AnyOut: Anytime Outlier Detection on Streaming Data

Conference Paper · April 2012

DOI: 10.1007/978-3-642-29038-1_18

4 authors, including:

Ira Assent
Aarhus University
96 PUBLICATIONS **2,276** CITATIONS
SEE PROFILE

Thomas Seidl
Ludwig-Maximilians-University of Munich
348 PUBLICATIONS **5,714** CITATIONS
SEE PROFILE

AnyOut: Anytime Outlier Detection on Streaming Data

Ira Assent[1], Philipp Kranen[2], Corinna Baldauf[2], and Thomas Seidl[2]

[1] Dept. of Computer Science, Aarhus University, Denmark
[2] Data Management and Data Exploration Group, RWTH Aachen University, Germany

Abstract. With the increase of sensor and monitoring applications, data mining on streaming data is receiving increasing research attention. As data is continuously generated, mining algorithms need to be able to analyze the data in a one-pass fashion. In many applications the rate at which the data objects arrive varies greatly. This has led to anytime mining algorithms for classification or clustering. They successfully mine data until the a priori unknown point of interruption by the next data in the stream.

In this work we investigate anytime outlier detection. Anytime outlier detection denotes the problem of determining within any period of time whether an object in a data stream is anomalous. The more time is available, the more reliable the decision should be. We introduce AnyOut, an algorithm capable of solving anytime outlier detection, and investigate different approaches to build up the underlying data structure. We propose a confidence measure for AnyOut that allows to improve the performance on constant data streams. We evaluate our method in thorough experiments and demonstrate its performance in comparison with established algorithms for outlier detection.

1 Introduction

Wide availability of sensors, surveillance and measuring technology has lead to a remarkable increase in streaming data. Streaming data is typically continuously collected, and thereby needs to be analyzed in a one-pass manner as the data arrives. This is in contrast to traditional databases, where data may be accessed randomly and more than once.

Existing stream mining algorithms, usually assume constant streams in the sense that the time between the arrival of any two objects is fixed. This assumption does not hold for applications that collect data as and when required. For example, sensor networks minimize energy consumption and communication overhead by sending information to a server only in the case of events or changes. As a consequence, the time for analyzing the data stream may vary greatly. When a burst occurs, little time is available. When stream speed is slow, the additional time should be used to improve accuracy. This type of behavior characterizes the new family of anytime mining algorithms, which received considerable research attention for clustering and classification [18,29,14,38,11].

S.-g. Lee et al. (Eds.): DASFAA 2012, Part I, LNCS 7238, pp. 228–242, 2012.

Outlier detection has been defined as finding "an observation which deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" [22]. In this work we investigate the anytime outlier detection problem. We introduce our AnyOut technique, which uses a cluster-based approach to represent data in a hierarchical fashion. As long as time is available, the hierarchy is traversed to determine an outlier score based on the deviation between object and cluster. Until interrupted, more fine grained resolution of the clustering structure is analyzed. We discussed a sketch of the basic idea in a short workshop paper [5]. Here we investigate two ways to build up the tree structure and propose a confidence measure for our algorithm to harness its strength also for constant data streams. Besides the novel confidence measure and construction method a major contribution is a thorough evaluation against established baseline algorithms. We show how AnyOut successfully detects outliers in various streams settings and compare its performance to LOF, ABOD and OPTICSOF.

2 Related Work

Outlier detection is sometimes studied as a supervised problem, i.e., similar to unbalanced classification [45,16]. Supervised methods require labeled outliers as training data. This is often not the case in practice.

Unsupervised approaches do not assume labeled training data, but identify outliers based on their deviation from the remainder of the data. In statistical outlier detection, it is assumed that the data follows a certain distribution, and objects that do not well fit this assumption are outliers [6,35,36]. Distance-based outlier detection finds objects that show a high distance to most other objects [27]. Clustering-based outlier detection uses clusters to identify the inherent structure of valid data, and finds objects that are not clustered well [15,23]. While many methods separate outliers and inliers (valid data), finding a clear boundary may prove difficult. The Local Outlier Factor (LOF) method relaxes this requirement in favor of a scoring function that reflects the degree of deviation [9]. The result is then not a set of outliers, but a ranking by degree of outlierness.

Also for time series data, outlier detection has been proposed [37]. Time series are sequences of values in temporal order. Please note that while this bears some resemblance with streaming data, the important difference is that time series are assumed to be available entirely at the time of outlier detection. Moreover, the goal is not to identify outlying objects as in data streams, but outlying patterns of several temporally neighboring values. Thus, time series outlier detection is different from outlier detection in data streams.

Recently, approaches for outlier detection on data streams have been proposed [2,4,40,41,17,43,25,42,10]. However, all of these approaches assume streams of fixed arrival rates and do not meet the requirements of anytime outlier detection: interruptible and improvement of accuracy with more time.

Anytime algorithms have been studied in artificial intelligence [28]. In anytime learning, the training phase for supervised learning is restricted [18,14].

Other approaches monitor the performance of anytime algorithms [21]. Anytime clustering is addressed by [29], anytime classification by [11,38]. We presented a preliminary version of our anytime outlier concept at a KDD workshop [5].

3 Detecting Outliers in Streaming Data

In outlier detection, the goal is to identify objects which deviate from the remainder of the data. Here, we abstract from the concrete notion of what constitutes an outlier in order to formalize the anytime outlier detection problem. Different outlier detection paradigms may be followed in order to address this problem. In Subsections 3.1 and 3.2, we propose a cluster-based solution.

As briefly sketched in Related Work as well, it may prove difficult to determine clear decision boundaries between outliers and inliers. Typically, objects deviate from the majority of the data objects to a varying degree. Consequently, many outlier detection techniques aim to capture this degree of deviation in a corresponding outlier score (e.g. LOF [9]). We assume that outlier detection is concerned with the computation of an outlier score that expresses the degree of outlierness. Please note that this is without loss of generality, as algorithms that separate outliers and inliers can be considered binary special cases.

Common for all approaches on streaming data is that they process the data as it arrives. Existing work on streaming outlier detection, however, has assumed that the stream of incoming data objects is of constant speed, i.e. the time that passes between any two objects arriving in the stream is constant for the entire stream. In many applications, e.g. in sensor networks that are optimized for low power consumption and communication overhead, data is generated depending on changes in the outside environment. These streams are not of constant, but of varying inter-arrival rates. Consequently, the amount of time that is available for outlier detection varies as well.

As illustrated in Figure 1, as more objects arrive within shorter periods of time, decision on outlier detection needs to be taken in less time accordingly. Please note that since the duration of a burst or the number of objects that arrive within a certain time window is generally not bounded, simply buffering objects until the stream slows down is not an option. If the buffer size is exceeded, objects are lost and detection accuracy drops unexpectedly and unpredictably. On the other hand, if the stream speed is slow, an ideal outlier detection algorithm should be capable of making use of this time in order to improve accuracy.

A naive solution to the requirement of anytime behavior might be to resort to a number of outlier detection methods ordered by their reliability. These could be accessed one by one until this collection of methods is interrupted. Clearly,

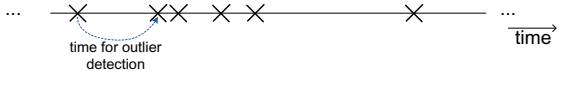

Fig. 1. Streaming data with varying inter-arrival rates

this method does not make best use of the time available as each method would have to start over from scratch.

An anytime outlier detection approach should thus be capable of

- fully using available time (more time leads to more accurate results)
- returning a result anytime (meaningful interruption as objects arrive)
- continuing computation incrementally (not simply starting over).

Definition 1. ***Anytime outlier detection.*** *Given a data stream of data objects o_i arriving at a priori unknown inter-arrival rate, the anytime outlier detection problem is to compute an outlier score $s(o_i)$ in the time t_i between the arrival of o_i and its successor o_{i+1}. The larger t_i, the more accurate the outlier score $s(o_i)$ should be.*

Please note that evaluation of the accuracy of outlier scores $s(o_i)$ is not straightforward. If synthetic or manually labeled data is available for empirical studies, this constitutes a ground truth for accuracy assessment. In practice, however, such ground truth is typically not available, and domain experts need to judge the quality of the outlier scores that are assigned to objects in the stream. This issue is not specific to streaming environments, but affects outlier detection research in general.

3.1 AnyOut : Overview over Our Method

In our AnyOut method, we propose following a cluster-based approached for outlier detection. As mentioned in the related work section, clustering methods have been successfully used to identify prevailing patterns of the data that serve as an input to the actual outlier detection.

In order to meet the requirements of fast initial response and improvement over time, we suggest using a hierarchy of clusters in a tree structure that is traversed until the algorithm is interrupted by the next arriving data object in the stream. Clusters at upper levels of the tree hierarchy subsume the more fine grained information at lower levels of the tree. The hierarchy of the tree thereby provides a natural organization of the clusters that can be incrementally accessed in order to refine the outlier score of the object in question. Initially, the object is compared only to the root node that describes the data distribution using few clusters. This comparison can be performed efficiently, thereby meeting the first requirement. Since more detailed information on the data distribution is available at lower levels of the tree, the reliability of outlier scores is typically improved. This aspect is empirically studied in the experiments, in Section 4.

3.2 Outlier Detection Using a Cluster Hierarchy

We introduced the ClusTree for clustering [29] as an extension to the R-tree family [19,38]. The core concept is the use of nodes to compactly represent clusters using cluster features. A cluster feature $CF = (n, LS, SS)$ is a tuple of the number of objects in the cluster n, of the linear sum of these objects LS, and of

their squared sum SS. Please note that both LS and SS are vectors of the same dimensionality as the data. The information in the tuple of a cluster feature CF suffices for the computation of cluster properties such as the mean and the variance of the objects within the cluster. Cluster features have been successfully used to summarize clusters in several existing works [3,44,29]. The ClusTree is created and updated like any multidimensional index structure. As an extension to these multidimensional indexes, the ClusTree additional provides buffer entries that are used for anytime insertion of clusters. Since anytime clustering is not the focus of this work, details on these buffers are omitted here. Interested readers are referred to [29].

Figure 2 presents an overview over the ClusTree structure. A node in this illustration consists of two entries that each contain a cluster feature (depicted here as a curve that represents the data distribution within the cluster), a pointer to child nodes and a buffer that might contain a cluster feature as well. Pointers to child nodes are used for navigation during anytime processing to go to finer representations of a particular cluster.

Given a data set of objects, the question is how to create the tree structure. While one could simply follow the bottom-up insertion method, a better structure can be achieved using top-down tree creation methods, also termed bulk loading in the indexing community. The general idea is to initialize the indexing structure such that data within a node is closely related. In this work, we follow a bulk loading method presented in [32] using the EM (expectation maximization) algorithm [12].

For anytime outlier detection, we propose using the structure of the ClusTree in order to perform cluster-based outlier score computation. The idea is to compare the object to clusters at the levels of the tree in a top-down fashion as long as the time allowance dictated by the stream permits.

Assessing the degree of outlierness in our AnyOut approach is based on the degree of accordance of the object with the closest cluster feature at the current level of resolution. Whenever a new object o_{i+1} arrives in the stream, this interrupts the descent down the tree with the current object o_i. We then compare the object o_i to the cluster feature to assess the outlier degree.

In order to define the actual outlier score, we reinvestigate the information stored in the cluster features. As mentioned above, cluster features provide sufficient information to compute statistical properties of the objects within the

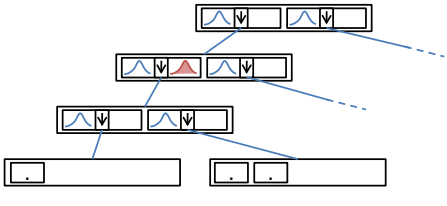

Fig. 2. ClusTree: Node entries represent clusters(descriptor, child pointer, buffer); child nodes more fine grained clusters of the parent data

cluster. Thus, objects in a cluster can be interpreted as observations of the data distribution in this part of feature space. Based on this intuition, we propose a *mean outlier score* that computes the degree of outlierness of an object o_i as the extent of deviation of o_i from the mean of the closest cluster feature. Since AnyOut operates directly on the ClusTree, the closest cluster feature is simply the one that is reached through tree traversal until the point of interruption by the next data object in the stream.

Definition 2. ***Mean outlier score.*** *For any data object o_i, the mean outlier score $s_m(o_i)$ is defined as $s_m(o_i) := dist(o_i, \mu(e_s))$, where $\mu(e_s)$ is the mean of entry e_s in the ClusTree that o_i is inserted into when the next object o_{i+1} of the data stream arrives.*

The mean outlier score thus assesses the deviation of the current object from the mean of the data distribution in the cluster feature of the current tree entry.

We propose a second way of assessing the outlierness that further extends the notion that cluster features represent the data distribution. By interpreting the cluster features as parameters of a Gaussian distribution of the data objects in this subtree, we arrive at a second outlierness score based on the density of the object. Recall that the Gaussian probability density of an object o_i for mean μ_{e_s} and covariance matrix Σ_{e_s} of an entry e_s is computed as

$$g(o_i, e_s) = \frac{1}{(2\pi)^{d/2} \cdot det(\Sigma_{e_s})^{1/2}} e^{\left(-\frac{1}{2}(o_i - \mu_{e_s})^T \Sigma_{e_s}^{-1} (o_i - \mu_{e_s})\right)} \quad (1)$$

where $det(\Sigma_{e_s})$ is the determinant and $\Sigma_{e_s}^{-1}$ the inverse of Σ_{e_s}. Please note that we can estimate the probability density of o_i on e_s based exclusively on e_s's cluster feature. While the full covariance is not available using a cluster feature (storing covariances is of quadratic complexity instead of linear complexity), using a matrix of variances has been empirically shown to achieve high accuracy results e.g. in classification [38]. Thus, a cluster feature is sufficient to compute an outlier score that reflects the density of the object with respect to the data distribution.

Definition 3. ***Density outlier score.*** *For any data object o_i, the density outlier score $s_d(o_i)$ is defined as $s_d(o_i) := 1 - g(o_i, e_s)$ (cf. Equation 1), where e_s is the entry in the ClusTree that o_i is inserted into when the next object o_{i+1} of the data stream arrives.*

Both the mean outlier score and the density outlier score reflect the degree of outlierness of the object at the point of interruption. In both cases, the data distribution of the closest cluster is the basis for the score computation. They differ in the way the cluster feature is interpreted: the mean outlier score only takes the center of mass of the data into account, whereas the density outlier score takes the overall data distribution into account, assuming a Gaussian distribution.

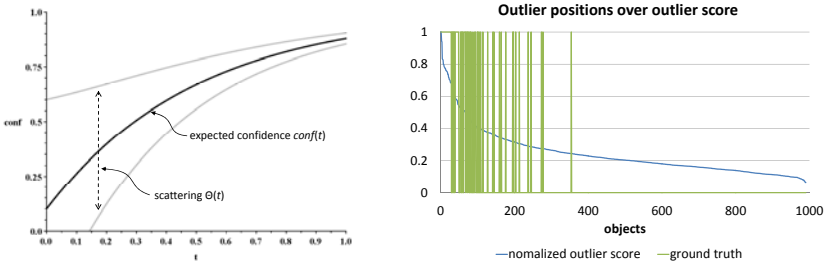

Fig. 3. Left: Expected confidence and scattering. Right: Actual outlier positions.

3.3 AnyOut Confidence Measure for Constant Streams

The benefit of anytime algorithms on constant data streams has first been shown in [33]. The scenario is as follows: the items arrive in a constant time interval t_a, the features are taken at time t_f and a decision must be available at time t_d. Instead of computing each object consecutively for a constant time t_a, the approach takes a window of several objects and processes these in parallel. Thereby it can freely distribute the available time among the objects, i.e. it can spend less time on *simple* cases and more time on more *difficult* cases.

The core point is the notion of simple and difficult, which is handled using a confidence measure. All objects are initialized and for each object a confidence value for the current result is computed that indicates how certain the algorithm is with the current output for that object. Just after the initialization the expected confidence value is rather low. If more time is used for computation, the expected confidence value increases. However, the actual confidence values will be scattered around the expected value, i.e. some items have a higher confidence and other lower than expected. The amount of scattering decreases with increasing computation time (cf. Figure 3 left). The scattered confidences are used to distribute the computation time, i.e. the item with the least confidence value is allowed to improve its result by one step (e.g. descending one more level in the AnyOut algorithm). This process is repeated until the time is used up, i.e. until the decision for the first item has to made. At that time the next window of items is taken to be processed in parallel. A second approach was proposed in [33], which uses a FiFo queue that contains all objects between t_f and t_d. We will use both approaches to evaluate AnyOut , for more details refer to [33].

To test the performance of the AnyOut algorithm on constant data streams we hence need a confidence measure for the current outlier score of an object. To this end we make use of the positions of the outliers in the ranking returned by the AnyOut algorithm. Figure 3 shows the results for the vowel data set (middle level of the tree, leaving out one class as outliers, cf. Section 4) using the mean outlier score. Optimally all outliers (green bars) would be on the far left. Obviously the objects with the highest outlier score are false alarms. Similar distributions are observable for other data sets. Therefore we use

$$conf(o) = e^{-s(o)}$$

as a simple and straight forward confidence measure, where $s(o)$ is the current outlier score of object o. Intuitively, we check for the putative outliers, i.e. the highest ranked objects, whether they are really outliers by giving them more computation time.

4 Experiments

In the following we first analyze the performance of our proposed AnyOut algorithm with respect to the individual levels of the tree structure as well as on different stream settings. In Section 4.3 we then compare our performance against established baseline methods, namely LOF, ABOD and OPTICSOF. We use the implementations of these algorithms provided in the ELKI framework [1], which is publicly available from the ELKI homepage[1]. Finally we discuss results for data streams containing drift and novelty in Section 4.4. All experiments use real world data sets of different characteristics from the UCI machine learning repository [24].

We evaluate the algorithms after all objects are processed, thereby we have a complete ranking of all objects with respect to their assigned outlier score in descending order. We use the ranking and the ground truth to compute the ROC curve for the results, which plots the true positive rate (TPR) over the false positive rate for each prefix of the ranking. From the ROC plots we derive the AUC value (area under the ROC curve [8]) as a first measure. We compute two further standard measures for ranking quality, namely the Spearman ranking coefficient (SRC) [39] and Kendall's Tau [26].

4.1 Level Analysis

For all experiments in Sections 4.1 and 4.2 we perform four fold cross validation. More precisely, we build up the tree (incrementally or with the described bulk loading technique) using the training set and analyze its performance. Continuous learning is investigated in Sections 4.3 and 4.4. To generate outliers in the following we leave out one class in the training set, i.e. the objects from the left out class which are contained in the test set are the outliers $\mathcal{O}$. In the plots we report the averaged performance values over all classes and folds.

To analyze the individual levels of the resulting tree structures in our AnyOut algorithm, we created one ranking for each level of the tree and computed the measures introduced above. Figure 4 shows the ROC plots for the vowel, pendigits and letter data sets. Each plot contains a ROC curve for the root level, the leaf level and a level in the middle. The tree structures in this experiment were build using incremental insertion and the employed outlier score was the mean outlier score. What is clearly visible from the results on all data sets, is that the basic principle of the AnyOut algorithm is effective, i.e. the quality of the results improves if more time is available (deeper levels are reachable).

[1] ELKI project homepage: `http://elki.dbs.ifi.lmu.de/`

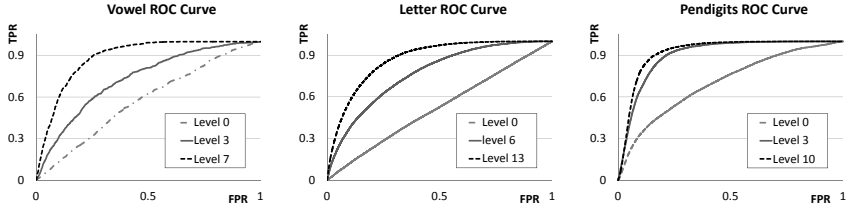

Fig. 4. ROC curves for the vowel, letter and pendigits data sets

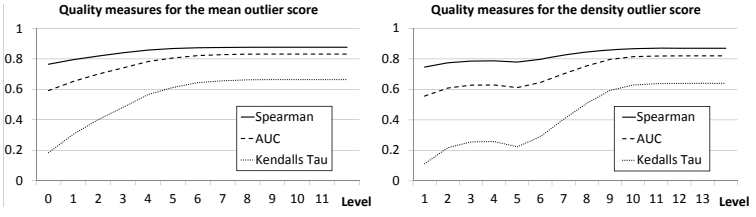

Fig. 5. Quality Measures for mean and density outlier scores on vowel

Similar observations can be made using the other quality measures. Figure 5 shows the results for Spearman's ranking coefficient, AUC and Kendalls Tau. The two plots compare the mean outlier score and the density outlier score on the vowel data set using bulk loading to construct the tree. (We compare incremental insertion and bulk loading in the next section.) The density outlier score yields slightly worse rankings than the mean outlier score. Moreover, the mean outlier score yields a constant quality improvement, whereas the density outlier score shows a slight reduction in ranking quality on intermediate levels. The mean outlier score showed better results throughout the data sets and we therefore employ it in the following.

4.2 AnyOut Performance on Various Stream Settings

To evaluate the anytime performance of AnyOut under variable stream scenarios, we recapitulate a stochastic model that is widely used to model random arrivals [13]. A Poisson process describes streams where the inter-arrival times are independently exponentially distributed. Poisson processes are parameterized by an arrival rate parameter λ:

Definition 4. ***Poisson stream.*** *A probability density function for the inter-arrival time of a Poisson process is exponentially distributed with parameter λ by*

$$p(t) = \lambda \cdot e^{-\lambda t}$$

The expected inter-arrival time of an exponentially distributed random variable with parameter λ is $E[t] = \frac{1}{\lambda}$.

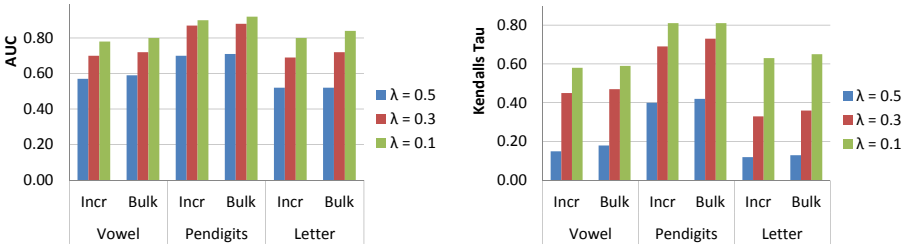

Fig. 6. Results on anytime streams

We use a Poisson process to model random stream arrivals for each fold of the cross validation. We randomly generate exponentially distributed inter-arrival times for different values of λ. If a new object arrives (the time between two objects has passed) we interrupt the AnyOut algorithm and store the outlier score for the object. We repeat this experiment using different expected inter-arrival times $\frac{1}{\lambda}$, where a unit corresponds to a level of the tree structure. We assume that any object arrives at the earliest after the initialization phase of the previous object, i.e. after evaluating the root of the tree.

Figure 6 shows the results for Poisson streams using the vowel, pendigits and letter data sets. In these experiments we compare the performance of the incremental insertion to the described bulk loading using the EM algorithm. In each group of bars the values for three different λ values are shown, while a smaller value corresponds to a slower stream according to Definition 4, i.e. more expected time per object.

Starting with the vowel data set in Figure 6 we see for the AUC measure and for Kendall's Tau, that both the incremental insertion and the bulk loading yield better qualities on slower streams. This is in line with the results from the level analysis in Section 4.1. Comparing the two tree construction methods we observe that on the one hand the bulk loading yields better results for any speed on both measures, but on the other hand the advantage is smaller than we expected.

The results on pendigits and letter (Figure 6) confirm both of the above findings. With a higher expected time per object (smaller λ value) the AnyOut algorithm shows its effectiveness and produces better rankings. The difference between incremental insertion and bulk loading performance are rather small but constant.

Summarizing the evaluation on varying data streams we can conclude that the proposed AnyOut method is effective for anytime outlier detection.

For constant streams, Figure 7 shows the corresponding results for the vowel data set using the window approach and FiFo approach introduced in Section 3.3. We evaluated different window and FiFo sizes from 2 to 12 (denoted as WS and FS in Figure 7). As above we performed four fold cross validation and computed the quality measures based on the final ranking of all objects. The results from Figure 7 (left) show that the performance of the AnyOut algorithm improves with larger

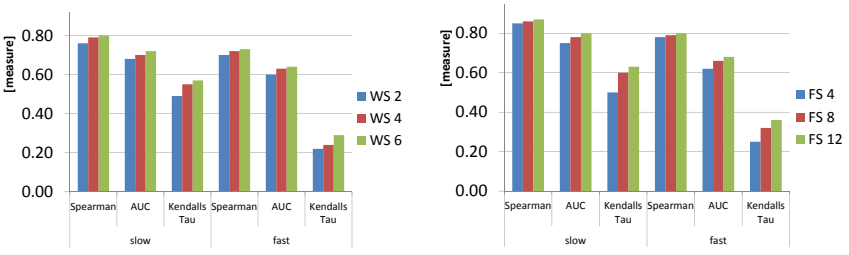

Fig. 7. AnyOut on constant data streams

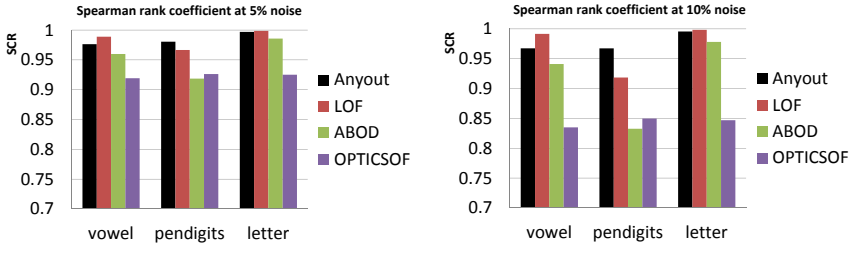

Fig. 8. Comparing AnyOut against baseline methods for 5% and 10% noise

window sizes for slow and fast streams. The results using the FiFo approach are even slightly better (cf. right part of Figure 7), which is attributable to the greater flexibility according to the comparably larger set of objects. This is in line with the results from [33] for anytime classification on constant streams and shows the effectiveness of the proposed confidence measure for the AnyOut algorithm.

4.3 Comparison to Baseline Methods

We assess the performance of AnyOut against established outlier detection methods LOF, ABOD and OPTICSOF from the ELKI framework on the same real world data sets. To generate outliers we added a certain percentage of noise uniformly distributed in the feature space. AnyOut processed the points one after the other incrementally inserting them into the tree structure and returning an outlier score. The competing approaches were given the entire data and were allowed random access. We tested k-values from 1 to 100 for LOF and OPTICSOF and report the best results per data set. This way we compare against the best possible values for AUC, SRC and Kendalls Tau from LOF, ABOD and OPTICSOF.

Figure 8 shows the resulting SRC values for all four approaches with 5 and 10 percent noise respectively. On vowel and letter the LOF algorithm achieves the highest SRC values for both noise settings. AnyOut and ABOD yield comparable yet slightly smaller SRC values and OPTICSOF follows at larger distance. On the pendigits data set AnyOut performs even slightly better than LOF, while the other two approaches perform at a lower level. The effects are slightly more pronounced at the higher noise level.

	Anyout				LOF				ABOD			
Noise	AUC	SRC	K. Tau	Time	AUC	SRC	K. Tau	Time	AUC	SRC	K. Tau	Time
5%	0.979	0.976	0.958	**24**	**0.996**	**0.989**	**0.992**	957	0.953	0.960	0.905	457
10%	0.979	0.967	0.958	**22**	**0.996**	**0.992**	**0.991**	1326	0.932	0.941	0.863	478

Fig. 9. Comparing AnyOut against baseline methods for different constraints on vowel

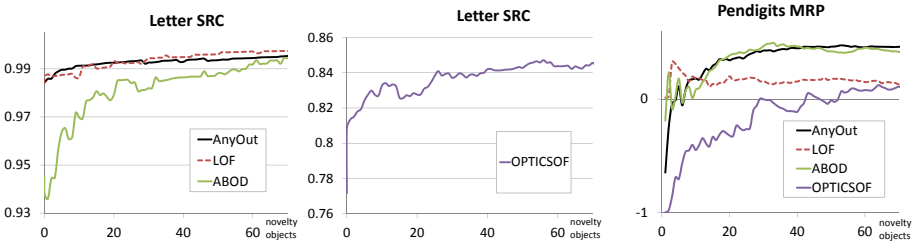

Fig. 10. Learning rate of the individual approaches in case of novelty

Figure 8 certifies AnyOut a competitive performance with respect to the Spearman rank coefficient. In Figure 9 we show the results on vowel for all three measures. Additionally we report the time in milliseconds that the algorithm used per object. As for the SRC in Figure 8, OPTICSOF performed worst and we spare the details for readability. Highlighted cells indicate best performance. LOF shows continuously best ranking values, closely followed by AnyOut and then ABOD at some distance. This is in line with the previous results. For the processing time we see that AnyOut is faster by at least one order of magnitude even for this small data set; on larger data sets the effect increases dramatically. This is however not surprising, since AnyOut is designed to be fast (logarithmic descent in the tree structure) while the competing approaches are not. Thus, we compare the ranking performance of our AnyOut against the best possible baseline: outlier approaches that are not capable of fast interruptible anytime processing.

4.4 Evolving Data Streams: Drift and Novelty

To test the performance of the algorithms for novelty and drift[2] in the data stream we used the same setup as in the previous section and left out one class in the first half of the stream. During the second half of the stream we evaluated the ranking after each object from the new class. The left part of Figure 10 shows the resulting SRC values for AnyOut, LOF and ABOD, the corresponding values for OPTICSOF are shown in the middle. ABOD shows the strongest reaction to the novel class. The SRC values are clearly lower for the first five novelty objects. After that they gradually increase until twenty objects from the novelty class have arrived. This effect is less pronounced for OPTICSOF, while the SRC values are generally lower (cf. Section 4.3). The reaction in terms of

[2] 'Novelty' refers to an entirely new concept (new class, new cluster), while 'Drift' refers to changes in the distribution of an existing concept.

SRC values is the smallest for LOF and AnyOut. Similar plots result for the other ranking measures. Since the reaction to an entirely new class is already very small, the reaction to gradually drifting concepts is negligible and not visible in the corresponding plots (not shown due to space limitations).

To further analyze the reasons for the minuscule reactions to novelty we investigate the positions of the novelty objects in the ranking of all objects. At any time we know the number n of outliers (noise objects). We normalize the ranking positions such that -1 indicates the highest rank (most probable outlier), 0 corresponds to the border object (n-th object) and 1 indicates the lowest rank (least probable outlier). As above, we evaluate the rankings after each novelty object. This time we compute the 'mean relative position' (MRP) of the novelty objects, i.e. the average of their ranking positions normalized as described above. The expected value for non-outlier objects is 0.5. Figure 10 shows the results for all four approaches on pendigits. OPTICSOF assigns the highest rank -1 to the first novelty object, after that the average rank value gradually increases as the algorithm learns the new concept. For LOF and ABOD the MRP hardly ever shows negative values. AnyOut still assigns a very high rank to the first novelty object (-0.65), but learns the new concept rapidly. After five to ten novelty objects the MRP value is above zero, i.e. the new class is no longer considered an outlier class.

In summary, AnyOut successfully learns novel concepts fast and handles drifting concepts, while being capable of anytime processing.

5 Conclusion and Future Work

In this paper we proposed the study of the anytime outlier detection problem for varying and constant data streams. We first analyzed and discussed the existing work on outlier detection and stream outlier detection before we formally defined the novel anytime outlier detection problem. We proposed a first algorithm called AnyOut long with two construction heuristics and a confidence measure for application with constant streams. We analyzed the structure and performance of AnyOut on various data streams using real world data. Finally we compared our method against established baseline algorithms finding comparable performance despite the largely reduced runtime. In future research more sophisticated confidence measures can be investigated as well as anytime outlier algorithms using other paradigms such as density based approaches.

The MOA framework [7] is an open source benchmarking software for data stream mining, similar to WEKA [20]. Recently we extended the MOA framework to support clustering and clustering evaluation on evolving data streams [30,34]. We now added stream outlier detection to the MOA framework [31] and integrated the AnyOut algorithm as well as a wrapper for the outlier methods provided by the ELKI framework [1].

Acknowledgments. This work has been supported by the UMIC Research Centre, RWTH Aachen University, Germany.

References

1. Achtert, E., Kriegel, H.-P., Reichert, L., Schubert, E., Wojdanowski, R., Zimek, A.: Visual Evaluation of Outlier Detection Models. In: Kitagawa, H., Ishikawa, Y., Li, Q., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 5982, pp. 396–399. Springer, Heidelberg (2010)
2. Aggarwal, C.C.: On abnormality detection in spuriously populated data streams. In: SDM (2005)
3. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: VLDB, pp. 81–92 (2003)
4. Angiulli, F., Fassetti, F.: Detecting distance-based outliers in streams of data. In: CIKM (2007)
5. Assent, I., Kranen, P., Baldauf, C., Seidl, T.: Detecting outliers on arbitrary data streams using anytime approaches. In: StreamKDD Workshop in Conjunction with 16th ACM SIGKDD (2010)
6. Barnett, V., Lewis, T.: Outliers in Statistical Data, 3rd edn. Wiley (1994)
7. Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., Seidl, T.: Moa: Massive online analysis, a framework for stream classification and clustering. Journal of Machine Learning Research - Proceedings Track 11, 44–51 (2010)
8. Bradley, A.: The use of the area under the ROC curve in the evaluation of machine learning algorithms. Pattern Recognition 30(7), 1145–1159 (1997)
9. Breunig, M., Kriegel, H.-P., Ng, R., Sander, J.: LOF: Identifying density-based local outliers. In: ACM SIGMOD, pp. 93–104 (2000)
10. Cao, H., Zhou, Y., Shou, L., Chen, G.: Attribute Outlier Detection over Data Streams. In: Kitagawa, H., Ishikawa, Y., Li, Q., Watanabe, C. (eds.) DASFAA 2010. LNCS, vol. 5982, pp. 216–230. Springer, Heidelberg (2010)
11. DeCoste, D.: Anytime query-tuned kernel machines via cholesky factorization. In: SDM (2003)
12. Dempster, A.P., Laird, N.M.L., Rubin, D.B.: Maximum likelihood from incomplete data via the em algorithm. J. Royal Stat. Soc. B 39(1), 1–38 (1977)
13. Duda, R., Hart, P., Stork, D.: Pattern Classification, 2nd edn. Wiley (2000)
14. Esmeir, S., Markovitch, S.: Interruptible anytime algorithms for iterative improvement of decision trees. In: UBDM Workshop at KDD (2005)
15. Ester, M., Kriegel, H.-P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases. In: KDD (1996)
16. Foss, A., Zaïane, O., Zilles, S.: Unsupervised Class Separation of Multivariate Data through Cumulative Variance-Based Ranking. In: ICDM (2009)
17. Franke, C., Gertz, M.: Detection and exploration of outlier regions in sensor data streams. In: ICDM Workshops, pp. 375–384 (2008)
18. Grefenstette, J., Ramsey, C.: An Approach to Anytime Learning. In: Workshop on Machine Learning, pp. 189–195 (1992)
19. Guttman, A.: R-trees: A dynamic index structure for spatial searching. In: ACM SIGMOD (1984)
20. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.: The weka data mining software: an update. SIGKDD Expl. Newsl. 11(1), 10–18 (2009)
21. Hansen, E.A., Zilberstein, S.: Monitoring anytime algorithms. SIGART Bulletin 7(2), 28–33 (1996)
22. Hawkins, D.: Identification of outliers. Chapman and Hall, New York (1980)
23. He, Z., Xu, X., Deng, S.: Discovering cluster-based local outliers. Pattern Recognition Letters (2003)

24. Hettich, S., Bay, S.: The UCI KDD archive (1999), http://kdd.ics.uci.edu
25. Hoang Vu, N., Gopalkrishnan, V., Namburi, P.: Online Outlier Detection Based on Relative Neighbourhood Dissimilarity. In: Bailey, J., Maier, D., Schewe, K.-D., Thalheim, B., Wang, X.S. (eds.) WISE 2008. LNCS, vol. 5175, pp. 50–61. Springer, Heidelberg (2008)
26. Kendall, M.: A new measure of rank correlation. Biometrika 30(1-2), 81 (1938)
27. Knorr, E., Ng, R., Tucakov, V.: Distance-based outliers: algorithms and applications. In: VLDBJ (2000)
28. Kotenko, I., Stankevitch, L.: The control of teams of autonomous objects in the time-constrained environments. In: ICTAI, pp. 158–163 (2002)
29. Kranen, P., Assent, I., Baldauf, C., Seidl, T.: Self-adaptive anytime stream clustering. In: ICDM (2009)
30. Kranen, P., Kremer, H., Jansen, T., Seidl, T., Bifet, A., Holmes, G., Pfahringer, B.: Clustering performance on evolving data streams: Assessing algorithms and evaluation measures within moa. In: ICDM (2010)
31. Kranen, P., Kremer, H., Jansen, T., Seidl, T., Bifet, A., Holmes, G., Pfahringer, B., Read, J.: Stream Data Mining using the MOA Framework. In: Lee, S.-G., et al. (eds.) DASFAA 2012, Part II. LNCS, vol. 7239, pp. 309–313. Springer, Heidelberg (2012)
32. P. Kranen and T. Seidl. Harnessing the strengths of anytime algorithms for constant data streams. *DMKD Journal (19)2, ECML PKDD Special Issue*, 2009.
33. Kranen, P., Seidl, T.: Harnessing the strengths of anytime algorithms for constant data streams. DMKD Journal 2(19) (2009) ECML PKDD Special Issue
34. Kremer, H., Kranen, P., Jansen, T., Seidl, T., Bifet, A., Holmes, G., Pfahringer, B.: An effective evaluation measure for clustering on evolving data streams. In: ACM SIGKDD, pp. 868–876 (2011)
35. Müller, E., Schiffer, M., Seidl, T.: Adaptive outlierness for subspace outlier ranking. In: CIKM, pp. 1629–1632. ACM (2010)
36. Müller, E., Schiffer, M., Seidl, T.: Statistical selection of relevant subspace projections for outlier ranking. In: ICDE, pp. 434–445. IEEE Computer Society (2011)
37. Muthukrishnan, S., Shah, R., Vitter, J.: Mining deviants in time series data streams. In: SSDBM (2004)
38. Seidl, T., Assent, I., Kranen, P., Krieger, R., Herrmann, J.: Indexing density models for incremental learning and anytime classification on data streams. In: EDBT (2009)
39. Spearman, C.: The Proof and Measurement of Association between Two Things. The American Journal of Psychology 15(1), 72–101 (1904)
40. Subramaniam, S., Palpanas, T., Papadopoulos, D., Kalogeraki, V., Gunopulos, D.: Online outlier detection in sensor data using non-parametric models. In: VLDB, pp. 187–198 (2006)
41. Yamanishi, K., Takeuchi, J., Williams, G., Milne, P.: On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms. DMKD Journal 8(3), 275–300 (2004)
42. Yang, D., Rundensteiner, E.A., Ward, M.O.: Neighbor-based pattern detection for windows over streaming data. In: EDBT, pp. 529–540 (2009)
43. Zhang, J., Gao, Q., Wang, H.: Spot: A system for detecting projected outliers from high-dimensional data streams. In: ICDE, pp. 1628–1631 (2008)
44. Zhang, T., Ramakrishnan, R., Livny, M.: BIRCH: an efficient data clustering method for very large databases. In: ACM SIGMOD (1996)
45. Zhu, C., Kitagawa, H., Faloutsos, C.: Example-Based Robust Outlier Detection in High Dimensional Datasets. In: ICDM (2005)