Computational Intelligence

AN INTERNATIONAL JOURNAL

Computational Intelligence, Volume 33, Number 2, 2017

MINING EVOLVING DATA STREAMS WITH PARTICLE FILTERS

RICKY FOK,[1] AIJUN AN,[1] AND XIAOGANG WANG[2]

[1]*Department of Electrical Engineering and Computer Science, York University, Toronto, Ontario, Canada*
[2]*Department of Mathematics and Statistics, York University, Toronto, Ontario, Canada*

We propose a particle filter-based learning method, PF-LR, for learning logistic regression models from evolving data streams. The method inherently handles concept drifts in a data stream and is able to learn an ensemble of logistic regression models with particle filtering. A key feature of PF-LR is that in its resampling, step particles are sampled from the ones that maximize the classification accuracy on the current data batch. Our experiments show that PF-LR gives good performance, even with relatively small batch sizes. It reacts to concept drifts quicker than conventional particle filters while being robust to noise. In addition, PF-LR learns more accurate models and is more computationally efficient than the gradient descent method for learning logistic regression models. Furthermore, we evaluate PF-LR on both synthetic and real data sets and find that PF-LR outperforms some other state-of-the-art streaming mining algorithms on most of the data sets tested.

Received 14 August 2014; Revised 12 December 2014; Accepted 25 May 2015

Key words: concept drift, ensemble methods, high dimensional data stream mining.

1. INTRODUCTION

A challenge in learning from data streams is that the underlying model generating the data evolves over time with unknown dynamics, a phenomenon known as concept drift. If left untreated, learned models become outdated, leading to classification errors. To keep models up-to-date, methods of handling drifts must be imposed. In this article, we propose a method to handle concept drifts by performing sequential Bayesian inference on streaming data. We model concept drifts as the change in the hidden (or state) variables of a Hidden Markov Model (Breiman and Petrie 1966) and employ particle filtering (Doucet and Johansen 2011), a sequential Monte Carlo (SMC) method, to discover the most up-to-date concepts. Particle filtering can be considered as an ensemble-learning method that generates a fixed number of classifiers (called particles) at each time step from the most important classifiers in the previous step. Particles at time n are generated from a proposal distribution conditioned on the values of previous hidden variables. This proposal distribution is usually assumed to be a commonly used Gaussian distribution. The likelihood values of the generated particles on the most recent data instance (i.e., the observation at time n) are used to compute the weights of the particles. Subsequent particles are generated near the ones with the highest weights. At any time point, an ensemble of particles (i.e., classifiers) can be selected for use in classification that can be carried out using weighted majority voting from the ensemble. Although a particle filter can naturally handle data streams, its predictive performance, especially on learning from noisy and concept-drifting data streams, has much room to improve.

We improve the classification performance with particle filtering in the following aspects. First, we propose a modification to the particle filter where the quality of a particle is measured by the TA instead of the likelihood function used in the conventional particle filter. We argue that the TA better approximates the predictive accuracy (as it approaches the same asymptotic limit as the predictive accuracy) than the likelihood function, and modify

Address correspondence to Ricky Fok, Department of Computer Science and Engineering, York University, Toronto, Ontario, Canada M3J 1P3; e-mail: ricky.fok3@gmail.com

the resampling step of particle filtering to resample from only the particles that maximize the TA. Second, instead of processing training instances one by one (i.e., generating a new set of particles based on each instance) as in conventional particle filter methods, we process the input data stream batch by batch where each batch contains a small set of training instances. Compared with updating particles based on single instances, batch-based learning is more resistant to noise, reducing the chance of producing models that overfit to random noise. Batch-based learning can better distinguish between noise and a concept drift because a concept drift is more persistent in a batch than random noise. Third, as mentioned earlier, a particle filter produces an ensemble of classifiers, each corresponding to a particle. In conventional methods, an ensemble of fixed size is used in classification (e.g., the top 20 particles are used in classification with a majority voting scheme). We propose to use the particles with the maximum TA to classify new instances. Because the number of such particles can vary batch by batch, the size of the ensemble used in our classification procedure varies from time to time. Our experiments show that such an ensemble adapts to a concept drift more quickly than a fixed sized ensemble where the size is specified by the user. This is because of the fact that a smaller number of particles achieve the highest TA immediately after a drift and that such particles can give higher predictive performance in the data of the new concept, while an ensemble of fixed size would contain particles that may still model the old concept. Fourth, to ensure that the current ensemble is at least as good as the one in the last batch (especially when there is no drift between the two batches), the particle mean over the ensemble for the last batch, chosen as a representation of the ensemble, is also considered for membership in the current ensemble. If the particle mean also gives the best TA in the current batch, it will be included in the current ensemble and takes part in voting for the prediction. Otherwise, it is simply discarded.

We apply the modified particle filter to learn an ensemble of logistic regression models from streaming data. The resulting algorithm, logistic regression with particle filtering (PF-LR), is evaluated thoroughly in our experiments. We find that PF-LR is robust to noise and quick to react to concept drifts at the same time. It has good learning performance even with relatively small batches. In the evaluation, we use both synthetic and real data sets, and compare PF-LR to a number of other algorithms, including the particle filter that uses likelihood for particle selection, the auxiliary particle filter (APF) (Pitt and Shephard 1999), a regularized version of APF (RegAPF), logistic regression with gradient descent, and three state-of-the-art data streaming mining algorithms (Naive Bayes with Dynamic Weighted Majority (DWM-NB) (Kolter and Maloof 2007), Hoeffding Tree with Leveraging Bagging (LB-HT) (Bifet et al. 2010a)) and the Vowpal Wabbit system (VW) (Langford et al. 2007), which is a fast implementation of stochastic gradient descent. We also compare PF-LR with batch-based support vector machine and k-nearest neighbors. We show that PF-LR outperforms these methods in terms of predictive accuracy and speed to react to concept drifts on streaming data. Furthermore, even in the case of a very imbalanced class distribution as in the CIRCLES (Nishida and Yamauchi 2007) data set where the signal-to-noise ratio of the minority class can be very low, PF-LR still performs very well.

Particle filters are known to suffer from accuracy drop when learning from data with high dimensionality, such as in the case of the HYPERPLANE data set (Hulten et al. 2001). To handle high-dimensional data, we implemented a naive dimensional reduction (DR) mechanism with drift detection based on accuracy drop. When the data set has high dimensionality, the DR procedure is triggered by a TA drop below a specified threshold. We evaluate PF-LR with this dimension reduction procedure on the HYPERPLANE data set, and find that PF-LR gives comparable performance with DWM-NB at a moderately high number of dimensions, $D \simeq 30$, with a sufficiently high batch size of 100 instances, while PF-LR and DWM-NB outperforms LB-HT and VW on all numbers of dimensions tested.

This shows the potential of PF-LR with a more efficient DR mechanism to tackle concept drifts in high dimensions.

The contributions of this article are summarized as follows:

- We develop an algorithm based on particle filtering and conduct extensive experiments with commonly used synthetic and real world benchmark data. We show that PF-LR outperforms state-of-the-art stream mining algorithms on streaming data, that is, Hoeffding Tree with Leveraging Bagging (Bifet et al. 2010a), DWB-NM (Kolter and Maloof 2007), the VW system (Langford et al. 2007), batch-based support vector machine and k-nearest neighbor, with higher accuracy and quicker reaction time to drifts. We also compare PF-LR with the well-known APF (Pitt and Shephard 1999) and its regularized version, and show that PF-LR outperforms these algorithms on the tested data sets.
- We propose to use batch-based learning in particle filtering and use a variable-sized ensemble for prediction. The ensemble used for prediction contains the particles resampled from the ones with the maximum TA. We show that the modified particle filter is robust to noise when used to learn regression coefficients with quick reaction time to concept drifts.
- We test PF-LR on high-dimensional data with a simple DR mechanism to show that PF-LR has the potential in obtaining good predictive performance when coupled with a reasonable DR mechanism. It implies that better performance could be obtained with a more refined DR method (which is not the focus of this article).
- We test various quality measures and selection schemes. We find that PF-LR performs the best with TA as a quality measure and choosing only the best performing classifiers on each batch.
- This is the first attempt that uses TA as a criterion for particle selection and resampling. We show that this results in little overfitting for logistic regression and that it gives higher predictive accuracy than using the likelihood function as carried out in conventional particle filters. We also show that our method performs better than gradient descent for learning logistic regression models.

The organization of this paper is as follows. In the next section, we review related work in the literature. Section 3 first gives a justification for particle filtering to be a strong candidate to handle concept drifts. It then presents the features and details of our proposed algorithm, PF-LR. In Section 4, we present a procedure for DR, which will be used in our experiments for testing the performance of PF-LR on high-dimensional data sets. The experimental evaluation in which we compare PF-LR with other algorithms is given in Section 5 where we illustrate the ability of PF-LR to handle concept-drifting and noisy data streams. The discussion and conclusions are given in Section 6.

2. RELATED WORK

Mining streaming data involves making inferences on time-varying data. More often than not, the underlying attributes in relation to the observed data also change (Aggarwal 2007), a phenomenon known as concept drift. Particle filtering is an SMC method used to estimate time series of hidden variables with the most recent data. It has been used in mathematical finance (Hedibert and Tsay 2011), in tracking and navigation (Gustafsson et al. 2002). However, there have been very few (if any) attempts to apply particle filtering to handle drifts. As an application to classification tasks, particle filtering is an ensemble method that tracks the movement of decision boundaries estimated by the regression

coefficients (i.e., the classifiers). Issues related to concept drifts are extensively studied in the literature. Recent reviews include Kadlec et al. (2011), Alberg et al. (2012), Moreno–Torres et al. (2012), and Gama et al. (2014). In this section, we survey previous work on ensemble methods, existing methods to handle drifts, and particle filtering.

2.1. Ensemble Learning

Ensemble methods (for a review, see Rokach 2010) combine multiple classifiers to make an overall prediction. Various methods differ in their mechanisms of combining predictions, and in the ways that different classifiers are learned.

Weighted majority (Littlestone and Warmuth 1994) makes the overall prediction by the weighted average over the ensemble of classifiers. For an ensemble with M classifiers, each with weight w_i, the overall prediction $\bar{p}$ is

$$\bar{p} = \frac{\sum_i^M w_i p_i}{\sum_i^M w_i},$$

where p_i is the prediction given by the ith classifier. The weight of each classifier is proportional to its performance. For example, Littlestone and Warmuth (1994) starts with $w_i = 1$ for each classifier and is reduced by half whenever the classifier predicts incorrectly.

Stacking (Wolpert 1992) is a classifier combination method where the overall prediction is given by a meta-classifier trained on the outputs of classifiers in the ensemble. The purpose of the meta-classifier is to learn the performances of the classifiers and making adjustments if necessary. The performances of the classifiers are evaluated by cross validation. The outputs of these classifiers (and the class labels of the training examples for training the meta-classifier) are passed onto the meta-classifier. The overall output is then given by the meta-classifier.

The bagging method (Breiman 1996) generates M data sets from the training data by sampling with replacement (bootstrapping). The M generated data sets have the same size as the training data from which an ensemble of M classifiers are obtained. Then majority voting is used to obtain an overall prediction.

Boosting (Freund and Schapire 1996) is a method to improve the performance of a "weak" classifier, whose predictive accuracy can be as low as just better than random guessing. A well-known algorithm that employs boosting is Adaboost (Freund and Schapire 1999). In Adaboost, classifiers are trained focusing on the subset of data being misclassified by the previous classifier, with its weight given by its performance. This process iterates until a specified number of classifiers are trained. The final prediction is given by the weighted majority and is expected to be more accurate because of the diversity of the individual classifiers. A similar method is Arcing (Breiman 1998), which uses another weighting procedure and decisions are made with majority voting.

2.2. Learning with Concept Drifts

Concept drift (Schlimmer and Granger 1986a; Maloof 2005) is a phenomenon in a data stream where the underlying model that generates the data changes over time. Special attention to cope with changing concepts is necessary as they deteriorate classification accuracy if left untreated (Schlimmer and Granger 1986b). To handle drifts, algorithms must contain mechanisms to forget past examples when drifts occur and learn the most recent concepts. Common techniques employed include (1) using ensemble methods, (2) forgetting past instances by weighting, and (3) using sliding windows. Early examples include STAGGER (Schlimmer and Granger 1986b) and the FLORA systems (Widmer and Kubat 1996). More

recently, Streaming Ensemble Algorithm (SEA) (Street and Kim 2001), Concept-adapting Very Fast Decision Tree (CVFDT) (Hulten et al. 2001), using ensemble methods (Wang et al. 2003), and DWM (Kolter and Maloof 2007), a method using χ^2-test along with rule induction (Sotoudeh and An 2010). Furthermore, an ideal algorithm to cope with concept drifts should have the following features (Tsymbal 2004): fast adaptation to drift, robustness to noise and recognizing recurring concepts.

STAGGER (Schlimmer and Granger 1986b) maintains a set of concept descriptions with corresponding weights. The weights are adjusted when new examples arrive. Also, the weights decay over time in order to cope with drifts. The FLORA systems (Widmer and Kubat 1996) keeps a window of currently reliable examples and hypotheses. The size of the window is adjustable based on heuristics. When an old concept reappears, FLORA is able to use previously stored instances for learning. SEA (Street and Kim 2001) is an ensemble learning algorithm. Rather than generating multiple same-sized data sets with bagging or boosting, SEA builds an ensemble of classifiers by learning from sequential chunks to satisfy the one-pass constraint on stream mining. DWM (Kolter and Maloof 2007) is an adaptive weighting scheme. It maintains a set of classifiers and updates their weights according to the performance of each classifier. New classifiers are created, and outdated ones are removed based on the global performance. Predictions are combined using weighted majority. Wang et al. (2003) proposes an approach for handling drifts with ensemble methods. There, different classifiers are trained on chunks of data. Each of them is weighted by their classification performance. The predictions are combined by weighted majority voting. CVFDT (Hulten et al. 2001) is an extension of an earlier VFDT (Domingos and Hulten 2000) with a drift handling mechanism. VFDT is a decision tree algorithm using the Hoeffding bound (Hoeffding 1963). CVFDT adapts to changing concepts by replacing out-of-date subtrees based on their classification accuracy. Sotoudeh and An (2010) proposed a method to detect drifts using the χ^2-test along with rule-based classification. Their detection method is sensitive to partial drifts where drifts occur only in a subspace of the feature space. When a drift is detected, rule quality measures are used to judge the relevance of old rules and old instances. Finally, the VW (Langford et al. 2007) is a system focused on the learning speed. It consists of optimization methods that employ online gradient descent with adaptive learning rates.

Recent variants based on the methodologies outlined earlier have been proposed. Drift detection methods include work carried out by Gomes et al. (2011), Ang et al. (2012), and Bouchachia (2011a). Ikonomovska et al. (2011) proposed a variant of the Hoeffding tree; Adae and Berthold (2013), Yao et al. (2012), Zhao et al. (2011), and Gama et al. (2013) employed sliding windows. Bouchachia (2011b) proposed an algorithm that removes and adds classifiers for drift adaptation. Minku and Yao (2011) proposed another ensemble algorithm. Gama and Kosina (2011) and Z'liobaite et al. (2012) studied situations where drifts are expected to happen.

2.3. Particle Filtering

Particle filtering (Doucet and Johansen 2011), or SMC, has been extensively used in the inference of hidden Markov models (Breiman and Petrie 1966), where time series data are generated by a hidden Markov chain. Importance sampling (see, e.g., Geweke 1989) is used with the most up-to-date data. In every time step, the draws from importance sampling, called particles, are used to estimate the most recent probability distribution of hidden variables (also called state variables), given the current prior distributions of the hidden variables conditioned on their previous values. Resampling on the estimated distribution is often performed to move the particles to regions with high probability densities so that improbable

particles are discarded. An alternative method to resampling is to simply look ahead of time to propagate particles with the best expected future weight. This is the basis of the auxiliary particle filter (Pitt and Shephard 1999).

Regularized particle filters summarized in Casarin and Marin (2009) employ the same regularization procedures as in Liu and West (2001) and Musso et al. (2001), where the prior distribution of current conditional distributions of hidden variables is generated by a set of hyperparameters. It is shown there that the regularized auxiliary particle filter outperforms other regularized particle filters with a large particle number.

Particle filters can be optimized to improve their efficiency. The most natural approach is performed with particle swarm optimization (PSO) (Kennedy and Eberhart 1995). PSO is a stochastic optimization method where at each time step, the particles are moved to maximize an objective function based on a random linear combination of the best positions visited by each particle, and the best known global position. The resulting particle filter, PSO-PF (Klamargias et al. 2008) (also see Ji et. al. (2008)) was shown to be more efficient than the particle filter, especially in the presence of noise. Finally, Grest and Krueger (2007) combined gradient descent optimization with particle filtering. Finally, the concept of particle filtering has been applied to static problems. For instance, Moral et al. (2006) developed a framework for such purpose, which they call Monte Carlo samplers. Chopin (2002) applied batch processing in particle filtering for static models. However, it is unclear how batch processing would affect the performance of particle filters in the presence of drifts.

Because particle filtering is designed to estimate the hidden attributes of time series data, it would be a useful tool to mine data streams. Particle filtering can be thought as an ensemble method, where new classifiers are generated from the ones with high performances, while low performing ones are discarded. However, existing particle filters weight particles in proportion to their likelihoods. In noisy environments, these methods tend to overfit. Our proposed algorithm employs particle filtering with the TA of each particle as its weight. Majority voting is used to estimate the best classifier for the most current concept. This is repeated for each time step, replacing the previously learned classifiers when better ones are generated. At the same time, replacing out-of-date classifiers as drifts occur. We found that this procedure leads to very little overfitting.

3. PARTICLE FILTERING FOR CLASSIFICATION

In this section, we first provide details of the conventional particle filtering process and show that particle filtering can naturally adapt to concept drifts. We then present the features and algorithmic details of our proposed PF-LR method. Last, we discuss how TA as a quality measure would better approximate the predictive accuracy compared with using the likelihood as the quality measure.

3.1. The Drift Adaptability of Particle Filters

Particle filtering is a method of sequential Bayesian analysis with SMC on hidden Markov models (HMM). It has been shown to be effective in modeling dynamical models (Doucet and Johansen 2011). In particular, for a time series of model parameters (i.e., the hidden states of the HMM) $\boldsymbol{\beta}^{(1:T)} := \{\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(T)}\}$, particle filtering can be used to estimate the posterior distribution of the model parameters $p(\boldsymbol{\beta}^{(1:T)}|\mathbf{x}^{(1:T)})$ by inferencing on observed time series data $\mathbf{x}^{(1:T)}$. Two assumptions of HMM are, first, that the model parameters evolve in such a way that it is only dependent on the most recent value. In other words, $p(\boldsymbol{\beta}^{(n)}|\boldsymbol{\beta}^{(1:n-1)}) = p(\boldsymbol{\beta}^{(n)}|\boldsymbol{\beta}^{(n-1)})$. Second, the observed data at time n are

generated solely by $\boldsymbol{\beta}^{(n)}$, the parameters at the corresponding time. That is, $p(\mathbf{x}^{(n)}|\boldsymbol{\beta}^{(1:n)}) = p(\mathbf{x}^{(n)}|\boldsymbol{\beta}^{(n)})$. By the repeated use of Bayes's theorem, the posterior distribution of the model parameters can be written as

$$p\left(\boldsymbol{\beta}^{(1:T)}|\mathbf{x}^{(1:T)}\right) = p_1\left(\boldsymbol{\beta}^{(1)}|\mathbf{x}^{(1)}\right)\prod_{n=2}^{T} p_n\left(\boldsymbol{\beta}^{(n)}|\mathbf{x}^{(n)}, \boldsymbol{\beta}^{(n-1)}\right). \tag{1}$$

The series of product on the right-hand side is highly suggestive that the inference can be carried out sequentially. In fact, SMC estimates the conditional probability at each time with importance sampling by generating model parameters from a chosen proposal function and calculating their weights. We write the proposal function as

$$\left(\boldsymbol{\beta}^{(1:T)}\right) = q_1\left(\boldsymbol{\beta}^{(1)}\right)\prod_{n=2}^{T} q_n\left(\boldsymbol{\beta}^{(n)}|\boldsymbol{\beta}^{(n-1)}\right). \tag{2}$$

Dividing equation (1) by equation (2) gives the importance weights

$$w^{(1:T)} = w^{(1)}\prod_{n=2}^{T} w^{(n|n-1)},$$

where the incremental weight is

$$w^{(n|n-1)} = \frac{p_n\left(\boldsymbol{\beta}^{(n)}|\mathbf{x}^{(n)}, \boldsymbol{\beta}^{(n-1)}\right)}{q_n\left(\boldsymbol{\beta}^{(n)}|\boldsymbol{\beta}^{(n-1)}\right)}.$$

Now, suppose at time n, we have an approximation of the posterior through time 1 and time $n-1$ given by M particles and their weights $\{\boldsymbol{\beta}_i^{(1:n-1)}, w_i^{(1:n-1)}\}$, where $i = \{1, \ldots, M\}$ and observed data $\mathbf{x}^{(n)}$, sequential importance sampling proceeds as follows:

(1) Generate M particles of $\boldsymbol{\beta}_i^{(n)}$ from $q_n(\cdot|\boldsymbol{\beta}^{(n-1)})$, where $i = \{1, \ldots, M\}$ is the particle index.
(2) Calculate the M incremental weights $w_i^{(n|n-1)}$ for each particle.
(3) Calculate the weights $w_i^{(1:n)} = w_i^{(1:n-1)} \times w_i^{(n|n-1)}$.

Then, we have updated the posterior estimation from the one at time $n-1$ to time n, $p(\boldsymbol{\beta}^{(1:n)}|\mathbf{x}^{(1:n)})$, represented by the tuple $\{\boldsymbol{\beta}^{(1:n)}, w_i^{(1:n)}\}$. In practice, the calculation of the incremental weights is usually simplified by writing the conditional posterior as

$$p_n\left(\boldsymbol{\beta}^{(n)}|\mathbf{x}^{(n)}, \boldsymbol{\beta}^{(n-1)}\right) = g_n\left(\mathbf{x}^{(n)}|\boldsymbol{\beta}^{(n)}\right) f_n\left(\boldsymbol{\beta}^{(n)}|\boldsymbol{\beta}^{(n-1)}\right)$$

and setting the proposal function to be $q_n(\boldsymbol{\beta}^{(n)}|\boldsymbol{\beta}^{(n-1)}) = f_n(\boldsymbol{\beta}^{(n)}|\boldsymbol{\beta}^{(n-1)})$. Then the incremental weight for each particle is just the likelihood, $w_i^{n|n-1} = g_n(\mathbf{x}^{(n)}|\boldsymbol{\beta}^{(n)})$. As the final step, the particles are resampled M times with probabilities given by their normalized weights. This is to remove particles with low weights while duplicating the ones closer to the posterior maximum so as to reduce the particle variance at subsequent times. When the

variance of the weights is large, resampling often removes all particles other than the one with the largest weight. Pitt and Shephard (1999) proposed the auxiliary particle filter. In essence, it employs a procedure to replace resampling without the aforementioned deficit by generating an auxiliary variable for each particle—an index denoting the parents of the particles in the current time step. The auxiliary variables are generated with a distribution proportional to the weight of each particle, and therefore, improbable particles are less likely to contribute.

In essence, the particle filter tracks the movement of model parameters as data arrive. To understand that this is important for classification tasks, consider the following example. Suppose one chooses a regression model for classification. When a drift occurs, the change in the model parameters corresponds to the movement of the decision boundaries over time. Therefore, particle filters could be used to handle drifts by tracking the movement of decision boundaries.

As an illustration, a typical result of the particle filter is shown in Figure 1. Suppose that the evolution of the parameters $\boldsymbol{\beta}^{(n)}$ and $\mathbf{x}^{(n)}$ is according to two-dimensional Gaussian distributions

$$\begin{aligned} \boldsymbol{\beta}^{(n)} &\sim \mathcal{N}\left(\boldsymbol{\beta}^{(n-1)}, \mathbf{I}\right) \\ \mathbf{x}^{(n)} &\sim \mathcal{N}\left(\boldsymbol{\beta}^{(n)}, \mathbf{I}\right). \end{aligned} \tag{3}$$

We imposed two types of drifts in the data set—ones that are small and gradual throughout in accordance with equation (3), as well as a sudden jump at $t = 50$. In either case, the particle filter is able to track the movement of the model parameters. This provides a motivation that particle filtering can be used in classification tasks to handle drifts, and it is natural to construct a drift tolerant algorithm with the particle filter. Of course, concept drifts can be expected to occur almost all the time outside of experimental conditions, because the underlying mechanisms that generate such data are often complex, dynamical, and even unknown.

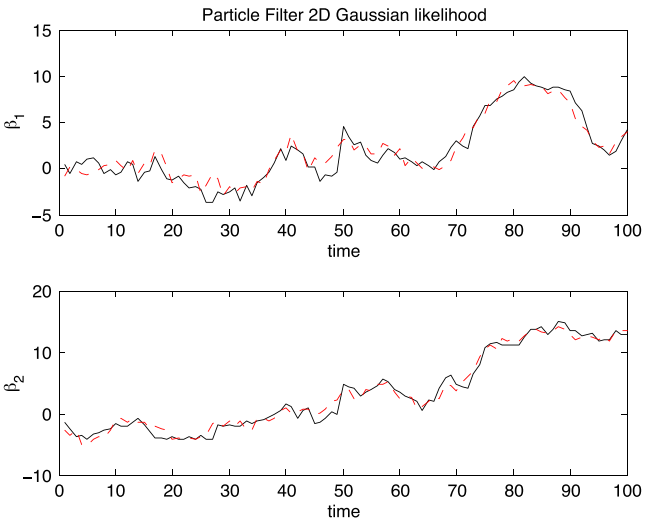

FIGURE 1. A typical result from particle filtering. The model parameters are denoted by β_1 and β_2 on the vertical axis. The true model parameters are denoted by the black solid line, whereas the estimated posterior mean is in red dashed. [Color figure can be viewed at wileyonlinelibrary.com]

The tracking property of particle filters suggests that it would be an excellent candidate in mining such data. In the upcoming subsection, we discuss the essential features of the particle filter algorithm we propose in this article for classification learning.

3.2. Features of Particle Filter-Based Learning Method

We present the essential features possessed by the proposed particle filter algorithm (PF-LR) for learning classification models from data streams before turning to the details of the algorithm. For the purpose of constructing a fast and accurate algorithm for learning logistic regression models, our algorithm possesses the following properties: (1) choosing the model parameters that gives the maximum TA, (2) batch processing, (3) discarding all previously processed data, (4) no drift detection when the batch size is small, and (5) a statistically monotonic increasing predictive accuracy in static situations. Each of the features is discussed in the following.

In practice, it is important to realize that maximizing the likelihood does not imply maximum predictive accuracy. Traditionally, a resampling or an auxiliary variable sampling procedure is applied to suppress the exponential error from SMC using weights proportional to the likelihood function. However, during our investigation, we found that using the likelihood as the weights gives rather poor results. Instead of weighting by the likelihood, we choose the set of parameters that results in the highest classification accuracy during training. We found that this leads to less overfitting for logistic regression. In Section 3.4, we argue that using TA is expected to give a higher performance than using the likelihood function.

Because PF-LR performs regression, batch processing is a natural choice. To satisfy the one-pass constraint for data stream mining (Aggarwal 2007), we opt for a batch-by-batch processing rather than using sliding windows, so each training example is processed only once to reduce the running time. Further, using batch processing is more noise-resistant than example-by-example processing. Handling data example-by-example cannot distinguish between noise and drifts. In the case of high dimensionality, large batches are usually required for accuracy. Under such circumstances, the batches usually contain numerous drifts leading to the deterioration of the classification performance. In such cases, a drift detection mechanism with DR techniques can improve the performance.

In order to obtain the most up-to-date and accurate estimates of the regression parameters, we discard all previously processed data and use only the current batch to estimate the model parameters for the current time. This is to make the algorithm faster to react to concept drifts and at the same time, save memory space in the stream-mining environment.

The particle filter is very adaptive to concept drifts as seen in the previous section. In low dimensions, it is not necessary to implement a drift detection algorithm. On the other hand, one could argue that having a drift detection algorithm combined with accumulating data would give a more precise regression at the cost of computation time. However, we do not find this necessary at present as the algorithm presented in this article already outperforms other algorithms in classification accuracy in low dimensions. In Section 5.7, we implemented a simple drift detection algorithm along with a DR procedure and show that it improves the performance of the algorithm in high-dimensional cases.

Because we do not process previous data, it is important to ensure that the learned parameters converge. In other words, we require that the predictive accuracy to be an increasing monotonic function of time in the absence of drifts. This is to say that the learned parameters must be at least as good as the previous ones when no drift occurs. Recall that the particle filter algorithm learns by choosing the set of parameters giving the best accuracy on training data. To ensure a strictly increasing accuracy, we compare the performances of

each set of model parameters generated by SMC along with the mean of the learned parameters from the last batch. Then, the learned parameters in subsequent times are guaranteed to be at least as good as previous ones in noiseless environments.

We expect our algorithm to be highly competitive with others in mining real data in terms of classification accuracy, and stability against drifts. The algorithm and the effects of different input parameters are discussed in the next two subsections. Then we perform experiments to test the algorithm's performance in Section 5.

3.3. The Particle Filter-Based Learning Method Algorithm

We use the following notations, asterisked quantities, $\boldsymbol{\beta}^*$, denote the learned parameters selected from a set of particles, $\boldsymbol{\beta}$. The algorithm proceeds as follows: given a batch of data and the learned parameters from the previous batch, $\boldsymbol{\beta}^{*(n-1)}$, an ensemble of parameters (i.e., the particles) is randomly drawn from a chosen proposal distribution conditioned on previous particles with the largest TA. For the m-th particle, $\boldsymbol{\beta}_m^{(n)}$, the conditional proposal function is chosen to be the multivariate Gaussian distribution with mean $\boldsymbol{\beta}_{k_m^{(n-1)}}^{*(n-1)}$ and a diagonal covariance matrix Σ as an input, where $k_m^{(n-1)}$ is an index denoting the parent of $\boldsymbol{\beta}_m^{(n)}$. This process is to be repeated for subsequent batches. Also, we choose to learn logistic regression models as an example to test the proposed particle filter algorithm, and refer the particle filter with logistic regression algorithm as PF-LR.

The algorithm uses the following notations. The a-th instance in batch n is denoted by $\mathbf{x}_a^{(n)}$ and its class label is y_a, where $y_a \in \{0, 1\}$. The logistic function is

$$p\left(y_a^{(n)} = 1\right) = f(\eta_a) = \frac{1}{1 + e^{\eta_a\left(\boldsymbol{\beta}^{(n)}, \mathbf{x}_a^{(n)}\right)}}. \tag{4}$$

We use the following sign convention

$$\eta_a\left(\boldsymbol{\beta}^{(n)}, \mathbf{x}_a^{(n)}\right) = \boldsymbol{\beta}^{(n)} \cdot \left(-1, \mathbf{x}_a^{(n)}\right).$$

The prior distribution and the proposal function are set to be equal. The proposal function $q_n(\boldsymbol{\beta}^{(n)}|\boldsymbol{\beta}^{(n-1)})$ is set to be a multivariate Gaussian distribution with mean $\boldsymbol{\beta}^{(n-1)}$ and covariance matrix Σ.

The algorithm for each batch n is given in algorithm 1. After generating particles from the proposal distribution, the performance of each particle is evaluated by matching its predicted class labels with the true ones. Prediction of class labels is carried out by checking whether the value of the logistic function in equation (4) is larger than 0.5 for each instance, that is, $f(\boldsymbol{\beta}^{(n)}, \{\mathbf{x}, y\}^{(n)}) > 0.5$. The "mean particle" from the last batch, which is the mean of the classifier learned from the last batch, is also considered as a "generated" particle for this batch. The purpose of this is to stabilize PF-LR in static situations, so that subsequently learned classifiers are not worse than previous ones because of the random nature of Monte Carlo methods. After the performance of each particle is obtained, M auxiliary indices $k_m^{(n)}$, where $m \in \{1, \ldots, M\}$, are sampled uniformly among the indices corresponding to the best performing particles[1]. The purpose of this step is to assign the parents of $\boldsymbol{\beta}_m^{(n+1)}$ to be $\boldsymbol{\beta}_{k_m^{(n)}}^{(n)}$.

[1] As the batch size is typically smaller than the particle number, there will typically be more than one particle that gives the highest TA. For each particle, the logistic function with threshold set to be 0.5 is used to obtain the TA on each batch. Algorithm 2 shows this procedure.

The algorithm at each time step returns the set of best performing classifiers and the auxiliary indices for the next step. The inputs of PF-LR are the batch size B, parameters for the proposal (Gaussian) distribution Σ, the particle number M, and the training data. The rest of the inputs shown in algorithm 1 are obtained from time $n-1$. We assume that the components of all $\boldsymbol{\beta}$ are uncorrelated with each other, and a component-wise notation is used in the algorithm, where we write the m-th particle $\boldsymbol{\beta}_m^{(n)}$ as its i-th component $\beta_{im}^{(n)}$. Because the covariance matrix is diagonal with the i-th element being σ_i, drawing from the multivariate Gaussian proposal distribution is equivalent to generating each particle component $\beta_{im}^{(n)}$ from a one-dimensional Gaussian distribution $\mathcal{N}\left(\beta_{im}^{(n)}, \sigma_i\right)$ one by one.

Algorithm 1. PF-LR for the n-th batch

Inputs:

$\{\mathbf{x}, y\}^{(n)}$, the training data in batch n

D, the dimension of all $\boldsymbol{\beta}$

B, the number of instances in a batch

M, the number of particles

σ_i, the parameters of the proposal function

$\boldsymbol{\beta}_0^{(n-1)}$, the combined parameters from the previous batch

$k_m^{(n-1)}$, the indices where the m-th particle will be generated from

Outputs:

$\boldsymbol{\beta}_0^{(n)}$, the combined regression parameters for batch n

$\{\boldsymbol{\beta}^{*(n)}\}$, the ensemble of parameters learned from batch n

$k_m^{(n)}$, the indices where the m-th particle will be generated from in the next batch

Step 1: For $m \leftarrow 1 : M$

Step 1.1 : For $i \leftarrow 1 : D$, Generate $\beta_{im}^{(n)} \sim \mathcal{N}\left(\beta_{i,k_m^{(n-1)}}^{(n-1)}, \sigma_i\right)$.

Step 1.2 : Calculate TA $A_m \leftarrow A_m[f(\boldsymbol{\beta}_m^{(n)}, \{\mathbf{x}, y\}^{(n)}), B]$ from Algorithm 2.

Step 2: Calculate training accuracy using the classifier from last batch
$A_0 \leftarrow A_0[f(\boldsymbol{\beta}_0^{(n-1)}, \{\mathbf{x}, y\}^{(n)}), B]$

Step 3: Uniformly generate a list of M indices $k_m^{(n)} \in K^{(n)}$ for each particle, where $K^{(n)} = \{k^{(n)} | A_{k^{(n)}} = \max\{A_j\}, j \in \{0, \ldots, M\}\}$

Step 4: Get the averaged best performing particle for the next batch
$\boldsymbol{\beta}_0^{(n)} \leftarrow \bar{\boldsymbol{\beta}}$, where $\bar{\boldsymbol{\beta}}$ is the mean of $\boldsymbol{\beta}_{k^{(n)}}^{(n)}$ over all $k^{(n)}$

Step 5: $\{\boldsymbol{\beta}^{*(n)}\} \leftarrow$ the ensemble of $\boldsymbol{\beta}^{(n)}$ that gives the highest accuracy

Step 6: return $\boldsymbol{\beta}_0^{(n)}$, $\{\boldsymbol{\beta}^{*(n)}\}$ and $k_m^{(n)}$

Algorithm 2. Calculating accuracy from the training set
Inputs: $f(\cdot), \{\mathbf{x}, y\}^{(n)}, \boldsymbol{\beta}_m$, B
Outputs: A_m, the accuracy $\in [0, 1]$

Step 1: $A'_m \leftarrow 0$
Step 2: For $a \leftarrow 1 : B$
If $(f(\boldsymbol{\beta}_m, x_i^{(n)}) > 0.5$ AND $y_a^{(n)} = 1)||(f(\boldsymbol{\beta}_m, x_a^{(n)}) < 0.5$ AND $y_a^{(n)} = 0)$
then $A'_m \leftarrow A'_m + 1$
Step 3: return A'_m / B

Here we discuss the impacts of the inputs to the performance of PF-LR. In fact, a good choice of these parameters depends on whether the decision boundaries are static or dynamic. In static situations, large batch sizes would result in more accurate estimates of the regression parameters. However, in the presence of concept drifts, models learned from large batches do not necessarily give a better accuracy because the model parameters could vary significantly within a batch, and the model learned from the batch would not be representative of the truth. This results in a poor predictive accuracy as the model parameters are not learned well. On the other hand, too small of a batch could lead to a poor estimation of model parameters simply because of data sparsity.

The parameters of the proposal function σ_i control the spread of ensembles from the previously learned classifier, $\boldsymbol{\beta}^{*(n-1)}$. Therefore, σ_i controls the recovery time post-drift and the precision of learned parameters. In the absence of drifts, a small spread is preferred because it gives a higher precision. In contrast, a larger spread is favorable when the data contain large or rapid drifts because a larger spread allows the particle filter to search for the best estimate for the model parameters in a wider range. For fast drift recovery, the value of σ_i should be chosen so that PF-LR would generate at least one particle close to the new concept for a given particle number M with high probability.

The particle number M is the number of parameters generated from the proposal distribution of each batch. Generally, a larger M increases the precision of parameter estimation and improves drift recovery, but at the expense of computational efficiency. Finally, because the algorithm does not contain any adaptive elements, its complexity is linear. For a total of T batches with B instances of each batch and M particles, the complexity is of $O(MTB)$. Experimental results of the complexity is given in Section 5.

3.4. Likelihood versus Training Accuracy

Because of the small patch size required to give an accurate description of changing concepts, some conventional methods relying on optimizing the predictive accuracy may become ineffective for mining streaming data. For instance, predictive accuracy obtained from holding out a subset of the small batch at hand becomes unreliable, while cross validation may become too costly. This section shows that using the TA for particle selection in classification tasks is more suitable than using the likelihood function as performed in conventional particle filtering.

For batch n, finding an estimator that maximizes the TA is equivalent to minimizing the expected loss function

$$\mathop{\mathbb{E}}_{a \in S} \left\{ \left(y_a - I \left[f\left(\boldsymbol{\beta}^{(n)}, \mathbf{x}_a^{(n)} \right) > 0.5 \right] \right)^2 \right\},$$

where S is the set of indices denoting the training data, $I[\alpha]$ denotes the indicator function, with $I[\alpha] = 1$ if α is true and 0 otherwise. The predictive accuracy at batch n is calculated by using the estimator that maximizes the classification accuracy on a testing data set, which is independent and identically distributed (i.i.d.) as the training data. Given testing data $\{\mathbf{x}'_{a'}, y'_{a'}\}$, for $a' \in S'$, where S' is the set of indices of the testing data, the loss function corresponding to the predictive accuracy is

$$\mathop{\mathbb{E}}_{a' \in S'} \left\{ \left(y'_{a'} - I\left[f\left(\boldsymbol{\beta}^{(n)}, \mathbf{x}'^{(n)}_{a'} \right) > 0.5 \right] \right)^2 \right\}$$

Both loss functions have the same form. Because $\mathbf{x}$ and $\mathbf{x}'$ are i.i.d., the loss functions must also be i.i.d.. Therefore, as sizes of S and S' tend to ∞, both loss functions approach the same limit. As a result, maximizing the TA tends to maximize the predictive accuracy in the limit of large batch sizes.

The likelihood function is

$$L(\boldsymbol{\beta}|\mathbf{x}) = \prod_{a=1}^{S} [f(\boldsymbol{\beta}, \mathbf{x}_a)]^{y_a} [1 - f(\boldsymbol{\beta}, \mathbf{x}_a)]^{1-y_a}, \tag{5}$$

which is not i.i.d. with the predictive accuracy because they do not have the same functional form. Maximizing the likelihood is not guaranteed to maximize the predictive accuracy even in the asymptotic limit with large batch sizes. Therefore, classification tasks using the TA as a quality measure is expected to outperform the ones using the likelihood.

The presence of concept drifts poses another difficulty for the likelihood approach. Suppose a concept drift occurs between instances c and $c + 1$, with θ and θ' denoting the old and new concepts, respectively. Then the likelihood function for this batch would be

$$L(\theta, \theta'|\mathbf{x}) = \prod_{a=1}^{c} [f(\theta, \mathbf{x}_a)]^{y_a} [1 - f(\theta, \mathbf{x}_a))]^{1-y_a} \prod_{a=c+1}^{S} [f(\theta', \mathbf{x}_a))]^{y_a} [1 - f(\theta', \mathbf{x}_a))]^{1-y_a}.$$

However, the maximum likelihood approach aims to find the maximum likelihood estimator that maximizes equation (5), which assumes that all instances in the batch belong to the same concept. This is to say that the maximum likelihood estimator would maximize an unsuitable objective function. This is not the case with maximum TA, because it finds an estimator that still maximizes the predictive accuracy in the asymptotic limit. Therefore, it is expected that TA is more efficient for particle selection than the likelihood function. In Section 5, we will show that this is indeed the case.

4. DIMENSIONAL REDUCTION WITH DRIFT DETECTION

It is well known that particle filters perform poorly in high dimensions. To alleviate the detrimental effects from the high number of dimensions, we implement a drift detection algorithm with a DR procedure that is only used in high-dimensional cases. We say a drift is said to have occurred if the TA is less than a predefined threshold, A_c. Suppose this happens at batch n, the maximum likelihood estimator $\boldsymbol{\beta}_{MLE}$ for the current batch is found with gradient descent. Then, the likelihood ratio of the fitted model to the model $\boldsymbol{\beta}'_k$, where the k-th component of regression coefficient is set to zero (i.e., $\beta_k = 0$, where $k \in \{1, \ldots, D\}$), is calculated. The likelihood ratio with the k-th component set to zero is

$$D_k = -2 \log \frac{L\left(\boldsymbol{\beta}'_k | \mathbf{x}\right)}{L\left(\boldsymbol{\beta}_{MLE} | \mathbf{x}\right)},$$

which is χ^2 distributed. A χ^2 test for each D_i is performed to test the significance of each component, giving a p-value, p_k for each component k. DR is achieved by choosing only the significant components and setting all others to zero. To do this, a projection vector $\mathbf{P}$ with the k-th component $P_k = I[p_k < 0.05]$ is obtained, where $I[\cdot]$ denotes the indicator function. Then the particles are projected by $\mathbf{P}$ to give the dimensionally reduced particles.

Algorithm 3 shows this procedure, which is carried out immediately after obtaining the training accuracies of the particles when dealing with high-dimensional data. This algorithm returns the maximum likelihood estimator of the current batch, $\boldsymbol{\beta}^{(n)}_{MLE}$, and a projection vector $P^{(n)}$ onto the subspace where the p-values of the components are less than 0.05. These quantities are used in the next batch. If a drift is detected at batch n, the particles at batch $n+1$ are generated from $\boldsymbol{\beta}^{(n)}_{MLE}$, and the training accuracies $A_i, i \in \{1, \ldots, M\}$ are calculated from the projected particles, $\boldsymbol{\beta}^{(n+1)}_m = \mathbf{P}^{(n)} \cdot \boldsymbol{\beta}^{(n)}_m$. The projection vector is kept constant until another drift is detected. The TA, A_0, calculated from the mean of the best performing particles in the last batch, is not projected by $\mathbf{P}^{(n)}$ so that the mean is still in the subspace of the previous concept. This makes the algorithm more robust to noise because particles in the new concept is compared along with the previous particle mean that resides in the subspace of the previous concept. The analysis in Section 5.7 shows that a DR procedure dramatically improves the performance of PF-LR in high dimensions.

Algorithm 3. Dimensional reduction with drift detection

Inputs:
$\{\mathbf{x}, y\}^{(n)}$, data in batch n
$max\{A_i\}$, the maximum training accuracies of all particles,
A_c, the threshold on the global training accuracy
Outputs:
$P^{(n)}$, a projection vector for the relevant dimensions
$\boldsymbol{\beta}^{(n)}_{MLE}$, the maximum likelihood estimator for batch n.

Step 1: If $\max\{A_i\} < A_c$
then $\boldsymbol{\beta}^{(n)}_{MLE} \leftarrow$ the maximum likelihood estimator using gradient descent

Step 2: For $k = 1 : D$

Step 2.2: $\boldsymbol{\beta}'^{(n)}_k \leftarrow$ the fitted model with $\beta_k = 0$ using gradient descent

Step 2.3: $D_k \leftarrow -2 \log \frac{L\left(\boldsymbol{\beta}'^{(n)}_k | \mathbf{x}^{(n)}\right)}{L\left(\boldsymbol{\beta}^{(n)}_{MLE} | \mathbf{x}^{(n)}\right)}$

Step 2.4: $P^{(n)}_k \leftarrow I[p_k < 0.05]$

Step 3: return $P^{(n)}$, $\boldsymbol{\beta}^{(n)}_{MLE}$

5. EXPERIMENTAL EVALUATION

In this section, we compare the performance of the PF-LR with other algorithms. First, we compare the performances of using TA and using the likelihood as quality measures. We then compare different particle selection methods, namely, using either a fixed number of best performing particles or just the best performing ones. Then, we compare PF-LR with the APF and its regularized variant (RegAPF) as well as the regularized version of PF-LR (RegPF-LR).

We also compare PF-LR to other data streaming classification algorithms: Hoeffding Tree (Domingos and Hulten 2000) LB-HT[2] (Bifet et al. 2010a), DWM-NB[3] (Kolter and Maloof 2007), the VW system[4] (Langford et al. 2007), batch Support Vector Machine (batch-SVM[5]), and batch k-nearest neighbor (batch-kNN[6]). SVM constructs a linear decision boundary, such that the distance between the boundary and the nearest data points of the two classes is maximized. kNN predicts the class membership as the majority class within the k nearest instances of a test point. DWM-NB is an online learning algorithm that processes one example at a time without drift detection, whereas LB-HT employs adaptive sliding windows as a drift-detection mechanism. Vowpal Wabbit is a scalable implementation of stochastic gradient descent. Unless otherwise stated, throughout this section we choose the batch size to be $B = 50$, the particle number $M = 100$, and the parameters of the proposal function $\sigma_i = 0.1$ for all i. This fixed choice of σ_i is justified by the results comparing our algorithm with the regularized particle filters shown in this section.

5.1. Data Sets

We use three commonly used synthetic data sets (SEA (Street and Kim 2001), CIRCLES (Nishida and Yamauchi 2007), and HYPERPLANE (Hulten et. al. 2001)), another synthetic data set we created that contains many drifts (which we call MANY), a real electricity pricing data set (Harries 2007) to demonstrate PF-LR's drift adaptability and noise resistance on streaming data, and two real static data set obtained from the UCI Machine Learning Repository (Bache and Lichman 2013), breast cancer and skin segmentation data, to show that PF-LR can be used in real static data set for incremental learning.

All synthetic data sets used in this section are generated by the data generator from Minku et al. (2010) and contain 10% class noise unless stated otherwise. The specifics of each data set are given as follows. The sizes of the SEA, HYPERPLANE, and CIRCLES data set are the same used in Kolter and Maloof (2007). The size of the MANY data set is determined by requiring that a drift occurs every 100 examples for a total of 20 concepts.

- The SEA data set contains feature variables $\mathbf{x} \in [0, 10]^3$, with class labels $y = 1$ if $x_1 + x_2 < \mu$ and $y = 0$ otherwise, where the value of μ changes over time to simulate concept drifts. The SEA data set in our experiments has four concepts (i.e., μ takes a value from $\{8, 9, 7, 9.5\}$ in turn). The training data set contains 40,000 examples with

[2] The input parameters we used for LB-HT are $ensembleSize = 10, weightShrink = 6, deltaAdwin = 0.002, gracePeriod = 200, splitConfidence = 0, tieThreshold = 0.05$ and the splitting criterion is according to the information gain.

[3] The input parameters we used for DWM-NB are $\beta = 0.5, \theta = 0.01, maxExperts = \infty$, and $period = 50$.

[4] The default parameters of VW are used, and we only allow 1 pass through each batch.

[5] We follow the method of Shalev-Shwartz et al. (2007) and set $\lambda = 0.001$.

[6] We set k = 10., which is the default in MOA.

TABLE 1. The concepts, their signal-to-noise ratio, and their class IR in CIRCLES.

Concepts	β_0'	β_1'	β_2'	ρ	IR
1	0.15	0.2	0.5	0.17	0.018
2	0.2	0.4	0.5	0.30	0.032
3	0.25	0.6	0.5	0.51	0.052
4	0.3	0.8	0.5	0.67	0.076

The IR is calculated from the areas in the feature space corresponding to the two classes, that is, $IR = \pi\beta_0'^2 / \left(4 - \pi\beta_0'^2\right)$.

10,000 examples for each concept. As a preprocessing step, the feature variables are normalized to the range [0,1].

- The feature space of CIRCLES is specified by $\mathbf{x} \in [0, 2]^2$, with class labels $y = 1$ if $\eta' < 0$ (i.e., inside a circle) and $y = 0$ otherwise , where

$$\eta' = -\beta_0'^2 + \left(x_1 - \beta_1'\right)^2 + \left(x_2 - \beta_2'\right)^2.$$

Concept drifts are simulated by changing the values for β_0', β_1', and β_2' (which represent the radius and center of the circle). The CIRCLE data set contains four concepts whose corresponding β_i' (for $i = 0, 1, 2$) values are shown in Table 1. The training set for CIRCLE contains 40,000 examples with 10,000 examples for each concept. In addition, the training set is imbalanced, with the examples inside the circle belonging to the minority class. The class imbalance ratio for each concept is also shown in Table 1. The four concepts also have an increasing signal-to-noise ratio, defined by

$$\rho = \frac{\text{number of true class labels in the minority class}}{\text{number of false class labels in the minority class}}.$$

and listed in Table 1. Because the training sets are highly imbalanced, we test the algorithm on testing data that contain an equal number of the two classes for each concept, giving a baseline accuracy of 50%.

- The feature space of MANY is such that $\mathbf{x} \in [0, 10]^3$. It contains 20 concepts with drifts occurring every other batch. An instance belongs to class 1 if

$$-\beta_0' + \beta_1' x_1 + \beta_2' x_2 + \beta_3' x_3 < 0,$$

and it belongs to class 0 otherwise. The β_i' (for $i = 0, 1, 2, 3$) values for the 20 concepts are listed in Table 2. The instances are normalized to [0,1] before processing. The training data set for MANY contains 2000 examples, with 100 examples belonging to each concept.

- HYPERPLANE contains features variables $\mathbf{x} \in [0, 1]^{10}$, with class labels $y = 1$ if $(x_i + x_{i+1} + x_{i+2})/3 > 0.5$, and $y = 0$ otherwise. The data set has four concepts each with a different i value taken from $\{1, 2, 4, 7\}$. The training data set for HYPERPLANE contain 40,000 examples, with 10,000 examples for each concept. To test the algorithms on high-dimensional data, we further increase the number of dimensions D by appending the data set with feature variables uniformly generated on $[0, 1]^d$, where $d \in \{5, 10, 15, 20\}$. When dimensionality is not mentioned for this data set, $D = 10$. The DR technique discussed in Section 4 is used only on this data set.

TABLE 2. The 20 concepts in MANY containing gradual small and sudden large drifts.

Concepts	β'_0	β'_1	β'_2	β'_3	Concepts	β'_0	β'_1	β'_2	β'_3
1	8.3	0.5	0.7	0.6	11	8.5	0.49	0.38	0.49
2	8	0.72	0.41	0.52	12	8	0.59	0.6	0.49
3	8.5	0.66	0.55	0.56	13	8.2	0.64	0.4	0.64
4	8.10	0.52	0.6	0.62	14	8	0.65	0.5	0.65
5	8.5	0.44	0.35	0.44	15	8.5	0.68	0.4	0.68
6	8.6	0.78	0.5	0.68	16	8.1	0.71	0.55	0.71
7	8.5	0.54	0.55	0.54	17	7.6	0.75	0.31	0.65
8	7.8	0.78	0.58	0.68	18	7.3	0.46	0.31	0.66
9	8.1	0.43	0.51	0.43	19	7	0.4	0.7	0.5
10	8.0	0.44	0.54	0.54	20	8	0.5	0.5	0.5

- The ELEC data set contains a time series of 45,312 instances recorded at 30-min intervals. The class labels are UP and DOWN, which indicate whether the price is higher than the moving average price over the last 24 h. The original data set contains eight variables including the ones denoting the time, *date, day*, and *period*. We ignore these three variables and rely solely on the particle filter to track the drifts as each batch is processed. The remaining five feature variables are *nswprice, nswdemand, transfer, vicprice*, and *vicdemand*. The values of each of these variables are normalized by a factor of 1,000 to bring them to the range of order 1.
- The Wisconsin Diagnostic Breast Cancer (WDBC) data set has 569 instances, each of which contains 30 features of cell nuclei, extracted from images of breast mass, and a label denoting the diagnosis (either "benign" or "malignant"). There are 357 benign and 212 malignant samples in the data set.
- The Skin Segmentation (SS) data set has 245,057 instances. Three attributes denoting the color (B, G, and R) values from face images of various age groups, race groups, and genders. The class labels denote whether the images are taken from skin samples. The classes are best separated by nonlinear decision boundaries in the feature space.

To obtain the predictive accuracy on each of the four synthetic data sets (SEA, CIRCLES, MANY, and HYPERPLANE), the learned model from each batch of the training data is tested on 1,000 testing examples generated from the same concept of the training batch. Because no testing data can be generated for the real ELEC data set, prequential evaluation is performed to obtain the predictive accuracy on this data set, where the model learned from the current batch is used to predict the class labels for the examples in the next batch. The reported predictive accuracies are averaged over 50 runs.

Care is taken when comparing algorithms with different learning techniques. The DWM-NB, LB-HT, and VW are incremental algorithms that process one example at a time, whereas PF-LR can process a batch of 50 instances for the synthetic data sets. To estimate the predictive accuracy of incremental algorithms, we train a model on each batch and then test it on the same 1,000 instances as PF-LR. The performances of these algorithms[7] are evaluated on 1,000 testing examples for each batch. We compare the average predictive accuracy of DWM-NB over a batch to those of PF-LR and LB-HT.

[7] We used Weka (Hall et al. 2007) to evaluate DWM-NB and MOA (Bifet et al. 2010b) to evaluate LB-HT.

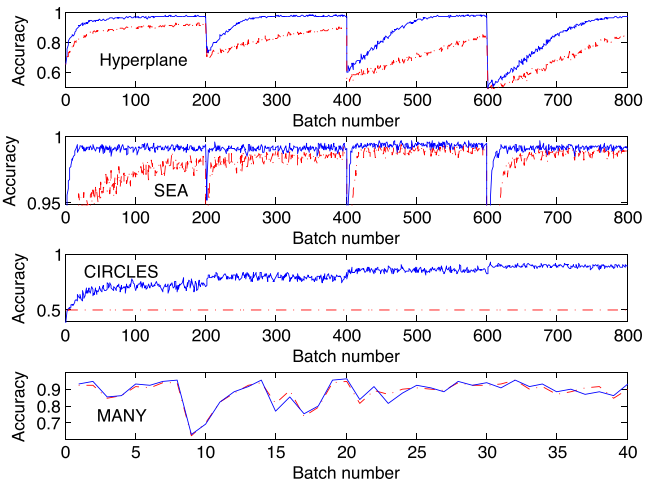

FIGURE 2. Predictive accuracy using the likelihood (dot-dashed) and training accuracy (solid). The noise level of the training data set is set to zero. Also, dimensional reduction is not used on the HYPERPLANE data set (where the number of dimensions is 10). Drifts occur right after batches 200, 400, and 600 on HYPERPLANE, SEA, and CIRCLES. On MANY, drifts occur at every other batch. [Color figure can be viewed at wileyonlinelibrary.com]

During our investigation, we found that an algorithm that learns a model with a linear decision boundary is not able to function well on the CIRCLES data set (which has a circle decision boundary), giving a predictive accuracy of 50%. Because particle filters are independent of the type of model it learns, a circular decision boundary is chosen for the logistic regression model[8] when learning from CIRCLES with all the particle filter methods evaluated in this section. However, because this choice may be too specific to this data set, we only use CIRCLES to compare different particle filter methods and omit the comparisons with DWM-NB, LB-HT, and VW on this data set because we are not able to adapt these three algorithms according to the data set.

5.2. Comparisons with the Maximum Likelihood Approach

In this section, we compare the results of using the likelihood function in the particle filter versus those of using the TA on the synthetic data sets. The noise level of the training data is set to zero in this experiment to emphasize that TA outperforms the likelihood as a quality measure regardless of noise. The particles that give the top-k values of the likelihood function are chosen to perform majority voting. Figure 2 shows that PF-LR with training accuracy outperforms PF-LR with the likelihood function on HYPERPLANE, SEA, and CIRCLES with $k = 20$. On MANY, the two algorithms have similar performances. We notice that the TA approach performs much better when the classes are highly imbalanced as in the CIRCLES data set, on which the approach with the likelihood function breaks down, with its predictive performance the same as random guessing at 50%.

Table 3 shows the overall performances (i.e., the average predictive accuracy over all the batches) of the training accuracy and the likelihood methods. Also shown in the table is the predictive accuracy obtained from the logistic regression model learned using the gradient descent method on every batch. We see that the TA method outperforms the likelihood method and gradient descent on SEA, CIRCLES, and HYPERPLANE, whereas the results

[8] That is, in Equation (4), η_a is set to $\beta_0^2 + (x_{1a} - \beta_1)^2 + (x_{2a} - \beta_2)^2$.

TABLE 3. Average predictive accuracies of particle filter-based learning method using particles with the highest training accuracy (MAX TA), using the top-k likelihoods (L), and from gradient descent (GD) on every batch.

Data set	MAX TA	Top 5 (L)	Top 20 (L)	Top 50 (L)	GD
SEA	0.990 ± 0.001	0.976 ± 0.002	0.978 ± 0.002	0.979 ± 0.003	0.894 ± 0.045
CIRCLES	0.808 ± 0.014	0.500 ± 0.000	0.500 ± 0.000	0.500 ± 0.000	0.580 ± 0.149
MANY	0.882 ± 0.030	0.889 ± 0.032	0.878 ± 0.036	0.866 ± 0.035	0.885 ± 0.034
HYPERPLANE	0.975 ± 0.013	0.615 ± 0.157	0.619 ± 0.195	0.602 ± 0.158	0.808 ± 0.034

Noiseless training sets are used.

TABLE 4. Comparison between the method of using the best performing particles (MAX TA) and the method of using particles with the top-k training accuracies.

Data set	MAX TA	Top 5 (TA)	Top 20 (TA)	Top 50 (TA)
SEA	0.981 ± 0.003	0.979 ± 0.004	0.972 ± 0.004	0.985 ± 0.002
CIRCLES	0.769 ± 0.003	0.754 ± 0.052	0.723 ± 0.030	0.575 ± 0.013
MANY	0.881 ± 0.041	0.878 ± 0.044	0.919 ± 0.016	0.872 ± 0.046
HYPERPLANE	0.968 ± 0.006	0.975 ± 0.005	0.976 ± 0.005	0.970 ± 0.007
ELEC (B=10)	0.876 ± 0.082	0.800 ± 0.194	0.836 ± 0.104	0.815 ± 0.121

Dimensional reduction is not used on the HYPERPLANE data set (where the number of dimensions is 10). The results here justify only taking the best performing particles in the selection procedure.

on the MANY data set are not statistically significant. These results illustrate that the likelihood is much more affected by small sample sizes (which are necessary for learning from data streams) than the TA method. Note that we have left out the result for the ELEC data because this experiment is conducted in a noiseless environment, and it is impossible to confirm that the ELEC data set is noiseless.

5.3. Using a Fixed Size Ensemble

Table 4 compares the predictive accuracies obtained by fixing the ensemble size to be the top-k performing particles regardless of their training accuracies with the PF-LR where only the particles giving the best TA are included in the ensemble. In this experiment, 10% noise is added to the SEA, MANY, and HYPERPLANE data sets by randomly flipping the class labels in the training data, and the noise level of CIRCLE is given in Table 1.

The results show that including only the top performing particles in the ensemble outperforms the method of a fixed ensemble size on two of the five data sets (i.e., CIRCLES and ELEC). On the three other data sets used, the two methods give comparable results. Note that the standard deviations of using a fixed size ensemble on CIRCLES (for all k values) and on ELEC (for $k = 20$) are much larger than the method of choosing only the best performing particles. This is to be expected, because particles are generated in the neighborhood of the best performing particles in the last concept. Immediately after a drift, only a small number of particles can give the maximum TA. Fixing the number of particles in an ensemble is likely to include those that are far from accurate.

5.4. Comparisons with the Auxiliary Particle Filters

This section compares PF-LR with the APF and its regularized counterpart (RegAPF). Figure 3 shows the results. For completeness, we also evaluated both APF and RegAPF using the TA as a quality measure for selecting particles. The results for these methods are summarized in Table 5. Note that the algorithms of APFs assume that we have complete

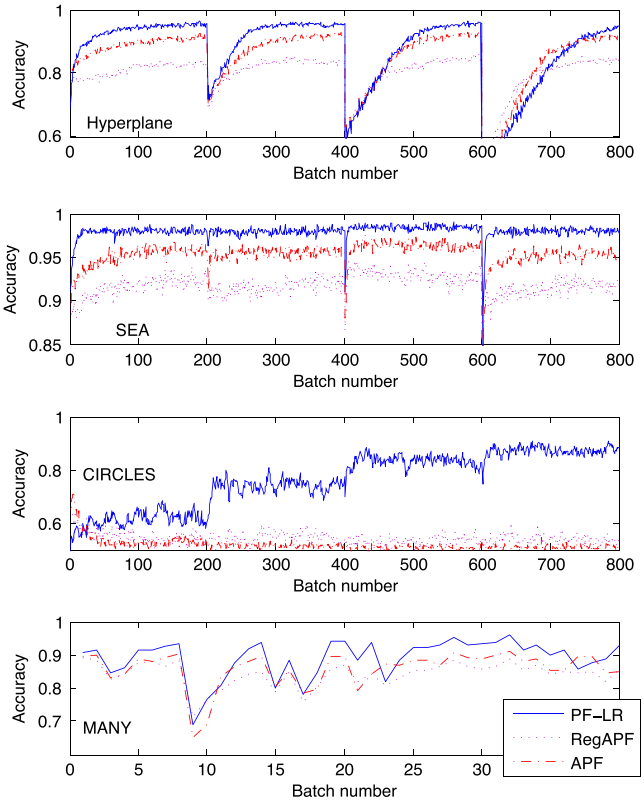

FIGURE 3. Predictive accuracy of auxiliary particle filter (APF), regularized APF, and particle filtering-based learning method (PF-LR). PF-LR outperforms both APF and RegAPF, which use the maximum likelihood estimator for the model parameters. [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 5. Average predictive accuracies of APF and RegAPF with the TA quality measure. The results here justify the use of a fixed σ_i.

Data set	PF-LR	APF (TA)	RegAPF (TA)
SEA	0.981 ± 0.030	0.933 ± 0.012	0.924 ± 0.010
CIRCLES	0.769 ± 0.030	0.503 ± 0.030	0.500 ± 0.000
MANY	0.881 ± 0.041	0.855 ± 0.067	0.854 ± 0.039
HYPERPLANE	0.868 ± 0.037	0.692 ± 0.056	0.697 ± 0.055

The results here justify the use of a fixed σ_i.
PF-LR, particle filter-based learning method; APF, auxiliary particle filter; RegAPF, regularized auxiliary particle filter; TA, training accuracy.

knowledge (e.g., the class labels) of the examples in the next batch. For prequential evaluation such as the one used on ELEC, the class labels of the next batch cannot be used in learning. Therefore, APFs are not suitable for prequential evaluations, and the results on the ELEC data set are omitted. Also, in order to have a fair comparison with the APF algorithms, we did not include DR on the HYPERPLANE data set in this experiment. We found that PF-LR outperforms APF and RegAPF on every data set tested.

5.5. The Regularized Particle Filter-Based Learning Method

In this section, we compare the performance of PF-LR with the performance of a regularized version of PF-LR (which we call RegPF-LR). The regularization part of RegPF-LR is based on the regularized particle filter (Casarin and Marin 2009). The formalism of RegPF-LR is discussed in appendix 6. Overall, we found that RegPF-LR does not outperform PF-LR.

To initiate RegPF-LR, a preliminary run of 500 iterations was performed on the first batch. This is to choose suitable initial values of the logistic regression coefficients and the proposal parameters. We compare the performances of RegPF-LR with those of PF-LR in Figure 4, which shows that PF-LR outperforms RegPF-LR on all synthetic data sets

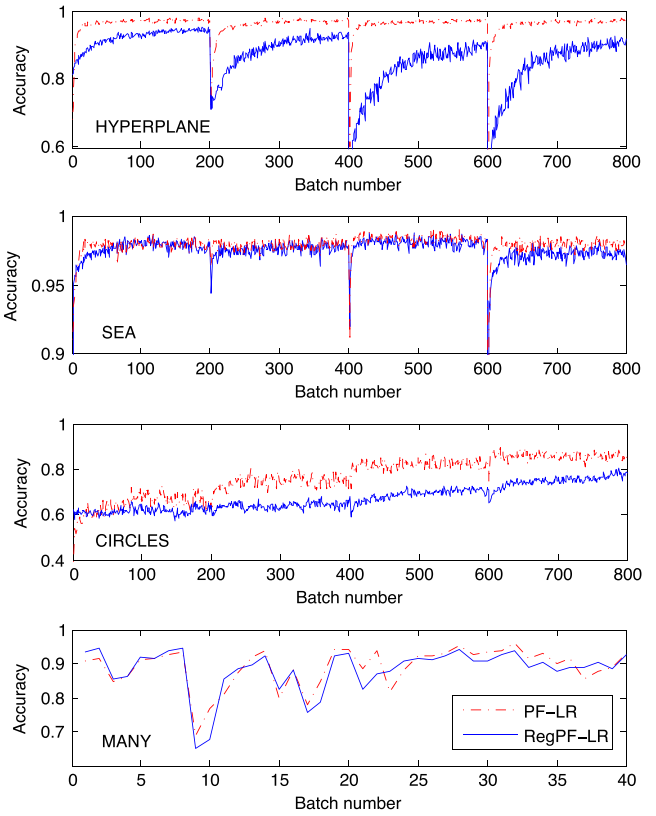

FIGURE 4. Comparisons between regularized particle filter-based learning method (RegPF-LR) and PF-LR. The results show that PF-LR outperforms RegPF-LR on HYPERPLANE, SEA, and CIRCLES and performs just as well in MANY. [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 6. Average predictive accuracies of PF-LR and RegPF-LR on the ELEC data set with batch size $B = 10$.

# of features	1	2	3	4	5
PF-LR	0.898 ± 0.009	0.891 ± 0.005	0.886 ± 0.012	0.883 ± 0.012	0.867 ± 0.082
RegPF-LR	0.910 ± 0.005	0.903 ± 0.007	0.882 ± 0.064	0.866 ± 0.052	0.860 ± 0.080

The number of features, D (for $D = 1$ to 5), means that the first D variables in the list of *nswprice, nswdemand, transfer, vicprice*, and *vicdemand* are used.
PF-LR, particle filter-based learning method; RegPF-LR, regularized particle filter.

considered. Table 6 shows the comparison between the two algorithms on the ELEC data set with respect to different numbers of features used in the evaluation. We found that when a very low number of features are used, RegPF-LR slightly outperforms PF-LR. However, PF-LR is able to outperform RegPF-LR as the number of features increases.

5.6. Comparisons with Naive Bayes with Dynamic Weighted Majority, Hoeffding Tree with Leveraging Bagging, and Vowpal Wabbit System

The average predictive accuracies over 50 runs with PF-LR, LB-HT, DWM-NB, and VW on SEA and MANY are shown in Figure 5. We defer the discussion of the comparison on HYPERPLANE to Section 5.7. As noted in Section 5.1, we omit the comparison with LB-HT, DWM-NB, and VW on the CIRCLE data because they are not good at and not adjusted to learning concepts with circular decision boundaries. The figure shows that PF-LR outperforms both DWM-NB, LB-HT, and VW. The average predictive accuracies and the corresponding 95% confidence intervals are listed in Table 7, in which we also include the results for the HYPERPLANE data set (although we defer the discussion on it to Section 5.7). The facts that DWM-NB, LB-HT, and VW retain historical data and that they are more prone to noise imply that the classifiers generated by these algorithms may not be up-to-date after drifts. In contrast, PF-LR relies more on the examples in the current batch, and its performance is expected to be higher in the case of constant drifts.

It can be seen that for the SEA data set, PF-LR recovers from drifts extremely fast. Whereas DWM-NB accumulates instances from earlier times, which leads to a significant drop in its predictive accuracy immediately after drifts. LB-HT is somewhat insensitive to drifts because its performance stays roughly constant. However, it is not able to give predictions as accurate as those of PF-LR and DWM-NB, and its drift recovery time is slow. The performance of VW is far worse than these algorithms with a predictive accuracy of 85.2%.

For the MANY data set, PF-LR almost always performs better than DWM-NB, especially between batch numbers 25 and 35, where PF-LR is seen to be unaffected by drifts and the performance of DWN-NB is seen to drop rapidly immediately after certain drifts. The behavior of LB-HT is interesting. It is least affected by the drift at batch number 8, presumably because of its drift detection mechanism. However, it is unable to attain high accuracies possibly because of contamination from out-of-date examples when the drifts are too small to be picked up by the drift detection. VW is not seen to be affected by the drift at batch number 8; however, it cannot obtain predictive accuracy higher than 85% for the majority of the batches.

Because PF-LR is a batch method, we also compare with support vector machine and k-nearest neighbor in batch mode, and the results are summarized in Table 7. Typically, these algorithms have worse performance than incremental methods on the data set and batch sizes analyzed.

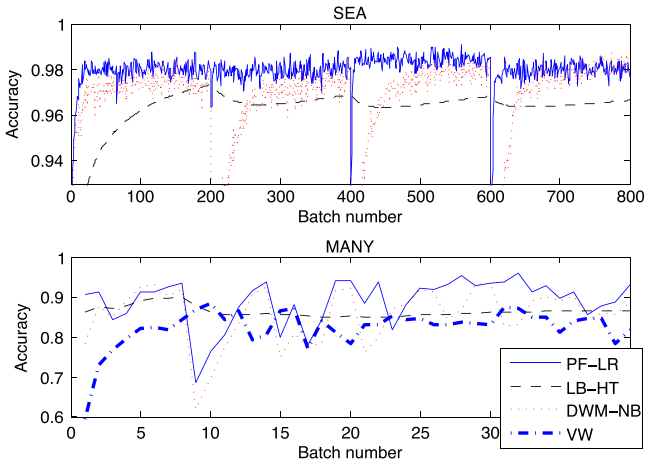

FIGURE 5. Predictive accuracies of particle filter-based learning method, Naive Bayes with Dynamic Weighted Majority, Hoeffding Tree with Leveraging Bagging, and Vowpal Wabbit (VW) system. The accuracy of VW on SEA (∼85%) is not shown as it is far below the accuracies of the other algorithms. [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 7. The average predictive accuracies of PF-LR and incremental algorithms for various data sets.

Data set	PF-LR (%)	DWM-NB (%)	LB-HT (%)	VW (%)
SEA	(**98.1** ± 0.3)	(96.5 ± 0.7)	(96.3 ± 0.5)	(85.2 ± 3.3)
MANY	(**88.1** ± 4.1)	(86.4 ± 1.6)	(86.4 ± 2.9)	(82.5 ± 10.0)
HYPERPLANE (B=100)	(**97.3** ± 0.6)	(96.0 ± 0.8)	(83.3 ± 0.5)	(81.8 ± 2.6)
ELEC (B=10)	(**87.6** ± 8.2)	80.8	86.3	67.9
WDBC (B=100)	(95.13 ± 2.18)	**96.0**	93.5	61.9
SS (B = 50)	(94.5 ± 0.002)	92.4	**99.5**	81.6

The highest value obtained for each data set is in bold.
PF-LR, particle filter-based learning method; DWM-NB, Naive Bayes with Dynamic Weighted Majority; LB-HT, Hoeffding Tree with Leveraging Bagging; VW, Vowpal Wabbit system.

The results in Table 7 show that PF-LR has better drift adaptability and predictive performance than DWM-MN, LB-HT, and VW on all the data sets tested except WDBC, where the difference to the best performing algorithm, DWM-NB, is statistically insignificant, and SS, where the decision boundary is nonlinear. On SS, PF-LR outperforms DWM-NB and VW, but is less impressive when compared with LB-HT. This is because the SS data set requires a nonlinear boundary decision boundary, and decision tree methods are expected to perform better than a simple logistic regression, which PF-LR employs.

Table 8 Shows the comparison of PF-LR with batch-based methods. There it is shown that PF-LR significantly outperforms batch-SVM on all the data sets tested. Similarly, batch-kNN has a much lower performance than PF-LR except for the SS data set.

5.7. Performance of Particle Filter-Based Learning in High Dimensions

This section investigates the effect of using DR with PF-LR on the HYPERPLANE (Hulten et al. 2001) data set. Figure 6 shows that using a DR procedure described in Section 4 dramatically improves the performance of PF-LR.

TABLE 8. The average predictive accuracies of PF-LR and batch-based algorithms for various data sets.

Data set	PF-LR (%)	Batch-SVM (%)	Batch-kNN (%)
SEA	(**98.1** ± 0.3)	(86.0 ± 3.1)	(95.6 ± 0.4)
MANY	(**88.1** ± 4.1)	(64.7 ± 9.22)	(85.7 ± 1.8)
HYPERPLANE (B=100)	(**97.3** ± 0.6)	(87.0 ± 1.2)	(84.9 ± 0.6)
ELEC (B=10)	(**87.6** ± 8.2)	67.7	77.9
WDBC (B=100)	(95.1 ± 2.18)	56.7	74.1
SS (B = 50)	(94.5 ± 0.002)	81.9	97.9

The highest value obtained for each data set is in bold.
PF-LR, particle filter-based learning method; Batch-SVM, batch mode supported vector machine; Batch-kNN, batch mode k-nearest neighbor.

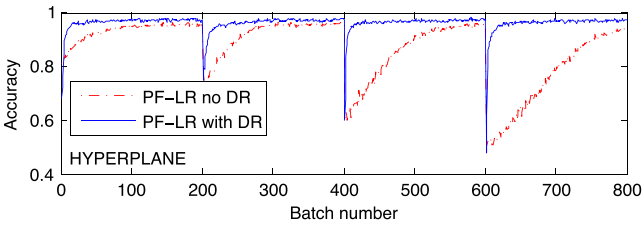

FIGURE 6. The improvement that resulted from dimensional reduction. The number of dimensions in the input HYPERPLANE data is 10 (i.e., D=10). [Color figure can be viewed at wileyonlinelibrary.com]

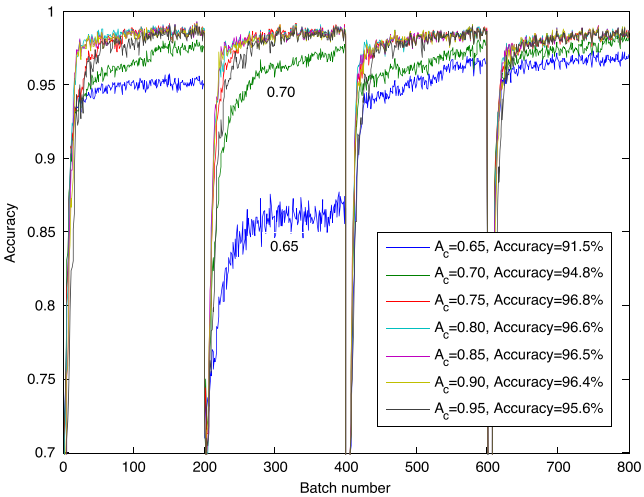

FIGURE 7. The performance of the particle filter-based learning method with various accuracy thresholds A_c. From top to bottom, the values of A_c ranges from 0.65 to 0.95 in increments of 0.05. The performance is robust with a choice of $A_c > 0.7$. The number of dimensions in the input HYPERPLANE data is 10 (i.e., D=10). [Color figure can be viewed at wileyonlinelibrary.com]

The DR algorithm contains an adjustable parameter, A_c. We found that the performance of PF-LR is robust against the choice of A_c over a large range. Figure 7 shows the predictive accuracy of PF-LR for different choices of A_c, in which the average predictive accuracy for

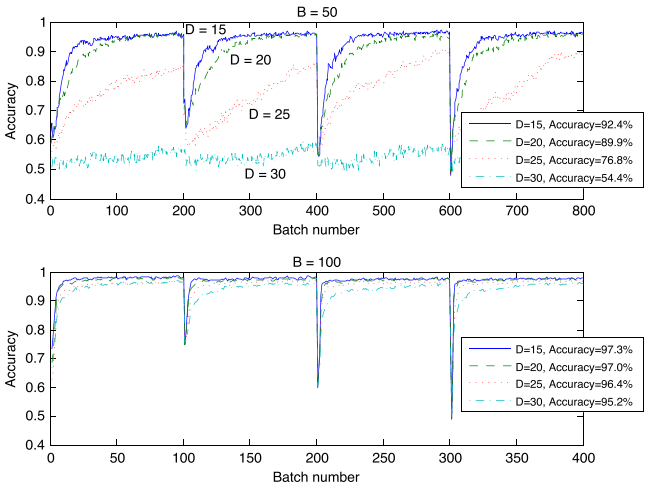

FIGURE 8. The predictive accuracy of the particle filter-based learning method at different batch sizes and numbers of dimensions. Top: batch size $B = 50$. Bottom: $B = 100$. [Color figure can be viewed at wileyonlinelibrary.com]

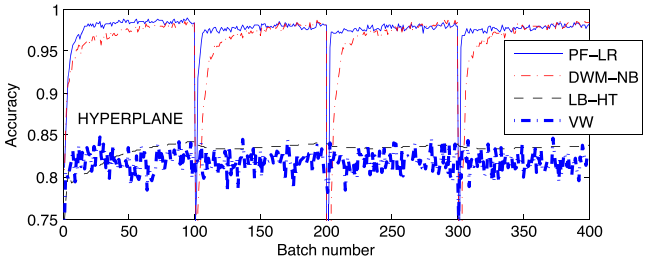

FIGURE 9. The predictive accuracies of the particle filter-based learning method, Naive Bayes with Dynamic Weighted Majority, Hoeffding Tree with Leveraging Bagging, and the Vowpal Wabbit system with batch size $B = 100$ and number of dimensions $D = 10$. [Color figure can be viewed at wileyonlinelibrary.com]

each A_c is also shown. It is clear that the results are insensitive to the choice of A_c for the range $0.7 < A_c < 0.95$. We must note that for extreme values of A_c, for instance, $A_c = 0.95$, the performance is somewhat less than that of a reasonable choice of A_c. The reason is that, at such high values, DR can be triggered in consecutive batches because particles that give training accuracies higher than A_c are less likely to be generated. As a consequence, particles are generated from $\boldsymbol{\beta}_{MLE}^{(n)}$ in every batch until DR stops being triggered. Because $\boldsymbol{\beta}_{MLE}^{(n)}$ does not give the best predictive accuracy, the overall performance of PF-LR with very high values of A_c drops.

Figure 8 shows the effect on the performance of PF-LR with DR as the number of dimensions, D, increases.[9] The performance deteriorates quite quickly as D increases to 30. The reason is that a batch size of 50 is insufficient for high dimensions. Once the batch size is increased to 100, the performance improves dramatically.

Finally, we compare PF-LR with DR and batch size $B = 100$ with DWM-NB, LB-HT, and VW. Figure 9 shows the case with $D = 10$, and Table 9 summarizes the average

[9] See Section 5.1 for how we increase the number of dimensions in HYPERPLANE.

TABLE 9. The average predictive accuracies of PF-LR with DR, DWM-NB, LB-HT, and VW in various numbers of dimensions D.

Accuracy	PF-LR	DWM-NB	LB-HT	VW
$D = 10$	0.973 ± 0.006	0.960 ± 0.008	0.833 ± 0.005	0.818 ± 0.026
$D = 15$	0.970 ± 0.009	0.954 ± 0.007	0.829 ± 0.006	0.770 ± 0.030
$D = 20$	0.964 ± 0.011	0.950 ± 0.007	0.825 ± 0.006	0.745 ± 0.031
$D = 25$	0.952 ± 0.020	0.946 ± 0.007	0.822 ± 0.006	0.725 ± 0.030
$D = 30$	0.937 ± 0.028	0.951 ± 0.007	0.818 ± 0.007	0.706 ± 0.032

PF-LR, particle filter-based learning method; DWM-NB, Naive Bayes with Dynamic Weighted Majority; LB-HT, Hoeffding Tree with Leveraging Bagging; VW, Vowpal Wabbit system.

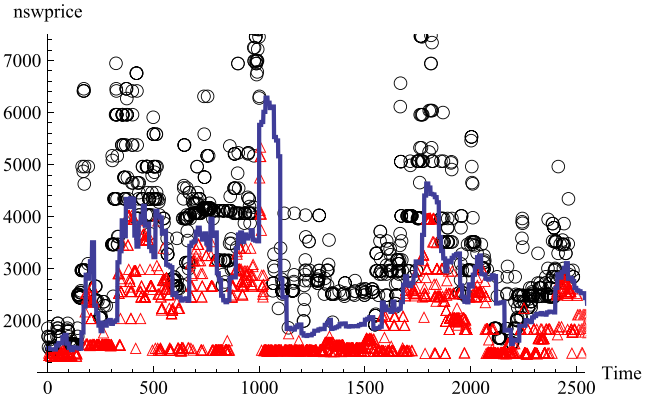

FIGURE 10. Values of the feature variable $nswprice$ of the first 2500 instances. The markers denote the classes DOWN (red triangle) and UP (black circle). It is evident that drifts occurs throughout the data set. The solid line shows the decision boundary learned by particle filter-based learning method. [Color figure can be viewed at wileyonlinelibrary.com]

predictive accuracies from $D = 10$ to $D = 30$. We found that PF-LR is comparable with DWM-NB, and both are much better than LB-HT and VW.

5.8. Real Data Analysis: Electricity Pricing

In this section, we apply PF-LR to the electricity pricing data set (ELEC) (Harries 2007), which has been extensively studied by other authors (Z'liobaite 2013). Here we illustrate how the predictive performance of PF-LR changes with respect to the batch size and the feature set used.

We found that PF-LR gives the best performance when using only one feature, *nswprice*, when a reasonable batch size is chosen. Including other features would slightly deteriorate the performance by up to a few percent. Figure 10 shows the values of *nswprice* of the first 2500 instances. An instance is shown using a circle if its class label is UP, and a triangle if its class label is DOWN. The predicted concept is shown as the boundary separating the two classes. Because this data set contains frequent drifts, it is crucial to use a batch size as small as possible so that the concept in the next batch is approximately the same as the current batch. On the other hand, the batch size must be large enough so that logistic regression is accurate.

As mentioned in Section 5.1, prequential accuracies are used as the predictive performance measure for this real data set. The results for different batch sizes and numbers of variables used are shown in Table 10 and Figure 11. As expected, the performance of PF-LR improves as the batch size decreases, because of the fact that smaller batches give a better description of the current concept. However, its precision also drops (i.e., its 95% confidence interval also increases) as logistic regression becomes unreliable because of data sparseness. As a sanity check, we plot the decision boundary obtained by PR-LR in Figure 10. In analogy to Figure 1, we see that PF-LR is able to track the movement of the decision boundary very well.

Figure 11 shows that using only the *nswprice* variable gives the best results. However, when we compare PF-LR with other algorithms, all five features are used to be consistent with the results on this data set in the literature.

TABLE 10. Average predictive accuracies of particle filter-based learning method on the electricity pricing data with varying batch size, B.

# of features	1	2	3	4	5
$B = 2$	0.914 ± 0.020	0.918 ± 0.010	0.912 ± 0.016	0.904 ± 0.022	0.861 ± 0.193
$B = 4$	0.910 ± 0.021	0.911 ± 0.008	0.906 ± 0.009	0.900 ± 0.031	0.877 ± 0.121
$B = 6$	0.908 ± 0.010	0.905 ± 0.009	0.901 ± 0.009	0.898 ± 0.016	0.874 ± 0.117
$B = 8$	0.901 ± 0.010	0.899 ± 0.008	0.892 ± 0.011	0.887 ± 0.014	0.872 ± 0.086
$B = 10$	0.898 ± 0.009	0.891 ± 0.005	0.886 ± 0.012	0.883 ± 0.012	0.867 ± 0.082
$B = 15$	0.881 ± 0.009	0.877 ± 0.006	0.872 ± 0.011	0.869 ± 0.013	0.854 ± 0.079
$B = 20$	0.864 ± 0.013	0.863 ± 0.006	0.859 ± 0.008	0.854 ± 0.013	0.840 ± 0.074

The number of features, D (for $D = 1$ to 5), means that the first D variables in the list of *nswprice, nswdemand, transfer, vicprice*, and *vicdemand* are used.

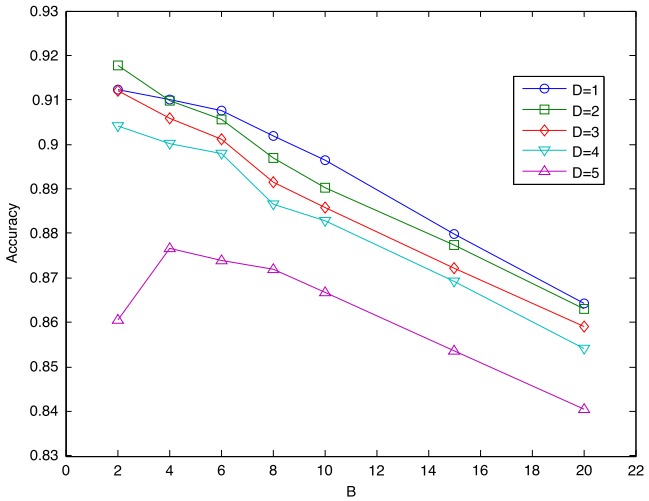

FIGURE 11. Predictive accuracy of the ELEC data set as a function of batch size B. The markers denote the number of features, D (for $D = 1$ to 5), which means that the first D variables in the list of *nswprice, nswdemand, transfer, vicprice*, and *vicdemand* are used. [Color figure can be viewed at wileyonlinelibrary.com]

TABLE 11. Recorded training times in milliseconds for PF-LR in Matlab with an AMD Phenom II X4 995 Processor at 3.21GHz. T is the total number of batches in the training set.

Training time (ms)	PF-LR	Number of Instances	T
SEA	404.7 ± 7.3	40,000	800
CIRCLES	655.3 ± 7.3	40,000	800
MANY	25.4 ± 2.8	2,000	40
HYPERPLANE	875.3 ± 106.6	40,000	800
ELEC (B=10)	1,907 ± 54.3	45,312	4,531
WDBC (B=100)	17.1 ± 9.3	569	5
SS (B = 50)	2,510 ± 28	245,057	4,901

5.9. Real Data Analysis: Breast Cancer and Skin Segmentation

The WDBC data set is a high-dimensional static data set containing 30 features. Based on the analysis on HYPERPLANE, a batch size of $B = 100$ is chosen. Because the data set is static, drift detection is not used. The best performing algorithms are PF-LR and DWM-NB, with DWM-NB slightly outperforming PF-LR.

In feature space, the classes in the SS data set are distributed in a way that cannot be well separated by a linear decision boundary. Therefore, it is expected that LR would be outperformed by decision trees and clustering algorithms such as kNN. Indeed, LB-HT achieved the best predictive accuracy at 99.5%, followed by batch-kNN at 97.9%, then PF-LR at 94.5%.

The results for both data sets with prequential evaluation are shown in Table 7.

5.10. Run Time and Memory Requirement

Finally, we report the recorded the training time for PF-LR in Table 11.[10] Note that ELEC, SEA, and CIRCLES have similar numbers of both instances and dimensions. However, the number of batches in ELEC is five times larger than those for SEA and CIRCLES. In Table 11, we see that the training time for ELEC is roughly five times than those for SEA and CIRCLES. This linear dependence on the total batch number is manifest between other combinations of data sets. In our MATLAB (MathWorks, Natick, MA) implementation of PF-LR, we vectorized all calculations within a batch, keeping only the loop over different batches. The loop over the batches gives the running time a linear dependence on T, which is the total number of batches in the training set. Because MATLAB is optimized for operations involving vectors and matrices, the dependence of the running time on the batch size B is suppressed.

By comparing between ELEC with SEA and CIRCLES where the number of instances are roughly the same, we see the training time for ELEC is roughly five times those of SEA and CIRCLES. At the same time, the number of batches in ELEC is five times larger than those for SEA and CIRCLES. Finally, we note that the memory requirement is trivial for PF-LR because only the data in the batch being analyzed are stored.

[10] We implemented PF-LR in Matlab. These tests were performed with the AMD Phenom II X4 995 Processor at 3.21GHz.

6. DISCUSSION AND CONCLUSION

We used TA for particle selection instead of the likelihood function as carried out in conventional particle filtering and developed PF-LR. This article demonstrated that PF-LR outperforms other state-of-the-art algorithms in terms of drift recovery and accuracy for the data sets analyzed. Synthetic data sets were used in the evaluation to develop an understanding of PF-LR—the SEA data set was used to demonstrate the drift tolerance of PF-LR. CIRCLES was used to show its applicability to imbalanced and noisy data. Its reaction to drifts was further tested on a data set with frequent drifts. Then we applied PF-LR to a real data set—the electricity pricing data—to show that it is able to obtain the highest accuracy reported in the literature recorded by Z'liobaite (2013). We found that the PF-LR's run time is linear in the number of batches processed when implemented in MATLAB.

We showed that PF-LR outperforms APF and RegAPF. We also illustrated that using TA as a quality measure is better suited for classification tasks than using the likelihood function as in conventional PF. Comparisons were also made between using a fixed size ensemble and using all best-performing particles for each batch. We found that using all best-performing particles outperforms the method of fixed ensemble sizes in data sets with frequent drifts or class imbalance.

The results clearly show that PF-LR outperforms DWM-NB, LB-HT, and VW. It is important to note that DWM-NB, LB-HT, and VW use very different learning mechanisms. In particular, DWM-NB and VW are online algorithm that forget previous instances based on the classifier's performances. DWM-NB handles drifts by adaptively creating and removing classifiers. VW uses recent data to learn by gradient descent with an adaptive learning rate, while LB-HT is a decision tree method using adaptive sliding windows for drift detection. Furthermore, we have shown that PF-LR can be adapted to be used with other regression models where a nonlinear decision boundary was used to analyze the CIRCLES data set. This flexibility is not exhibited by algorithms such as Naive Bayes and decision trees. The fact that PF-LR outperforms algorithms with different learning mechanisms consistently shows its potential in mining evolving streaming data. PF-LR also outperforms batch-SVM and batch-kNN on all data sets tested, with the exception that batch-kNN outperforms PF-LR on the SS data set as the decision boundary is required to be nonlinear.

Granted, PF-based algorithms handle concept drifts exceptionally well. They are not without limitations. As PF is an MC method, it suffers from the curse of dimensionality in the same way as other MC techniques. An analysis with high-dimensional data set was performed in Section 5.7. There, we see that PF-LR with a DR method can perform well up to 30 dimensions.

Perhaps another attractive feature of PF is that it is very intuitive and theoretically motivated. The manner how each input parameter of PR-LF affects the performance is manifest and they can be adjusted, or even be made adaptive, to fit the analysis at hand with ease. In particular, in situations where accuracy is preferred over computation time, the particle number M can be made larger. Data sets showing large and rapid drifts can be handled by either choosing a proposal distribution with wide spread to improve drift recovery or decreasing the batch size. For more static circumstances, a proposal function with narrow spread as well as a large batch size can be chosen to maximize the precision and accuracy, respectively.

On choosing the batch size, the batch size should be selected to ensure that our algorithm is sensitive to concept drift and robust to noise to reduce false positives. Because PF-LR relies on the chosen statistical model, a default batch size is 50, which is the more or less agreed sample size required to ensure enough power to test the significance of parameters

in statistical models. The common recommendation for the Central Limit Theorem to be applicable is also around 50. Thus, we set the batch size to be 50 when the number of features is small or moderate (Breiman 1992; Van der Vaart 2000). When the dimensionality is high, then the sample size should increase in principle if all features are equally important. However, in practice, many features might not be relevant, or they could be combined by using DR techniques. The batch size could be kept at around 50 if an effective reduction of dimensions can be achieved.

As we only implemented the PF algorithm with the most basic features, a naive choice of the proposal function with parameters σ_i and a fixed batch size, the possibilities to boost the performance of PF-LR are many. For example, a drift detection algorithm could be used and update the spread parameters σ_i of the proposal function accordingly—using a narrow spread for static situations and a wide spread immediately after drifts. Also, a large change in the learned parameter usually implies a drift is occurring. Therefore, an alternative method would be updating σ_i in a way that depends on the magnitude in which the parameters have changed from the last batch. We leave the opportunities to improve PF-LR for future research.

REFERENCES

ADAE, I., and M. BERTHOLD. 2013. Eve: a framework for event detection. Evolving Systems, **4**: 61–70.

AGGARWAL, C. C. 2007. Data Streams: Models and Algorithms. Springer: New York, p. 41.

ALBERG, D., M. LAST, and A. KANDEL. 2012. Knowledge discovery in data streams with regression tree methods. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, **2**: 69–78.

ANG, H., V. GOPALKRISHNAN, I. Z'LIOBAITE, M. PECHENIZKIY, and S. HOI. 2012. Predictive handling of asynchronous concept drifts in distributed environments. IEEE Transactions on Knowledge and Data Engineering, **25**(10): 2343–2355.

BACHE, K., and M. LICHMAN. 2013. UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences. http://archive.ics.uci.edu/ml.

BREIMAN, L. 1992. Probability, Classics in Applied Mathematics, Vol. 7. SIAM: Philadelphia, PA.

BREIMAN, L. E., and T. PETRIE. 1966. Statistical inference for probabilistic functions of finite state Markov chains. The Annals of Mathematical Statistics, **36**(6): 1554–1563.

BIFET, A., G. HOLMES, and B. PFAHRINGER. 2010a. Leveraging bagging for evolving data streams. *In* Proceedings of the 2010 European Conference on Machine Learning and Knowledge Discovery in Databases: Part I, ECML PKDD–10, Barcelona, Spain, pp. 135–150.

BIFET, A., G. HOLMES, R. KIRKBY, and B. PFAHRINGER. 2010b. MOA: Massive online analysis. Journal of Machine Learning Research, **11**: 1601–1604.

BOUCHACHIA, A. 2011a. Fuzzy classification in dynamic environments. Soft Computing, **15**(5): 1009–1022.

BOUCHACHIA, A. 2011b. Incremental learning with multi-level adaptation. Neurocomputing, **74**(11): 1785–1799.

BREIMAN, L. 1996. Bagging predictors. Machine Learning, **24**: 123–140.

BREIMAN, L. 1998. Arcing classifiers. The Annals of Statistics, **26**(3): 801–849.

CASARIN, L., and J. MARIN. 2009. Online data processing: comparison of Bayesian regularized particle filters. Electronic Journal of Statistics, **3**: 239–258.

CHOPIN, N. 2002. A sequential particle filter method for static models. Biometrika, **89**(3): 539–552.

DOMINGOS, P., and G. HULTEN. 2000. Mining high-speed data streams. *In* Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM Press: New York, pp. 71–80.

DOUCET, A., and A. M. JOHANSEN. 2011. A tutorial on particle filtering and smoothing: fifteen years later. *In* Handbook of Nonlinear Filtering. Oxford University Press: Oxford, UK, pp. 656–704.

FREUND, Y., and R. E. SCHAPIRE. 1996. Experiments with a new boosting algorithm. *In* Proceedings of the Thirteenth International Conference on Machine Learning. Morgan Kaufmann: San Francisco, CA, pp. 148–156.

FREUND, Y., and R. E. SCHAPIRE. 1999. A short introduction to boosting. Journal of Japanese Society for Artificial Intelligence, **14**(5): 771–780.

GAMA, J., and P. KOSINA. 2011. Learning about the learning process. *In* Proceedings of the 10th International Conference on Advances in Intelligent Data Analysis. IDA. Springer: Berlin Heidelberg, pp. 162–172.

GAMA, J., R. SEBASTIAO, and P. RODRIGUES. 2013. On evaluating stream learning algorithms. Machine Learning, **90**(3): 317–346.

GAMA, J., A. BIFET, A. BOUCHACHIA, M. PENCHENIZKIY, and I. Z'LIOBAITE. 2014. A survey on concept drift adaptation. ACM Computing Surveys, **46**(4): 44.

GEWEKE, J. 1989. Bayesian inference in econometric models using Monte Carlo integration. Econometrica, **57**: 1317–1339.

GOMES, J. B., E. M. GUIZ, and P. A. C. SOUSA. 2011. Learning recurring concepts from data streams with a context-aware ensemble. *In* Proceedings of the ACM Symposium on Applied Computing. SAC: New York, pp. 994–999.

GREST, D., and V. KRUEGER. 2007. Gradient-enhanced particle filter for vision-based motion capture. human motion understanding, modeling, capture and animation. Lecture Notes in Computer Science, **4814**: 28–41.

GUSTAFSSON, F., F. GUNNARSSON, N. BERGMAN, U. FORSSELL, J. JANSSON, R. KARLSSON, and P. J. NORDLUND. 2002. Particle filters for positioning, navigation and tracking. IEEE Transactions on Signal Processing, **50**(2): 425–437.

HARRIES, M. 2007. Splice-2 comparative evaluation: electricity pricing. *In* Technical report, University of New South Wales.

HALL, M., E. FRANK, G. HOLMES, B. PFAHRINGER, P. REUTEMANN, and I. H. WITTEN. 2007. The WEKA data mining software: an update. SIGKDD Explorations, **11**(1): 10–18.

HEDIBERT, F. L., and R. S. TSAY. 2011. Particle filters and Bayesian inference in financial econometrics. Journal of Forecasting, **30**: 168–209.

HOEFFDING, W. 1963. Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association, **58**(301): 13–30.

HULTEN, G., L. SPENCER, and P. DOMINGOS. 2001. Mining time-changing data streams. *In* Proceedings of the 7th ACM SIGKDD, New York, pp. 97–106.

IKONOMOVSKA, E., J. GAMA, and S. DZEROSKI. 2011. Learning model trees from evolving data streams. Data Mining Knowledge Discovery, **23**(1): 128–168.

JI, C., Y. ZHANG, M. TONG, and S. YAN. 2008. Parallel problem solving from nature – PPSN X. Lecture Notes in Computer Science, **5199**: 909–918.

KADLEC, P., R. GRBIC, and B. GABRYS. 2011. Review of adaptation mechanisms for data-driven soft sensors. Computers & Chemical Engineering, **35**(1): 1–24.

KENNEDY, J., and R. EBERHART. 1995. Particle swarm optimization. *In* Proceedings of IEEE International Conference on Neural Networks IV, Perth, Australia, pp. 1942–1948.

KLAMARGIAS, A. D., K. E. PARSOPOULOS, P. D. ALEVIZOS, and M. N. VRAHATIS. 2008. Particle filtering with particle swarm optimization in systems with multiplicative noise. *In* Genetic and Evolutionary Computation Conference 2008, Atlanta, GA, pp. 57–62.

KOLTER, J. Z., and M. A. MALOOF. 2007. Dynamic weighted majority: an ensemble method for drifting concepts. Journal of Machine Learning Research, **8**: 2755–2790.

LANGFORD, J., L. LI, and A. STREHL. 2007. Vowpal Wabbit open source project. *In* Technical Report, Yahoo.

LITTLESTONE, N., and M. K. WARMUTH. 1994. The weighted majority algorithm. Information and Computation, **108**: 212–261.

LIU, J., and M. WEST. 2001. Combined parameter and state estimation in simulation based filtering, Sequential Monte Carlo Methods in Practice. Springer-Verlag: New York.

MALOOF, M. A. 2005. Concept drift. *In* Encyclopedia of Data Warehousing and Mining: 202–206. Available at: http://www.irma-international.org/chapter/concept-drift/10593/.

MINKU, L. L., A. P. WHITE, and X. YAO. 2010. The impact of diversity on on-line ensemble learning in the presence of concept drift. IEEE Transactions on Knowledge and Data Engineering, **22**(5): 730–742.

MINKU, L., and X. YAO. 2011. DDD: A new ensemble approach for dealing with concept drift. IEEE Transactions on Knowledge and Data Engineering, **24**(4): 619–633.

MORAL, P. D., A. DOUCET, and A. JASRA. 2006. Sequential Monte Carlo samplers. Journal of the Royal Statistical Society: Series B (Statistical Methodology), **68**(3): 411–436.

MORENO-TORRES, J. G., T. RAEDER, R. ALAIZ-RODRIGUEZ, N. V. CHAWLA, and F. HERRERA. 2012. A unifying view on dataset shift in classification. Pattern Recognition, **45**(1): 521–530.

MUSSO, C., N. OUDJANE, and F. LEGLAND. 2001. Improving regularized particle filters, Sequential Monte Carlo methods in Practice. Springer-Verlag: New York.

NISHIDA, K., and K. YAMAUCHI. 2007. Detecting concept drift using statistical testing. *In* Discovery Science. Springer: Heidelberg Berlin, pp. 264–269.

PITT, M., and N. SHEPHARD. 1999. Filtering via simulation: auxiliary particle filters. Journal of the American Statistical Association, **94**(446): 590–599.

ROKACH, L. 2010. Ensemble-based classifiers. Artificial Intelligence Review, **33**(1-2): 1–39.

SCHLIMMER, J. C., and R. H. GRANGER. 1986. Beyond incremental processing: tracking concept drift. *In* Proceedings of the Fifth National Conference on Artificial Intelligence, Philadelphia, PA, pp. 502–507.

SCHLIMMER, J. C., and R. H. GRANGER. 1986. Incremental learning from noisy data. Machine Learning, **1**(3): 317–354.

SHALEV-SHWARTZ, S., Y. SINGER, and N. SREBRO. 2007. Pegasos: primal estimated sub-gradient solver for SVM. *In* 24th International Conference on Machine Learning, Corvalis, OR, pp. 807–814.

SOTOUDEH, D., and A. AN. 2010. Partial drift detection using a rule induction framework. *In* Proceedings of the 19th ACM International Conference on Information and Knowledge Management. ACM: New York, pp. 769–778.

STREET, W., and Y. KIM. 2001. A streaming ensemble algorithm (SEA) for large-scale classification. *In* Proceedings of the 7th ACM SIGKDD, New York, pp. 377–382.

TSYMBAL, A. 2004. The Problem of Concept Drift: Definitions and Related Work. Computer Science Department, Trinity College, Dublin, Ireland.

VAN DER VAART, A. W. 2000. Asymptotic Statistic. Cambridge University Press: New York.

WANG, H., W. FAN, P. YU, and J. HAN. 2003. Mining concept-drifting data streams using ensemble classifiers. ACM SIGKDD: 226–235. Available at: http://dl.acm.org/citation.cfm?id=956778.

WIDMER, G., and M. KUBAT. 1996. Learning in the presence of concept drift and hidden contexts. Machine Learning, **23**(1): 69–101.

WOLPERT, D. H. 1992. Stacked generalization. Neural Networks, **5**(2): 241–259.

YAO, R., Q. SHI, C. SHEN, Y. ZHANG, and A. VAN DEN HENGEL. 2012. Robust tracking with weighted online structured learning. *In* Proceedings of the 12th European Conference on Computer Vision, ECCV, Florence, Italy, pp. 158–172.

ZHAO, P., S. HOI, R. JIN, and T. YANG. 2011. Online AUC maximization. *In* Proceedings of the 28th Internarional Conference on Machine Learning, Bellevue, WA, pp. 233–240.

Z'LIOBAITE, I., A. BIFET, M. M. GABER, B. GABRYS, J. GAMA, L. L. MINKU, and K. MUSIAL. 2012. Next challenges for adaptive learning systems. SIGKDD Explorations, **14**(1): 48–55.

Z'LIOBAITE, I. 2013. How good is the electricity benchmark for evaluating concept drift adaptation. arXiv:1301.3524v1.

APPENDIX A

A.1. Regularized PF-LR

In regularized particle filtering, the parameters for the conditional proposal function, Σ, are regularized by a conditional hyperprior distribution with hyperparameters a and h, where a controls the mean of the hyperprior and h controls its variance. We choose the covariance matrix Σ to be a diagonal matrix with elements σ_i, where $i \in \{1, \ldots, D\}$ and D the dimensionality of the regression coefficients $\boldsymbol{\beta}$. We choose the hyperprior on σ_i to be a log-normal distribution. At batch n, the i-th component of the m-th particle is denoted by the tuple $\left(\beta_{im}^{(n)}, \sigma_{im}^{(n)}\right)$. Let $\theta_{im}^{(n)} = \log \sigma_{im}^{(n)}$ and $\bar{\theta}_i^{(n)}$ be its mean over the particle index, and the generation scheme is

$$\theta_{im}^{(n)} \sim \mathcal{N}\left(a\theta_{im}^{(n-1)} + (1-a)\bar{\theta}_i^{(n-1)}, h^2\right)$$
$$\beta_{im}^{(n)} \sim \mathcal{N}\left(\beta_{im}^{(n-1)}, \left(\sigma_{im}^{(n)}\right)^2\right).$$

Algorithm 4 shows RegPF-LR. It is structurally similar to that of PF-LR, other than the extra steps involved in the generation of the proposal parameters σ_{im}. The algorithm of the regAPF is described in Casarin and Marin (2009).

Algorithm 4. Classification algorithm with a regularized PF-LR

Inputs:
$\{\mathbf{x}, y\}^{(n)}$, the training data in batch n
B, number of instances in a batch
M, number of particles
$\boldsymbol{\theta}_0^{(n-1)}$, the combined logarithm of the parameters of the proposal function
$\boldsymbol{\beta}_0^{(n-1)}$, the combined parameters from the previous batch

Outputs:
$\left(\boldsymbol{\beta}_0^{(n)}, \boldsymbol{\theta}_0^{(n)}\right)$, the combined particle for batch n
$\{\boldsymbol{\beta}^{*(n)}\}$, the trained regression parameters for batch n
$k_m^{(n)}$, the auxiliary indices for batch n

Step 1: For $m \leftarrow 1 : M$

Step 1.1 : For $i \leftarrow 1 : D$,

Step 1.1.1 : Calculate $\bar{\theta}_i \leftarrow$ mean of $\theta_{im}^{(n-1)}$
Step 1.1.2 : Generate $\theta_{im}^{(n)} \sim \mathcal{N}\left(a\theta_{i,k_m^{(n-1)}}^{(n-1)} - (1-a)\bar{\theta}_i, h^2\right)$
Step 1.1.3 : Calculate $\sigma_{im}^{(n)} \leftarrow \exp\left(\theta_{im}^{(n)}\right)$
Step 1.1.4 : Generate $\beta_{im}^{(n)} \sim \mathcal{N}\left(\beta_{i,k_m^{(n-1)}}^{(n-1)}, \sigma_{im}^{(n)}\right)$

Step 1.2 : Calculate training accuracy $A_m \leftarrow A_m[f(\boldsymbol{\beta}_m^{(n)}, \{\mathbf{x}, y\}^{(n)}), B]$ from Algorithm 2.

Step 2: Calculate TA using the classifier from last batch
$A_0 \leftarrow A_0[f(\boldsymbol{\beta}_0^{(n-1)}, \{\mathbf{x}, y\}^{(n)}), B]$

Step 3: Uniformly generate a list of M indices $k_m^{(n)} \in K^{(n)}$ for each particle, where $K^{(n)} = \{k^{(n)} | A_{k^{(n)}} = \max\{A_j\}, j \in \{0, \ldots, M\}\}$

Step 4: Combine the classifiers
$\boldsymbol{\beta}_0^{(n)} \leftarrow \bar{\boldsymbol{\beta}}$, where $\bar{\boldsymbol{\beta}}$ is the mean of $\boldsymbol{\beta}_{k^{(n)}}^{(n)}$ over all $k^{(n)}$

Step 5: $\boldsymbol{\theta}_0^{(n)} \leftarrow \bar{\boldsymbol{\theta}}$, where $\bar{\boldsymbol{\theta}}$ is the mean of $\boldsymbol{\theta}_{k^{(n)}}^{(n)}$ over all $k^{(n)}$

Step 6: $\{\boldsymbol{\beta}^{*(n)}\} \leftarrow$ the ensemble of $\boldsymbol{\beta}^{(n)}$ that gives the highest accuracy

Step 7: return $\left(\boldsymbol{\beta}_0^{(n)}, \boldsymbol{\theta}_0^{(n)}\right)$, $\{\boldsymbol{\beta}^{*(n)}\}$, and $k_m^{(n)}$