Adaptive XML Stream Classification Using Partial Tree-Edit Distance

Dariusz Brzezinski and Maciej Piernik

Institute of Computing Science, Poznan University of Technology,
ul. Piotrowo 2, 60–965 Poznan, Poland
{dariusz.brzezinski,maciej.piernik}@cs.put.poznan.pl

Abstract. XML classification finds many applications, ranging from data integration to e-commerce. However, existing classification algorithms are designed for static XML collections, while modern information systems frequently deal with streaming data that needs to be processed on-line using limited resources. Furthermore, data stream classifiers have to be able to react to concept drifts, i.e., changes of the streams underlying data distribution. In this paper, we propose XStreamClass, an XML classifier capable of processing streams of documents and reacting to concept drifts. The algorithm combines incremental frequent tree mining with partial tree-edit distance and associative classification. XStreamClass was experimentally compared with four state-of-the-art data stream ensembles and provided best average classification accuracy on real and synthetic datasets simulating different drift scenarios.

Keywords: XML, data stream, classification, concept drift.

1 Introduction

In the past few years, several data mining algorithms have been proposed to discover knowledge from XML data [1,2,3,4]. However, these algorithms were almost exclusively analyzed in the context of static datasets, while in many new applications one faces the problem of processing massive data volumes in the form of transient data streams. Example applications involving processing XML data generated at very high rates include monitoring messages exchanged by web-services, management of complex event streams, distributed ETL processes, and publish/subscribe services for RSS feeds [4].

The processing of streaming data implies new requirements concerning limited amount of memory, short processing time, and single scan of incoming examples, none of which are sufficiently handled by traditional XML data mining algorithms. Furthermore, due to the nonstationary nature of data streams, target concepts tend to change over time in an event called *concept drift*. Concept drift occurs when the concept about which data is being collected shifts from time to time after a minimum stability period [5]. Drifts can be reflected by class assignment changes, attribute distribution changes, or an introduction of new classes (*concept evolution*), all of which deteriorate the accuracy of algorithms.

T. Andreasen et al. (Eds.): ISMIS 2014, LNAI 8502, pp. 10–19, 2014.

Although several general data stream classifiers have been proposed [5,6,7,8], they do not take into account the semi-structural nature of XML, like e.g. the XRules algorithm does for static data [1]. To the best of our knowledge, the only available XML stream classification algorithm was proposed by Bifet and Gavaldà [9]. However, this proposal focuses only on incorporating incremental subtree mining to the learning process and, therefore, does not fully utilize the structural similarities between XML documents. Furthermore, the classification method proposed by Bifet and Gavaldà is only capable of dealing with sudden concept drifts, but will not react to gradual drifts or concept evolution.

In this paper, we propose XStreamClass, a stream classification algorithm which employs incremental subtree mining and partial tree-edit distance to classify XML documents online. By dynamically creating separate models for each class, the proposed method is capable of dealing with concept evolution and gradual drift. Moreover, XStreamClass can be easily extended to a cost sensitive model, allowing it to handle skewed class distributions. We will show that the resulting system performs favorably when compared with existing stream classifiers, additionally being able to cope with different types of concept drift.

The remainder of the paper is organized as follows. Section 2 presents related work. In Section 3, we introduce a new incremental XML classification algorithm, which uses maximal frequent induced subtrees and partial tree-edit distance to perform predictions. Furthermore, we analyze possible variations of the proposed algorithm for different stream settings. The algorithm is later experimentally evaluated on real and synthetic datasets in Section 4. Finally, in Section 5 we draw conclusions and discuss lines of future research.

2 Related Work

As an increasingly important data mining technique, data stream classification has been widely studied by different communities; a detailed survey can be found in [5]. In our study, we focus on methods that adaptively learn from blocks of examples. One of the first of such block-based classifiers was the Accuracy Weighted Ensemble algorithm (AWE) [10], which trained a new classifier with each incoming block of examples to form a dynamically weighted and rebuilt classifier ensemble. More recently proposed block-based methods include Learn++NSE [11] which uses a sophisticated accuracy-based weighting mechanism and the Accuracy Updated Ensemble (AUE) [8] which incrementally trains its component classifiers after every processed block of examples.

However, all of the aforementioned algorithms are general classification methods, which are not designed to deal with semi-structural documents. On the other hand, although there exists a number of XML classifiers for static data [1,2], none of them is capable of incrementally processing streams of documents. To the best of our knowledge, the only streaming XML classifier is that proposed by Bifet and Gavaldà [9]. In this approach, the authors propose to adaptively mine closed frequent induced subtrees on batches of XML documents. The discovered subtrees are later used in the learning process, where labeled documents are encoded as tuples with attributes representing the occurrence/absence of frequent trees in a given document. Such tuples are later fed to a bagging or boosting ensemble of decision trees.

The proposed XStreamClass algorithm uses the AdaTreeNat [9] algorithm to incrementally mine maximal frequent induced subtrees and Partial Tree-edit Distance [12] to perform classification. Partial Tree-edit Distance (PTED) is an approximate subtree matching algorithm, which measures how much one tree needs to be modified to become a subtree of another tree. PTED is a combination of subtree matching [13] and tree-edit distance algorithms [14], and was designed specifically for XML classification.

3 The XStreamClass Algorithm

Existing data stream classification algorithms are not designed to process structural data. The algorithm of Bifet and Gavaldà [9] transforms XML documents into vector representations in order to process them using standard classification algorithms and, therefore, neglects the use of similarity measures designed strictly for XML. Furthermore, the cited approach is capable of dealing with sudden drifts, but not gradual changes or concept evolution. The aim of our research is to put forward an XML stream classifier that will use structural similarity measures and be capable of reacting to different types of drift. To achieve this goal, we propose to combine associative classification with partial tree-edit distance, in an algorithm called XStreamClass.

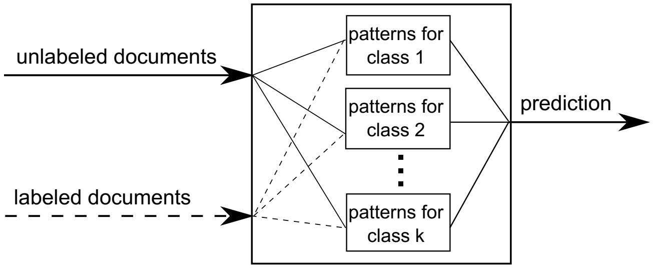

Fig. 1. XStreamClass processing flow

The XStreamClass algorithm maintains a pool of maximal frequent induced subtrees for each class and predicts the label of each incoming document by associating it with the class of the closest of all maximal frequent induced subtrees. The functioning of the algorithm can be divided into two subprocesses: training and learning. It is important to notice that, in accordance with the *anytime prediction* requirement of data stream classification [5], the training and learning processes can occur simultaneously and the algorithm is always capable of giving a prediction (Fig. 1). Algorithm 1 presents the details of the training, while Algorithm 2 summarizes the classification process.

Algorithm 1. XStreamClass: training

Input: $\mathcal{D}$: stream of labeled XML documents, m: buffer size, $minsup$: minimal support
Output: $\mathcal{P}$: set of patterns for each class

```
for all documents $d \in \mathcal{D}$ do
    $B \leftarrow B \cup \{d\}$;
    if $|B| = m$ then
        split documents into batches $B_i$ $(i = 1, 2, ..., k)$ according to class labels;
        $P_i \leftarrow AdaTreeNat_i(P_i, B_i, minsup)$;
        $\mathcal{P} \leftarrow P_1 \cup P_2 \cup ... \cup P_k$;
        $B \leftarrow \emptyset$
```

In the training process, labeled documents are collected into a buffer B. When the buffer reaches a user-defined size m, documents are separated according to class labels into batches B_i $(i = 1, 2, ..., k)$, were k is the number of classes. Each batch B_i is then incrementally mined for maximal frequent subtrees P_i by separate instances of the AdaTreeNat algorithm [9]. Since AdaTreeNat can mine trees incrementally, existing tree miners for each class are reused with each new batch of documents. Furthermore, in case of concept-evolution, a new tree miner can be created without modifying previous models. After the mining phase, frequent subtrees are combined to form a set of *patterns* $\mathcal{P}$, which is used during classification.

It is worth noticing that the training procedure can be slightly altered to achieve a more fluid update procedure. Instead of maintaining a single buffer B and waiting for m documents to update the model, one could create independent buffers for each class label. This would introduce the possibility of defining a different batch size for each class and enable better control of the training process for class-imbalanced streams. The influence of this fluid update strategy will be discussed in Section 4.

Algorithm 2. XStreamClass: classification

Input: $\mathcal{D}$: stream of unlabeled XML documents
Output: $\mathcal{Y}$: stream of class predictions

```
for all documents $d \in \mathcal{D}$ do
    calculate $\Delta(p, d)$ for each pattern $p \in \mathcal{P}$ using (1);
    $y \leftarrow$ class of $p = \arg\min \Delta(p, d)$ (or $p$ calculated according to (2));
    $\mathcal{Y} \leftarrow \mathcal{Y} \cup \{y\}$;
```

Classification is performed incrementally for each document using the set of current patterns $\mathcal{P}$. To assign a class label to a given document d, the algorithm calculates the partial tree-edit distance between d and each pattern $p \in \mathcal{P}$. The partial tree-edit distance is defined as follows [12]. Let s be a sequence of deletion or relabeling operations on leaf nodes or on the root node of a tree. A *partial tree-edit sequence* s between two trees p and d is a sequence which transforms p into any induced subtree of d. The cost $c(s)$ of a partial tree-edit sequence s is the total cost of all operations in s. *Partial tree-edit distance* $\Delta(p, d)$ between a pattern tree p and a document tree d is the minimal cost of all possible partial tree-edit sequences between p and d.

$$\Delta(p, d) = \min \{c(s) : s \text{ is a partial tree-edit sequence between } p \text{ and } d\} \quad (1)$$

After using (1) to calculate distances between d and each pattern $p \in \mathcal{P}$, XStreamClass assigns the class of the pattern closest to d. If there is more than one closest pattern, we propose a weighted voting measure to decide on the most appropriate class. In this scenario, each pattern is granted a weight based on its normalized support and size, and uses this value to vote for its corresponding class. The document is assigned the class c with the highest score, as presented in (2).

$$c = \underset{c_i, i=1,2,\ldots,k}{\arg\max} \sum_{\{p:class(p)=c_i\}} (support(p) \times size(p)) \quad (2)$$

In contrast to general block-based stream classifiers like AWE [10], AUE [8], or Learn++NSE [11], the proposed algorithm is designed to work strictly with structural data. Compared to XML classification algorithms for static data, such as XRules [1] or X-Class [2], we process documents incrementally, use PTED, and do not use default rules or rule priority lists. In contrast to [9], XStreamClass does not encode documents into tuples and calculates pattern-document similarity instead of pattern-document inclusion. Moreover, since XStreamClass mines for maximal frequent subtrees for each class separately, it can have different model refresh rates for each class. In case of class imbalance, independent refresh rates allow the algorithm to mine for patterns of the minority class using more documents than would be found in a single batch containing all classes. Additionally, this feature helps to handle concept-evolution without the need of rebuilding the entire classification model. Finally, because of its modular nature, the proposed algorithm can be easily implemented in distributed stream environments like Storm[1], which would enable high-throughput processing.

4 Experimental Evaluation

The aim of our experiments was to evaluate the XStreamClass algorithm on static and drifting data and compare it against four streaming classifiers employing the methodology presented in [9]: Online Bagging (Bag) [7], Accuracy Weighted Ensemble (AWE) [10], Learn++.NSE [11], and Accuracy Updated Ensemble (AUE) [8]. Bagging was chosen as the algorithm used by Bifet and Gavaldà to test their methodology and the remaining algorithms were chosen as strong representatives of block-based stream classifiers. We tested two versions of XStreamClass: one that synchronously updates all class models using a single batch (XSC) and one that updates each class model independently using separate batches for each class (XSC_F).

[1] `http://storm-project.net/`

XStreamClass was implemented in C# [2], tree mining was performed using a C++ implementation of AdaTreeNat [9], while the remaining classifiers were implemented in Java as part of the MOA framework [15]. The experiments were conducted on a machine equipped with a dual-core Intel i7-2640M CPU, 2.8Ghz processor and 16 GB of RAM. To make the comparison more meaningful, we set the same parameter values for all the algorithms. For ensemble methods we set the number of component classifiers to 10: AUE, NSE, and Bag have ten Hoeffding Trees, and since AWE uses static learners it has ten J48 trees. We decided to use 10 component classifiers as this was the number suggested and tested in [9]. The data block size used for block-based classifiers (AWE, AUE, NSE) was the same as the maximal frequent subtree mining batch size (see Section 4.1). For ensemble components we used Hoeffding Trees enhanced with Naive Bayes leaf predictions with a grace period $n_{min} = 100$, split confidence $\delta = 0.01$, and tie-threshold $\tau = 0.05$ [6].

4.1 Data Sets

During our experiments we used 4 real and 8 synthetic datasets. The real datasets were the `CSLOG` documents, which consist of web logs categorized into two classes, as described in [1,9]. The first four synthetic datasets were the `DS` XML documents generated and analyzed by Zaki and Aggarwal [1]. The additional four synthetic datasets were generated using the tree generation program of Zaki, as described in [9]. `NoDrift` contains no drift, `Sudden` contains 3 sudden drifts every 250k examples, `Gradual` gradually drifts from the 250k to 750k example, and `Evolution` contains a sudden introduction of a new class after the 1M example[2]. All of the used datasets are summarized in Table 1.

Table 1. Dataset characteristics

Dataset	#Documents	#Classes	#Drifts	Drift type	Minsup	Batch	Window size
`CSLOG12`	7628	2	-	unknown	5%	1000	-
`CSLOG123`	15037	2	-	unknown	4%	1000	-
`CSLOG23`	15702	2	-	unknown	6%	1000	-
`CSLOG31`	23111	2	-	unknown	3%	1500	-
`DS1`	91288	2	-	unknown	1%	5000	-
`DS2`	67893	2	-	unknown	1%	5000	-
`DS3`	100000	2	-	unknown	1%	5000	-
`DS4`	75037	2	-	unknown	0%	5000	-
`Evolution`	2000000	3	1	mixed	1%	10000	-
`Gradual`	1000000	2	1	gradual	1%	1000	1000
`NoDrift`	1000000	2	0	none	1%	10000	-
`Sudden`	1000000	2	3	sudden	1%	1000	1000

[2] Source code, test scripts, and datasets available at: `http://www.cs.put.poznan.pl/dbrzezinski/software.php`

As shown in Table 1, minimal support and batch size used for tree mining varied depending on the dataset size and characteristics. For datasets without drift or with concept evolution, we used incremental tree mining without any forgetting mechanism; for datasets with drift, a sliding window equal to the batch size [9]. All of the tested algorithms used the same patterns for classification.

4.2 Results

All of the analyzed algorithms were tested in terms of accuracy, classification time, and memory usage. The results were obtained using the test-then-train procedure [5], with pattern mining (model updates) occurring after each batch of examples. Tables 2–4 present average memory usage, batch classification time, and accuracy, obtained by the tested algorithms on all datasets, respectively.

In terms of memory usage, XStreamClass is the most efficient solution out of all the tested algorithms. This is especially visible for larger datasets, were general stream classifiers grow large classification models, while the proposed algorithm only needs to maintain a list of current maximal frequent subtrees.

As Table 2 shows, low memory consumption is achieved at the cost of relatively high classification time. XStreamClass needs to calculate the PTED between a document and each pattern, which is computationally more expensive than simple subtree matching. However, it is worth noticing that for larger streams XStreamClass offers comparable and sometimes better prediction speed due to its compact classification model.

Concerning accuracy, XStreamClass is the best algorithm on all but one dataset. This is a direct result of using of a tree similarity measure, which, in contrast to simple subtree matching, classifies a document even if it does not contain any of the patterns in the model. Moreover, since the classification model of XStreamClass depends strictly on the current set of frequent patterns, the algorithm has a forgetting model steered directly by adaptive pattern mining. This allows the proposed algorithm to react to changes as soon as they are visible in the patterns, in contrast to the compared algorithms, which would require a separate drift detection model. This was especially visible on the accuracy plot of the `Gradual` dataset presented in Fig. 2. One can see that around the 250k example accuracy slightly drops as gradual drift starts, but with time XStreamClass recovers from the drift.

It is also worth mentioning that independent batches for each class and, thus, asynchronous pattern mining for each class, offers better accuracy on practically all datasets. This is connected to the fact that in the XSC$_F$ processing scenario a larger number of examples is used for pattern mining for each class, which often allows to find better patterns. Furthermore, the `CSLOG` datasets are imbalanced with the majority class occurring three times as often as the minority class. Since XSC$_F$ waits for a larger sample of the minority class before pattern mining, it can achieve higher accuracy even on datasets as small as `CSLOG`.

To verify if the results of the compared classifiers are significantly different, we carried out statistical tests for comparing multiple classifiers over multiple datasets. We used the non-parametric Friedman test along with the Bonferroni-Dunn

Table 2. Average model memory usage [kB]

	AUE	AWE	Bag	NSE	XSC	XSC$_F$
CSLOG12	179.45	2376.80	441.36	38.11	**4.07**	4.78
CSLOG123	564.45	5226.85	889.44	55.04	**4.10**	4.82
CSLOG23	418.03	3642.58	749.76	53.86	**3.94**	5.09
CSLOG31	255.77	3642.58	453.46	48.41	**4.96**	6.50
DS1	1375.00	17114.75	2531.67	232.95	2.48	**2.47**
DS2	1331.89	15411.12	2631.16	613.50	**2.73**	2.80
DS3	1801.53	17574.12	3152.19	174.08	2.45	**2.39**
DS4	1510.63	16273.34	2604.43	123.87	**2.90**	3.53
Evolution	466.25	40979.98	1446.82	1539.28	2.63	**2.51**
Gradual	318.00	4746.09	1230.47	14522.56	**2.53**	2.63
NoDrift	500.81	40287.60	951.82	798.65	**2.54**	2.63
Sudden	263.39	4746.09	1093.75	13048.44	2.69	**2.67**

Table 3. Average block/batch classification time [s]

	AUE	AWE	Bag	NSE	XSC	XSC$_F$
CSLOG12	0.02	0.02	0.03	**0.01**	0.04	0.07
CSLOG123	0.06	**0.05**	0.07	0.05	0.14	0.15
CSLOG23	0.04	**0.04**	0.05	0.04	0.06	0.10
CSLOG31	0.02	0.01	0.03	**0.01**	0.03	0.04
DS1	0.35	0.31	0.43	**0.30**	1.22	0.31
DS2	0.33	**0.30**	0.41	0.30	1.39	0.35
DS3	0.41	0.36	0.47	0.35	1.08	**0.35**
DS4	0.57	0.54	0.63	**0.50**	2.06	0.62
Evolution	0.42	0.22	1.68	0.23	1.12	**0.11**
Gradual	0.03	**0.01**	0.12	0.02	0.07	0.12
NoDrift	0.44	0.25	1.03	0.27	0.83	**0.08**
Sudden	0.02	**0.01**	0.12	0.01	0.06	0.12

Table 4. Average classification accuracy [%]

	AUE	AWE	Bag	NSE	XSC	XSC$_F$
CSLOG12	76.91	76.91	75.60	76.91	75.77	**76.93**
CSLOG123	73.50	76.00	73.50	74.68	76.44	**78.80**
CSLOG23	77.14	77.01	74.71	76.06	78.19	**78.27**
CSLOG31	**76.52**	**76.52**	75.78	75.50	76.03	76.50
DS1	58.02	58.14	57.64	58.30	63.22	**64.95**
DS2	75.29	75.65	74.67	75.05	75.08	**79.78**
DS3	53.75	54.21	53.96	52.00	55.54	**59.51**
DS4	58.90	59.11	58.84	53.28	59.68	**61.31**
Evolution	53.79	54.07	53.81	52.05	**99.49**	**99.49**
Gradual	52.91	53.05	52.46	52.68	94.44	**97.65**
NoDrift	53.78	54.02	53.68	50.94	99.50	**99.99**
Sudden	52.58	52.49	52.45	51.20	96.33	**99.30**

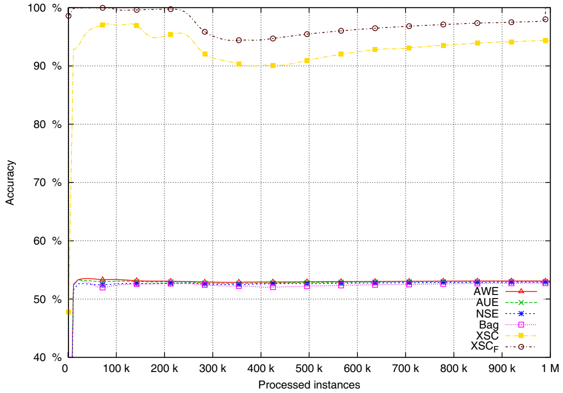

Fig. 2. Accuracy on the `Gradual` dataset

post-hoc test [16] to verify whether the performance of XSC/XSC$_F$ is statistically different from the remaining algorithms. The average ranks of the analyzed algorithms are presented in Table 5 (the lower the rank the better).

Table 5. Average algorithm ranks used in Friedman tests

	AUE	AWE	Bag	NSE	XSC	XSC$_F$
Accuracy	3.83	3.04	5.29	5.08	2.54	**1.21**
Memory	3.67	5.83	4.75	3.75	**1.33**	1.67
Testing time	3.33	**1.75**	4.92	1.83	5.17	4.00

By using the Friedman test [16] to verify the differences between accuracies, we obtain $FF_{Acc} = 25.38$. As the critical value for comparing 6 algorithms over 12 datasets for $p = 0.05$ is 2.38, the null hypothesis can be rejected and we can state that algorithm accuracies significantly differ from each other. Additionally, since the critical difference chosen by the Bonferroni-Dunn test is $CD = 1.97$, we can state that XSC$_F$ is significantly more accurate than AUE, Bag, and NSE. An additional one-tailed Wilcoxon test [16] shows that with $p = 0.001$ XSC$_F$ is also on average more accurate than AWE. A similar analysis ($FF_{Mem} = 71.01$, $FF_{Time} = 18.18$) shows that XSC is the most memory efficient algorithm and AWE and NSE are faster than XSC and XSC$_F$.

5 Conclusions

In this paper, we presented XStreamClass, the first algorithm to use tree similarity in classifying streams of XML documents. The algorithm combines incremental tree mining with partial tree-edit distance and associative classification.

Furthermore, we investigated different processing strategies to address the problem of class imbalance and concept evolution. Finally, we experimentally compared the proposed algorithm with the only competitive XML stream classification methodology. XStreamClass provided best classification accuracy and memory usage in environments with different types of drift as well as in static environments. Future lines of research will include different classification schemes using partial tree-edit distance and implementations for distributed environments.

Acknowledgments. D. Brzezinski's and M. Piernik's research is funded by the Polish National Science Center under Grants No. DEC-2011/03/N/ST6/00360 and DEC-2011/01/B/ST6/05169, respectively.

References

1. Zaki, M.J., Aggarwal, C.C.: Xrules: An effective algorithm for structural classification of xml data. Machine Learning 62(1-2), 137–170 (2006)
2. Costa, G., et al.: X-class: Associative classification of xml documents by structure. ACM Trans. Inf. Syst. 31(1), 1–3 (2013)
3. Brzezinski, D., et al.: XCleaner: A new method for clustering XML documents by structure. Control and Cybernetics 40(3), 877–891 (2011)
4. Mayorga, V., Polyzotis, N.: Sketch-based summarization of ordered XML streams. In: Ioannidis, Y.E., Lee, D.L., Ng, R.T. (eds.) ICDE, pp. 541–552. IEEE (2009)
5. Gama, J.: Knowledge Discovery from Data Streams. Chapman and Hall (2010)
6. Domingos, P., Hulten, G.: Mining high-speed data streams. In: Proc. 6th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min., pp. 71–80 (2000)
7. Oza, N.C., Russell, S.J.: Experimental comparisons of online and batch versions of bagging and boosting. In: Proc. 7th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min., pp. 359–364 (2001)
8. Brzezinski, D., Stefanowski, J.: Reacting to different types of concept drift: The accuracy updated ensemble algorithm. IEEE Trans. on Neural Netw. Learn. Syst. 25(1), 81–94 (2014)
9. Bifet, A., Gavaldà, R.: Adaptive xml tree classification on evolving data streams. In: Buntine, W., Grobelnik, M., Mladenić, D., Shawe-Taylor, J. (eds.) ECML PKDD 2009, Part I. LNCS, vol. 5781, pp. 147–162. Springer, Heidelberg (2009)
10. Wang, H., et al.: Mining concept-drifting data streams using ensemble classifiers. In: Proc. 9th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min., pp. 226–235 (2003)
11. Elwell, R., Polikar, R.: Incremental learning of concept drift in nonstationary environments. IEEE Trans. Neural Netw. 22(10), 1517–1531 (2011)
12. Piernik, M., Morzy, T.: Partial tree-edit distance. Technical Report RA-10/2013, Poznan University of Technology (2013), `http://www.cs.put.poznan.pl/mpiernik/publications/PTED.pdf`
13. Valiente, G.: Constrained tree inclusion. J. Discrete Alg. 3(2-4), 431–447 (2005)
14. Pawlik, M., Augsten, N.: RTED: A robust algorithm for the tree edit distance. PVLDB 5(4), 334–345 (2011)
15. Bifet, A., et al.: MOA: Massive Online Analysis. J. Mach. Learn. Res. 11, 1601–1604 (2010)
16. Demsar, J.: Statistical comparisons of classifiers over multiple data sets. J. Machine Learning Research 7, 1–30 (2006)