Mining Evolving Data Streams with Particle Filters

Ricky Fok, Aijun An and Xiaogang Wang

Technical Report CSE-2013-11

October 9 2013

Department of Computer Science and Engineering
4700 Keele Street, Toronto, Ontario M3J 1P3 Canada

Mining Evolving Data Streams with Particle Filters

Ricky Fok RICKY@CSE.YORKU.CA
Department of Computer Science and Engineering
York University
Toronto, ON M3J 1P3, Canada

Aijun An AAN@CSE.YORKU.CA
Department of Computer Science and Engineering
York University
Toronto, ON M3J 1P3, Canada

Xiaogang Wang STEVENW@MATHSTAT.YORKU.CA
Department of Mathematics and Statistics
York University
Toronto, ON M3J 1P3, Canada

Abstract

We propose a modified particle filter based learning method and apply it to learning logistic regression models from evolving data streams. The resampling step of the particle filter is replaced by choosing a set of regression coefficients (i.e., particles) that maximizes the training accuracy. The method inherently handles concept drifts in the data stream and is able to learn an ensemble of logistic regression models with particle filtering. We evaluate the method on both synthetic and real data sets and find that our method outperforms other state-of-the-art algorithms on the data sets tested.

Keywords: Concept Drift, Ensemble Methods, Stochastic Optimization, Data Stream Mining

1. Introduction

In this paper, 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 (HMM) (Baum and Petrie, 1966) and employ particle filtering (Doucet and Johansen, 2009), a sequential Monte Carlo method developed for this exact purpose, to discover the most up-do-date concepts. In terms of data mining, particle filtering is an ensemble method that generates a fixed number of classifiers (called particles) at each time step from the most important classifiers in the previous step[1]. The resulting algorithm, logistic regression with particle filtering (PF-LR), is a modified version of conventional particle filtering in order to reduce the extent of overfitting.

The particle filter is an algorithm with applications in object tracking (Gustafsson et. al., 2002). Particles at time n are generated from a prior distribution conditioned on the values of previous hidden variables. The prior is usually assumed to be a Gaussian distribution.

1. This is the auxiliary particle filter (Pitt and Shephard, 1999) which usually gives better performance than the original particle filter in Doucet and Johansen (2009).

The likelihood function of the generated particles are calculated with the observed data to be used as weights of the particles. Subsequent particles are generated near the ones with the highest values of the likelihood function. This suggests the use of weighted majority voting (Littlestone and Warmuth, 1994) in order to obtain a classifier from the ensemble of particles. We apply this approach to develop an ensemble method using particle filtering to binary classification tasks. Also, we consider a modification on the particle filter where the performances of particles are measured by the training accuracy instead of the likelihood function used in the conventional particle filter.

The PF-LR algorithm developed is very similar to the auxiliary particle filter (Pitt and Shephard, 1999). Two variants of auxiliary particle filters are evaluated on different data sets. We found that the classification algorithm using the regularized auxiliary particle filter does not outperform the one with the auxiliary particle filter. This is discussed in appendix A. Also, we evaluate the algorithms on the commonly used synthetic benchmarks with concept drifts, the SEA (Street and Kim, 2001) and CIRCLES (Nishida and Yamauchi, 2007). We also tested the algorithms on the real electricity pricing data (Harries, 1999). Our results are contrasted with other existing algorithms.

The contributions of this paper are as follows

- We model concept drifts with a Hidden Markov Model and present a drift tolerant algorithm, PF-LR, using particle filtering with linear complexity.
- This is the first attempt to use training accuracy as a criterion for particle selection. We show that this results in very little overfitting for logistic regression.
- We show that the modified particle filter is robust to noise when used to learn regression coefficients. Furthermore, the best performance is obtained by using training accuracy as the performance measure for each particle with majority voting to give the final prediction.
- We show that PF-LR outperforms Hoeffding Tree with Leveraging Bagging (Bifet et. al., 2010a) and Naive Bayes with Dynamic Weighted Majority (Kolter and Maloof, 2007).

The organization of this paper is as follows. In the next section we review related work in the literature. Section 3 gives the theoretical motivation that particle filtering is a strong candidate to handle concept drifts. The modification to the particle filter to alleviate overfitting is also discussed and its algorithm given. The comparison of its performance with other algorithms is given in section 4 where we illustrate the particle filter's ability to adapt to frequent drifts. The discussion and conclusions are given in section 5.

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 with data (Aggarwal, 2007), a phenomenon called concept drift. Particle filtering is a sequential Monte Carlo 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 has 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). 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

$$\bar{p} = \frac{\sum_i^M w_i p_i}{\sum_i^M w_i}, \tag{1}$$

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 due to 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, 1986; 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, 1986). 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, 1986) 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 Dynamic Weighted Majority (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, 1986) 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 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 are weighted by their classification performance. The predictions are combined by weighted majority voting. CVFDT (Hulten et. al., 2001) is a 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.

2.3. Particle Filtering

Particle filtering (Doucet and Johansen, 2009), or sequential Monte Carlo (SMC), has been extensively used in the inference of Hidden Markov Models (Baum and Petrie, 1966), where time series data are generated by a hidden Markov chain. Importance sampling (see, for example, 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 generate particles from particles with high probabilities in the last time step. This is the basis of the auxiliary particle filter (Pitt and Shephard, 1999).

Regularized particle filters are summarized in Casarin and Marin (2009), which employs 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 are 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 of 10000.

Particle filters can be optimized to improve their efficiency. The most natural approach is done 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 based optimization with particle filtering.

As 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 training accuracy 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 classifers 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 provide a theoretical justification to handle drifts with particle filtering. The modification to conventional particle filtering are presented. Lastly, we develop PF-LR, a classification algorithm using logistic regression with particle filtering.

3.1. Motivation

Particle filtering is a method of sequential Bayesian analysis with sequential Monte Carlo (SMC) on Hidden Markov Models (HMM). It has been shown to be effective in modelling dynamical models (Doucet and Johansen, 2009). In particular, for a time series of model parameters $\boldsymbol{\beta}^{(1:T)} := \{\boldsymbol{\beta}^{(1)}, \ldots, \boldsymbol{\beta}^{(T)}\}$, 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 is 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' Theorem, the posterior distribution of the model parameters can be written as

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

The series of product on the right hand side is highly suggestive that the inference can be done 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

$$q(\boldsymbol{\beta}^{(1:T)}) = q_1(\boldsymbol{\beta}^{(1)}) \prod_{n=2}^{T} q_n(\boldsymbol{\beta}^{(n)}|\boldsymbol{\beta}^{(n-1)}). \tag{3}$$

Dividing equation (2) by equation (3) gives the importance weights

$$w^{(1:T)} = w^{(1)} \prod_{n=2}^{T} w^{(n|n-1)}, \tag{4}$$

where the incremental weight is

$$w^{(n|n-1)} = \frac{p_n(\boldsymbol{\beta}^{(n)}|\mathbf{x}^{(n)}, \boldsymbol{\beta}^{(n-1)})}{q_n(\boldsymbol{\beta}^{(n)}|\boldsymbol{\beta}^{(n-1)})}. \tag{5}$$

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 $n-1$ 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 $p_n(\boldsymbol{\beta}^{(n)}|\mathbf{x}^{(n)}, \boldsymbol{\beta}^{(n-1)}) = g_n(\mathbf{x}^{(n)}|\boldsymbol{\beta}^{(n)}) f_n(\boldsymbol{\beta}^{(n)}|\boldsymbol{\beta}^{(n-1)})$ 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 $w_i^{(n)}$. This is to remove particles with low weights while duplicating the ones closer to the posterior maximum so as to reduce the numerical error at subsequent times. Pitt and Shephard (1999) proposed the auxiliary particle filter, a method to replace resampling without its deficits by generating an auxiliary variable for each particle - an index denoting the parents of the particles in the current

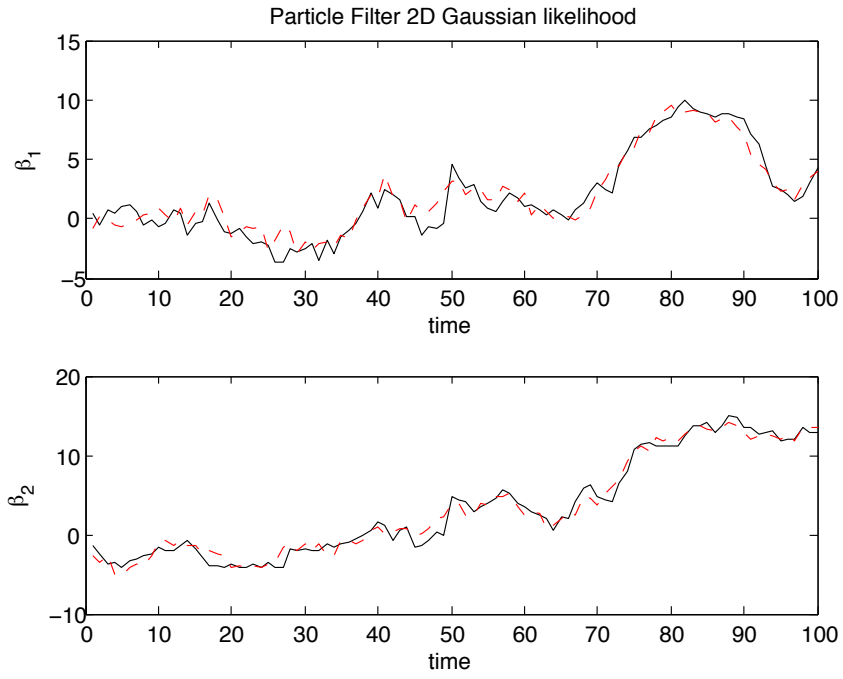

Figure 1: A typical result from the particle filtering. The true model parameters are denoted by the black solid line whereas the estimated posterior mean is in red dashed.

time step. The auxiliary variables are generated with a distribution proportional to each particle's weight 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. The evolution of the parameters $\boldsymbol{\beta}^{(n)}$ and $\mathbf{x}^{(n)}$ is according to two dimensional Gaussian distributions

$$\boldsymbol{\beta}^{(n)} \sim \mathcal{N}(\boldsymbol{\beta}^{(n-1)}, \mathbf{I}) \tag{6}$$

$$\mathbf{x}^{(n)} \sim \mathcal{N}(\boldsymbol{\beta}^{(n)}, \mathbf{I}) \tag{7}$$

We imposed two types of drifts - ones that are small and gradual throughout in accordance with equation (6), 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 the theoretical 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 as the underlying mechanisms that generate such data are often complex, dynamical and even

unknown. 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 a particle filter classification algorithm.

3.2. Features of a Particle Filter Classification Algorithm

We present the essential features required by the particle filter classification algorithm before turning to the discussion on the algorithm. For the purposes of constructing a fast and accurate algorithm, we choose to learn regression models with particle filtering. To this end we require the algorithm to possess the following properties: 1) choosing the model parameters that gives maximum training accuracy 2) batch processing, 3) discarding all previously processed data, 4) no drift detection and 5) a statistically monotonic increasing predictive accuracy in static situations. In this discussion we assume that the concepts are constantly drifting. Each of the features are discussed below.

In practice, it is important to realize that maximizing the likelihood does not imply maximum predictive accuracy due to the presence of noise. 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 it gives rather poor results due to overfitting. 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 very little overfitting for logistic regression. An intuitive argument for this is given in appendix B.

Since the particle filter classification algorithm 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. 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 dimensional reduction 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 because we assume the concepts are constantly changing and there is no guarantee that accumulating data would lead to a better estimate of the parameters.

The particle filter is extremely adaptive to concept drifts as seen in the previous sections. Therefore 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 paper already outperforms other algorithms in classification accuracy and competitive in complexity. In appendix C, we implemented a simple drift detection algorithm along with a dimensional reduction procedure and show that it improves the algorithm's performance in high dimensional cases.

As we do not process previous data, it is important to ensure that the learned parameters converge to the truth. 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 for all times. 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 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 complexity, 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 4.

3.3. Particle Filter Classification Algorithm (PR-LR)

The algorithm is remarkably simple. It 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) are randomly drawn from a chosen proposal distribution conditioned on previous particles with the largest training accuracy. 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 $\boldsymbol{\beta}_m^{(n)}$. Then the parameter estimates at the current batch $\boldsymbol{\beta}^{*(n)}$ is obtained by a majority voting from all particles that give the highest training accuracy. The rest of the ensemble is discarded. This process is to be repeated for subsequent batches. Also, we choose logistic regression as an example and refers to 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

$$f(\eta_a) = \frac{1}{1 + e^{\eta_a(\boldsymbol{\beta}^{(n)}, \mathbf{x}_a^{(n)})}}. \tag{8}$$

We use the following sign convention

$$\eta_a(\boldsymbol{\beta}^{(n)}, \mathbf{x}_a^{(n)}) = \boldsymbol{\beta}^{(n)} \cdot \begin{pmatrix} -1 \\ \mathbf{x}_a^{(n)} \end{pmatrix}. \tag{9}$$

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 Guassian 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 their predicted class labels with the true ones. Prediction of class labels is done by checking whether the value of the logistic function in equation (8) is larger than 0.5 for each instance, i.e. $f(\boldsymbol{\beta}^{(n)}, \{\mathbf{x}, y\}^{(n)}) > 0.5$, along with the previously learned classifier from the last batch. 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[2]. The purpose of this step is to assign the parents of $\boldsymbol{\beta}_m^{(n+1)}$ to be $\boldsymbol{\beta}_{k_m^{(n)}}^{(n)}$. Then, majority voting is performed to combine these particles. The algorithm returns the combined classifier and the auxiliary indices. The inputs of PF-LR are the batch size B, parameters for the proposal (Gaussian) distribution Σ, the particle number M and the data. The rest of the input shown in algorithm 1 are obtained from time $n-1$. For clarity, 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 $\beta_{im}^{(n)}$. As 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 $\mathcal{N}(\beta_{im}^{(n)}, \sigma_i)$ one by one.

Here we discuss the impact 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 a more accurate estimate of the regression parameters. However, in the presence of concept drifts, model 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 true model. 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 due to 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 since it gives a higher precision. In contrast, a larger spread is favorable when the data contains large or rapid drifts. Since 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 each batch. Generally, a larger M increases the precision of parameter estimation and improves drift recovery, but in the expense of computational time. Finally, since the algorithm does not contain any adaptive elements, its complexity is linear. For a total of T batches with B instances each batch and M particles, the complexity is of $O(MTB)$. Experimental results of the complexity is given in the next section.

4. Analysis

In this section we estimate the predictive accuracy of PF-LR on different data sets and compare against the performances of Hoeffding Tree (Domingos and Hulten, 2000) with Leverage Bagging (LB-HT[3]) (Bifet et. al., 2010a) and Naive Bayes with Dynamic Weighted

2. As the batch size is typically smaller than the particle number, there will typically be more than one particle that gives the highest training accuracy.
3. 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.

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

Input:
$\{\mathbf{x}, y\}^{(n)}$, the training data in batch n
B, the number of instances in a batch
M, the number of particles
σ_i, the parameters of the proposal function
$\boldsymbol{\beta}^{*(n-1)}$, the learned parameters from the previous batch
$k_m^{(n-1)}$, the indices where the m-th particle will be generated from
Output:
$\boldsymbol{\beta}^{*(n)}$, the trained regression parameters for batch n
$k_m^{(n)}$, the indices where the m-th particle will be generated from in the next batch

for $m \leftarrow 1$ **to** M **do**
  for $i \leftarrow 1$ **to** D **do**
    Generate $\beta_{im}^{(n)} \sim \mathcal{N}(\beta_{i,k_m^{(n-1)}}^{(n-1)}, \sigma_i)$
  end
  Calculate training accuracy $A_m \leftarrow A_m[f(\boldsymbol{\beta}_m^{(n)}, \{\mathbf{x}, y\}^{(n)}), B]$
end

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

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

Combine the classifiers with majority voting
$\boldsymbol{\beta}^{*(n)} \leftarrow \bar{\boldsymbol{\beta}}$, where $\bar{\boldsymbol{\beta}}$ is the mean of $\boldsymbol{\beta}_{k^{(n)}}^{(n)}$ over all $k^{(n)}$

return $\boldsymbol{\beta}^{*(n)}$ and $k_m^{(n)}$

Algorithm 2: Calculating accuracy from the training set

Input: $f(\cdot), \{\mathbf{x}, y\}^{(n)}, \boldsymbol{\beta}_m$, B
Output: A_m, the accuracry $\in [0, 1]$

$A'_m \leftarrow 0$
for $a \leftarrow 1$ **to** B **do**
  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**
    $A'_m \leftarrow A'_m + 1$
  end
end
return A'_m / B

Majority (DWM-NB[4]) (Kolter and Maloof, 2007). The difference between DWM-NB and LB-HT is that DWM-NB is an online algorithm without drift detection. Whereas LB-HT processes data batch by batch and employs adaptive sliding windows as a drift detection mechanism. 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.

We choose two commonly used synthetic data sets, SEA (Street and Kim, 2001) and CIRCLES (Nishida and Yamauchi, 2007), as benchmarks to demonstrate the drift and noise tolerance, respectively. We also generate a three dimensional data set with twenty concepts, with drifts occurring at every other batch. In the next section we apply PF-LR to the electricity pricing data set (Harries, 1999) to illustrate the applicability of PF-LR in real situations.

All synthetic data sets used in this section are generated by the data generator from Minku et. al. (2010) and contain 10% class noise. The specifics of each data set will be given in the following subsections.

Care must be taken when comparing algorithms with different learning techniques. The DWM-NB is an online algorithm that processes one example at a time. Whereas PF-LR and LB-HT process a batch of 50 for the synthetic data sets. To estimate the predictive accuracy of DWM-NB, we test on 20 instances for each training instance, so that DWM-NB is tested on the same 1000 instances in a batch of 50 instances. The performances of these algorithms[5] are evaluated on 1000 testing examples for each batch. We compare the average predictive accuracy of DWM-NB over a batch to the those of PF-LR and LB-HT.

4.1. SEA

The SEA data set contains feature variables $\mathbf{x} \in [0, 10]^3$, with class labels $y = 1$ if $x_1 + x_2 < \mu$, where $\mu = \{8, 9, 7, 9.5\}$. Before processing, the feature variables are normalized to the range [0,1]. The average predictive accuracy over 50 runs with PF-LR, LB-HT and DWM-NB are shown in figure 2. It shows that PF-LR outperforms both DWM-NB and LB-HT. The overall predictive accuracies and the corresponding 95% confidence interval are $(98.1 \pm 0.1)\%$ for PF-LR, $(96.5 \pm 0.1)\%$ for DWM-NB, and 96.3% for LB-HT. Note that 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 as 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 very slow.

4.2. CIRCLES

This subsection shows that PF-LR has better error tolerance than DWM-NB and LB-HT using a highly imbalanced synthetic data set, CIRCLES. Naive Bayes accumulates each instance, constructs classifiers and predicts class labels of instances accordingly. In the situations where the class noise overwhelms the true class label, as in the case of an

4. The input parameters we used for DWM-NB are: $\beta = 0.5, \theta = 0.01, maxExpoerts = \infty$, and $period = 50$.
5. We used Weka (Hall et. al., 2009) to evaluate DWM-NB, and MOA (Bifet et. al., 2010b) to evaluate LB-HT

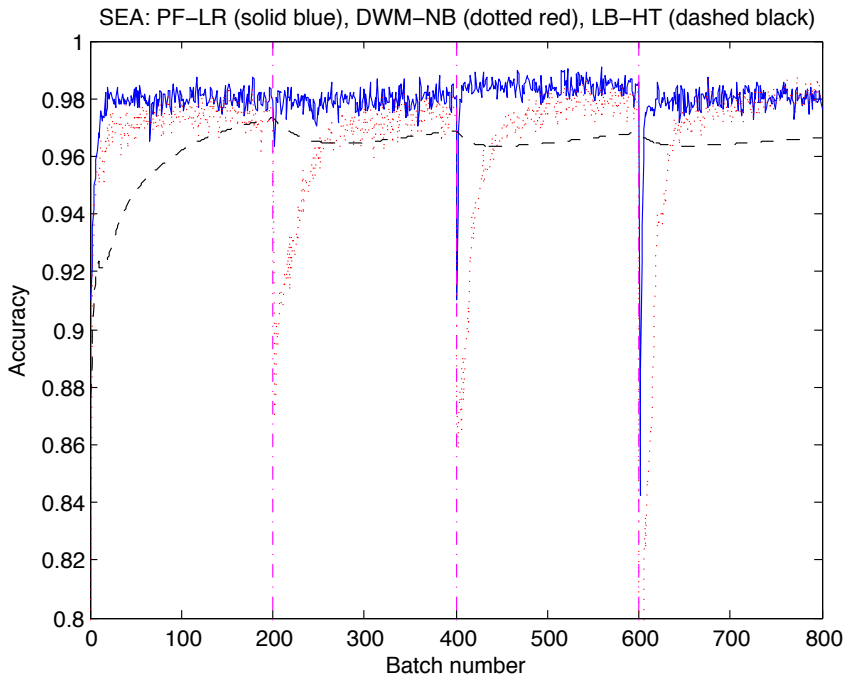

Figure 2: The averaged predictive accuracies of PF-LR (solid blue), LB-HT (dashed black) and DWM-NB (dotted red) for the SEA data set. The vertical magenta lines denote where the drifts occur.

imbalanced data set, it would be difficult for Naive Bayes to learn the model parameters. However, for algorithms such as PF-LR and LB-HT that analyze data batch by batch, the effects of class noise should be less profound. We show that this is indeed the case.

The feature space of CIRCLES is specified by $\mathbf{x} \in [0, 2]^2$. The data set CIRCLES has four concepts with increasing signal-to-noise ratio, defined by

$$\rho = \frac{\text{number of true class labels in the minority class}}{\text{number of false class labels in the minority class}}. \tag{10}$$

The signal-to-noise ratio and regression parameters for each concept are listed in table 1. Because the training sets are highly imbalanced, we test the algorithm on testing data that contains an equal number of the two classes for each concept, giving a baseline of 50%. We found that a linear decision boundary given by equation (8) gives a performance no better than random guessing. Therefore we used a circular decision boundary given by

$$\eta_a = (x_{1a} - \beta_1)^2 + (x_{2a} - \beta_2)^2 - \beta_0^2. \tag{11}$$

We also impose a periodic boundary condition on the sample space of CIRCLES by constraining the model parameters by the size of the sample size. In other words we take the modulo-2 of the particles immediately after their generation at each batch[6]. In other

6. The factor of 2 in moldulo corresponds to the length of each dimension in the feature space $[0, 2]^2$.

Table 1: The concepts and their signal-to-noise ratio in CIRCLES.

Concepts	β_0	β_1	β_2	ρ
1	0.15	0.2	0.5	0.17
2	0.2	0.4	0.5	0.30
3	0.25	0.6	0.5	0.51
4	0.3	0.8	0.5	0.67

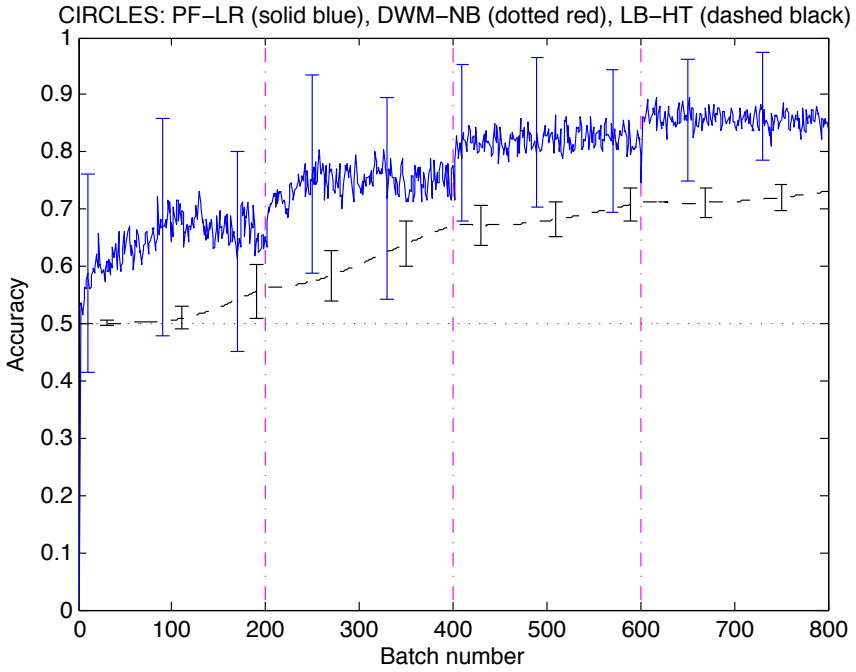

Figure 3: Comparisons of PF-LR (solid blue) with LB-HT (dashed black) and DWM-NB (dotted red) for the CIRCLES data set. The error bars are 68% confidence intervals. The vertical magenta lines denote where drifts occur.

words we apply the following line on algorithm 1, $\boldsymbol{\beta}^{(n)} \leftarrow \mathrm{mod}(\boldsymbol{\beta}^{(n)}, 2)$, after the generation step.

In figure 3 we compare the performances with the CIRCLES data set. The results show that DWM-NB is unable to give any sensible predictions. For PF-LR, when a more suitable regression model is chosen, i.e. regression with a circular decision boundary, PF-LR is able to learn the model parameters. Even though the performance of PF-LR comes with large uncertainty, it still outperforms LB-HT. The overall predictive accuracies and 95% confidence intervals are are $(79.1 \pm 1.1)\%$ for PF-LR, 50% for DWM-NB and $(63.1 \pm 0.2)\%$ for LB-HT. Even without taking the modulo, PF-LR still outperforms LB-HT and DWM-NB, with an overall predictive accuracy of $(70.6 \pm 1.3)\%$.

Table 2: The concepts in MANY.

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

4.3. A data set with many drifts

Another circumstance where PF-LR may perform better than other algorithms is where concepts are frequently changing. The fact that DWM-NB and LB-HT retain historical data and that it is more prone to noise implies the classifiers used by these algorithms may not be up-to-date after drifts. In contrast, PF-LR only relies on the examples in the current batch and its performance is expected to be higher in the case of constant drifts.

To compare the performance of PF-LR, a data set consisting of twenty concepts are generated. We will refer this data set as MANY. The twenty concepts are listed in table 2. The feature space as that same as that of the SEA data set, namely, $\mathbf{x} \in [0, 10]^3$. The instances are normalized to [0,1] before processing. The performance comparison is shown in figure 4. There, PF-LR almost always performs better than DWM-NB. Especially between batch numbers 25 to 35, where PF-LR is seen to be unaffected by drifts whereas the performance of DWN-NB is seen to drop rapidly immediately after each drift. The behavior of LB-HT is interesting. It is least affected by the drift at batch number 8, presumably due to its drift detection mechanism. However, it is unable to attain a high performance possibly due to contamination from out-of-date examples when the drifts are too small to be picked up by the drift detection. The overall predictive accuracies with this data set and the 95% confidence intervals are $(89.1 \pm 0.1)\%$ for PF-LR, $(85.6 \pm 0.1)\%$ for DWM-NB and $(86.4 \pm 0.1)\%$ for LB-HT.

4.4. Real Data Analysis: Electricity pricing

To illustrate the intuitions developed using the synthetic data sets, we apply PF-LR on the electricity pricing data set (Harries, 1999) which has been extensively studied by other authors (Z'liobaite, 2013). The data set contains a time series of 45,312 instances recorded at 30 minutes intervals. The class labels are UP and DOWN which indicate whether the price is higher than the moving average price over the last 24 hours. The original data set contains three variables denoting the time, $date, day$ and $period$. We ignore these variables and rely solely on the particle filter's ability to track the drifts as each batch is processed. The remaining 5 feature variables are $nswprice$, $nswdemand$, $transfer$, $vicprice$ and $vicdemand$. The values of each of these are divided by a factor of 1000 to bring them

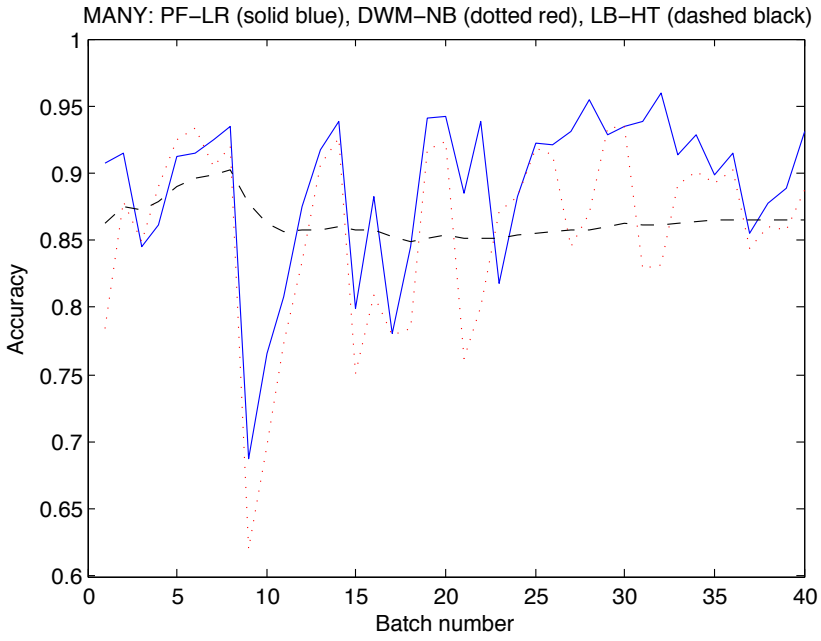

Figure 4: Comparisons of PF-LR (solid blue) with LB-HT (dashed black) and DWM-NB (dotted red) for the MANY data set. The vertical lines that denote the drift at every other batch is not shown for clarity.

to the range of order 1. Furthermore, we found PF-LR gives the best performance when using only one feature, *nswprice*. Including other features would slightly deteriorate the performance by up to a few percent. In this section we present the results using PF-LR and LB-HT including only the feature *nswprice* and compare the performance of DWM-NB given by Kolter and Maloof (2007).

Figure 6 shows the values of *nswprice*. Each estimated 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.

In contrast to the synthetic data set analyses earlier where independent testing data were available, the purpose of analyzing this time series is to predict whether the electricity price would rise above the 24 hour moving average. To do so, we estimate the regression parameters with the current batch and predict the class labels in the next batch. Then we repeat the process when the class labels for batch $n+1$ become known, a procedure referred to as prequential evaluation. The results for different batch sizes are shown in table 3 and figure 5. As expected, the performance of PF-LR improves as the batch size decreases, due to the fact that smaller batches give a better description of the current concept. However, when the batch size becomes too small, say $B < 6$, logistic regression becomes unreliable due to data sparseness. Consequently, the performance drops and its uncertainty grows. As

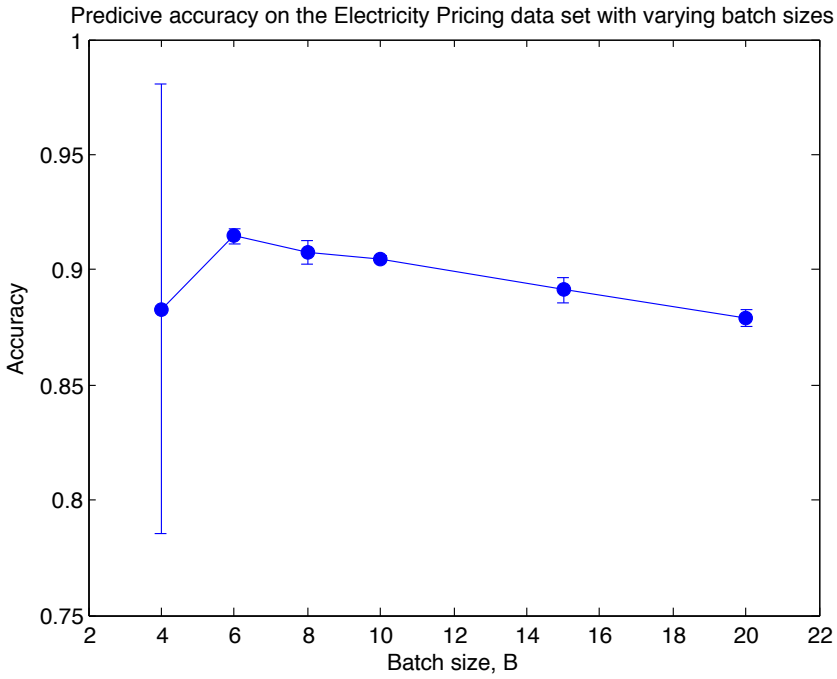

Figure 5: Predictive accuracy of PF-LR as a function of batch size B with $M = 100$. The error bars are 68% confidence intervals.

Table 3: Predictive accuracies of PF-LR on the electricity pricing data with varying batch size, B.

B	4	6	8
Accuracy	0.883 ± 0.195	0.915 ± 0.006	0.907 ± 0.01
B	10	15	20
Accuracy	0.905 ± 0.002	0.891 ± 0.011	0.879 ± 0.007

a sanity check, we plot the decision boundary obtained by PR-LR in figure 6. In analogy to figure 1, we see that PF-LR is able to track the movement of the decision boundary exceptionally well.

Finally, we report the recorded training times for PF-LR in table 4.[7] 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 5 times those for SEA and CIRCLES. At the same time, the number of batches in ELEC is 5 times larger 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 implementation of PF-LR we vectorized all

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

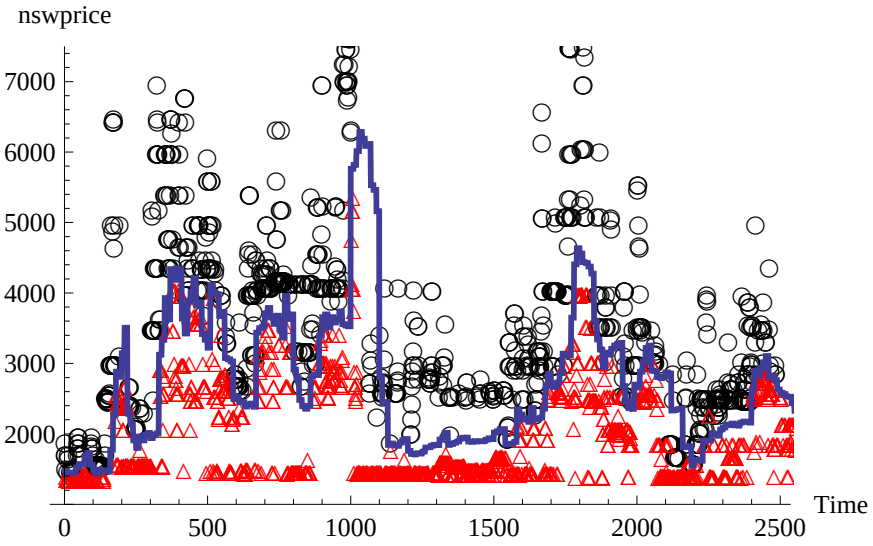

Figure 6: Values of the feature variable *nswprice* of the first 2500 instances. The markers denote the classes DOWN (red triangle) and UP (black circle). It is evident that drifts occurs throughout the data set. The solid line shows the decision boundary learned by PF-LR.

Table 4: Recorded training times in milliseconds for PF-LR in Matlab with an AMD Phenom II X4 995 Processor at 3.21GHz.

Training time (ms)	PF-LR	Number of Instances	T
SEA	404.7 ± 7.3	40000	800
CIRCLES	655.3 ± 7.3	40000	800
MANY	25.4 ± 2.8	2000	40
ELEC (B=10)	1907 ± 54.3	45312	4531

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. Because Matlab is optimized for operations involving vectors and matrices, the running time's dependence on the batch size B is suppressed.

5. Discussion and Conclusion

We used training accuracy for particle selection instead of the likelihood function as done in conventional particle filtering and developed PF-LR. This report demonstrated that PF-LR outperforms other state-of-the-art algorithms in terms of drift recovery and accuracy for the data sets tested. Synthetic data sets were analyzed 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 very noisy data. Its ability to adapt to drifts was further

Table 5: The overall predictive accuracies of PF-LR, DWM-NB, and LB-HT for the data sets tested. The highest value obtained for each data set is in bold.

Data set	PF-LR	DWM-NB	LB-HT
SEA	$(\mathbf{98.1} \pm 0.1)\%$	$(96.5 \pm 0.1)\%$	96.3%
CIRCLES	$(\mathbf{79.1} \pm 1.1)\%$	-	$(63.1 \pm 0.2)\%$
MANY	$(\mathbf{89.1} \pm 0.1)\%$	$(85.6 \pm 0.1)\%$	$(86.4 \pm 0.1)\%$
ELEC (B=10)	$(\mathbf{90.7} \pm 1.1)\%$	0.808%*	86.3%

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 (Z'liobaite, 2013). We found that the complexity is linear in the number of batches processed. We summarize our results[8] in table 5.

The results clearly show that PF-LR outperforms DWM-NB and LB-HT. It is important to note that DWM-NB and LB-HT uses very different learning mechanisms. In particular, DWM-NB is an online algorithm that forgets previous instances based on the classifier's performances. It handle drifts by adaptively creating and removing classifiers. While LB-HT is a decision tree method using adaptive sliding windows as a drift detection method. The fact that PF-LR outperforms algorithms with different learning mechanisms consistently shows its potential in mining evolving streaming data.

Granted, particle filter based algorithms handle concept drifts exceptionally well. They are not without limitations. As particle filtering is a Monte Carlo method, it suffers from the curse of dimensionality in the same way as other Monte Carlo techniques. An analysis with a ten dimensional synthetic data set is undertaken in appendix C. There, we see that PF-LR gives rather poor performances in high dimensions but it can be avoided by performing dimensional reduction methods.

Perhaps another attractive feature of particle filtering 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 the 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.

As we only implemented the particle filter 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 the

8. The result of DWM-NB for ELEC is taken from Kolter and Maloof (2007).

parameters have changed from the last batch. We leave the opportunities to improve PF-LR for future research.

Appendix A. Regularized Particle Filtering

In this appendix we compare the performance of an algorithm based on the regularized particle filter (Casarin and Marin, 2009), which we call RegPF-LR, with the other algorithms discussed in this paper. We found that RegPF-LR does not outperform PF-LR in all of the cases considered.

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 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 $(\beta_{im}^{(n)}, \sigma_{im}^{(n)})$. Let $\theta_{im}^{(n)} = \log \sigma_{im}^{(n)}$ and $\bar{\theta}_i^{(n)}$ be its mean over the particle index, the generation scheme is

$$\theta_{im}^{(n)} \sim \mathcal{N}(a\theta_{im}^{(n-1)} + (1-a)\bar{\theta}_i^{(n-1)}, h^2) \tag{12}$$

$$\beta_{im}^{(n)} \sim \mathcal{N}(\beta_{im}^{(n-1)}, (\sigma_{im}^{(n)})^2). \tag{13}$$

Algorithm 3 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}.

To initiate RegPF-LR, a preliminary run of 500 iterations was performed on the first batch. This is so that a suitable initial values of the logistic regression coefficients and the proposal parameters are chosen. Then we compare the performances of RegPF-LR to those of PF-LR DWM-NB and LB-HT in figures 7. We found that RegPF-LR performs marginally better than DWM-NB and LB-HT in the SEA, CIRCLES and MANY data sets. However, PF-LR outperforms RegPF-LR in all the data sets considered. In CIRCLES, even though PF-LR and RegPF-LR achieve similar predictive accuracies, the uncertainty associated with RegPF-LR is consistently larger than that of PF-LR. In MANY, PF-LR constantly outperforms RegPF-LR. In MANY, even though the performance of PF-LR can be worse than that of RegPF-LR at times, the overall accuracy of PF-LR, with 89.1%, is higher than that of RegPF-LR at 88.1%.

Algorithm 3: Classification algorithm with a regularized particle filter

Input:
$\{\mathbf{x}, y\}^{(n)}$, the training data in batch n
B, number of instances in a batch
M, number of particles
$\boldsymbol{\theta}^{*(n-1)}$, the logarithm of the parameters of the proposal function
$\boldsymbol{\beta}^{*(n-1)}$, the learned parameters from the previous batch
Output:
$\beta_i^{*(n)}$, the trained regression parameters for batch n

for $m \leftarrow 1$ **to** M **do**
  for $i \leftarrow 1$ **to** D **do**
    Calculate $\bar{\theta}_i \leftarrow$ mean of $\theta_{im}^{(n-1)}$
    Generate $\theta_{im}^{(n)} \sim \mathcal{N}(a\theta_{i,k_m^{(n-1)}}^{(n-1)} - (1-a)\bar{\theta}_i, h^2)$
    Calculate $\sigma_{im}^{(n)} \leftarrow \exp(\theta_{im}^{(n)})$
    Generate $\beta_{im}^{(n)} \sim \mathcal{N}(\beta_{i,k_m^{(n-1)}}^{(n-1)}, \sigma_{im}^{(n)})$
  end
  Calculate training accuracy $A_m \leftarrow A_m[f(\boldsymbol{\beta}_m^{(n)}, \{\mathbf{x}, y\}^{(n)}), B]$
end

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

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

Combine the classifiers with majority voting
$\boldsymbol{\beta}^{*(n)} \leftarrow \bar{\boldsymbol{\beta}}$, where $\bar{\boldsymbol{\beta}}$ is the mean of $\boldsymbol{\beta}_{k^{(n)}}^{(n)}$ over all $k^{(n)}$
$\boldsymbol{\theta}^{*(n)} \leftarrow \bar{\boldsymbol{\theta}}$, where $\bar{\boldsymbol{\theta}}$ is the mean of $\boldsymbol{\theta}_{k^{(n)}}^{(n)}$ over all $k^{(n)}$
return $(\boldsymbol{\beta}^{*(n)}, \boldsymbol{\theta}^{*(n)})$ and $k_m^{(n)}$

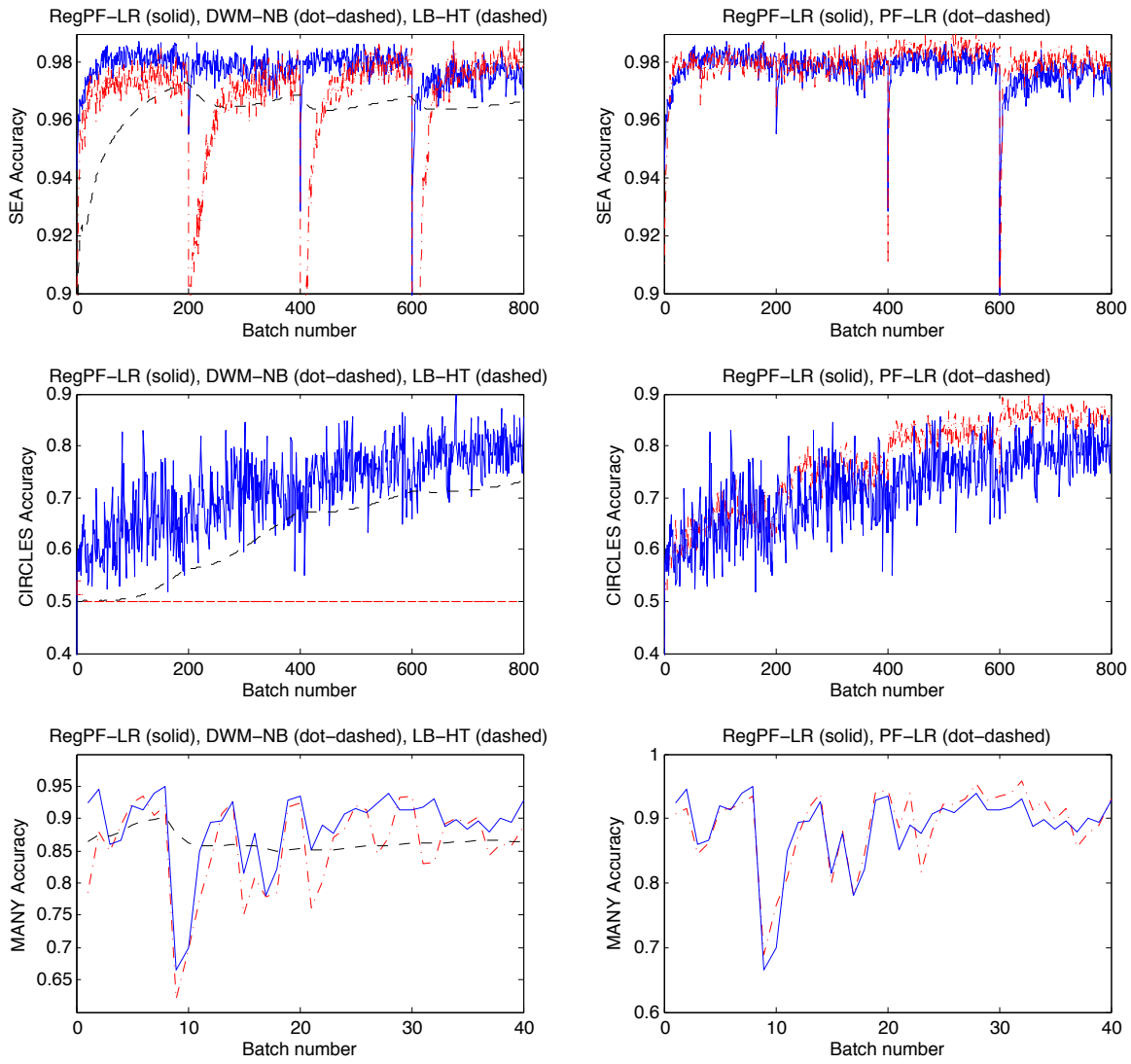

Figure 7: Comparison between RegPF-LR, DWM-NB, and LB-HT (left column), and between RegPF-LR and PF-LR (right column).

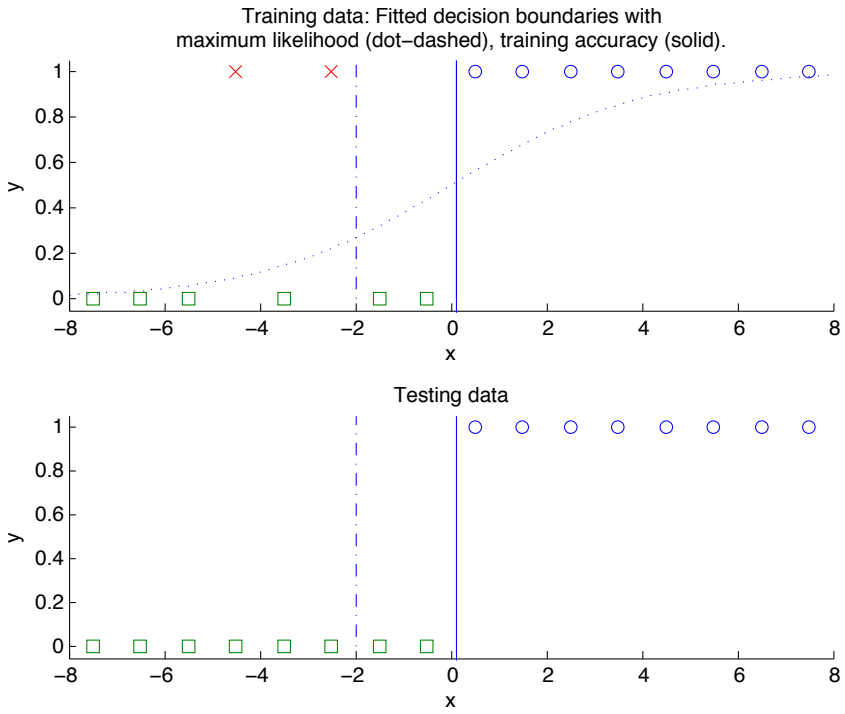

Figure 8: This figure shows that the maximum likelihood estimate (dot-dashed) is more sensitive to noise than the one obtained by maximum training accuracy (solid).

Appendix B. Dealing with Overfitting

Because of the small patch sizes required to give an accurate description of changing concepts, some conventional methods relying on optimizing the predictive accuracy becomes ineffective for mining streaming data. For instance, predictive accuracy obtained from holding out a subset of the batch at hand becomes unreliable. While cross validation may become too costly. This section illustrates qualitatively the mechanism in which PF-LR overcomes overfitting by maximizing the training accuracy.

Maximizing the training accuracy to fit a logistic regression model is less sensitive to the method of maximum likelihood. To see this, we will first look at a simple one dimensional case and generalize the argument to higher dimensions. Figure 8 shows two data sets with class labels $y \in \{0, 1\}$ on the y-axis and the features are evenly distributed on the x-axis. The training data set at the top has two instances affected by noise, marked by X. The testing data set is shown below, with the true decision boundary at $x = 0$. The vertical lines represent the decision boundary obtained by maximum likelihood (dot-dashed) and that obtained by maximum training accuracy (solid). The class labels of each instance are assigned $y = 1$ if it lies to the right of the boundary, and $y = 0$ otherwise. As seen in figure 8, a biased noise distribution skews the maximum likelihood estimate. However the fit obtained by maximum training accuracy is unaffected, as the training accuracy is still maximized at the true value. Because of the skewed likelihood estimate, the predictive

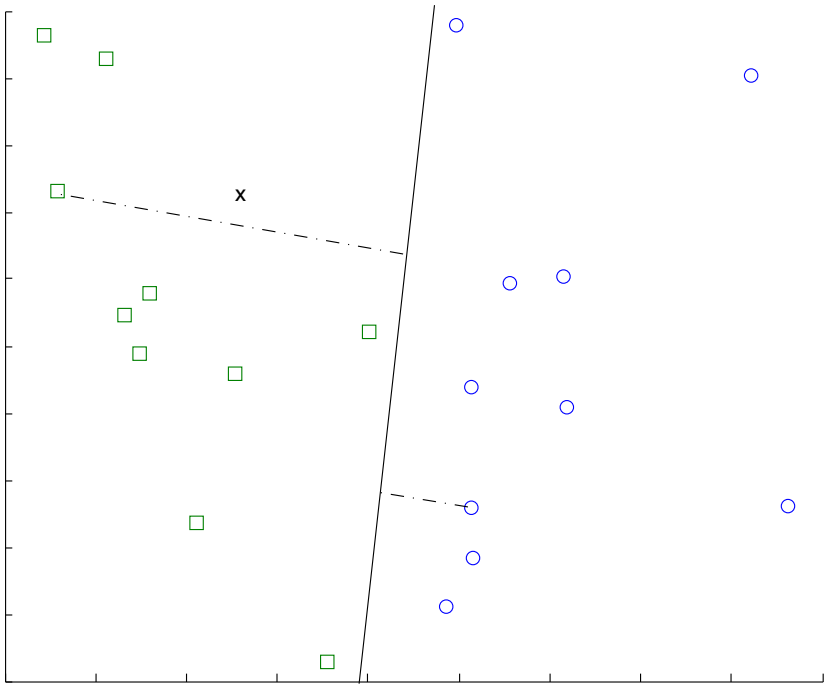

Figure 9: This figure shows the projection of instances into the one dimensional case. The true decision boundary is denoted by the solid line. The dot-dashed lines show the distances from the instance to the decision boundary along the direction perpendicular to it.

accuracy is affected. On the other hand, the estimate from maximum training accuracy is more robust to noise. This is shown in the bottom plot of figure 8.

Generalizing the above argument to higher dimension is straight forward. The scenario at any dimension can be reduced to the one dimensional case and apply the same argument in the previous paragraph. This is done by retaining only the projection of feature variables in the direction perpendicular to the decision boundary. A two dimensional case is shown in figure 9. Finally, this argument remains valid in the case where the distribution of instances are not even.

To further support our claim. We compare the results of maximum likelihood versus that of maximum training accuracy for the synthetic data sets we used in the main sections. Figure 10 shows that PF-LR with training accuracy outperforms PF-LR with the likelihood function in all synthetic data sets used. Further, the training accuracy approach continues to work when the noise level is high as in the CIRCLES data set. Where as the approach with the likelihood function breaks down, with its predictive performance no better than random guessing.

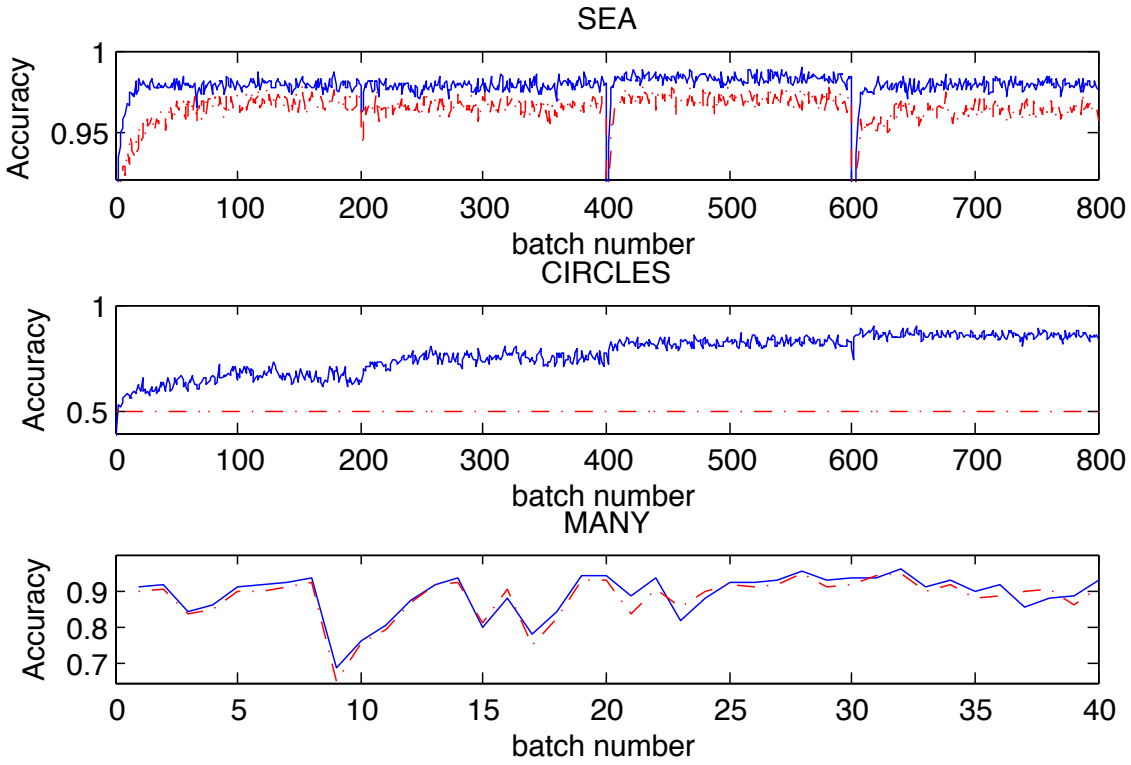

Figure 10: Predictive accuracy using the likelihood (dot-dashed) and training accuracy (solid).

Appendix C. The HYPERPLANE Data Set

The HYPERPLANE data set (Hulten et. al., 2001) contains variables $\mathbf{x} \in [0,1]^{10}$, with class labels $y = 1$ if $(x_i + x_{i+1} + x_{i+2})/3 > 0.5$, where $i = \{1, 2, 4, 7\}$. Since particle filtering is a Monte Carlo method, it suffers from the curse of dimensionality and particle filters perform poorly in high dimensions.

To alleviate the detrimental effects from the high number of dimensions, we implement a drift detection algorithm and then apply a dimensional reduction procedure. We say a drift is said to have occur if the training accuracy is less than 0.75. When this happens (say, at time n) we perform a logistic fit with the current batch and record the estimated coefficients $\boldsymbol{\beta}^{*(n)}$. Then we reduce the dimensionality by testing the logistic fit against the null hypothesis where each of the regression coefficients vanishes, $\beta_k^{(n)} = 0$, where k denotes the k-th component of $\boldsymbol{\beta}^{(n)}$, and keep only the ones with p-values less 0.05. This corresponds to rejecting the hypothesis of $\beta_k^{(n)} = 0$ with 95% confidence.

Figure 11 shows the effects of implementing a dimensional reduction procedure with drift detection. It is clear that dimensional reduction dramatically improves the performance of PF-LR. In figure 12, we compare the performances of PF-LR and DWM-NB. We see that PF-LR is on par with DWM-NB after dimensional reduction.

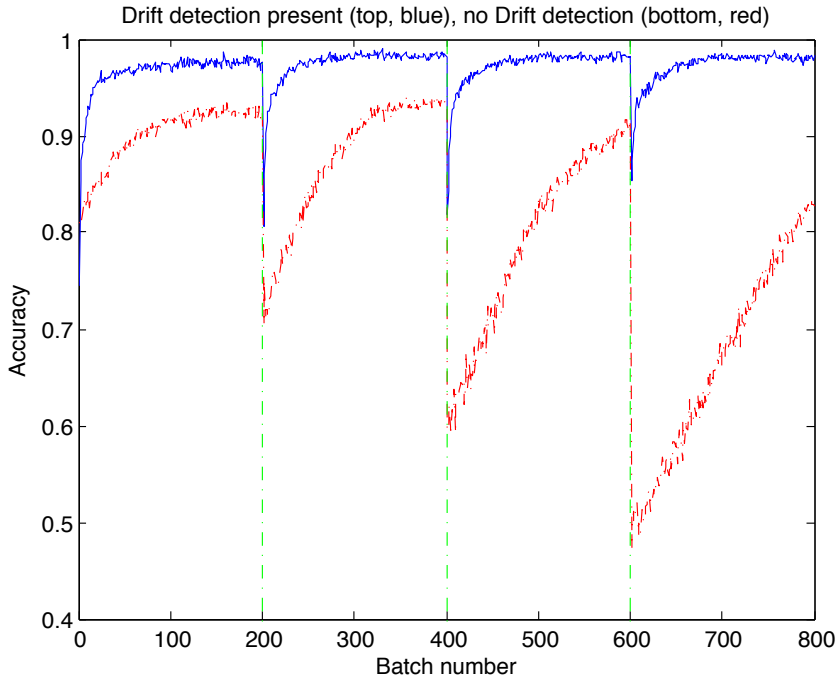

Figure 11: Blue: With drift detection. Red: Without drift detection.

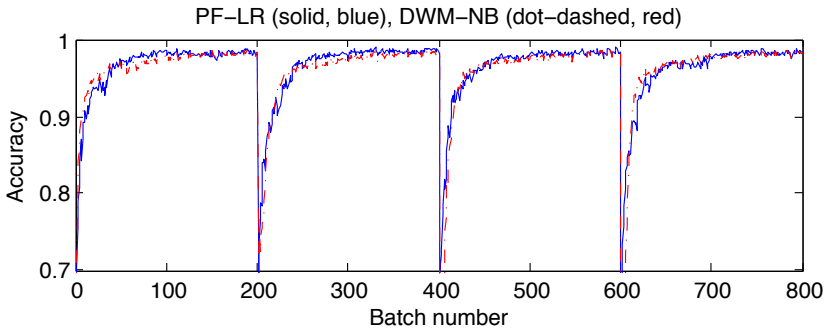

Figure 12: Particle Filter vs DWM-NB. The results shown for particle filter is the average over 50 runs on the same training set.

References

A. Doucet, A. M. Johansen. A tutorial on particle filtering and smoothing: fifteen years later. In Handbook of Nonlinear Filtering (editors D.Crisan and B.Rozovsky). Cambridge: Cambridge University Press, 2009.

C. C. Agarwal. Data streams: models and algorithms. Page 41. ISBN- 10: 0-387-28759-0. Springer, 2007.

L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state Markov chains. The annals of mathematical statistics 37 (6): 15541563, 1966.

A. Bifet, G. Holmes, and B. Pfahringer. Leveraging bagging for evolving data streams. In Proceedings of the 2010 European conference on machine learning and knowledge discovery in databases: Part I, ECML PKDD10, pages 135150, 2010c.

A. Bifet, G. Holmes, R. Kirkby, B. Pfahringer. MOA: Massive Online Analysis. Journal of Machine Learning Research 11, 1601-1604, 2010.

L. Breiman. Bagging predictors. Machine Learning, 24:123140, 1996.

L. Breiman. Arcing classifiers. The Annals of Statistics, 26(3):801849, 1998.

R. Casarin and J. Marin. Online data processing: comparison of bayesian regularized particle filters. Electronic Journal of Statistics. Vol. 3, 239258, 2009.

P. Domingos and G. Hulten. Mining high-speed data streams. In Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 7180. ACM Press, New York, NY, 2000.

Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In Proceedings of the Thirteenth International Conference on Machine Learning, pages 148156. Morgan Kaufmann, San Francisco, CA, 1996.

Y. Freund and R. E. Schapire. A Short Introduction to Boosting. Journal of Japanese Society for Artificial Intelligence, 14(5):771-780, September, 1999.

J. Geweke. Bayesian inference in econometric models using Monte Carlo integration, Econometrica, Vol. 57, pp. 1317-1339, 1989.

D. Grest, V. Krueger. Gradient-Enhanced Particle Filter for Vision-Based Motion Capture. Human Motion Understanding, Modeling, Capture and Animation. Lecture Notes in Computer Science Volume 4814, pp 28-41, 2007.

Gustafsson, F. Gunnarsson, N. Bergman, U. Forssell, J. Jansson, R. Karlsson, and P-J. Nordlund. Particle filters for positioning, navigation and tracking. IEEE Trans. Signal Processing, vol. 50, no. 2, pp. 425437, Feb. 2002.

M. Harries. Splice-2 comparative evaluation: Electricity pricing. Technical report, University of New South Wales, 1999.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten. The WEKA data mining software: An update; SIGKDD Explorations, Volume 11, Issue 1, 2009.

W. Hoeffding. Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association, 58(301):1330, 1963.

G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In Proc. of the 7th ACM SIGKDD, pages 97106. NY, USA, 2001.

C. Ji, Y. Zhang, M. Tong and S. Yang. Parallel problem solving from nature PPSN X, Lecture Notes in Computer Science Volume 5199, pp 909-918, 2008.

J. Kennedy and R. Eberhart. Particle swarm optimization. Proceedings of IEEE International Conference on Neural Networks IV. pp. 19421948, 1995.

A.D. Klamargias, K.E. Parsopoulos, Ph.D., Alevizos, M.N., Vrahatis. Particle Filtering with Particle Swarm Optimization in Systems with Multiplicative Noise, Genetic and Evolutionary Computation Conference 2008 (GECCO'08), Atlanta (GA), USA, pp. 57-62, 2008.

J. Z. Kolter, & M. A. Maloof. Dynamic weighted majority: An ensemble method for drifting concepts. Journal of Machine Learning Research 8:27552790, 2007.

N. Littlestone and M. K. Warmuth. The Weighted majority algorithm. Information and Computation, 108:212261, 1994.

J. Liu and M. West. Combined parameter and state estimation in simulation based filtering. In Doucet, A., de Freitas, N., and Gordon, N., editors, Sequential Monte Carlo Methods in Practice. Springer-Verlag, 2001.

F. L. Hedibert and R.S. Tsay. Journal of Forecasting J. Forecast. 30, 168209, 2011

M. A. Maloof. Concept drift. In J. Wang, editor, Encyclopedia of data warehousing and mining, pages 202206. Information Science Publishing, Hershey, PA, 2005.

L.L. Minku, A.P.White, X.Yao. The impact of diversity on on-line ensemble learning in the presence of concept drift, IEEE Transactions on Knowledge and Data Engineering, vol.22, no.5, pp. 730742, 2010.

C. Musso, N. Oudjane, , and F.Legland. Improving regularized particle filters. In Doucet, A., de Freitas, N., and Gordon, N., editors, sequential Monte Carlo methods in Practice. Springer-Verlag, 2001.

K. Nishida and K. Yamauchi. Detecting concept drift using statistical testing. In Discovery Science, pages 264269. Springer, 2007.

M. Pitt and N. Shephard. Filtering via simulation: auxiliary particle filters. Journal of the American Statistical Association, 94(446):590599, 1999

M.K. Pitt, R.S. Silva, R.S. Giordani and R. Kohn. Auxiliary particle filtering within adaptive Metropolis-Hastings sampling, http://arxiv.org/abs/1006.1914, 2010

L., Rokach. Ensemble-based classifiers. Artificial Intelligence Review 33 (1-2): 139, 2010

J. C. Schlimmer and R. H. Granger. Beyond incremental processing: Tracking concept drift. In Proceedings of the Fifth National Conference on Artificial Intelligence, pages 502507. AAAI Press, Menlo Park, CA, 1986.

J. C. Schlimmer , R. H.Granger. Incremental learning from noisy data, Machine Learning, 1(3), 317-354, 1986.

D.Sotoudeh, A. An. CIKM10, Proceedings of the 19th ACM international conference on Information and knowledge management. Pages 769-778, ACM New York, NY, USA 2010.

W. Street and Y. Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In Proc. of the 7th ACM SIGKDD. NY, USA, 2001.

A. Tsymbal. The problem of concept drift: definitions and related work. Computer Science Department, Trinity College Dublin, 2004.

H. Wang, W. Fan, P. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In ACM SIGKDD, pages 226235, 2003.

G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. Machine learning, 23(1):69101, 1996.

D. H. Wolpert. Stacked generalization. Neural Networks, 5(2):241259, 1992.

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