On the Evaluation of Outlier Detection and One-Class Classification: A Comparative Study of Algorithms, Model Selection, and Ensembles [Supplementary Material]

Henrique O. Marques[1*], Lorne Swersky[2], Jörg Sander[2], Ricardo J. G. B. Campello[3] and Arthur Zimek[1]

[1]University of Southern Denmark, Denmark.
[2]University of Alberta, Canada.
[3]University of Newcastle, Australia.

*Corresponding author(s). E-mail(s): oli@sdu.dk;
Contributing authors: jsander@ualberta.ca;
ricardo.campello@newcastle.edu.au; zimek@imada.sdu.dk;

1 Methods and their Properties

1.1 Description of the Methods

Gaussian Mixture Model (GMM)

The most widely used parametric model is the Gaussian distribution (Bishop, 2007), where, in the most simple case, a single Gaussian probability density function,

$$p_{Gauss}(\mathbf{x}|\boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \, \mathrm{e}^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})}, \tag{1}$$

is fit to the inlier data, with $\boldsymbol{\mu}$ being the mean, Σ being the covariance matrix, and d being the dimensionality of the data. Using a single Gaussian distribution, however, makes a very strong assumption about the distribution being unimodal, which is often violated.

A more flexible model is a Gaussian Mixture Model (GMM) (Bishop, 2007), where the distribution is assumed to be the result of a mixture of different Gaussian distributions. The means μ_i and covariances Σ_i of the individual Gaussian components can efficiently be estimated by an Expectation-Minimization (EM) algorithm (Dempster et al, 1977). Aside from the inherited disadvantages of parametric methods, the main disadvantages of the Gaussian density are related to the covariance matrix. For high dimensional data, this model suffers from very large covariance matrices, which makes it expensive to compute its inverse. In case the inverse of the covariance matrix cannot be calculated (*e.g.* data with singular directions), it has to be approximated, for example, by adding a small constant ϵ to the diagonal, or alternatively, approximated by its pseudo-inverse (Strang, 2016).

Parzen Window (PW)

PW is a non-parametric method based on Parzen Density Estimation (Parzen, 1962) that estimates the density of the data using a mixture of kernels centered on each of the N individual training instances. In our study, we use Gaussian kernels with diagonal covariance matrices $\Sigma_i = hI$, where h is a width parameter. The probability of an instance being an inlier is then computed as:

$$\mathrm{PW}(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} p_{Gauss}(\mathbf{x}|\mathbf{x}_i, hI) \tag{2}$$

The training for the Parzen density consists of the determination of the parameter h, which can be optimized using the maximum likelihood solution (Duin, 1976). Alternatively, the user can supply the parameter h. In the latter case, the computational cost for training is negligible. Testing new instances, however, is computationally expensive. During the testing phase, the distances of the new instance to the training instances have to be computed, which also imposes a storage limitation, since the training instances have to be stored and typically large training sets are required to produce a good density estimation.

Support Vector Data Description (SVDD)

SVDD (Tax and Duin, 2004) is a boundary-based one-class classification method inspired by Support Vector Machines (SVM) (Vapni, 1995) used in regular classification problems. The primary difference between SVDD and SVM is that while an SVM attempts to separate two or more classes with a maximum margin hyperplane, SVDD instead will enclose the inlier class in a minimum volume hypersphere by minimizing the following error:

$$\mathcal{E}(R, \mathbf{a}, \boldsymbol{\xi}) = R^2 + C\sum_i \xi_i, \tag{3}$$

subject to the constraints:

$$\|\mathbf{x}_i - \mathbf{a}\|^2 \leq R^2 + \xi_i, \quad \xi_i \geq 0, \quad \forall i, \tag{4}$$

where R is the radius of the hypersphere, $\mathbf{a}$ is the center of the hypersphere, $\boldsymbol{\xi}$ are slack variables allowing training observations $\mathbf{x}$ to fall outside the SVDD boundary, and C is a penalty (regularization) parameter.

Like traditional SVMs, the above formulation can also be extended to non-linearly transformed spaces using kernel methods. In the case of a kernel with the property $K(\mathbf{x}_i, \mathbf{x}_i) = 1$, $\forall_i$, such as the Gaussian kernel that we use in our experiments, OC-SVM (Schölkopf et al, 2001) and SVDD (Tax and Duin, 2004) find the same decision boundary when some conditions are met (Tax and Duin, 2004; Schölkopf et al, 2001; Tax, 2001).

Linear Programming (LP)

LP (Pekalska et al, 2002) is a boundary method which, instead of using the explicit feature space, utilizes a dissimilarity measure to compare new instances to the inlier instances in the training set. The basic assumption is that instances belonging to the inlier class are similar to each other, while the outliers are dissimilar to inliers. The use of a dissimilarity measure makes the method very convenient for applications where it is difficult to define suitable features for other approaches, as *e.g.* in unstructured data (strings, graphs or shapes). The dissimilarity measure used must meet the criteria defined by the authors: reflectivity, positivity, and symmetry. LP constructs the boundary by minimizing the volume of a simplex (Bazaraa et al, 2009) with the main vertex being the origin and the other vertices resulting from the intersection of the boundary and the axes of the dissimilarity space.

For unbounded dissimilarity measures, instances with large dissimilarities values (due to noise or the existence of outliers in the training data) may affect the decision boundary. In order to make the classifier robust against such values, the dissimilarities are transformed by the sigmoid function, $sigm(x) = \frac{2}{1+e^{x/s}}$, such that large values are bounded by 1. In this case, the user has to define the parameter s for the slope of the sigmoid function.

k-Nearest Neighbor Data Description (kNN_{local})

The k-nearest Neighbor Data Description approach (de Ridder et al, 1998), which we call here kNN_{local}, avoids explicit density estimations by only using the distances to the nearest neighbors. It resembles local outlier detection methods in that it approximates the local (normalized) distance of the training instances, however in a simpler way. An observation is classified under kNN_{local} by computing the ratio between the distance from an observation to its k^{th} nearest neighbor $\text{NN}_\text{k}(\mathbf{x}_i)$, and the distance between the k^{th} nearest neighbor and that neighbor's k^{th} nearest neighbor:

$$\text{kNN}_{local}(\mathbf{x}_i, k) = \frac{d(\mathbf{x}_i, \text{NN}_\text{k}(\mathbf{x}_i))}{d(\text{NN}_\text{k}(\mathbf{x}_i), \text{NN}_\text{k}(\text{NN}_\text{k}(\mathbf{x}_i)))} \tag{5}$$

kNN_{local} has the same computational issues when testing new instances as PW. However, according to Tax (2001) the method performs best in small sample size, where overall computational efforts are low. For larger sample sizes and noisy data, reported quality of results is rather poor (Tax, 2001).

Auto-Encoder Networks

An Auto-Encoder (Japkowicz et al, 1995; Tax, 2001) is a neural network with a sigmoidal transfer function, a single hidden layer, and a parameter-defined number of hidden units trained on the inlier class. The network is trained to reproduce the input patterns at its output layer (*i.e.*, it should perform the identity operation) by minimizing the Mean Square Error (MSE). In order to classify a new instance, the instance is given as input to the network and the difference between the original input and the network's output defines the reconstruction error.

An Auto-Encoder can learn very flexible models. However, it suffers from the same problems as the conventional neural networks for classification problems, such as the high number of parameters to tune (learning rate, number of neurons, number of layers, number of epochs, etc), local minimum, weight initialization, etc. (Freeman and Skapura, 1991; Goodfellow et al, 2016).

Deep SVDD

Deep SVDD (Ruff et al, 2018) is an end-to-end deep learning approach for one-class classification which jointly trains a deep neural network while optimizing a data-enclosing hypersphere of the smallest size in output space. In contrast to kernel-based SVDD, Deep SVDD learns useful feature representations of the data together with the one-class classification objective by employing a neural network that is jointly trained to map the data into a hypersphere of minimum volume. The Deep SVDD minimizes the following objective function:

$$\frac{1}{N}\sum_{i=1}^{N}\|\phi(\mathbf{x}_i, W) - c\|^2 + \frac{\delta}{2}\sum_{j=1}\|W^j\|^2, \delta > 0, \tag{6}$$

where ϕ is the neural network with the corresponding set of weights W trained to learn a transformation that minimizes the volume of a data-enclosing hypersphere centered on a predetermined point c. The second term is a standard weight decay regularizer.

Local Outlier Factor (LOF)

LOF (Breunig et al, 2000) is an unsupervised outlier detection method that, similarly to kNN_{local}, compares the local density of an observation to that of its neighbors. The distances between observations are replaced by reachability distances, defined as:

$$\text{reach-dist}_k(\mathbf{x}_i \leftarrow \mathbf{x}_j) = \max\{d(\mathbf{x}_j, \text{NN}_k(\mathbf{x}_j)), d(\mathbf{x}_i, \mathbf{x}_j)\} \tag{7}$$

The local reachability density of an observation $\mathbf{x}_i$ is then defined as the inverse average reachability distance from the set of $\mathbf{x}_i$'s neighbors, $k\text{NN}(\mathbf{x}_i)$, that are within the k nearest neighbor distance around $\mathbf{x}_i$:

$$\text{lrd}_\text{k}(\mathbf{x}_i) = \frac{|\,k\text{NN}(\mathbf{x}_i)|}{\sum_{\mathbf{x}_j \in k\text{NN}(\mathbf{x}_i)} \text{reach-dist}_k(\mathbf{x}_i \leftarrow \mathbf{x}_j)} \tag{8}$$

Finally, the LOF score of an observation is computed by comparing the lrd of the observation with that of its neighbors:

$$\text{LOF}_k(\mathbf{x}_i) = \frac{\sum_{\mathbf{x}_j \in k\text{NN}(\mathbf{x}_i)} \frac{\text{lrd}_k(\mathbf{x}_j)}{\text{lrd}_k(\mathbf{x}_i)}}{|\,k\text{NN}(\mathbf{x}_i)|} \tag{9}$$

Local Correlation Integral (LOCI)

LOCI (Papadimitriou et al, 2003) is an unsupervised outlier detection method which analyzes the density of an observation at multiple neighborhood radii φr of a given maximum radius r, where $\varphi \in (0, 1]$. For each observation $\mathbf{x}_i$, a (local) r-neighborhood $\mathcal{N}(\mathbf{x}_i, r) = \{\mathbf{x} | d(\mathbf{x}_i, \mathbf{x}) \leq r\}$ and a (local) r-density $n(\mathbf{x}_i, r) = |\mathcal{N}(\mathbf{x}_i, r)|$ are defined.

The average φr-density inside an r-neighborhood around an observation $\mathbf{x}_i$ is then defined as:

$$\hat{n}(\mathbf{x}_i, r, \varphi) = \frac{\sum_{\mathbf{x}_j \in \mathcal{N}(\mathbf{x}_i, r)} n(\mathbf{x}_j, \varphi r)}{n(\mathbf{x}_i, r)}, \tag{10}$$

and the multi-granularity deviation factor (MDEF) is given by:

$$\text{MDEF}(\mathbf{x}_i, r, \varphi) = 1 - \frac{n(\mathbf{x}_i, \varphi r)}{\hat{n}(\mathbf{x}_i, r, \varphi)} \tag{11}$$

An observation $\mathbf{x}_i$ is classified using the following score:

$$\sigma\,\text{MDEF}(\mathbf{x}_i, r, \varphi) = \frac{\sigma_{\hat{n}}(\mathbf{x}_i, r, \varphi)}{\hat{n}(\mathbf{x}_i, r, \varphi)}, \tag{12}$$

which is the normalized standard deviation $\sigma_{\hat{n}}(\mathbf{x}_i, r, \varphi)$ of $n(\mathbf{x}_i, \varphi r)$ for $\mathbf{x}_i \in \mathcal{N}(\mathbf{x}_i, r)$. With these quantities, the LOCI score is computed as follows:

$$\text{LOCI}(\mathbf{x}_i, \varphi) = \max_{r \in \mathcal{R}} \left\{ \frac{\text{MDEF}(\mathbf{x}_i, r, \varphi)}{\sigma\,\text{MDEF}(\mathbf{x}_i, r, \varphi)} \right\} \tag{13}$$

k-Nearest Neighbor (kNN_{global})

The k-Nearest Neighbor approach, which we call here kNN_{global}, has been originally introduced as an unsupervised distance-based outlier detection method (Ramaswamy et al, 2000). Its score is the numerator of Equation (5):

$$\text{kNN}_{global}(\mathbf{x}_i, k) = d(\mathbf{x}_i, \text{NN}_\text{k}(\mathbf{x}_i)), \tag{14}$$

which makes the score global rather than local.

Angle-Based Outlier Detection (ABOD)

ABOD (Kriegel et al, 2008) is a global outlier detection algorithm which uses not only the distances between points but primarily the variance of the angles between points. ABOD computes the variance of the angles between point $\mathbf{x}_i$ and all other pairs of points in the dataset $\mathbf{X}$, weighted by the inverse of the distances to the respective points. This weighting factor is important since the angle to a pair of points varies naturally more for larger distances. The Angle-Based Outlier Factor (ABOF) is defined as follows:

$$\mathrm{ABOF}(\mathbf{x}_i) = \mathrm{VAR}_{\mathbf{x}_j, \mathbf{x}_k \in \mathbf{X}_{\mathrm{train}}} \left(\frac{\langle \mathbf{x}_i - \mathbf{x}_j, \mathbf{x}_i - \mathbf{x}_k \rangle}{\|\mathbf{x}_i - \mathbf{x}_j\|^2 \cdot \|\mathbf{x}_i - \mathbf{x}_k\|^2} \right) \tag{15}$$

Subspace Outlier Degree (SOD)

SOD (Kriegel et al, 2009) analyzes for each object how well it fits into the subspace that is spanned by a set of reference objects. In order to define the reference set, the authors suggest using Shared Nearest Neighbors (SNN) in the full space. Once this reference set has been defined, the *relevant subspace* is determined as the set of attributes in which the variance of the objects of the reference set to its center is small compared to the expected variance. The outlier scoring of SOD is computed as the Euclidean distance of an object to the center of the reference set in the relevant subspace, normalized by the number of relevant attributes.

Global-Local Outlier Scores from Hierarchies (GLOSH)

GLOSH (Campello et al, 2015) is an unsupervised outlier detection algorithm based on the hierarchical density estimates provided by the hierarchical clustering algorithm HDBSCAN*. After a density-based clustering hierarchy is computed for the whole dataset, the GLOSH score for each observation $\mathbf{x}_i$ can be computed based on the difference in density around $\mathbf{x}_i$ and the highest density inside the cluster closest to $\mathbf{x}_i$ (from a density-connectivity perspective) in the HDBSCAN* hierarchy, defined as follows:

$$\mathrm{GLOSH}(\mathbf{x}_i) = \frac{\lambda_{\max}(C_{\mathbf{x}_i}) - \lambda(\mathbf{x}_i)}{\lambda_{\max}(C_{\mathbf{x}_i})}, \tag{16}$$

where $\lambda(\mathbf{x}_i)$ is the density of $\mathbf{x}_i$ and $\lambda_{\max}(C_{\mathbf{x}_i})$ is the highest density of an observation inside the closest cluster $C_{\mathbf{x}_i}$, where densities are estimated by a k-nearest neighbor density estimator. The closest cluster $C_{\mathbf{x}_i}$ is the one that $\mathbf{x}_i$ belongs to at the density level of $\mathbf{x}_i$.

To apply GLOSH in a one-class classification scenario, we can construct initially the HDBSCAN* hierarchy using the training data, and then use this hierarchy as a fixed "model" to compute outlier scores for unseen data. In order to classify a new instance $\mathbf{x}_i$, we must determine which is the closest cluster $C_{\mathbf{x}_i}$ in the fixed hierarchy. This can be achieved by first adding a given instance $\mathbf{x}_i$ to the Minimum Spanning Tree (MST) which underlies the

HDBSCAN* hierarchy; $\mathbf{x}_i$ is connected to the training instance $\mathbf{x}_j$ with the smallest "distance" in the density space in which the MST is constructed. We can find the closest cluster $C_{\mathbf{x}_i}$ for an instance $\mathbf{x}_i$ by removing the edges of the MST in decreasing order of weight. The closest cluster $C_{\mathbf{x}_i}$ is the one that $\mathbf{x}_i$ belongs before being detached from the MST.

There are two situations that may occur when connecting the new instance $\mathbf{x}_i$ to the training instance $\mathbf{x}_j$. The first situation is that the edge connecting $\mathbf{x}_i$ and $\mathbf{x}_j$ is removed before $\mathbf{x}_j$ is separated from the MST. In this case, the closest cluster to $\mathbf{x}_i$ will be the cluster that $\mathbf{x}_j$ belongs to when $\mathbf{x}_i$ is detached. We show an example of this situation in Figure 1.

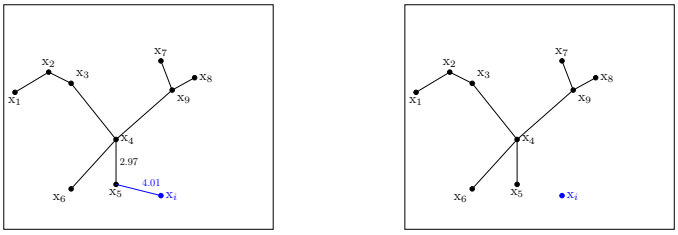


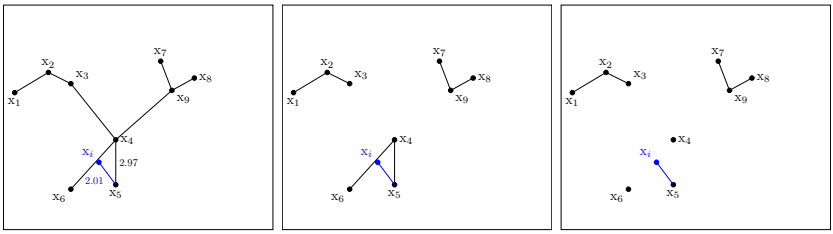
Fig. 1: MST which underlies the HDBSCAN* hierarchy in the density space with minimum cluster size $(m_{clSize}) = 3$. The new instance $\mathbf{x}_i$ is connected to the training instance with the smallest distance $\mathbf{x}_5$ (Figure 1(a)). The edge connecting $\mathbf{x}_i$ and $\mathbf{x}_5$ is removed before $\mathbf{x}_5$ is separated from the MST and so the closest cluster to $\mathbf{x}_i$ is the one $\mathbf{x}_5$ is assigned at this density level. In this case, it is the cluster where all instances remain clustered (Figure 1(b))

The second situation is that $\mathbf{x}_j$ is separated from the MST before the edge between $\mathbf{x}_i$ and $\mathbf{x}_j$ is removed. In this case, the closest cluster for $\mathbf{x}_i$ is the same as the closest cluster for $\mathbf{x}_j$. For this situation, we show an example in Figure 2.

Once we have identified the closest cluster for $\mathbf{x}_i$, we can compute its GLOSH score using Equation (16).

Isolation Forest (iForest)

iForest (Liu et al, 2008, 2012) is an unsupervised outlier detection algorithm based on the concept of isolation which can be seen as a particular kind of density estimate. The concept of isolation in this context means "separating an instance from the rest of the instances", which is achieved by building an ensemble of binary trees called isolation trees (iTree). An iTree recursively divides a dataset by randomly selecting an attribute and a split value until either the resulting partitions/nodes have only one instance or all instances in a partition/node have the same value. Anomalies are more likely to be isolated



(a) (b) (c)

Fig. 2: MST which underlies the HDBSCAN* hierarchy in the density space with minimum cluster size (m_{clSize}) = 3. The new instance $\mathbf{x}_i$ is connected to the training instance with the smallest distance $\mathbf{x}_5$ (Figure 2(a)). The edge connecting $\mathbf{x}_5$ is separated from the MST before the edge between $\mathbf{x}_i$ and $\mathbf{x}_5$ (Figure 2(c)) and so the closest cluster to $\mathbf{x}_i$ is the closest cluster for $\mathbf{x}_5$. In this case, it is the cluster with the instances $\mathbf{x}_4$, $\mathbf{x}_5$ and $\mathbf{x}_6$ (Figure 2(b))

closer to the root of an iTree, whereas normal instances are more likely to be isolated at deeper levels of an iTree. Therefore, anomalies are those instances that have short average path lengths in the iTrees.

1.2 Time Complexity

The computational complexity of the one-class classification algorithms can be analyzed for the two different sub-tasks: the training time $\mathcal{O}(f_{tr}(N))$ and the testing (or consultation) time $\mathcal{O}(f_{te}(M))$, as a function of the training set size (N) and the test set size (M), respectively. In general, the algorithms can be divided into two groups, eager learning and lazy learning (Webb, 2011). In regular classification problems, the majority of computation of eager learning algorithms occurs at training time, while lazy learning algorithms spend more time when testing. In one-class classification, however, if one wants to determine the threshold to separate inliers from outliers when using lazy learning algorithms, the model should also be applied to the training data (for example, as discussed in section 3.3), which can result in higher complexity in the training time than in the testing when $N > M$. On the other hand, if one is not interested in labeling, but only in the ranking, the complexity for training time is usually null[1]. Another disadvantage of lazy learning algorithms is the fact that the training data has to be stored for consultation during the testing phase, which also imposes a storage restriction.

In Table 1, we provide the complexity of the methods discussed here. For networks such as Auto-encoder and Deep SVDD, the training time complexity depends on the training algorithm used, but it is usually linear in the training set size (N) per each training iteration. Some algorithms, such as the Stochastic Gradient Descent (SGD) methods (Goodfellow et al, 2016), can perform the

[1]For some methods, we still have to apply the model in the training data anyway, for example, for LOF adapted to OCC, we still have to pre-compute the required quantities.

Table 1: Time Complexity of Methods.
N: training set size, M: test set size, d: dataset dimensionality, t: method parameter. *per iteration

	Training time $\mathcal{O}(f_{tr}(N))$	Testing time $\mathcal{O}(f_{te}(M))$
Auto Enc.	$\mathcal{O}(N(t+d+2dt))*$	$\mathcal{O}(dM)$
DSVDD	$\mathcal{O}(t)*$	$\mathcal{O}(dM)$
GMM	$\mathcal{O}(d^3Nt + d^2Nt)$	$\mathcal{O}(d^3Mt)$
iForest	$\mathcal{O}(t_1^2 t_2)$	$\mathcal{O}(Mt_1)$
LP	$\mathcal{O}(dN^3)$	$\mathcal{O}(dM)$
SVDD	$\mathcal{O}(dN^3)$	$\mathcal{O}(dM)$
ABOD	$\mathcal{O}(dN^3)$	$\mathcal{O}(dMN^2)$
GLOSH	$\mathcal{O}(dN^2)$	$\mathcal{O}(dMN)$
kNN_{global}	$\mathcal{O}(dN^2)$	$\mathcal{O}(dMN)$
kNN_{local}	$\mathcal{O}(dN^2)$	$\mathcal{O}(dMN)$
LOCI	$\mathcal{O}(dN^3)$	$\mathcal{O}(dMN^2)$
LOF	$\mathcal{O}(dN^2)$	$\mathcal{O}(dMN)$
PW	$\mathcal{O}(dN^2)$	$\mathcal{O}(dMN)$
SOD	$\mathcal{O}(dN^2)$	$\mathcal{O}(dMN)$

iteration in time complexity less than N, by using only a sample of the training data to compute an unbiased estimate of the gradient. Most deep learning methods, such as Deep SVDD, use SGD to alleviate the computational burden caused by deep number of layers/neurons. The bottleneck in the training of a neural network is usually on the number of weights to train. While for Deep SVDD, there is no predetermined number of weights, for Auto-encoder, the number of weights is equal to $(t + d + 2dt)$, where d is the dimensionality of the dataset, and t is the user-defined number of neurons. Therefore, the use of Auto-encoder for high-dimensional datasets can be expensive. The time complexity in the number of weights also depends on the training algorithm to be used, for example, in the case of Conjugate Gradient, it is linear, while for Levenberg-Marquardt, it is cubic (LeCun et al, 1998). The convergence for Levenberg-Marquardt, however, is usually much faster, but the high time complexity makes it suitable only for small network sizes. Note that, although the time complexity per iteration for Deep SVDD ($\mathcal{O}(\#weights)$) is smaller when compared to autoencoder ($\mathcal{O}(N \times \#weights)$) due to the SGD, the number of iterations required for the convergence of the autoencoder is smaller (Goodfellow et al, 2016). Also, usually the number of weights of the Deep SVDD $\gg$ number of weights of the autoencoder.

The training for GMMs consists of finding the distribution parameters of the Gaussians (covariance matrices and means). It is usually achieved using an Expectation-Maximization (EM) algorithm. For each iteration of the EM algorithm, the E-step takes $\mathcal{O}(d^3Nt)$ and the M-step takes $\mathcal{O}(d^2Nt)$, where d is the dimensionality of the dataset, and t is the user-defined number of clusters. Noticing that, due to the inversion of the covariance matrix ($\mathcal{O}(d^3)$), this

algorithm is expensive for high-dimensional datasets. For testing, the computation of the likelihood of each testing point belonging to the clusters takes $\mathcal{O}(d^3Mt)$.

The training time complexity for iForest uses only a user-defined sampling t_1 of the dataset to build the iTrees. The time complexity to build each of the t_2 iTrees defined by the user is quadratic in t_1, which gives an overall complexity for training equal to $\mathcal{O}(t_1^2 t_2)$. The structure of the iTrees is equivalent to that of Binary Search Trees (BST), which makes the expected time complexity for testing a new instance $\mathcal{O}(\log t_1)$, but the worst-case is $\mathcal{O}(t_1)$.

Both LP and SVDD have to solve an optimization problem during the training phase. While LP has to solve a linear programming (LP) problem, SVDD has to solve a quadratic programming (QP) problem. Both problems can be solved in polynomial time ($\mathcal{O}(N^3)$) using, for example, Interior Point Methods (Woodsend, 2009; Campbell and Bennett, 2000). When using a linear kernel, however, some algorithms in the literature can solve the problem efficiently in linear time (Joachims, 2006; Erfani et al, 2015).

For ABOD, the time complexity relies on the computation of the angles between each point and all other pairs of points in the dataset, which makes the overall time complexity for training $\mathcal{O}(N^3)$ and testing $\mathcal{O}(MN^2)$. Similarly, the time complexity for LOCI relies on the computation of the MDEF for each observation. When considering all possible radii r, it also takes $\mathcal{O}(N^3)$ for training and $\mathcal{O}(MN^2)$ for testing.

For kNN_{global}, kNN_{local}, LOF, and SOD the most expensive operation is the computation of the k nearest neighbors for all N observations. Without using any spatial index structures, this computation takes $\mathcal{O}(N^2)$. However, with the use of spatial index structures, such as k-d trees and R-trees (Friedman et al, 1977; Roussopoulos et al, 1995), the k nearest neighbors for all N observations can in practice often be determined much faster. For testing new instances, we have to compute the k nearest neighbors of these instances to all instances of the training set ($\mathcal{O}(MN)$). Similarly to k-nearest neighbor-based classifiers, the PW needs to compute the distance to all the observations in the training set in order to estimate the densities. Therefore, the PW time complexity is the same as that of the k-nearest neighbor-based methods.

The training phase of GLOSH consists of building the HDBSCAN* hierarchy. The two most expensive operations are the computation of the core distance, which involves computing the k nearest neighbors for all N observations ($\mathcal{O}(N^2)$), and the construction of the MST ($\mathcal{O}(N^2)$). Therefore, the overall time complexity for training is $\mathcal{O}(N^2)$. For testing, GLOSH has to compute the core distance for each testing point ($\mathcal{O}(MN)$), and find their respective closest objects in the MST ($\mathcal{O}(MN)$). In total, the testing time complexity is $\mathcal{O}(MN)$.

References

Bazaraa MS, Jarvis JJ, Sherali HD (2009) Linear Programming and Network Flows, 4th edn. Wiley-Interscience

Bishop CM (2007) Pattern Recognition and Machine Learning, 5th edn. Springer

Breunig MM, Kriegel H, Ng RT, et al (2000) LOF: identifying density-based local outliers. In: Proceedings of the 2000 SIGMOD International Conference on Management of Data. ACM, pp 93–104, https://doi.org/10.1145/342009.335388

Campbell C, Bennett KP (2000) A linear programming approach to novelty detection. In: Proceedings of the 13th NIPS International Conference on Neural Information Processing Systems. MIT Press, pp 395–401

Campello RJGB, Moulavi D, Zimek A, et al (2015) Hierarchical density estimates for data clustering, visualization, and outlier detection. ACM Trans Knowl Discov Data 10(1):5:1–5:51. https://doi.org/10.1145/2733381

Dempster AP, Laird NM, Rubin DB (1977) Maximum likelihood from incomplete data via the em algorithm. Journal of the Royal Statistical Society 39(1):1–38

Duin RPW (1976) On the choice of smoothing parameters for parzen estimators of probability density functions. IEEE Trans Computers 25(11):1175–1179. https://doi.org/10.1109/TC.1976.1674577

Erfani SM, Baktashmotlagh M, Rajasegarar S, et al (2015) R1SVM: A randomised nonlinear approach to large-scale anomaly detection. In: Proceedings of the 29th AAAI Conference on Artificial Intelligence. AAAI Press, pp 432–438

Freeman JA, Skapura DM (1991) Neural Networks - Algorithms, Applications, and Programming Techniques. Addison-Wesley

Friedman JH, Bentley JL, Finkel RA (1977) An algorithm for finding best matches in logarithmic expected time. ACM Trans Math Softw 3(3):209–226. https://doi.org/10.1145/355744.355745

Goodfellow IJ, Bengio Y, Courville AC (2016) Deep Learning. MIT Press

Japkowicz N, Myers C, Gluck MA (1995) A novelty detection approach to classification. In: Proceedings of the 4th IJCAI International Joint Conference on Artificial Intelligence. Morgan Kaufmann, pp 518–523

Joachims T (2006) Training linear SVMs in linear time. In: Proceedings of the 12th SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp 217–226, https://doi.org/10.1145/1150402.1150429

Kriegel H, Schubert M, Zimek A (2008) Angle-based outlier detection in high-dimensional data. In: Proceedings of the 14th SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp 444–452, https://doi.org/10.1145/1401890.1401946

Kriegel H, Kröger P, Schubert E, et al (2009) Outlier detection in axis-parallel subspaces of high dimensional data. In: Proceedings of the 13th PAKDD Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, pp 831–838, https://doi.org/10.1007/978-3-642-01307-2_86

LeCun Y, Bottou L, Orr GB, et al (1998) Efficient BackProp. In: Montavon G, Orr GB, Müller K (eds) Neural Networks: Tricks of the Trade. Springer, p 9–50, https://doi.org/10.1007/3-540-49430-8_2

Liu FT, Ting KM, Zhou Z (2008) Isolation forest. In: Proceedings of the 8th ICDM International Conference on Data Mining. IEEE Computer Society, pp 413–422, https://doi.org/10.1109/ICDM.2008.17

Liu FT, Ting KM, Zhou Z (2012) Isolation-based anomaly detection. ACM Trans Knowl Discov Data 6(1):3:1–3:39. https://doi.org/10.1145/2133360.2133363

Papadimitriou S, Kitagawa H, Gibbons PB, et al (2003) LOCI: fast outlier detection using the local correlation integral. In: Proceedings of the 19th ICDE International Conference on Data Engineering. IEEE Computer Society, pp 315–326, https://doi.org/10.1109/ICDE.2003.1260802

Parzen E (1962) On estimation of a probability density function and mode. The annals of mathematical statistics 33(3):1065–1076

Pekalska E, Tax DMJ, Duin RPW (2002) One-class LP classifiers for dissimilarity representations. In: Proceedings of the 15th NIPS International Conference on Neural Information Processing Systems, Advances in Neural Information Processing Systems. MIT Press, pp 761–768

Ramaswamy S, Rastogi R, Shim K (2000) Efficient algorithms for mining outliers from large data sets. In: Proceedings of the 2000 SIGMOD International Conference on Management of Data. ACM, pp 427–438, https://doi.org/10.1145/342009.335437

de Ridder D, Tax DMJ, Duin RPW (1998) An experimental comparison of one-class classification methods. In: Proceedings of the 4th ASCI Advanced School for Computing and Imaging, pp 213–218

Roussopoulos N, Kelley S, Vincent F (1995) Nearest neighbor queries. In: Proceedings of the 1995 SIGMOD International Conference on Management of Data. ACM Press, pp 71–79, https://doi.org/10.1145/223784.223794

Ruff L, Görnitz N, Deecke L, et al (2018) Deep one-class classification. In: Proceedings of the 35th ICML International Conference on Machine Learning. PMLR, pp 4390–4399

Schölkopf B, Platt JC, Shawe-Taylor J, et al (2001) Estimating the support of a high-dimensional distribution. Neural Comput 13(7):1443–1471. https://doi.org/10.1162/089976601750264965

Strang G (2016) Introduction to Linear Algebra, 5th edn. Wellesley-Cambridge Press

Tax DMJ (2001) One-class classification. PhD thesis, Delft University of Technology

Tax DMJ, Duin RPW (2004) Support vector data description. Mach Learn 54(1):45–66. https://doi.org/10.1023/B:MACH.0000008084.60811.49

Vapni VN (1995) The Nature of Statistical Learning Theory. Springer, https://doi.org/10.1007/978-1-4757-2440-0

Webb GI (2011) Lazy learning. In: Sammut C, Webb GI (eds) Encyclopedia of Machine Learning. Springer, p 571–572, https://doi.org/10.1007/978-0-387-30164-8_443

Woodsend K (2009) Using interior point methods for large-scale support vector machine training. PhD thesis, University of Edinburgh