Outlier Detection in Data Streams

A Research Survey

Aditi Singla
Indian Institute of Technology
Delhi

Ankush Phulia
Indian Institute of Technology
Delhi

Vaibhav Bhagee
Indian Institute of Technology
Delhi

ABSTRACT

This paper discusses various techniques to detect outliers from streaming data, where the features or the data itself can evolve over time.

CCS CONCEPTS

• **Information systems → Data stream mining**; • **Computing methodologies → Anomaly detection**; *Ensemble methods*;

KEYWORDS

Outlier Detection, Data Streams, Feature evolution

ACM Reference Format:
Aditi Singla, Ankush Phulia, and Vaibhav Bhagee. 2018. Outlier Detection in Data Streams: A Research Survey. In *Proceedings of Data Mining (COL761).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In data mining, the task of Outlier Detection is to identify those data points in a dataset which differ significantly from the rest of the points in the dataset and occur in rarity. There are various examples of scenarios where detecting outlier data samples is of use for eg. fraud detection in the financial sector, separation of "noisy" samples from the normal ones to refine the dataset for further processing, detection of "anomalous" event samples in system logs, to prevent large scale system failures etc.

Traditional algorithms have looked at outlier detection in scenarios where the dataset is "static", i.e the data does not change over the period of time and the algorithm has the access to the entire dataset, in memory. However, as the size of the datasets grows large, it might not be always easy to get the entire dataset in memory, for processing. Moreover, there are many scenarios like prevention of system failure, where the data samples like logs, are generated temporally, in a continuous fashion. In such cases, the outlier detection algorithm can never have access to the complete dataset and the analysis for outliers needs to be performed over the "seen" data.

As a part of this survey, we present and discuss various algorithms and techniques, which have been proposed in context of detecting outliers in the setting of streaming data. In particular, we discuss the challenges which are posed when detecting outliers in data streams what approaches are followed to overcome these.

COL761, July - November 2018, IIT Delhi

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 MOTIVATION

Outlier detection on large datasets has been traditionally looked at alongside clustering and density estimation. Many of the earliest outlier detection algorithms have been the ones performing clustering and in process identifying the "noisy" samples. However, in high dimensional data[2], the neighbourhood and clustering based outlier detection algorithms, face issues due to sparsity, failing to escape the curse of dimensionality. In order to avoid the distance computation among the points, exact and approximate algorithms have been proposed, based on projection of data onto smaller dimensions. The basic intuition behind this set of algorithms is that the outlier behaviour of a point is more pronounced in the subspaces of a high dimensional vector space. Hence, these algorithms are known to give better results than the former set of algorithms.

A key assumption which these algorithms make is that the entire dataset is available for processing, in memory. For smaller datasets, this assumption is decent enough to make. However, data has been growing at an exponential rate and modern data problems related to knowledge discovery in databases work on very large datasets, which can be of giga scale or tera scale in size. These datasets cannot be stored in the main memory for processing and therefore need to be chunked and processed. This is an example of a stream which is of finite size.

In addition to that, consider the problem of failure detection in systems, which has been described above. In such a typical modern day distributed system, there are multiple micro-services which send their logs to a central process responsible for collection and analysis of logs. In such a case, detection and analysis of log lines corresponding to "anomalous" events, only has access to the logs which have been collected by the process up to that instance of time. Hence, the outlier detection algorithm is looking at a stream of data which is potentially infinite in size.

Recent work[3] looks at streams which can evolve over time in multiple ways:

- Feature values of existing data points can change over time
- New unseen features can emerge over a period of time

These type of feature-evolving streams are common in scenarios like data center monitoring, where the jobs, which are submitted over time, can have feature values like number of unsuccessful retries, number of threads, system calls etc. and can have features,

which get added over time, like URLs of log files, repositories etc.

The existing algorithms and techniques for data streams fail to work for feature evolving streams as they make an assumption that the data streams cannot change dimensionality and the values of existing samples cannot be modified. The authors propose an efficient algorithm for this case, which generalises for the static and the row streams as well.

3 PROBLEM FORMULATION

The problem we are looking at essentially, is as follows: Given a data stream $D = \{x1, x2, ...\}$, we want to identify the data points, which are outliers. In the preliminary version of the problem, we assume the number of dimensions of every data item as a constant, say k.

There have been various techniques[6][5][1] which have been proposed to deal with outliers in streaming data. These techniques can be coarsely divided into - *Distance based*, *Ensemble based* and *Streaming based* techniques.

The outlier detection problem in streams can be extended to the feature-evolution setting, described as follows: Given a data stream D containing updates of the form (id, f, δ) where f could be a feature which has been seen or unseen until that instant and id is the unique identifier corresponding to the data point. δ is the update to the feature value of the data point as described above.

The key ideas behind these seminal contributions have been looked at in detail, in the next section.

4 RELATED WORK

4.1 Distance based outlier detection

Many of the early approaches towards anomaly detection in data streams, employ distance based methods. These distance based approaches take into account, the mutual distance of points, in the dataset, to identify outliers as a bi-product of clustering.

However, the outlier detection algorithms outlined above face issues due to sparsity and curse of dimensionality. In an attempt to formulate the algorithms to overcome the above mentioned issues, Aggarwal et al utilise the idea of projecting the data onto the smaller dimensions[2] and propose exact and approximate algorithms, based on the same.

A key challenge to address in this approach is to define the semantics associated with a point being an outlier, in a subspace of the original dataset, besides being computationally efficient.

Outliers in low dimensional projections

In order to overcome the shortcomings of a high number of dimensions, they project the data points onto a lower dimensional space. Every attribute in the dataset is divided into several equi-depth ranges, dividing the complete space into cubes of different dimensions.

Now, they select ranges from k different dimensions and consider the resulting k - dimensional cubes formed. The idea is to identify the most sparse of such k - dimensional cubes. In order to calculate how sparse the cubes are, Aggarwal et al define a metric, called the *sparsity coefficient*, to identify the extent of deviation in the actual number of points within a cube, from its probabilistic estimate.

Let us assume that every attribute is divided into r equi-depth ranges. Then, because of this being an equi-depth split, each range gets $f = 1/r$ fraction of the points. Now, is the distribution of points had been *independent* and $uniform$, then, a k - dimensional cube as described by the authors, would have contained a point with a probability f^k which is Bernoulli distributed.

Considering that the dataset consists of N points, the estimated number of points inside a k - dimensional cube becomes a Normal distributed random variable (by the application of the central limit theorem), with a mean $N.f^k$ and variance $N.f^k.(1 - f^k)$. Thus, the *sparsity coefficient* is defined as,

$$s = (n - N.f^k)/(N.f^k.(1 - f^k))^{0.5}$$

Given this, the problem finding outliers becomes that of finding the cubes with elements having the most negative sparsity coefficients. This search can be conducted in a simple brute force fashion or using evolutionary algorithms described by the authors and discussed below, in brief.

Evolutionary algorithms for Outlier Detection

The key idea here is to encode the information into strings and then use suitable crossover techniques to generate fitter and fitter strings, where the $fitness$ is defined using a suitable fitness function.

For every dimension, the grid range is encoded as a number between 1 and r, where r is the number of equi-depth ranges as described above, or a $*$ to denote a don't care. The sparsity coefficient is used as the fitness function, to evaluate the fitness of the cube, which is encoded cube represents. The sampling probability of a particular string, representing a cube, is a function of how negative is the sparsity coefficient for that cube.

The encoding and the fitness function, described above, are used in conjunction with a simple *two point crossover* technique where a random crossover point is chosen for 2 strings and their right parts are crossed over. In this way, the search is carried out until a fixed number of iterations or until the set of outliers stops changing for a specified number of pre-determined iterations.

The outlier detection algorithm described in this section essentially uses the idea of projection of points on lower dimensional subspaces while preserving the semantics of being an outlier. The idea of choosing the k - dimensional cube with the most negative sparsity coefficient, is in some way, an attempt to choose a set of

attributes out of the large number of dimensions, which best preserve the semantics associated with the outlier points.

The latter idea has been explored further as the *ensemble* based methods gained popularity in the machine learning community. The former idea has also been explored by various authors while proposing various outlier detection techniques based on *subspace partitioning*. Both these techniques have been discussed in significant detail, in the subsequent sections.

4.2 Ensemble based outlier detection

Model ensembling is a technique used in machine learning, which involves training multiple models over same or different subsets of data, having same or a subset of the features and then aggregating the results of these models to obtain a stronger classifier. The components of an ensemble can be trained in a way such that they could be dependent on each other, like sequential ensembles[4] or they could be independent of each other, like in bootstrap aggregating.

Both sequential and independent ensembles[4][1] have been employed for the task of outlier detection in data streams. In data stream setting, ensembles are useful to train multiple models for the anomaly detection task. Aggarwal et al describe the different categories of outlier ensembles and the key ideas behind their proposition[1].

Classification based on component independence

Outlier ensembles can be broadly classified on the basis of independence of the component models in the ensemble. The component models can both be *sequential* i.e depend on the previous model outputs for training and prediction, or can work in an *independent* fashion, from the other component models.

In *sequential ensembles*, the component algorithms for anomaly detection are applied in a sequential fashion, to incorporate the results of the algorithm which was previously applied. These type of ensembles begin with simpler classifiers and boost their accuracy by sequentially applying classification algorithms, to build stronger classifiers. This approach is popularly known as *boosting*, in machine learning literature. The final output of the ensemble can be computed in multiple ways. It can either be a weighted combination of the outputs of the component models, or just the output of the final model.

An example of a sequential ensemble is a two phase algorithm which the authors have outlined and discussed in the paper. In this approach, the first phase is to remove the data points which can be trivially considered as outlier points. The subsequent phase is to construct a more robust classifier for a refined outlier detection. Here, the output of the second model is considered as the ensemble output, as the second modeled works on a refined version of the dataset.

Another idea is to explore subspaces of data which are capable of pronounced discriminative behaviour. Once a subspace of data is identified to be sufficiently able to discriminate between the outlier data points from the other data points, it is recursively explored further. This is a natural setting where sequential ensembles are put to use, to identify discriminative subspaces of an already sufficiently discriminative subspace. Here however, the scoring function should incorporate the scores obtained from the models trained on the subspaces considered and hence is a combination of the outputs of the component models.

Independent ensembles are the ones where multiple independent component models are trained either on the full dataset or a subset of it, with their outputs being combined using a suitable method. The idea here is to use every component model to extract information related to a subset of the feature space, a subset of the data, or just train different models over the entire dataset and feature space and combine their results.

Classification based on constituent components

The ensembles can also be classified based on the working of their component models. As has been pointed out earlier, the component models could work on a subset of the data, a subspace of the entire feature space or the entire dataset and the feature space, where each of the component models are different.

Model centered ensembles are ones where multiple classification models are trained on the entire dataset and combine their scores. The idea here is to utilise the strengths of the different models to detect outliers and take incorporate the output of each model while calculating the final score. This type of an approach is popularly used to train a classifier where the same component algorithm is instantiated multiple times, with different parameters and run on the same dataset.

A key challenge in this scenario is combination of the outputs of the component models, to compute the final score. The model outputs need to be normalised and should have same semantic meaning, before the combination. The challenges have been further elaborated in the next section.

Examples of such ensembles include LOF methods, where multiple nearest neighbour classifiers are trained with a different neighbourhood size. Another example is the LOCI method, which makes use of a sampled neighbourhood, varying the sample size across the different component models, coupled with a $best - fit$ combination.

In *data centered* ensembles, subsets of the dataset are explored keeping the model same, across the ensemble components. The subsets can be constructed by either considering *horizontal partitions* of the dataset, where a subset of the data samples are considered for training by every model, or considering a *vertical partition* of the dataset, by considering a subset of features for all the data points. A popular example of such a partition is *feature bagging* where different feature subspaces are sampled using statistical methods and the LOF score is computed over each subspace. The scores are then combined using softmax, to assign the final outlier scores.

Normalisation and combination of model scores

An important component of the ensemble techniques is the normalisation technique used to normalise the component model scores. Normalisation is necessary as different models output scores which can range from probabilities to absolute scores where a higher score can mean that the corresponding sample is more likely to be an outlier, or vice versa.

An idea which the authors propose is to convert the model scores into probabilities, with the underlying assumption that the data points and the outliers are sampled from the same probability distribution, whose parameters can be learned using expectation maximisation, as is done in many unsupervised ML models, used for clustering, like Gaussian Mixture Models etc.

Lastly, the choice of combination function is also important based on the semantics associated with the component models of the ensemble. Popularly used combination functions include max, min, average, weighted average or the last model's output. Average functions are known to dilute the individual model scores and hence the authors caution against their use unless the component models are also selected keeping this consideration in mind.

4.3 Streaming based outlier detection

We have already discussed about the infinite length of the data streams which makes it infeasible to load the data in memory and perform anomaly detection. Besides that, the presence of outliers in streams is rare i.e most of the data samples in a stream are not outliers. An important observation here is that due this rarity of outliers, the scenario of training classifiers for anomaly detection can be considered as an semi supervised learning setting, where the training data or the labels corresponding to the outliers are not always available at the time of training.

The most important aspect which is addressed in this section is the idea of evolving streams, where the data evolves over time and hence might lead to a change in the definition of what properties make a data point qualify to be an outlier. Much of the work discussed previously does not address this issue related to time evolving data.

Tan et al, in their work[5], outline the construction of an ensemble of trees, called *streaming half space chains*, which perform semi supervised, one-class detection of the rare outliers. These attempt to detect outliers by capturing the density of points in multiple different partitions of the subspaces. As opposed to other outlier detection algorithms, this model processes the data in a single pass. Another advantage of this model is that the construction of trees happens before the data processing, essentially making this model data independent.

To incorporate the time evolving nature of the data, the authors introduce the concept of a window. Two windows called the *current* and the $reference$ windows are maintained to capture the time varying nature of the stream. At any given time, the $reference$ window is used to classify the incoming data samples into outliers or otherwise. However, the incoming samples are also maintained in the *current* window to capture the density profile of this incoming data. As the *current* window becomes full, it replaces the reference window and is subsequently used for classification, while the new *current* window resumes the task of learning.

Construction of Half Space Trees

A set of nodes forms a $Half\ Space\ Tree$ which is essentially a full binary tree. A dimension is randomly sampled at every node, belonging to its *work space* and then this work space is bisected into two half spaces. This processes continues recursively down the constructed half spaces, until the tree of the desired depth is created. The job of each node is to keep track of the *mass profile* of the data, in the work space which this node represents. Every node also maintains the mass profiles in the reference window and the current window, in variables called r and l respectively.

Before defining the *work space* corresponding to every node, the authors make an assumption that the attribute ranges are normalised to [0,1]. Given this assumption, let s be a number which is generated from the interval [0,1] $uniformly\ at\ random$. A $work\ range$ of $[s - 2.max(s, 1-s),\ s + 2.max(s, 1-s)]$ is defined using this sampled value of s. This is done for every dimension. This set of work ranges for every dimension constitutes the *work space* of a node. The key to the success of half space chains is that every half space tree should have a diverse set of work spaces for every node.

Anomaly score computation

The anomaly score for a half space tree is computed by considering the path which is traversed in the corresponding half space tree. The score at every node needs to be scaled according to the depth of the node in the tree as under the assumption of a uniform data distribution, such an exponential scaling would relate the mass profiles across nodes residing and different depths in the half space tree.

The final score is computed as $n.r * 2^{n.k}$ where $n.r$ is the reference window count and $n.k$ is the depth at a node n, which is a terminal node encountered in the path taken for a data point x in a half space tree T. While computing the score, the $n.l$ count which maintains the *current* window count at a node n encountered in a path from the root to a terminal node, also gets updated. The final ensemble score is a sum of the scores allotted by the individual half space trees.

4.4 Feature evolving data streams

All the algorithms discussed till now, work with data streams where the data points have a fixed dimensionality. The features may evolve over time, as has been discussed in the previous section, but the assumption prevents new features from emerging over time.

Manzoor et al in their recent work[3] discuss many real world applications where emergence of new features in data streams is

very natural. These applications have been discussed in earlier sections, to motivate outlier detection in data streams which are feature evolving.

Due to the feature evolving nature of the stream, the set of points need to be in memory all the time. Given the fact that new features can be added, the space allocation for the data points can't be bounded before hand and hence is challenging. Also, the feature updates need to be fast and efficient.

The algorithm proposed by the authors is a constant space and constant time per update algorithm which combines ideas from subspace partitioning and ensembling, while incorporating the feature evolving nature of the input stream.

Stream Hash

Dimensionality reduction of the feature space has been at the core of many data mining algorithms to get around the dimensionality curse. Among the popular techniques used for the same, distance preserving techniques like *Random projections* are a fast and an efficient way, to project the data onto a smaller feature subspace.

In *Random Projections*, the data points are projected onto a dimension of size K, by considering a set of K Gaussian random vectors. Thus, for a d - dimensional data point x, a set of Gaussian random vectors $\{r_1, r_2 ... r_K\}$ is chosen and the embedding in K - dimensions is computed as $y = (x^T r_1, x^T r_1 ..., x^T r_K)$. For the task of outlier detection, we attempt to obtain projections across multiple small feature subspaces, by selecting the random vectors where only *one third* of the features have a non zero value. This helps us to look for outliers which are embedded in low dimensional subspaces, which otherwise would have been difficult to obtain, due to irrelevant features having a non zero value.

A major issue in feature evolving streams is the fact that new features could emerge with time and so, the actual dimensionality of the data stream is not known. To get around this problem, the authors propose replacing Gaussian random vectors with hash functions. Consider a family h_i of hash functions. We consider K hash functions $h_1(.), h_2(.) .... h_K(.)$, from this family. Every hash function maps a $feature$ to a real value. The hash functions are constructed to maintain the probability distribution of a sparse random vectors, as discussed above

Now, the projection y of the point x, becomes,

$$y[i] = \sum_j h_i(f_j)x[j]$$

. The hash function helps in associating non zero values with unseen features. Considering updates of the form (id, f, δ), as described in the problem formulation, the computation of the projection y becomes

$$y_{id}[i] += h_i(f)\delta$$

.

Half space chains

Once the projections are computed in the lower dimensional subspace, the authors use streaming half space chains, as discussed in the previous section, to learn an ensemble of semi supervised classification models, for outlier detection. The construction of half space chains and the scoring function remain the same as discussed before.

Non-stationarity of evolving point

In order to take care of the time evolving data, the idea of keeping a *current* and a $reference$ window is used. As discussed before, the $reference$ window parameters are used for outlier detection while the *current* window parameters are computed in a streaming fashion.

Caching of data points

Due to the emergence of new features, a data point might see a feature update some time later in future, after it has been processed. Thus, earlier we could get rid of the data points which have been processed but now, we can not discard those points.

For this, the authors propose a fixed size *LRU* cache to keep track of the processed points. When an incremental update is processed, the corresponding point is removed from the half space chain and then the updated point is added back by suitable modification of the mass profiles in the half space trees. Also, the position of the point in the *LRU* cache is updated, to the head of the list.

The authors use this algorithm to solve the outlier detection problem in static and streaming settings with a constant feature space dimensionality and evaluate its performance. The algorithm is able to match the performance of the state of the art, in both cases.

5 LIMITATIONS OF EXISTING WORK

Up till now, we have looked at various algorithms for outlier detection for streaming data. We have also looked at an efficient algorithm for the setting where the data stream is feature evolving. Most of the above mentioned approaches have considered the outlier detection task to be a two-class classification problem. The setting however is semi-supervised, due to to the rarity of outlier samples at train time.

For a given data sample, all the features do not contribute equally towards it being an outlier or not. For a set of data, only a subset of the features may be able to determine the outliers. Moreover, in the case when the data is changing over time or new features are evolving, this subset of features may change over time.

For the models to be widely adopted, the classification of a point as an outlier should be accompanied by a *reason* or an *explanation* of why this point has been classified as an outlier. As the data analytics and machine learning communities push harder for explainable

models, such a feature selection becomes even more relevant. Consider for eg. a task to detect anomalous credit card transactions for fraud detection. In such a case, the anomaly detection algorithm can reinforce its prediction with a proof, which can essentially be the discriminative features and their values, to add more semantic relevance.

While the above algorithms take care of the time evolving data and the feature evolving nature of the stream, the algorithms do not help in identifying and considering only the relevant features, which help in discriminating outliers from the non outliers.

6 PROPOSED PROBLEM FORMULATION

Given a data stream D containing updates of the form (id, f, δ) where f could be a feature which has been seen or unseen until that instant and id is the unique identifier corresponding to the data point. δ is the update to the feature value of the data point as described above.

We need to train a model to detection outliers from this stream D. Besides this, the model should also be able to output a relevance score for the features. Finally, the model should expose and API where we should be able to ask the model to perform outlier detection using the top k relevant features. Here k is the input and a high relevance corresponds to a hight feature relevance score.

7 CONCLUSION

We have looked at the problem of finding outliers in a stream of data and given a mathematical formulation of the same. In this process, we have also outlined the associated challenges with mining of data streams including but not limited to memory constraints, efficiency and time evolving data.

We have discussed various key techniques which have been proposed, over the years, for detecting outliers in data streams. These techniques borrow ideas from statistics as well as machine learning and can be broadly classified into - *distance based*, *ensemble based* and *streaming based* methods.

An important limitation of the above methods is that they do not work in the setting where new features could emerge in the data samples over time. Towards the end, we formulate the problem of outlier detection in data streams which are feature-evolving and discuss about $x'Stream$, a solution which overcomes the limitations of the existing methods.

It proposes a constant time per update algorithm for outlier detection. The algorithm matches the state of the art in the static and streaming data setting, where the feature space has a fixed dimensionality.

REFERENCES

[1] Charu C Aggarwal. 2013. Outlier ensembles: position paper. *ACM SIGKDD Explorations Newsletter* 14, 2 (2013), 49–58.
[2] Charu C Aggarwal and Philip S Yu. 2001. Outlier detection for high dimensional data. In *ACM Sigmod Record*, Vol. 30. ACM, 37–46.
[3] Emaad Manzoor, Hemank Lamba, and Leman Akoglu. 2018. xStream: Outlier Detection in Feature-Evolving Data Streams. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD '18)*. ACM, New York, NY, USA, 1963–1972. https://doi.org/10.1145/3219819.3220107
[4] Shebuti Rayana, Wen Zhong, and Leman Akoglu. 2016. Sequential Ensemble Learning for Outlier Detection: A Bias-Variance Perspective. *CoRR* abs/1609.05528 (2016). arXiv:1609.05528 http://arxiv.org/abs/1609.05528
[5] Swee Chuan Tan, Kai Ming Ting, and Tony Fei Liu. 2011. Fast anomaly detection for streaming data. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, Vol. 22. 1511.
[6] Luan Tran, Liyue Fan, and Cyrus Shahabi. 2016. Distance-based outlier detection in data streams. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1089–1100.