Relative Density & Median One-Class Classifiers

Jimmy Azar ja45@aub.edu.lb
Department of Industrial Engineering & Management
American University of Beirut
Beirut 11-0236, Lebanon

Abstract

We introduce two classifiers for anomaly detection: the *relative density one-class classifier* and the *median one-class classifier*. The first is based on kernel density estimation, while the second is distance-based. The two classifiers are related from an algorithmic and implementation perspective, and novel scalable versions are presented. We demonstrate their ability on artificial datasets as proof of concept alongside the isolation forest algorithm, one-class support vector machine, and local outlier factor. We also experiment with these classifiers using the NSL-KDD dataset for building an intrusion detection system. We address both binary classification for differentiating between normal network traffic data and attacks, as well as multiclass classification for differentiating between different types of attacks. In the latter case, we emulate a multiclass classifier by training multiple one-class classifiers and combining their decisions through a majority vote. Finally we present the *median transform* as a pre-processing method that can complement one-class classifiers and can be especially useful in the unsupervised setting. Source code for all experiments is available on GitHub[1].

Keywords: Anomaly detection, one-class classification, intrusion detection system

1. Introduction

In contrast to standard multiclass classification methods which aim to differentiate between two or more well-sampled classes, one-class classification attempts to describe a single class of objects called the *target* class. In such problems, examples of non-target objects are often scarce or difficult to define. One-class classification (OCC) (Moya et al., 1993) addresses the problem when one class is severely undersampled or even completely absent compared to the other. This imbalance can be problematic when training a standard binary or multiclass classifier especially due to highly imbalanced class priors and misclassification costs. In this paper, we refer to objects belonging to the *target* class as *inliers* (or target objects), whereas those that do not as *outliers*. One-class classifiers may be viewed as either density-based (e.g. Parzen OCC), reconstruction-based (e.g. relying on k-means), or boundary based (e.g. one-class SVM) (Tax, 2001). Some of the simplest one-class classifiers are parametric assuming a Gaussian distribution over the target class; one such version is based on the minimum covariance determinant (Rousseeuw, 1984). Other classifiers rely on clustering methods such as k-means (MacQueen, 1967), Gaussian mixture model (Bishop, 1995), or self-organizing maps (Kohonen, 2001). Some of the most successful methods have been the one-class SVM (Tax, 2001), isolation forest (Liu et al., 2008), and local outlier factor (Breunig et al., 2000). These classifiers have been used in various anomaly detection applications. One common

1. https://github.com/jimmyazar/OCC/

application is in building a network intrusion detection system (IDS). A benchmark dataset in this area is NSL-KDD (Tavallaee et al., 2009), an improved version of the original KDD dataset ("University of California, Irvine", 1999). Early work in this field included the use of statistical approaches such as Hotelling's T^2 test and chi-square test (Ye et al., 2002; Feinstein et al., 2003), association rules (Su et al., 2009), clustering-based methods (Leung and Leckie, 2005; Zhang and Zulkernine, 2006), PCA-based methods (Ringberg et al., 2007), k-nearest neighbor approach (Liao and Vemuri, 2002), artificial neural networks (Liu et al., 2002), and support vector machines (Hu et al., 2003). More recent approaches that have regained attention with the rise of deep learning are based on autoencoders (Andrews et al., 2016; Zhou and Paffenroth, 2017), and several publications have used these for network intrusion detection (Shone et al., 2018; Gurung et al., 2019). Additionally, one-class SVM has been trained on features learned by deep belief networks and has shown comparable performance to deep autoencoders (Erfani et al., 2016). Other recent methods include a generalized one-class SVM based on a Bayesian framework (Turkoz et al., 2020), as well as a modified optimum-path forest algorithm (Bostani and Sheikhan, 2017) which employs k-means clustering to improve the original graph-based method.

In this paper, we present one-class classifiers that do not assume a parametric model or make assumptions concerning the generative distribution of the target class. This gives the methods flexibility which may be tuned by a minimal number of hyperparameters. One of the two classifiers presented requires kernel density estimation, and both classifiers require k-nearest neighbor identification. Both these classical concepts become prohibitive for large datasets, and several approximative approaches have been proposed in the past (Girolami and He, 2003; Wang et al., 2009). In this work, we present scalable implementations from an algorithmic perspective to overcome this problem. We also compare the classifiers' performance alongside isolation forest, one-class SVM, and local outlier factor on artificial data and the NSL-KDD dataset. As in (Domingues et al., 2018), we evaluate these classifiers based on the area under the ROC curve, however we use the full NSL-KDD dataset as opposed to a subset of the original KDD dataset, where in the latter, discerning performance differences becomes difficult due to redundancies in the data. We explore the situation of normal network traffic versus attacks, as well as the case of multiple classes representing different types of attacks. In the latter case, we emulate a multiclass classifier by combining several one-class classifications. Using one-class classifier ensembles to decompose a multiclass problem can outperform the standard multiclass approach despite the fact there is less information available during the training phase (Krawczyk et al., 2015). This is so because one-class classifiers can deal with data imbalance in a way that is not biased towards the majority class. An important feature is that the one-class classifier is able to focus on each class independently (Krawczyk et al., 2018), whereby the classifier can adjust itself to the target class. Finally, in the complete absence of target/outlier labels, recent work has suggested the use of a clustering-based proxy (C-proxy) measure to optimize hyperparameters of one-class classifiers (Yu and Kang, 2019). We present an alternative approach denoted the *median transform* which can create a new (target-only) dataset from the existing one. One-class classifiers can then be reliably trained on the transformed dataset, and hyperparameter tuning is simplified as artificial outliers may be generated from a uniform distribution around the target class as needed.

2. Relative density classifier

We introduce a one-class classifier, denoted *relative density one-class classifier* or RDOCC, which is based on kernel density estimation (Parzen, 1962). For a given training set $\mathcal{X}_{tr}$ consisting of n inlier (target) objects (possibly in addition to outlier objects), we compute the density estimate for any object x_i as

$$D(x_i) = \frac{1}{n} \sum_{t=1}^{n} exp\left(-\frac{||x_t - x_i||^2}{2h^2}\right) \in [0,1],$$

where $x_t \in \mathcal{X}_{tr} : x_t$ is an inlier. To classify the object x_i, we compare $D(x_i)$ with the average density estimate of the k-nearest neighbors (in $\mathcal{X}_{tr}$) to x_i. In particular, we classify according to the following difference:

$$\left[D(x_i) - \frac{1}{k} \sum_{j \in knn(x_i)} D(x_j) \right] \underset{outlier}{\overset{inlier}{\gtrless}} \theta_{fracrej},$$

where $knn(x_i)$ represents the indices of the k-nearest neighbors to object x_i which are inliers in $\mathcal{X}_{tr}$, and the threshold $\theta_{fracrej}$ is learned from the training set $\mathcal{X}_{tr}$ and will correspond to a unique *reject fraction* indicating the fraction of training inlier objects to reject from the target class.

If an object x_i is distant from the training inlier data, the kernel density estimate $D(x_i)$ will be relatively small, whereas the density estimate of its k-nearest neighbors embedded deeper in the target class will be larger in comparison. Intuitively, this can signal that the object is an outlier.

The relative density classifier has two hyperparameters in addition to the reject fraction $f \in [0,1]$: the (Gaussian) kernel scale parameter h and the number of nearest neighbors to consider k.

3. Median one-class classifier

The median is a natural remedy to outliers. Because it is based on rank statistics, the median is a robust central tendency measure, immune to outliers compared to the mean. *Median filtering* is a well known concept in the image analysis community and a powerful tool to remove *salt and pepper* noise (i.e. outlier speckles) from images. An $N \times N$ kernel (filter) is passed over the pixels of a corrupted image, and the central pixel is replaced by the median of its $N \times N$ neighborhood pixel values. Visually, this results in a remarkable clean-up of the image. We extend the idea of using a neighborhood median into anomaly detection and construct a one-class classifier.

We introduce the median one-class classifier (MOCC) which is based on a distance rather than density measure. For a given training set $\mathcal{X}_{tr}$ consisting of n inlier (target) objects in p dimensions (possibly in addition to outlier objects), and for a given object x_i to be classified, we compute the median of the k-nearest neighbors (in $\mathcal{X}_{tr}$) to x_i. This median object x_m is computed by considering the median of each feature among the k-nearest neighbors, i.e.:

$$x_m = \underset{j}{\text{median}}(x_j = (x_{j1}, ..., x_{jp})) = (\text{median}(x_{j1}), ..., \text{median}(x_{jp})), j \in knn(x_i),$$

where $knn(x_i)$ represents the indices of the k-nearest neighbors to object x_i which are inliers in $\mathcal{X}_{tr}$. Note x_m may not necessarily coincide with any known object in the training set. To classify the object x_i, we compute the Euclidean distance from x_i to x_m. In particular, we classify according to the following rule:

$$||x_i - x_m|| \underset{inlier}{\overset{outlier}{\gtrless}} \theta_{fracrej}.$$

The threshold $\theta_{fracrej}$ is learned from the training set $\mathcal{X}_{tr}$ and will correspond to a unique *reject fraction* indicating the fraction of training inlier objects to reject from the target class. Depending on how distant an object x_i is from the median of its k-nearest (inlier) neighbors, it will be labeled as either an inlier or outlier.

MOCC has two hyperparameters: a reject fraction $f \in [0,1]$, and the number of nearest neighbors to consider, k.

4. Algorithm

Algorithm 1 describes an implementation of the relative density classifier, and Algorithm 2 provides a scalable version for large datasets in a way that bypasses memory limitations. Algorithm 2 is comparatively complex and relies on dividing data into manageable blocks and aggregating results in a novel way after processing each block. This approach which we call *blocking* works well in this context because we are only interested in the row sums of processed blocks, and sums can be aggregated. By blocking, we reduce each block into a single column, and we avoid computing or storing the full distance matrix between objects, which is infeasible for large datasets.

Likewise, Algorithm 3 describes an implementation of MOCC, and Algorithm 4 presents a scalable version which also relies on *blocking* for computing the k-nearest neighbors in large datasets. Again blocking works well in this case because we can compute the k-nearest neighbors in each processed block independently, and then once all blocks have been processed, we identify the k-nearest neighbors of a given object from among all the stacked k-nearest neighbors found from each block. This way we are also able to map the correct k-nearest neighbor indices to locations in the full distance matrix as if we had hypothetically computed the full distance matrix between objects. By analogy, if we wanted to select the top 5 candidates out of a 100, we divide the 100 candidates into manageable blocks or groups of say 20, and identify the top 5 in each group. Lastly, we identify the top 5 from among the short-listed candidates of each group. This is how blocking works in the case of MOCC as applied on 2D matrices. Algorithms 2 and 4 are implemented in `R` language as `rdocc_scalable` and `mocc_scalable` respectively, and source code is available in GitHub[2].

2. https://github.com/jimmyazar/OCC/

Algorithm 1: Relative density classifier (RDOCC)

```
Input : Training data $\mathcal{X}_{tr}$, query objects $\mathcal{X}_q$, number of neighbors $k$, kernel parameter $h$,
        reject fraction $f$
// Find density at training objects:
1  Compute $n \times n$ distance matrix $D$ over $\mathcal{X}_{tr}$ ;
2  Compute kernels: $[g_{ij}] \leftarrow exp\left(-\frac{1}{2}\frac{D^2}{h^2}\right)$; $D^2$ denotes element-wise exponentiation ;
3  Sum kernel contributions: $A \leftarrow \sum_j g_{ij}$ ;
4  Set (local) density estimates: $P_l \leftarrow A/n$ ;
// Find average density at k-nn neighbors:
5  for $i$ = 1,...,$n$ do
6      Find set $\mathcal{N}$ of $k$ nearest neighbors to object $x_i \in \mathcal{X}_{tr}$;
7      Compute kernel values: $[g_{lj}] \leftarrow exp\left(-\frac{1}{2}\frac{D_{lj}^2}{h^2}\right)$; $l \in \mathcal{N}$, $j = 1, ..., n$ ;
8      Sum kernel contributions: $B_l \leftarrow \frac{1}{n}\sum_j g_{lj}$ ;
9      Set (wide) density estimate: $(P_w)_i \leftarrow \frac{1}{k}\sum_l B_l$
10 end
11 Set relative distances: $relDist \leftarrow P_l - P_w$ ;
12 Sort $relDist$ in descending order ;
13 Compute number of (training) objects accepted: $nobj_{acc} \leftarrow \lfloor(1-f)n\rfloor$ ;
// Set threshold $\theta$:
14 if $nobj_{acc} == 0$ then
15     $\theta \leftarrow \infty$;
16 else if $nobj_{acc} == n$ then
17     $\theta \leftarrow -\infty$ ;
18 else
19     $\theta \leftarrow relDist[nobj_{acc}]$
20 end
// Find density at test objects:
21 Compute $n^{'} \times n$ distance matrix $D^{'}$ between $\{(\mathcal{X}_q)_{n^{'} \times p}, (\mathcal{X}_{tr})_{n \times p}\}$ ;
22 Compute kernels: $[g^{'}_{ij}] \leftarrow exp\left(-\frac{1}{2}\frac{D^{'2}}{h^2}\right)$; $D^{'2}$ denotes element-wise exponentiation ;
23 Sum kernel contributions: $A^{'} \leftarrow \sum_j g^{'}_{ij}$ ;
24 Set (local) density estimates: $P^{'}_l \leftarrow A^{'}/n$ ;
// Find average density at k-nn neighbors:
25 for $i$ = 1,...,$n^{'}$ do
26     Find set $\mathcal{N}^{'}$ of $k$ nearest neighbors from $\mathcal{X}_{tr}$ to object $x^{'}_i \in \mathcal{X}_q$;
27     Compute kernel values: $[g^{'}_{lj}] \leftarrow exp\left(-\frac{1}{2}\frac{D_{lj}^2}{h^2}\right)$; $l \in \mathcal{N}^{'}$, $j = 1, ..., n$ ;
28     Sum kernel contributions: $B_l \leftarrow \frac{1}{n}\sum_j g^{'}_{lj}$ ;
29     Compute (wide) density estimate: $(P^{'}_w)_i \leftarrow \frac{1}{k}\sum_l B_l$
30 end
31 Compute relative distances: $relDist^{'} \leftarrow P^{'}_l - P^{'}_w$ ;
32 Compute query labels: $l_q \leftarrow \mathbb{I}(relDist^{'} < \theta)$; $\mathbb{I}$ := element-wise indicator function ;
Output: Query labels $l_q$ (0:=inlier, 1:=outlier)
```

Note that when computing kernel values $g^{'}_{lj}$ (line 27), the *training* set distance matrix D_{lj} is used since the k-nearest neighbors belong to the training set $\mathcal{X}_{tr}$ (line 26). Algorithm

1 requires the computation of distance matrices D (line 1) and $D^{'}$ (line 21) which is feasible for small datasets consisting of a few thousand samples depending on system RAM allocation. Therefore, for large datasets, we introduce a scalable version that operates within RAM limits by dividing the dataset into blocks, performing block-wise computations, and aggregating results without needing to compute the full distance matrices or hold these in memory. In the scalable version, Algorithm 2 replaces lines 1-20, and a similar procedure replaces lines 21-31. Moreover, the input k can be a vector instead of a scalar, and the output will be a list of test labels for each case of k.

Algorithm 2: Scalable density estimation: replaces lines 1-20 in Algorithm 1

```
Input : Training data $(\mathcal{X}_{tr})_{n \times p}$, vector $k = (k_1, ..., k_r)$ of number of neighbors, kernel
        parameter $h$, maximum block width or height $m$
1  Create $\lceil n/m \rceil (= l)$ blocks $(Q_{m \times p})_i; i = 1, ..., l$ from $\mathcal{X}_{tr}$ ;
2  Initialization: $K \leftarrow Z_{ml \times l}$ where $Z :=$ zero matrix ;
3  $k\_max \leftarrow max(k)$ ;
4  for $i$ = 1,...,$l$ do
5      for $j$ = 1,...,$l$ do
6          Compute $m \times m$ distance matrix $(b_{m \times m})_{ij}$ between $\{Q_i, Q_j\}$ ;
7          Compute kernels: $[g_{ij}] \leftarrow exp\left(-\frac{1}{2}\frac{b_{ij}^2}{h^2}\right)$; $b_{ij}^2 :=$ element-wise exponentiation ;
8          Store indices of $k\_max$ nearest neighbors for each object $x \in Q_i$ in
            $U_{ij} := [u_{yz}]_{ij}; y = 1, ..., m; z = 1, ..., k\_max$ ;
9          Store $k\_max$ nearest neighbor distances for each object $x \in Q_i$ in
            $D_{ij} := [d_{yz}]_{ij}; y = 1, ..., m; z = 1, ..., k\_max$ ;
10         Reduce block to a single column by summing kernels: $s \leftarrow \frac{1}{v}\sum_{j=1}^{v} g_{ij}$ ;
11         Store column result $s$ in $K$ ;
12     end
13 end
14 Compute total kernel sum per object: $A \leftarrow \sum_{j=1}^{l} K_{ij}$ ;
15 Trim out the last $(ml - n)$ rows of $A$ ;
16 Set (local) density estimates: $P_l \leftarrow A/n$ ;
17 for $i$ = 1,...,$r$ do
18     Find indices $\mathcal{L}$ of $k(i)$ nearest neighbors across the $l \times k\_max$ columns of $D$ ;
19     Map indices $\mathcal{L}$ using $U$ to index locations $\mathcal{L}^{'}$ in hypothetical full distance matrix over
        $\mathcal{X}_{tr}$ ;
20     Trim out the last $(ml - n)$ rows of $\mathcal{L}^{'}$ giving $\mathcal{L}^{''}$ ;
21     Compute $t \leftarrow A[\mathcal{L}^{''}]$ ;
22     Set (wide) density estimates: $P_w \leftarrow \frac{1}{k}\sum_b t_{ab}$ ;
23     Set relative distances: $relDist \leftarrow P_l - P_w$ ;
24     Sort $relDist$ in descending order ;
25     Compute number of (training) objects accepted: $nobj_{acc} \leftarrow \lfloor (1 - f)n \rfloor$ ;
       // Set threshold $\theta$:
26     if $nobj_{acc} == 0$ then
27         $\theta(i) \leftarrow \infty$;
28     else if $nobj_{acc} == n$ then
29         $\theta(i) \leftarrow -\infty$ ;
30     else
31         $\theta(i) \leftarrow relDist[nobj_{acc}]$
32     end
33 end
```

Algorithm 3: Median one-class classifier (MOCC)

```
   Input  : Training data $\mathcal{X}_{tr}$, query objects $\mathcal{X}_q$, number of neighbors $k$, reject fraction $f$
 1 Compute $n \times n$ distance matrix $D$ over $\mathcal{X}_{tr}$ ;
   // Find k-nn neighbors of each training object:
 2 for $i$ = 1,...,$n$ do
 3     Find set $\mathcal{N}$ of $k$ nearest neighbor indices to object $x_i \in \mathcal{X}_{tr}$;
 4     Compute the median: $x_m = \underset{j \in \mathcal{N}}{\text{median}}(x_j)$ ;
 5     Compute distance: $relDist_i = ||x_i - x_m||$
 6 end
 7 Sort $relDist$ in ascending order ;
 8 Compute number of (training) objects accepted: $nobj_{acc} \leftarrow \lfloor (1-f)n \rfloor$ ;
   // Set threshold $\theta$:
 9 if $nobj_{acc} == 0$ then
10     $\theta \leftarrow -\infty$;
11 else if $nobj_{acc} == n$ then
12     $\theta \leftarrow \infty$ ;
13 else
14     $\theta \leftarrow relDist[nobj_{acc}]$
15 end
16 Compute $n^{'} \times n$ distance matrix $D^{'}$ between $\{(\mathcal{X}_q)_{n^{'} \times p}, (\mathcal{X}_{tr})_{n \times p}\}$ ;
   // Find k-nn neighbors of each test object:
17 for $i$ = 1,...,$n^{'}$ do
18     Find set $\mathcal{N}^{'}$ of $k$ nearest neighbor indices from $\mathcal{X}_{tr}$ to object $x_i^{'} \in \mathcal{X}_q$;
19     Compute the median: $x_m = \underset{j \in \mathcal{N}^{'}}{\text{median}}(x_j)$ ;
20     Compute distance: $relDist_i^{'} = ||x_i^{'} - x_m||$ ;
21 end
22 Compute query labels: $l_q \leftarrow \mathbb{I}(relDist^{'} > \theta)$; $\mathbb{I}$ := element-wise indicator function ;
   Output: Query labels $l_q$ (0:=inlier, 1:=outlier)
```

Algorithm 3 requires the computation of distance matrices D (line 1) and $D^{'}$ (line 16) which is feasible for datasets consisting of a few thousand samples depending on system RAM allocation. Therefore, for large datasets, we introduce a scalable version: Algorithm 4 replaces lines 1-15, and a similar procedure replaces lines 16-21. Moreover, the input k can be a vector instead of a scalar, and the output will be a list of test labels for each case of k.

Algorithm 4: Scalable MOCC: replaces lines 1-15 in Algorithm 3

```
Input : Training data $(\mathcal{X}_{tr})_{n\times p}$, vector $k = (k_1, ..., k_r)$ of number of neighbors, maximum
        block width or height $m$
1  Create $\lceil n/m \rceil (= l)$ blocks $(Q_{m\times p})_i; i = 1, ..., l$ from $\mathcal{X}_{tr}$ ;
2  $k\_max \leftarrow max(k)$ ;
3  for $i$ = 1,...,$l$ do
4      for $j$ = 1,...,$l$ do
5          Compute $m \times m$ distance matrix $(b_{m\times m})_{ij}$ between $\{Q_i, Q_j\}$ ;
6          Store indices of $k\_max$ nearest neighbors for each object $x \in Q_i$ in
            $U_{ij} := [u_{yz}]_{ij}; y = 1, ..., m; z = 1, ..., k\_max$ ;
7          Store $k\_max$ nearest neighbor distances for each object $x \in Q_i$ in
            $D_{ij} := [d_{yz}]_{ij}; y = 1, ..., m; z = 1, ..., k\_max$ ;
8      end
9  end
10 for $i$ = 1,...,$r$ do
11     Find indices $\mathcal{L}$ of $k(i)$ nearest neighbors across the $l \times k\_max$ columns of $D$ ;
12     Map indices $\mathcal{L}$ using $U$ to index locations $\mathcal{L}^{'}$ in hypothetical full distance matrix over
        $\mathcal{X}_{tr}$ ;
13     Trim out the last $(ml - n)$ rows of $\mathcal{L}^{'}$ giving $\mathcal{L}^{''}$ ;
14     for $i$ = 1,...,$n$ do
15         Set $\mathcal{N}$ as $i^{th}$ row of $\mathcal{L}^{''}$ ;
16         Compute the median: $x_m = \underset{j\in\mathcal{N}}{\text{median}}(x_j)$ ;
17         Compute distance: $relDist_i = ||x_i - x_m||$ ;
18     end
19     Sort $relDist$ in ascending order ;
20     Compute number of (training) objects accepted: $nobj_{acc} \leftarrow \lfloor (1 - f)n \rfloor$ ;
       // Set threshold $\theta$:
21     if $nobj_{acc} == 0$ then
22         $\theta(i) \leftarrow -\infty$;
23     else if $nobj_{acc} == n$ then
24         $\theta(i) \leftarrow \infty$ ;
25     else
26         $\theta(i) \leftarrow relDist[nobj_{acc}]$
27     end
28 end
```

5. Application

In this section, we test the performance of the relative density one-class classifier and the median one-class classifier along with three other classifiers on artificial datasets as well as the NSL-KDD dataset (Tavallaee et al., 2009).

5.1 One-class classifiers

In addition to RDOCC and MOCC, we experiment with three well-known one-class classifiers, namely the isolation forest classifier (Liu et al., 2008), support vector machine (Tax, 2001), and local outlier factor (Breunig et al., 2000). We present a summary of all classifiers and describe their usage in the experiments that follow.

5.1.1 Relative density classifier (RDOCC)

As explained in a previous section, RDOCC relies on a comparison between kernel density estimation at the local level and the wider neighborhood level. This gives it greater flexibility than the Parzen one-class classifier but also results in an additional hyperparameter, namely k, which needs to be tuned. In general, RDOCC is tunable through its three hyperparameters: h, k, and f. To some extent, RDOCC can be viewed as a hybrid between the Parzen one-class classifier (Tax, 2001) and the k-nearest neighbor one-class classifier (Ramaswamy et al., 2000): like the latter, it incorporates a neighborhood measure, but like the former, it is based on density rather than distance. In the experiments described in this work, the scalable version of RDOCC is used throughout and is denoted by `rdocc_scalable` in the code repository.

5.1.2 Median one-class classifier (MOCC)

MOCC relies on the distance from an object to the median of its k-nearest neighbors belonging to the inlier training set. MOCC has one fewer hyperparameter to tune than RDOCC. In general, MOCC is tunable through its two hyperparameters: k and f. As a distance-based classifier, MOCC shares some features with the k-nearest neighbor one-class classifier as it incorporates a neighborhood measure. In the experiments described in this work, the scalable version of MOCC is used throughout and is denoted by `mocc_scalable` in the code repository.

5.1.3 Isolation forest (ISF)

The isolation forest classifier (Liu et al., 2008) assigns an anomaly score for an object based on the expected tree depth until the object is isolated at a leaf. Outliers require fewer splits to become isolated, and so their path length down the tree would be short compared to inliers. We use an `R` implementation of isolation forest through the function isolationForest in the `solitude` package. We introduce the reject fraction option through a wrapper function around isolationForest which we denote by iForest in the code repository. In iForest, the anomaly scores of training set inliers are first obtained through an internal call to isolationForest. The anomaly scores returned by isolationForest are then sorted and a threshold is selected from these based on the reject fraction. Finally, anomaly score predictions over test set objects are computed using the trained model, and the scores are transformed to hard labels ("1" for outliers and "0" for inliers) according to whether each score is above the threshold or not.

5.1.4 One-class support vector machine (SVM)

The one-class support vector machine (Tax, 2001) tries to fit a hypersphere around the target (inlier) class while allowing for slack variables associated with each object. The tradeoff between setting the hypersphere margin (radius) and penalizing slack variables is controlled by a regularization hyperparameter. With the use of a kernel function, the assumption of a hypersphere is relaxed and the resulting decision boundaries can be very complex in the original feature space. We use an `R` implementation of one-class ν-SVM (Schölkopf et al., 2001) through the function svm in the `e1071` package which in turn is based on libsvm

(Chang and Lin, 2011). We select a radial basis function (RBF) as kernel type which introduces a hyperparameter γ controlling the 'width' of the kernel. The hyperparamter γ, in addition to regularization parameter ν require tuning, and this is done by grid search.

5.1.5 Local outlier factor (LOF)

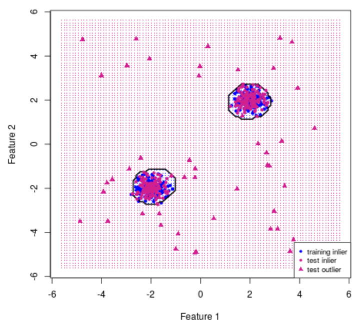
The local outlier factor algorithm (Breunig et al., 2000) is related to the k-nearest neighbor one-class classifier and is based on the idea of "reachability distance" to a given object from its neighbors. The LOF algorithm constructs a form of density estimate based on reachability distance and is also related to RDOCC in that it compares a local measure to that of neighbors. It does so by considering the ratio between local and wider neighbor reachability densities. We use an `R` implementation of local outlier factor through the function lof in the `Rlof` package. We introduce the reject fraction option through a wrapper function around lof which we denote by lof_wrapper in the code repository. Note that LOF has no prediction functionality for new test objects because the method is unsupervised and simply returns anomaly scores for each object in a given data configuration. Introducing new test objects all at once results in a new data configuration, affecting neighborhoods and anomaly scores. Therefore, we have extended LOF externally by allowing for anomaly score prediction through adding one test object at any given time to the training set. That is, we iterate through all test objects one at a time to obtain their anomaly scores in relation to the static training set. In this manner, even if a group of outlier test objects which are closely similar form their own cluster, each of these objects will be independently assigned a high anomaly score. In lof_wrapper, the anomaly scores over the training set are first obtained through an internal call to lof. The anomaly scores are then sorted and a threshold, θ, is selected from these based on the reject fraction. Finally, anomaly score predictions over test set objects are computed by iteratively adding only one test object to the training set, computing its anomaly score, and then removing it, including the next test object, and repeating the process. The scores are transformed to hard labels ("1" for outliers and "0" for inliers) according to whether each score is above the threshold θ or not. Note that in the experiments that follow we limit the size of training, validation, and test sets in the case of LOF due to the demanding runtime required for this method (lof function) on a single system despite the option of multithreading. Also, generating the decision boundary for visualization in the case of artificial data in 2D is avoided due to the runtime complexity involved.

5.2 Artificial data: proof of concept

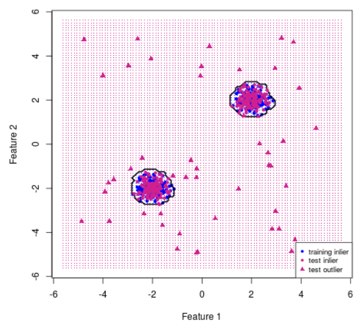
As proof of concept, we experiment with RDOCC and MOCC on artificial data and compare to ISF, SVM, and LOF. We test and visualize the performance of these classifiers on four artificially generated, two-dimensional datasets. In all experiments that follow, the hyperparameters for any given classifier are tuned by performing a grid search to minimize the following error related to *balanced accuracy*: $e = (FPR + FNR)/2$ where $FPR = \frac{FP}{FP+TN}$ is the false positive rate and $FNR = \frac{FN}{FN+TP}$ is the false negative rate with $TP =$ true positive, TN = true negative, FP = false positive, and FN = false negative. Source code and an HTML document for the experiments are available in GitHub[3].

5.2.1 Gaussian dataset

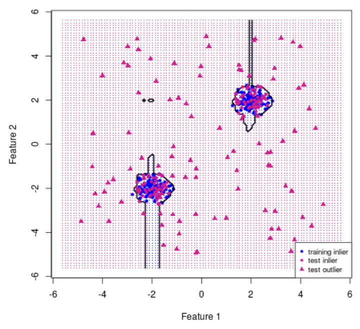
This dataset consists of two circularly symmetric Gaussian distributions of inliers a distance apart in 2D. Outliers are generated from a uniform distribution. Figure 1(a-b) shows the dataset and the decision boundary resulting from training RDOCC and MOCC. The training phase involves a training set consisting purely of inliers and a validation set for tuning hyperparameters consisting of inliers and outliers. Performance measures on the test set are shown in Table 1 in terms of sensitivity, specificity, balanced accuracy, precision, recall, and F_1-score. The decision boundaries of ISF, SVM, and LOF are also visualized in Figure 1(c-e). The receiver operating characteristic (ROC) curve for each classifier is shown in Figure 5(a), and the area under the ROC curve is shown in Table 5.



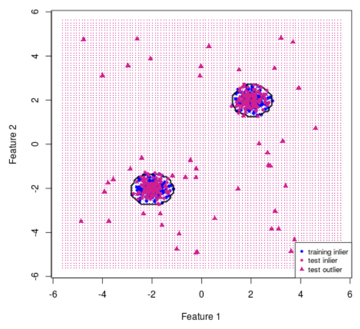
(a)



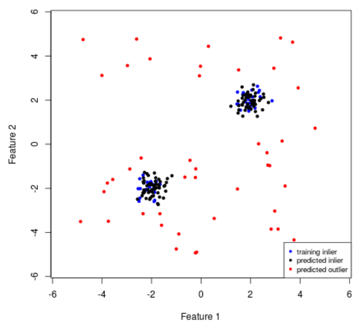
(b)



(c)



(d)



(e)

Figure 1: "Gaussian" dataset: (a) RDOCC (b) MOCC (c) Isolation forest (d) Support vector machine (e) Local outlier factor

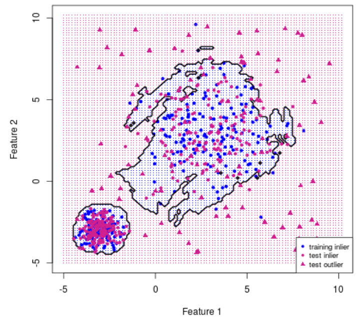
3. https://jimmyazar.github.io/OCC/demo_rdocc_mocc.html

Confusion matrix measures					
	Specificity	Sensitivity (= Recall)	Balanced accuracy	Precision	F_1-score
RDOCC	0.98	0.90	0.94	0.9	0.9
MOCC	0.988	0.94	0.964	0.94	0.94
ISF	0.952	0.94	0.946	0.797	0.862
SVM	0.96	0.94	0.95	0.825	0.879
LOF	1	0.9	0.95	1	0.947

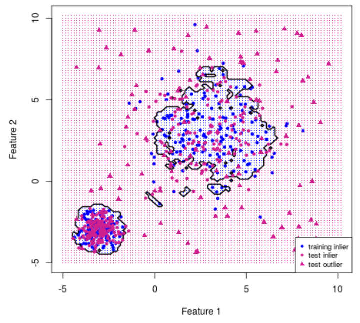
Table 1: "Gaussian" dataset: confusion matrix measures for different classifiers.

5.2.2 variableDensity dataset

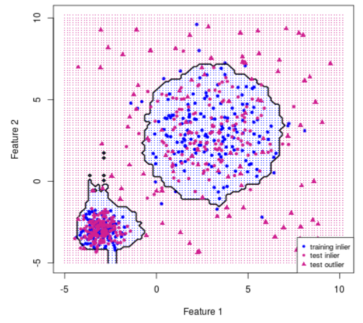
This dataset consists of two isotropic Gaussian distributions of inliers in 2D with different covariance matrices. The two Gaussians are such that one has a mean $\mu_1 = (-3, -3)$ and covariance matrix $\Sigma_1 = \begin{pmatrix} 0.3 & 0 \\ 0 & 0.3 \end{pmatrix}$, and the other has a mean $\mu_2 = (3, 3)$ and covariance matrix $\Sigma_2 = \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix}$. Outliers are generated from a uniform distribution. Note that this dataset is challenging for simple kernel density estimation not based on a relative difference measure since the less dense area may be wrongly assigned to the outlier class. RDOCC however is based on comparing local density (as contributed by the inlier training set) to that of neighboring objects. Figure 2(a-b) shows the dataset and the decision boundary resulting from training RDOCC and MOCC. Performance measures on the test set are shown in Table 2. The decision boundaries of ISF, SVM, and LOF are also visualized in Figure 2(c-e). The receiver operating characteristic (ROC) curve of each classifier is shown in Figure 5(b), and the area under the ROC curve is shown in Table 5.



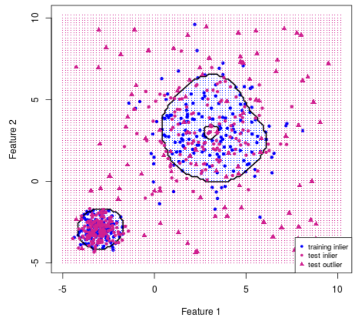
(a)



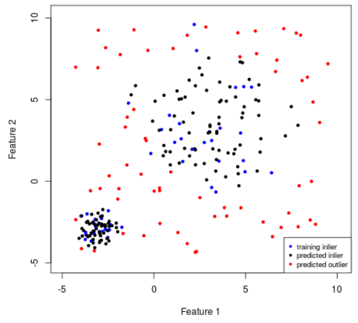
(b)



(c)



(d)



(e)

Figure 2: "variableDensity" dataset: (a) RDOCC (b) MOCC (c) Isolation forest (d) Support vector machine (e) Local outlier factor

Confusion matrix measures					
	Specificity	Sensitivity (= Recall)	Balanced accuracy	Precision	F_1-score
RDOCC	0.947	0.68	0.813	0.81	0.739
MOCC	0.84	0.82	0.83	0.631	0.713
ISF	0.95	0.69	0.82	0.821	0.75
SVM	0.813	0.84	0.827	0.6	0.7
LOF	0.96	0.65	0.805	0.942	0.769

Table 2: "variableDensity" dataset: confusion matrix measures for different classifiers.

5.2.3 Boomerang dataset

This dataset consists of two boomerang-shaped branches of inliers, with the addition of outliers generated from a uniform distribution. Figure 3(a-b) shows the dataset and the decision boundary resulting from training RDOCC and MOCC. Performance measures on the test set are shown in Table 3. The decision boundaries of ISF, SVM, and LOF are also visualized in Figure 3(c-e). The receiver operating characteristic (ROC) curve of each classifier is shown in Figure 5(c), and the area under the ROC curve is shown in Table 5.

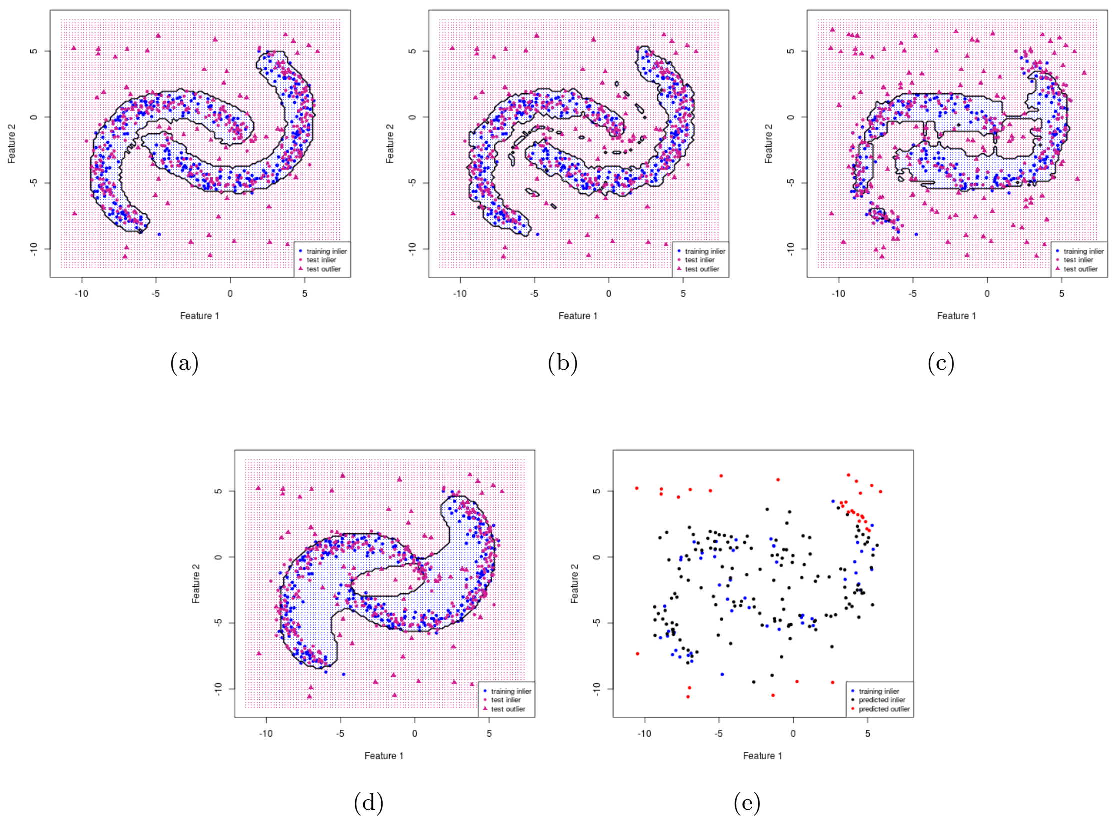


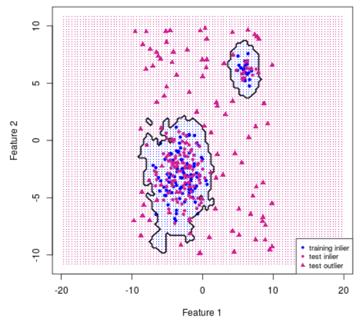
Figure 3: “boomerang” dataset: (a) RDOCC (b) MOCC (c) Isolation forest (d) Support vector machine (e) Local outlier factor

Confusion matrix measures					
	Specificity	Sensitivity (= Recall)	Balanced accuracy	Precision	F_1-score
RDOCC	0.95	0.733	0.842	0.786	0.759
MOCC	0.89	0.68	0.785	0.607	0.642
ISF	0.737	0.72	0.728	0.406	0.519
SVM	0.837	0.627	0.732	0.49	0.55
LOF	0.91	0.32	0.615	0.727	0.444

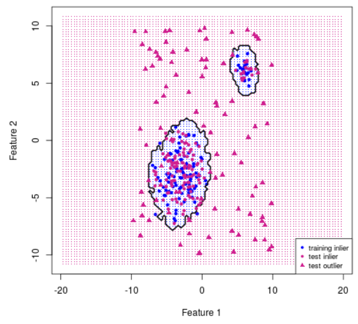
Table 3: “boomerang” dataset: confusion matrix measures for different classifiers.

5.2.4 Island dataset

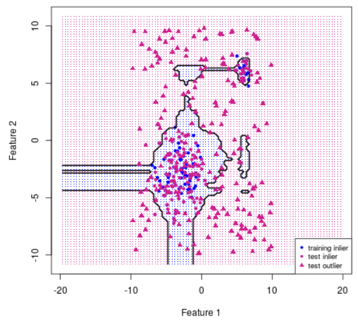
This dataset consists of a main Gaussian distribution and a distant, relatively small “island” of objects. Depending on the hyperparameter k, the relative density classifier is able to label the island as inlier (for small k) or outlier (for large k). All hyperparameters are optimized using a validation set. Figure 4(a-b) shows the dataset and the decision boundary resulting from training RDOCC and MOCC. Performance measures on the test set are shown in Table 4. The decision boundaries of ISF, SVM, and LOF are also visualized in Figure 4(c-e). The receiver operating characteristic (ROC) curve of each classifier is shown in Figure 5(d), and the area under the ROC curve is shown in Table 5.



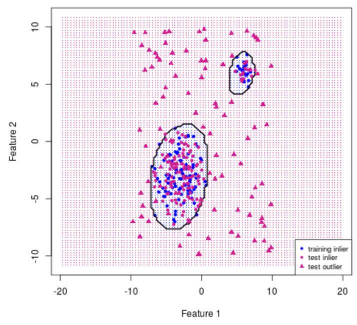
(a)



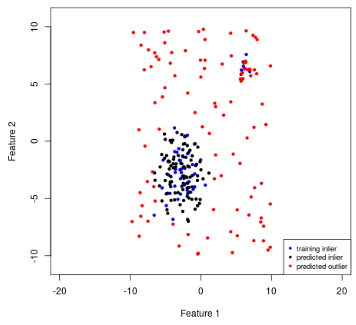
(b)



(c)



(d)



(e)

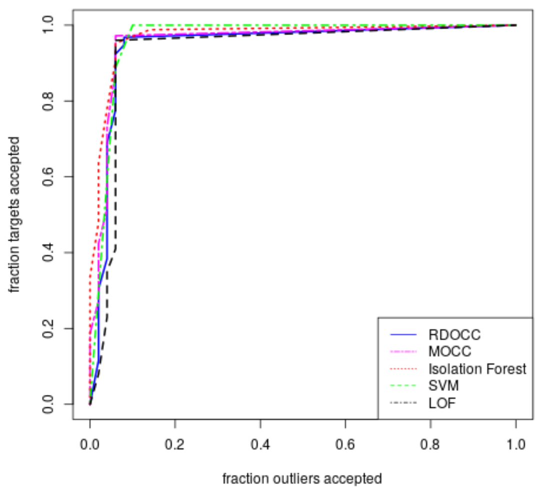
Figure 4: “island” dataset: (a) RDOCC (b) MOCC (c) Isolation forest (d) Support vector machine (e) Local outlier factor

Confusion matrix measures					
	Specificity	Sensitivity (= Recall)	Balanced accuracy	Precision	F_1-score
RDOCC	1	0.81	0.905	1	0.895
MOCC	0.975	0.83	0.903	0.966	0.892
ISF	0.975	0.76	0.868	0.962	0.849
SVM	0.983	0.85	0.917	0.977	0.909
LOF	0.83	0.86	0.845	0.835	0.847

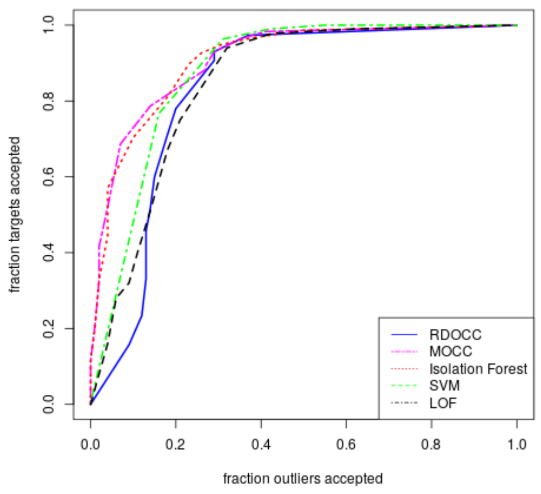
Table 4: “island” dataset: confusion matrix measures for different classifiers.

5.2.5 ROC/AUC Comparison

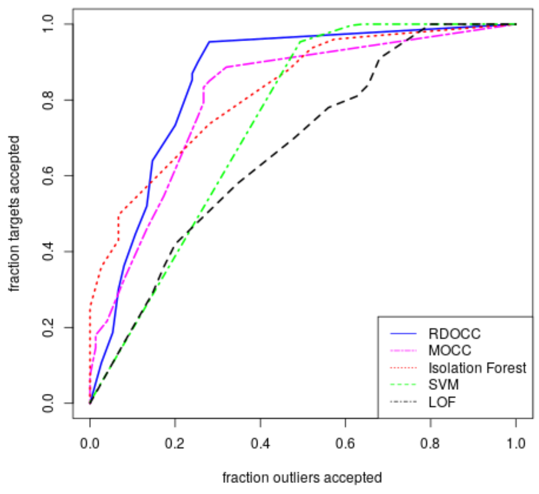
To compare the performance of classifiers we use an ROC curve as a plot of true positive rate (i.e. sensitivity) versus false positive rate (i.e. 1−specificity). The ROC curves for various classifiers across the different datasets are shown in Figure 5, and the AUC values are shown in Table 5. The isolation forest classifier appears to outperform the other classifiers on the Gaussian dataset, whereas MOCC performs best on the variableDensity dataset, and RDOCC outperforms others on the boomerang and island dataset.



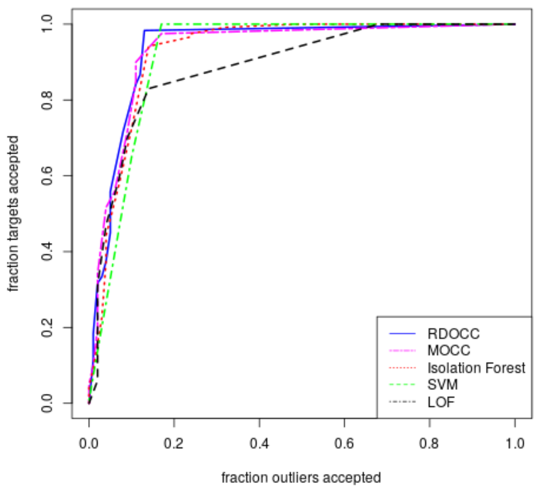
(a)



(b)



(c)



(d)

Figure 5: ROC curves: (a) Gaussian dataset (b) variableDensity dataset (c) boomerang dataset (d) island dataset

Area under the ROC curve (AUC)				
	Gaussian dataset	variableDensity dataset	boomerang dataset	island dataset
RDOCC	0.947	0.834	**0.852**	**0.937**
MOCC	0.957	**0.91**	0.809	0.932
ISF	**0.972**	0.909	0.824	0.928
SVM	0.965	0.878	0.74	0.921
LOF	0.933	0.846	0.671	0.89

Table 5: AUC values for various classifiers

5.3 NSL-KDD dataset

As an application in anomaly detection, we test out RDOCC and MOCC on the NSL-KDD dataset (Tavallaee et al., 2009) which is an improved and more challenging version of the original KDD dataset ("University of California, Irvine", 1999) released by DARPA, in which redundancies were removed. The NSL-KDD dataset includes 41 network traffic attributes with a training set that includes 125,973 instances (67,343 normal; 58,630 attack) and a test set that includes 22,544 instances (9711 normal; 12833 attack). Apart for the normal class, the training set contains 22 different types of attacks, and the test set contains 37 types of attacks; both are listed in Table 6. The different types of attacks can be grouped into four main classes: probing attacks, denial of service (DoS) attacks, user-to-root attacks (U2R), and remote-to-local attacks (R2L).

There are 17 attack types that appear only in the test set and not in the training set: apache2 (DoS), httptunnel (R2L), mailbomb (DoS), mscan (Probing), named (R2L), processtable (DoS), ps (U2R), saint (Probing), sendmail (R2L), snmpgetattack (R2L), snmpguess (R2L), sqlattack (U2R), udpstorm (DoS), worm (DoS), xlock (R2L), xsnoop (R2L), and xterm (U2R). In this work, we do not discard these attack types or treat them as an "unknown" class, but rather consider these as relating to novel probing, DoS, U2R, or R2L attacks.

We build a network intrusion detection system (IDS) based on RDOCC and MOCC. We address binary classification (normal versus attack) as well as multiclass classification with classes representing normal cases, probing attacks, DoS, U2R, and R2L attacks. The multiclass situation is handled by constructing five one-class classifiers in a one-against-all setting, followed by a majority voting scheme. Source code and an HTML document for the experiments are available in GitHub[4].

4. https://jimmyazar.github.io/OCC/demo_NSL_KDD.html

Label type	**#Training**	**#Test**	**Label type**	**#Training**	**#Test**
apache2	0	737	portsweep	2,931	157
back	956	359	processtable	0	685
buffer_overflow	30	20	ps	0	15
ftp_write	8	3	rootkit	10	13
guess_passwd	53	1,231	saint	0	319
httptunnel	0	133	satan	3,633	735
imap	11	1	sendmail	0	14
ipsweep	3,599	141	smurf	2,646	665
land	18	7	snmpgetattack	0	178
loadmodule	9	2	snmpguess	0	331
mailbomb	0	293	sqlattack	0	2
mscan	0	996	teardrop	892	12
multihop	7	18	udpstorm	0	2
named	0	17	warezmaster	20	944
neptune	41,214	4,657	worm	0	2
nmap	1,493	73	xlock	0	9
normal	67,343	9,711	xsnoop	0	4
perl	3	2	xterm	0	13
phf	4	2	spy	2	0
pod	201	41	warezclient	890	0

Table 6: Raw class label types in NSL-KDD dataset

5.3.1 Binary classification

For each classifier (RDOCC, MOCC, ISF, SVM, LOF), we build and test an IDS for differentiating between normal network traffic data and attacks (probing, DoS, U2R, R2L). Nominal features are first one-hot encoded, and the training data is normalized. Then we apply principal component analysis (PCA) to reduce dimensionality down to four features. A plot of the first two principal components is shown in Figure 6. Using a subset of the training set, we tune hyperparameters, and then measure performance on the test set. Measures derived from the confusion matrix are summarized in Table 7, and a plot of ROC curves is shown in Figure 7 along with AUC values in Table 7. The best performing classifiers in this case are MOCC and ISF.

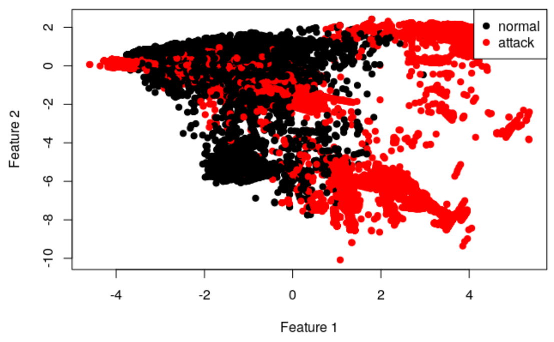


Figure 6: First two principal components of PCA: normal versus attack

Confusion matrix measures & AUC						
	Specificity	Sensitivity (= Recall)	Balanced accuracy	Precision	F_1-score	AUC
RDOCC	0.968	0.641	0.804	0.964	0.77	0.77
MOCC	0.879	0.888	0.884	0.907	0.897	0.923
ISF	0.934	0.667	0.801	0.931	0.777	0.911
SVM	0.982	0.592	0.787	0.978	0.737	0.663
LOF	0.79	0.46	0.625	0.687	0.551	0.821

Table 7: normal-vs.-attack following PCA

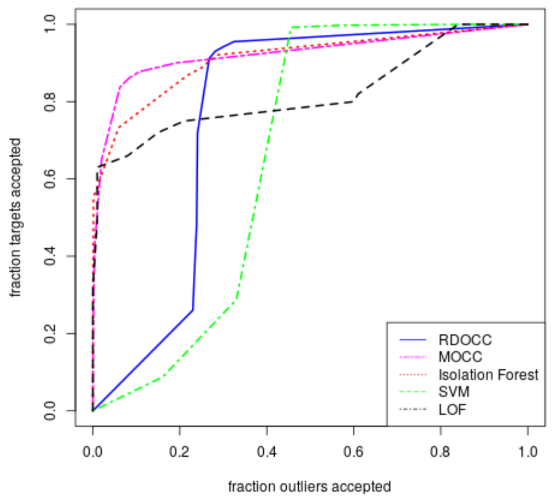


Figure 7: ROC curves: normal versus attack following PCA

5.3.2 Multiple one-class classifications

We emulate a multiclass classifier by training multiple one-class classifiers and combining their decisions by a majority vote. Each one-class classifier is trained in a one-against-all scheme for each of the five classes. Ties are broken by assigning to the class with the nearest mean from among the classes that happen to be in tie. In case an object is rejected as outlier by all 5 classifiers, it is assigned to the class with the nearest mean.

Following normalization based on the training set, the Fisher mapping is applied to reduce dimensionality of the multiclass dataset. Then we consider one class at a time against all other classes, and train the classifiers and test their performances. Results for all five classes are shown below.

Normal-versus-all: Measures derived from the confusion matrix are summarized in Table 8, and a plot of ROC curves is shown in Figure 8 along with AUC values in Table 8.

Confusion matrix measures & AUC						
	Specificity	Sensitivity (= Recall)	Balanced accuracy	Precision	F_1-score	AUC
RDOCC	0.904	0.626	0.765	0.896	0.737	0.697
MOCC	0.884	0.902	0.893	0.911	0.907	0.927
ISF	0.914	0.803	0.858	0.925	0.859	0.935
SVM	0.954	0.753	0.854	0.956	0.843	0.883
LOF	0.95	0.67	0.81	0.931	0.779	0.953

Table 8: normal-vs.-all

Probing-versus-all: Measures derived from the confusion matrix are summarized in Table 9, and a plot of ROC curves is shown in Figure 8 along with AUC values in Table 9.

Confusion matrix measures & AUC						
	Specificity	Sensitivity (= Recall)	Balanced accuracy	Precision	F_1-score	AUC
RDOCC	0.681	0.864	0.772	0.957	0.908	0.795
MOCC	0.34	0.989	0.664	0.926	0.956	0.685
ISF	0.641	0.91	0.776	0.955	0.932	0.811
SVM	0.835	0.459	0.647	0.959	0.621	0.754
LOF	0.85	0.88	0.865	0.854	0.867	0.863

Table 9: probing-vs.-all

DoS-versus-all: Measures derived from the confusion matrix are summarized in Table 10, and a plot of ROC curves is shown in Figure 8 along with AUC values in Table 10.

Confusion matrix measures & AUC						
	Specificity	Sensitivity (= Recall)	Balanced accuracy	Precision	F_1-score	AUC
RDOCC	0.807	0.838	0.823	0.898	0.867	0.804
MOCC	0.659	0.999	0.829	0.855	0.922	0.844
ISF	0.671	0.998	0.835	0.86	0.924	0.857
SVM	0.69	0.994	0.842	0.866	0.926	0.831
LOF	0.62	0.98	0.8	0.721	0.831	0.755

Table 10: DoS-vs.-all

U2R-versus-all: Measures derived from the confusion matrix are summarized in Table 11, and a plot of ROC curves is shown in Figure 8 along with AUC values in Table 11.

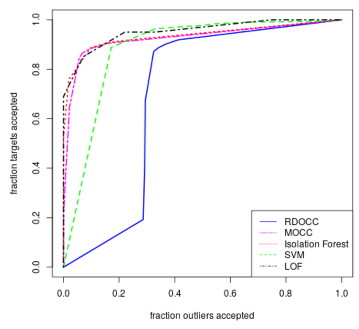
Confusion matrix measures & AUC						
	Specificity	Sensitivity (= Recall)	Balanced accuracy	Precision	F_1-score	AUC
RDOCC	0.522	0.545	0.534	0.997	0.705	0.581
MOCC	0.657	0.412	0.534	0.998	0.583	0.623
ISF	0.582	0.779	0.681	0.998	0.875	0.764
SVM	0.448	0.953	0.7	0.998	0.975	0.671
LOF	0.313	0.96	0.637	0.676	0.793	0.467

Table 11: U2R-vs.-all

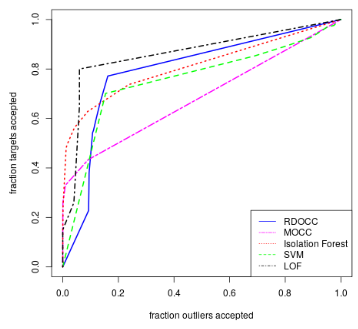
R2L-versus-all: Measures derived from the confusion matrix are summarized in Table 12, and a plot of ROC curves is shown in Figure 8 along with AUC values in Table 12.

Confusion matrix measures & AUC						
	Specificity	Sensitivity (= Recall)	Balanced accuracy	Precision	F_1-score	AUC
RDOCC	0.244	0.981	0.613	0.898	0.938	0.689
MOCC	0.058	0.999	0.529	0.878	0.935	0.623
ISF	0.114	0.996	0.555	0.884	0.937	0.723
SVM	0.227	0.975	0.601	0.896	0.934	0.668
LOF	0.35	1	0.675	0.606	0.755	0.819

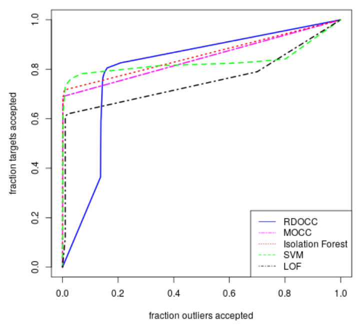
Table 12: R2L-vs.-all



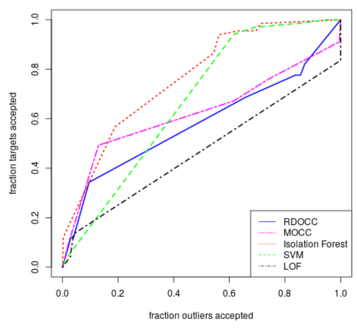
(a)



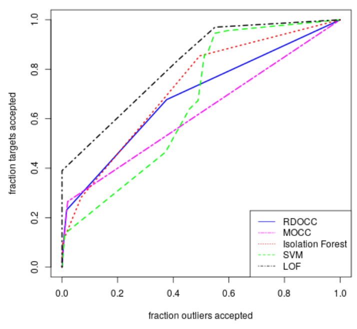
(b)



(c)



(d)



(e)

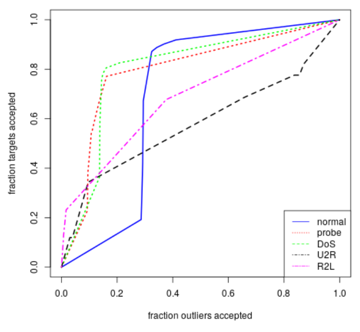
Figure 8: ROC curves: (a) Normal versus all (b) Probing versus all (c) DoS versus all (d) U2L versus all (e) R2L versus all

Majority vote: For a given classifier (e.g. RDOCC), its five one-class classifications are combined through a majority vote when labeling a new test object. The resulting confusion matrices for each classifier across the five classes are shown in Table 13, along with the

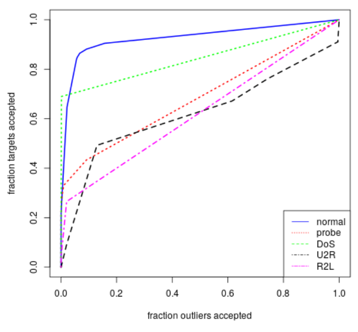
ROC curves for each class in Figure 9 and AUC values in Table 14. Note that for LOF, the confusion matrix is not available in Table 13 because due to runtime constaints the test set used was a random variable subset of the full test set (i.e. not fixed), and therefore the majority rule cannot be used to combine decisions.

		Normal	Probing	DoS	U2R	R2L
RDOCC	Normal	8756	321	1410	43	1633
	Probing	678	1775	635	0	14
	DoS	165	270	5397	0	82
	U2R	49	55	13	17	506
	R2L	63	0	3	7	652
		Normal	Probing	DoS	U2R	R2L
MOCC	Normal	8677	291	461	11	981
	Probing	213	1688	531	0	14
	DoS	60	232	5674	0	0
	U2R	752	210	792	55	1737
	R2L	9	0	0	1	155
		Normal	Probing	DoS	U2R	R2L
ISF	Normal	8931	398	987	22	2110
	Probing	651	1780	535	0	12
	DoS	85	228	5605	0	0
	U2R	28	15	331	40	358
	R2L	16	0	0	5	407
		Normal	Probing	DoS	U2R	R2L
SVM	Normal	8957	329	1303	25	2078
	Probing	633	1958	704	0	31
	DoS	62	134	5451	0	0
	U2R	20	0	0	35	22
	R2L	39	0	0	7	756

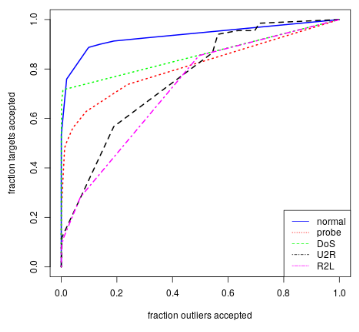
Table 13: Confusion matrices



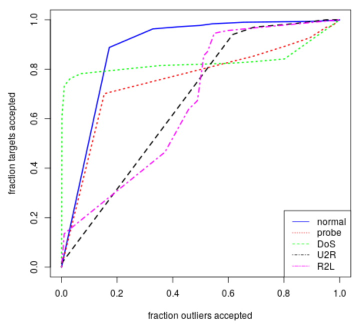
(a)



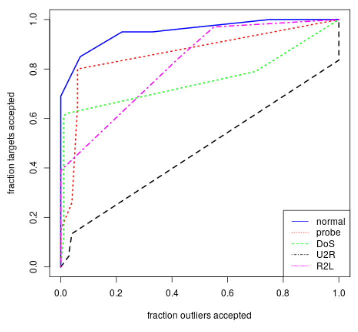
(b)



(c)



(d)



(e)

Figure 9: ROC curves: (a) RDOCC (b) MOCC (c) ISF (d) SVM (e) LOF

Area under the ROC curve (AUC)					
	Normal-vs-all	Probing-vs-all	DoS-vs-all	U2R-vs-all	R2L-vs-all
RDOCC	0.697	0.795	0.804	0.581	0.689
MOCC	0.927	0.685	0.844	0.623	0.623
ISF	0.935	0.811	0.857	0.764	0.723
SVM	0.883	0.754	0.831	0.671	0.668
LOF	0.953	0.863	0.755	0.467	0.819

Table 14: AUC values for various classifiers

Table 14 shows that the results in general are close among classifiers. LOF appears to be performing the best on this dataset for 3 out of 5 of one-against-all settings, with very close results compared to ISF and MOCC in the normal-vs-all case, and close results to ISF in probing-vs-all, and ISF in R2L-vs-all. However the best classfiers are ISF and MOCC in DoS-vs-all, and ISF in U2R-vs-all. RDOCC is able to outperform MOCC in probing-vs-all and R2L-vs-all.

We note that it is possible to use the majority rule to combine *different* classifiers, choosing the best-performing one for each of the one-against-all settings. This would expectedly result in an improved IDS, however in this work, we are mainly interested in comparing capabilities of the different classifiers.

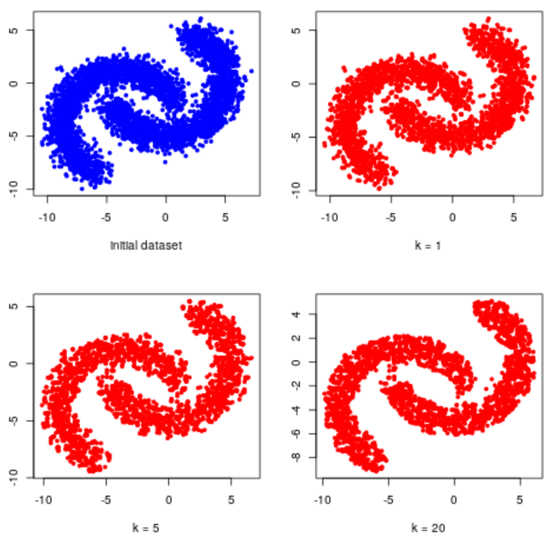
6. Median transform

In line with MOCC, if we *replace* each training object with the median of its k-nearest neighbors, we create a new dataset, close to the original, but which may be comprised only of inliers. We denote this method of filtering or data transformation by the *median transform*. The median transform is a pre-processing step that can be applied before training a one-class classifier, especially in the unsupervised case when it is not known which training objects are inliers or outliers. With the resulting dataset assumed to consist solely of target objects, any one-class classifier can be trained on the median-transformed data.

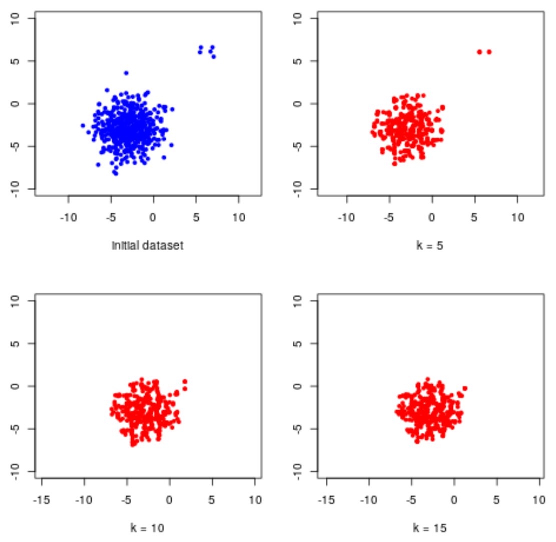
In summary, the median transform takes as input an unlabeled data matrix and transforms it into another of the same size. As with MOCC, in computing a median object, we identify the k-nearest neighbors and consider the median value of each feature from among these. The transform requires setting the parameter k representing the number of neighbors to consider. The larger k is, the greater is the effect of the median transform in eliminating outlier objects by replacing these with more in-bound objects.

We provide a scalable implementation of the median transform in `R` as `mediantx`. It accepts an unlabeled data matrix and a single parameter k, the number of neighbors. Figure 10 shows the effect of the transform on the “boomerang” and “island” datasets for different values of k. Source code and an HTML document for the experiments are available in GitHub[5].

5. `https://jimmyazar.github.io/OCC/demo_median_transform.html`



(a)



(b)

Figure 10: Median transform: (a) “boomerang” dataset (b) “island” dataset

7. Conclusions

The two classifiers RDOCC and MOCC presented in this work add to the arsenal of one-class classifiers and to its diversity which is useful when tackling various types of imbalanced data. As with any classifier, performance is data-dependent to an extent. In general, RDOCC is expected to perform well when enough objects are provided for accurate density estimation, and MOCC is generally a simple, yet powerful classifier requiring one less hyperparameter to tune compared to RDOCC. The reliance of these classifiers on distance matrices would normally result in poor memory scalability which would hinder their usage on large datasets. This drawback was identified early on in this work, and novel scalable algorithms were presented for both classifiers to circumvent this problem. Both RDOCC and MOCC can result in complex decision boundaries and have shown good performance on par with other one-class classifiers such as isolation forest, one-class support vector machine, and local outlier factor. A shortcoming of RDOCC and MOCC is that they are not inherently probabilistic. Possibilities for future work include extending the classifiers to have a probabilistic output as well as combining the classifiers when decomposing the multiclass case through more sophisticated ensemble methods which make use of soft labels and a parallel implementation.

References

J. Andrews, E. Morton, and L. Griffin. Detecting anomalous data using auto-encoders. *International Journal of Machine Learning and Computing*, 6:21, 2016.

C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, 1995.

H. Bostani and M. Sheikhan. Modification of supervised opf-based intrusion detection systems using unsupervised learning and social network concept. *Pattern Recognition*, 62:56–72, 2017.

M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, volume 29, pages 93–104, 2000.

C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui. A comparative evaluation of outlier detection algorithms: Experiments and analyses. *Pattern Recognition*, 74:406–421, 2018.

S. M. Erfani, S. Rajasegarar, S. Karunasekera, and C. Leckie. High-dimensional and large-scale anomaly detection using a linear one-class svm with deep learning. *Pattern Recognition*, 58(12):121–134, 2016.

L. Feinstein, D. Schnackenberg, R. Balupari, and D. Kindred. Statistical approaches to ddos attack detection and response. In *Proceedings of DARPA Information Survivability Conference and Exposition*, pages 303–314, 2003.

M. A. Girolami and C. He. Probability density estimation from optimally condensed data samples. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1253–1264, 2003.

S. Gurung, M. K. Ghose, and A. Subedi. Deep learning approach on network intrusion detection system using nsl-kdd dataset. *International Journal of Computer Network and Information Security*, 11(3):8, 2019.

W. Hu, Y. Liao, and V. R. Vemuri. Robust support vector machines for anomaly detection in computer security. In *Proceedings of the International Conference on Machine Learning and Applications*, pages 168–174, 2003.

T. Kohonen. *Self-organizing maps.* Springer series in information sciences, 30. Springer, Berlin, 3rd edition, 2001.

B. Krawczyk, M. Woźniak, and F. Herrera. On the usefulness of one-class classifier ensembles for decomposition of multi-class problems. *Pattern Recognition*, 48(12):3969–3982, 2015.

B. Krawczyk, M. Galar, M. Woźniak, H. Bustince, and F. Herrera. Dynamic ensemble selection for multi-class classification with one-class classifiers. *Pattern Recognition*, 83: 34–51, 2018.

K. Leung and C. Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In *Proceedings of Twenty-Eighth Australasian Computer Science Conference*, volume 38 of *CRPIT*, pages 333–342, 2005.

Y. Liao and V. R. Vemuri. Use of k-nearest neighbor classifier for intrusion detection. *Computers & Security*, 21(5):439–448, 2002.

F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *Proceedings of Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE Computer Society, 2008.

Z. Liu, G. Florez, and S. M. Bridges. A comparison of input representations in neural networks: A case study in intrusion detection. In *Proceedings of the International Joint Conference on Neural Networks*, volume 2, pages 1708–1713, 2002.

J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

M. M. Moya, M. W. Koch, and L. D. Hostetler. One-class classifier networks for target recognition applications. In *Proceedings of world congress on neural networks*, pages 797–801, 1993.

E. Parzen. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962.

S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In *Proceedings of the ACM SIGMOD international conference on Management of data*, pages 427–438, New York, NY, USA, 2000.

H. Ringberg, A. Soule, J. Rexford, and C. Diot. Sensitivity of pca for traffic anomaly detection. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS*, pages 109–120, 2007.

P. J. Rousseeuw. Least median of squares regression. *American Statistical Association*, 79 (388):871–880, 1984.

B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson. Estimating the support of a high-dimensional distribution. *Neural Computation*, 13(7):1443–1471, 2001.

N. Shone, T. N. Ngoc, V. D. Phai, and Q. Shi. A deep learning approach to network intrusion detection. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2(1): 41–50, 2018.

M.-Y. Su, G.-J. Yu, and C.-Y. Lin. A real-time network intrusion detection system for large-scale attacks based on an incremental mining approach. *Computers & Security*, 28 (5):301–309, 2009.

M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani. A detailed analysis of the kdd cup 99 data set. *Proceedings of the IEEE Symposium on Computational Intelligence for Security and Defense Applications*, pages 1–6, 2009.

D. M. J. Tax. *One-class classification: Concept learning in the absence of counter-examples*. PhD thesis, Technische Universiteit Delft, 2001.

M. Turkoz, S. Kim, Y. Son, M. K. Jeong, and E. A. Elsayed. Generalized support vector data description for anomaly detection. *Pattern Recognition*, 100:107119, 2020.

"University of California, Irvine". UCI Knowledge Discovery in Databases Archive, 1999. URL https://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html. (accessed 2020-06-18).

X. Wang, P. Tiňo, M. A. Fardal, S. Raychaudhury, and A. Babul. Fast parzen window density estimator. In *Proceedings of the International Joint Conference on Neural Networks*, pages 3267–3274, 2009.

N. Ye, S. M. Emran, Q. Chen, and S. Vilbert. Multivariate statistical analysis of audit trails for host-based intrusion detection. *IEEE Transactions on Computers*, 51(7):810–820, 2002.

J. Yu and S. Kang. Clustering-based proxy measure for optimizing one-class classifiers. *Pattern Recognition Letters*, 117:37–44, 2019.

J. Zhang and M. Zulkernine. Anomaly based network intrusion detection with unsupervised outlier detection. In *Proceedings of IEEE International Conference on Communications*, pages 2388–2393, 2006.

C. Zhou and R. C. Paffenroth. Anomaly detection with robust deep autoencoders. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 665–674, 2017.