Exchangeability Martingales for Selecting Features in Anomaly Detection

Giovanni Cherubin GIOVANNI.CHERUBIN.2013@LIVE.RHUL.AC.UK
Royal Holloway University of London, UK
Adrian Baldwin ADRIAN.BALDWIN@HP.COM
Jonathan Griffin JONATHAN.GRIFFIN@HP.COM
HP Labs Security Lab, Bristol, UK

Editors: Alex Gammerman, Vladimir Vovk, Zhiyuan Luo, Evgueni Smirnov and Ralf Peeters

Abstract

We consider the problem of feature selection for unsupervised anomaly detection (AD) in time-series, where only normal examples are available for training. We develop a method based on exchangeability martingales that only keeps features that exhibit the same pattern (i.e., are i.i.d.) under normal conditions of the observed phenomenon. We apply this to the problem of monitoring a Windows service and detecting anomalies it exhibits if compromised; results show that our method: i) strongly improves the AD system's performance, and ii) it reduces its computational complexity. Furthermore, it gives results that are easy to interpret for analysts, and it potentially increases robustness against AD evasion attacks.

Keywords: plug-in martingales, exchangeability, feature selection, anomaly detection, conformal prediction, information security

1. Introduction

The goal of Anomaly Detection (AD) is to identify test objects that do not conform to an expected normal behavior; we work under the assumption that training data contains only normal examples (*unsupervised* AD), and will call *anomalies* non-conforming objects in the test data. AD methods are widely used to detect anomalous events, such as frauds, mechanical faults, and software misbehaviors.

In recent years, information security researchers increasingly directed their attention to AD to identify attacks to systems. AD is particularly effective in scenarios where the observed phenomenon (e.g., the status of an operating system or a network) exhibits the same pattern under normal conditions; in these circumstances, one can use AD methods to identify anomalies and report them as attacks. AD in security is also motivated by the fact that, while it is generally cheap for adversaries to morph their attack vectors to bypass deterministic checks (e.g., in the context of malware, by using polymorphism), changing the attack's behavior to fit in with the normal behavior may have a larger cost.

AD applications to security problems need to consider carefully that: i) attacks are not necessarily anomalies with respect to the monitored features, and ii) to limit false positives, the AD system should only monitor features that indeed exhibit the same pattern under normal conditions. The former is application-specific, and it can be approached by monitoring as many features as possible. The latter, on the other hand, can be solved by performing feature selection, and it is inspiration for the method we propose in this paper.

© 2018 G. Cherubin, A. Baldwin & J. Griffin.

1.1. Feature selection for AD

As with most machine learning problems, feature selection has a critical importance for AD, since using the right features generally leads to i) reducing computational complexity, and ii) improving the convergence rate of the learning algorithm, potentially making it more robust to noise. However, in terms of feature selection, there is a fundamental difference between supervised learning and unsupervised AD. In a supervised setting, features' utility is determined as their ability to separate objects belonging to different classes; moreover, low variance features can usually be discarded. In AD, on the other hand, we are only given normal data; this means the only assertion we can make about features in this context is whether they indeed follow the same pattern under normal conditions or not. Remarkably, in this case, features with low variance under normal conditions may be useful for detecting anomalies (e.g., a constant value that changes with a class of anomalies can be very useful); for the same reason, high variance does not imply a feature should be discarded.

This inherent constraint of the AD setting was often ignored by previous work on feature selection for AD. In fact, many researchers proposed feature selection methods that use anomalous examples together with normal data (Kayacik et al., 2005; Stein et al., 2005; Iglesias and Zseby, 2015). Arguably, their approach, whilst having the advantage of supervised learning selection methods, biased the selected features to the kinds of anomalies provided within training data, which does not necessarily represent all possible anomalies. Other researchers approached AD feature selection by using dimensionality reduction methods (e.g., PCA) (Flynn and McLoone, 2011; Puggini and McLoone, 2017); the output of these methods, however, lacks of interpretability.

Kloft et al. (2008) did feature selection respecting the constraint; specifically, they extended the optimization task of Support Vector Data Description (SVDD) to account for features' importance.

1.2. Contributions

We present a feature selection method for AD that only looks at normal examples. The method is based on the following intuition: a feature should be used to detect anomalies if, under normal conditions, it exhibits the same pattern. This intuition is formalized by the i.i.d. property: we will say that a feature is i.i.d. if its values over time appear to be independently sampled from the same distribution. In this paper, we test if a feature is i.i.d. by using a method for testing exchangeability that is based on Plug-in martingales (Fedorova et al., 2012)[1], and we discard any feature that does not pass such test.

As a motivating example, we consider the problem of monitoring a Windows service, and detecting misbehaviors that may be the consequence of attacks. In a general computing platform such as a Windows PC, it could be very hard to capture normal behavior associated with various user activities and usage patterns. However, there are standard services that run within the OS that are likely to have a much more predictable behavior. Such services could be attacked as a route to hiding malware through techniques such as DLL injection; furthermore, services that provide security functions could be maliciously patched by an adversary to stop executing them.

1. To test for exchangeability is equivalent to testing for i.i.d.. A sequence of random variables is *exchangeable* if their joint probability distribution is the same for any permutation of the sequence.

Within the problem of monitoring a Windows service, our method allows the following automated strategy: 1) initialize the features to all possible measurements of a Windows service (e.g., memory, CPU and resource usage); 2) collect data for such features under normal conditions; 3) remove the features that do not exhibit the same pattern over time by using martingale-based exchangeability testing; 4) perform AD using standard algorithms (e.g., One-class SVM). This approach, which can be applied to various AD problems other than security, has the following advantages: i) it permits automation without expert knowledge of the data, and yet ii) it allows an analyst to supervise and control the feature selection process. We further argue that, in a security context, this feature selection method can make an AD system more robust to AD evasion techniques (subsection 4.4).

2. Problem setting

We consider the problem of AD on a time series of objects $\{x_t : t = 1, 2, ...\}$, where objects are d-dimensional vectors $x_t \in \mathbb{R}^d$. In a training phase[2], we access a sequence of n training objects, $x_1, ..., x_n$, which we assume are all labeled as normal. In a test phase, we observe a new object, x_{n+1}, and are asked to predict whether this object is anomalous with respect to the training data, or whether it is normal (i.e., it belongs to the same distribution).

We refer to the i-th feature of an object with x_t^i. Let $\Phi = \{1, ..., d\}$ be the original set of features. In this paper, we seek to find a set of features $\Phi' \subseteq \Phi$ such that an AD algorithm trained only considering features Φ' of the objects, obtains better predictions on test data than when it uses all the features. We define our evaluation criteria in subsection 4.3.

3. Feature selection using exchangeability martingales

A feature is not useful for AD purposes if its values within normal data do not exhibit the same pattern over time. We leverage this intuition to propose the following feature selection technique for AD: if the values of the i-th feature within training data, $x_1^i, ..., x_n^i$, do not appear i.i.d., we will remove such feature from the feature set. We first give an overview of the technique, and then provide the details throughout the section.

We remark that testing the i.i.d. assumption for a sequence is equivalent to testing its exchangeability. Vovk et al. (2003) proposed exchangeability martingales as a tool to test exchangeability for a sequence of objects as follows. Note that an exchangeability martingale is defined for a Conformal Predictor (CP) (subsection 3.1) and a betting strategy (subsection 3.2). Consider the i-th feature, and let $v_t = x_t^i$, for $t = 1, ..., n$. To verify that a sequence of feature values $v_1, v_2, ..., v_n$ is exchangeable (and thus i.i.d.) one can:

1. compute a p-value p_t for each feature value v_t, with $t = 1, ..., n$, by using a CP in online setting;
2. use the betting strategy to compute a martingale value M_t for each p-value;
3. reject exchangeability if some M_t exceeds a selected threshold ϑ.

2. Whilst our formulation of the problem and experiments assumes batch predictions, one can use features selected using the method we propose in an online setting.

This method works because of the following: i) if the sequence of values v_t is indeed i.i.d., then the CP returns p-values that are distributed uniformly in $[0,1]$ (Theorem 1); and ii) if p-values p_t are uniformly distributed, then the martingale M_t will increase above the chosen threshold ϑ (which is generally set to 20 or 100) only with low probability (Theorem 2).

The remainder of this section describes the theoretical results that support the method above, and application-specific details.

3.1. Conformal Predictors

A CP, $\mathcal{C}^A : \mathbb{R} \times \mathbb{R}^{n-1} \mapsto [0,1]$, is a wrapper around a scoring function (*nonconformity measure*) $A : \mathbb{R} \times \mathbb{R}^{n-1} \mapsto \mathbb{R}$, which, given a training sequence of objects $v_1, ..., v_{n-1}$, associates a p-value $p_n \in [0,1]$ to a new object v_n (Algorithm 1). For any nonconformity measure A, and for any n, a CP guarantees the following:

Theorem 1 (Vovk et al. (2005)) *Consider a sequence of objects $v_1, ..., v_n$. In an online setting, compute a p-value for each object as $p_t = \mathcal{C}^A(v_t, \lbag v_1, ..., v_{t-1} \rbag)$, for $t = 1, ..., n$. Then, if $v_1, ..., v_n$ are exchangeable, p-values $p_1, ..., p_n$ are distributed uniformly in $[0,1]$.*

To test that a sequence of feature values $v_1, ..., v_n$ is exchangeable, we will first compute the corresponding p-values $p_1, ..., p_n$ in an online setting by using a CP; then, we will use an exchangeability martingale to verify that p-values are uniformly distributed, as it is described in the next section. In experiment, we will use a CP with k-NN nonconformity measure A, which is defined as follows: let $v_{(i)}$ be the i-th closest object to v_n in a sequence $v_1, ..., v_{n-1}$, according to the Euclidean distance d; then,

$$A(v_n, \lbag v_1, ..., v_{n-1} \rbag) = \sum_{i=1}^{k} d(v_n, v_{(i)}) \quad .$$

This nonconformity measure is computationally efficient and well performing, which makes it one of the most commonly used. We remark that the guarantees formulated in this section are independent of the chosen nonconformity measure.

Algorithm 1: Smoothed CP for computing a p-value

Function `CP`$(v_n, \lbag v_1, ..., v_{n-1} \rbag)$**:**
- $V \leftarrow \lbag v_1, ..., v_{n-1}, v_n \rbag$
- **for** $i = 1, ..., n$ **do**
  - $\alpha_i \leftarrow A(v_i, V \setminus v_i)$
- **end**
- $\tau \leftarrow^{\$} Uni(0,1)$ ▷ Sample τ uniformly in $[0,1]$
- $p_n \leftarrow \frac{\#\{i|\alpha_i > \alpha_n\} + \tau \#\{i|\alpha_i = \alpha_n\}}{n}$

return p_n

3.2. Exchangeability martingales

An exchangeability martingale is a sequence of non-negative random variables $M_0, M_1, ...$ that keeps the conditional expectation:

$$M_t = E(M_{t+1}|M_1, ..., M_t), \quad t = 1, 2, ... \quad ,$$

where E is the expected value with respect to any exchangeable distribution; we assume M_0 is constant, $M_0 = 1$. Consider testing if a sequence of p-values $p_1, ..., p_n$ is exchangeable; we define an exchangeability martingale for a betting function $b : [0, 1] \times [0, 1]^* \mapsto [0, \infty)$ as:

$$M_t = \prod_{i=1}^{t} b_i(p_i), \quad t = 1, 2, ...,$$

where $b_i(p_i) = b(p_i, ⟅p_1, ..., p_{i-1}⟆)$ is the betting function computed on the previous p-values and evaluated on p_i. A proof that M_t satisfies the martingale's property is in Fedorova et al. (2012). Because $M_t = b_t(p_t)M_{t-1}$, the martingale's values can be updated efficiently.

Intuitively, a martingale can be thought as a betting strategy for a capital investing game without bankruptcy, where a player observes outcomes of events, and makes a bet on their next value; the player starts with a capital 1, and their capital increases the more predictable are the outcomes (Fedorova et al., 2012); within this analogy, the betting function determines how quickly the player becomes rich in the case of predictable outcomes.

Thanks to the following result, an exchangeability martingale can be used to test whether a sequence of p-values is uniformly distributed:

Theorem 2 (Ville (1939)) *For a martingale $M_1, M_2, ...$ and $\forall \vartheta \geq 1$:*

$$P(\exists t : M_t \geq \vartheta) \leq \frac{1}{\vartheta} \quad .$$

In other words, it is unlikely that a martingale takes large values. Common values for the threshold ϑ are 20 and 100, corresponding to significance levels 0.05 and 0.01.

To test if a sequence of p-values $p_1, ..., p_n$ is distributed uniformly in $[0, 1]$, we associate a martingale value to each of them in an online setting, and reject the hypothesis if the martingale increases more than the desired threshold.

3.3. Plug-in martingales

Fedorova et al. (2012) introduced Plug-in martingales, where the betting function b is a probability density estimate given previous p-values $p_1, ..., p_{n-1}$ and it is evaluated on p_n. Specifically, they used kernel density estimation (KDE), with Gaussian kernel and bandwidth h chosen according to Silverman's "rule of thumb"; to improve the estimate, they computed KDE on the extended set of p-values: $\{p_t, -p_t, 2 - p_t\}$, for each $t = 1, ..., n - 1$, and normalized in $[0, 1]$. In experiments, we will use the same approach.

4. Empirical Evaluation

We evaluate exchangeability martingales-based feature selection by performing AD on a Windows service dataset (`Windows-service`); we use One-class SVM as the AD algorithm, and evaluate its performances before and after feature selection.

4.1. Data

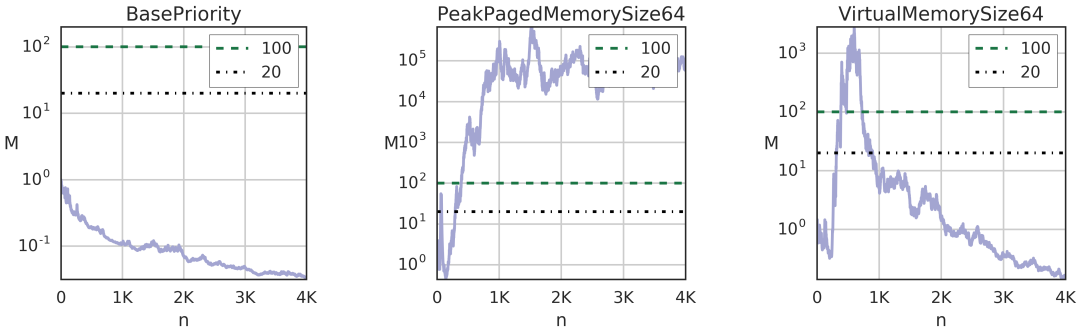
We consider the problem of monitoring a Windows service, and detecting anomalies in its behavior. To this end, and to avoid using malware on a system in normal use, we created a Windows service, and optionally enabled some anomalous functionalities during data collection; we henceforth refer to this service when operating under normal conditions as the *normal* service. The dataset includes 20051 examples, 8963 of which represent normal behavior; examples have 88 features each. Training and test data were collected in separate runs; specifically, training data only contains normal behavior, test data contains both normal and combinations of anomalous behaviors; we collected roughly 24 hours of observations for each kind of behavior.

Follows a description of the service's main functionality, its anomalous behaviors, and of the kinds of features we collected.

Normal behavior The normal service has functionalities emulating executable control functions that are often found in Next Generation AntiVirus systems (NGAV); these look for known fingerprints of good and bad software as they are executed. Specifically, it monitors OS events that occur as executable code is loaded, whether into a new process or as a DLL that is dynamically loaded into an existing process; it then computes a cryptographic hash of the executable file, which could be compared against a lists of known good or bad code. The behavior of this service will partly respond to events that relate to usage patterns; however, we expect its operations will mostly follow the same pattern over time.

Anomalous behaviors We embedded additional behaviors into our service that act as if malware had been injected as our service runs. While collecting training data, these behaviors were disabled so that only normal behavior was captured. During the collection of test data, we enabled each and combinations of these additional behaviors along with marking the recorded data as being normal or anomalous. Anomalous behaviors are described into details in Appendix.

Features We used the Windows process monitoring API to create features that capture process performance information. This includes user and privileged CPU usage, user and privileged memory usage, various working set and virtual memory measurements, the number of threads, processor priority, and handle counts. In addition, we recorded information about network connections made by the process, along with the protocol type, and a flag determining whether the connection was to an intranet address or to an external Internet address. We also recorded the number and class of DLLs loaded into the service. Note that malware may use a reflective DLL loading technique, so its own code may not be listed; therefore we did not load separate DLLs associated with the anomalous behaviors; however, the use of library calls may cause further system DLLs to load.



(*a*) An i.i.d. feature. (*b*) A non-i.i.d. feature. (*c*) Uncertain, removed for $\vartheta = 20$.

Figure 1: Martingales plots for evaluating individual features. I.i.d. features are kept.

Data collection During data collection, the Windows service was run separately in normal and anomalous conditions; the latter accounted for various combinations of the anomalous behaviors. The service was run and data collected on a Windows 10 laptop over a few days. Whilst data was collected, the laptop was used as a normal work system; this included performing a variety of routine tasks, from creating and editing documents (Word, PowerPoint and Excel), web browsing, and using Visual Studio for development. Data was sampled every 10 seconds.

As preprocessing, we first one-hot encoded categorical values. Then, for a subset of features, we derived "diff" features by subtracting the feature's value at time $t-1$ to its value at time t; this is a standard practice in time series analysis, which tends to produce stationary features from non-stationary ones.

4.2. Feature selection

We consider individually each feature in the training data, and compute its martingales as in section 3. Common threshold choices for the exchangeability test are 20 or 100, corresponding to a probability of mistakenly rejecting the i.i.d. hypothesis of 0.05 and 0.01; we select $\vartheta = 20$, although we notice that $\vartheta = 100$ would have made no difference on this dataset. A feature is discarded if its martingale is ever greater than ϑ.

Figure 1 represents the values of the exchangeability martingales for three features. We notice that these plots can be easily interpreted by a domain expert, who for example may choose to include a feature even if its martingale exceeds the selected threshold for a small amount of examples (Figure 1c). Furthermore, to avoid such uncertainty, and because p-values from which martingales are computed depend on a source of randomness, an analyst may run the method for various initialization seeds, and then make a decision based on the respective plots. In experiments, we did not use any expert heuristics of this kind, and we simply discarded a feature if its martingale ever exceeded $\vartheta = 20$.

A further intuition of how the method works is given by looking at how discarded features behave over time. Figure 2 shows the values of a subset of features over time,

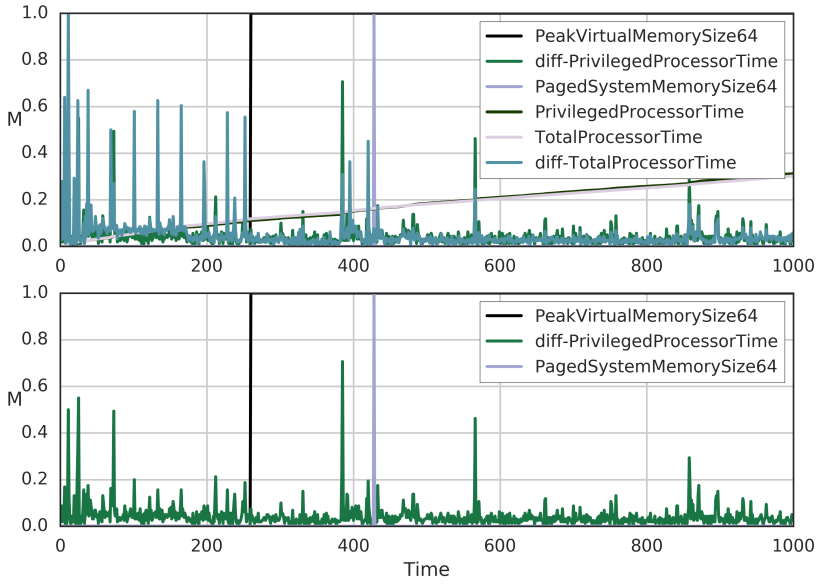


Figure 2: A subset of features of the `Windows-service` dataset, plotted in time before and after martingales feature selection (threshold: 20).

before and after feature selection. We observe that monotonic increasing features, such as `TotalProcessorTime` and `PrivilegedProcessorTime`, are removed. We also observe that feature `diff-TotalProcessorTime` was removed; indeed, the upper part of Figure 2 shows this feature exhibits pattern changes.

4.3. Evaluation

We evaluate the performance of an AD algorithm before and after feature selection, with respect to the size of the training set. To this end, we first train the algorithm only on 4 hours of observations, and test it against a much longer trace; then, we measure its performances while increasing the length of its training data up to roughly 24 hours.

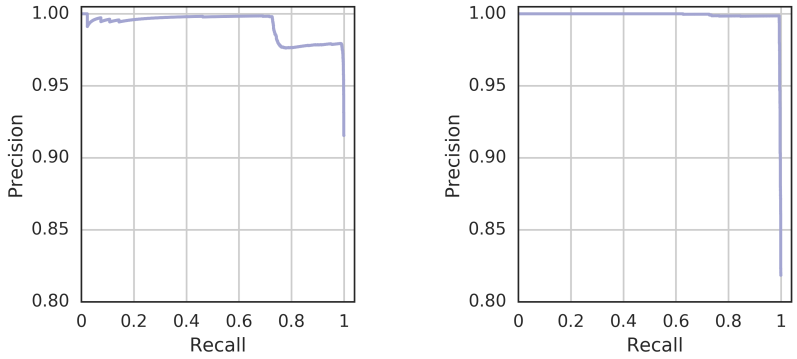
We use One-class SVM as an AD algorithm[3], and we evaluate its predictions by measuring precision, recall and F1 score. We select an RBF kernel for One-class SVM, whose parameter γ we pick according to a grid search within $[10^{-9}, 10^{3}]$ to maximize the F1 score on validation data. One-class SVM also accepts a parameter ν, which represents the probability of a training object to be an anomaly; since we use a training set of normal examples, we use the rule of thumb, $\nu = 1/n$, where n is the size of training data.

We first train One-class SVM on 4 hours of training data (1296 examples), and evaluate it on the test data (9608 examples, 4909 of which normal), before and after feature selection.

3. In experiments, we use the `scikit-learn` implementation of One-class SVM (Pedregosa et al., 2011), and `random-world` for CP and exchangeability martingales (Cherubin, 2017).

Feature selection	# features	Precision	Recall	F1
None	88	0.725	0.999	0.840
Martingales ($\vartheta = 20$)	72	0.858	0.999	0.923

Table 1: AD performances before and after feature selection on `Windows-service`.



(a) Before feature selection. (b) After feature selection.

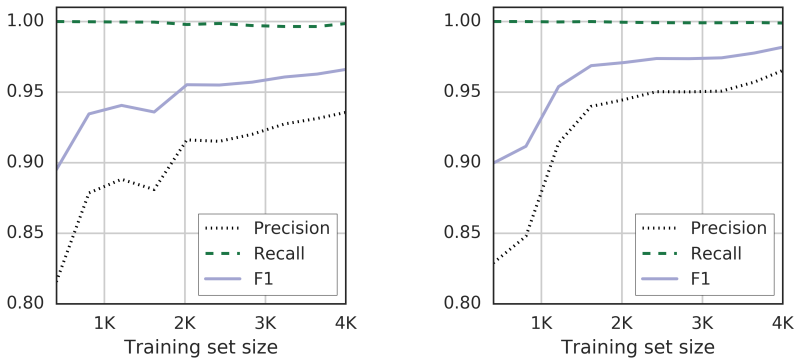
Figure 3: Precision-recall curves before and after feature selection.

Because the performance measures we use are robust for both balanced and imbalanced data, the fact that our dataset is balanced does not affect the generality of the results.

Table 1 shows the AD performances, before and after feature selection. We observe that, whilst the recall is close to 1 in both cases, feature selection causes a major improvement in the precision; this is the consequence of a strong decrease in false positives (1780 before feature selection, 775 after), which virtually did not affect true positives (4694 before, 4693 after)[4]. This also suggests that the model after feature selection is more robust (subsection 4.4). Precision-recall curves, plotted using One-class SVM's decision function, also confirm better performance after feature selection (Figure 3).

We further evaluate feature selection using a larger training set of $n = 4052$ examples, by training One-class SVM on subsets of $10\%, 20\%, ..., 100\%$ of its examples[5]. Figure 4 shows that the AD method after feature selection converges quickly to optimal performance. However, we notice that, as the training data grows, One-class SVM tends to perform similarly before and after feature selection; indeed, reducing the feature set can never improve the asymptotic performance of a learning algorithm, but it can boost its finite sample convergence rate (Devroye et al., 2013).

4. In practical applications, one could further reduce false positives by using a shifting window of predictions, and setting a threshold on the anomalies count.
5. We repeat this for 10 random shuffles of the original dataset, and then average the results.



(*a*) Before feature selection. (*b*) After feature selection.

Figure 4: Precision, recall and F1 score as the size of training data increases.

4.4. Robustness of predictions

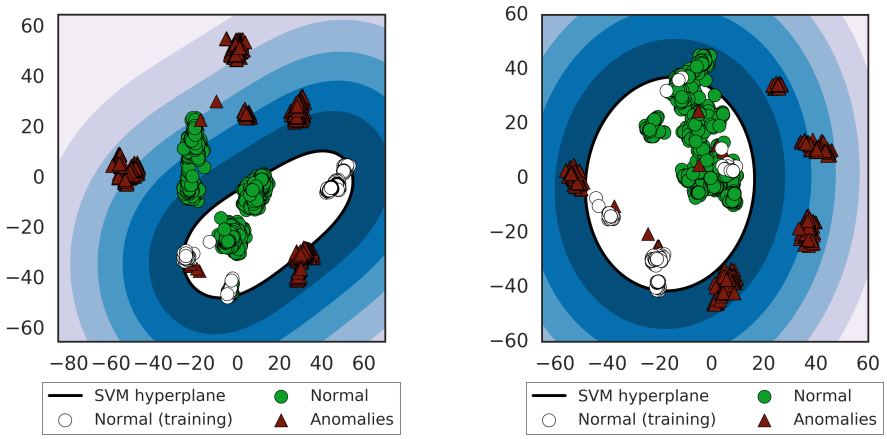
Most of the workload of the Windows service we consider is based on system-generated events, which should be easily learnable; however, part of its behavior depends on user patterns, and while such behavior could be learned by collecting data for a long time, this would lead to a less robust model. This aspect plays an important role in AD evasion attacks, where an adversary wants to craft a malicious service that evades the AD's checks.

To better understand the effects of feature selection on the robustness of One-class SVM, we visualize its hyperplane for a 2D representation of the data. Specifically, we use a combination of t-SNE (Maaten and Hinton, 2008) and k-NN regressor to map the original data into two dimensions[6], and then use One-class SVM to perform AD in such space. We stress that, because 2D objects will not keep their original disposition in space, we will only use these plots as an intuition of the result of feature selection in high dimensions.

We perform this before (Figure 5a) and after (Figure 5b) feature selection. We notice that normal objects before feature selection form various clusters, some of which are not captured by the One-class SVM model. Figure 5b suggests that the reduced feature space better characterizes the features that represent the normal behavior of the service, and it brings test examples closer to the training data. Importantly, after feature selection, objects exhibit a more regular behavior, which is easily characterized by two groups; this indicates that an AD algorithm after feature selection will be able to create tighter boundaries.

There is a trade-off between the amount of data one needs to collect for training, and the need for a suitable reduction in features. More training data may allow the AD algorithm to fit more complex distributions (Figure 5a); however, this will also produce less robust models, which may cause sudden spikes in false positive rates during normal operations (e.g., in the context of our application to Windows data, when new software is installed).

6. We use t-SNE to map a subset of 1000 objects into 2 dimensions, and then use k-NN regressor to embed the remaining objects into a 2D space. This improves the computational complexity, and in our experiments it produces better embeddings than vanilla t-SNE.



(*a*) Before feature selection. (*b*) After feature selection.

Figure 5: One-class SVM on a t-SNE projection of data before and after feature selection.

5. Conclusion and Future Work

When applying AD, it is essential to select the right set of features to ensure the robustness of the learning algorithm. In this paper, we suggested that features for unsupervised AD should only be rejected when their values do not exhibit the same pattern over time (i.i.d. assumption). Based on this intuition, we introduced a feature selection method that tests the i.i.d. hypothesis for each feature, and discards those failing the test; we do this by using exchangeability martingales, a tool derived from CP. Whilst previous research constructed AD techniques from CP and exchangeability martingales (Ho, 2005; Laxhammar and Falkman, 2010; Cherubin et al., 2015; Ishimtsev et al., 2017), this is, to the best of our knowledge, their first use for feature selection. Interestingly, our approach allows automatic feature selection while retaining the ability for an expert analyst to review the process.

We applied this method to the problem of monitoring a Windows service and detecting anomalous (potentially malicious) behavior. A comparison before and after feature selection indicates that our method strongly improves the precision of the AD algorithm, without affecting its recall. Furthermore, we argued it favors the robustness of the model, which is particularly desired in applications where an adversary may try to evade the AD system.

With regards to the application to Windows data, our aim was to test whether we could use AD to differentiate a service's normal behavior from behaviors with additional or reduced functionalities, and to select which features characterize the problem space. Future work may look at performing AD across different devices, including a variety of processors, physical memory and OS versions.

Future work may apply this technique to AD problems from different domains; also, whilst we only evaluated our technique in an AD setting using One-class SVM, the same method can improve the performance of other AD algorithms (subsection 4.4). More generally, we expect that the same approach can be applied to many other unsupervised learning problems, such as clustering.

Acknowledgments

We are thankful to Dan Ellam and Alexander Gammerman for useful discussion. We thank the anonymous reviewers, whose suggestions led to improvements of this paper. This work was done while Giovanni Cherubin was visiting HP Labs Security Lab in Bristol. Cherubin was supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/K035584/1).

Appendix A. Anomalous Data

The following table describes the anomalies we injected into the Windows service during data collection.

Name	**Description**
File scanning	This caused the service to scan through files on the local system. This behavior could be associated with malware trying to find files that may contain potentially valuable data (such as documents, presentations, spreadsheets), or with the early phase of many pieces of ransomware which often create an index of interesting files prior to starting encrypting them.
C&C	Most malware needs to call home to a command and control (C&C) server to receive instructions and updates, This will often happen by malware regularly beaconing to a web address over HTTP(S) thus allowing malware installed on a laptop to get out through most firewalls. Malware may call home to a hard coded address (IP or DNS address) or it may use a Domain Generation Algorithm (DGA) which generates a large number of DNS requests one of which will have been registered by the malware owner. We added this command and control behaviour into our service.
Monitoring disabled	Where there are security monitoring functions running in a service such as this one, malware may try to prevent detection by disabling either the service or just the monitoring functions. Thus our third anomaly class involved turning off the monitoring functions within the service.

References

Giovanni Cherubin. Random-world: Rust implementation of machine learning methods for confident prediction. https://github.com/gchers/random-world, 2017.

Giovanni Cherubin, Ilia Nouretdinov, Alexander Gammerman, Roberto Jordaney, Zhi Wang, Davide Papini, and Lorenzo Cavallaro. Conformal clustering and its application to botnet traffic. In *International Symposium on Statistical Learning and Data Sciences*. Springer, 2015.

Luc Devroye, László Györfi, and Gábor Lugosi. *A probabilistic theory of pattern recognition*, volume 31. Springer Science & Business Media, 2013.

Valentina Fedorova, Alex J. Gammerman, Ilia Nouretdinov, and Volodya Vovk. Plug-in martingales for testing exchangeability on-line. In *Proceedings of the 29th International Conference on Machine Learning, ICML*, 2012.

Beibei Flynn and Seán McLoone. Max separation clustering for feature extraction from optical emission spectroscopy data. *IEEE Transactions on Semiconductor Manufacturing*, 24(4), 2011.

Shen-Shyang Ho. A martingale framework for concept change detection in time-varying data streams. In *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005.

Félix Iglesias and Tanja Zseby. Analysis of network traffic features for anomaly detection. *Machine Learning*, 101(1-3), 2015.

Vladislav Ishimtsev, Alexander Bernstein, Evgeny Burnaev, and Ivan Nazarov. Conformal k-nn anomaly detector for univariate data streams. In *Conformal and Probabilistic Prediction and Applications, COPA*, 2017.

H Günes Kayacik, A Nur Zincir-Heywood, and Malcolm I Heywood. Selecting features for intrusion detection: A feature relevance analysis on kdd 99 intrusion detection datasets. In *Proceedings of the third annual conference on privacy, security and trust*, 2005.

Marius Kloft, Ulf Brefeld, Patrick Düessel, Christian Gehl, and Pavel Laskov. Automatic feature selection for anomaly detection. In *Proceedings of the 1st ACM workshop on Workshop on AISec*. ACM, 2008.

Rikard Laxhammar and Göran Falkman. Conformal prediction for distribution-independent anomaly detection in streaming vessel data. In *Proceedings of the First International Workshop on Novel Data Stream Pattern Mining Techniques*. ACM, 2010.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov), 2008.

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

Luca Puggini and Seán McLoone. Forward selection component analysis: Algorithms and applications. *IEEE transactions on pattern analysis and machine intelligence*, 39(12), 2017.

Gary Stein, Bing Chen, Annie S Wu, and Kien A Hua. Decision tree classifier for network intrusion detection with ga-based feature selection. In *Proceedings of the 43rd annual Southeast regional conference-Volume 2*. ACM, 2005.

Jean Ville. *Etude critique de la notion de collectif*. Gauthier-Villars Paris, 1939.

Vladimir Vovk, Ilia Nouretdinov, and Alexander Gammerman. Testing exchangeability on-line. In *Machine Learning, Proceedings of the 20th International Conference ICML*, 2003.

Vladimir Vovk, Alexander Gammerman, and Glenn Shafer. *Algorithmic learning in a random world*. Springer Science & Business Media, 2005.