Evolving Systems (2010) 1:199–207
DOI 10.1007/s12530-010-9010-1

ORIGINAL PAPER

A fast and light classifier for data streams

Vahida Attar · Pradeep Sinha · Kapil Wankhade

Received: 19 April 2010 / Accepted: 4 July 2010 / Published online: 23 July 2010

Abstract Analysis of data streams is becoming a key area of data mining research, as the number of applications demanding such processing increases. Modern information technology allows information to be collected at a far greater rate than ever before. Machine learning offers promise of a solution, but the field mainly focuses on achieving high accuracy when data supply is limited. While this has created sophisticated classification algorithms, many do not cope with increasing data set size. When the data set size gets to a point where it could be considered to represent a continuous supply or data stream then incremental classification algorithms are required. When tackling with non-stationary concepts, ensemble of classifiers has several advantages over single classifier methods. They are easy to scale and parallelize, they can adapt to change quickly by pruning underperforming parts of the ensemble and they therefore usually generate more accurate concept descriptions and efficient results. But the effectiveness of an algorithm cannot simply be assessed by accuracy alone. Consideration needs to be given to the memory available to the algorithm and the speed at which data is processed in terms of both the time taken to predict the class of a new data sample and the time taken to include this sample in an incrementally updated classification model. This paper proposes a fast and light classifier for data stream classification.

Keywords Boosting · Fast and light · Sliding window · Hoeffding tree · Stream data mining

V. Attar (✉) · K. Wankhade
Department of Computer Engineering,
College of Engineering Pune, Pune, India
e-mail: vahida.comp@coep.ac.in

K. Wankhade
e-mail: kaps.wankhade@gmail.com

P. Sinha
High Performance Computing, Centre for Development of Advanced Computing (C-DAC), Pune, India
e-mail: psinha@cdac.in

1 Introduction

Machine learning methods are often applied to problems, where data is collected over an extended period of time. In many real world applications this introduces the change of underlying data distribution over the time. Knowledge discovery and data mining from time-changing data streams and concept drift handling on data streams have become important topics in the machine learning community and have gained increased attention recently. For example, companies collect an increasing amount of data like sales figures and customer data to find patterns in the customer behavior and to predict future sales. As the customer behavior tends to change over time, the model underlying successful predictions should be adapted accordingly.

Mining data streams of changing class distribution is important for real time business decision support. The stream classifier must evolve to reflect current class distribution. Learning from data stream is research area of increasing importance. Typical applications include network traffic monitoring, credit card fraud detection and sensor network management systems. Challenges are posed by data ever increasing in amount and in speed, as well as the constantly evolving concepts underlying the data.

Two fundamental issues of performance and adaption are to be addressed by any continuous mining attempt. Stream data stream classification algorithms have constraints like requirement of online responses, limited memory and computation resources. So the algorithm should make use of very light memory resources for storage of intermediate as well as final decision models and learning should be very fast, preferably in single pass of the data.

In context of data streams the underlying concept that maps the features to class labels may drift due to gradual or sudden changes of the external environment such as increase of network traffic or failures in sensors.

Ensemble classification technique is advantageous over single classification method. It is combination of several base models and it is used for continuous learning. Ensemble classifier has better accuracy over single classification technique. Bagging and boosting are two of the most well-known ensemble learning methods due to their theoretical performance guarantees and strong experimental results. Boosting has attracted much attention in the machine learning community as well as in statistics mainly because of its excellent performance and computational attractiveness for large datasets.

This paper makes an attempt to introduce a fast and light classifier for stream data. It uses siding window and dynamic weight assignment techniques. Related work in stream data classification is discussed in Sect. 2. Section 3 defines some fundamental concepts used in the design of algorithm. The proposed method and the algorithm is explained in Sect. 4. Section 5 supports the performance with results. Finally, Sect. 6 concludes the paper with suggestions for extension of work.

2 Related work

A substantial amount of work has focused on continuous mining of data streams. Domingos and Hulten (2000) devised a novel decision tree algorithm, VFDT, in order to overcome the long training times issue. The VFDT algorithm is based on a decision-tree learning method combined with sub-sampling of the entire data stream. The size of the sub-sample is calculated using distribution-free bounds called *Hoeffding bounds* under the assumption that the data is generated by a stationary distribution. For this reason, the method can process each example in constant time and memory being able to incorporate tens of thousands of examples per second using off-the-shelf hardware. The main drawback of the VFDT algorithm is its inability to cope with concept drifts. This was extended to CVFDT in an attempt to handle concept drift (Hulten et al. 2001). The CVFDT algorithm mines high-speed data streams under the approach of one-pass mining. The one pass mining approach does not recognize the changes which have occurred in the model during the data arrival process. Although the CVFDT algorithm seems to be an effective method for incremental updating of the classification model induced from a dynamic data stream, the claim is that the accuracy of such an incremental model cannot be greater than the best sliding window model.

Freund and Schapire (1996) proposed AdaBoost, a boosting algorithm. The AdaBoost algorithm adjusts adaptively to the errors of weak hypotheses, thus overcoming the problem of boosting by majority. In this algorithm, each new model in sequence will concentrate more on correctly classifying the examples that the previous models misclassified and concentrate less on those that are already correct.

An ensemble method for drifting data streams is primarily relied on bagging-style techniques (Street and Kim 2001; Oza and Russell 2001). Street and Kim (2001) gave an ensemble algorithm SEA that builds one classifier per data chunk independently. Adaptability relies solely on retiring old classifiers one at a time. Oza and Russell (2001) used online bagging and boosting algorithms for stream data classification. Online bagging is a good approximation to batch bagging to the extent that their base model learning algorithms produce similar models when trained with similar distributions of training examples. Given a training dataset T of size N, standard batch bagging creates M base models. Each model is trained by calling the batch learning algorithm Lb on a bootstrap sample of size N created by drawing random samples with replacement from the original training set.

OzaBoost Oza and Russell (2001), an online boosting algorithm is designed to correspond to the batch boosting algorithm AdaBoost. It generates a sequence of base models $h_1, h_2,\ldots,h_M$ using weighted training sets (weighted by $D_1, D_2,\ldots,D_M$) such that the training examples misclassified by model h_{m-1} are given half the total weight when generating model h_m and the correctly classified examples are given the remaining half of the weight. When the base model learning algorithm cannot learn with weighted training sets, one can generate samples with replacement according to D_m. In AdaBoost, an example's weight is adjusted based on the performance of a base model on the entire training set while in online boosting; the weight adjustment is based on the base model's performance only on the examples seen earlier.

Angelov et al. (2008) proposed two novel approaches for on-line evolving fuzzy classifiers, called eClass and FLEXFIS-Class. Both applied with different model architectures, including single model (SM) and multi-model (MM) architecture. eClass can have a multi-input–multi-output (MIMO) architecture with multiple hyper-planes as consequents of each fuzzy rule. eClass and FLEXFIS-Class methods are designed to work on a per-sample basis and are thus one-pass, incremental. Additionally, their structure (FRB) is evolving rather than fixed. Both methods evolve their rule structure as well as adapt their parameters in antecedent and consequent parts on demand from newly loaded samples. Authors extended the family of FRB classifiers with an evolving structure (fuzzy rules) called eClass that includes eClass0 and eClass1in (Angelov and Zhou 2008). Both classifiers are based on eTS-type fuzzy systems. The eClass0 uses class labels as outputs. Starting "from scratch" is a feature that is unique for eClass classifiers. The architecture of eClass1 differs significantly. It is based on regression over the features producing an estimate of the possibility that a data sample may belong to a certain class. It attains higher classification rates. Angelov and Kelly et al. (Kelly et al. 2008) set out to determine the possibility of achieving and improving the current correct diagnosis rate of cervical cancer by combining ATR-FTIR spectroscopy with a self-learning fuzzy classifier, eClass.

Pelossof et al. (2008) has designed OCBoost a new online coordinate boosting algorithm. It updates the weights of a boosted classifier, which yields a closer approximation to the edges found by Freund and Schapire's AdaBoost algorithm than previous online boosting algorithms. This algorithm minimizes AdaBoost's exponential loss function when training AdaBoost with m training examples, then adding a single example to the training set, and retraining with the new set of $m + 1$ examples.

Recently, new ensemble methods were devised by Bieft et al. (2009) using bagging and adaptive size hoeffding tree technique for classification of evolving data streams. OzaBagASHT is new bagging method which uses Adaptive Size Hoeffding Tree that sets the size for each tree. The maximum allowed size for the nth ASHT is twice the maximum allowed size for the $(n-1)$th tree. This method was attempted to improve bagging performance by increasing tree diversity.

The fast and light requirements of algorithm exclude high cost algorithms like SVM and k-nearest neighbor algorithm. k-NN is easy to implement by computing the distances from the test sample to all stored vectors, but it is computationally intensive, especially when the size of the training set grows. So because of this it is not useful for concept drifting data streams classification. Decision trees with fewer nodes as base decision models and ensemble technique for learning are preferred.

3 Preliminary definitions

Some of the fundamental concepts used in the design of proposed model are discussed in this section.

3.1 Boosting

Boosting is a general method for improving the accuracy of any given learning algorithm. Boosting has aims to combine "weak" classifiers to obtain one "strong" classifier. This is an iterative algorithm and in each iteration, a weak classifier is selected by minimizing an average training error. Figure 1 describes the structure of one iteration.

- Given: a training set of m examples e.g. $S = [(x_1, y_1),\ldots, (x_m, y_m)]$, with labels $y_i \in \{-1, +1\}$. Distribution p_l over the training examples, e.g. $p_l\,(0) = 1/m$.
- Training stage: the weak learner algorithm should generate a sequence of weak hypotheses (classifiers) with mean low error rate higher than 1/2, with high probability. For computing of hypothesis error ε, the weak hypotheses h compared to the labels y.

The first effect of boosting is that it generates a hypothesis whose error on the training set is small by combining many hypotheses whose error may be large (but still better than random guessing). It seems that boosting may be helpful on learning problems having either of the following two properties: (1) observed examples tend to have varying degrees of hardness. For such problems, the boosting algorithm tends to generate distributions that concentrate on the harder examples, thus challenging the weak learning algorithm to perform well on these harder parts of the sample space. (2) Learning algorithm is sensitive to changes in the training examples so that significantly different hypotheses are generated for different training sets. However, unlike bagging, boosting tries actively to force the weak learning algorithm to change its hypotheses by constructing a "hard" distribution over the examples based on the performance of previously generated hypotheses.

The second effect of boosting has to do with variance reduction. Intuitively, taking a weighted majority over many hypotheses, all of which were trained on different samples taken out of the same training set, has the effect of reducing the random variability of the combined hypothesis. Thus, like bagging, boosting may have the effect of producing a combined hypothesis whose variance is significantly lower than those produced by the weak learner.

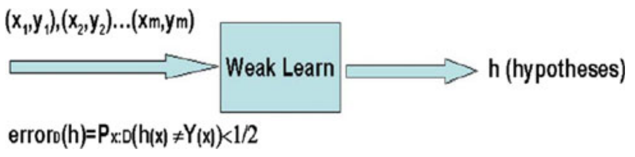

$(x_1,y_1),(x_2,y_2)\ldots(x_m,y_m)$
Weak Learn
h (hypotheses)
$error_D(h)=P_{x:D}(h(x) \neq Y(x))<1/2$

Fig. 1 Weak learner algorithm

However, unlike bagging, boosting may also reduce the bias of the learning algorithm.

3.2 Adaptive sliding window

In ADWIN (Albert Bifet and Ricard Gavald‘a 2007), a window is maintained that keeps the most recently read examples, and from which older examples are dropped according to some set of rules. The contents of the window can be used for the three tasks: (1) to detect change (e.g. by using some statistical test on different sub windows), (2) to obtain updated statistics from the recent examples and (3) to have data to rebuild or revise the model(s) after data has changed.

ADWIN is a change detector and estimator that solve in a well-specified way the problem of tracking the average of a stream of bits or real-valued numbers. ADWIN keeps a variable-length window of recently seen items, with the property that the window has the maximal length statistically consistent with the hypothesis “there has been no change in the average value inside the window”.

The idea of ADWIN method is simple: whenever two “large enough” sub windows of W exhibit “distinct enough” averages, one can conclude that the corresponding expected values are different, and the older portion of the window is dropped. The meaning of “large enough” and “distinct enough” can be made precise again by using the Hoeffding bound.

ADWIN is parameter and assumption-free in the sense that it automatically detects and adapts to the current rate of change. Its only parameter is a confidence bound δ, indicating how confident we want to be in the algorithm’s output, inherent to all algorithms dealing with random processes.

Also important for our purposes, ADWIN does not maintain the window explicitly, but compresses it (Datar et al. 2002). This means it keeps a window of length W using only O (logW) memory and O (logW) processing time per item, rather than the O (W) one expects from a naïve implementation. The technical result about performance of adaptive sliding window is:

False positive bound: if μt has remained constant within window W then the probability that adaptive sliding window shrinks window at most δ.
False negative bound: suppose window W divides into W0W1 in which W1 contains the most recent items and we have $|\mu W0 - \mu W1| > 2\varepsilon_{\text{cut}}$. Then with probability $1-\delta$ adaptive sliding window shrinks window W to W1, or shorter.

3.3 Hoeffding tree

A Hoeffding tree (Domingos and Hulten 2000) is an incremental, anytime decision tree induction algorithm that is capable of learning from massive data streams, assuming that the distribution generating examples does not change over time. Hoeffding trees exploit the fact that a small sample can often be enough to choose an optimal splitting attribute. This idea is supported mathematically by the Hoeffding bound, which quantifies the number of observations needed to estimate some statistics within a prescribed precision. More precisely, the Hoeffding bound states that with probability $1-\delta$, the true mean of a random variable of range R will not differ from the estimated mean after n independent observations by more than,

$$\varepsilon = \sqrt{\frac{R^2 \ln(1/\delta)}{2n}} \tag{1}$$

where n is the number of observations, ε is the Hoeffding bound and R is the base 2 logarithm of the number of possible class labels. A theoretically appealing feature of Hoeffding trees not shared by other incremental decision tree learners is that it has sound guarantees of performance.

4 Fast and light classifier

We use the boosting ensemble method since this supervised learning procedure provides a number of formal guarantees. Freund and Schapire (1996) proved a number of positive results about its generalization performance. More importantly, Friedman et al. (1998) showed that boosting is particularly effective when the base models are simple. This is most desirable for fast and light ensemble learning on stream data.

A typical boosting ensemble usually contains hundreds of classifiers. However, this lengthy learning procedure does not apply to data streams, where we have limited storage but continuous income data. Past data cannot stay long before making place for new data. Our boosting algorithm takes few classifier and adaptive sliding window. It is designed to retain the essential idea of boosting—sample weights updating according to concept drift.

Fast and light classifier (Fig. 2) simulates sampling with replacement using the Poisson distribution. An example’s weight is adjusted based on the base model’s performance only on the examples seen earlier.

In this algorithm, sliding window is used as *change detector* since it shrinks window if and only if there has been significant change in recent examples. Also used as *estimator* for the current average of the sequence it is reading since, with high probability, older parts of the window with a significantly different average are automatically dropped.

Basically, algorithm is light weight means that it uses less memory. Windowing technique does not store whole window explicitly but instead of this it only stores statistics

Fig. 2 Fast and light classifier algorithm

```
          Maintain boosting ensemble $E_b$ with classifiers
                         $\{C_1, C_2, \ldots, C_m\}$, $m \in \{1, 2, \ldots, M\}$
Input: stream S with examples $\{x_1, x_2, \ldots, x_t\} \in X$
          Base leaner- Hoeffding Tree
Output: a composite model $h_{fin}$
Method:
   1.  initialize base models $h_m$ for all $m \in \{1, 2, \ldots, M\}$, $w_{cc}=0$ , $w_{ic}=0$
   2.  for all training examples $x_t \in X$ do
   3.        set weight of example $w_d=1$
   4.        for m=1, 2,…., M do
   5.            set k = Poisson($w_d$)
   6.             for n=1, 2, …., k do
   7.                 update $h_m$ with the current example
   8.             end for
   9.             if $h_m$ correctly classifies the example then
  10.                 $w_{cc}= w_{cc}+ w_d$
  11.                 $w_d= w_d[N/2\ w_{cc}]$
  12.             else
  13.                 $w_{ic}= w_{ic} + w_d$
  14.                 $w_d= w_d[N/2\ w_{ic}]$
  15.              end if
  16.     initialize W as an empty list of buckets
  17.        initialize Width, Variance and Total
  18.        for each t > 0 do
  19.            Insert Element ($x_t$, W)
  20.               for every split of W into $W = W_0 . W_1$
  21.                    check for the contents of windows
  22.                    if $|\hat{\mu}_{W0} - \hat{\mu}_{W1}| < \varepsilon_{cut}$
  23.                         Change Alarm $\longleftarrow$ true
  24.                    end if
  25.               end for
  26.          end for
  27.  if change detected
  28.          $\varepsilon_m = w_{ic} / ( w_{cc} + w_{ic})$
  29.          $w_d= (1- \varepsilon_m)/ \varepsilon_m$
  30.  end if
  31.          end for
  32.   end for
  33.  return final hypothesis $h_{fin}(x)$
```

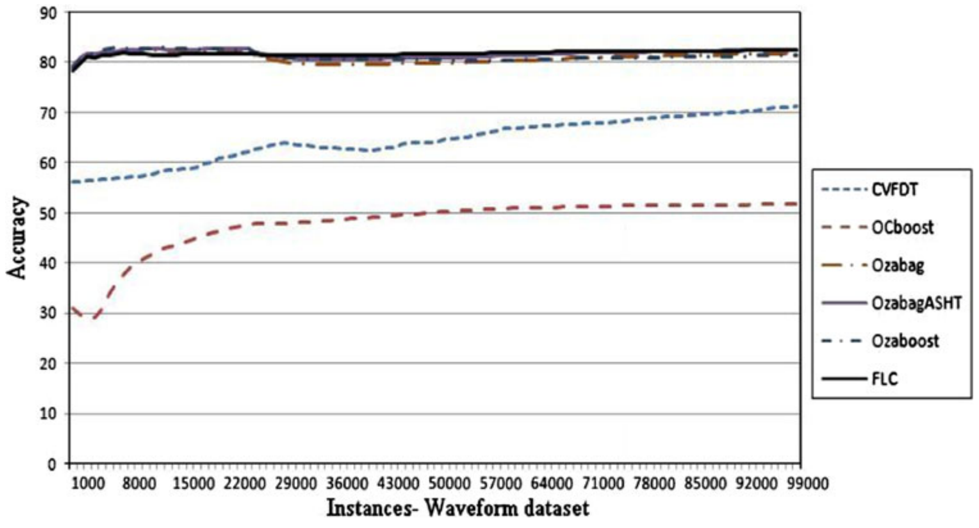
required for further computation. In this proposed algorithm, the data structure maintains an approximation of the number of 1's in a sliding window of length W with logarithmic memory and update time. The data structure is adaptive in a way that can provide this approximation simultaneously for about O (logW) sub windows whose lengths follow a geometric law, with no memory overhead with respect to keeping the count for a single window. Keeping exact counts for a fixed-window size is impossible in sub linear memory. This problem can be tackled by shrinking or enlarging the window strategically so that what would otherwise be an approximate count happens to be exact. More precisely, to design the algorithm a parameter M is chosen which controls the amount of memory used to O (MlogW/M) words. Boosting technique improves the performance.

Table 1 Comparison in terms of time, accuracy and memory using Waveform dataset

Classifier	Waveform data set		
	Time (s)	Accuracy	Memory (MB)
CVFDT	43.2	71.35	107.24
OzaBag	42.17	81.97	0.186
OzaBagASHT	50.58	82.67	0.121
OCBoost	55.67	51.84	0.142
OzaBoost	45.8	81.42	0.192
FLC	59.1	82.6	0.017

With Interleaved Test-Then-Train procedure the statistics are updated with every example in the stream, and can be recorded at that level of detail if desired. For efficiency

Fig. 3 Learning curves for waveform dataset with gradual concept drift



reasons a sampling parameter can be used to reduce the storage requirements of the results, by recording only at periodic intervals like the holdout method.

5 Experimental setup and results

Proposed algorithm is evaluated on both real and synthetic datasets with variable size and also introducing concept drift. The results are compared with some benchmarking algorithms like CVFDT, OzaBag, OzaBagASHT, OCBoost and OzaBoost the ensemble versions of bagging and boosting, to show the performance improvement in terms of time and memory requirement with improving accuracy.

5.1 Implementation details

The experiments were performed on a 2.59 GHz Intel Dual Core processor with 3 Gigabyte main memory, running Ubuntu 8.04. The fast and light classifier algorithm proposed in this paper is implemented in the Java programming language on Linux platform. For comparison of the performance different algorithms available in VFML (Hulten and Domingos 2003), WEKA (Witten and Frank 2005) and MOA (Geoffrey Holmes et al. 2007) toolkits are used.

5.2 Evaluation method: Interleaved Test-Then-Train

This scheme evaluates data stream algorithms to interleave testing with training. Each individual example can be used to test the model before it is used for training, and from this the accuracy can be incrementally updated. When intentionally performed in this order, the model is always being tested on examples it has not seen. This scheme has the advantage that no holdout set is needed for testing, making maximum use of the available data. It also ensures a smooth plot of accuracy over time, as each individual example will become increasingly less significant to the overall average.

The disadvantages of this approach are that it makes it difficult to accurately separate and measure training and testing times. Also, the true accuracy that an algorithm is able to achieve at a given point is obscured. Algorithms will be punished for early mistakes, regardless of the level of accuracy they are eventually capable of. This effect will diminish over time.

Table 2 Comparison in terms of time, accuracy and memory using SEA dataset

Classifier	SEA data set		
	Time (s)	Accuracy	Memory (MB)
CVFDT	6.98	82.74	25.87
OzaBag	2.46	84.42	0.021
OzaBagASHT	2.84	85.2	0.012
OCBoost	3.72	83.76	0.017
OzaBoost	2.8	84.9	0.026
FLC	2.6	88.11	0.007

5.3 Synthetic datasets

The synthetic data sets are generated with drifting concepts based on SEA and Waveform. For all the datasets, the ensemble size is 10 and 10% of noise is introduced. CVFDT, OzaBag, OzaBoost, OzaBagASHT, OCBoost and fast and light classifier (FLC) algorithms are evaluated on the common hardware settings and same parameters.

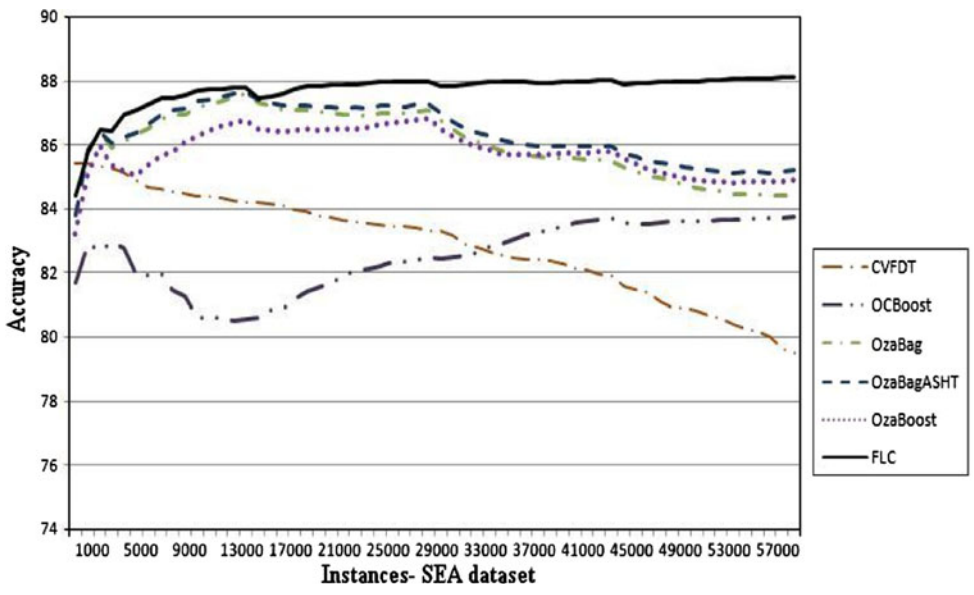


Fig. 4 Learning curves for SEA dataset with abrupt concept drift

5.3.1 *Gradual concept drift using waveform dataset*

The goal of the task is to differentiate between three different classes of waveform, each of which is generated from a combination of two or three base waves. The optimal Bayes classification rate is known to be 86%. There are two versions of the problem, wave 21 which has 21 numeric attributes, all of which include noise and wave 40 which introduces an additional 19 irrelevant attributes. In this experiment, wave 40 version dataset of 100,000 instances is generated. The points of dataset are further divided into four blocks of different concepts (25,000 instances each). For concept drifting the attribute numbers are varied as 10, 15, 20 and 25 (Table 1). Figure 3 shows learning curves for the Waveform dataset.

5.3.2 *Abrupt concept drift using SEA dataset*

This artificial dataset contains abrupt concept drift, first introduced in (Oza and Russell 2001). It is generated using three attributes, where only the two-first attributes are relevant. All three attributes have values between 0 and 10. The points of the dataset are divided into four blocks with different concepts. In each block, the classification is done using $f_1 + f_2 \leq \theta$, where f_1 and f_2 represent the first two attributes and θ is a threshold value. The most frequent values are 8, 9, 7 and 9.5 for the data blocks. Ten percent of class noise was then introduced into each block of data by randomly changing the class value of 10% of instances. The 60,000 instances are divided into four blocks and each blocks having different concepts (Table 2). The learning curves for the SEA dataset are plotted in Fig. 4.

Table 3 Real dataset information

Dataset	No. of instances	No. of attributes	Size (MB)
Adult	48,842	14	7.5
Census-income	199,522	40	99.1
KDDCUP-10%	494,020	42	71.4
Covertype	581,011	54	71.7

5.4 Experiments on real datasets

In this subsection we further verify our algorithm on real life data sets from UCI machine learning repository—http://archive.ics.uci.edu/ml/datasets.html. The datasets selected vary in the number of instances from 48,842 to 1045,000 and attribute size of 14–54 and the real data sets may have gradual or abrupt concept drifts at one or more points in the whole data sets.

The details of different real datasets having variable instances are tabulated in Table 3.

Fast and light classifier and other algorithms are used for the classification of above data sets independently. Performances in terms of classification accuracy, time and memory requirements are noted. Graphical comparison of fast and light classifier with OzaBag, OzaBagASHT, OCBoost and OzaBoost for all these data sets is plotted in Fig. 5.

5.5 Sensitivity of algorithm

Sensitivity is used to check how the fluctuation of noise level may influence the quality of a classifier. Sensitivity is involved with the percentage of noise in the dataset and minimum window size. Figure 6a shows the relationship

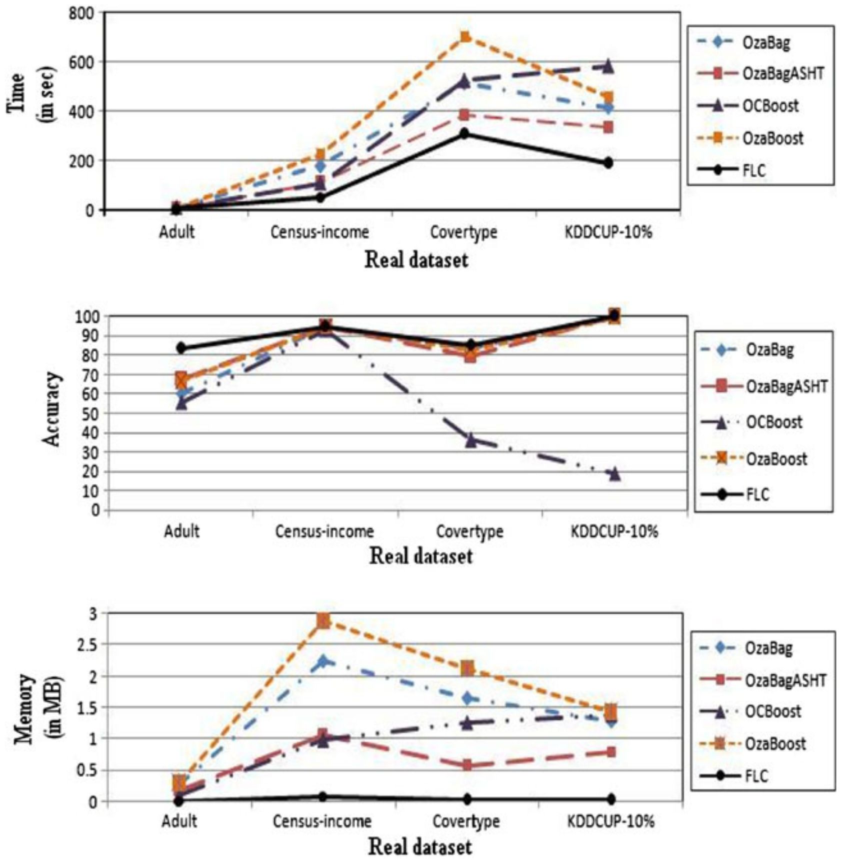


Fig. 5 Comparison in terms of time, accuracy and memory using real data sets

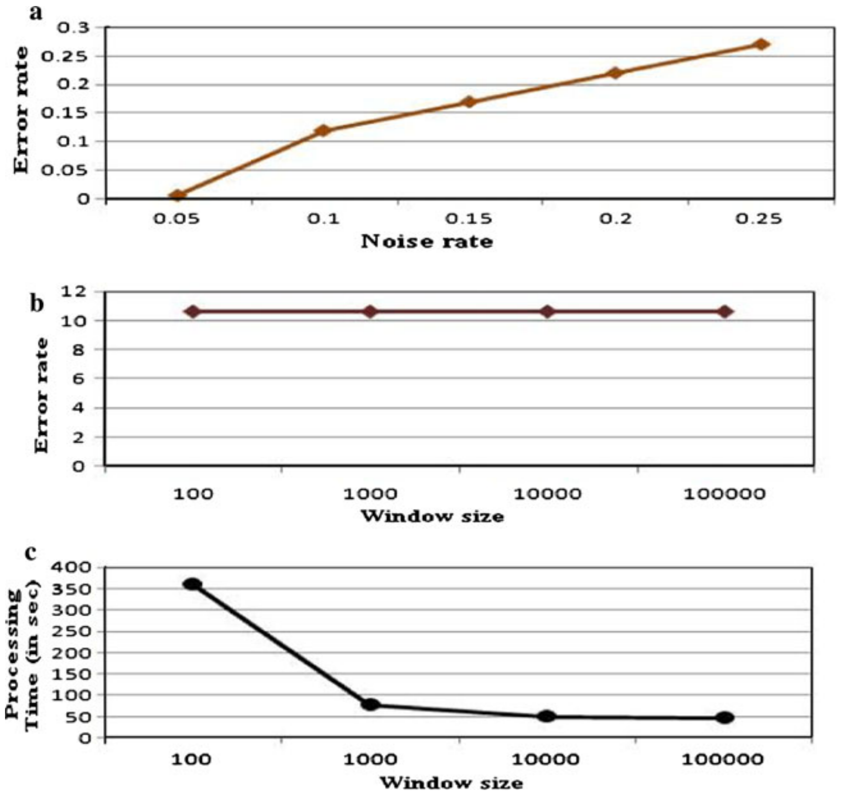


Fig. 6 Sensitivity: **a** noise versus error, **b** window size versus error and **c** window size versus processing time

between the noise and error rate. The error rate of the classifier increases proportionally to the percentage changes of the noise. This means that the performance of this algorithm will degrade when lots of noise presents in the dataset.

To check sensitivity of FLC algorithm against varying window size, we have used varying minimum window size ranges from 100 to 100,000. Figure 6b shows that error is independent of window size. Also Fig. 6c shows the processing time for varying window size. Generally, it will take longer time to complete the task using smaller window.

6 Conclusion and future work

Fast and light classifier, a new ensemble method for classification of stream data has been proposed in this paper. It uses adaptive windowing technique for change detection and estimation and boosting technique with hoeffding tree for building ensembles.

Experimental results show that the fast and light classifier proposed and implemented performs comparatively better with respect to accuracy, time and memory in classifying both synthetic and real data sets. It also deals better with concept a drift which is crucial problem of evolving data streams. It can be observed that as the size of data set increases performance comparatively improves. Sensitivity of the algorithm with respect to noise and window size is also discussed.

Future research work would be to handle the concept drift effects more effectively for abrupt drifts and the known real data sets. Using this methodology, we also would like to design parallel version of fast and light classifier and evaluate the same on GPU or suitable parallel architecture.

References

Angelov P, Zhou X (2008) Evolving fuzzy-rule-based classifiers from data streams. In: IEEE Trans Fuzzy Syst, ISSN 1063-6706, special issue on Evolving Fuzzy Systems, vol 16, no. 6, pp 1462–1475

Angelov P, Lughofer E, Zhou X (2008) Evolving fuzzy classifiers with different architectures. Fuzzy Sets Syst 159:3160–3182

Bieft A, Holmes G, Pfahringr B, Kirkby R, Gavalda R (2009) New ensemble methods for evolving data streams. In KDD, pp 139–148

Bifet A, Gavald'a R (2007) Learning from time changing data with adaptive windowing. In: SIAM Int Conf Data Min, pp 443–448

Datar M, Gionis A, Indyk P, Motwani R (2002) Maintaining stream statistics over sliding windows. SIAM J Comput 14(1):27–45

Domingos P, Hulten G (2000) Mining high-speed data streams. In: Knowledge discovery and data mining, pp 71–80

Dong G, Han J, Lakshmanan LVS, Pei J, Wang H, Yu P (2003) Online mining of changes from data streams: research problems and preliminary results. In: ACM SIGMOD MPDS

Freund Y, Schapire R (1996) Experiments with a new boosting algorithm. In: ICML, pp 148–156

Friedman J, Hastie T, Tibshirani R (1998) Additive logistic regression: a statistical view of boosting. In: The annals of statistics, 28(2), pp 337–407

Holmes G, Kirkby R, Pfahringer B (2007) MOA: massive online analysis. http://sourceforge.net/projects/moa-datastream

Hulten G, Domingos P (2003) VFML—a toolkit for mining high-speed time-changing data streams. http://www.cs.washington.edu/dm/vfml/

Hulten G, Spencer L, Domingos P (2001) Mining time-changing data streams. In: KDD'01, San Francisco, CA, pp 97–106

Kelly J, Angelov P, Walsh M, Pollock H, Pitt M, Martin-Hirsch and Martin F (2008) A self-learning fuzzy classifier with feature selection for intelligent interrogation of mid-IR spectroscopy data derived from different categories of exfoliative cervical cytology. Int J Comput Intell Res, ISSN 0974-1259, special issue on the future of fuzzy systems research, vol 4, no. 4, pp 392–401

Oza N, Russell S (2001) Experimental comparisons of online and batch versions of bagging and boosting. In: ACM SIGKDD, pp 359–364

Oza N, Russell S (2001) Online bagging and boosting. In: Artificial intelligence and statistics, Morgan Kaufmann, pp 105–112

Pelossof R, Jones M, Vovsha I, Rudin C (2008) Online coordinate boosting. http://arxiv.org/abs/0810.4553, pp 1–9

Street W, Kim Y (2001) A streaming ensemble algorithm (sea) for large-scale classification. In: ACM SIGKDD

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, 2nd edn. paper ISBN 0-12-088407-0, pp 1–525