





Multidimensional Adaptative k NN over Tracking Outliers (Makoto)

Jessy Colonval^(✉)  and Fabrice Bouquet 

FEMTO-ST Institut, CNRS, Univ. de Franche Comté (UFC), Besançon, France
{jessy.colonval,fabrice.bouquet}@femto-st.fr

Abstract. This paper presents an approach to detect outliers present in a data set, also called aberration. These outliers often cause problems to the learning algorithms by deviating their behavior, which makes them less efficient. It is therefore necessary to identify and remove them during the cleaning data step before the learning process. For this purpose, a method that detects if data is an outlier from its k nearest neighbors is proposed for multidimensional data sets. In order to make the method more accurate, the number of k nearest neighbors chosen is adaptive for each class present in the data set, and each neighbor has a different weight in the decision, depending on their respective proximity. The proposed method is called Makoto for Multidimensional Adaptative k NN Over Tracking Outliers. The effectiveness of this method is compared with four other known methods based on different principles: LOF (Local Outlier Factor), Isolation forest, One Class SVM and Inter Quartile Range (IQR). Thus, on the basis of 406 synthetic data sets and 17 real data sets with distinct characteristics, the Makoto method appears to be more efficient.

Keywords: Machine learning · Data filtering · Spatial Outlier Detection · Kernel Functions · Adaptative k NN · Correlation

1 Introduction

An important principle in the use of artificial intelligence algorithms is the quality of the learning data. The implementation of tools to help filter the data is a critical point [14]. We are interested here in a particular case, that of outliers. These are data that contain information that is incompatible with the rest of the data set. This data can have two effects:

1. Weaken the predictive power of the model obtained at the end of the learning phase.
2. Weaken the score obtained from the model during its validation phase.

It is important to define a few terms that will be used in this paper. A data set can be visualized as a table with rows and columns. Each row represents an individual, often unique, while the columns represent the characteristics of that

individual. There are several terms to designate them, but here, the rows will be called *points* and the columns will be called *attributes*.

In general, there are two types of attributes [1, Chap 11]:

- **Behavioral attributes** is an attribute of interest measured for each point. Points often have only one, but they can have several. These attributes are often non-spatial because it measures a certain quantity. For example, the type of glass, the presence of a heart abnormality or the description of an image. The values of a behavioral attribute are often called class, but here we will call it behavioral value to avoid confusion.
- **Contextual attributes** is an attribute expressing the characteristics of a point is defined on a continuous domain of values, also called spatial attributes. A point often has several of them. For example: the composition of a glass pane, the number of heartbeats per minute, or the color of a pixel.

We are interested in the detection of what we call spatial outliers, i.e. established from a neighborhood. The main criterion of these types of outlier search is the auto-correlation property, i.e. the fact that data in a neighborhood are closely correlated. Thus, an outlier is defined as an abrupt change in behavioral attributes among nearby points according to their contextual attributes.

However, there are two ways to create a neighborhood, depending on the nature of the data used:

- **Multidimensional methods** determine the neighborhood based on a distance between each point.
- **Graph-based methods** determine the neighborhood from the linkage relations between the points. For example, by using edges between nodes, where each node is associated with behavioral attributes.

We are only interested in multidimensional data sets. Thus, the neighborhood will be established from a distance calculation, euclidean for the Makoto method.

Neighborhood-based multidimensional spatial outlier detection methods have already been proposed [5, 13]. They differ in the way they establish the neighborhood and how they combine them to make an outlier prediction. The problem is that these techniques consider the neighbors equitably in the final decision, and the majority use all attributes for the creation of the neighborhood.

First, considering all neighbors as equal can be problematic because the neighbors are not equidistant from the starting point. And according to the auto-correlation property, it would make sense to give more importance to the nearest neighbors. For example, when the difference between the nearest and the farthest neighbor is important, then it is intuitive to want to give more importance to the nearest one. However, there are several ways to weight a neighborhood so, depending on the method chosen, it is possible to obtain several outlier predictions. Some methods have decided to generalize this idea and uses different weighting methods of the neighborhood [11, 16].

Second, using all contextual attributes in the establishment of the neighborhood can be problematic because of the likely presence of randomly distributed

attributes. These attributes are noise and can hinder outlier detection by making some of them undetectable or by considering data as falsely outlier [10]. Moreover, using fewer attributes leads to performance and resource gains, but this is not the main objective

In Sect. 2 we will describe the proposed method in detail and then Sect. 3 will compare this method with others from the state of the art to demonstrate its effectiveness.

2 Principle of the Method Makoto

This section presents the principle of the outlier detection method *Makoto* (Multidimensional Adaptive k NN Over Tracking Outliers). It can detect outliers only on multidimensional data sets, i.e. with continuous values. This method establishes the outliers using a neighborhood whose number is adaptive according to the distribution of the data set (Sect. 2.3). Each neighbor has a different weight in the decision-making, and this weight is computed using a kernel function (Sect. 2.1 and 2.2). In order to avoid being biased by the choice of a kernel function and to be more confident in the detection, we decide to use several kernel functions and to make a majority vote among their decisions to determine if a point is an outlier or not. Moreover, we use several subsets of the original data set to compute different neighborhoods that will each bring a prediction so that in the end the majority decision will prevail (Sect. 2.4).

2.1 Kernel Functions

To determine the weight of each neighbor, we decide to use the kernel functions [9]. This function gives a weight according to the distance of a neighbor to the origin point. We consider that the distances are between -1 and 1, where 0 is the shortest distance and -1 and 1 are the farthest.

These functions have several properties. They reach their maximum in 0 and decrease as the distance increases. Opposite distances are considered to be equivalent, i.e., their weight is the same. The weighting must be positive or equal to 0. Thus, the following properties must be respected [7]:

- positive, $\forall x \in \mathbb{R}, f(x) \geq 0$;
- max in 0, $\max_{x \in \mathbb{R}} f(x) = f(0)$;
- opposite, $\forall x \in \mathbb{R}, f(x) = f(-x)$;
- continuous by party $\forall x, y \in \mathbb{R}, x < y, f(x) \geq f(y)$.

As mentioned at the beginning of this section, this method use a majority vote on several kernel functions. There are several kernel functions in the literature, only the most common will be studied [9]. However, as shown in Fig. 1, some of the kernel functions are similar. The risk if we use all these functions, in the majority vote, is that a subpart of these functions influence the vote because they often obtain identical results. Thus, some votes will be counted several

- (a) Rectangular: $\begin{cases} \frac{1}{2} & \text{if } |x| \leq 1; \\ 0 & \text{otherwise.} \end{cases}$
- (b) Triangular: $\begin{cases} (1 - |x|) & \text{if } |x| \leq 1; \\ 0 & \text{otherwise.} \end{cases}$
- (c) Biweight: $\begin{cases} \frac{15}{16}(1 - x^2)^2 & \text{if } |x| \leq 1; \\ 0 & \text{otherwise.} \end{cases}$
- (d) Triweight: $\begin{cases} \frac{35}{32}(1 - x^2)^3 & \text{if } |x| \leq 1; \\ 0 & \text{otherwise.} \end{cases}$
- (e) Tricube: $\begin{cases} \frac{70}{81}(1 - |x|^3)^3 & \text{if } |x| \leq 1; \\ 0 & \text{otherwise.} \end{cases}$
- (f) Epanechnikov: $\begin{cases} \frac{3}{4}(1 - x^2) & \text{if } |x| \leq 1; \\ 0 & \text{otherwise.} \end{cases}$
- (g) Gaussian: $\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$
- (h) Cosine: $\begin{cases} \frac{\pi}{4} \cos(\frac{\pi}{2}x) & \text{if } |x| \leq 1; \\ 0 & \text{otherwise.} \end{cases}$
- (i) Inverse: $\begin{cases} \infty & \text{if } x = 0; \\ \frac{1}{|x|} & \text{otherwise.} \end{cases}$

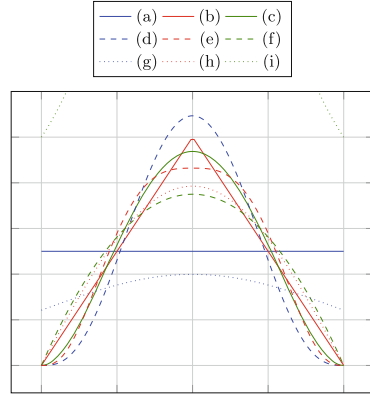


Fig. 1. Kernel functions below in common coordinate system.

times and will have more importance than they should have, which will bias the result. So the decision will be taken only by a specific profile of kernel function and the others will be ignored.

To verify this bias, we calculate the correlation of the votes that each kernel function produces. We calculate this score from the real data sets, presented in the Sect. 3.2, with the Pearson’s method. For each of them, the outlier detection is performed with each kernel function to see which points are commonly considered as outliers and then to establish a correlation score to determine which functions often give the same results. Thus, the values in Table 1 are the mean and standard deviation of the correlation scores obtained for a pair of kernel functions over all data sets.

Assuming that correlation scores above 90% give such similar results that it is equivalent to doing the same thing then these functions can be separated into only three different groups:

- Rectangular and Gaussian;
- Triangular, Biweight, Triweight, Tricube, Cosine and Epanechnikov;
- Inverse.

As previously mentioned, if we use simultaneously these functions in the majority vote, then the decision will be made only by the second group. So the final result will have globally the same result as if it had been done with one of the functions of the second group. This vote will be biased and questions the relevance of using all these functions. Thus, we recommend the use of several functions that have different profiles and capture different behaviors.

Table 1. Correlation scores of outlier votes for each kernel function.

	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(h)	(i)
(a)	— ± 0.14	0.68 ± 0.14	0.66 ± 0.14	0.62 ± 0.14	0.66 ± 0.14	0.73 ± 0.12	0.94 ± 0.04	0.71 ± 0.12	0.49 ± 0.17
(b)	0.68 ± 0.14	— ± 0.03	0.96 ± 0.03	0.93 ± 0.05	0.96 ± 0.04	0.92 ± 0.04	0.74 ± 0.12	0.94 ± 0.04	0.71 ± 0.14
(c)	0.66 ± 0.14	0.96 ± 0.03	— ± 0.03	0.94 ± 0.03	0.98 ± 0.02	0.91 ± 0.05	0.72 ± 0.12	0.93 ± 0.04	0.71 ± 0.14
(d)	0.62 ± 0.14	0.93 ± 0.05	0.94 ± 0.03	— ± 0.04	0.93 ± 0.04	0.86 ± 0.07	0.68 ± 0.13	0.87 ± 0.07	0.76 ± 0.13
(e)	0.66 ± 0.14	0.96 ± 0.04	0.98 ± 0.02	0.93 ± 0.04	— ± 0.05	0.92 ± 0.05	0.72 ± 0.12	0.93 ± 0.04	0.70 ± 0.15
(f)	0.73 ± 0.12	0.92 ± 0.04	0.91 ± 0.05	0.86 ± 0.07	0.92 ± 0.05	— ± 0.10	0.79 ± 0.10	0.98 ± 0.01	0.65 ± 0.16
(g)	0.94 ± 0.04	0.74 ± 0.12	0.72 ± 0.12	0.68 ± 0.13	0.72 ± 0.12	0.79 ± 0.10	— ± 0.10	0.78 ± 0.10	0.52 ± 0.16
(h)	0.71 ± 0.12	0.94 ± 0.04	0.93 ± 0.04	0.87 ± 0.07	0.93 ± 0.04	0.98 ± 0.01	0.78 ± 0.10	— ± 0.16	0.66 ± 0.16
(i)	0.49 ± 0.17	0.71 ± 0.14	0.71 ± 0.14	0.76 ± 0.13	0.70 ± 0.15	0.65 ± 0.16	0.52 ± 0.16	0.66 ± 0.16	—

The use of kernel functions, in Makoto method, is based on the assumption that the closer the neighbors are, the more they share common characteristics with the point of origin. However, this neighborhood is established from a Euclidean distance calculation which, by its simplicity, gives only an approximation of the real distance between the points. Thus, it is possible that the points considered to be nearest by the Euclidean calculation are not in reality so, and therefore do not have the characteristics closest to the point of origin. By putting aside this principle, we can imagine other kernel functions which take more care of further points or located on specific distance slices. For example, from the sinusoidal, we get the functions in Fig. 2. Makoto will use these kernel functions with the addition of the Rectangular function.

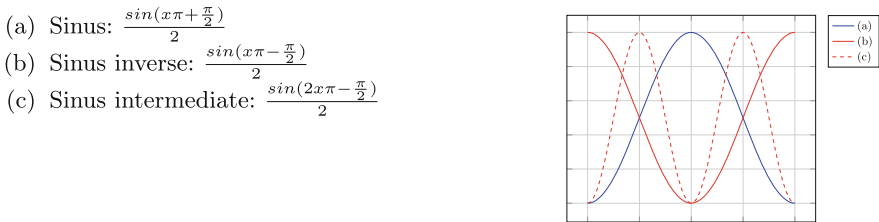


Fig. 2. Kernel functions sinusoidal.

2.2 Weighted k Nearest Neighbor Classification Method

These methods are based on the idea that the closest neighbors should have a higher weight than the farthest neighbors in the prediction. This prediction will be used to determine if a point is an outlier. To do this, the distances on which the search for neighbors is based must be transformed into weight using the functions presented in Sect. 2.1.

The wk NN classification method is used to combine the behavioral attributes of the neighborhood in order to make a prediction. It has been generalized to take the existence of several behavioral attributes within the same point. The prediction algorithm takes the following form:

Step 1 \rightarrow Define the number of neighbors k , the distance function d , usually the Euclidean distance, and the kernel function f .

Step 2 \rightarrow Let $O = \{(X_i, B_i) | i \in [1, n]\}$ the set of n points where X_i is the set of contextual attributes and B_i is the set of behavioral attributes. Steps 3 to 7 are applied for each object, we note the current one (X, B) .

Step 3 \rightarrow Find the k nearest neighbors to the current point using the method distance only on contextual attributes, such as $d(X, X_i)$. This set will be denoted $K = \{(X_{k_i}, B_{k_i}) | i \in [1, k]\}$.

Step 4 \rightarrow As in the previous step, find the $k + 1$ nearest neighbor to the current object. This neighbor will be denoted (X_{k+1}, B_{k+1}) and will be used to normalize the distances between the point and its neighbors.

Step 5 \rightarrow Normalize the distances of the k nearest neighbors such as: $D_{k_i} = D(X_{k_i}, X_{k+1}) = \frac{d(X, X_{k_i})}{d(X, X_{k+1})}$

Step 6 \rightarrow Transform the normalized distance into weight using the kernel function, $w_{k_i} = f(D_{k_i})$.

Step 7 \rightarrow Let $b \in B$ be a behavioral attribute and C_b the set of possible classes for b then the set of predictions is defined as the class with the highest weighting for each behavioral attribute:

$$\hat{B}_{(X,B)} = \{\forall b \in B | \max_b (\forall c \in C_b | \sum_{k_i \in K} \begin{cases} \text{if } c_{k_i} = c, w_{k_i} \\ \text{otherwise, } 0 \end{cases})\} \quad (1)$$

For example, a data set whose contextual attributes are planar coordinates and the behavioral attribute is a colored geometric figure, a *red circle* or a *blue rectangle*. Let a point among this set whose coordinates are center and its unknown geometrical shape will be symbolized by a *green star*. The algorithm presented above is used to determine the shape of the object. To begin, the five closest neighbors are found using the Euclidean distance. To normalize the distances, a sixth neighbor is found, which is not visually represented because it has no influence in the final decision. The weight of each neighbor can be calculated from the normalized distance. Thus, the prediction in the shape of the point can change according to the chosen kernel function. Figure 3 plots the weight of each neighbor and the resulting prediction for the Rectangular, Triangular, Triweight and Gaussian kernel functions. The gradient represents the relative weight according to the relative distance to the $k + 1$ neighbor. The darker it is, the more weight the neighbor will have.

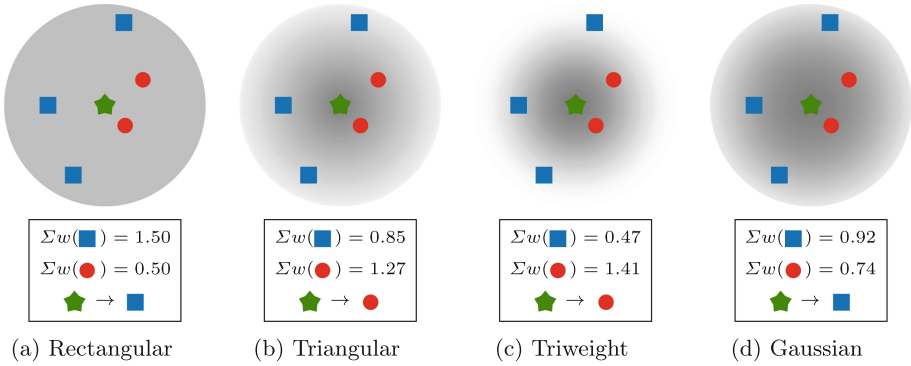


Fig. 3. Predictions according to the chosen kernel function.

This example illustrates the influence of kernel functions in the prediction. Here, the *blue rectangles* are in the majority but further from the evaluated point than the *red circles*. Thus, the prediction changes according to the difference between the maximum and minimum of the kernel function used (see Fig. 1). In this example, the Triangular and Triweight functions vary enough that the proximity of the red circles counterbalances the numerical advantage of the blue rectangles. In contrast, the Rectangular and Gaussian functions vary little or not at all to allow this.

2.3 Adaptive k per Class

The choice of the number of neighbors k is important because it affects the predictions made. A bad value of k decreases the quality of prediction and thus the quality of outlier detection. If the number of neighbors k is too large, then small groups of points with the same behavioral values may be underrepresented by different neighboring groups and therefore the prediction will not be representative of reality. The presence of this type of such cases when the number of representatives for each behavioral value is unbalanced. For example, in Fig. 3 the number of neighbors k is equal to 5 but if this number is decreased then the weights given by the kernel functions change and the predictions of Rectangular and Gaussian move from *blue rectangle* to *red circle*. And can change if the number k had been higher. Finally, is 5 the right value of k for this example?

Moreover, choosing a unique number of neighbors k is not always relevant, and it is better to choose an adaptive number depending on the point being evaluated. Several methods to establish these numbers have been proposed, and their efficiency compared to the use of a unique k has already been demonstrated [6]. The main problem is the asymmetry of the data sets in the distribution of behavioral values. A number k that is relevant for the majority classes can be destructive for the minority ones. If this number is too high compared to the absolute number of representatives, then we will fall into the case discussed

above. This will cause the suppression of these minority classes. Based on this principle, the optimal number k is different for each behavioral value.

The formula for calculating the number of neighbors per class is inspired by the one proposed by Baoli et al. [3]. It has been modified to avoid the use of constants, so the constant α is removed to be replaced by the lower bound equal to 3 to handle cases where the value given by the equation is too low. While the starting value k in the original equation is approximated by the formula $\sqrt{\frac{\max(\forall y \in V_b | N(y))}{2}}$.

Let $b \in B$ be a behavioral attribute and V_b the set of behavioral values of b then for each value $x \in V_b$ the number of neighbors is determined according to the following equation:

$$3 \leq \frac{\sqrt{\frac{\max(\forall c \in V_b | N(c))}{2}} * N(x)}{\max(\forall y \in V_b | N(y))} \leq \min(\forall y \in V_b | N(y)) \tag{2}$$

where $N(-)$ gives the number of elements with the same behavioral value provides as parameter. The number of neighbors must be between 3 and the number of elements of the smallest class. The lower bound is 3 because it is the minimum number of neighbors that we consider reasonable to take to make a decision, less would be absurd. The upper bound is the number of elements of the minority class, so a behavioral value cannot be underrepresented in its neighborhood by a majority class and can influence decisions.

To illustrate the relevance of the upper bound, the *Shuttle* data set (see Sect. 3.2) will be used as an example. The specificity of this data set is the great disparity in the distribution of points for each behavioral value, i.e. between 10 and 45 000. Thus, in the case where a point initially has a behavioral value of **1** when it should be **6**, then the number k used without the bound is equal to 150 (see Table 2). Logically, these nearest neighbors will have a behavioral value equal to **6**, but it only has 10 points in the data set. In the best case where these points were the 10 nearest neighbors, there are still 140 neighbors who will probably be of the majority behavioral value, i.e. **1**. Even with a good neighborhood weighting, the 140 neighbors will have more weight and this hypothetical outlier cannot be detected. It is therefore necessary to limit this number of neighbors in order to ensure that even minority behavioral values can have an influence in the detection.

Table 2. The neighbor number k calculated with and without the upper bound in Eq. 2 for *Shuttle* data set.

Behavioral value	1	2	3	4	5	6	7
Number of members	45 586	50	171	8 903	3 267	10	13
the number k	without bound	150	3	3	29	10	3
	with bound	10	3	3	10	10	3

2.4 Sub Data Sets and Sub Contextual Attributes and Filtering

In order to make the method more efficient and generalized to avoid **overfitting**, we decide to use this algorithm on a defined number of sub-data-sets n_s inspired by the functioning of **Random Forest** machine learning algorithms. Let n_c be the number of contextual attributes, then $n_s = \sqrt{n_c} * 10$. To do this, we must first decide how to create a subset from a data set. The objective is to have several sub-data-sets, i.e., with only a part of the points, in order to evaluate all the points in slightly different configurations to finally determine that the point is an outlier if it is in the majority of cases.

The subsets should respect several characteristics if they want to use them for this outlier detection method. First, all points must be represented, i.e., a point must be present in at least one subset, so that all points can be evaluated at least once. This feature is most likely to be respected when there are numerous subsets to be generated, however we choose to force it algorithmically in order to ensure it in all circumstances. Second, all subsets must have a distribution of behavioral values similar to the original data set in order to have a representation close to the original data set.

To respect the first condition, an algorithm is used to create the minimum number of subsets that allows the presence of all points at least once. This algorithm only needs the percentage of points present on each subset, p_r . It must not be too low in order to have subsets close from the original set to have reliable decisions, e.g. 70%. Thus, this minimum number of subsets created is equal to $\lfloor \frac{1}{p_r} \rfloor$. Then the rest of the subsets, i.e. $n_s - \lfloor \frac{1}{p_r} \rfloor$, are created randomly. All these subsets always respect the second condition.

Still for the same purpose as the subsets, only a part of the contextual attributes are used. Half of the attributes are kept, and each subset uses a different subgroup of attributes. But before that, a slight filtering of the contextual attributes is performed in order to remove those that are considered useless in establishing the behavioral value because they will influence the creation of neighborhoods, deviating them from reality and making the decision less reliable. Thus, we decide to remove from the process all attributes that are not correctly correlated with at least one of the other attributes. This number must be low enough so that only noisy attributes are removed, e.g. ± 0.1 . Finally, the correlation matrix is calculated using the **MIC** method, which is more accurate than the **Pearson** and **Spearman** methods which are commonly used.

3 Experimentation

For the experimentation, we study two kinds of data sets. The first is synthetic one and the second is real data sets proposed by the community. Computations have been performed on the supercomputer facilities of the *Mésocentre de calcul de Franche-Comté*. The details of the results on the synthetic data sets and the real data sets used are present in a GitHub directory¹.

¹ <https://github.com/JessyColonval/Makoto>.

During these experiments, Makoto will have 70% of the points of the original data set for each subset and the filtering of contextual attributes is equal to 0.1 of correlation. As comparison, 4 state-of-the-art methods are used: IQR [15], LOF [4], SVM [2] and Isolation Forest [12]. They are chosen because they are often used for outlier detection and have different approaches. However, there are other methods that are more recent as PyOD [17]. But it doesn't use the same logic of detection, in fact they assume that the training subset doesn't contain outliers and the detection is performed only on the test subset.

For validation of certain results, we will use the same 6 *machine learning* (ML) algorithms at three different ways (see Sects. 3.1 and 3.2). They come from the Python library *scikit-learn*² and represent different approaches in order to avoid being biased by the results that only one type of *machine learning* would give. These algorithms are: *SVC*, *KNeighborsClassifier*, *RandomForestClassifier* (RF), *ExtraTreesClassifier* (ET), *GradientBoostingClassifier* and *LogisticRegression* (LR). All meta-parameters have the default values, except for ET/RF where `n_estimator` is 200 and LR where `max_iter` is 1000.

3.1 On Synthetic Data Sets

These data sets are created with the same algorithm that was designed to generate the *Madelon* data set [8]. They all have only one behavioral attribute whose values are homogeneously distributed. And all contextual attributes are useful for establishing the behavioral value, i.e., there is no noise. The original generation does not contain any outliers, they are added later and are therefore known for further experimentation.

Table 3. Characteristics of synthetic data sets.

samples	attributes	classes
250	10 ... 50, 25	2 ... 5
500	10 ... 70, 25, 75	2 ... 5
1000	10 ... 100, 25, 75	2 ... 7
2500	10 ... 100, 25, 75	2 ... 8
5000	10 ... 100, 25, 75, 125	2 ... 11
10000, 25000	10, 50, 75, 100, 250	2, 3, 5, 7, 9, 11

The outliers are created using the 6 ML algorithms described earlier. The idea is to detect the most useful points for predictions in order to change their behavioral value. Thus, the chances of having strong outliers which are harmful to the predictions is increased. While a purely random method wouldn't prevent the selection of points that are not very useful for the prediction. For this purpose, the data sets are randomly divided into a training subset (70%) and a test

² <https://scikit-learn.org/stable/index.html>.

subset (30%). These 6 ML algorithms are trained with the training subset, then we see if they are able to correctly predict the points contained in the test subset. These actions are repeated 10 times, and count for each point the number of times they have been correctly predicted. Finally, the points that will change in behavioral value are all those with the lowest scores, and stop once the desired number of outliers is created. In cases where there are more than 2 behavioral values, the new one is chosen randomly.

Thus, with these methods, we create 406 data sets having the characteristics presented in Table 3 and having 5% outliers. These data sets have distinct characteristics to ensure that there is no significant difference in behavior based on the numbers of points, of attributes or of behavioral values. For those with a number of points between 250 and 5 000, the numbers of attributes increase with a step of 10 (except for 25 and 75) and the numbers of behavioral attributes increase with a step of 1. Data sets with 10 000 and 25 000 points are less exhaustive in their number of attributes and behavioral values due to computational time concerns during their generation and outlier detection.

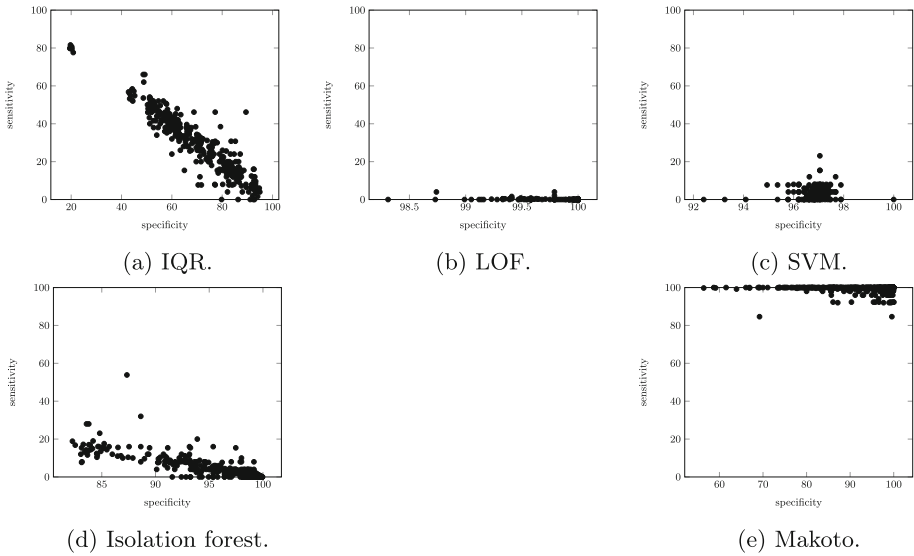


Fig. 4. Specificity and sensitivity of the 5 methods.

Since the generated outliers are known, then it's possible to use the sensitivity and specificity scores to determine which one is the best limit. As a reminder, the calculation of the sensitivity and specificity is based on the numbers of true positive (TP), true negative (TN), false positive (FP) and false negative (FN). The sensitivity measures the ability of a test to give a positive result when a hypothesis is verified, it's calculated by: $\frac{TP}{TP+FN}$. In this case, it is interpreted by the percentage of correctly detected outliers. And the specificity measures the

ability of a test to give a negative result when the hypothesis is not verified, it's calculated by: $\frac{TN}{TN+FP}$. In this case, it is interpreted by the percentage of undetected points that are actually healthy. These metrics are interpreted together, a good method must give two scores as close to 100% as possible.

Figure 4 compares the sensitivity and specificity scores obtained with the 5 methods on synthetic data sets. A good method should give sensitivity and specificity scores close to 100%, so visually the points should be at the top right of the plot. **Makoto** is clearly the best method, in most cases it is able to detect numerous outliers ($\geq 80\%$) while keeping well the healthy points ($\geq 80\%$). However, there are some cases where too many healthy points are removed (60–79%) but this is still acceptable compared to the other solutions. While the **LOF**, **SVM** and **Isolation Forest** keep the healthy points well but remove very few outliers ($\leq 20\%$), which makes these methods not very useful. **IQR** is the worst because these solutions are either useless, i.e. few outliers detected with healthy points kept, or harmful, i.e. too many healthy points deleted.

3.2 Real Data Sets

In order to confirm the efficiency of **Makoto**, the comparison continues with 17 real data sets whose contextual attributes have continuous values. They all come from the UCI Machine Learning Repository³ database, except Mammography, which comes from the BCSC (Breast Cancer Surveillance Consortium) site⁴. The characteristics of these data sets cover several combinations (Table 4), in order to verify that the behavior of a method doesn't change according to these.

Table 4. Characteristics of real data sets.

Data sets	Points	Attributes	Behavioral values	Data sets	Points	Attributes	Behavioral values
annthyroid	7 200	21	3	multiple_ features	2 000	619	10
breastW	683	9	2	musk	6 598	166	2
cardio	2 126	21	10	parkinson	756	752	2
glass	214	9	6	pendigits	10 992	16	10
ionosphere	351	33	2	satimage2	6 435	36	6
isolet	7 797	617	26	shuttle	58 000	9	7
letter_ recognition	20 000	16	26	wine	178	13	3
mammo- graphy	11 183	6	2	wine- quality	1 599	11	6

The metrics used to compare the methods studied are different from those used for the synthetic data sets. This is because these data sets come from the

³ <https://archive.ics.uci.edu/ml/index.php>.

⁴ <https://www.bscs-research.org/>.

Table 5. Comparison of cross-validation, false positive and class keeping scores of IQR, LOF, SVM, Isolation Forest and Makoto methods on real data sets.

Data sets	IQR			LOF			SVM			IF			Makoto		
	CV	FP	CK	CV	FP	CK	CV	FP	CK	CV	FP	CK	CV	FP	CK
amthyroid	Error		✗	98.12%	94.72%	✓	97.12%	97.98%	✓	97.68%	94.22%	✓	Error		✗
	53.60%			± 0.22	± 0.12		± 0.22	± 0.06		± 0.19	± 0.07		7.42%		
breastW	97.13%	91.66%	✓	96.73%	95.08%	✓	97.09%	99.67%	✓	98.61%	70.86%	✓	98.8%	6.22%	✓
	± 1.12	± 0.36		± 1.19	± 0.25		± 0.91	± 0.0		± 0.29	± 0.90		± 0.72	± 0.94	
cardio	81.4%	60.35%	✗	82.45%	71.67%	✓	82.06%	85.30%	✓	81.23%	72.71%	✓	90.06%	25.53%	✓
	± 1.94	± 0.19		± 1.04	± 1.14		± 1.5	± 0.3		± 1.26	± 0.35		± 0.97	± 0.36	
glass	73.46%	28.12%	✗	73.49%	40.98%	✓	72.28%	49.17%	✓	74.75%	48.7%	✓	82.72%	21.99%	✓
	± 4.87	± 1.09		± 5.15	± 1.25		± 4.66	± 1.76		± 5.05	± 1.71		± 4.92	± 0.98	
ionosphere	95.12%	59.47%	✓	94.23%	55.61%	✓	91.21%	91.67%	✓	90.81%	86.54%	✓	97.0%	23.76%	✓
	± 2.58	± 0.43		± 2.00	± 0.72		± 1.88	± 0.0		± 1.97	± 0.52		± 1.75	± 1.24	
isolet	Error		✗	93.6%	75.0%	✓	93.53%	90.55%	✓	93.1%	93.4%	✓	95.23%	39.48%	✓
	96.63%			± 0.5	± 0.0		± 0.44	± 0.25		± 0.49	± 0.14		± 0.34	± 0.61	
letter recognition	90.04%	68.37%	✓	91.48%	94.52%	✓	91.64%	92.75%	✓	91.48%	88.15%	✓	95.03%	49.01%	✓
	± 0.38	± 0.23		± 0.25	± 0.23		± 0.32	± 0.12		± 0.27	± 0.09		± 0.26	± 0.18	
mammography	99.41%	94.33%	✓	98.66%	90.98%	✓	98.69%	91.23%	✓	99.26%	91.06%	✓	99.66%	7.06%	✓
	± 0.04	± 0.03		± 0.09	± 0.19		± 0.12	± 0.16		± 0.03	± 0.2		± 0.09	± 0.13	
multiple features	Error		✗	98.33%	3.33%	✓	98.29%	90.14%	✓	99.0%	91.15%	✓	99.1%	30.36%	✓
	97.65%			± 0.47	± 10.54		± 0.45	± 0.54		± 0.45	± 0.2		± 0.4	± 2.22	
musk	91.38%	59.22%	✓	96.54%	93.17%	✓	96.45%	99.31%	✓	96.39%	98.78%	✓	98.15%	14.4%	✓
	± 4.61%	± 0.37		± 0.35	± 0.41		± 0.27	± 0.25		± 0.34	± 0.06		± 0.28	± 0.45	
parkinson	Error		✗	86.67%	83.26%	✓	86.45%	87.03%	✓	87.04%	86.19%	✓	87.0%	22.6%	✓
	100%			± 1.67	± 0.79		± 1.9	± 1.08		± 1.78	± 1.14		± 1.74	± 1.69	
pendigits	98.58%	85.18%	✓	98.67%	76.34%	✓	98.52%	94.99%	✓	98.87%	50.45%	✗	98.91%	44.56%	✓
	± 0.2	± 0.17		± 0.2	± 0.45		± 0.16	± 0.16		± 0.23	± 0.17		± 0.16	± 0.57	
satimage2	89.19%	96.65%	✓	89.89%	76.35%	✓	89.28%	97.56%	✓	88.48%	58.24%	✓	95.25%	10.93%	✓
	± 0.45	± 0.16		± 0.52	± 0.49		± 0.52	± 0.06		± 0.75	± 4.04		± 0.5	± 0.27	
shuttle	99.83%	81.91%	✗	Error		✗	Error		✓	99.78%	60.8%	✗	99.43%	92.26%	✓
	± 0.05	± 0.0		19.00%			2.79%			± 0.03	± 0.04		± 0.04	± 0.13	
wine	98.03%	90.2%	✓	98.11%	70.67%	✓	98.3%	88.12%	✓	95.96%	95.83%	✓	98.65%	53.33%	✓
	± 1.66	± 0.0		± 1.54	± 1.41		± 1.45	± 1.01		± 2.81	± 0.0		± 1.22	± 2.64	
winequality	64.42%	51.85%	✓	63.66%	41.31%	✓	63.48%	35.72%	✓	63.99%	46.05%	✓	Error		✗
	± 2.17	± 0.61		± 1.69	± 1.02		± 1.85	± 1.16		± 1.95	± 0.57		27.39%		
	25.20%			2.94%			2.88%			16.45%					

real world, and we're looking for only the outliers contains in these data sets. Thus, it is not possible to measure the ability to detect outliers because we don't know which points are real outliers. To circumvent this problem, the comparison is done using the cross-validation and false positivity scores.

The cross-validation score will measure the performance of the 6 ML algorithms described above after removing outliers. The idea is that the presence of

outliers reduces the performance of these algorithms. Thus, by measuring this score for each method, it's possible to determine which removal was the most beneficial for these ML algorithms, therefore which method is the most efficient. To obtain this score, the data set is randomly separate into two subsets: a training subset (70%) and a test subset (30%) which keeps the proportion of the behavioral values of the original set. Then, each of the ML algorithms is trained with the training subset, and we look at their ability to correctly predict the behavioral value of the points contained in the test subset. This operation gives a score as a percentage, the closer it is to 100% the more the algorithm is able to correctly predict this point and the more effective the training, the better the quality of the data. This process is repeated 10 times, i.e. with 10 different separation of the subsets, then averaged in order to have more reliable results.

The false positivity score will measure the relevance of the points that have been removed. It is assumed that a true outlier is a point that will be wrongly predicted, and a false outlier is a point that will be correctly predicted, so that it could have been kept in the training set. However, this way of calculation is imperfect. If too many outliers are removed, then the data set is denatured and those predictions made are not relevant. It works better with few outliers and should be read in addition to the other scores. This score is calculated with the same logic as the cross-validation score. Except that the training set are the healthy points and the test set are the outliers. For the same reasons, this process is repeated 10 times, but only changing the random seeds of the ML algorithms and keeping the same subsets.

The Table 5 gives all the results obtained from the 5 methods on the 17 real data sets. The **CV** columns give, respectively, the averages of the cross-validation scores and the standard deviations, and then the percentage of outliers detected among all points. The **FP** columns give the mean of the false positivity scores and the standard deviations. The **CK** columns indicate if at least one behavioral value was completely considered as an outlier. The cells labeled **Error** appears when a method have denatured the data set too much to be able to compute a part of the metrics. The best methods are those with a high cross-validation score, a low false positivity score, no missing behavioral values and a reasonable number of detected outliers ($\leq 25\%$). Thus, **Makoto** outperforms the other methods on 15 of the 17 data sets.

4 Conclusion

This paper presented a complete outlier detection method, Makoto, and showed its effectiveness by comparing it with 4 other methods in the literature on synthetic and real data sets. However, some of Makoto's results can be improved by changing some of these meta-parameters or the way they are calculated. A possible extension would be to establish a better methodology to choose them. Moreover, this method is similar to a machine learning algorithm and could be used to predict behavioral values.

Acknowledgement. Work supported by the French National Research Agency (contract ANR-18-CE25-0013) and by the EIPHI Graduate School (contract ANR-17-EURE-0002)

References

1. Aggarwal, C.C.: Outlier Analysis. Springer International Publishing, Cham (2015). <https://doi.org/10.1007/978-3-319-47578-3>
2. Amer, M., Goldstein, M., Abdennadher, S.: Enhancing one-class support vector machines for unsupervised anomaly detection. In: Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description, ODD 2013, pp. 8–15. Association for Computing Machinery, New York (Aug 2013). <https://doi.org/10.1145/2500853.2500857>
3. Baoli, L., Qin, L., Shiwen, Y.: An adaptive k -nearest neighbor text categorization strategy. ACM Trans. Asian Lang. Inform. Process. **3**(4), 215–226 (2004). <https://doi.org/10.1145/1039621.1039623>
4. Breunig, M., Kriegel, H.P., Ng, R., Sander, J.: LOF: identifying density-based local outliers. In: ACM Sigmod Record, vol. 29, pp. 93–104 (Jun 2000). <https://doi.org/10.1145/342009.335388>
5. Chehreghani, M.H.: K -nearest neighbor search and outlier detection via minimax distances. In: Proceedings of the 2016 SIAM International Conference on Data Mining, p. 9. Society for Industrial and Applied Mathematics (2016). <https://doi.org/10.1137/1.9781611974348.46>
6. Dietterich, T., Wettschereck, D., Wettschereck, D., Dietterich, T.G.: Locally adaptive nearest neighbor algorithms. In: Advances in Neural Information Processing Systems 6, pp. 184–191. Morgan Kaufmann (1994)
7. Epanechnikov, V.A.: Non-parametric estimation of a multivariate probability density. Theory Probabil. Appli. **14**(1), 153–158 (1969). <https://doi.org/10.1137/1114019>
8. Guyon, I., Gunn, S., Ben-Hur, A., Dror, G.: Design and Analysis of the NIPS2003 Challenge, vol. 207, pp. 237–263 (Nov 2008). https://doi.org/10.1007/978-3-540-35488-8_10
9. Hechenbichler, K., Schliep, K.: Weighted k -Nearest-Neighbor Techniques and Ordinal Classification. discussion paper 399 (Jan 2004)
10. Keller, F., Muller, E., Bohm, K.: HiCS: high contrast subspaces for density-based outlier ranking. In: 2012 IEEE 28th International Conference on Data Engineering, pp. 1037–1048 (Apr 2012). <https://doi.org/10.1109/ICDE.2012.88>
11. Kou, Y., Lu, C.T., Chen, D.: Spatial weighted outlier detection. In: Proceedings of the 2006 SIAM International Conference on Data Mining, p. 5. Proceedings, Society for Industrial and Applied Mathematics (Apr 2006). <https://doi.org/10.1137/1.9781611972764.71>
12. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation-Based Anomaly Detection. ACM Trans. Knowl. Dis. Data **6**(1), 3:1–3:39 (2012). <https://doi.org/10.1145/2133360.2133363>
13. Lu, C., Chen, D., Kou, Y.: Algorithms for spatial outlier detection. In: Third IEEE International Conference on Data Mining, pp. 597–600 (Nov 2003). <https://doi.org/10.1109/ICDM.2003.1250986>

14. Thung, F., Wang, S., Lo, D., Jiang, L.: An empirical study of bugs in machine learning systems. In: 2012 IEEE 23rd International Symposium on Software Reliability Engineering, pp. 271–280 (Nov 2012). <https://doi.org/10.1109/ISSRE.2012.22>
15. Whaley, D.L.: *The Interquartile Range: Theory and Estimation* (2005)
16. Zhang, S., Wan, J.: Weight-based method for inside outlier detection. *Optik* **154**, 145–156 (2018). <https://doi.org/10.1016/j.ijleo.2017.09.116>
17. Zhao, Y., Nasrullah, Z., Li, Z.: Pyod: a python toolbox for scalable outlier detection. *J. Mach. Learn. Res.* **20**(96), 1–7 (2019). <http://jmlr.org/papers/v20/19-011.html>