain all classifiers to minimally differ from each other, that is, a diverse set. Generating a great quantity of redundant classifiers cannot do any good to the overall decision quality but will surely negatively impact the ensemble memory and processing consumption. In a data stream context, the cardinality of the ensemble can be either defined prior to the execution or vary during execution. There are good reasons to support both approaches, for example, the resources needed for a fixed set of classifiers are easier to estimate and control, while an ensemble that can selectively add or remove classifiers has more flexibility, for example, remove redundant classifiers and save resources or add more classifiers to cover different parts of the classification space.

Ensemble methods that vary its size dynamically, like SAE [Gomes and Enembreck 2013] and DWM [Kolter et al. 2003], are intuitively better suited for a highly dynamic task, such as data stream classification. However, in practice, these ensembles can yield too little or too many classifiers, because their heuristic method, which dictates when to add or remove classifiers, is not suitable for the given data stream. To avoid too many classifiers, some methods [Kolter and Maloof 2005; Barddal et al. 2014; Gomes and Enembreck 2014] include a parameter to define the "maximum" number of classifiers of an ensemble. The complexity behind the definition of a heuristic method for adding and removing classifiers, and the success of existing stream ensembles [Oza 2005; Bifet et al. 2009, 2010a; Brzezinski and Stefanowski 2014] that use a predefined number of classifiers, explains the lack of interest in the development of strategies

for dynamically sizing the ensemble (this is observable in Table I, where algorithms' cardinality is often "fixed").

3.4.2. Learning Mode. The ability to learn new concepts (*plasticity*) while retaining previously learned knowledge (*stability*) is referred to as the stability-plasticity dilemma [Lim and Harrison 2003]. This dilemma is pervasive in the evolving data stream setting [Gama et al. 2014], as it is expected that evolving data streams alternate between drifting and stable periods. Therefore, any data stream classifier, including ensembles, must incorporate mechanisms to adapt its model to concept drifts, while accounting for periods of concept stability.

There are different types of concept drifts that may occur as well as several approaches for coping with them. For example, a classifier may periodically forget its model and learn a new model on the most recent n instances or reset the current model if a change has been triggered by a drift detector algorithm. These methods are different approaches for dealing with concept drift that can be adapted to either ensembles or single classifiers. Although ensemble-based classifiers have the advantageous characteristic of being flexible, as its members can learn at different rates (see Section 3.3), new classifiers can be added and existing classifiers may be updated, replaced, reset, or even removed selectively.

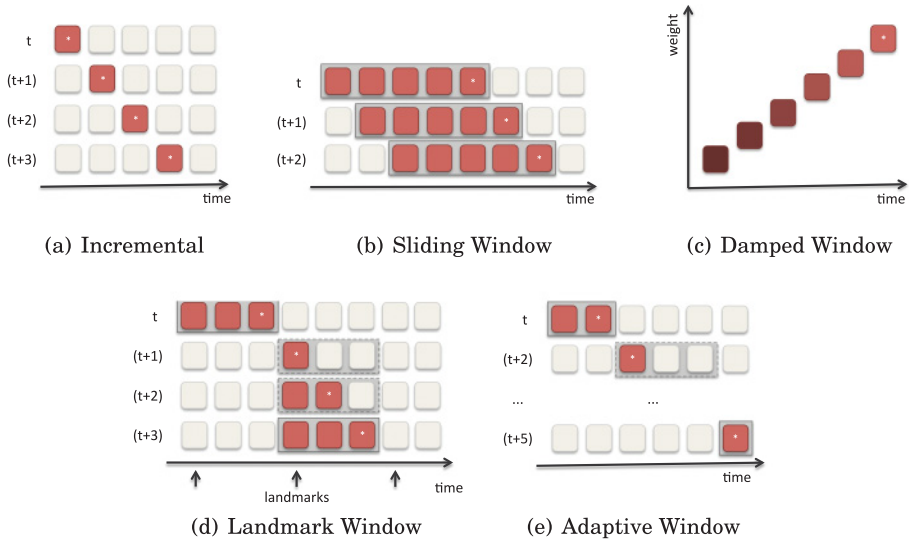
Most stream classifiers assume that *recency* is analogous to *relevance* when it comes to weighting instances for training, and ensembles are no exception to that. The reasoning behind this assumption is simple: Old instances are associated with previously outdated concepts, while new instances are committed to the most current concept. In practice, real-world problems may not adhere to this assumption as concepts can reoccur either periodically (e.g., seasons of the year) or erratically. If concepts reappear, then it is a waste of resources to re-learn old concepts over and over again. Therefore, tracking and dealing with recurring concept drifts is a difficult task in which the classifier must provide efficient answers to the following questions:

—When to store a previously learned model?
—When should a model be removed/updated?
—When (and how) to evaluate old models for "activating" them?

One can only justify using a recurring concept drift strategy if storing and updating previous models is more efficient, with respect to resources and accuracy, than relearning the model. FLORA3 [Widmer and Kubat 1996] was one of the first algorithms that dealt with recurring concept drifts. More recently, several ensemble classifiers [Katakis et al. 2010; Elwell and Polikar 2011; Jaber et al. 2013a; Ortíz Díaz et al. 2015] were designed for dealing with recurring concept drifts.

Ensemble-based algorithms can be more flexible with respect to concept drift adaptation. For example, several algorithms [Street and Kim 2001; Gomes and Enembreck 2013; Brzezinski and Stefanowski 2014; Gomes and Enembreck 2014; Gomes et al. 2015] use a background learner, that is, an auxiliary single learner trained only on the most recent instances alongside the other ensemble members but that does not influence overall decisions. Whenever it is necessary to reset an ensemble member, the background learner replaces it. There are at least two advantages that supports this approach. First, the background learner already has a trained model, and thus it will not take too long until it starts positively impacting the overall ensemble decision. Second, assuming that recency implies relevance, the background learner model might overcome existing models as it has been trained only on the latest instances.

Previous works organize data stream learning in different categorizations according to which specific learning problem the work discusses [Gama et al. 2014; Silva et al. 2013; Gama 2010; Read et al. 2012]. Gama et al. [2014] organize learning into three



(a) Incremental (b) Sliding Window (c) Damped Window

(d) Landmark Window (e) Adaptive Window

Fig. 6. Learning modes for data streams.

categories: **learning mode**: whether the algorithms retrain models or incrementally update them; **adaption methods**: concerns how adaptation to drifts happens, either proactively (blind strategy) or reactively (informed); and **model management**: divided into three aspects of ensemble learning (dynamic combination, continuous updates of learners, and structural updates). Read et al. [2012] provide an extensive discussion of the advantages and disadvantages of incremental and batch learners, while Silva et al. [2013] focus on learning modes for clustering algorithms, and thus they discuss learning according to window-based models, that is, landmark window, sliding window, and damped window.

Ensemble classifiers designed to deal with evolving data streams may combine more than one learning strategy, as explained in Section 3.3, through, for example, the combination of incremental learners and a window-based approach. In the remainder of this section, we present our attempt to classify how learning happens on ensemble classifiers. We subdivide learning into two general classes: incremental and window based. Many of the existing stream ensemble learners fall into the latter category, which is further divided into sliding windows, damped windows, landmark windows, and adaptive windows. Figure 6 illustrates learning mode according to our categorization.

In Section 3.3, we have briefly discussed how the base learner and the ensemble may operate at different learning rates. Throughout the rest of this section, we discuss each learning mode individually and present, whenever possible, examples of ensemble classifiers that instantiate the corresponding learning mode.

Incremental. A batch learner must store a batch of instances before using them for training; conversely, an incremental learner is trained on instances as they arrive. As a consequence, incremental learners better adhere to the four constraints[7] suggested in Bifet et al. [2010b] (see Section 2) and are often preferred on a data stream setting. As discussed in Read et al. [2012], both approaches have advantages and disadvantages. There is a large amount of algorithms to choose from when using batch learners, while,

[7](i) Process instances one at a time and inspect them only once, (ii) use a limited amount of memory, (iii) use a limited amount of processing time, and (iv) be ready to predict at any time.

comparatively, the amount of incremental learners available is small [Read et al. 2012]. On the other hand, batch learners require parameterizing the amount of instances to be used for training, that is, the batch size, which is critical for obtaining accurate models yet difficult to define for evolving data streams. Generally, incremental learners are more effective when applied to streams that exhibit gradual or incremental drifts or when combined with drift detectors. In the occasion of an abrupt drift, an incremental learner (without the aid of a drift detector) may take longer to recover as its model is influenced by the concepts it has previously been presented to, while a batch learner completely discards its previous models periodically. Examples of incremental learners include Bayesian classifiers [John and Langley 1995], like Naive Bayes; decision trees, like Hoeffding Trees [Domingos and Hulten 2000]; Stochastic Gradient Descent variations [Wang et al. 2012]; Instance-based (Lazy) methods [Beringer and Hüllermeier 2007; Zhang et al. 2011a]; and ensemble classifiers [Oza 2005; Bifet et al. 2010a] as long as its base learners are also incremental learners.

Landmark windows. Landmarks can be used to separate the stream into disjoint chunks of data. A landmark can be defined using the number of instances seen since in the previous landmark or according to a specific time frequency. Whenever a new landmark is reached, all instances in the previous chunk are discarded. Some ensemble classifiers use landmark windows (batches) of a fixed size n to control the periodicity of the ensemble updates, such as classifiers' removals, resets, additions, or statistics reset. This approach was first introduced in the SEA [Street and Kim 2001] and later used in other algorithms, such as DWM [Kolter et al. 2003], AddExpert [Kolter and Maloof 2005], AUE [Brzeziński and Stefanowski 2011], SAE2 [Gomes and Enembreck 2014], OAUE [Brzezinski and Stefanowski 2014], and others. It seems reasonable that if incremental learners are used, then using a predefined fixed landmark window is unnecessary. However, many ensemble classifiers for data streams combine landmark windows and incremental base learners. This design choice may allow reasonably fast adaptation to abrupt drifts (given small values of n), while it allows incremental updates of ensemble members, which help addressing gradual and incremental drifts. The fixed landmark window approach permits the use of traditional batch-learning algorithms for stream learning. In this case, a batch learner is trained on instances from window w, and its model is used to classify instances from the next window $w+1$. After window $w+1$ is over, the model learned on w is replaced by a model trained on $w+1$. If this approach is used for adapting a batch learner for stream learning, then some problems may arise, most notably as follows: Training is concentrated during the transitions between windows, and, therefore, if new instances arrive at a fast pace, then it is necessary to account for prediction delays while a new model is being trained, and very often batch learners needs to be trained on large amounts of data in order to yield accurate models, and thus the window must be very large or the learned model will be weak. Finally, if a concept drift happens, then it will not be taken into account until the window ends and the new model is generated, and thus adaptation to abrupt drifts will be slow. Despite the simplicity of using fixed landmark windows, it is difficult to define the landmark size parameter n. On one hand, if the stream has abrupt drifts, then smaller values of n are better, since ensemble updates will happen more often. On the other hand, if the stream is stationary (no drifts) or if drifts are gradual, then larger window sizes are advised, since the ensemble members will be capable of training on a larger number of instances before an ensemble update takes place. However, if the stream exhibits different types of drifts interlaced with periods of stability, then any predefined window size will most likely be unsatisfactory. Finally, by using incremental base learners instead of batch base learners, the ensemble might be capable of adapting to gradual or incremental drifts simply because its members' learned models must not necessarily be discarded after every window.

Sliding windows. Sliding windows are similar to landmark windows in the sense that both define a window size n, although sliding windows discard only one instance at a time. Instance-based classifiers [Khan et al. 2002; Law and Zaniolo 2005; Gaber et al. 2005; Beringer and Hüllermeier 2007; Zhang et al. 2011a] can be viewed as incremental learners and as sliding window-based methods. They are incremental learners since their "models" are updated after every new instance [Read et al. 2012], but, as memory is limited, it is necessary to forget one instance to make room for another. Discarding the influence of a single instance from the model is easy for instance-based methods, viable for Bayesian classifiers, and very difficult for decision trees.

Damped windows. The damped window, or time-fading model, associates weights to instances based on their age. Thus, instances that have been recently presented to the learner will have a higher weight than those that were presented a long time ago. It is possible to use a linear or exponential decay function to assign weights. The base learner must differentiate between instances weights, such that these must influence its learning, otherwise the damped window model degenerates into a simple sliding window.

Adaptive windows. The adaptive window model can be viewed as a landmark window with varying values of n. Assuming that the stream contains drifts with varying extents and rates, using windows of different sizes is a suitable strategy. The problem is how to dynamically adjust n according to observations of the stream. The FLORA2 algorithm [Widmer and Kubat 1996] uses a heuristic (Window Adjustment Algorithm) to augment or shrink the window size based on yet another heuristic that guesses whether a drift has occurred. This approach for adjusting the window size may be useful in practice; however, it depends on fixed thresholds to define by "how much" the size should be decreased or increased. On top of that, it depends on a heuristic to determine whether the current concept is stable or a drift is happening. A different approach is used by ADWIN Bagging [Bifet et al. 2009] and Leveraging Bagging [Bifet et al. 2010a], where both algorithms use the ADWIN drift detector to selectively reset classifiers. Concretely, in these algorithms, a classifier is reset whenever its associate ADWIN (ADaptive WINdow) detector signals that a drift has occurred. Thus, the ensemble may end up with classifiers with varying levels of commitment to the current concept.

4. DISCUSSION

In this section, we discuss ensemble algorithms for data stream learning from a broader perspective, including comments about new algorithm proposals (evolutionary heuristics), computational resources management, and big data stream analysis concerns.

4.1. Heuristics in the Development of New Ensemble Algorithms

As different learning strategies and heuristics are combined into novel ensemble methods the aggregate behavior may be hard to comprehend (and justify). For example, one may propose a new ensemble method that uses resampling along with random subspaces for training a set of heterogeneous classifiers, which are periodically updated whenever a drift detection algorithm indicates that a change occurred. These updates may include the following: removing classifiers with estimated accuracy below a given threshold, adding new classifiers trained only on the most recent data, storing classifiers deemed as relevant in the case where the concept they represent reoccurs, pruning "redundant" classifiers, and many more. Now, an important question may be raised about this hypothetical ensemble: Does the ensemble perform well because of the combination of all its methods or simply because some of them are very effective, while others are effectively useless or even harmful?

To answer these questions, one may present an in-depth analysis of how each of the ensemble's operators behaves individually and in combination with other operators.

This approach requires setting up experiments that are beyond measuring solely the overall ensemble accuracy. For example, SAE2 [Gomes and Enembreck 2014] uses a weighted graph to determine how classifiers votes are combined. Authors advise combining classifiers by obtaining the maximal cliques of a dichotomized version of this graph. Since SAE2 includes many other operators, there was no sufficient evidence to show if this complicated architecture and combination method results in practical benefits. To clarify this, Gomes and Enembreck [2014] compare the algorithm's performance, varying its combination method, and concluded that, all things being equal, using subgroups generated by maximal cliques is justifiable. A similar check is presented for the ensemble of Restricted Hoeffding trees [Bifet et al. 2012], where the authors note that experiments using Online Bagging [Oza 2005] with 100 classifiers did not achieve performance as good as the method presented in the article, and, thus, its performance could not be due solely to the ensemble cardinality. An example of an operator that would benefit from detailed analysis appears in HEFT-Stream [Nguyen et al. 2012], where heterogeneous base learners are used with the goal of enhancing diversity. HEFT-Stream also includes an operator to periodically replace low-performance classifiers, if a drift has been detected, by a new classifier whose base learner corresponds to that of the classifier with highest individual estimate accuracy. In practice, the ensemble often converges to a homogeneous set formed by classifiers whose base learner is capable of obtaining high individual accuracy. From one perspective, this characteristic of HEFT-Stream seems to fail to achieve a diverse set of ensemble members by using heterogeneous base learners, although it can be useful as only the "fittest" base learner remains. The only caution is that one should not rely solely on the heterogeneity of HEFT-Stream for inducing diversity into the ensemble. That is probably the reason why authors in Nguyen et al. [2012] use two other methods for inducing diversity, that is, instance resampling and feature selection.

In general, it may be difficult to isolate aspects of the method for comparisons, but it is worthwhile to verify if such experiments are possible while proposing a new technique, especially if it lacks theoretical guarantees.

4.2. Computational Resources Management and Big Data Stream Mining

Efficiently managing computational resources is of critical importance for data stream processing. Strategies to cope with this have been proposed for single classifiers as well as for ensemble classifiers. For instance, Hoeffding Trees [Domingos and Hulten 2000] uses a mechanism to freeze further node splitting if memory surpasses a user-given threshold, while SAE [Gomes and Enembreck 2013], SAE2 [Gomes and Enembreck 2014], and CBEA [Rushing et al. 2004] remove redundant ensemble members if their recent predictions are too similar (SAE and SAE2) or their coverage overlaps during training (CBEA). These techniques do not enhance the learner performance from an accuracy perspective; in fact, sometimes they might negatively impact it. Nevertheless, when applying data stream learning on real-world problems, it is expected that one may need to balance the tradeoff between computational resources and accuracy.

There are limits to what can be accomplished with algorithms executing in a single machine, even if they are multi-threaded. As a consequence, in the past few years the machine-learning community has invested a lot of research effort into highly scalable and distributed systems. It is not straightforward to adapt existing data stream ensemble learners, or single learners, for that matter, to work in a distributed environment. Efforts have been driven towards integrated platforms for stream learning in this context, which resulted in frameworks (or libraries) such as the Apache Scalable Advanced

Massive Online Analysis (SAMOA) framework [De Francisci Morales and Bifet 2015] and JUBATUS [Hido et al. 2013].[8]

Similarly to the migration of techniques, methods, and concepts from batch learning to online learning, in the near future, effort may be invested into bringing stream learner concepts to a big data stream scenario. We believe ensemble learners will play an important role in this new era of big data stream learning systems, as one of the major "faults" of ensembles is high demand from memory and processing time, which is admissible if processing is distributed. Therefore, in the near future, we may see more and more classifiers developed for big data stream processing, such as the Streaming Parallel Decision Tree [Ben-Haim and Tom-Tov 2010], HSMiner [Haque et al. 2014], and Vertical Hoeffding Tree [Kourtellis et al. 2016].

4.3. Ensemble Classifiers in Stream Setting Concerns

Many of the algorithms discussed in this work use the ensemble to solve a variety of problems that may arise in a data stream setting. For example, it has been used to solve concept evolution, tracking recurrent drift, using the output of feature selection techniques, and most notably to recover from drifts. Since ensemble usage for recovering from concept drifts has been widely explored and discussed [Kuncheva 2004a, 2008; Minku et al. 2010; Žliobaitė 2010; Hoens et al. 2012], we focus our discussion on emerging concerns in data stream analysis, which, despite being present in many different works, have not been thoroughly discussed in a survey before.

Concept evolution. New classes may appear, while existing classes may disappear from data as time goes by. OVA approaches can effectively be used along with an ensemble-based classifier to solve this problem. A general approach would be as follows: (1) Train one classifier to differentiate each possible class from all other classes; (2) if a new class appears, then train a new classifier on it; and (3) if a class disappears, then simply remove the classifier associated with it. Concrete examples of algorithms that deal with concept evolution using an OVA approach includes OVA Decision Trees [Hashemi et al. 2009], Learn^{++}.NC, and Learn^{++}.UDNC [Ditzler et al. 2010]. The OVA approach simply addresses the concept evolution problem assuming that new classes are somehow indicated to the system. A more comprehensive approach is to assume that the system is unaware of the appearance of novel classes, thus, besides dealing with their existence, the system must also detect when they appear. To deal with this problem, the authors of the DXMiner algorithm [Masud et al. 2010] assume that an instance must be closer to instances of its own class (cohesion) and further apart from instances of other classes (separation), and, thus, if a novel class appears, then its instances will be far from existing class instances and close together. Two other approaches are proposed in the HSMiner algorithm [Parker et al. 2012]; the first is very simple and assumes that any instance with a predicted label confidence below a certain threshold is novel. The second method, proposed in Parker et al. [2012], uses a sieve-like approach to iteratively refine instances on a data chunk to create a pseudo data chunk, in which each original instance is given a label of either "novel" (marked as outlier by the sieve algorithm) or "known," and afterwards a classifier is used to distinguished between "novel" and "known" classes. Finally, one could use a clustering algorithm to detect novel classes like the algorithms in Parker and Khan [2013] and Parker and Khan [2015]. The bottom line is that an ensemble-based method is often used along with any of these techniques, and although it may not be used to detect novel classes, the ensemble approach is very useful to dynamically include novel class

[8]These frameworks are described in Section 4.4.

instances into the system without major changes to the already-learned model (i.e., other ensemble members).

Feature evolution, drift, and selection. Features may appear or disappear as time passes, which is a problem, since most classifiers need to know in advance which features exist. On top of that, features may drift [Barddal et al. 2015], rendering them irrelevant for predicting future instances. To cope with these problems, it is possible to use an algorithm to select the current relevant features as in the HEFT-Stream algorithm [Nguyen et al. 2012]. Also, one can use the ensemble structure to easily remove or add the influence of specific features by using single-feature classifiers, such that if a feature disappears or is identified as irrelevant, then its influence can be completely removed from the whole system by removing the classifier associated with it. This approach is similar to that mentioned previously to cope with concept evolution (one classifier per class) and is exactly the approach used in HSMiner [Parker et al. 2012], with the addition of using different classifiers according to the feature domain (see Section 3.1.1). On top of that, using single-feature classifiers, or a limited size of features per classifier, gives more scalability to the learning algorithm, as its processing can be distributed among multiple processes or nodes [Haque et al. 2014] (see Section 4.2).

Temporal dependencies. Temporal dependencies occur when the current instance x^t class label is influenced by previous instances' ($x^{t-1}, x^{t-2}, \dots$) class labels. Temporal dependencies for data streams differ from temporal dependencies in time-series (i.e., serial correlation or autocorrelation) analysis, as mentioned in Section 2. To the best of our knowledge, only one approach to deal with temporal dependencies for data stream learning has been presented in the literature, that is, a generic wrapper that incorporates the temporal component in the feature set [Bifet et al. 2013]. We believe that temporal dependencies could be dealt with by ensemble-based methods similarly to how feature and concept evolution are addressed. A general approach would include the following: (1) Detect that temporal dependencies do exist for the given stream; (2) create a feature representing the temporal dependence; and (3) train a single feature classifier using this feature. To detect temporal dependencies, one could use an approach similar to those used to detect concept drifts that are based on classifiers accuracy [Bifet and Gavaldà 2007], but instead of accuracy use the kappa-temporal statistic κ_{per}[9] as proposed in Žliobaitė et al. [2015]. A further step on the suggested approach could include some form of weighting to assign more influence on the overall decision to the learner that has access to the temporal dependence feature.

Partially labeled instances. It is often expensive as well as time consuming to label instances, and thus a more realistic data stream classification setting should consider that only a small portion of labeled instances will be available or that it might take a long time before an instance class label becomes available. This setting confronts with the widely adopted scenario in which the class label, for instance, x^t, is available before instance x^{t+1} is presented to the learner. This problem has been tackled with different strategies [Qin et al. 2013; Borchani et al. 2011]; some of them include a combination of clustering techniques and ensemble classifiers [Masud et al. 2008; Ryu et al. 2012; Sethi et al. 2014; Parker and Khan 2015; Zhi et al. 2015]. In Ryu et al. [2012], spherical clusters are generated around dense regions of the space, and when a cluster is established, a classifier model is generated for it. Each classifier prediction is weighted according to its respective cluster distance to the instance to be predicted, and the overall result is obtained by aggregating all weighted predictions. New instances outside of any existing cluster vicinity become potential seeds for new clusters,

[9]In Žliobaitė et al. [2015], the No-Change classifier is named the Persistent classifier, and thus it is denoted as *per* in κ_{per}.

which they effectively become if their region becomes dense. In Masud et al. [2008], instances are grouped into microclusters, which are then used as input to a k-Nearest Neighbor ensemble to predict new instances' class labels. The algorithms presented in Ryu et al. [2012] and Masud et al. [2008] use clustering methods based on a radius measure, similarly to the classic k-means algorithm, and therefore they are unable to capture non-spherical clusters and often degrade to a single large cluster. To avoid shortcomings associated with radius-based clustering algorithms, in Sethi et al. [2014] a grid-based density clustering method [Chen and Tu 2007] was used on the Structured Ensemble for Partially Labeled Streams (SE-PLS) algorithm. Experiments reported in Sethi et al. [2014] show that SE-PLS is capable of obtaining high accuracy, even if only 10% of instances are labeled. Another notable ensemble-based method that deals with partially labeled instances is the SluiceBox AnyMeans (SluiceBoxAM) algorithm [Parker and Khan 2015]. SluiceBoxAM is based on the SluiceBox algorithm [Parker and Khan 2013], which already combined different methods to address existing problems in real-world applications of data stream classification, such as multi-domain features, concept drift, novel class detection, feature evolution, and others. Besides using a clustering method (AnyMeans) capable of discovering non-spherical clusters, SluiceBoxAM can be used with other ensemble classifiers; for example, in Parker and Khan [2015], authors report the performance of SluiceBoxAM combined with the leveraging bagging algorithm [Bifet et al. 2010a]. It is reasonable to question why ensemble classifiers are useful on a semi-supervised stream setting. According to the works previously mentioned [Ryu et al. 2012; Masud et al. 2008; Sethi et al. 2014; Parker and Khan 2015], the reason can be summarized in one word: flexibility. For example, it is easier to associate one classifier per cluster [Sethi et al. 2014; Parker and Khan 2015] and use a weighted combination to obtain predictions, since whenever a cluster is removed, its influence on the overall model can be completely removed simply by deleting the learner associated with it.

4.4. Open-Source Software

In this section, we present existing open-source frameworks and libraries and one workflow engine that comprise data stream learning, including ensemble-based algorithms. These frameworks facilitate collaboration among research groups and allow researchers to directly test their ideas without being concerned with common problems like algorithm evaluation or parallel implementations.

Massive Online Analysis[10] **[Bifet et al. 2010b].** The MOA framework was developed to provide tools for data stream analysis. MOA provides many data stream learning algorithms, including ensemble classifiers. Besides learning algorithms, MOA also provides data generators (e.g., Random Tree Generator, SEA, AGR), evaluation methods (e.g., periodic holdout, test-then-train, prequential), and statistics (CPU time, RAM-hours, kappa). MOA can be used through a GUI (Graphical User Interface) or on a command line, which facilitates running batches of tests. MOA is implemented in Java and shares many characteristics with the WEKA framework [Hall et al. 2009], such as allowing users to extend the framework by inheriting abstract classes. Very often researchers make their source code available as a MOA extension.[11]

Advanced Data Mining and Machine Learning System (ADAMS)[12] **[Reutemann and Vanschoren 2012].** Advanced Data Mining and Machine Learning System (ADAMS) is a workflow engine designed to prototype and maintain complex knowledge workflows. ADAMS is not a data stream learning tool *per se*, but it can be

[10]http://moa.cms.waikato.ac.nz.
[11]http://moa.cms.waikato.ac.nz/moa-extensions/.
[12]https://adams.cms.waikato.ac.nz.

combined with MOA, and other frameworks like SAMOA and WEKA, to perform data stream analysis.

Scalable Advanced Massive Online Analysis[13] **[De Francisci Morales and Bifet 2015].** SAMOA combines stream mining and distributed computing (i.e., MapReduce) and is described as a framework as well as a library. As a framework, SAMOA allows users to abstract the underlying stream processing execution engine and focus on the learning problem at hand. Currently, it is possible to use Storm (http://storm.apache.org), S4 (http://incubator.apache.org/s4), or Samza (http://samza.incubator.apache.org). SAMOA provides adapted versions of stream learners for distributed processing, including the Vertical Hoeffding Tree algorithm [Kourtellis et al. 2016], bagging, and boosting.

JUBATUS[14] **[Hido et al. 2013].** JUBATUS is a framework that uses a loose model-sharing architecture for execution of big data stream learning algorithms. Similarly to MapReduce, JUBATUS defines three fundamental operations: Update, Mix, and Analyze. Update represents the training phase of the learning algorithm. Mix refers to sharing models, instead of data, between servers. Finally, the Analyze operation is used for prediction.

Vowpal Wabbit (VW).[15] Vowpal Wabbit (VW) is an open-source machine-learning library with an efficient scalable implementation that includes several learning algorithms. VW has been used to learn from a terafeature dataset using 1,000 nodes in approximately 1h [Agarwal et al. 2014].

StreamDM.[16] StreamDM is an open-source framework for big data stream mining that uses the Spark Streaming [Zaharia et al. 2013] extension of the core Spark API (Application Program Interface).

Others. Some algorithms are available through specific implementations outside existing frameworks. We provide a non-exhaustive list of such implementations:

—Learn^{++}.NSE: https://github.com/gditzler/IncrementalLearning
—Online Non-stationary Boosting (ONSBoosting): http://www.cs.man.ac.uk/∼pococka4/ONSBoost.html
—M^3 (MOA extension): https://github.com/bigfastdata/MOA-IRND
—SAE2 and SFNC (MOA extension): https://sites.google.com/site/moasocialbasedalgorithms/home
—DWM, RCD, and Learn^{++}.NSE (MOA extension): https://sites.google.com/site/moaextensions/

5. CONCLUSION

Data stream learning poses several problems, as it involves maintaining an up-to-date model induced from data arriving at high speeds using limited resources. Ensemble-based methods are suitable choices to cope with data streams, as they often achieve high accuracy and can be combined with flexible operators to address issues such as concept drifts. This article presents the main characteristics of ensemble learners, identifies open-source software, and discusses current and expected future trends in ensemble learning for data streams, such as concept and feature evolution, diversity measurement, big data stream learning, temporal dependencies, and instance labeling.

It is expected that in the near future ensemble learners will be thoroughly explored for big data stream learning, especially methods that do not rely on dependent base

[13]http://samoa.incubator.apache.org.
[14]http://jubat.us/.
[15]https://github.com/JohnLangford/vowpal_wabbit/.
[16]http://huawei-noah.github.io/streamDM/.

learners and can deal with high dimensionality (e.g., by employing random subspaces), as these can be easily parallelized and are scalable. Also, there is a trend to move from the traditional classification setting to new methods that deal with challenging real-world scenarios, especially semi-supervised (partially labeled) and imbalanced data streams, in which the flexibility of ensemble learners is well appreciated. Finally, new data stream ensemble learners should include some form of mechanism to take into account temporal dependencies, concept evolutions, and feature drifts, as these have a high impact on the overall classification performance.

REFERENCES

Hanady Abdulsalam, David B. Skillicorn, and Patrick Martin. 2007. Streaming random forests. In *Proceedings of the 11th International Database Engineering and Applications Symposium, 2007 (IDEAS'07)*. IEEE, 225–232.

Hanady Abdulsalam, David B. Skillicorn, and Patrick Martin. 2008. Classifying evolving data streams using dynamic streaming random forests. In *Proceedings of the International Conference on Database and Expert Systems Applications*. Springer, 643–651.

Alekh Agarwal, Olivier Chapelle, Miroslav Dudík, and John Langford. 2014. A reliable effective terascale linear learning system. *J Mach. Learn. Res.* 15, 1 (2014), 1111–1133.

Charu C. Aggarwal. 2007. *Data Streams—Models and Algorithms*. Advances in Database Systems, Vol. 31. Springer.

Rakesh Agrawal, Tomasz Imilielinski, and Arun Swani. 1993. Database mining: A performance perspective. *IEEE Trans. Knowl. Data Eng.* 5, 6 (Dec. 1993), 914–925.

Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Rev. Mod. Phys.* 74, 1 (2002), 47.

Ethem Alpaydin and Cenk Kaynak. 1998. Cascading classifiers. *Kybernetika* 34, 4 (1998), 369–374.

Yali Amit and Donald Geman. 1997. Shape quantization and recognition with randomized trees. *Neur. Comput.* 9, 7 (1997), 1545–1588.

Robert E. Banfield, Lawrence O. Hall, Kevin W. Bowyer, and W. Philip Kegelmeyer. 2005. Ensemble diversity measures and their application to thinning. *Inf. Fus.* 6, 1 (2005), 49–62.

Jean Paul Barddal, Heitor Murilo Gomes, and Fabrício Enembreck. 2014. SFNClassifier: A scale-free social network method to handle concept drift. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC'14)*. ACM, 786–791.

Jean Paul Barddal, Heitor Murilo Gomes, and Fabrício Enembreck. 2015. A survey on feature drift adaptation. In *Proceedings of the International Conference on Tools with Artificial Intelligence (ICTAI'15)*. IEEE.

Jean Paul Barddal, Heitor Murilo Gomes, Fabrício Enembreck, and Bernhard Pfahringer. 2016. A survey on feature drift adaptation: Definition, benchmark, challenges and future directions. *J. Syst. and Software* (2016).

Yael Ben-Haim and Elad Tom-Tov. 2010. A streaming parallel decision tree algorithm. *J. Mach. Learn. Res.* 11 (2010), 849–872.

Jürgen Beringer and Eyke Hüllermeier. 2007. An efficient algorithm for instance-based learning on data streams. In *Industrial Conference on Data Mining*. Springer, 34–48.

Alina Beygelzimer, Satyen Kale, and Haipeng Luo. 2015. Optimal and adaptive algorithms for online boosting. (2015), 2323–2331.

Albert Bifet, Eibe Frank, Geoff Holmes, and Bernhard Pfahringer. 2012. Ensembles of restricted Hoeffding trees. *ACM Trans. Intell. Syst. Technol.* 3, 2 (2012), 30.

Albert Bifet and Ricard Gavaldà. 2007. Learning from time-changing data with adaptive windowing. In *SIAM*.

Albert Bifet and Ricard Gavaldà. 2009. Adaptive learning from evolving data streams. In *Proceedings of the International Symposium on Intelligent Data Analysis*. Springer, 249–260.

Albert Bifet, Geoff Holmes, Richard Kirkby, and Bernhard Pfahringer. 2010b. MOA: Massive online analysis. *J. Mach. Learn. Res.* 11 (2010), 1601–1604.

Albert Bifet, Geoff Holmes, and Bernhard Pfahringer. 2010a. Leveraging bagging for evolving data streams. In *PKDD*. 135–150.

Albert Bifet, Geoff Holmes, Bernhard Pfahringer, Richard Kirkby, and Ricard Gavaldà. 2009. New ensemble methods for evolving data streams. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM SIGKDD, 139–148.

Albert Bifet, Jesse Read, Indrė Žliobaitė, Bernhard Pfahringer, and Geoff Holmes. 2013. Pitfalls in benchmarking data stream classification and how to avoid them. In *Machine Learning and Knowledge Discovery in Databases*. Springer, 465–479.

Jock A. Blackard and Denis J. Dean. 1999. 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.

Hanen Borchani, Pedro Larrañaga, and Concha Bielza. 2011. Classifying evolving data streams with partially labeled data. *Intell. Data Anal.* 15, 5 (2011), 655–670.

Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. 2008. *Metalearning: Applications to Data Mining*. Springer Science & Business Media.

Leo Breiman. 1996. Bagging predictors. *Mach. Learn.* 24, 2 (1996), 123–140.

Leo Breiman. 2001. Random forests. *Mach. Learn.* 45, 1 (2001), 5–32.

Gavin Brown, Jeremy Wyatt, Rachel Harris, and Xin Yao. 2005. Diversity creation methods: A survey and categorisation. *Inf. Fus.* 6, 1 (2005), 5–20.

Dariusz Brzeziński and Jerzy Stefanowski. 2011. Accuracy updated ensemble for data streams with concept drift. In *Hybrid Artificial Intelligent Systems*. Springer, 155–163.

Dariusz Brzezinski and Jerzy Stefanowski. 2014. Combining block-based and online methods in learning ensembles from concept drifting data streams. *Inf. Sci.* 265 (2014), 50–67.

Dariusz Brzezinski and Jerzy Stefanowski. 2016. Ensemble diversity in evolving data streams. In *Proceedings of the International Conference on Discovery Science*. Springer, 229–244.

Philip K. Chan and Salvatore J. Stolfo. 1995. A comparative evaluation of voting and meta-learning on partitioned data. In *ICML*. 90–98.

Philip K. Chan and Salvatore J. Stolfo. 1997. On the accuracy of meta-learning for scalable data mining. *J. Intell. Inf. Syst.* 8, 1 (1997), 5–28.

Shang-Tse Chen, Hsuan-Tien Lin, and Chi-Jen Lu. 2012. An online boosting algorithm with theoretical justifications. In *Proceedings of the International Conference on Machine Learning (ICML)*.

Yixin Chen and Li Tu. 2007. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*. ACM, New York, NY, 133–142.

Fang Chu and Carlo Zaniolo. 2004. Fast and light boosting for adaptive mining of data streams. In *Advances in Knowledge Discovery and Data Mining*. Springer, 282–292.

Jacob Cohen and others. 1960. A coefficient of agreement for nominal scales. *Educ. Psychol. Meas.* 20, 1 (1960), 37–46.

Jean Charles de Borda. 1781. *Memoire Sur Les Elections Au Scrutin*. Historie de l'Academie Royale des Sciences, Paris.

Gianmarco De Francisci Morales and Albert Bifet. 2015. SAMOA: Scalable advanced massive online analysis. *J. Mach. Learn. Res.* 16 (2015), 149–153.

Magdalena Deckert. 2011. Batch weighted ensemble for mining data streams with concept drift. In *Foundations of Intelligent Systems*. Springer, 290–299.

Thomas G. Dietterich. 2000. Ensemble methods in machine learning. In *Multiple Classifier Systems*. Springer, 1–15.

Thomas G. Dietterich and Ghulum Bakiri. 1995. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research* 2 (1995), 263–286.

Gregory Ditzler, Michael D. Muhlbaier, and Robi Polikar. 2010. Incremental learning of new classes in unbalanced datasets: Learn++.UDNC. In *Multiple Classifier Systems*. Lecture Notes in Computer Science, Vol. 5997. Springer, Berlin, 33–42. DOI:http://dx.doi.org/10.1007/978-3-642-12127-2_4

Gregory Ditzler and Robi Polikar. 2013. Incremental learning of concept drift from streaming imbalanced data. *IEEE Trans. Knowl. Data Eng.* 25, 10 (2013), 2283–2301.

Gregory Ditzler, Manuel Roveri, Cesare Alippi, and Robi Polikar. 2015. Learning in nonstationary environments: A survey. *IEEE Comput. Intell. Mag.* 10, 4 (2015), 12–25.

Carlotta Domeniconi and Bojun Yan. 2004. Nearest neighbor ensemble. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004 (ICPR'04)*, Vol. 1. IEEE, 228–231.

Pedro Domingos and Geoff Hulten. 2000. Mining high-speed data streams. In *Proc. of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM SIGKDD, 71–80.

Ms Snehlata Dongre and Latesh Malik. 2013. Algorithm to handle concept drifting in data stream mining 1. (2013).

Ryan Elwell and Robi Polikar. 2011. Incremental learning of concept drift in nonstationary environments. *IEEE Trans. Neur. Netw.* 22, 10 (2011), 1517–1531.

Alan Fern and Robert Givan. 2003. Online ensemble learning: An empirical study. *Mach. Learn.* 53, 1–2 (2003), 71–109.

Yoav Freund and Robert E. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* 55, 1 (1997), 119–139.

Yoav Freund, Robert E. Schapire, and others. 1996. Experiments with a new boosting algorithm. In *ICML*, Vol. 96. 148–156.

Mohamed Medhat Gaber, Shonali Krishnaswamy, and Arkady Zaslavsky. 2005. On-board mining of data streams in sensor networks. In *Advanced Methods for Knowledge Discovery from Complex Data*. Springer, 307–335.

João Gama. 2010. *Knowledge Discovery from Data Streams* (1st ed.). Chapman & Hall/CRC.

João Gama and Pavel Brazdil. 2000. Cascade generalization. *Mach. Learn.* 41, 3 (2000), 315–343. DOI: http://dx.doi.org/10.1023/A:1007652114878

João Gama and Mohamed Medhat Gaber. 2007. *Learning from Data Streams*. Springer.

João Gama and Petr Kosina. 2009. Tracking recurring concepts with meta-learners. In *Progress in Artificial Intelligence*. Springer, 423–434.

João Gama and Pedro Rodrigues. 2009. Issues in evaluation of stream learning algorithms. In *15th ACM SIGKDD*. ACM SIGKDD, 329–338.

João Gama, Indre Zliobaite, Albert Bifet, Mykole Pechenizkiy, and Abderlhamid Bouchachia. 2014. A survey on concept drift adaptation. *ACM Comput. Surv.* 46, 4, Article 44 (Mar. 2014), 37 pages. DOI: http://dx.doi.org/10.1145/2523813

Heitor Murilo Gomes, Jean Paul Barddal, and Fabrício Enembreck. 2015. Pairwise combination of classifiers for ensemble learning on data streams. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing (SAC) (SAC'15)*. ACM, 941–946.

Heitor Murilo Gomes and Fabrício Enembreck. 2013. SAE: Social adaptive ensemble classifier for data streams. In *Proceedings of the 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*. 199–206. DOI: http://dx.doi.org/10.1109/CIDM.2013.6597237

Heitor Murilo Gomes and Fabrício Enembreck. 2014. SAE2: Advances on the social adaptive ensemble classifier for data streams. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC) (SAC'14)*. ACM, 199–206.

Paulo Mauricio Gonçalves Jr and Roberto Souto Maior de Barros. 2013. RCD: A recurring concept drift framework. *Pattern Recogn. Lett.* 34, 9 (2013), 1018–1025.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: An update. *ACM SIGKDD Expl. Newslett.* 11, 1 (2009), 10–18.

Ashraful Haque, Brendon Parker, Latifur Khan, and Bhavani Thuraisingham. 2014. Evolving big data stream classification with MapReduce. In *Proceedings of the 2014 IEEE 7th International Conference on Cloud Computing (CLOUD)*. IEEE, 570–577.

Sattar Hashemi, Ying Yang, Zahra Mirzamomen, and Mohammadreza Kangavari. 2009. Adapted one-versus-all decision trees for data stream classification. *IEEE Trans. Knowl. Data Eng.* 21, 5 (2009), 624–637.

Shohei Hido, Seiya Tokui, and Satoshi Oda. 2013. Jubatus: An open source platform for distributed online machine learning. In *NIPS 2013 Workshop on Big Learning, Lake Tahoe*.

Tin Kam Ho. 1995. Random decision forests. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition, 1995*, Vol. 1. IEEE, 278–282.

Tin Kam Ho. 1998a. Nearest neighbors in random subspaces. In *Proceedings of the Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR)*. Springer, 640–648.

Tin Kam Ho. 1998b. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.* 20, 8 (1998), 832–844.

T. Ryan Hoens, Robi Polikar, and Nitesh V. Chawla. 2012. Learning from streaming data with concept drift and imbalance: An overview. *Progr. Artif. Intell.* 1, 1 (2012), 89–101.

Geoffrey Holmes, Richard Kirkby, and Bernhard Pfahringer. 2005. Stress-testing hoeffding trees. In *Proceedings of the European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 495–502.

Y. S. Huang and C. Y. Suen. 1993. The behavior-knowledge space method for combination of multiple classifiers. In *Proceedings of the IEEE Computer Vision and Pattern Recog.* IEEE, 347–352.

Geoff Hulten, Laurie Spencer, and Pedro Domingos. 2001. Mining time-changing data streams. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 97–106.

Ghazal Jaber, Antoine Cornuéjols, and Philippe Tarroux. 2013a. A new on-line learning method for coping with recurring concepts: The ADACC system. In *Neural Information Processing*. Springer, 595–604.

Ghazal Jaber, Antoine Cornuéjols, and Philippe Tarroux. 2013b. Online learning: Searching for the best forgetting strategy under concept drift. In *Neural Information Processing*. Springer, 400–408.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. 1991. Adaptive mixtures of local experts. *Neur. Comput.* 3, 1 (1991), 79–87.

Anil K. Jain, Robert P. W. Duin, and Jianchang Mao. 2000. Statistical pattern recognition: A review. *IEEE Trans. Pattern Anal. Mach. Intell.* 22, 1 (2000), 4–37.

George H. John and Pat Langley. 1995. Estimating continuous distributions in bayesian classifiers. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 338–345.

Michael I. Jordan and Robert A. Jacobs. 1994. Hierarchical mixtures of experts and the EM algorithm. *Neur. Computa.* 6, 2 (1994), 181–214.

Ioannis Katakis, Grigorios Tsoumakas, and Ioannis Vlahavas. 2010. Tracking recurring contexts using ensemble classifiers: An application to email filtering. *Knowl. Inf. Syst.* 22, 3 (2010), 371–391.

Maleq Khan, Qin Ding, and William Perrizo. 2002. K-nearest neighbor classification on spatial data streams using p-trees. In *Advances in Knowledge Discovery and Data Mining*. Springer, 517–528.

Jeremy Z. Kolter, Marcus Maloof, and others. 2003. Dynamic weighted majority: A new ensemble method for tracking concept drift. In *Proceedings of the 3rd IEEE International Conference on Data Mining, 2003 (ICDM 2003)*. IEEE, 123–130.

Jeremy Z. Kolter and Marcus A. Maloof. 2005. Using additive expert ensembles to cope with concept drift. In *Proceedings of the 22nd International Conference on Machine Learning (ICML'05)*. ACM, New York, NY, 449–456. DOI:http://dx.doi.org/10.1145/1102351.1102408

Nicolas Kourtellis, Gianmarco De Francisci Morales, Albert Bifet, and Arinto Murdopo. 2016. VHT: Vertical hoeffding tree. In *2016 IEEE International Conference on Big Data (Big Data)*. 915–922. DOI:http://dx.doi.org/10.1109/BigData.2016.7840687

Ludmila I. Kuncheva. 2003. That elusive diversity in classifier ensembles. (2003), 1126–1138.

Ludmila I. Kuncheva. 2004a. Classifier ensembles for changing environments. In *Multiple Classifier Systems*. Springer, 1–15.

Ludmila I. Kuncheva. 2004b. *Combining Pattern Classifiers: Methods and Algorithms*. John Wiley & Sons.

Ludmila I. Kuncheva. 2008. Classifier ensembles for detecting concept change in streaming data: Overview and perspectives. In *2nd Workshop SUEMA*. 5–10.

Ludmila I. Kuncheva and Juan J. Rodríguez. 2014. A weighted voting framework for classifiers ensembles. *Knowl. Inf. Syst.* 38, 2 (2014), 259–275.

Ludmila I. Kuncheva, Juan J. Rodríguez, Catrin O. Plumpton, David E. J. Linden, and Stephen J. Johnston. 2010. Random subspace ensembles for fMRI classification. *IEEE Trans. Med. Imag.* 29, 2 (2010), 531–542.

Ludmila I. Kuncheva and Christopher J. Whitaker. 2003. Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Mach. Learn.* 51, 2 (2003), 181–207.

Ludmila I. Kuncheva, Christopher J. Whitaker, Catherine A. Shipp, and Robert P. W. Duin. 2003. Limits on the majority vote accuracy in classifier fusion. *Pattern Anal. Appl.* 6, 1 (2003), 22–31.

Yan-Nei Law and Carlo Zaniolo. 2005. An adaptive nearest neighbor classification algorithm for data streams. In *Knowledge Discovery in Databases: PKDD 2005*. Springer, 108–120.

Vincent Lemaire, Christophe Salperwyck, and Alexis Bondu. 2015. A survey on supervised classification on data streams. In *Business Intelligence*. Springer, 88–125.

Peipei Li, Xindong Wu, Xuegang Hu, and Hao Wang. 2015. Learning concept-drifting data streams with random ensemble decision trees. *Neurocomputing* (2015).

Chee Peng Lim and Robert F. Harrison. 2003. Online pattern classification with multiple neural network systems: An experimental study. *IEEE Trans. Syst. Man Cybernet. C* 33, 2 (2003), 235–247.

Nick Littlestone and Manfred K. Warmuth. 1994. The weighted majority algorithm. *Inf. Comput.* 108, 2 (1994), 212–261.

Dragos D. Margineantu and Thomas G. Dietterich. 1997. Pruning adaptive boosting. In *ICML*, Vol. 97. Citeseer, 211–218.

Mohammad M. Masud, Qing Chen, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. 2010. Classification and novel class detection of data streams in a dynamic feature space. In *Machine Learning and Knowledge Discovery in Databases*. Springer, 337–352.

Mohammad M. Masud, Jing Gao, Latifur Khan, Jiawei Han, and Bhavani Thuraisingham. 2008. A practical approach to classify evolving data streams: Training with limited amount of labeled data. In *8th IEEE International Conference on Data Mining, 2008 (ICDM'08)*. IEEE, 929–934.

Christopher J. Merz. 1996. Dynamical selection of learning algorithms. In *Learning from Data*. Springer, 281–290.

Leandro L. Minku, Allan P. White, and Xin Yao. 2010. The impact of diversity on online ensemble learning in the presence of concept drift. *IEEE Trans. Knowl. Data Eng.* 22, 5 (2010), 730–742.

Leandro L. Minku and Xin Yao. 2012. DDD: A new ensemble approach for dealing with concept drift. *IEEE Trans. Knowl. Data Eng.* 24, 4 (2012), 619–633. DOI:http://dx.doi.org/10.1109/TKDE.2011.58

Michael D. Muhlbaier, Apostolos Topalis, and Robi Polikar. 2009. Learn^{++}.NC: Combining ensemble of classifiers with dynamically weighted consult-and-vote for efficient incremental learning of new classes. *IEEE Trans. Neur. Netw.* 20, 1 (2009), 152–168.

Hai-Long Nguyen, Yew-Kwong Woon, Wee-Keong Ng, and Li Wan. 2012. Heterogeneous ensemble for feature drifts in data streams. In *Advances in Knowledge Discovery and Data Mining*, Pang-Ning Tan, Sanjay Chawla, ChinKuan Ho, and James Bailey (Eds.). Lecture Notes in Computer Science, Vol. 7302. Springer, Berlin, 1–12.

Kyosuke Nishida, Koichiro Yamauchi, and Takashi Omori. 2005. Ace: Adaptive classifiers-ensemble system for concept-drifting environments. In *Multiple Classifier Systems*. Springer, 176–185.

Agustín Ortíz Díaz, José del Campo-Ávila, Gonzalo Ramos-Jiménez, Isvani Frías Blanco, Yailé Caballero Mota, Antonio Mustelier Hechavarría, and Rafael Morales-Bueno. 2015. Fast adapting ensemble: A new algorithm for mining data streams with concept drift. *Sci. World J.* 2015 (2015).

N. C. Oza. 2005. Online bagging and boosting. In *Proceedings of the 2005 IEEE International Conference on Systems, Man and Cybernetics*, Vol. 3. 2340–2345. DOI:http://dx.doi.org/10.1109/ICSMC.2005.1571498

Brandon Parker and Latifur Khan. 2013. Rapidly labeling and tracking dynamically evolving concepts in data streams. In *Proceedings of the 2013 IEEE 13th International Conference on Data Mining Workshops*. 1161–1164. DOI:http://dx.doi.org/10.1109/ICDMW.2013.37

Brandon Parker, Ahmad M. Mustafa, and Latifur Khan. 2012. Novel class detection and feature via a tiered ensemble approach for stream mining. In *Proceedings of the 2012 IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI)*, Vol. 1. IEEE, 1171–1178.

Brandon Shane Parker and Latifur Khan. 2015. Detecting and tracking concept class drift and emergence in non-stationary fast data streams. In *Proceedings of the 29th AAAI Conference on Artificial Intelligence*.

Brandon S. Parker, Latifur Khan, and Albert Bifet. 2014. Incremental ensemble classifier addressing non-stationary fast data streams. In *Proceedings of the 2014 IEEE International Conference on Data Mining Workshop (ICDMW)*. IEEE, 716–723.

Raphael Pelossof, Michael Jones, Ilia Vovsha, and Cynthia Rudin. 2009. Online coordinate boosting. In *Proceedings of the 2009 IEEE 12th International Conference on Computer Vision Workshops (ICCV Workshops)*. IEEE, 1354–1361.

Bernhard Pfahringer, Geoffrey Holmes, and Richard Kirkby. 2007. New options for hoeffding trees. In *Proceedings of the 20th Australian Joint Conference on Advances in Artificial Intelligence (AI'07)*. Springer-Verlag, Berlin, 90–99.

Adam Pocock, Paraskevas Yiapanis, Jeremy Singer, Mikel Luján, and Gavin Brown. 2010. Online non-stationary boosting. In *Multiple Classifier Systems*. Springer, 205–214.

Robi Polikar. 2006. Ensemble based systems in decision making. *IEEE Circ. Syst. Mag.* 6, 3 (2006), 21–45.

Robi Polikar, Lalita Updа, Satish S. Updа, and Vasant Honavar. 2001. Learn++: An incremental learning algorithm for supervised neural networks. *IEEE Trans. Syst. Man Cybernet. C* 31, 4 (2001), 497–508.

Xiangju Qin, Yang Zhang, Chen Li, and Xue Li. 2013. Learning from data streams with only positive and unlabeled data. *J. Intelli. Inform. Syst.* 40, 3 (2013), 405–430.

J. Ross Quinlan. 1993. C4.5: Programs for machine learning. Vol. 1.

Sasthakumar Ramamurthy and Raj Bhatnagar. 2007. Tracking recurrent concept drift in streaming data using ensemble classifiers. In *Proceedings of the 6th International Conference on Machine Learning and Applications, 2007 (ICMLA'07)*. IEEE, 404–409.

Jesse Read, Albert Bifet, Bernhard Pfahringer, and Geoff Holmes. 2012. Batch-incremental versus instance-incremental learning in dynamic and evolving data. In *Advances in Intelligent Data Analysis XI*. Springer, 313–323.

Peter Reutemann and Joaquin Vanschoren. 2012. Scientific workflow management with ADAMS. In *Machine Learning and Knowledge Discovery in Databases*. Springer, 833–837.

Lior Rokach. 2009. Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography. *Comput. Stat. Data Anal.* 53, 12 (2009), 4046–4072.

Lior Rokach. 2010. Ensemble-based classifiers. *Artif. Intell. Rev.* 33, 1–2 (2010), 1–39.

John Rushing, Sara Graves, Evans Criswell, and Amy Lin. 2004. A coverage based ensemble algorithm (CBEA) for streaming data. In *Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence, 2004 (ICTAI'04)*. IEEE, 106–112.

Joung Woo Ryu, Mehmed M. Kantardzic, and Myung-Won Kim. 2012. Efficiently maintaining the performance of an ensemble classifier in streaming data. In *Convergence and Hybrid Information Technology*. Springer, 533–540.

Cullen Schaffer. 1993. Selecting a classification method by cross-validation. *Mach. Learn.* 13, 1 (1993), 135–143.

Martin Scholz and Ralf Klinkenberg. 2007. Boosting classifiers for drifting concepts. *Intell. Data Anal.* 11, 1 (2007), 3–28.

Tegjyot Singh Sethi, Mehmed Kantardzic, Elaheh Arabmakki, and Hanquing Hu. 2014. An ensemble classification approach for handling spatio-temporal drifts in partially labeled data streams. In *Proceedings of the 2014 IEEE 15th International Conference on Information Reuse and Integration (IRI)*. IEEE, 725–732.

Martin Sewell. 2008. Ensemble learning. *RN* 11, 2 (2008).

Claude Elwood Shannon. 1948. A mathematical theory of communication. *Bell Syst. Techn. J.* 27, 3 (1948), 379–423 and 623–656.

Jonathan A. Silva, Elaine R. Faria, Rodrigo C. Barros, Eduardo R. Hruschka, André C. P. L. F. de Carvalho, and João Gama. 2013. Data stream clustering: A survey. *ACM Comput. Surv.* 46, 1 (2013), 13:1–13:31.

Marina Skurichina and Robert P. W. Duin. 2002. Bagging, boosting and the random subspace method for linear classifiers. *Pattern Anal. Appl.* 5, 2 (2002), 121–135.

Kenneth O. Stanley. 2003. *Learning Concept Drift with a Committee of Decision Trees*. Informe Técnico: UT-AI-TR-03-302, Department of Computer Sciences, University of Texas at Austin (2003).

W. Nick Street and YongSeog Kim. 2001. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the 7th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 377–382.

Alexey Tsymbal. 2004. *The Problem of Concept Drift: Definitions and Related Work*. Technical Report.

Sergey Tulyakov, Stefan Jaeger, Venu Govindaraju, and David Doermann. 2008. Review of classifier combination methods. In *Machine Learning in Document Analysis and Recognition*. Springer, 361–386.

Jan N. van Rijn, Geoffrey Holmes, Bernhard Pfahringer, and Joaquin Vanschoren. 2015. Having a blast: Meta-learning and heterogeneous ensembles for data streams. In *Proceedings of the 2015 IEEE International Conference on Data Mining (ICDM)*. IEEE, 1003–1008.

Haixun Wang, Wei Fan, Philip S. Yu, and Jiawei Han. 2003. Mining concept-drifting data streams using ensemble classifiers. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 226–235.

Shuo Wang, Leandro L. Minku, and Xin Yao. 2013. A learning framework for online class imbalance learning. In *Proceedings of the 2013 IEEE Symposium on Computational Intelligence and Ensemble Learning (CIEL)*. IEEE, 36–45.

Shuo Wang, Leandro L. Minku, and Xin Yao. 2015. Resampling-based ensemble methods for online class imbalance learning. *IEEE Trans. Knowl. Data Eng.* 27, 5 (2015), 1356–1368.

Shuo Wang, Leandro L. Minku, and Xin Yao. 2016. Dealing with multiple classes in online class imbalance learning. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence. IJCAI/AAAI Press*. 2118–2124.

Zhuang Wang, Koby Crammer, and Slobodan Vucetic. 2012. Breaking the curse of kernelization: Budgeted stochastic gradient descent for large-scale SVM training. *J. Mach. Learn. Res.* 13, 1 (2012), 3103–3131.

Kapil K. Wankhade, Snehlata S. Dongre, Kalpana A. Mankar, and Prashant K. Adakane. 2012. A new adaptive ensemble boosting classifier for concept drifting stream data. In *Proceedings of the 2011 3rd International Conference on Computer Modeling and Simulation (ICCMS'11)*.

Geoffrey I. Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. 2016. Characterizing concept drift. *Data Min. Knowl. Discov.* 30, 4 (2016), 964–994.

Brent Wenerstrom and Christophe Giraud-Carrier. 2006. Temporal data mining in dynamic feature spaces. In *Proceedings of the 6th International Conference on Data Mining, 2006 (ICDM'06)*. IEEE, 1141–1145.

Tom White. 2012. *Hadoop: The Definitive Guide*. O'Reilly Media, Inc.

Gerhard Widmer and Miroslav Kubat. 1996. Learning in the presence of concept drift and hidden contexts. *Mach. Learn.* 23, 1 (1996), 69–101.

David H Wolpert. 1992. Stacked generalization. *Neur. Netw.* 5, 2 (1992), 241–259.

Kevin Woods, Kevin Bowyer, and W. Philip Kegelmeyer Jr. 1996. Combination of multiple classifiers using local accuracy estimates. In *Proceedings of the 1996 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1996 (Proceedings CVPR'96)*. IEEE, 391–396.

Michał Woźniak, Manuel Graña, and Emilio Corchado. 2014. A survey of multiple classifier systems as hybrid systems. *Inf. Fus.* 16 (2014), 3–17.

Lei Yu and Huan Liu. 2003. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *ICML*, Vol. 3. 856–863.

Sun Yue, Mao Guojun, Liu Xu, and Liu Chunnian. 2007. Mining concept drifts from data streams based on multi-classifiers. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops, 2007 (AINAW'07)*, Vol. 2. IEEE, 257–263.

G. Udny Yule. 1900. On the association of attributes in statistics: With illustrations from the material of the childhood society, &c. *Philos. Trans. Roy. Soc. Lond. Ser. A* (1900), 257–319.

Matei Zaharia, Tathagata Das, Haoyuan Li, Timothy Hunter, Scott Shenker, and Ion Stoica. 2013. Discretized streams: fault-tolerant streaming computation at scale. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP'13)*. ACM, New York, NY, USA, 423–438. DOI: http://dx.doi.org/10.1145/2517349.2522737

Peng Zhang, Byron J. Gao, Xingquan Zhu, and Li Guo. 2011a. Enabling fast lazy learning for data streams. In *Proceedings of the 2011 IEEE 11th International Conference on Data Mining (ICDM)*. IEEE, 932–941.

Peng Zhang, Xingquan Zhu, Yong Shi, Li Guo, and Xindong Wu. 2011b. Robust ensemble learning for mining noisy data streams. *Dec. Supp. Syst.* 50, 2 (2011), 469–479.

Weimei Zhi, Huaping Guo, Ming Fan, and Yangdong Ye. 2015. Instance-based ensemble pruning for imbalanced learning. *Intell. Data Anal.* 19, 4 (2015), 779–794.

Zhi-Hua Zhou. 2012. *Ensemble Methods: Foundations and Algorithms*. CRC Press.

Xingquan Zhu, Xindong Wu, and Ying Yang. 2004. Dynamic classifier selection for effective mining from noisy data streams. In *Proceedings of the 4th IEEE International Conference on Data Mining, 2004 (ICDM'04)*. IEEE, 305–312.

Indrė Žliobaitė. 2009. Combining time and space similarity for small size learning under concept drift. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems*. Springer, 412–421.

Indrė Žliobaitė. 2010. Learning under concept drift: An overview. *arXiv Preprint arXiv:1010.4784* (2010).

Indrė Žliobaitė, Albert Bifet, Jesse Read, Bernhard Pfahringer, and Geoff Holmes. 2015. Evaluation methods and decision theory for classification of streaming data with temporal dependence. *Mach. Learn.* 98, 3 (2015), 455–482.

Received December 2015; revised February 2017; accepted February 2017