# Optimised one-class classification performance

**Oliver Urs Lenz · Daniel Peralta · Chris Cornelis**

**Abstract** We provide a thorough treatment of hyperparameter optimisation for three data descriptors with a good track-record in the literature: Support Vector Machine (SVM), Nearest Neighbour Distance (NND) and Average Localised Proximity (ALP). The hyperparameters of SVM have to be optimised through cross-validation, while NND and ALP allow the reuse of a single nearest-neighbour query and an efficient form of leave-one-out validation. We experimentally evaluate the effect of hyperparameter optimisation with 246 classification problems drawn from 50 datasets. From a selection of optimisation algorithms, the recent Malherbe-Powell proposal optimises the hyperparameters of all three data descriptors most efficiently. We calculate the increase in test AUROC and the amount of overfitting as a function of the number of hyperparameter evaluations. After 50 evaluations, ALP and SVM both significantly outperform NND. The performance of ALP and SVM is comparable, but ALP can be optimised more efficiently, while a choice between ALP and SVM based on validation AUROC gives the best overall result. This distils the many variables of one-class classification with hyperparameter optimisation down to a clear choice with a known trade-off, allowing practitioners to make informed decisions.

**Keywords** Data descriptors · Hyperparameter optimisation · Novelty detection · One-class classification · Semi-supervised outlier detection

## 1 Introduction

The goal of one-class classification (Tax (2001), also known as novelty detection or semi-supervised outlier detection), is to form, on the basis of a representative

O.U. Lenz · D. Peralta · C. Cornelis
Department of Applied Mathematics, Computer Science and Statistics, Ghent University, Belgium
E-mail: {oliver.lenz,chris.cornelis}@ugent.be

D. Peralta
Data Mining and Modelling for Biomedicine group, VIB Center for Inflammation Research, Ghent University, Belgium
E-mail: daniel.peralta@irc.vib-ugent.be

sample, a model of a target class of data that can later be used to predict whether unseen instances belong to the target class. Such one-class classification algorithms are called data descriptors. The difference with binary classification lies in the fact that a data descriptor only uses training data belonging to the target class.

A number of data descriptors that have been proposed in the literature require setting one or more hyperparameters. In recent research (Lenz et al. 2021), we have identified optimal default hyperparameter values for a number of data descriptors, and evaluated their performance. These default values allow the data descriptors to be used on one-class classification problems without negative training samples. However, for problems where a (limited) sample of negative data is available, this data can be used to optimise, or 'tune', the hyperparameter values. One such setting is the use of data descriptors as building blocks in a multi-class classification ensemble (Ban and Abe 2006).

The performance of data descriptors with hyperparameter optimisation has previously been evaluated by Janssens et al. (2009) and Swersky et al. (2016). In the present article, we go further, and try to answer the question: "What is the best way to optimise the hyperparameter values of a data descriptor?". Whereas both previous works use a naive grid search to optimise hyperparameter values, we evaluate a representative selection of optimisation algorithms, and identify the most effective strategy. We also explain how data descriptors based on nearest neighbour searches can be optimised using efficient leave-one-out validation instead of ordinary cross-validation.

A second question that we address is: "How long should hyperparameter optimisation run for?" Besides being inefficient, grid search also requires fixing the total number of evaluations before optimisation begins. In contrast, the optimisation algorithms that we consider sequentially select points in the hyperparameter space to evaluate. This allows us to present our results in terms of the number of evaluations, giving practitioners more insight into the effect of this choice.

In the experiments performed by Swersky et al. (2016), the Support Vector Machine (SVM) and Nearest Neighbour Distance (NND) were the two highest-ranked data descriptors with hyperparameters. SVM also was the joint-best data descriptor in Janssens et al. (2009), while NND was not tested there. In our own experiments (Lenz et al. 2021) with default hyperparameter values, SVM and NND were the second and third best-performing data descriptors. The best-performing data descriptor was our own proposal, Average Localised Proximity (ALP), which remains untested in the context of hyperparameter optimisation. On this basis, we have chosen to compare SVM, NND and ALP in the present article.

Both previous studies evaluated performance with a Nemenyi test on mean ranks (Demšar 2006). This forced them to amalgamate results from one-class classification problems derived from the same dataset, of which there were 24 in Janssens et al. (2009) and 30 in Swersky et al. (2016). The latter study could not establish a difference in performance between SVM and NND that was statistically significant. To address this, we draw from a larger number of datasets (50), and we compare pairs of data descriptors using a clustered Wilcoxon signed rank test (Rosner et al. 2006) that allows us to use the full results from all 246 one-class classification problems. This choice is additionally motivated by the criticism that the $p$-value generated by the Nemenyi test for a pair of machine algorithms is too strongly dependent on the inclusion of other algorithms in the comparison (Benavoli et al. 2016).

By using a suitable optimisation procedure, by including ALP in our comparison, by performing our experiments with more datasets, and by employing a more precise statistical test, we can also provide a stronger answer to the question "What is the best data descriptor for one-class classification with hyperparameter optimisation?"

We proceed by discussing the various data descriptors (Section 2) and optimisation algorithms (Section 3) in this article, explaining how our experiments are structured (Section 4), discussing the results of these experiments (Section 5) and presenting our conclusions (Section 6).

## 2 Data descriptors

In this section, we list the data descriptors under consideration in the present paper. For a more detailed discussion, the reader is referred to Lenz et al. (2021). In this section, we focus on how the hyperparameters of these data descriptors can be optimised. The goal in each case is to maximise the area under the receiver operator characteristic (AUROC).

### 2.1 Support Vector Machine

The Support Vector Machine (SVM) data descriptor was proposed independently by Tax and Duin (1999a,b) and Schölkopf et al. (1999, 2001). Both variants fit a surface to isolate the training data in the feature space, and allow application of the kernel trick to transform the feature space. The recommended kernel is the Gaussian kernel, parametrised with a value $c \in (0, \infty)$. With the Gaussian kernel, the two variants become equivalent. We use the Schölkopf variant, which solves an optimisation problem to fit a hyperplane between most of the target data and the origin, at a maximal distance to the origin. A hyperparameter $\nu \in (0, 1]$ controls the relative weight placed respectively on maximising distance to the origin, and not leaving training instances on the same side as the origin.

Thus, there are two hyperparameters that have to be chosen, and can be optimised with training data: $\nu$ and $c$. Because many of the optimisation methods that we consider require compact hyperparameter domains, we reparametrise $c$ as $\frac{c'}{1-c'}$, and optimise $c'$ in $[10^{-6}, 1-10^{-6}]$, and restrict the domain of $\nu$ to $[10^{-6}, 1]$.

In order to optimise $\nu$ and $c'$, we apply five-fold cross-validation to obtain five splits of the available training data into a smaller training set and a validation set. For each training set, we select the target class instances to obtain a target set. We evaluate a pair of hyperparameter values by constructing a model on the target set, querying with the respective validation sets, and calculating the mean of the resulting AUROC scores. This means that optimising the hyperparameters of SVM requires constructing 5 models for each pair of values to be evaluated.

### 2.2 Nearest Neighbour Distance

Nearest Neighbour Distance (NND) is a much simpler data descriptor than SVM, and goes back to at least Knorr and Ng (1997); it classifies instances in accordance

with the distance to their $k$th nearest neighbour in the target data. In principle, the distance measure can be chosen freely, but in order to allow the efficient form of optimisation discussed below, we fix this choice to the Manhattan metric, which generally gives better results than the Euclidean metric (Lenz et al. 2021). This leaves $k$ as the only hyperparameter to be optimised. Since it encodes a magnitude, we optimise $k$ logarithmically. To avoid having to work with extremely large arrays, and knowing that its optimal default value is simply 1 (Lenz et al. 2021), we limit $k$ to $100 \log n$.

Because NND is so simple, $k$ can be optimised more efficiently than the hyperparameters of SVM. Firstly, it is not necessary to completely recalculate a new NND model for each value of $k$. if $k_{\max}$ is the maximum value for $k$ that we want to consider, we only require a single sorted $k_{\max}$-nearest neighbour query, since this contains all $i$th nearest neighbours for all $i \leq k_{\max}$. Secondly, instead of 5-fold cross-validation, we can use an efficient form of leave-one-out validation, by using all training data as a single training and validation set. For non-target class instances, this is not a problem, but target class will be queried against themselves. We can correct this by substituting a $k_{\max} + 1$-nearest neighbour query for target class instances, and removing the first nearest neighbour distances, which are 0, since these correspond to the instances themselves. We thus obtain a single set of scores, from which we can calculate the AUROC.

### 2.3 Average Localised Proximity

Average Localised Proximity (ALP) was first proposed in Lenz et al. (2021). For a choice of integers $k$ and $l$, the average localised proximity of an instance $y$ (3) is defined as the ordered weighted average of $k$ localised proximities (2), each of which is calculated from a nearest neighbour distance of $y$ and the weighted average of $l$ local nearest neighbour distances relative to $y$ (1).

$$D_i(y) = \sum_{j \leq l} w_j^l \cdot d_i(\mathrm{NN}_j(y)). \tag{1}$$

$$\mathrm{lp}_i(y) = \frac{D_i(y)}{D_i(y) + d_i(y)}. \tag{2}$$

$$\mathrm{alp}(y) = \underset{i \leq k}{w^k \downarrow} \mathrm{lp}_i(y). \tag{3}$$

The distance measure used is the Manhattan metric, and the weights are linearly decreasing: $\frac{p}{p(p+1)/2}, \frac{p-1}{p(p+1)/2}, \ldots, \frac{1}{p(p+1)/2}$, for $p = k, l$. This leaves $k$ and $l$ to be optimised, which respectively determine the scale at which nearest neighbour distances are considered and the amount of localisation. As with NND, we optimise $k$ and $l$ logarithmically.

Since ALP is based on nearest neighbour queries, it offers some of the same advantages of NND. Most importantly, we only need a single $\max(k_{\max}, l_{\max})$ nearest neighbour query if we want to evaluate values of $k$ and $l$ up to $k_{\max}$ and $l_{\max}$.

We can also apply efficient leave-one-out validation, although this is a bit more complicated than with NND. The average localised proximity of a target

class instance $y$ is calculated on the basis of nearest neighbour distances of $y$, and of local nearest neighbour distances relative to $y$, and we have to correct both. The former can be corrected as with NND, by taking the $k+1$ nearest neighbour distances of $y$ and removing the first (the distance of $y$ to itself). In order to correct the local distances of a neighbour $x$ of $y$, we also consider its $k+1$ nearest neighbours, and remove $y$ if it is among these, and otherwise remove the last nearest neighbour distance.

Increasing $k$ and $l$ has two effects: it draws in more distant neighbours of $y$, and it decreases the slope of the weight vectors, making the contribution of successive neighbours more equal. The asymptotic limit of this process is a weight vector with equal weights everywhere. A more practical limit is that $k$ and $l$ cannot grow beyond the number of target instances $n$. However, we can simulate higher values for $k$ and $l$ by truncating the weight vector and multiplying by a constant to ensure that its sum still equals 1. This also allows us to address the same computational issue with large datasets mentioned for NND, namely that evaluating a pair of values for $k$ and $l$ on the whole training set involves $(k+1) \cdot l \cdot n$ distance values. If $(k+1) \cdot l$ is large, processing a single array with all distance values requires a large amount of memory. For these reasons, we let $k$ and $l$ range up to $5n$, but truncate distance values and weight vectors after $\min(n, 20 \log n)$. This is informed by the knowledge that the optimal default values for $k$ and $l$ are $5.5 \log n$ and $6 \log n$ respectively (Lenz et al. 2021).

## 3 Optimisation algorithms

Optimisation problems are typically formulated in terms of a problem function $f : P \longrightarrow \mathbb{R}$, where the *problem space* $P$ is a subspace of $\mathbb{R}^m$ for some $m$ that is often required to be compact. Depending on the context, the goal of optimisation is to find points in $P$ that minimise or maximise $f$. In the present article, we wish to maximise the validation AUROC of our data descriptors, and the problem space is determined by the hyperparameters that we optimise.

There is an important distinction between algorithms that aim to find a local optimum of $f$ in $P$, and those that aim to identify the global optimum of $f$ in $P$. An essential characteristic of global optimisation algorithms is that they have to balance the exploration of areas of the problem space with large uncertainty and the exploitation of areas where function performance is known to be good.

An intrinsic disadvantage of local optimisation algorithms is that they require a choice of starting point, and may get stuck in a local optimum if this starting point is chosen poorly. However, the optimisation problems of finding the best hyperparameter values for our data descriptors may be close to unimodal, in which case this risk could be relatively small. For this reason, we include two classical local optimisation algorithms that are easy to implement.

### 3.1 Random search

Purely random search is a surprisingly potent global optimisation strategy. By continuing to evaluate arbitrary points in the problem space, we can expect to eventually get arbitrarily close to the global optimum. In particular, random search

is a more efficient algorithm than grid search (Bergstra and Bengio 2012). Because this strategy uses no information from previous evaluations, it serves as a good baseline.

### 3.2 Hooke-Jeeves

The local search algorithm proposed by Hooke and Jeeves (1961) passes through the problem space in steps. Each step follows a pattern, which is the vector sum of a number of substeps along each coordinate axis. The pattern is adjusted with each step, by optionally adding or subtracting a substep along each coordinate axis, depending on which option results in the greatest decrease in the objective function value. If the pattern cannot be adjusted to produce a step with any improvement, a new pattern is created from scratch. If no such pattern can be found either, the substep size is decreased.

We use the implementation provided by pymoo (Blank and Deb 2020) and its default values. As starting values we use the optimal default values identified in Lenz et al. (2021).

### 3.3 Nelder-Mead

The local search algorithm proposed by (Nelder and Mead 1965) is based on an earlier proposal by Spendley et al. (1962). It iterates on $m + 1$ points that can be viewed as the vertices of an $m$-dimensional simplex. The algorithm lets this simplex 'walk' through the problem space by replacing its worst vertex in each iteration. Each step is directed towards the mid-point of the remaining vertices. The new vertex is either placed between the worst vertex and this mid-point (shrinking the simplex), or beyond the midpoint (reflecting and optionally extending it). If none of these options improve upon the worst vertex, the entire simplex is shrunk, with only the best vertex remaining in place.

The theoretical performance of Nelder-Mead optimisation had long been unclear, until Torczon (1989) demonstrated with a concrete example that Nelder-Mead can converge on points that are not local optima, even with problems that are twice differentiable. Nevertheless, Nelder-Mead optimisation has been very popular due to its relative simplicity, and because it seems to converge very quickly in many simple practical applications (Wright 1995).

We use the implementation provided by SciPy (Virtanen et al. 2020), with starting simplices centred around the optimal default values identified in Lenz et al. (2021).

### 3.4 Kushner-Sittler

A global optimisation method was first proposed by Kushner (1962, 1964), based in part on unpublished work by Robert A Sittler (see Betrò (1991), Jones (2001) and Brochu et al. (2009) for overviews of later developments). This has been referred to as simply *global optimisation*, or *Bayesian optimisation*, because its central idea is to iteratively use the Bayesian information criterion to select the next point in

the problem space to evaluate. We assume that the unknown problem function is drawn from a random distribution of functions, which is traditionally modelled as a Gaussian process. In each step, we can calculate the conditional probability $p(y|x)$ that the problem function will obtain certain values at a given point $x$ in the problem space, in light of the function values that have already been calculated. These conditional probabilities are then reduced to a single score for each $x$ with an activation function, and the point with the maximal such score is selected as the next point to evaluate. The activation function most often used today, already hinted at in Kushner (1962), is *expected improvement*, the expected value of $\max(y - y^*, 0)$ under the model for some $y^* \in \mathbb{R}$, typically the largest evaluated function value so far.

Kushner-Sittler optimisation transforms the original optimisation problem into a series of new optimisation problems for each iteration. These new optimisations incur a certain cost themselves, but this cost is only dependent on the dimensionality of the original problem, so for problems that are costly to evaluate, the trade-off is worthwhile. Note also that as with the original problem, these optima can generally only be approximated, but it is (often tacitly) assumed that this is not problematic.

We use the implementation provided by Emukit (Paleyes et al. 2019), with the first point chosen randomly.

### 3.5 Bergstra-Bardenet

A more recent variant of Kushner-Sittler optimisation, motivated in particular by high-dimensional and conditional problem spaces, is the Tree-structured Parzen Estimator (TPE) proposal by Bergstra et al. (2011). It lets the target value $y^*$ correspond to a quantile of the evaluated function values. This induces a split between small and large values, and the corresponding distributions $p(y < y^*)$ and $p(y > y^*)$, which can be modelled with two Parzen Estimators $l(x)$ and $g(x)$. By reformulating $p(y|x)$ in terms of $p(x|y)$, $p(y)$ and $p(x)$, the authors then show that the expected improvement of the original model is maximal when $\frac{g(x)}{l(x)}$ is maximal.

We use the Adaptive TPE implementation of hyperopt (Bergstra et al. 2011).

### 3.6 Malherbe-Powell

Global optimisation algorithms often proceed from the assumption that the problem function satisfies certain smoothness conditions. In particular, for any $k > 0$, we can define the class of $k$-Lipschitz functions as those functions $f$ that satisfy (4):

$$\forall x_1, x_2 \in A : |f(x_1) - f(x_2)| \leq k \cdot |x_1 - x_2| \tag{4}$$

Malherbe and Vayatis (2017) propose that we can use this assumption to restrict the search to certain parts of the problem space. They propose the LIPO algorithm, in which random candidates are drawn from the problem space, but only those candidates are evaluated that potentially improve upon the current optimum, in view of the candidates evaluated so far and (4).

In general, $k$ may be difficult to estimate, and we simply want to assume that some such $k$ exists for a problem function. For this purpose, Malherbe & Vayatis (2017) also propose the AdaLIPO algorithm. This alternates LIPO with purely random search, and increases $k$ whenever (4) is no longer satisfied.

AdaLIPO was further modified into MaxLIPO and implemented into the dlib library by King (2009, 2017). MaxLIPO presents three improvements. Firstly, it incorporates a noise term that prevents $k$ from approaching infinity if the problem function is not in fact $k$-Lipschitz because it has small discontinuities. Secondly, it employs separate values of $k$ for each dimension. And thirdly, instead of selecting new candidates at random, it identifies the candidate with the largest potential improvement in light of (4).

A more fundamental problem of AdaLIPO that carries over to MaxLIPO is that while it is seemingly able to quickly locate the neighbourhood of the global optimum, it then takes much longer to approach the optimum itself. This can be understood, since by considering the maximal potential improvement, rather than some form of expected improvement, these algorithms place a greater emphasis on exploration than exploitation. To address this, King (2017) lets MaxLIPO alternate with the trust region approach of the local optimiser BOBYQA (Powell 2009), a bounded version of the earlier NEWUOA proposal (Powell 2004).

## 4 Experimental setup

In order to enable comparison with using default hyperparameter values, we closely follow the experimental setup of Lenz et al. (2021). We use the same collection of 246 one-class classification problems, drawn from 50 datasets from the UCI machine learning repository (Dua and Graff 2019), rescaled through division by the interquartile range (Rousseeuw and Croux 1993) of the target data. We have implemented NND and ALP in our own Python wrapper fuzzy-rough-learn (Lenz et al. 2020), while we use scikit-learn (Pedregosa et al. 2011) for SVM and nearest neighbour queries.

For each one-class classification problem, we apply 5-fold cross-validation. We measure the performance of a data descriptor with specific hyperparameter values in terms of the area under the receiver operator curve (AUROC). For each division, we measure validation AUROC using nested cross- or leave-one-out validation as explained in Section 2, as well as test AUROC by retraining the data descriptor on all of the training data and evaluating its performance on the test data.

We maximise validation AUROC by applying the optimisation algorithms from Section 3. Each optimiser is allowed a maximum budget of 50 evaluations of hyperparameter values. Although the NND and ALP hyperparameters $k$ and $l$ are discrete, we optimise them as if they were continuous. As a result, there may be some duplication between optimisation steps as different points in the problem space are discretised back to the same value(s). With local optimisers that have reached a local optimum, as well as with small datasets in general, the optimisation search may never reach 50 evaluations. Therefore, we terminate all optimisation searches after 100 steps.

We structure our analysis in terms of the number of evaluations. For each cross-validation division and number of evaluations, we use the hyperparameter values that maximise validation AUROC up to that point. We start our analysis

by identifying the most suitable optimiser for each data descriptor, and adopt this for the rest of our analysis. We compare data descriptors both by summarising performance with the mean test AUROC, and by looking at individual results at the level of cross-validation divisions. In both cases, we apply a weighting scheme such that the 50 datasets from which the one-classification problems are drawn all contribute equally. In scatter plots, this is reflected in the size of the markers. We measure rank correlation with the weighted Kendall's $\tau$ (Vigna 2015).

In order to determine statistical significance, we apply clustered Wilcoxon signed-rank tests (Rosner et al. 2006) on the mean AUROC scores for each one-class classification problem. When we compare data descriptors to each other, we use the Holm-Bonferroni method (Holm 1979) to correct for family-wise error. Since there are only two comparisons for each data descriptor, this entails multiplying the smaller of the corresponding $p$-values by 2.

## 5 Results and analysis

Figure 1 shows the performance of the different optimisers with the data descriptors. The graph for SVM does not include results for the *skin* and *miniboone* datasets, because evaluating all optimisers on these large datasets proved to be infeasible.

With just one hyperparameter, NND is easier to optimise than ALP and SVM, and the optimisers all seem to reach their maximal value. This even includes random optimisation, which together with Malherbe-Powell and Bergstra-Bardenet achieves the highest weighted mean AUROC, illustrating that with enough evaluations, it will always match or outperform local optimisation strategies, which can get stuck in local optima. What is most surprising here is the poor performance of Kushner-Sittler optimisation, which seems to reach a plateau that stays below Nelder-Mead. It is possible that the discontinuity of the hyperparameter $k$ causes Kushner-Sittler optimisation to get stuck in local neighbourhoods.

In contrast, our maximum budget of fifty evaluations is not enough for random optimisation to outperform all local methods on the two-dimensional optimisation problems of ALP and SVM. In both cases, Malherbe-Powell optimisation becomes the best-performing method after a dozen of optimisations, whereas Kushner-Sittler and Bergstra-Bardenet optimisation perform not quite as good. All non-random optimisers seem to converge quicker with ALP than with SVM, and offer a greater advantage over random optimisation, which may be due to the essential finiteness of the number of hyperparameter combinations of ALP, and also because its problem curve may on average be easier to optimise.

On the basis of these experiments, we can say that Malherbe-Powell is the most consistently good option for these optimisation problems, and we will use its results for the proceeding analysis (including results on *skin* and *miniboone* for SVM). However, we also note that some of the differences are very small, and practitioners may want to prioritise ease of implementation when selecting an optimiser.

Figure 2 shows the distribution of the hyperparameter values after 50 evaluations. Generally speaking, these distributions are relatively uniform, which suggests that the chosen parametrisations are efficient. However, we note that for

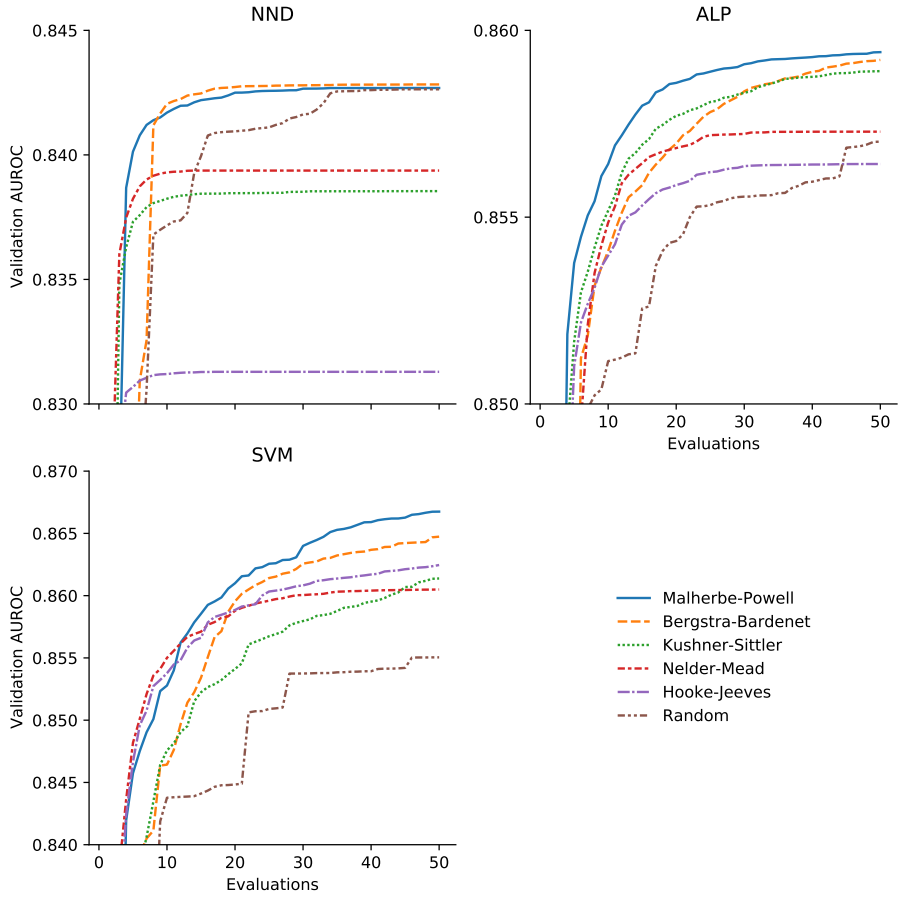
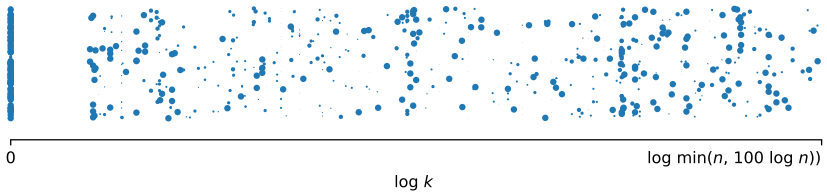



Fig. 1: Weighted mean validation AUROC of data descriptors with hyperparameters tuned with a number of different optimisers.

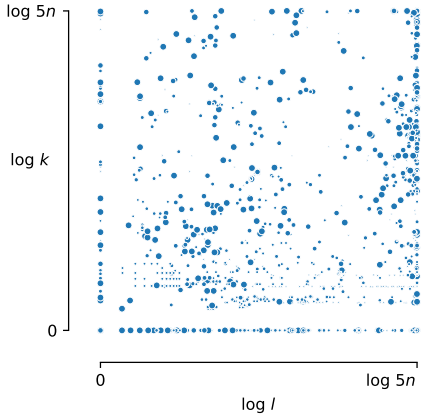
NND, $k$ equals 1 in 39% of all one-class classification problems. This indicates that $k$ often doesn't need to be optimised.

When we apply a clustered Wilcoxon signed rank test to compare the AUROC obtained with hyperparameter optimisation and the AUROC obtained with default hyperparameter values from Lenz et al. (2021), we find that optimised values start to outperform default values with great certainty ($p < 0.01$) within 2 (ALP), 3 (SVM) and 4 (NND) evaluations. With optimised values, SVM and NND also perform significantly better ($p < 0.01$) than ALP, the best data descriptor with default values, after 4 (SVM) and 20 (NND) evaluations. However, hyperparameter optimisation is not guaranteed to increase AUROC, especially with smaller datasets (Figure 3).

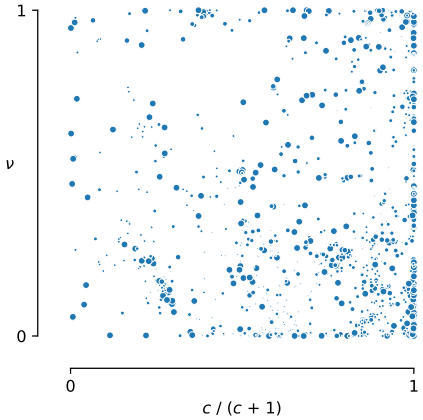
Figure 4 shows the weighted mean test AUROC of the data descriptors, as well as the difference with respect to the validation AUROC. This can be interpreted as the amount of overfitting, and should be taken into account when estimating the




(a) NND




(b) ALP




(c) SVM

Fig. 2: Distribution of optimised hyperparameter values for NND, ALP and SVM after 50 evaluations.

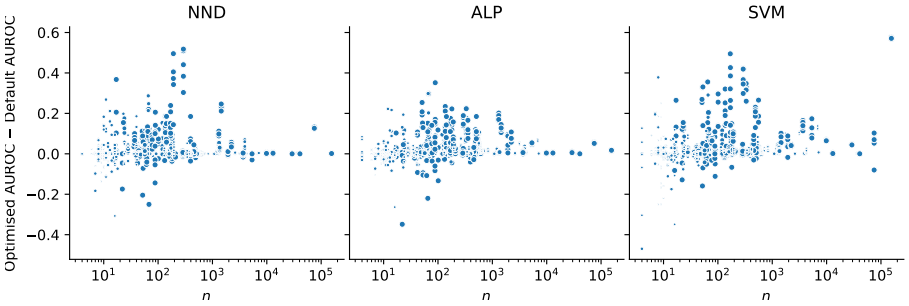



Fig. 3: Increased AUROC due to hyperparameter optimisation over default hyperparameter values, as a function of target class size $n$.

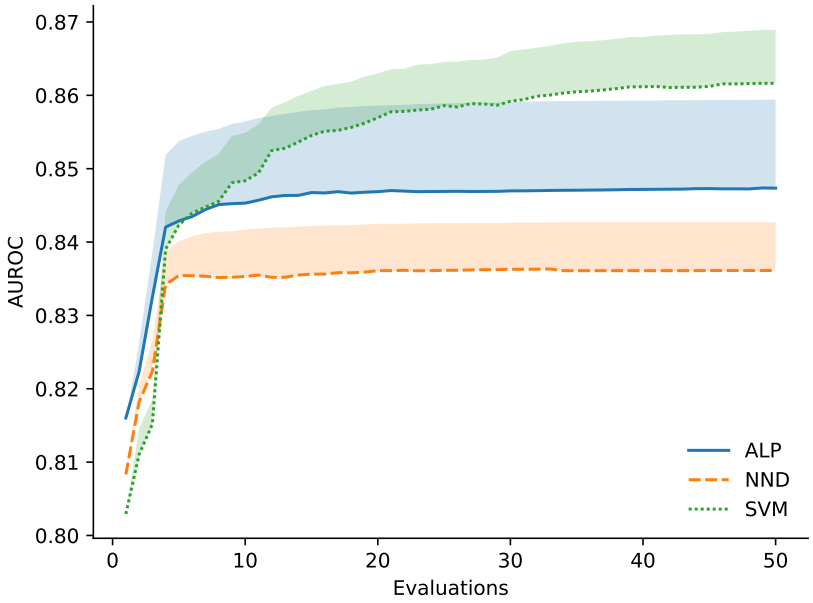



Fig. 4: Weighted mean test AUROC (solid line) and difference with validation AUROC (shaded) of data descriptors.

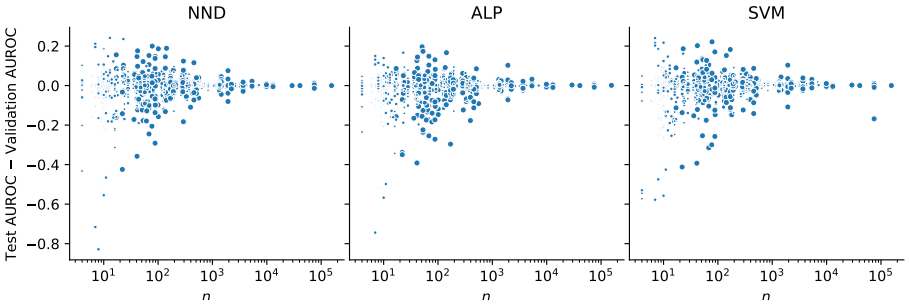



Fig. 5: Difference between test and validation AUROC as a function of target class size $n$.

real AUROC on the basis of validation AUROC. Overfitting is approximately twice as large with ALP as with SVM and NND, but all three data descriptors show essentially the same pattern: validation AUROC becomes increasingly accurate as the target set size grows (Figure 5). The weighted Kendall's $\tau$ for the differences between validation and test AUROC after 50 evaluations is 0.46 for ALP and SVM, 0.57 for ALP and NND, and 0.59 for SVM and NND.

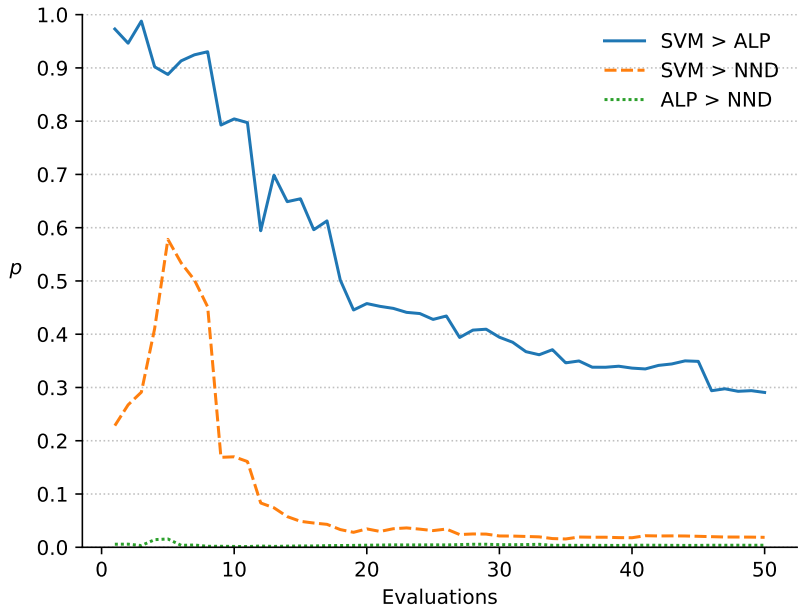



Fig. 6: One-sided $p$-values of clustered Wilcoxon signed rank tests that one data descriptor is better than another (uncorrected for multiple testing).

Table 1: $p$-values of clustered Wilcoxon signed rank tests after 50 evaluations, corrected for multiple testing.

| Test | $p$ |
|---|---|
| SVM > ALP | 0.29 |
| SVM > NND | 0.037 |
| ALP > NND | 0.0077 |

The three data descriptors display varying sensitivity to hyperparameter optimisation. All three test AUROC curves increase steeply for 4 evaluations and then flatten out. However, for SVM, the initial rise is steeper and its curve continues to increase for much longer, allowing it to quickly surpass NND and ALP from a lower starting position. NND, ALP and SVM approach their final scores (after 50 evaluations) to within 0.001 points after respectively 5, 13 and 37 evaluations.

To determine whether these difference in performance are statistically significant, we perform one-sided clustered Wilcoxon signed rank tests. The resulting $p$-values after each evaluation are displayed in Figure 6. Since these $p$-values are one-sided, the $p$-value for the opposite test can be obtained by subtracting the value from 1. Table 1 lists the $p$-values after 50 evaluations, corrected for multiple testing.

Table 2: Fraction of one-class classification problems with higher training and test AUROC by ALP or SVM.

| Validation AUROC | Test AUROC | | | |
|---|---|---|---|---|
| | ALP < SVM | ALP = SVM | ALP > SVM | Total |
| ALP < SVM | 0.37 | 0.031 | 0.088 | 0.49 |
| ALP = SVM | 0.00080 | 0.033 | 0.0052 | 0.039 |
| ALP > SVM | 0.12 | 0.038 | 0.31 | 0.47 |
| Total | 0.49 | 0.10 | 0.40 | |

Based on these experiments, we can confidently say that with sufficient evaluations, ALP and SVM outperform NND. We have far less certainty about the relative performance of ALP and SVM. The data suggests that when the number of evaluations is small, ALP outperforms SVM, and vice-versa when the number of evaluations is large, but there is a large possibility that these observations are simply due to chance.

If we focus on the performance after 50 evaluations (Table 2), we see that SVM obtains a higher AUROC than ALP slightly more often than vice-versa, both on validation and test data. This confirms that the average performance of ALP and SVM is very close in practice. However, for individual classification problems, the choice still matters. We note that which data descriptor performs better is fairly consistent between validation and test data. If for each classification problem, we choose the data descriptor that obtains a higher validation AUROC (choosing ALP in event of a tie), it will perform worse on test data in only 21% of cases. The advantage of this combination of ALP and SVM over either of ALP or SVM on its own is highly significant ($p < 0.0001$), regardless of whether we choose for each fold separately or on the basis of the mean validation AUROC across folds.

Figure 7 shows the distributions of test AUROC scores after 50 evaluations. These are highly rank-correlated: the weighted Kendall's $\tau$ is 0.80 for ALP and SVM, 0.83 for ALP and NND, and 0.86 for SVM and NND. One visually identifiable difference is that SVM has a lot fewer scores below 0.5 than ALP and NND. However, all AUROC below 0.5 is worse than chance. If no AUROC significantly higher than 0.5 can be achieved on a classification problem, then the problem is essentially unsolvable. For the purpose of the present analysis, we can express the difficulty of a one-class classification problem as the maximum of the AUROC achieved by ALP and SVM. Figure 8 plots the relative performance of ALP and SVM against this difficulty. SVM is better able to separate more difficult problems, but for problems for which a good AUROC of 0.8 or more can be achieved, ALP beats SVM more often (46%) than vice-versa (41%), with a weighted mean difference of 0.0021.

Finally, Figure 9 displays the run times of hyperparameter optimisation per evaluation. All three data descriptors require a similar amount of time for their first evaluation, although this is evaluation-dependent. However, the greater efficiency of optimisation with ALP and especially NND (as explained in Section 2) is clearly apparent thereafter. Each additional evaluation takes 9.37 times as long for SVM as for ALP, and 16.3 times as long for ALP as for NND. This effect is compounded by the finding, reported above, that optimisation of NND requires fewer evaluations

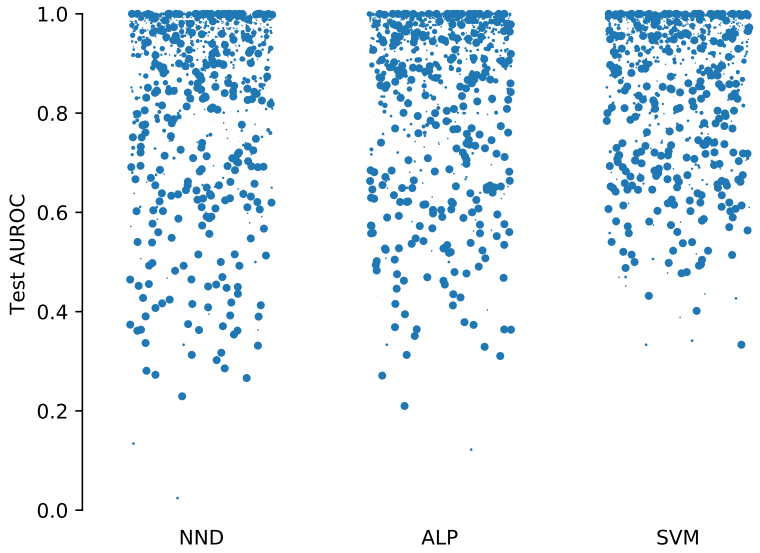



Fig. 7: Distribution of test AUROC scores.

than ALP, which in turn requires fewer evaluations than SVM. In fact, NND requires so few evaluations that its line in Figure 9 stops short.

## 6 Conclusion

In this paper, we have presented a thorough analysis of hyperparameter optimisation for the three most promising data descriptors from the literature, SVM, NND and ALP. We have explained how NND and ALP can be optimised efficiently with a single nearest neighbour query and leave-one-out validation, while SVM requires a similar computation for each hyperparameter evaluation. We then evaluated the performance of hyperparameter optimisation empirically, based on a large selection of 246 one-class classification problems drawn from 50 datasets.

Of a selection of optimisation algorithms, the recent Malherbe-Powell approach provides the best overall performance with all three data descriptors. ALP is more sensitive to overfitting than SVM and NND, but in all three cases overfitting reduces with target set size. For all three data descriptors, optimised hyperparameters significantly outperform default hyperparameter values after a handful of optimisations. As predicted, different hyperparameter values can be evaluated more efficiently for NND and ALP than for SVM. In addition, the optimisation of NND and ALP requires fewer evaluations than SVM. After 50 evaluations, ALP and SVM significantly outperform NND. SVM also outperforms ALP on our datasets, but the difference is not significant.

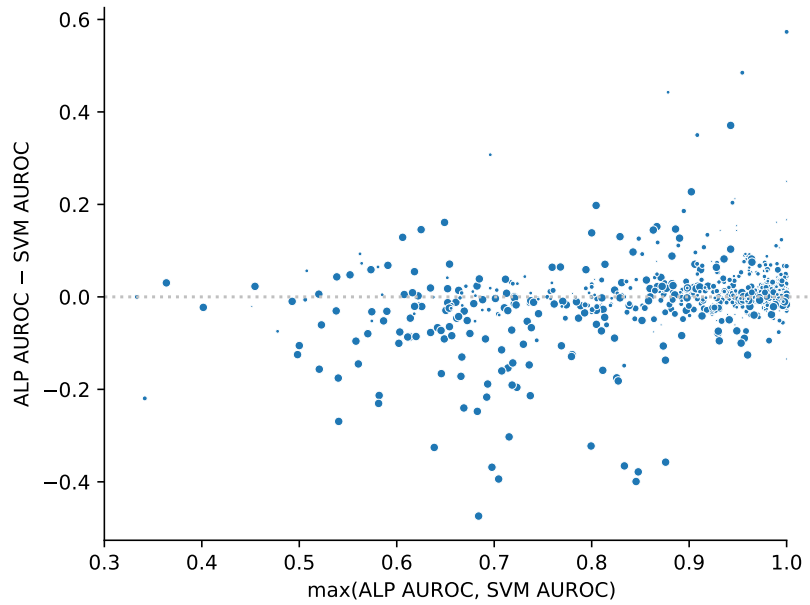



Fig. 8: Difference between ALP and SVM AUROC, as a function of the difficulty of one-class classification problems, expressed by the maximum of ALP and SVM AUROC.

A more detailed look at the difference between ALP and SVM revealed that their strengths are to some extent complementary, and that selecting one or the other based on their validation AUROC gives the best results. The strongest relative advantage of SVM lies with difficult one-class classification problems, while ALP performs better with problems with which a good AUROC of 0.8 or higher can be achieved.

Overall, we come to the following conclusion. NND is a very simple data descriptor that can be optimised very efficiently. While the resulting gain in performance is limited, it nevertheless leads to results that are generally better than what can be obtained with a data descriptor with default hyperparameter values. SVM is a data descriptor with excellent performance, but it is expensive to optimise. The performance of ALP rivals that of SVM, and potentially surpasses it with one-class classification problems that admit a good solution, but it can be optimised much more efficiently.

Therefore, we find that ALP is a good default choice, while NND may appeal to practitioners constrained by a smaller computational budget. If the absolute best performance is desired, we recommend that practitioners consider both ALP and SVM, and make the choice dependent on validation AUROC.

In future research, we think that it could be worthwhile to investigate in greater detail what properties of datasets determine the relative strengths and weaknesses

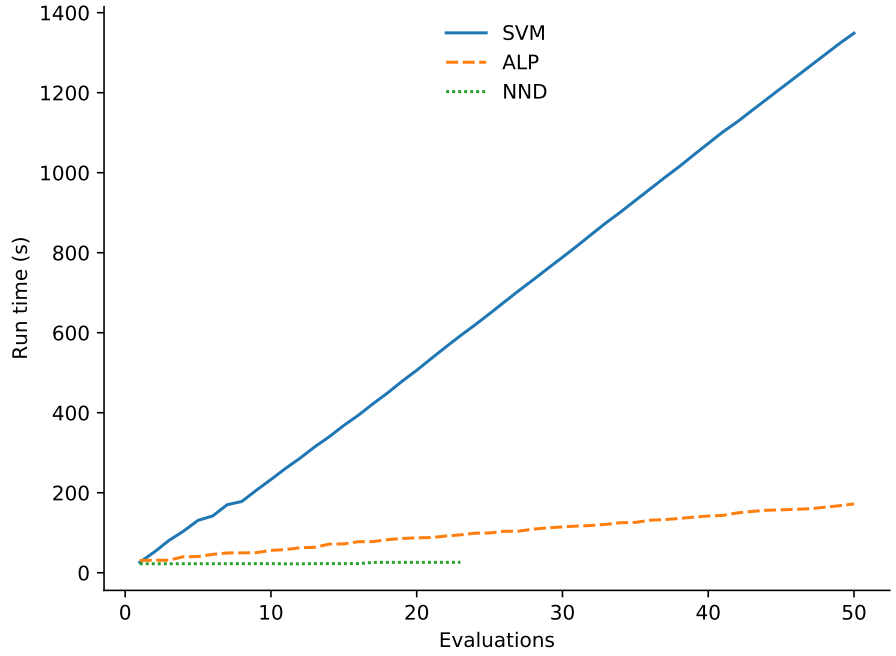



Fig. 9: Mean run times (5 runs) of hyperparameter optimisation with Malherbe-Powell on a training set with $2^{13}$ target class instances and $2^{13}$ other instances, drawn from the *miniboone* dataset (target class 1).

of ALP and SVM. A deeper understanding of this question could in turn be applied to modify the ALP and SVM algorithms.

We have focused our attention in this paper on a handful of hyperparameters with the most immediate impact on the classification of different datasets. But these are not the only choices available to a practitioner. Hyperparameter optimisation is a specific form of model selection, and conversely, any modification to a classification algorithm can be seen as a hyperparameter choice. In particular, one large topic that we have set aside in the present paper is the possibility to change how similarity and difference are measured, by choosing a different metric, kernel and/or scaling function. These are essentially open-ended choices, so part of the challenge lies in delineating the search area.

In the nearby future, we plan to investigate how the optimisation of hyperparameters can best be integrated into data descriptor ensembles in a multi-class setting.

**Acknowledgements**

The research reported in this paper was conducted with the financial support of the Odysseus programme of the Research Foundation – Flanders (FWO). D.

Peralta is a Postdoctoral Fellow of the Research Foundation – Flanders (FWO, 170303/12X1619N).

### Conflict of interest

The authors declare that they have no conflict of interest.

### References

Ban T, Abe S (2006) Implementing multi-class classifiers by one-class classification methods. In: IJCNN 2006: Proceedings of the IEEE International Joint Conference on Neural Networks, IEEE, pp 327–332

Benavoli A, Corani G, Mangili F (2016) Should we really use post-hoc tests based on mean-ranks? Journal of Machine Learning Research 17(5):152–161

Bergstra J, Bengio Y (2012) Random search for hyper-parameter optimization. Journal of Machine Learning Research 13(10):281–305

Bergstra J, Bardenet R, Bengio Y, Kégl B (2011) Algorithms for hyper-parameter optimization. In: NIPS 2011: Proceedings of the 25th Annual Conference on Neural Information Processing Systems, NIPS, Advances in Neural Information Processing Systems, vol 24, pp 2546–2554

Betrò B (1991) Bayesian methods in global optimization. Journal of Global Optimization 1(1):1–14

Blank J, Deb K (2020) Pymoo: Multi-objective optimization in python. IEEE Access 8:89497–89509

Brochu E, Cora VM, de Freitas N (2009) A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Tech. Rep. UBC TR-2009-023, University of British Columbia, Department of Computer Science, URL `http://arxiv.org/abs/1012.2599`

Demšar J (2006) Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research 7(1):1–30

Dua D, Graff C (2019) UCI machine learning repository. URL `http://archive.ics.uci.edu/ml`

Holm S (1979) A simple sequentially rejective multiple test procedure. Scandinavian Journal of Statistics 6(2):65–70

Hooke R, Jeeves TA (1961) "direct search" solution of numerical and statistical problems. Journal of the ACM 8(2):212–229

Janssens JHM, Flesch I, Postma EO (2009) Outlier detection with one-class classifiers from ML and KDD. In: ICMLA 2009: Proceedings of the Eighth International Conference on Machine Learning and Applications, IEEE, pp 147–153

Jones DR (2001) A taxonomy of global optimization methods based on response surfaces. Journal of Global Optimization 21(4):345–383

King DE (2009) Dlib-ml: A machine learning toolkit. Journal of Machine Learning Research 10(60):1755–1758

King DE (2017) A global optimization algorithm worth using. `http://blog.dlib.net/2017/12/a-global-optimization-algorithm-worth.html`, last accessed 6 Jan 2021

Knorr EM, Ng RT (1997) A unified notion of outliers: Properties and computation. In: KDD-97: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, AAAI, pp 219–222

Kushner HJ (1962) A versatile stochastic model of a function of unknown and time varying form. Journal of Mathematical Analysis and Applications 5(1):150–167

Kushner HJ (1964) A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. Journal of Basic Engineering 86(1):97–106

Lenz OU, Peralta D, Cornelis C (2020) fuzzy-rough-learn 0.1: a Python library for machine learning with fuzzy rough sets. In: IJCRS 2020: Proceedings of the International Joint Conference on Rough Sets, Springer, Lecture Notes in Artificial Intelligence, vol 12179, pp 491–499

Lenz OU, Peralta D, Cornelis C (2021) Average Localised Proximity: a new data descriptor with good default one-class classification performance, URL `http://arxiv.org/abs/2101.11037`, under review

Malherbe C, Vayatis N (2017) Global optimization of Lipschitz functions. In: ICML 2017: Proceedings of the 34th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol 70, pp 2314–2323

Nelder JA, Mead R (1965) A simplex method for function minimization. The Computer Journal 7(4):308–313

Paleyes A, Pullin M, Mahsereci M, Lawrence N, González J (2019) Emulation of physical processes with Emukit. In: NeurIPS 2019: Workshop on Machine Learning and the Physical Sciences, NeurIPS

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay É (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12(85):2825–2830

Powell MJD (2004) The NEWUOA software for unconstrained optimization without derivatives. Tech. Rep. NA2004/08, University of Cambridge, Department of Applied Mathematics and Theoretical Physics, URL `http://www.damtp.cam.ac.uk/user/na/NA{_}papers/NA2004{_}08.pdf`

Powell MJD (2009) The BOBYQA algorithm for bound constrained optimization without derivatives. Tech. Rep. NA2009/06, University of Cambridge, Department of Applied Mathematics and Theoretical Physics, URL `http://www.damtp.cam.ac.uk/user/na/NA{_}papers/NA2009{_}06.pdf`

Rosner B, Glynn RJ, Lee MLT (2006) The Wilcoxon signed rank test for paired comparisons of clustered data. Biometrics 62(1):185–192

Rousseeuw PJ, Croux C (1993) Alternatives to the median absolute deviation. Journal of the American Statistical Association 88(424):1273–1283

Schölkopf B, Platt JC, Shawe-Taylor J, Smola AJ, Williamson RC (1999) Estimating the support of a high-dimensional distribution. Tech. Rep. MSR-TR-99-87, Microsoft Research, Redmond, Washington

Schölkopf B, Platt JC, Shawe-Taylor J, Smola AJ, Williamson RC (2001) Estimating the support of a high-dimensional distribution. Neural Computation 13(7):1443–1471

Spendley W, Hext GR, Himsworth FR (1962) Sequential application of simplex designs in optimisation and evolutionary operation. Technometrics 4(4):441–461

Swersky L, Marques HO, Sander J, Campello RJGB, Zimek A (2016) On the evaluation of outlier detection and one-class classification methods. In: DSAA

2016: Proceedings of the 3rd IEEE International Conference on Data Science and Advanced Analytics, IEEE, pp 1–10

Tax DMJ (2001) One-class classification: Concept learning in the absence of counter-examples. PhD thesis, Technische Universiteit Delft

Tax DMJ, Duin RPW (1999a) Data domain description using support vectors. In: ESANN 1999: Proceedings of the Seventh European Symposium on Artificial Neural Networks, D-Facto, pp 251–256

Tax DMJ, Duin RPW (1999b) Support vector domain description. Pattern Recognition Letters 20(11–13):1191–1199

Torczon VJ (1989) Multidirectional search: a direct search algorithm for parallel machines. PhD thesis, Rice University

Vigna S (2015) A weighted correlation index for rankings with ties. In: WWW '15: Proceedings of the 24th international conference on World Wide Web, pp 1166–1176

Virtanen P, Gommers R, Oliphant TE, Haberland M, Reddy T, Cournapeau D, Burovski E, Peterson P, Weckesser W, Bright J, van der Walt SJ, Brett M, Wilson J, Millman KJ, Mayorov N, Nelson ARJ, Jones E, Kern R, Larson E, Carey CJ, Polat İ, Feng Y, Moore EW, VanderPlas J, Laxalde D, Perktold J, Cimrman R, Henriksen I, Quintero EA, Harris CR, Archibald AM, Ribeiro AH, Pedregosa F, van Mulbregt P, SciPy 1.0 Contributors (2020) SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. Nature Methods 17(3):261–272

Wright MH (1995) Direct search methods: Once scorned, now respectable. In: Numerical analysis 1995: Proceedings of the 16th Dundee Biennial Conference on Numerical Analysis, Longman, Pitman Research Notes in Mathematics Series, vol 344, pp 191–208