Pattern Recognition 81 (2018) 280–293

Contents lists available at ScienceDirect

Pattern Recognition

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

Variational inference based bayes online classifiers with concept drift adaptation

Thi Thu Thuy Nguyen[a], Tien Thanh Nguyen[a,b], Alan Wee-Chung Liew[a,*], Shi-Lin Wang[c]

[a] School of Information and Communication Technology, Griffith University, Gold Coast Campus, QLD 4222, Australia
[b] School of Applied Mathematics and Informatics, Hanoi University of Science and Technology, Vietnam
[c] School of Information Security Engineering, Shanghai Jiaotong University, Shanghai, China

A R T I C L E I N F O

Article history:
Received 15 November 2017
Revised 7 March 2018
Accepted 8 April 2018
Available online 9 April 2018

Keywords:
Online learning
Variational inference
Bayesian classifier
Data stream
Concept drift

A B S T R A C T

We present VIGO, a novel online Bayesian classifier for both binary and multiclass problems. In our model, variational inference for multivariate distribution technique is exploited to approximate the class conditional probability density functions of data in an online manner. To handle concept drift that could arise in streaming data, we develop 2 new adaptive methods based on VIGO, which we called VIGOw and VIGOd. While VIGOw naturally adapts to any kind of changing environments, VIGOd maximises the benefit of a static environment as long as it does not detect any change. Extensive experiments on big/medium real-world/synthetic datasets demonstrate the superior performance of our algorithms over many state-of-the-art methods in the literature.

1. Introduction

Nowadays very often data come in the form of streams. Examples of such data can be easily seen in many real-world applications like network traffic, sensor networks, web searches, stock market systems and others. Storing large volumes of streaming data in the machine's main memory is often infeasible and traditional offline method where the prediction is made based on learning the entire training dataset at once becomes impractical. Moreover, offline algorithms are not applicable in real-time learning scenarios where a stream of data is arriving, and predictions must be made before all the data is seen. Therefore, online learning is emerging as an efficient machine learning method for large-scale applications, especially those with streaming data. In an online learning process, predictive models can be updated after the arrival of every new data point (one-by-one) in a sequential fashion or defer until a group of points has arrived (minibatch-by-minibatch) to reduce the effect of noise in the data. They do not require the whole data set to be stored or loaded into memory, but just make use of a single/set of observations and then discard them before the next observations are used.

Online learning can deal with many tasks such as classification, regression and clustering. In this paper, we focus on online classification algorithms with full feedback (i.e. supervised learning). An online classification task usually involves the three main steps:

- *Predict*: When a new instance $\mathbf{x}_t$ arrives, a prediction $\hat{y}_t$ is made using the current model L_t.
- *Calculate the suffered loss:* After making the prediction, the true label y_t is revealed, and the loss $l(y_t, \hat{y}_t)$ can be estimated to measure the difference between the learner's prediction and the revealed true label y_t.
- *Update:* Based on the result of the loss, the learner can use the sample $(\mathbf{x}_t, y_t)$ to update the classification model $(L_t \to L_{t+1})$.

From this framework, we can see that online learning algorithms avoid re-training when adding new data. Besides the requirement of accurate and rapid learning and prediction on-the-fly without storing past instances, another challenge to any online method is that it does not know in advance if the data stream is stationary with stable concepts or evolving over time with changing concepts.

In this paper, we introduce a novel online classifier (VIGO) based on the variational inference (VI) technique. In our framework, two learning phases (learning from past instances (prior information) and from recent instances (through sufficient statistics)) flexibly support each other. They are also naturally separated which offer us the opportunity to focus more on recent information or to detect concept drifts. This resulted in 2 new adaptive

* Corresponding author.
E-mail addresses: thithuthuy.nguyen@griffithuni.edu.au (T.T.T. Nguyen), tienthanh.nguyen2@griffithuni.edu.au (T.T. Nguyen), a.liew@griffith.edu.au (A.W.-C. Liew), wsl@sjtu.edu.cn (S.-L. Wang).

https://doi.org/10.1016/j.patcog.2018.04.007
0031-3203/© 2018 Elsevier Ltd. All rights reserved.

methods named VIGOw and VIGOd, respectively. Our algorithms are *second-order* generative models, where distributions are placed not only on the data of each class but also on the model parameters. They do not require storing more than a single instance in the main memory. We evaluate the performance of our proposed methods by comparing them with recent or well-known online methods including kernel-based DualSGD (Dual space gradient descent) [1], FOGD (Fourier online gradient descent) and NOGD (Nystrom ONLINE GRADIENT DEScent) [2]; ensemble-based BLAST [3]; state-of-the-art second-order linear AROW (adaptive regularisation of weights) [4]; widely used first-order linear PA (passive aggressive learning) [5], the most used decision tree HT (Hoeffding tree) [6]. We also compare our proposed methods with ONBG (online Naïve Bayes for Gaussians) [7]—a first-order generative method. Over and above that, the experiment is extended to estimate the adaptability of VIGOw/d in mining evolving data streams with concept drifts. A number of recent or commonly-used adaptive stream learning methods such as SAM–kNN (kNN classifier with Self Adjusting Memory) [8,9], kNN–PAW (kNN with probabilistic adaptive windowing) [10], ensemble-based DACC (dynamic adaptation to concept changes) [11], and HAT (Hoeffding adaptive tree) [12] are used as benchmark algorithms.

The remainder of this paper is organised as follows. In Section 2, we have a brief review of online methods, especially the ones we used as benchmark algorithms in this paper. After that, the background about Bayesian methods and variational inference techniques is summarised in Section 3. Online variational inference for Gaussian (VIGO) is introduced in Section 4. VIGO with built-in concept drift detector (VIGOd) is the topic of Section 5. Section 6 introduces online variational inference weighted for multivariate Gaussian (VIGOw). Experimental results are provided in Section 7. The final section contains conclusion and suggestions for future work.

2. Related work

2.1. Online classifiers

In this section, we discuss online classifiers in general and about the one we use as benchmark algorithms in our experiments in more details. From the three main steps of online classification, we can see that different online algorithms are mainly distinguished in terms of the different type of loss function $l(y_t, \hat{y}_t)$ and different way of updating $L_t \rightarrow L_{t+1}$. Given a problem instance to be classified represented by a vector $\mathbf{x} = (x_1, x_{2,} \ldots x_D) \in \mathbb{R}^D$, linear methods use the predictive rule: $\hat{y}_t = \text{argmax}_{i\in\{1,\cdots,K\}} \mathbf{w}_i \cdot \mathbf{x}_t$, where K is the number of classes and $\mathbf{w}_i$ is the weight vector of class i $(i = 1, \ldots, K)$. Many popular first-order linear methods such as Perceptron [13], OGD (online gradient descent) [14] and PA (passive aggressive learning) [5] are additive algorithms, i.e., when an instance $\mathbf{x}_t$ is misclassified, the weight vector $\mathbf{w}$ is usually updated by shifting along the direction of $\mathbf{x}_t$: $\mathbf{w} + \alpha_t \mathbf{x}_t \rightarrow \mathbf{w}$, where α_t weighs the misclassified instance. These methods only utilise the first-order information of the received instances, thus maintaining a single point solution for the classification model at any trials. Later on, to better exploit the underlying structures between features, second-order online learning algorithms such as SOP (second-order perceptron) [15], SCW (soft confidence weighted learning) [16], and AROW (adaptive regularisation of weight vectors) [4] have been proposed. Most of the second-order learning algorithms typically assume the weight vector follows a Gaussian distribution $\mathbf{w} \sim \mathcal{N}(\mu, \boldsymbol{\Sigma})$. They not only find the most likely solution for $\mathbf{w}$ but also the distribution of all possible solutions, hence taking advantage of the training data more efficiently. Although linear methods have high time efficiency, they learn linear prediction models which are not flexible enough for many real-world applications.

The limitation of online linear methods in classifying data with nonlinear dependency has motivated the research in online kernel-based methods, which apply linear models in the kernelized feature space to handle the nonlinear separation of data. Conventionally, for the kernel-based predictive model, a set of support vectors (SV) is maintained in main memory and any misclassified new incoming instances are kept. This results in an unbounded SV set during the online learning process. One notable research direction to tackle this key challenge of kernel online learning is to use a fixed-size budget with different budget maintenance strategies (e.g., removal, projection, or merging) [17]. In another direction, a recent method [2] transforms data from the input space to the random-feature space, and then performed stochastic gradient descent in the feature space to create Fourier online gradient descent (FOGD) and Nystrom online gradient descent (NOGD). Recently, to make online kernel methods more scalable, Dual space gradient descent (DualSGD) [1] utilises random features as an auxiliary space to maintain information from data points removed during budget maintenance.

Another widely used approach to deal with the online classification problems is to apply tree-based models. Among incremental trees, Hoeffding tree (HT) [6] is used the most (especially as base learners of ensemble methods, see e.g. [18]) because it has a good performance guaranteed by Hoeffding bound. However, this guarantee does not work for small datasets.

Being one of the most powerful methods, Bayesian classifiers are very flexible generative algorithms which give access to the posterior class probabilities as well as the full data distribution. This information is especially valuable in an online setting, where samples are discarded after use and cannot be retrieved later. Given a problem instance to be classified represented by a vector $\mathbf{x} = (x_1, x_{2,} \ldots x_D) \in \mathbb{R}^D$, a Bayesian classifier based on Bayes' theorem predicts the label y of $\mathbf{x}$ from the label set $\{1, 2, \cdots, K\}$ as

$$y = \text{argmax}_{k\in\{1,2,\cdots,K\}} p(y = k|\mathbf{x})$$
$$\sim \text{argmax}_{k\in\{1,2,\cdots,K\}} p(y = k) p(\mathbf{x}|y = k)$$

where $p(y = k|\mathbf{x})$ is the posterior probability that $\mathbf{x}$ belongs to the class k, $p(y = k)$ is the prior probability of class k, and $p(\mathbf{x}|y)$ is the class conditional probability density function, respectively. Different Bayesian methods are distinguished by the way they approximate $p(\mathbf{x}|y)$. Naïve Bayes is the simplest Bayesian classifier which is called 'naive' because it assumes independence of the attributes given the label. In ONBG (online Naïve Bayes for Gaussians) [7], every attribute x_i of $\mathbf{x}$ follows a univariate Gaussian distribution $p(x_i|y = k) = \mathcal{N}(x_i|\mu_i, \sigma_i^2)$, $i \in \{1, \cdots, D\}$, and the maximum likelihood estimates of parameters μ_i, σ_i^2 are updated on-the-fly based on the training set coming so far. To estimate the parameters of the approximating distributions for $p(\mathbf{x}|y)$, optimisation techniques using latent variables like expectation-maximisation (EM) and variational inference (VI) (see e.g. [19]) are employed. expectation-maximisation (EM) is a popular two-stage iterative technique for finding the parameters of flexible but complicated distributions like mixture of Gaussians (see e.g. [19]) (where all mixing coefficients are positive), or linear combination of continuous or discrete Gaussians [20,21] where the mixing coefficients can be both negative and positive. While also using the coordinate ascent update like EM, VI is a second-order approach where the distribution of each parameter is estimated instead of just the point estimate. To deal with streaming data and big data applications, some algorithms based on incremental EM (see e.g. [19]) and stochastic VI were developed [22–24]. Variational inference often uses the variational lower bound on the marginal likelihood as an objective function, and stochastic variational inference (SVI) [22–24] applies a variant of stochastic gradient descent to this objective for latent Dirichlet allocation in the domain of topic modelling of document collections. Although the stochastic gradient is computed for a single, small subset of data points (documents) at a time, the targeted posterior is still a posterior for D data points where D is the capacity of the full dataset. Therefore, posterior for D' ($D' \neq D$) data points, that has been processed thus far, are not obtained as part of the analysis. Furthermore, this also prevents the application of SVI from learning streaming data where D is unknown in advance. This undesirable property of SVI is overcome by SDA-Bayes [25], a framework for Streaming Distributed Asynchronous computation of a Bayesian posterior. The case study of SDA-Bayes is also latent Dirichlet allocation which shows that SDA-Bayes improves run time with no performance cost compared to SVI. In this paper we introduce a new online Bayesian classifier based on variational inference for multivariate Gaussian which applies VI technique in multiclass classification for a broad range of datasets. The proposed method outperforms the classification accuracy of online linear methods, the time efficiency of kernel based methods, and overcome the limitation of Hoeffding trees on small datasets. In addition, we show the flexibility of the VI framework in adaptation to concept drifts. Our earlier paper about Bayesian online classifiers [26] has presented some preliminary work, which we largely extend here.

The above mentioned online classifiers can be combined in an ensemble framework to create committee classifiers to obtain better performance. To do this, we might train multiple single classifier models and then make prediction using different combining rules. Recently, a Bayesian framework (BE) [27] was proposed for recursively estimating the classifier weights in online learning of a classifier ensemble. Another recent interesting work, named as BLAST [3], studies the use of heterogeneous ensembles comprised of fundamentally different model types. BLAST introduces the Online Performance Estimation framework, which can be used in data stream ensembles to weigh the votes of ensemble members differently across the stream.

2.2. Learning in evolving environments

Besides overcoming the difficulty of sequential learning, online methods are expected to better handle dynamically changing environments with concept drifts. Formally, a concept drift is defined as the change of joint distribution $p(\mathbf{x}, y)$, which is a widely encountered issue in data stream classification. Concept drifts can be caused by many factors such as the changes in class-conditional probability $p(\mathbf{x}|y)$, or more seriously in prior class probability $p(y)$. The latter is usually concerned with the emergence of a new class, the disappearance of an outdated class or the reoccurrence of previously seen classes. Unlike tree-based methods, Bayesian methods, where different classes are being trained separately and independently, are very flexible to adapt to concept drifts in $p(y)$. Indeed, when a new class appears, Bayesian methods just learn a new model for that class, and if a class disappears, they will deactivate the model of that outdated class. However, with a change in class-conditional probability $p(\mathbf{x}|y)$, the adaptation is not trivial. In addition, concept drifts can happen at different rate. For example, abrupt drifts are severe/sudden shift within the class-conditional distribution which may be caused by a malfunctioning sensor; whilst gradual drifts are evolving changes over time (e.g. a sensor slowly wears off and becomes less accurate).

A recent survey on concept drift adaption can be found in [28]. Briefly, as the assumption behind most adaptive learning algorithms is that the newer data are more informative for the current prediction, learning under concept drifts needs strategies for not only updating the predictive model with new information but also forgetting the old information. To do this, a *sliding window* is the most widely used idea, which maintains a window containing the most recent instances, and from it, older instances are removed according to some set of rules. In general, the window size can be fixed or variable over time. Sliding windows of a fixed size save a new instance as it arrives and at the same time drop the oldest one. This strategy is simple and is used as a baseline for the evaluation of new algorithms, but it is not really flexible as usually we do not know in advance the time-scale of change in the streams. In contrast, sliding windows of variable size (see e.g. [29,30]) modify the length of a window over time and often relies on the use of a concept drift detector. Here, the window is shortened if a change is detected (so that the training data reflect the most recent concept) or lengthened otherwise. A different strategy uses a decay function to weight the importance of instances according to their age (see e.g. [31]). In this case, instead of the window size, a decay constant should be chosen to match the rate of change.

It is worth mentioning that there are two ways of applying window techniques for learning: externally and internally. In the first kind of approach, the related concept drift detector (for example, ECDD [32], HDDM [33]) uses a window to monitor the error rate of the current model, which under stable distributions should keep decreasing or at most stabilise. When this error rate grows significantly, the detector declares a change and requires the base learning algorithm to revise or rebuild the model with fresh data. Since this approach uses only the error stream, they consider the underlying online classifiers as a black-box and do not query any of its intrinsic properties. The second drift detection approach is to embed the window *inside* the learning algorithm and continuously monitor the statistics of the learning algorithm in the window. The adaptive sliding window ADWIN [30] is one typical example of this internal approach. If two large enough subwindows W_0 and W_1 of the current learning window W exhibit distinct enough means, ADWIN concludes that the expected values within the windows are different and the older subwindow is discarded. The Hoeffding bound is applied to define the value $\epsilon_{cut} = \sqrt{\frac{1}{2m} \cdot \ln \frac{4|W|}{\delta}}$, and if the difference between the average of the two subwindows is greater than ϵ_{cut}, they are considered as distinct enough. Here, $|W|$ denotes the length of W, m is the harmonic mean of $|W_0|$ and $|W_1|$, i.e. $m = \frac{1}{1/|W_0| + 1/|W_1|}$, and $\delta \in (0, 1)$ is a predefined confidence parameter. Applying ADWIN in Hoeffding tree (HT) results in a Hoeffding adaptive tree (HAT) [12].

Lately, we have witnessed the emergence of kNN-based adaptive online learning methods [8–10,34]. The main reason for this is the simplicity of non-parametric kNN (k-nearest neighbour) method, which assigns an instance to the class that is the most popular among its k nearest neighbours. Unlike Hoeffding trees, a kNN method cannot learn from a stream indefinitely without discarding data as it needs to search through its finite internal buffer of instances to find the neighbours of each coming test instance. This discarding requirement helps kNN adapt to concept drifts to some extent automatically. In fact, [35,36] have demonstrated the amazingly good performance of kNN over a range of methods for data stream learning. In [10], kNN–PAW combines kNN with Probabilistic Adaptive Windowing, where instances from the window are eliminated randomly leading to a mix of recent and older instances while giving greater weight to newer ones. Another very recent adaptive classifier named SAM–kNN (kNN classifier with Self Adjusting Memory) [8, 9] constructs dedicated models for the current and former concepts through Short-Term and Long-Term memories respectively and apply them according to the demands of the given situation.

Another approach to deal with drifting data streams is adaptive ensemble methods where each base learner in the pool keeps adapting to new incoming data until being removed due to bad performance. A new base classifier can also be added to the pool to capture new knowledge. In [11], DACC (dynamic adaptation to concept changes) randomly remove a member from the worst half of the pool and insert a new base learner. In order to control the rate of deletion, two consecutive deletions are separated by a pre-defined number of time steps so that all the learners get mature enough before a deletion operation.

3. Variational inference for multivariate Gaussian

Given a problem instance to be classified represented by a vector $\mathbf{x} = (x_1, x_{2,} \ldots x_D) \in \mathbb{R}^D$. A Bayesian classifier based on Bayes' theorem predicts the label y of $\mathbf{x}$ from the label set $\{1, 2, \cdots, K\}$ as

$$y = \mathrm{argmax}_{k\in\{1,2,\cdots,K\}} p(y = k|\mathbf{x}) \\ \sim \mathrm{argmax}_{k\in\{1,2,\cdots,K\}} p(y = k)p(\mathbf{x}|y = k)$$

where $p(y = k|\mathbf{x})$ is the posterior probability that $\mathbf{x}$ belongs to the class k, $p(y = k)$ is the prior probability of class k, and $p(\mathbf{x}|y)$ is the class conditional probability density function, respectively.

The class prior $p(y = k)$ can often be estimated simply from the fractions of training data in each of the classes. For online setting, at time step t

$$p(y = k) = \frac{c_k}{c} \tag{1}$$

where c_k is the number of instances of class k arriving before $\mathbf{x}_t$, and c is the number of all instances arriving before $\mathbf{x}_t$. Based on the way of approximating $p(\mathbf{x}|y = k)$, $k \in \{1, \ldots, K\}$, we have different Bayesian algorithms. In the literature, we witness the superior performance of second-order online linear methods over the first-order linear methods. First-order algorithms only give a point estimate for each parameter, whereas second-order algorithms give a distribution of all possible solutions for the parameter. Hence, not only the most likely solution (mean or mode) is found but also its confidence (through the variance), thus taking advantage of the training data more efficiently. This induces us to build a second-order online Bayesian method.

Recently, we introduced variational inference (VI) for multivariate Gaussian distribution (VIG) [37] to approximate $p(\mathbf{x}|y = k)$ for each class k. The VIG algorithm has been demonstrated to offer superior performance for batch learning under an ensemble framework. In this paper, we propose a novel VI-based online Bayesian method (VIGO) and its two adaptive versions (VIGOw/d). They offer the same treatment for binary and multiclass cases, i.e. there is no need for any kinds of 1-vs-all or 1-vs-1 strategies which is required by a number of binary classifiers (for example SOP [15]). Before describing our novel online method, we briefly summarise the VIG method [37] here. VIG applies variational inference technique to approximate the multivariate Gaussian model $\mathcal{N}(\mathbf{x}|\mu, \mathbf{\Sigma})$ for $p(\mathbf{x}|y = k)$ of each class $k \in \{1, \ldots, K\}$.

In contrast to maximum-likelihood learning, VI treats the parameters $(\boldsymbol{\mu}, \mathbf{\Sigma})$ as random variables and a prior is placed over the parameters to obtain the posterior distribution $p(\mu, \mathbf{\Sigma}|\mathbb{X})$, where $\mathbb{X} = \{\mathbf{x}_j | j = 1, \ldots, n\}$ is the training set, which is assumed to be drawn independently from the multivariate Gaussian distribution $\mathcal{N}(\mathbf{x}|\mu, \mathbf{\Sigma})$. The idea behind VI method is to approximate the posterior distribution $p(\mu, \mathbf{\Sigma}|\mathbb{X})$ of hidden variables $\boldsymbol{\mu}, \mathbf{\Sigma}$ given observed data $\mathbb{X}$ by a more easily accessible distribution $q(\boldsymbol{\mu}, \mathbf{\Sigma})$ which minimizes the Kullback–Leibler divergence $\mathrm{KL}(q||p)$ between $p(\mu, \mathbf{\Sigma}|\mathbb{X})$ and $q(\boldsymbol{\mu}, \mathbf{\Sigma})$. To minimise $\mathrm{KL}(q||p)$, we maximise $\mathcal{L}(q) = \ln p(\mathbb{X}) - \mathrm{KL}(q||p)$. As $\mathrm{KL}(q||p) \geq 0$, $\mathcal{L}(q)$ is a lower bound on the log marginal probability $\ln p(\mathbb{X})$.

The conjugate prior of a multivariate Gaussian distribution, $p(\boldsymbol{\mu}, \mathbf{\Lambda})$, where $\mathbf{\Lambda} = \mathbf{\Sigma}^{-1}$ is the precision matrix, with unknown $\boldsymbol{\mu}$ and $\mathbf{\Lambda}$ is given by the Gaussian–Wishart distribution: $p(\mu, \mathbf{\Lambda}) = p(\mu|\mathbf{\Lambda})p(\mathbf{\Lambda})$, where $p(\boldsymbol{\mu}|\mathbf{\Lambda})$ is a Gaussian distribution:

$$p(\mu|\mathbf{\Lambda}) = \mathcal{N}\left(\mu|\mathbf{m}_0, (\beta_0\mathbf{\Lambda})^{-1}\right) = (2\pi)^{-\frac{D}{2}}|\beta_0\mathbf{\Lambda}|^{\frac{1}{2}} \\ \exp\left\{-\frac{1}{2}(\mu - \mathbf{m}_0)^{\mathrm{T}}\beta_0\mathbf{\Lambda}(\mu - \mathbf{m}_0)\right\}$$

and $p(\mathbf{\Lambda})$ is a Wishart distribution:

$$p(\mathbf{\Lambda}) = \mathcal{W}(\mathbf{\Lambda}|\mathbf{W}_0, \nu_0) \\ = \mathrm{B}(\mathbf{W}_0, \nu_0)|\mathbf{\Lambda}|^{\frac{(\nu_0 - D - 1)}{2}} \exp\left\{-\frac{1}{2}\mathrm{Tr}\left(\mathbf{W}_0^{-1}\mathbf{\Lambda}\right)\right\},$$

$$\mathrm{B}(\mathbf{W}_0, \nu_0) = |\mathbf{W}_0|^{\frac{-\nu_0}{2}}\left(2^{\frac{\nu_0 D}{2}}\pi^{\frac{D(D-1)}{4}}\prod_{i=1}^{D}\Gamma\left(\frac{\nu_0 + 1 - i}{2}\right)\right)^{-1}$$

where $\mathbf{m}_0$ and β_0 are the D-dimension mean vector and the scale of the precision matrix $\mathbf{\Lambda}$ of the Gaussian distribution $p(\boldsymbol{\mu}|\mathbf{\Lambda})$, $\mathbf{W}_0$ and ν_0 are the $D \times D$-dimension scale matrix and the number of degrees of freedom of the Wishart distribution $p(\mathbf{\Lambda})$, Tr(.)denotes the trace operator of a matrix, and $\Gamma(.)$ denotes the Gamma function defined by $\Gamma(.) = \int_0^\infty x^{t-1}e^{-x}dx$.

From [37], the variational solution for parameters $\boldsymbol{\mu}$ and $\mathbf{\Lambda} = \mathbf{\Sigma}^{-1}$ are given as follows. $\mu \sim \mathrm{q}(\mu) = \mathcal{N}(\mu|\mathbf{m}, \mathbf{H}^{-1})$, is a Gaussian with mean $\mathbf{m}$ and precision $\mathbf{H}$ given by (2) and (3):

$$\mathbf{m} = \frac{\beta_0\mathbf{m}_0 + n\bar{\mathbf{x}}}{\beta_0 + n} \tag{2}$$

$$\mathbf{H} = (\beta_0 + n)\mathbb{E}[\mathbf{\Lambda}]. \tag{3}$$

$\mathbf{\Lambda} \sim q(\mathbf{\Lambda}) = \mathcal{W}(\mathbf{\Lambda}|\mathbf{W}, \nu)$, is a Wishart with the number of degrees of freedom ν and the scale matrix $\mathbf{W}$ given by (4) and (5):

$$\nu = \nu_0 + n + 1 \tag{4}$$

$$\mathbf{W}^{-1} = \mathbf{W}_0^{-1} + (\beta_0 + n)\mathbf{H}^{-1} + \mathbf{S} + \frac{\beta_0 n}{\beta_0 + n}\mathbf{J} \tag{5}$$

where

$$\bar{\mathrm{x}} = \frac{1}{n}\sum_{j=1}^{n}\mathbf{x}_j \tag{6}$$

$$\mathbf{S} = \sum_{j=1}^{n}\left(\mathbf{x}_j - \bar{\mathbf{x}}\right)\left(\mathbf{x}_j - \bar{\mathbf{x}}\right)^{\mathrm{T}} \tag{7}$$

$$\mathbf{J} = (\bar{\mathbf{x}} - \mathbf{m}_0)(\bar{\mathbf{x}} - \mathbf{m}_0)^{\mathrm{T}} \tag{8}$$

Thus, we have expressions for the optimal distributions of $\boldsymbol{\mu}$ and $\mathbf{\Lambda}$, each of which depends on values evaluated with respect to the other distributions such as the expectation $\mathbb{E}[\mathbf{\Lambda}]$ of $\mathbf{\Lambda}$ and the precision $\mathbf{H}$ of $\boldsymbol{\mu}$. To start the iterative re-estimation procedure of VI method, we can make an initial guess for the moment $\mathbb{E}[\mathbf{\Lambda}]$, say, $\mathbb{E}[\mathbf{\Lambda}] = \nu_0\mathbf{W}_0$, and use this to re-compute the distribution $q(\boldsymbol{\mu})$. Given this revised distribution we can then use the precision $\mathbf{H}$ to recompute the distribution $q(\mathbf{\Lambda})$, and so on.

The lower bound $\mathcal{L}(q)$ of the variational inference for the multivariate Gaussian distribution is given by

$$\mathcal{L}(q) = \ln B(\mathbf{W}_0, \nu_0) - \ln B(\mathbf{W}, \nu) - \frac{1}{2}[nD\ln(2\pi) - D\ln(\beta_0) - \nu D \\ + \ln|\mathbf{H}| + \nu\mathrm{Tr}(\mathbf{SW}) + \nu\mathrm{Tr}\left(\mathbf{W}_0^{-1}\mathbf{W}\right) + \frac{\beta_0 n\nu}{\beta_0 + n}\mathrm{Tr}(\mathbf{JW})\Bigg].$$

We have the following algorithm for multivariate Gaussian distribution estimation (Algorithm 1) [37].

In the algorithm above, the 4 variables of $q(\boldsymbol{\mu})$ and $q(\mathbf{\Lambda})$are updated step by step from their initial values. The updating process will stop when the change in lower bound value $\mathcal{L}(q)$ is smaller than a specified threshold ε. Only 3 or 4 iterations are

Algorithm 1 VIG.

```
Input: Dataset $\mathbb{X}$, threshold $\varepsilon$, $\mathbf{m}_0$, $\beta_0$, $\nu_0$, $\mathbf{W}_0$, $\mathbb{E}[\mathbf{\Lambda}] = \nu_0\mathbf{W}_0$
Output: $\mathbf{m}$, $\mathbf{H}$ of $q(\mu) = \mathcal{N}(\mu|\mathbf{m}, \mathbf{H}^{-1})$ and $\mathbf{W}$, $\nu$ of $q(\mathbf{\Lambda}) = \mathcal{W}(\mathbf{\Lambda}|\mathbf{W}, \nu)$
$i := 1$
  for each $i$
    Update $\mathbf{m}$, $\mathbf{H}$ using (2), (3)
    Update $\nu$, $\mathbf{W}$ using (4), (5)
    if $i > 1$ and $\mathcal{L}_i(q) - \mathcal{L}_{i-1}(q) < \varepsilon$
      break;
  end if
$i := i + 1$
  end for
```

typically needed to achieve convergence with a threshold set as $\varepsilon = 1e - 10$ [37]. In the input of Algorithm 1, the default values are used if they are not mentioned otherwise. They are: $\varepsilon = 1e - 10$; $\mathbf{m}_0 = (0, \ldots, 0)^T$- D-dimension vector of zero elements, $\beta_0 = 0$; $\nu_0 = D$; $\mathbf{W}_0 = \mathbf{I}$- a $D \times D$ dimension identity matrix, $j = 1, \ldots K$. Strictly, $\beta_0 > 0$, $\nu_0 > D - 1$ is required to have proper prior. However, the model still runs well when $\beta_0 = 0$ (improper prior) [19]. We choose $\beta_0 = 0$ as it reflects the real initial situation when we have not yet processed any instances.

Estimates for the variables can then be derived in the standard Bayesian ways, e.g. calculating the mean of the distribution to get a single point estimate or deriving a credible interval, highest density region, etc. In practice, ones often choose $\mathbb{E}[\mu] = \mathbf{m}$ as the value of $\boldsymbol{\mu}$, and $\mathbb{E}[\mathbf{\Lambda}] = \nu\mathbf{W}$ as the value of $\mathbf{\Lambda}$ when $\mathcal{N}(\mathbf{x}|\mu, \mathbf{\Lambda}^{-1})$ needs to be evaluated.

4. Online variational inference for multivariate Gaussian (VIGO)

We assume that the class conditional probability density functions $p(\mathbf{x}|y = k)$ are multivariate Gaussians $N(\mu_k, \mathbf{\Lambda}_k^{-1})$ for $k = 1, \ldots, K$, and use variational inference technique to update the distributions of $\boldsymbol{\mu}_k$ and $\mathbf{\Lambda}_k$. Because of the large amount of data in online applications, the assumption about multivariate Gaussian distributions of data in each class becomes even more accurate. As in Bayesian methods, the distribution $p(\mathbf{x}|y = k)$ of each class $k \in \{1, \ldots, K\}$ can be updated independently, we only show the inference for class k, the same process is done for other classes.

To make the algorithm operate in online mode, two main questions need to be answered:

- How to update the model on-the-fly?
- When to update the model?

To answer the first question, we notice that the updating Eqs. (2)–(5) of hyperparameters $\mathbf{m}$, $\mathbf{H}$, ν, $\mathbf{W}$ essentially depend on the sufficient statistics $\bar{\mathbf{x}}$, $\mathbf{S}$, $\mathbf{J}$. Therefore, when a new instance arrives, instead of update $\mathbf{m}$, $\mathbf{H}$, ν, $\mathbf{W}$ directly, we can first update these sufficient statistics $\bar{\mathbf{x}}$, $\mathbf{S}$, $\mathbf{J}$. Considering the results (6)–(8) for the sufficient statistics $\bar{\mathbf{x}}$, $\mathbf{S}$, $\mathbf{J}$, which we will denote by $\bar{\mathbf{x}}_n$, $\mathbf{S}_n, \mathbf{J}_n$ when they are based on n observations, we derive a formulation of sequential learning as below.

Lemma 1. *Sufficient statistics for current training instance* $(\mathbf{x}_t, y_t)$ *can be updated in a sequential manner as follows:*

$$\bar{\mathbf{x}}_t = \frac{t-1}{t}\bar{\mathbf{x}}_{t-1} + \frac{1}{t}\mathbf{x}_t \tag{9}$$

$$\mathbf{S}_t = \mathbf{S}_{t-1} + \frac{t}{t-1}(\mathbf{x}_t - \bar{\mathbf{x}}_t)(\mathbf{x}_t - \bar{\mathbf{x}}_t)^T \tag{10}$$

$$\mathbf{J}_t = (\bar{\mathbf{x}}_t - \mathbf{m}_0)(\bar{\mathbf{x}}_t - \mathbf{m}_0)^T \tag{11}$$

Therefore, when an instance $(\mathbf{x}_t, y_t)$ arrives, VIGO uses $\mathbf{x}_t$ to update the sufficient statistics of the true class y_t and then discards $(\mathbf{x}_t, y_t)$ permanently.

Algorithm 2 VIGO.

```
Initialise: threshold $\varepsilon$, $\mathbf{m}_0^{(j)}$, $\beta_0^{(j)}$, $\upsilon_0^{(j)}$, $\mathbf{W}_0^{(j)}$, $c_j = 1$, $j = 1, \ldots K$, $c = K$;
for $t = 1, 2, \ldots$
  Receive an incoming instance: $\mathbf{x}_t$;
  Predict the class label:
$p(y = j) = c_j/c$; $\hat{y}_t = \operatorname{argmax}_{j\in\{1,\cdots,K\}} p(y = j)p(\mathbf{x}_t|y = j)$;
  Reveal the true class label from the environment: $y_t = k$;
$c_k = c_k + 1$; $c = c + 1$;
  Update sufficient statistics $\bar{\mathbf{x}}^{(k)}, \mathbf{S}^{(k)}, \mathbf{J}^{(k)}$ for class $k$; $n^{(k)} = n^{(k)} + 1$; Discard $\mathbf{x}_t$;
  if $\hat{y}_t \neq y_t$
    Use VIG to update $\mathbf{m}^{(k)}, \mathbf{H}^{(k)}, \upsilon^{(k)}$, $\mathbf{W}^{(k)}$ for class $k$;
  $\mathbf{m}_0^{(k)} = \mathbf{m}^{(k)}$; $\nu_0^{(k)} = \upsilon^{(k)}$; $\mathbf{W}_0^{(k)} = \mathbf{W}^{(k)}$; $\beta_0^{(k)} = \beta_0^{(k)} + n^{(k)}$;
  Reset $\bar{\mathbf{x}}^{(k)}, \mathbf{S}^{(k)}, \mathbf{J}^{(k)}$; $n^{(k)} = 0$;
  end if
end for
```

Moving to the second question, it is natural to update the predictive model when it makes a wrong prediction, i.e. when the 0–1 loss function is > 0. For each incoming instance $\mathbf{x}_t$, we first predict its label $\hat{y}_t$, then reveal the true label, say $y_t = k$. Next, $\mathbf{x}_t$ is used to update sufficient statistics $\bar{\mathbf{x}}^{(k)}, \mathbf{S}^{(k)}, \mathbf{J}^{(k)}$ for the ground truth class k. If the prediction is wrong $(\hat{y}_t \neq y_t)$, we will use VIG to update the hyperparameters $\boldsymbol{\mu}_k$ and $\mathbf{\Lambda}_k$ of $p(\mathbf{x}|y = k) = N(\mu_k, \mathbf{\Lambda}_k^{-1})$ for class k as follows. In the input of Algorithm 1, we do not need dataset $\mathbb{X}$ as we already have the sufficient statistics $\bar{\mathbf{x}}^{(k)}, \mathbf{S}^{(k)}, \mathbf{J}^{(k)}$ (dataset $\mathbb{X}$ is only used to calculate the sufficient statistics). The prior information $\mathbf{m}_0^{(k)}$, $\beta_0^{(k)}$, $\nu_0^{(k)}$, $\mathbf{W}_0^{(k)}$ is extracted from the current state of the model. We then run Algorithm 1 to update the predictive model and assign new values for $\mathbf{m}_0^{(k)}$, $\beta_0^{(k)}$, $\nu_0^{(k)}$, $\mathbf{W}_0^{(k)}$. It is clear that $\mathbf{m}_0^{(k)}$, $\nu_0^{(k)}$ $\mathbf{W}_0^{(k)}$can take the value of the updated $\mathbf{m}^{(k)}$, $\nu^{(k)}$, $\mathbf{W}^{(k)}$. Regarding $\beta_0^{(k)}$, from (2)–(5), we can see that $\beta_0^{(k)}$ weights the prior $\mathbf{m}_0^{(k)}$ in the posterior $\mathbf{m}^{(k)}$ whereas $n^{(k)}$ weights the mean $\bar{\mathbf{x}}^{(k)}$ of the $n^{(k)}$ data points used to build the sufficient statistics. So it is natural for $\beta_0^{(k)}$ to be equal to the number of past instances used to build the model, that means its updating equation has the following form: $\beta_0^{(k)} = \beta_0^{(k)} + n^{(k)}$. After being used to update the model, the sufficient statistics $\bar{\mathbf{x}}^{(k)}, \mathbf{S}^{(k)}, \mathbf{J}^{(k)}$ and $n^{(k)}$ need to be reset (to zero value), and ready for incremental update by new incoming instances of class k. Below, the pseudo-code of VIGO is given (Algorithm 2).

5. VIGO with built-in concept drift detector (VIGOd)

It can be seen that VIGO learn its model on the whole data stream, where the past is contained in the prior information and the present in the sufficient statistics. Indeed, for each class, the mean calculated by (2) is the average value of all instances belonging to that class. In a stationary data stream learning, this lossless property is desirable as it shows that the model learned by an online method can be asymptotically arbitrarily close to that of the offline batch method regardless of the incremental training order and the requirement of not storing data for online learners.

However, in evolving data stream learning, the lossless property can make the algorithm slow to adapt to concept drifts because adaptation happens only by the natural dilution of old concepts due to the new incoming data. In this section, a simplified ADWIN-based technique is embedded in VIGO to make it adaptive, we call this adaptive VIGO version as VIGOd.

First, it can be seen that right before we update the model for any class, say class k, the VI framework naturally gives us two windows: the prior information window W_p has $\beta_0^{(k)}$ instances with mean $\mathbf{m}_0^{(k)}$, and the sufficient statistics window W_s has $n^{(k)}$instances with mean $\bar{\mathbf{x}}^{(k)}$. This strongly suggests us to combine VIGO with ADWIN [30], which totally avoids the main limitation (memory requirement) caused by ADWIN. Indeed, as mentioned in [30], a window W has a total of $|W|$ possibilities for partitioning it into W_0 and W_1, where $|\cdot|$ denotes the cardinality or the length of a set. In our VI framework, instead of doing $|W|$ different tests for cutting W into W_0 and W_1, the test is done only 1 time to check if the older subwindow W_p can be dropped from the current learning window $W = W_p \cup W_s$.

The inputs of ADWIN [30] are a confidence value $\delta \in (0, 1)$ and a sequence of real values $W = \{z_1, z_2, \ldots, z_n\}$. ADWIN requires that the input sequence to be bounded, i.e. with all $z_i \in [0, 1]$, which can be achieved by rescaling the original data. Let μ_t be the expected value of z_t when it is drawn according to a distribution D_t, $\hat{\mu}_W$ be the observed average of the elements in W, and μ_W the unknown average of μ_t for $t \in W$. Consider a partitioning with 2 subwindows W_0 and W_1, $W = W_0 \cup W_1$, with $|W_0| = n_0$, $|W_1| = n_1$, $n_0 + n_1 = n$, where W_1 contains the most recent items. Using the Hoeffding bound, ADWIN defines $n_h = \frac{1}{1/n_0 + 1/n_1}$, which is the harmonic mean of n_0 and n_1, and $\epsilon_{cut} = \sqrt{\frac{1}{2n_h}.\ln\frac{4}{\delta}}$. ADWIN then performs a statistical test for the difference in distributions in W_0 and W_1 by checking whether the observed average in both subwindows differs by more than the threshold ϵ_{cut}. If yes, ADWIN declares that there is a concept drift between W_0 and W_1, and the model using ADWIN will remove W_0 (the older subwindow) from the whole learning window W, and only learn on W_1. This test is performed $|W|$ times for the $|W|$ possible partitioning.

To apply this technique in VIGOd, we notice that our instances have D real attributes. If there is at least an attribute i having the absolute difference between its values for the observed means $\mathbf{m}_0^{(k)} = (m_{01}^{(k)}, m_{02}^{(k)}, \ldots, m_{0D}^{(k)})$ of the instances in W_p and $\bar{\mathbf{x}}^{(k)} = (\bar{x}_1^{(k)}, \bar{x}_2^{(k)}, \ldots, \bar{x}_D^{(k)})$ of that in W_s being greater than ϵ_{cut}, i.e. $|m_{0i}^{(k)} - \bar{x}_i^{(k)}| \geq \epsilon_{cut}$, a drift is detected and we reset the prior information for the class distribution of class k, i.e. $\beta_0 = 0$, that means the model for class k is now updated only based on the sufficient statistics computed from current instances (see (2)–(5)). At the same time the prior class distributions $p(y = j),\ \forall j = 1, \ldots K$ in (1) are reset as follows: $c_j = 1,\ c = K$ and $p(y = j) = 1/K$.

To avoid problems with multiple hypothesis testing (since we will do the test for D different attributes) and we want the global error to be below δ, we use $\delta' = \delta/D$ and calculate n_h and ϵ_{cut} for class k of VIGOd as follows:

$$n_h = \frac{1}{1/\beta_0^{(k)} + 1/n^{(k)}};\ \delta' = \frac{\delta}{D};\ \epsilon_{cut}^{(k)} = \sqrt{\frac{1}{2n_h}.\ln\frac{4}{\delta'}} = \sqrt{\frac{1}{2n_h}.\ln\frac{4D}{\delta}} \tag{12}$$

As in [30], we have a theoretical result about our adaptive ADWIN as below.

Lemma 2. *For every class k, with data window $W = W_p \cup W_s$, $|W_p| = \beta_0^{(k)}$, $|W_s| = n^{(k)}$, we have*

1. (False positive rate bound). If μ_t remains constant within W, the probability that adaptive ADWIN shrinks the window from W to W_s is at most δ.
2. (False negative rate bound). Suppose that for $W = W_p \cup W_s$ we have $|\mu_{W_p} - \mu_{W_s}| > 2\epsilon_{cut}$. Then with probability at least $1 - \delta$, a change is detected and adaptive ADWIN shrinks W to W_s.

The proof of Lemma 2 is omitted as it is similar to that of Theorem 3.1 in [30].

We choose the default value for confidence $\delta = 0.3$ as if δ is too big, the *False positive rate bound* and *False negative rate bound* in Lemma 2 would not be meaningful practically. On the other hand, if δ is too small, since an instance can have many attributes, $\delta' = \frac{\delta}{D}$ can be very small, ϵ_{cut} would be big and the adaptive ADWIN would be too conservative.

Algorithm 3 VIGOd.

```
  Initialise: minibatch size |B|, threshold ε, confidence δ
$\mathbf{m}_0^{(j)}, \beta_0^{(j)}, \upsilon_0^{(j)}, \mathbf{W}_0^{(j)}, c_j = 1,\ n^{(j)} = 0,\ j = 1, \ldots K;\ c = K;$
  Building step:
$t = 1;$
  while ∃ class having less than |B| arriving instances
    Receive an incoming instance: $\mathbf{x}_t$;
    Predict the class label:
$p(y = j) = c_j/c;\ \hat{y}_t = \underset{j \in \{1,\cdots,K\}}{\mathrm{argmax}}\, p(y = j)p(\mathbf{x}_t|y = j);$
    Reveal the true class label from the environment: $y_t = k$;
$c_k = c_k + 1; c = c + 1;$
    Update sufficient statistics for class $k$; $n^{(k)} = n^{(k)} + 1$; Discard $\mathbf{x}_t$
    if $(\hat{y}_t \neq y_t)$ or $n^{(k)} = |B|$
      Update model for class $k$;
    end if
$t = t + 1;$
    end while
$t_{next} = t$
  For $t = t_{next},\ t_{next} + 1, \ldots$
    Receive an incoming instance: $\mathbf{x}_t$;
    Predict the class label:
$p(y = j) = c_j/c;\ \hat{y}_t = \underset{j \in \{1,\cdots,K\}}{\mathrm{argmax}}\, p(y = j)p(\mathbf{x}_t|y = j);$
    Reveal the true class label from the environment: $y_t = k$;
$c_k = c_k + 1; c = c + 1;$
    Update sufficient statistics for class $k$; $n^{(k)} = n^{(k)} + 1$; Discard $\mathbf{x}_t$
    if $n^{(k)} = |B|$
      Calculate $\epsilon_{cut}^{(k)}$ using (12)
      if concept drift detected
$\beta_0^{(k)} = 0;$
$c_j = 1,\ \forall j = 1, \ldots K;\ c = K;$
    end if
    Update model for class $k$;
  end if
end for
procedure Update model for class $k$;
  Use VIG to update $\mathbf{m}^{(k)}, \mathbf{H}^{(k)}, \upsilon^{(k)}, \mathbf{W}^{(k)}$ for class $k$;
    $\mathbf{m}_0^{(k)} = \mathbf{m}^{(k)};\ \upsilon_0^{(k)} = \upsilon^{(k)};\ \mathbf{W}_0^{(k)} = \mathbf{W}^{(k)};\ \beta_0^{(k)} = \beta_0^{(k)} + n^{(k)};$
  Reset $\bar{\mathbf{x}}^{(k)}, \mathbf{S}^{(k)}, \mathbf{J}^{(k)};\ n^{(k)} = 0;$
end procedure
```

To ensure the length of W_s is not too small, as well as to reduce the time cost compared with VIGO, VIGOd only updates model for class k if $n^{(k)}$=|B|, $k = 1, .., K$, where $|B|$ is predefined. That means parameters$\mathbf{m}^{(k)}, \mathbf{H}^{(k)}, \upsilon^{(k)}, \mathbf{W}^{(k)}$ are not updated until the sufficient statistics $\bar{\mathbf{x}}^{(k)}, \mathbf{S}^{(k)}, \mathbf{J}^{(k)}$ have collect information from $|B|$recent instances of class k. Generally, there is a trade-off in the choice of $|B|$: the greater $|B|$ is, the shorter the running time but the lower the accuracy as the algorithm delays updating the model until it collects enough instances for at least one of the classes. Following the state-of-the-art Hoeffding tree where in default, each node waits for $n_{min} = 200$ instances to come before checking for any update, in our proposed method we use default value of $|B| = 200$.

In addition, to build a stable model, before updating the model when $|B| = 200$, we run the *building step* until all classes have collected information from at least $|B|$ instances. During the building step, the proposed algorithm will update the model whenever it make a mistake and then reset $n^{(k)}$= 0. Below, the pseudo-code of VIGOd is given (Algorithm 3).

In the *Initialise* part of Algorithm 3, the default values are used if they are not mentioned otherwise. They are: |B|=200; $\varepsilon = 1e-1$; $\delta = 0.3$; $\mathbf{m}_0^{(j)} = (0, \ldots, 0)^T$- D-dimension vector of zero elements, $\beta_0^{(j)} = 0,\ \upsilon_0^{(j)} = D,\ \mathbf{W}_0^{(j)} = \mathbf{I}$ - $D \times D$dimension identity matrix, $j = 1, \ldots K$.

6. Online variational inference weighted for multivariate Gaussian (VIGOw)

In this section, we present VIGOw, which also updates the model for class k if $n^{(k)}$ reaches a predefined value |B|, but it does not use a built-in drift detector. VIGOw is focused on the idea of

Algorithm 4 VIGOw.

```
  Initialise: minibatch size |B|, threshold $\varepsilon$, confidence $\delta$
$\mathbf{m}_0^{(j)}, \beta_0^{(j)}, \upsilon_0^{(j)}, \mathbf{W}_0^{(j)}, c_j = 1, j = 1, \ldots K; c = K;$
  Building step:
  $t = 1;$
    while $\exists$ class having less than |B| arriving instances
    Receive an incoming instance: $\mathbf{x}_t$;
Predict the class label:
$p(y=j) = c_j/c;\ \hat{y}_t = \underset{j \in \{1,\cdots,K\}}{\operatorname{argmax}}\, p(y=j)p(\mathbf{x}_t|y=j);$
    Reveal the true class label from the environment: $y_t = k$;
$c_k = c_k + 1; c = c + 1;$
    Update sufficient statistics for class $k$; $n^{(k)} = n^{(k)} + 1$; Discard $\mathbf{x}_t$;
    if $(\hat{y}_t \neq y_t)$ or $n^{(k)} = |B|$
      Update model for class $k$;
    end if
    $t = t + 1;$
    end while
$t_{next} = t$
  For $t = t_{next}, t_{next} + 1, \ldots$
    Receive an incoming instance: $\mathbf{x_t}$;
    Predict the class label:
$p(y=j) = (c_j^{old} + c_j)/(|B| + c);\ \hat{y}_t = \underset{j \in \{1,\cdots,K\}}{\operatorname{argmax}}\, p(y=j)p(\mathbf{x}_t|y=j);$
    Reveal the true class label from the environment: $y_t = k$;
$c_k = c_k + 1; c = c + 1;$
    if $c = |B|$
    $c_j^{old} = c_j,\ c_j = 0,\ \forall j = 1, .., K;\ c = 0$
    end if
    Update sufficient statistics for class $k$; $n^{(k)} = n^{(k)} + 1$; Discard $\mathbf{x}_t$;
    if $n^{(k)} = |B|$
      Update model for class $k$;
    end if
  end for
  procedure Update model for class $k$;
    Use VIG to update $\mathbf{m}^{(k)}, \mathbf{H}^{(k)}, \upsilon^{(k)}, \mathbf{W}^{(k)}$ for class $k$;
    $\mathbf{m}_0^{(k)} = \mathbf{m}^{(k)};\ \upsilon_0^{(k)} = \upsilon^{(k)};\ \mathbf{W}_0^{(k)} = \mathbf{W}^{(k)};\ \beta_0^{(k)} = |B|$
    Reset $\bar{\mathbf{x}}^{(k)}, \mathbf{S}^{(k)}, \mathbf{J}^{(k)};\ n^{(k)} = 0;$
  end procedure
```

using a decay function to weigh the importance of instances according to their age [31]. In a decay function, a decay factor α ($0 < \alpha < 1$) is multiplied to the weight of the old information to phase it out. However, the variational inference framework allows us to put more emphasis on the recent instances without using a decay factor. To do that, instead of update $\beta_0^{(k)} = \beta_0^{(k)} + n^{(k)}$ for the model of class k as in VIGO, we just assign $\beta_0^{(k)} := |B|$. By doing this, in (2) we can see that the mean $\bar{\mathrm{x}}$ of |B| recent instances in the sufficient statistics window W_s has the same weight as the mean $\mathbf{m}_0^{(k)}$ of all old instances in the prior information window W_p even though $|W_p| > |B|$. Over time, the old instances get smaller and smaller weight in (2). In fact, each recent instance in W_s has weight$=1$, and all the instances in W_p has weight <1 so that the weight sum of all instances in W_p is |B| and newer instances in W_p have bigger weight than the older instances.

To build an adaptive online method, we also update the prior class probability $p(y=j),\ j = 1, .., K$ based on a sliding window of size |B| as follows. After every |B| instances arrive, we use c_j^{old} to record the number of instances of class j in the previous window: $c_j^{old} = c_j,\ \ j = 1, ..., K;$ then we reset $c_j = 0,\ \ j = 1, .., K; c = 0$. At each time step t: $p(y=j) = (c_j^{old} + c_j)/(|B| + c),\ \ j = 1, .., K$. Similar to VIGOd, VIGOw also use default $|B| = 200$.

Below, the pseudo-code of VIGOw is given (Algorithm 4).

7. Experiments

7.1. Experiment for stationary data stream learning

7.1.1. Dataset

To evaluate the performance of the proposed methods in different scenarios (where data may be very rare or extremely redundant), we conduct experiments on 30 datasets (see Table 1). Two of which (*Poker* and *Airlines*) are large-scale datasets with millions of data points, and the others are of medium and small size. They can be downloaded from LIBSVM [38] and UCI [39] repositories, except *Airlines*. Information about flights in the year of 2008 was obtained from American Statistical Association's website (http://stat-computing.org/dataexpo/2009/), and 8 features were extracted as in [40] to create the *Airlines* dataset. A flight will be considered as *delayed* if the delay time of departure is above 15 minutes, and *non-delayed* otherwise.

7.1.2. Experimental setting

Five diverse online learning classifiers, namely most widely used first-order linear PA [5], state-of-the-art second-order linear AROW [4], the most-used decision tree HT [6], recent heterogeneous ensemble BLAST [3], and the simplest first-order generative ONBG (online Naïve Bayes for Gaussians) (described in [7], however in our experiment we do not apply random projection). The code of HT, BLAST, PA and AROW is available in the library MOA [41] and LIBOL [42] (noting that linear methods like PA and AROW have different implementation for binary and multiclass cases). Default parameters were used for benchmark methods if available. For VIGOd/w, we also use the default parameters (batch size $|B| = 200$, confidence $\delta = 0.3$) as discussed in the earlier sections. All datasets are normalized into the range [0, 1] before being run by VIGOd.

Testing and training are done simultaneously from the first coming instance (i.e. prequential evaluation) and sequential data come one-by-one. To enable fair side-by-side comparisons between the different methods, we run the experiment 10 times for each algorithm on each dataset with different random permutations of the training data instances. The mistake rate (%) and time cost (seconds) are taken in average together with the standard deviation over all runs.

When comparing the proposed methods with any benchmark algorithms, Wilcoxon signed-rank test [43] with the level of significance set to 0.05 is used. In this test, the null hypothesis is that the performances of two methods are not different, and the alternative hypothesis is that they are different.

7.1.3. Accuracy

Table 2a,b,c present the mean and standard deviation (STD) of mistake rates for the 5 benchmark algorithms (AROW, PA, HT, BLAST, ONBG) and VIGO(d/w) over the 30 datasets. The best results (lowest mistake rates) are shown in bold.

From the average results over all 30 datasets (see the last rows of Table 2c and Fig. 1), it can be seen that in general, VIGO has the lowest mistake rate. Among the 5 benchmark method, ensemble BLAST achieves the best performance (29.67%), which is still over 5.5% worse than VIGO (24.14%).

In the default setting of MOA [41], BLAST combines 5 base classifiers: Naïve Bayes, perceptron, stochastic gradient descent, kNN, and Hoeffding tree, so it is not surprising that BLAST achieves a better average accuracy than any of its base learner, in particular, the state-of-the-art Hoeffding tree. However, both methods are far worse than VIGO in learning small sequential data where Hoeffding bound often need to wait for more coming instances to make an effective split at a node. To illustrate, for 3 rare datasets, namely Libras (360 data points, 15 class, 90 features), Soybean-large (307 instances, 19 class, 35 features) and Zoo (101 instances, 7 class, 16 features), HT and BLAST have the mistake rates higher than that of VIGO from 10 to 17%. This demonstrates the much superior ability of VIGO in exploiting the underlying structure of data.

Among the linear methods, first-order PA performs more poorly than second-order AROW, and the same happens to Bayesian

Table 1
Information of datasets used in evaluation.

Dataset	#Attributes	#Classes	#Observations	Dataset	#Attributes	#Classes	#Observations
Airlines	8	2	5,929,413	Marketing	13	9	6876
Balance	4	3	625	Nursery	8	5	12,960
Chess-krvk	6	18	28,056	Penbased	16	10	10,992
Cod-rna	8	2	331,152	Poker	10	10	1,025,010
Conn-bench-vowel	10	11	528	Skin-nonskin	3	2	245,057
Contraceptive	9	3	1473	Sonar	60	2	208
Dermatology	34	6	358	Soybean-large	35	19	307
Glass	9	6	214	Tae	5	3	151
Ijcnn1	22	2	141,691	Texture	40	10	5500
Ionosphere	34	2	351	Tic-tac-toe	9	2	958
Iris	4	3	150	Vehicle	18	4	846
Led7digit	7	10	500	Vowel	13	11	990
Letter	16	26	20,000	Waveform	21	3	5000
Libras	90	15	360	Wine-white	11	7	4898
Magic	10	2	19,020	Zoo	16	7	101

Table 2a
Mistake rate (%) in comparison with AROW, PA, HT, BLAST and ONBG.

Dataset	AROW	PA	HT	BLAST	ONBG	VIGO	VIGOd	VIGOw
Airlines	18.15	28.03	**17.60**	17.72	19.11	18.68	18.85	18.71
	±0.00	±0.01	**± 0.02**	± 0.02	±0.02	±0.01	±0.01	±0.01
Balance	13.10	21.34	13.50	13.54	13.28	**11.28**	12.56	**11.28**
	±0.74	±1.07	±0.96	±0.86	±0.98	**±0.45**	±0.37	**±0.45**
Chess-krvk	81.10	85.24	70.13	64.85	70.80	61.51	**60.75**	61.51
	±0.90	±0.18	± 0.14	±0.25	±0.21	±0.25	**±0.22**	±0.25
Cod-rna	5.07	21.06	5.29	5.27	23.53	**4.87**	4.92	4.91
	±0.01	±0.05	±0.09	±0.09	±0.07	**±0.01**	±0.01	±0.01
Conn-bench-vowel	62.03	76.48	40.64	40.80	40.97	**21.00**	39.07	**21.00**
	±1.93	±1.42	±0.72	±0.77	±0.82	**±0.99**	±1.50	**± 0.99**
Contraceptive	51.19	63.56	53.63	53.35	53.29	**50.10**	52.03	50.22
	±0.71	±1.18	±0.76	±0.70	±0.89	**±0.76**	±0.83	± 0.94
Dermatology	8.27	52.88	14.16	11.54	6.76	9.11	**6.34**	9.11
	±1.01	±2.21	±0.61	±0.73	±0.61	±0.64	**±0.59**	±0.64
Glass	48.97	72.48	53.60	46.36	50.89	**45.05**	48.88	**45.05**
	±4.68	±2.84	±3.69	±2.04	±2.07	**±2.16**	± 1.92	**±2.16**
Ijcnn1	8.45	10.47	4.77	**4.70**	10.13	8.25	9.43	8.21
	±0.04	±0.06	±0.42	**± 0.38**	±0.16	±0.07	±0.05	± 0.04
Ionosphere	17.35	22.54	18.35	18.40	15.44	**10.43**	**10.43**	**10.43**
	±0.76	±1.00	±1.71	±1.83	±1.91	**±0.76**	**±0.76**	**±0.76**

Table 2b
Mistake rate (%) in comparison with AROW, PA, HT, BLAST and ONBG.

Dataset	AROW	PA	HT	BLAST	ONBG	VIGO	VIGOd	VIGOw
Iris	12.40	39.53	9.40	9.60	7.93	**7.53**	10.13	**7.53**
	±2.85	±3.60	±1.38	± 1.47	± 1.31	**±1.40**	±7.77	**±1.40**
Led7digit	32.78	53.02	35.54	34.26	34.04	33.16	33.16	33.16
	±1.20	±2.15	±0.97	±0.94	±1.08	±0.95	±0.95	±0.95
Letter	46.87	53.09	36.83	37.00	36.74	**13.73**	20.85	13.96
	±2.21	±0.13	±0.23	±0.28	±0.25	**±0.21**	±0.51	±0.23
Libras	47.64	84.78	47.14	47.53	70.81	30.64	**29.53**	30.64
	±3.05	±1.38	±1.68	±2.07	±0.75	±8.24	**±1.19**	±8.24
Magic	21.76	38.34	21.78	**19.83**	27.40	21.77	22.26	21.72
	±0.07	±0.32	±0.72	**±0.54**	±0.16	±0.12	±0.09	±0.12
Marketing	71.48	78.40	69.80	70.00	69.78	**69.29**	69.05	69.38
	±1.12	±0.56	±0.24	±0.39	±0.23	**±0.36**	±0.37	±0.40
Nursery	24.64	38.29	9.43	9.13	9.53	6.68	**6.54**	6.68
	±0.30	±0.41	±0.27	±0.26	±0.16	±0.13	**±0.13**	±0.13
Penbased	18.51	17.73	12.81	6.39	14.74	3.68	**2.51**	3.75
	±3.58	±0.26	±0.51	±0.12	±0.16	±0.11	**± 0.11**	±0.12
Poker	50.88	56.97	46.66	46.68	49.89	**45.17**	**45.18**	45.89
	±0.66	±0.03	±1.35	±1.35	±0.00	**± 0.01**	**± 0.01**	±0.06
Skin-nonskin	9.17	15.53	1.07	**0.85**	7.60	1.65	1.56	1.67
	±0.04	±0.07	±0.10	**±0.09**	±0.01	±0.02	±0.02	±0.02

Table 2c
Mistake rate (%) in comparison with AROW, PA, HT, BLAST and ONBG.

Dataset	AROW	PA	HT	BLAST	ONBG	VIGO	VIGOd	VIGOw
Sonar	26.88	43.08	32.45	33.22	35.72	24.90	**22.26**	24.90
	±2.33	±1.71	±1.75	±1.60	±2.72	±1.91	**±1.41**	±1.91
Soybean-large	24.89	56.03	33.35	34.11	26.55	22.31	**21.11**	22.31
	±1.90	±2.05	±1.62	±1.64	±2.27	±0.96	**±1.12**	±0.96
Tae	62.32	65.36	52.65	55.37	51.85	**50.86**	51.46	**50.86**
	±2.79	±2.75	±1.57	±3.23	±2.43	**±1.59**	±1.78	**±1.59**
Texture	**2.97**	25.87	22.80	7.29	24.87	**2.68**	8.17	2.98
	±0.45	±0.67	±0.46	±0.30	±0.41	**±0.24**	±1.38	±0.47
Tic-tac-toe	33.99	44.10	31.55	31.27	31.00	**26.96**	27.37	27.14
	±0.63	±1.59	±0.96	±0.88	±0.94	**±0.65**	±1.02	±0.77
Vehicle	32.80	69.42	53.79	37.45	53.56	**21.99**	33.36	**21.99**
	± 3.07	±0.98	±1.61	±1.48	±1.83	**±0.81**	±1.51	**±0.81**
Vowel	62.88	80.29	40.78	40.89	41.08	**21.55**	32.75	**21.55**
	±2.34	±1.19	±0.84	±0.86	±0.78	**±0.65**	±0.99	**±0.65**
Waveform	16.01	20.61	19.28	18.11	19.23	16.72	**15.76**	16.95
	±0.53	±0.43	±0.11	±0.18	±1.11	±0.36	**±0.21**	±0.23
Wine-white	49.76	67.22	55.59	50.02	50.57	**48.84**	50.80	**48.84**
	±0.85	±0.60	±0.34	±0.62	±0.20	**±0.81**	±1.14	**±0.81**
Zoo	15.64	29.31	24.35	24.65	15.45	13.86	**13.27**	13.86
	±0.97	±1.99	±1.61	±1.96	±2.13	±1.60	**±1.10**	±1.60
Average	32.60	47.70	31.62	29.67	32.75	**24.14**	26.01	**24.21**
	±1.39	±1.10	±0.91	±0.93	±0.89	**±0.91**	±0.97	**± 0.92**

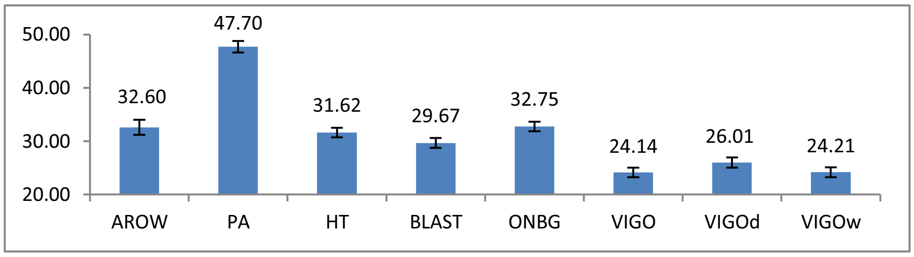

Fig. 1. Average mean ± average STD over 30 datasets with respect to mistake rate.

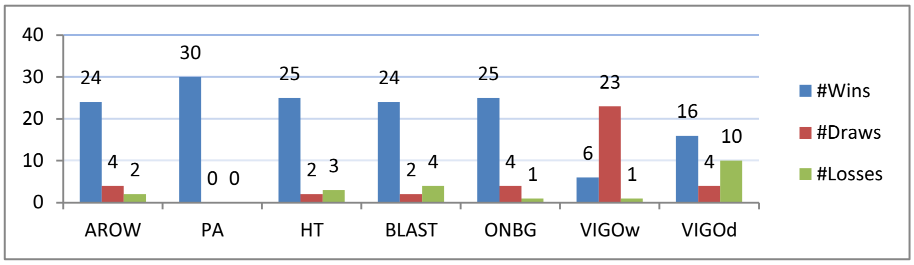

Fig. 2. Statistical test results comparing VIGO to the benchmark methods with respect to mistake rate.

methods where first-order ONBG is outperformed by second-order VIGO. This suggests that summarizing all the information in the training data into a single point estimate of the model is less effective in exploiting the training data. The lower average accuracy of AROW in comparison with VIGO also confirms the limitation of linear models in learning real-world datasets.

We verify the experiment result statistically by Wilcoxon Sign Rank test [43] between VIGO(d/w), and the 5 benchmark methods. For reference, the resulting p-values are shown in Tables S1-3 in the supplementary material, where those <0.05 indicates a significant difference between compared methods. For each dataset, a mentioned algorithm wins (loses to) a benchmark algorithm if p-value <0.05 and in all runs, it gets lower mistake rate more (less) times than the benchmark algorithm; otherwise the two algorithms have a draw. Based on the statistical test, we depict results comparing VIGO(d/w) with the benchmark algorithms in Figs. 2–4. To compare 2 algorithms, say VIGO and AROW, we look at 3 columns (named #Wins, #Draws, #Losses) of different heights (24, 4, 2) belonging to AROW in Fig. 2. This illustrates that over 30 datasets, VIGO wins/ has a draw with/ loses to AROW on 24/4/2 datasets respectively. From all the three figures, it can be seen that VIGO and VIGOd/w significantly win more time than any benchmark algorithms.

Between themselves, VIGO performs better than VIGOd and VIGOw, and VIGOw slightly outperforms VIGOd with 14 wins/6 draws/10 losses.

7.1.4. Time complexity

To estimate the time efficiency of all mentioned methods, the time cost result for each dataset is presented in Table S4 in the supplementary material, and the average time costs are shown in Fig. 5. Clearly, ONBG and BLAST are the most time-consuming algorithms. The execution time of VIGO almost doubles that of VIGOd/w, but is still quite moderate. AROW, PA, HT, VIGOd and VIGOw are fast algorithms.

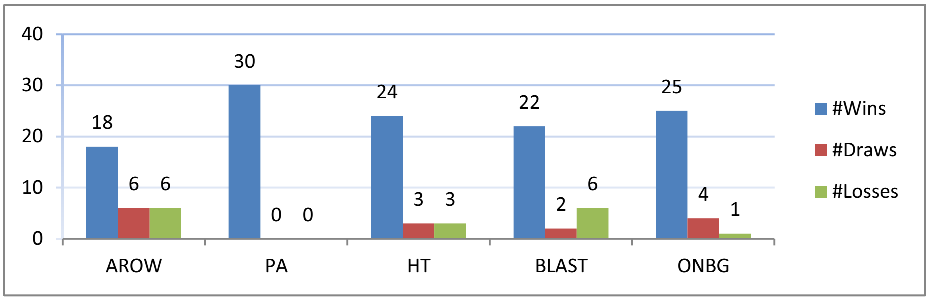

Fig. 3. Statistical test results comparing VIGOd to the benchmark methods with respect to mistake rate.

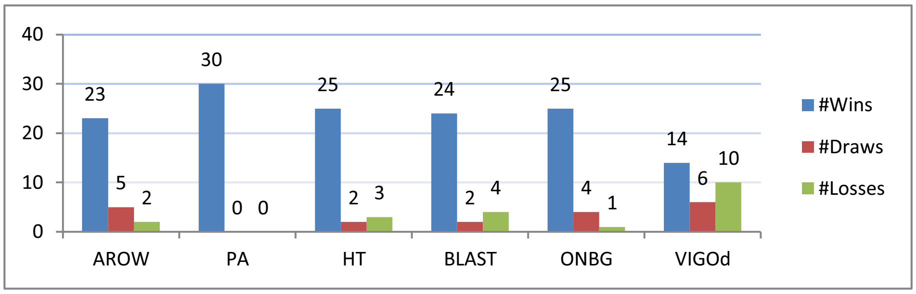

Fig. 4. Statistical test results comparing VIGOw to the benchmark methods with respect to mistake rate.

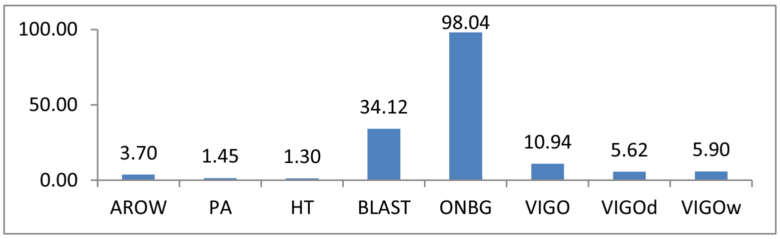

Fig. 5. Average runtime (seconds) of VIGOw/d and benchmark algorithms in static setting.

Table 3
Mistake rate (%) and Time cost (seconds) in comparison with FOGD, NOGD and DualSGD.

Dataset Algorithm	Cod-rna Mistake rate (Time)	Ijcnn1 Mistake rate (Time)	Poker Mistake rate (Time)	Airlines Mistake rate (Time)
FOGD	7.15 ± 0.03 (53.45)	9.41 ± 0.03 (25.93)	52.28 ± 0.04 (928.89)	20.98 ± 0.01 (1270.75)
NOGD	7.83 ± 0.06 (105.18)	10.43 ± 0.08 (59.36)	**44.90±0.16 (4920.33)**	25.56 ± 0.01 (3553.50)
DualSGD-Hinge	4.92 ± 0.25 (28.29)	8.35 ± 0.20 (12.12)	46.73 ± 0.22 (139.87)	19.28 ± 0.00 (472.21)
DualSGD-Logit	**4.83±0.21 (31.96)**	8.82 ± 0.24 (13.30)	46.65 ± 0.14 (133.50)	19.28 ± 0.00 (523.23)
VIGO	4.87 ± 0.01 (6.99)	8.25 ± 0.07 (3.87)	45.17 ± 0.01 (95.76)	**18.68±0.01 (207.01)**
VIGOd	4.92 ± 0.01 (5.55)	9.43 ± 0.05 (2.42)	45.18 ± 0.01 (50.59)	18.85 ± 0.01 (96.72)
VIGOw	4.91 ± 0.01 (5.91)	**8.21±0.04 (2.64)**	45.89 ± 0.06 (49.28)	18.71 ± 0.01 (105.64)

7.1.5. . Comparison with recent kernel-based FOGD, NOGD, DualSGD

Table 3 shows the result of recently proposed kernel-based DualSGD-Hinge [1], DualSGD-Logit [1], FOGD[2], and NOGD [2]. As only 4 datasets, namely *Ijcnn1, Cod-rna, Poker*, and *Airlines* were used for these benchmark algorithms [1], we only conducted the comparison on the 4 datasets. It is worth mentioning that all datasets are normalised into the range [0, 1] before being run by FOGD, NOGD, DualSGD-Hinge, DualSGD-Logit, and VIGOd.

It can be seen from Table 3 that compared to each of the 4 benchmark algorithms (FOGD, NOGD, DualSGD-Hinge, DualSGD-Logit), each of our proposed methods (VIGO, VIGOd, VIGOw) gets the lower mistake rate on at least 3 out of 4 datasets, except VIGOd versus DualSGD-Logit. Between these two methods, VIGOd runs better on big datasets (*Poker* and *Airlines*), but worse on medium datasets (*Cod-rna* and *Ijcnn1*). Regarding the time complexity, the 3 variational inference based algorithms are significantly faster than the 4 kernel-based methods. One reason for the accuracy and time efficiency of VI framework is that it does not need to maintain a set of support vectors, hence does not have to worry about losing information during budget maintenance.

7.2. Experiment for evolving data stream learning

7.2.1. Dataset

We use MOA [41]—the prevalent framework for learning streaming data to generate three data streams with concept drifts as follows. The *SEA* [44] and *Random tree* [6] data streams were created using the *Sea Generator* and *Random Tree Generator* of MOA. Both of them contain 35,000 data points, and the concept drift occurs *gradually* from 10001th to 11000th data points, that mean both concepts are present over this period of 1000 instances. The SEA data stream [44] is generated by MOA using three at-

Table 4
Mistake rate (%) and time cost (seconds) of online methods in dynamic setting.

Dataset Algorithm	SEA Mistake rate (Time)	Random tree Mistake rate (Time)	Hyperplane Mistake rate(Time)	SEA-A Mistake rate Time)	Random tree-A Mistake rate (Time)	Hyperplane-A Mistake rate (Time)	Average Mistake rate (Time)
kNN	2.91 (2.34)	32.19 (5.73)	16.44 (944.55)	3.12 (2.31)	37.82 (5.75)	19.79 (947.41)	18.71 (318.02)
SAM-kNN	**1.92** **(11.03)**	34.43 (3.00)	14.14 (1932.30)	**2.37** **(10.58)**	40.26 (2.12)	16.47 (1272.81)	18.27 (538.64)
kNN-PAW	2.69 (4.02)	31.98 (8.64)	15.89 (1574.58)	3.06 (4.11)	38.00 (8.47)	24.03 (1592.75)	19.28 (532.10)
DACC	6.62 (0.56)	37.49 (1.61)	19.48 (198.25)	6.71 (0.55)	43.98 (1.56)	19.68 (197.16)	22.33 (66.62)
HAT	2.37 (0.23)	**29.43** **(0.38)**	11.76 (176.55)	3.29 (0.19)	38.83 (0.38)	18.08 (57.11)	17.29 (39.14)
VIGO	2.99 (0.66)	32.97 (1.99)	**11.70** **(271.92)**	3.93 (0.69)	39.29 (2.22)	50.00 (609.12)	23.48 (147.77)
VIGOd	2.84 (0.58)	30.66 (0.72)	**11.70** **(159.70)**	2.81 (0.59)	**36.56** (0.71)	**15.21** **(160.30)**	**16.63** **(53.77)**
VIGOw	2.23 (0.68)	30.65 (0.82)	12.43 (175.15)	2.63 (0.61)	36.81 (0.78)	17.41 (172.40)	17.03 (58.41)

tributes (x_1,x_2, x_3), where only the first two attributes are relevant. If $x_1 + x_2 \leq \theta$, the class label is *false*, otherwise it is *true*; where the threshold θ often takes one of the value 8, 9, 7 and 9.5 as the concept for each data block. In our experiment, *SEA* contains three attributes (x_1,x_2, x_3) with 2 classes, where arbitrarily $x_1 + x_2 \leq \theta_1 = 8$ was chosen for the first (initial) concept and $x_1 + x_2 \leq \theta_2 = 7$ for the second concept (the concept change). The *Random Tree Generator* [6] is used to creates a data stream with 10 continuous attributes and three classes. A drift was created by generating a different random tree. *Hyperplane* data streams [6] were generated by *Hyperplane generator* of MOA, where a hyperplane in D-dimensional space $\sum_{i=1}^{D} w_i x_i = w_0$ slowly rotates continuously changing the linear decision boundary of the stream. Instances for which $\sum_{i=1}^{D} w_i x_i \geq w_0$ are labeled positive, and instances for which $\sum_{i=1}^{D} w_i x_i < w_0$ are labeled negative. In our experiment, *Hyperplane* data stream contains 10 million data instances, each with 5 numeral attributes with 10% noise added to make the problem more challenging.

Furthermore, we also tested our proposed algorithms' performance of tackling abrupt concept drifts which quite often happen with data streams. Following the way of creating abrupt concept drifts in [12], we created stream data with abrupt concept drifts from the 3 stream datasets mention above. After every $N = 10{,}000$ instances arrived, we abruptly exchange the concepts creating the instances. In *SEA*, we label first $N = 10{,}000$ instances using concept 1 ($x_1 + x_2 \leq \theta_1 = 8$), the next N instances using concept 2 ($x_1 + x_2 \leq \theta_2 = 7$), the following N instances using concept 1, the N instances after that using concept 2, and so on. Similarly, in *Random Tree*, the first N instances are created by tree 1, the next N instances by tree 2, and so on. For *Hyperplane*, we classify the first N instances using $\sum_{i=1}^{D} w_i x_i \geq w_0$, the next N instances using $\sum_{i=1}^{D} w_i x_i \leq w_0$, and so on. By this way we have 3 challenging data streams with abrupt concept drifts named *SEA-A, Random Tree-A*, and *Hyperplane-A*, respectively.

7.2.2. Experimental setting

We compare VIGOd/w with the recent kNN-based methods SAM–kNN [8,9], kNN–PAW and kNN [10]; a recent ensemble method DACC [11]; and a commonly used adaptive tree HAT [12]. We also run VIGO to see how VIGOd/w improve the performance of VIGO in changing environments. All the benchmark methods are implemented in MOA [41].

In the dynamic setting, each classifier has a single run through the data stream. Each instance was tested before being used for training (Interleaved Test-Then-Train or Prequential).

7.2.3. Result and discussion for dynamic setting

Table 4 presents the result of 5 benchmark algorithms (kNN, SAM–kNN, kNN–PAW, DACC, HAT) and VIGO(d/w) on the 6 streaming datasets. The best results (lowest mistake rates) are shown in bold.

For the first three datasets with gradual concept drifts, VIGO still achieves competitive performance compared with VIGOd/w. However, for challenging streams with abrupt concept drifts, VIGO is poorer than its adaptive versions. Between the two, VIGOd is slightly better with 3 wins/1 draw/2 losses over VIGOw.

Compared with adaptive tree HAT, both VIGOd and VIGOw perform better with 4 wins/2 losses each. For Random Tree stream, as expected, HAT achieves the best accuracy because Hoeffding tree algorithm utilises the same underlying structure for data classification.

VIGOd and VIGOw win DACC (with Naïve Bayes as the default base learners as in MOA) impressively on all streams. Actually, VIGO outperforms Naïve Bayes method (see the comparison of VIGO with BLAST in Section 7.1.3), and this leads to the better performance of VIGOd/w vs DACC which also uses Naïve Bayes as the base learners.

Our two proposed adaptive methods also totally win kNN (default setting as in MOA: the number of neighbours k = 10, the maximum number of instances to store = 1000). kNN–PAW (combination of kNN and Probabilistic Adaptive Windowing) is worse than VIGOw (6 losses/ 0 wins), and only wins VIGOd on 1 out of 6 data streams (SEA).

Finally, VIGOd is better than the latest kNN-based adaptive method SAM–kNN (4 wins/ 2 losses), whilst VIGOw is as good as SAM–kNN (3 wins/ 3 losses). However, over the 6 data streams, VIGOw has a smaller average error rate (17.03%) than SAM–kNN (18.27%).

Again, we can see that VI-based framework is more competitive than kNN-based framework: in the static setting, VIGO wins BLAST which uses kNN as one of its base learners, and in the dynamic setting, VIGOd/w wins kNN, kNN–PAW and SAM–kNN.

Regarding the execution time (see Fig. 6), SAM-kNN and kNN-PAW are the most time-consuming methods, the next is kNN. DACC requires nearly triple the time compared with HAT. The runtime of VIGOd/w is almost double that of HAT, but is still moderate compared with the other benchmark methods.

8. Conclusion

We have presented a new Bayesian online learning method VIGO and its 2 adaptive versions (VIGOd and VIGOw) which

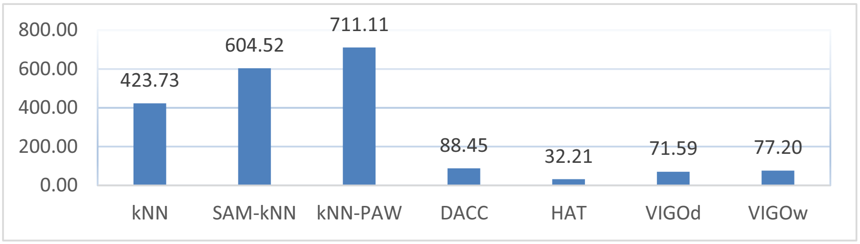

Fig. 6. Average runtime (seconds) of VIGOw/d and benchmark algorithms in dynamic setting.

achieve superior performance over many recent or well-known online learning methods. In particular, VIGOd and VIGOw are two fast adaptive methods that run well in both stationary and changing environments. Unlike many online learning algorithms, our algorithms do not require the storage of arriving instances as they are discarded immediately after used. Furthermore, in the framework of our proposed methods, models for different classes are trained independently, that make them naturally parallel. Undoubtedly, a parallel implementation will further decrease the run time of our proposed methods. Another strong point of our methods is the generative nature of our models, which give access to the full data distribution and class distribution at any time step. This can be used in many advanced online learning tasks, for example, in imbalanced learning to calculate the expected loss of cost-sensitive model, in online active learning to measure the ambiguity of an instance to decide if a query should be made, or in semi-supervised learning, where an unlabelled instance can be used to update models of different classes with different weights based on the confidence (inferred from the information about the posterior probability $p(y|\mathbf{x})$). This will be our future work.

Acknowledgements

Thi Thu Thuy Nguyen is supported by the Australian Government Research Training Program Scholarship to undertake this research.

Supplementary materials

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.patcog.2018.04.007.

References

[1] T. Le, T.D. Nguyen, V. Nguyen, D. Phung, Dual space gradient descent for online learning, NIPS, 2016.
[2] J. Lu, S.C.H. Hoi, J. Wang, P. Zhao, Z.-Y. Liu, Large scale online kernel learning, J. Mach. Learn. Res. 17 (2016) 1–43.
[3] J.N. van Rijn, G. Holmes, B. Pfahringer, J. Vanschoren, Having a Blast: meta-learning and heterogeneous ensembles for data streams, in: ICDM, 2015, pp. 1003–1008.
[4] K. Crammer, A. Kulesza, M. Dredze, Adaptive regularization of weight vectors, Mach. Learn. 91 (2013) 155–187.
[5] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, Y. Singer, Online passive aggressive algorithms, J. Mach. Learn. Res. 7 (2006) 551–585.
[6] P. Domingos, G. Hulten, Mining high-speed data streams, in: KDD, 2000, pp. 71–80.
[7] T.T. Nguyen, T.T.T. Nguyen, X.C. Pham, A.W.C. Liew, J.C. Bezdek, An ensemble-based online learning algorithm for streaming data, 2017. https://arxiv.org.
[8] V. Losing, B. Hammer, Self-Adjusting Memory: how to deal with diverse drift types, IJCAI, 2017.
[9] V. Losing, B. Hammer, H. Wersing, KNN classifier with Self Adjusting Memory for heterogeneous concept drift, in: ICDM, 2016, pp. 291–300.
[10] A. Bifet, B. Pfahringer, J. Read, G. Holmes, Efficient data stream classification via probabilistic adaptive windows, SAC, 2013.
[11] G. Jaber, A. Cornuéjols, P. Tarroux, Online learning: searching for the best forgetting strategy under concept drift, in: ICONIP, 2013, pp. 400–408.
[12] A. Bifet, R. Galvalda, Adaptive learning from evolving data streams, IDA, 2009.
[13] F. Rosenblatt, The perceptron: a probabilistic model for information storage and organization in the brain, Psychol. Rev. 65 (6) (1958) 386–408.
[14] J. Zinkevich, Online convex programming and generalized infinitesimal gradient ascent, in: ICML, 2003, pp. 928–936.
[15] N. Cesa-Bianchi, A. Conconi, C. Gentile, A second-order perceptron algorithm, SIAM J. Comput. 34 (2005) 640–668.
[16] J. Wang, P. Zhao, S.C.H. Hoi, Exact soft confidence-weighted learning, in: ICML, 2012, pp. 107–114.
[17] Z. Wang, K. Crammer, S. Vucetic, Breaking the curse of kernelization: budgeted stochastic gradient descent for large-scale SVM training, J. Mach. Learn. Res. 13 (2012) 3103–3131.
[18] X.C. Pham, M.T. Dang, S.V. Dinh, S. Hoang, T.T. Nguyen, A.W.C. Liew, An online machine learning method based on random projection and Hoeffding tree classifier, DICTA, 2017.
[19] C.M. Bishop, Pattern Recognition And Machine Learning, Springer, 2006.
[20] A.A. Farag, A. El-Baz, G. Gimel'farb, Density estimation using modified expectation-maximization algorithm for a linear combination of Gaussians, ICIP, 2004.
[21] A. El-Baz, G. Gimel'farb, EM based approximation of empirical distributions with linear combinations of discrete Gaussians, ICIP, 2007.
[22] M. Hoffman, D.M. Blei, F. Bach, Online learning for latent Dirichlet allocation, NIPS, 2010.
[23] M. Hoffman, D.M. Blei, J. Paisley, C. Wang, Stochastic variational inference, J. Mach. Learn. Res. 14 (2013) 1303–1347.
[24] C. Wang, J. Paisley, D.M. Blei, Online variational inference for the hierarchical Dirichlet process, Artif. Intell. Stat. (2011).
[25] T. Broderick, N. Boyd, A. Wibisono, A.C. Wilson, M.I. Jordan, Streaming variational Bayes, NIPS, 2013.
[26] T.T.T. Nguyen, T.T. Nguyen, X.C. Pham, A.W.-C. Liew, Y. Hu, T. Liang, C.-T. Li, A novel online Bayes classifier, DICTA, 2016.
[27] Q. Bai, H. Lam, S. Sclaroff, A Bayesian framework for online classifier ensemble, ICML, 2014.
[28] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, A. Bouchachia, A survey on concept drift adaptation, ACM Comput. Surv. 46 (4) (2014) 1–37.
[29] R. Klinkenberg, T. Joachims, Detecting concept drift with support vector machines, in: ICML, 2000, pp. 487–494.
[30] A. Bifet, R. Gavaldà, Learning from time-changing data with adaptive windowing, SIAM International Conference on Data Mining, 2007.
[31] E. Cohen, M. Strauss, Maintaining time-decaying stream aggregates, ACM Symposium on Principles of Database Systems, 2003.
[32] G.J. Ross, N.M. Adams, D.K. Tasoulis, D.J. Hand, Exponentially weighted moving average charts for detecting concept drift, Pattern Recognit. Lett. 33 (2) (2012) 191–198.
[33] I. Frias-Blanco, J. del Campo-Avila, G. Ramos-Jimenez, R. Morales-Bueno, A. Ortiz-Diaz, Y. Caballero-Mota, Online and non-parametric drift detection methods based on Hoeffding's bound, IEEE Trans. Knowl. Data Eng. 27 (3) (2015) 810–823.
[34] M. Tennant, F. Stahl, G. Di Fatta, J.B. Gomes, Towards a parallel computationally efficient approach to scaling up data stream classification, in: Research and Development in Intelligent Systems XXXI, Springer International Publishing, 2014, pp. 51–65.
[35] J. Read, A. Bifet, B. Pfahringer, G. Holmes, Batch-incremental versus instance-incremental learning in dynamic and evolving data, IDA, 2012.
[36] A. Shaker, E. Hüllermeier, Instance-based classification and regression on data streams, in: Learning in Non-Stationary Environments, Springer, New York, 2012, pp. 185–201.
[37] T.T. Nguyen, T.T.T. Nguyen, X.C. Pham, A.W.C. Liew, A novel combining classifier method based on variational inference, Pattern Recognit. 49 (2016) 198–212.
[38] C.-C. Chang, C.-J. Lin, LIBSVM: a library for support vector machines, ACM Trans. Intell. Syst. Technol. 2 (3) (2011). Software available athttp://www.csie.ntu.edu.tw/~cjlin/libsvm.
[39] M. Lichman, UCI Machine Learning Repository, University of California, School of Information and Computer Science, Irvine, CA, 2013 http://archive.ics.uci.edu/ml.
[40] J. Hensman, N. Fusi, N.D Lawrence, Gaussian processes for big data, in: UAI, 2013, pp. 282–290.

[41] A. Bifet, G. Holmes, B. Pfahringer, P. Kranen, H. Kremer, T. Jansen, T. Seidl, MOA: massive online analysis, a framework for stream classification and clustering,, in: Journal of Machine Learning Research Workshop and Conference Proceedings, Vol.11: Workshop on Applications of Pattern Analysis, 2010. (the latest release of June 2017). URL http://moa.cms.waikato.ac.nz.
[42] S.C.H. Hoi, J. Wang, P. Zhao, LIBOL: a library for online learning algorithms, J. Mach. Learn. Res. 15 (2014) 495–499.
[43] J. Demsar, Statistical comparisons of classifiers over multiple datasets, J. Mach. Learn. Res. 7 (2006) 1–30.
[44] W. Street, Y. Kim, A streaming ensemble algorithm (SEA) for large-scale classification, in: KDD, 2001, pp. 377–382.

Thi Thu Thuy Nguyen is currently a Ph.D. student at the School of Information & Communication Technology, Griffith University, Australia. She graduated from the Faculty of Mathematics, Voronezh State University, Russia in 2008. Her research interest is in the field of applied mathematics, machine learning, pattern recognition.

Tien Thanh Nguyen received PhD degree in computer science from the School of Information & Communication Technology, Griffith University, Australia in 2017. He is currently a Lecturer in the School of Applied Mathematics and Informatics, Hanoi University of Science and Technology, Vietnam. His research interest is in the field of machine learning, pattern recognition, and evolutionary computation. He is a member of the IEEE since 2014.

Alan Wee-Chung Liew is currently an Associate Professor at the School of Information & Communication Technology, Griffith University, Australia. His research interest is in the field of medical imaging, bioinformatics, computer vision, pattern recognition, and machine learning. He serves on the technical program committee of many international conferences and is on the editorial board of several journals, including the IEEE Transactions on fuzzy systems. He is a senior member of the IEEE since 2005.

Shi-Lin Wang received his B.Eng. degree in Electrical and Electronic Engineering from Shanghai Jiaotong University, Shanghai, China in 2001, and his Ph.D. degree in the Department of Computer Engineering and Information Technology, City University of Hong Kong in 2004. Since 2004, he has been with the School of Information Security Engineering, Shanghai Jiaotong University, where he is currently an Associate Professor. His research interests include image processing and pattern recognition. Dr. Wang is a senior member of the Institute of Electrical and Electronic Engineers (IEEE) and his biography is listed in Marquis Who's Who in Science and Engineering.