pyISC: A Bayesian Anomaly Detection Framework for Python

Blerim Emruli
RISE SICS Västerås
SE-722 13 Västerås, Sweden
blerim@sics.se

Tomas Olsson and **Anders Holst**
RISE SICS
SE-164 29 Kista, Sweden
tol@sics.se, aho@sics.se

Abstract

The pyISC is a Python API and extension to the C++ based Incremental Stream Clustering (ISC) anomaly detection and classification framework. The framework is based on parametric Bayesian statistical inference using the Bayesian Principal Anomaly (BPA), which enables to combine the output from several probability distributions. pyISC is designed to be easy to use and integrated with other Python libraries, specifically those used for data science. In this paper, we show how to use the framework and we also compare its performance to other well-known methods on 22 real-world datasets. The simulation results show that the performance of pyISC is comparable to the other methods. pyISC is part of the Stream toolbox developed within the STREAM project.

1 Introduction

Anomaly detection usually refers to the automatic detection of events or behaviors, which substantially deviate from those considered normal. Typically, anomaly detection is an unsupervised process, since it makes use of unlabeled data from previous instances in order to estimate a statistical model of normal behavior, and then compares current observations against that model. The key assumption is that the non-anomalous data points are substantially more common than the anomalous ones, and thus it is possible to distinguish the anomalous data points from the non-anomalous ones. Anomaly detection usually encompasses outlier detection and change detection, and it is closely related to clustering and forecasting methods (Papadimitriou 2009).

Identifying anomalies is important in a broad range of disciplines, among others: industrial process monitoring; condition-based monitoring; autonomic management of IT services, systems, and infrastructure; surveillance; and medical diagnosis and prognosis. Anomalies are generally categorized into three types: i) point anomalies where each data point is analyzed individually compared to all data points, ii) collective anomalies where data points are analyzed together in groups, and iii) context anomalies where data points are analyzed with respect to a context. The anomaly detector presented in this paper is meant to be used for analyzing point anomalies. Usually anomaly detectors either output an anomaly score that measures the anomalousness of data or classify into normal or anomalous data. Both approaches are considered in this paper.

Since 2001, our institute during several basic and applied research projects has applied, further developed, and fine-tuned anomaly detection methods. The findings resulted in a method called the Bayesian Principal Anomaly (BPA), which for several years has been successfully used and evaluated in several real-world applications. For example, intrusion detection (Dey 2009), maritime domain awareness (Holst et al. 2012a), condition-based monitoring (Holst et al. 2012b), and predictive policing (Holst and Bjurling 2013). Initially, the BPA method was implemented in C++ and the framework was coined Incremental Stream Clustering (ISC), from which the name of the presented framework derives, pyISC. pyISC is designed to be easy to use and easy to integrate with other data-driven libraries in Python, which currently is a very popular programming language for data science.

In the rest of the paper, we first introduce some related work on anomaly detection. Then, we present some theoretical perspective and necessary equations that motivate the BPA method. Next, we systematically describe the pyISC framework. Then, in the experiment section, we present novel experimental results of BPA method through pyISC on real-world datasets. The last section ends the paper with a brief evaluation of the framework and the results obtained from the experiments, and elicits some opportunities and challenges for future work.

2 Related Work

Anomaly detection is an important problem that has been researched within diverse areas, spanning from statistics and machine learning to signal processing and information theory. There are several ways how to categorize the anomaly detection methods (Chandola, Banerjee, and Kumar 2009; Lavin and Ahmad 2015).

For sake of brevity, here we identify two key categories: the ones that assume that the data follow some particular statistical distribution, and the ones that do not hold any assumptions regarding the statistical distribution. The methods embracing the later are often presented as non-parametric (Subramaniam et al. 2006). These methods are general and flexible, since they do not assume or require a prior knowledge about the input data. However, when a parametric form is known, the correct parametric method will require much less data than the non-parametric methods in order to be useful and applicable to real-world scenarios (Holst et al. 2012b). This is key for many industrial scenarios and applications where obtaining, recording, and continuously communicating a large amount of data (Big Data) are true challenges. Moreover, event-based data approximately follow a Poisson distribution. Following this assumption, a parametric approach is certainly best suited for anomaly detection in several industrial applications (Dey 2009; Holst et al. 2012a; 2012b; Holst and Bjurling 2013).

For completeness, we include below some open-source frameworks for anomaly detection in time series. Etsy, a peer-to-peer e-commerce website, received critical acclaim by releasing the Skyline framework as an open-source project for detecting anomalies in streaming data (SkylineAD 2013). Skyline is written in Python and is no longer actively maintained. In 2015, Yahoo! and Twitter released their own open-source frameworks for anomaly detection. Yahoo's EGADS was built as a framework to be easily integrated into an existing monitoring infrastructure and that is completely written in Java (Laptev, Amizadeh, and Flint 2015). On the other hand, Twitter's framework monitors user engagement and aims to perform robust anomaly detection during breaking news, tweets, sporting events, holiday seasons, and so forth. Twitter's framework has been also used in commercial settings and it is written in R (Kejariwal 2013).

The focus of this paper is to briefly introduce the pyISC framework and present some novel experimental results using it. A deeper theoretical motivation and experimental demonstration of pyISC in comparison to other open source frameworks is outside the scope of this paper.

3 Bayesian Principal Anomaly

The main idea is to build a statistical model over normal instances (assumed to be substantially more), and then compare each incoming instance against that model. The assumption is that the parametric form of the normal instances is known, but we need to estimate its parameters from the observed data. Thus, we use Bayesian statistics to find the predictive distribution of the normal instances, and then, for a new observation, calculate the probability of getting something at least as unusual from that distribution. If the probability is sufficiently low, the observation is considered anomalous, or at least worth to pay extra attention.

The initial assumption is that the normal instances are generated by a known probability density $p(x|\theta)$ for a set of parameters θ. By definition, the smaller the probability of generating a new observation z from the distribution, the more anomalous is it considered. Thus, the principal anomaly of a new observation z is the probability of generating more common instances than z from the distribution:

$$A(z|\theta) = \int_{x\in\Omega} p(x|\theta), \qquad \Omega = \{x : p(x|\theta) > p(z|\theta)\}. \tag{1}$$

Figure 1 illustrates the principal anomaly. A desirable property of the principal anomaly is that it is directly connected to the rate of false alarms. For example, if we set a threshold on the principal anomaly of $1-\epsilon$ over which an observation is judged anomalous, the probability that a normal sample is wrongly detected as anomalous is then simply ϵ. In the example show in Figure 1, new observation z's probability $p(z|\theta)$ is low, therefore $A(z|\theta)$ would be considered as potential anomaly if a threshold for $A(z|\theta)$ is low enough.

The principal anomaly, although suitable in the formal sense, is not so convenient to work with since its anomaly values are most of the time very close to 1. Thus, a more useful and intuitive measure to work with, is the negative logarithm of the complement of the principal anomaly:

$$\Lambda(z|\theta) = -log(1 - A(z|\theta)), \tag{2}$$

which ranges from 0 to ∞ and increases with higher anomaly, such that each constant step higher represents a factor lower probability.

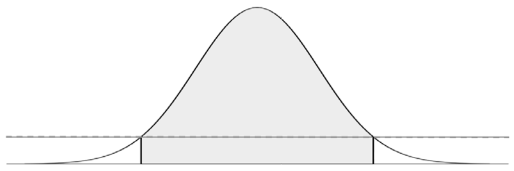
Figure 1: The shaded area refers to the principal anomaly $A(z|\theta)$, and the horizontal line refers to $p(z|\theta)$.

In the above example, it is assumed that the parameters θ of the statistical model are known. In practice, however, we usually do not know the parameter values of the model in advance, but all we have is a number of observations that is assumed to be generated by the model. Thus, let us now consider the case when the parameters θ are unknown and we have to estimate the distribution from a number of samples. More formally, the Bayesian approach finds the posterior distribution over the parameters θ given the set of training samples X:

$$P(\theta|X) \propto P(X|\theta)P(\theta) = \prod_i P(x_i|\theta)P(\theta). \tag{3}$$

Here, $P(\theta)$ is the prior distribution over the parameters. In general, we use a standard non-informative prior. Using this reasoning, we can now obtain the principal anomaly by integration over all possible parameter values:

$$A(z|X) = \int_\theta A(z|X)P(\theta|X), \tag{4}$$

which is defined as the Bayesian Principal Anomaly (Holst and Ekman 2011).

4 The pyISC Framework

The pyISC framework is a Python extension and wrapper of the ISC Framework that is implemented in C++ (Holst and Ekman 2011). The framework is available as open source at http://github.com/stream3/pyisc.

By linking the ISC framework to Python a wide set of advantages are gained. First, the ISC framework becomes much more accessible to the growing data science community that uses Python. Specifically, we have made an effort to make it compatible with the scikit-learn, which is widely regarded as the most used open source Python library for machine learning (Pedregosa et al. 2011). Similar to scikit-learn, the fundamental data structure for pyISC is the NumPy array, which underlies almost all numerical scientific computation in Python. Thus, as a consequence, many other great data science tools are also made available to use with ISC. One example is the Pandas library for managing and analyzing data (McKinney 2010). Next, Python has a less steep learning curve than C++. Third, through the dependency management systems, such as pypi and the Anaconda[1] or Enthought distributions[2], Python libraries are easier to distribute to many different platforms compared to C++ software. Fifth, and lastly, Python makes it easier to add new functionality using the ISC components as building blocks, which, if needed, can be transferred into C++. Thus, pyISC may develop towards the direction of adding more complex functionality, while we focus on keeping the ISC framework more lightweight and self-contained in order to target (resource constrained) embedded systems.

There are currently two type of detections supported in the latest version of pyISC: (1) anomaly detection, where the output is an anomaly score for each data point, and (2) outlier detection where a fraction of outliers (contaminations) are known beforehand and the output is a prediction of whether a data point is an outlier or not.

Besides being based on Bayesian inference, what sets pyISC apart from other toolkits and libraries is the easiness to combine different probability distributions to model multivariate data of different types. For example, instead of using only a multivariate Gaussian distribution, we can easily combine one multivariate Gaussian distribution, a simple Gaussian distribution and a Poisson distribution, depending on the variable types. Currently, pyISC supports the Gaussian distribution, multivariate Gaussian distribution, two-sided and one-sided Poisson distributions and an approximation to the Gamma distribution.

In the following sections, first we show how to create and use an anomaly detector, then an outlier detector, and in the final section, we briefly outline the logic of combining more than one type of probability distribution.

Anomaly Detection

An anomaly detector is constructed from a set of component distributions that defines the models used by the anomaly detector. Then, to train the anomaly detector, the *fit* method is called with some training data, and anomaly scores are computed with the *anomaly_score* method. Below, we show how to create and train a bivariate Gaussian distribution and how to compute anomaly scores:

```
import numpy as np
import pyisc
# Get some data:
X = [[20, 4], [1200, 130], [12, 8], [27, 8], [-9,
     13], [2, -6]]

# Create an anomaly detector where the numbers are
     column indices of the data:
anomaly_detector = pyisc.leftmarginAnomalyDetector(
    pyisc.leftmarginP_Gaussian([0,1])
)

# The anomaly detector is trained
anomaly_detector.leftmargin fit(np.array(X))

# Then, we can compute the anomaly scores for the
     data:
anomaly_detector.leftmarginanomaly_score(np.array(X))

# The result is anomaly scores (with two decimal
     precision):
array([ 0.10,  1.08,  0.10,  0.05,  0.67, 0.77])
```

By comparing the values in the list, the second elements easily stands out as the “most anomalous”.

Similarly, we can create an anomaly detector with the Gamma or Poisson distributions where the numbers are the column indices into the input data:

```
pyisc.leftmarginP_Gamma(frequency_column=0,
     period_column=1)

pyisc.leftmarginP_Poisson(frequency_column=0,
     period_column=1)
```

In case we have more than one known class of data points, it is also possible to train the detector to make a separate model for each class. In this case, if y is an array with two or more class labels, the anomaly detector can still be similarly trained and likewise compute the anomaly scores:

```
anomaly_detector.leftmargin fit(X,y)

anomaly_detector.leftmarginanomaly_score(X,y)
```

Outlier Detection

In a similar fashion as when we create an anomaly detector, we can create an outlier detector. The outlier detector differs from the anomaly detector in that a fraction of outliers (contaminations) are known beforehand and the output is a prediction of whether a new data point is an outlier or not. Consequently, the outlier detector can dynamically select a threshold to decide when a data point is an outlier or inlier from the training data. Below, we use the same data set as in previous section but now we know that there is one anomalous data point “an outlier” and five inliers in the data set:

```
import numpy as np
import pyisc

# Data with an outlier in element 2:
X = [[20, 4], [1200, 130], [12, 8], [27, 8], [-9,
     13], [2, -6]]
```

[1]https://www.continuum.io

[2]https://www.enthought.com

```
# Create an outlier detector with the known fraction
    of outliers: 1 of 6:
outlier_detector = pyisc.
    SklearnOutlierDetector(
    contamination=1.0/len(X),
    component_models=pyisc.P_Gaussian([0,1])
)

# The outlier detector is trained
outlier_detector.fit(np.array(X))

# Then, the data is classified into being outliers
    or not:
outlier_detector.predict(np.array(X))

# The result is classification of outliers (-1) and
    inliers (1):
array([ 1,  -1,  1,  1,  1, 1,  1])
```

Thus, we are able to detect the second element as an outlier. The outlier detector follows the API used in scikit-learn for outlier detection with known contamination.[3]

Combining Distributions

A feature that makes pyISC different from other machine learning libraries is the possibility to combine different distributions when creating a detector. This is achieved by providing a list of distributions instead of only providing a single distribution to the constructor.

For example, we can create an anomaly detector that combines a one-sided Poisson distribution, a two-sided Poisson distribution, and a Gaussian distribution like this:

```
anomaly_detector = pyisc.AnomalyDetector(
    component_models=[
        P_PoissonOnesided(1,0),
        P_Poisson(2,0),
        P_Gaussian(3)
    ],
    output_combination_rule=pyisc.cr_max
)
```

Moreover, there are available combination methods, such as: *cr_max*, *cr_min*, and *cr_plus*; where *cr_max* returns the maximum of the anomaly scores from each probability distribution, *cr_min* returns the minimum, while *cr_plus* adds all anomaly scores together.

5 Experiments

In the following sections, we will compare the pyISC framework with the standard methods for anomaly detection and outlier detection on real-world datasets.

Anomaly Detection

For testing the anomaly detection performance we used the same experimental setup described previously (Quinn and Sugiyama 2014), where the authors present the least-square anomaly detection algorithm (LSAD) and compares it with the one-class SVM (OCSVM) (Schölkopf et al. 1999), the distance of the k-nearest neighbor (KNN) (Chandola, Banerjee, and Kumar 2009) and the distance to the closest cluster center using k-means (KM) (Chandola, Banerjee, and Kumar 2009). We also add the isolation forest detector (IF) (Liu, Ting, and Zhou 2012) and the pyISC anomaly detector (PAD) to the compared anomaly detectors. The implementations (except LSAD and PAD) are the taken from the scikit-learn library (Pedregosa et al. 2011).

We have followed same initialization as in a previous study (Quinn and Sugiyama 2014), by using the default values from the LSAD implementation, OCSVM had its default values except for gamma (the kernel width) that was set to the same as the gamma parameter of the LSAD (automatically set), KNN used k=10 nearest neighbors (or the number of data points if less than 10), and KM used 20 clusters (or the number of data points if less than 20). For IF we also used the provided default values, while for the PAD, we used a simple, but scalable, set up with a single Gaussian distribution for each data feature and the combination rule was set to adding the resulting anomaly scores (*cr_plus*). Note that pyISC supports multivariate Gaussian distributions, however, since high-dimensional data will result in large covariance matrices, we decided to use only univariate distributions in the experiments and despite this impediment achieved acceptable results.

We also used 22 real-world datasets from the libsvm website[4] for evaluation (Quinn and Sugiyama 2014). For each dataset, we used the first class (class labels are numerically ordered from small to large values) as inliers and we used the second class as outliers, while the remaining classes were ignored. We evaluated the methods using stratified five-fold cross-validation that keeps the number of inliers and outliers equal in each fold. For each fold, we first trained the methods using only the inliers from the training set, and then we computed the anomaly scores on the test set.

The area under the curve (AUC) scores were then averaged over the five folds for each dataset and method. For LSAD, OCSVM, KNN and KM, the data was normalized using the training data to have mean 0.0 and standard deviation 1.0. Normalization did not affect the performance of IF or PAD.

The resulting mean AUC scores are shown in Table 1, where the bold indicates when there is a maximum AUC score and cursive indicates that there is a significant difference between a score and the maximum score across the methods. As significance test, we used the paired t-test with significance level $p = 0.05$. As can be seen, the performance of the methods varies a lot over the datasets, so there is no single best method for all datasets. Table 2 shows a summary of the performance of the anomaly detectors in terms of the number of datasets where they have the highest mean AUC and the number of instances where there is no statistical significance between the performance and the highest AUC. Obviously, OCSVM is a clear winner. OCSVM has the 7 highest mean AUC, followed by KM with 6 and KNN and PAD with 5 highest mean AUC. The OCSVM total is 20 followed by LSAD and KM that have 17, while KNN

[3]http://scikit-learn.org/stable/modules/outlier_detection.html

[4]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

Table 1: The mean AUC for each tested anomaly detection method.

Dataset			Method					
	d	N	LSAD	OCSVM	KNN	KM	IF	PAD
australian	14	690	*0.815*	0.839	0.845	*0.816*	**0.877**	0.848
breast-cancer	10	683	0.982	0.987	0.987	0.988	**0.995**	0.992
cod-rna	8	59535	*0.792*	0.838	**0.841**	*0.821*	*0.685*	*0.654*
colon-cancer	2000	62	0.719	**0.812**	0.750	**0.812**	0.700	0.700
diabetes	8	768	0.530	0.533	*0.439*	0.532	**0.568**	0.528
dna	180	1532	0.941	0.944	**0.970**	0.962	*0.649*	0.942
duke	7129	44	0.800	0.750	0.680	**0.812**	0.688	0.688
gisette	5000	7000	0.791	*0.804*	0.814	0.827	*0.498*	**0.828**
glass	9	146	0.805	*0.838*	*0.648*	**0.848**	0.714	0.671
heart	13	270	0.773	**0.836**	0.824	0.782	0.800	0.829
ijcnn1	22	49990	*0.727*	0.730	**0.732**	*0.628*	*0.582*	*0.576*
ionosphere	34	351	*0.221*	0.321	0.339	0.240	0.378	**0.401**
letter	16	1161	0.999	**0.999**	*0.993*	0.996	*0.960*	*0.917*
leu	7129	72	**0.940**	0.933	**0.940**	0.933	*0.533*	0.880
mnist	780	14780	0.946	0.955	0.959	*0.949*	0.965	**0.969**
mushrooms	112	8124	*0.985*	**0.999**	*0.981*	*0.986*	*0.908*	*0.971*
pendigits	16	1559	**1.000**	1.000	0.999	0.999	0.996	1.000
satimage	36	1551	1.000	**1.000**	1.000	**1.000**	*0.999*	*0.996*
sonar	60	208	**0.699**	0.591	*0.525*	0.668	*0.618*	0.658
usps	256	2199	0.991	0.991	0.994	0.994	*0.988*	**0.995**
vowel	10	180	0.997	**1.000**	*0.741*	**1.000**	*0.639*	*0.435*
wine	13	130	**1.000**	**1.000**	**1.000**	**1.000**	*0.964*	**1.000**

Table 2: The table shows the number of datasets that had the highest mean AUC for each method and number of times its AUC was not significantly different (at $p = 0.05$). The last row shows the total number of datasets with highest or not significantly different value.

	LSAD	OCSVM	KNN	KM	IF	PAD
Highest AUC	4	7	5	6	3	5
Not significant	13	13	11	11	7	11
Total	17	20	16	17	10	16

and PAD have 16. It would not be surprising that the non-parametric methods perform better than PAD because of its stronger assumptions of independent Gaussian distributed data. However, we can conclude that PAD performs rather well despite that the "true" data distribution is not modeled.

Outlier Detection

For the outlier detection evaluation, we again compared a set of methods on the same datasets as for the anomaly detection. However, this time we assume that the training data contains up to 25% outliers, but depending on the number of actual outliers in the training data, it can be a smaller fraction. The compared methods are the one-class SVM (OCSVM) (Schölkopf et al. 1999), isolation forest (IF) (Liu, Ting, and Zhou 2012), robust covariance estimation for robust elliptical Gaussian fitting (ELLIP) (Rousseeuw and Driessen 1999) and the pyISC outlier detector (POD). All outlier detectors are from the scikit-learn library (Pedregosa et al. 2011), except POD.

The OCSVM was initialized with default parameters and by setting $nu = 0.95 * outlier\_fraction + outlier\_fraction/5.0$. For the IF and ELLIP, the contamination parameter was set to the outlier fraction while the remaining parameters had their default values. Similar to the anomaly detector, the POD was initialized with a single Gaussian distribution for each feature in the dataset and the rule of combination was set to adding the resulting anomaly scores, while the contamination parameter was again set to the outlier fraction. Similarly to the anomaly detection evaluation, we use stratified five-fold cross-validation. For each fold, we trained the method using all the inliers in the training set but this time we also added outliers from the training set up to 25% of the used training data, and then, we evaluated the method on the test set. Each method was trained knowing the true fraction of outliers in the training data. As evaluation metric, we have used the accuracy of classifying the test set into inliers and outliers. Table 3 shows the mean classification accuracy from the evaluation. As before, bold indicates single maximum scores and cursive indicates that there is a significant difference between the score and the maximum score.

Table 3: The mean classification accuracy for each tested outlier detection method.

Dataset			Method			
	d	N	OCSVM	ELLIP	IF	POD
australian	14	690	*0.446*	–	0.681	**0.716**
breast-cancer	10	683	*0.351*	*0.697*	**0.963**	*0.927*
cod-rna	8	59535	**0.646**	*0.589*	*0.599*	*0.574*
colon-cancer	2000	62	0.485	0.390	**0.645**	0.613
diabetes	8	768	**0.651**	*0.359*	*0.370*	*0.361*
dna	180	1532	*0.514*	–	*0.514*	**0.806**
duke	7129	44	*0.539*	*0.378*	0.567	**0.679**
gisette	5000	7000	*0.505*	–	*0.470*	**0.654**
glass	9	146	**0.629**	–	0.554	0.473
heart	13	270	*0.444*	–	0.656	**0.700**
ijcnn1	22	49990	*0.820*	–	0.829	**0.829**
ionosphere	34	351	0.245	–	**0.308**	0.253
letter	16	1161	*0.566*	**0.793**	*0.633*	0.745
leukemia	7129	72	**0.583**	*0.223*	0.515	0.582
mnist	780	14780	**0.533**	–	*0.340*	*0.344*
mushrooms	112	8124	*0.657*	–	*0.780*	**0.844**
pendigits	16	1559	*0.500*	*0.960*	*0.747*	**0.992**
satimage	36	1551	*0.756*	**0.990**	*0.923*	*0.959*
sonar	60	208	**0.564**	0.505	0.466	*0.448*
usps	256	2199	**0.572**	–	*0.365*	*0.307*
vowel	10	180	0.500	**0.550**	0.489	**0.550**
wine	13	130	*0.768*	*0.732*	*0.830*	**0.922**

We used the paired t-test for testing statistical significance with significance level $p = 0.05$. The ELLIP method ran into problems with several datasets where it did not manage to fit the data, which is indicated with "–". Similar to the evaluation of anomaly detection, Table 4 shows a summary of the results by counting the number of highest mean accuracy scores, the number of scores not significantly different from the highest score, and their total. This time we have a clear winner in the POD that has the highest score for 10 datasets and non-significantly different for 5, thus in total 15. The closest IF with a total of 11 and OCSVM with 10. Thus, we can conclude that POD is best among the compared outlier detectors.

6 Summary and Conclusions

This paper has presented an open source Bayesian anomaly detection framework called pyISC.

pyISC is a Python API and extension to the C++ based Incremental Stream Clustering (ISC) anomaly detection and

Table 4: The table shows the number of datasets where each method had the highest mean accuracy value and number of times the accuracy value was not significantly different (at $p = 0.05$). The last row shows the total number of datasets with highest or not significantly different values.

	OCSVM	ELLIP	IF	POD
Highest AUC	7	3	3	10
Not significant	3	2	8	5
Total	10	5	11	15

classification framework (Holst and Ekman 2011). For completeness, we briefly included some related work. Next, we described the Bayesian Principal Anomaly (BPA) method that underlies pyISC and uses Bayesian statistical inference to provide a probability-based anomaly score. Third, we compared the pyISC anomaly detector to a set of standard methods and other available methods as Python implementations. We concluded that pyISC detectors had comparable performance to most of the other methods when doing anomaly detection using the AUC performance metric and better performance doing outlier detection when a fraction of outliers is known in the training data. The results are rather surprising since the other methods do not assume any particular probability distribution for the data and we did not select a statistical model that would fit the data well. Instead, we used a naive assumption of a univariate Gaussian distribution for each attribute, and thereby, the attributes are independent. This indicates that there is a great potential in improving the performance by selecting distributions that fit the data well. Thus, a potential future work would be to automatically select probability distributions for a dataset and thereby improve the performance of pyISC.

7 Acknowledgements

This work is funded by VINNOVA, grant number 2012-01277, together with Atlas Copco, Örebro and ABB Crane Systems, Västerås. Authors would like to thank the personal at Atlas Copco and ABB Ports for their help during the research.

References

Chandola, V.; Banerjee, A.; and Kumar, V. 2009. Anomaly detection: A survey. *ACM computing surveys (CSUR)* 41(3):15.

Dey, C. 2009. *Reducing IDS False Positives Using Incremental Stream Clustering (ISC) Algorithm.* MSc thesis, Department of Computer and Systems Sciences, Royal Institute of Technology, Sweden.

Holst, A., and Bjurling, B. 2013. A bayesian parametric statistical anomaly detection method for finding trends and patterns in criminal behavior. In *Intelligence and Security Informatics Conference (EISIC), 2013 European*, 83–88. IEEE.

Holst, A., and Ekman, J. 2011. Incremental stream clustering for anomaly detection and classification. In *Proceedings of the 11th Scandinavian Conference on Artificial Intelligence*, 100–107. IOS Press.

Holst, A.; Bjurling, B.; Ekman, J.; Rudström, Å.; Wallenius, K.; Björkman, M.; Fooladvandi, F.; Laxhammar, R.; and Trönninger, J. 2012a. A joint statistical and symbolic anomaly detection system: Increasing performance in maritime surveillance. In *Information Fusion (FUSION), 2012 15th International Conference on*, 1919–1926. IEEE.

Holst, A.; Bohlin, M.; Ekman, J.; Sellin, O.; Lindström, B.; and Larsen, S. 2012b. Statistical anomaly detection for train fleets. *AI Magazine* 34(1):33.

Kejariwal, A. 2013. Twitter anomaly detection. https://github.com/twitter/AnomalyDetection. [Online; accessed 15-11-2016].

Laptev, N.; Amizadeh, S.; and Flint, I. 2015. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1939–1947. ACM.

Lavin, A., and Ahmad, S. 2015. Evaluating real-time anomaly detection algorithms–the numenta anomaly benchmark. In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA)*, 38–44. IEEE.

Liu, F. T.; Ting, K. M.; and Zhou, Z.-H. 2012. Isolation-based anomaly detection. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 6(1):3.

McKinney, W. 2010. Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference*, 51 – 56.

Papadimitriou, S. 2009. Anomaly detection on streams. In *Encyclopedia of Database Systems*. Springer. 88–90.

Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; Vanderplas, J.; Passos, A.; Cournapeau, D.; Brucher, M.; Perrot, M.; and Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* 12:2825–2830.

Quinn, J. A., and Sugiyama, M. 2014. A least-squares approach to anomaly detection in static and sequential data. *Pattern Recognition Letters* 40:36–40.

Rousseeuw, P. J., and Driessen, K. V. 1999. A fast algorithm for the minimum covariance determinant estimator. *Technometrics* 41(3):212–223.

Schölkopf, B.; Williamson, R.; Smola, A.; and Shawe-Taylor, J. 1999. Sv estimation of a distribution's support. *Advances in neural information processing systems* 12.

SkylineAD. 2013. Etsy Skyline. https://github.com/etsy/skyline. [Online; accessed 15-11-2016].

Subramaniam, S.; Palpanas, T.; Papadopoulos, D.; Kalogeraki, V.; and Gunopulos, D. 2006. Online outlier detection in sensor data using non-parametric models. In *Proceedings of the 32nd international conference on very large data bases VLDB*, 187–198.