Anomalies detection in Hypothyroidism dataset

Diego Pesce
Department of Physics
Università degli studi di Milano Bicocca
Milan, Italy
d.pesce1@campus.unimib.it

Claudio Colturi
Department of Physics
Università degli studi di Milano Bicocca
Milan, Italy
c.colturi@campus.unimib.it

***Abstract*— In this paper we will carry out an anomalies detection analysis on the Hypothyroidism dataset. We will go through several methods suited to tackle the problem of mixed variable types.**

I. Introduction

Anomalies are identified intuitively as those observations that do not fit the behavior of the majority of the points. Detecting outlying points is crucial in many applications, since the generating process of an anomaly may provide profound insight on the situation under study. In the healthcare realm, an anomaly can represent an uncommon patient's condition or an unusual response to a given drug. It is self-evident that an automatic, rapid and strongly reliable detection of these situations can critically ease doctors work and in turn improve patients lifestyles. It is important to highlight that anomalies are not always significant in the context under analysis, indeed it is very likely that outliers are due to errors in the data entry process and therefore these observations are void of basically any meaning. Other applications of anomaly detection can be found in many other areas like physics, biology or economic.
A multitude of algorithms for anomalies detection are available both for continuous and categorical attributes, as well for mixed attributes types. In this paper we will present some of them, suited to tackle mixed variables types, in detail applied to hyperthyroidism dataset.

II. Dataset exploration

A. Structure

The Hypothyroidism dataset consists of

- N = 7200 observation
- M = 21 features of mixed types, more precisely:
  - 6 *continuous* attributes
  - 15 *binary* attributes

These data refers to geometrical measurements relative to the thyroidal apparatus.
Due to the different variable types the standard anomaly detection strategies must be reviewed and treated carefully.

B. Variable analysis

In order to study the attributes of the dataset we started by looking at the variable distributions and calculating some basic statistics.
First of all we checked the presence of missing values in the data but none was found. In second place we plotted the features marginal distributions in order to get an idea on how to preprocess the dataset.

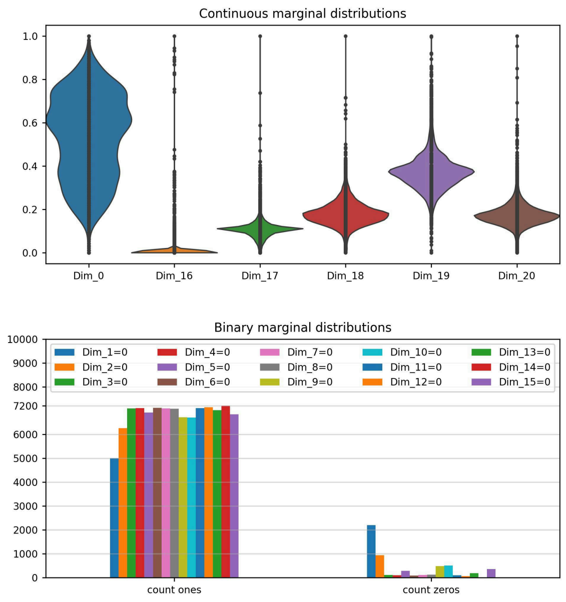

Figure 1:
(*up*): Marginal distribution for the 6 continuous attributes in the given dataset.
(*down*): Marginal distribution for the 15 binary attributes in the given dataset.

As it is evident in Figure 1 the continuous variables are already scaled in the range $[0, 1]$ therefore, since no scale problems were present, we decided to not further standardize these attributes.
We notice that the attribute "Dim_16" is strongly shrunk toward zero, we might expect this attribute to dominate in the anomaly detection process, but in facts only the interplay among features and the data interpretation (that, in this case, is missing) can tell wether an observation is an anom-

aly or not.
As far as binary attributes are concerned we noticed that some features attain, for the most of the observation, the same value (namely 1, in the nominal variable sense). Exactly as before we cannot, in principle, tell if these point will be detected as anomalies or not.
Striking is the case of "Dim_14=0" where just one sample has the value 0 for this attribute, in this case the column looks redundant hence we decided to drop it. Keeping this feature would result in increasing the similarity among data, something not beneficial for anomaly detection tasks.
No other preprocessing to the data have been applied.

III. Distance metric

Since we are dealing with mixed variable types, classical metrics (like euclidean, minkowski, jaccard etc...) cannot be applied. A common metric used in these situations is the Gower's metric [1].
This metric has been developed to deal with mixed variables types and the working principle is the following:

$$D_{\text{Gower}}(x_1, x_2) = \left(\frac{1}{p}\right) \sum_{\text{j}=1}^{p} d_j(x_1, x_2) \tag{1}$$

where $d_j(x_1, x_2)$ is the Gower's dissimilarity function computed separately for each descriptor j:

- *quantitative* descriptors: range-normalized Manhattan distance:

$$d_j(x_1, x_2) := \frac{|x_1^j - x_2^j|}{R_j} \tag{2}$$

- *qualitative* descriptors: dice distance:

$$d_j(x_1, x_2) := \begin{cases} 0 \text{ if } x_1^j = x_2^j \\ 1 \text{ otherwise} \end{cases} \tag{3}$$

In the following picture we show the histogram pairwise distances between the data attributes using Gower's metric.

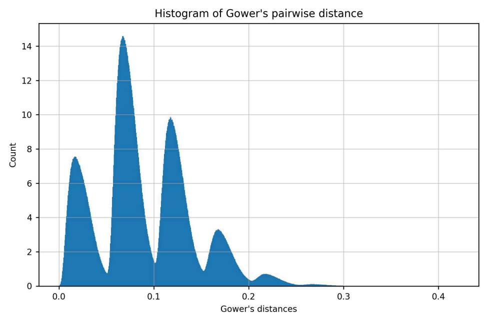

Figure 2: Histogram of Gower's pairwise distances between the samples in the dataset.

As we can see from Figure 2, the distances' distribution clearly presents different peaks. This leads to the possibility of finding a cluster structure into the given dataset as shown in the appendix *peaks in the distance histogram*.

IV. Anomaly detection: Algorithms

In this section we will briefly describe the algorithms used and their motivation.

A. *Isolation Forest*

The *Isolation Forest* algorithm [2] is a recursive algorithm that aims to isolate data samples in order to detect outlying points. Since the algorithm isolates the points recursively it can be represented by a tree-structured method. In this representation, the depth of the tree required to isolate a sample is equivalent to the difficulty in isolating it.
The empirical observation, reported in the original paper [2], is that the path length needed to isolate a normal point is much greater than the one needed for an anomaly.
The partitioning in the isolation process is performed by randomly selecting a feature and a splitting value for each node of the tree. Multiple trees are grown for each observation in order to obtain a more robust result. Finally, for every sample, the resulting path lengths of the trees are averaged thus obtaining an initial outlying level. It is important to remark that this average converges with the increasing number of trees, hence obtaining a meaningful estimation of the average path length.
Isolation forest provides also an anomaly score for each data sample, calculated as

$$s(x) := 2^{-\frac{\overline{h(x)}}{c(n)}} \tag{4}$$

where:

- $\overline{h(x)}$ is the average tree depth for the observation x;
- $c(n)$ is a normalization factor equal to the average depth in an unsuccessful search in Binary Search Tree with size *n*.

Let's see how to interpret this score:

- if $\overline{h(x)} \approx c(n)$ we are in the most uncertain case, indeed $s(x) \approx 0.5$
- if $\overline{h(x)} \ll c(n)$ the observation is most likely an outlying data, consequently $s(x) \approx 1$
- if $\overline{h(x)} \approx n - 1 \gg c(n)$, namely the tree is the the longest achievable, it means that the points is reasonably marked as inlier, hence $s(x) \approx 0$

This interpretation of the scores set a reasonable default cutoff for anomalies label to 0.5.
We decided to implement this method, with a few parameters, in order to get a first estimation of the percentage of anomalies in the dataset. Our strategy is to use this estimation to get an idea one the quality of the clustering solution

we will explain just below . This algorithm is implemented by *scikit-learn* [3]. It is important to notice that Isolation Forest can be run also on categorical attributes without further encondings.

B. Cluster based detection: DBSCAN and HDBSCAN

As already mentioned, the peaks in Figure 2 led us to hypothesize that an internal clustered structure is present in our data. We decided then to apply a clustering method capable of detecting anomalies. The main algorithm that suits this purpose are DBSCAN [4] and its evolutions.

1) *DBSCAN:* DBSCAN is a density-based clustering algorithm that, in the most common implementation, uses a center-based approach to define the density of the clusters. This technique allows the algorithm to classify a point as being a core, a border or a noise point.

Thanks to *scikit-learn* library [5] this algorithm can be easily implemented using the previously calculated distance matrix by setting the parameter *metric* = "precomputed", in turn it can be used with mixed variable types.

DBSCAN is based on two parameters, to which this algorithm is extremely sensitive:

- *MinPts*: minimum number of neighbors of a point in order to classify it as a core point;
- *Eps*: the distance that defines the neighbor of a point;

This algorithm labels a data point *p* in three ways:

- *p* is a *core* point if in the *Eps*-neighbor there are at least *MinPts* samples.
- *p* is a *border* point if it is not a core point but it belongs to the *Eps*-neighbor of a core point.
- *p* is a *noise* point if it is neither a core or a border point.

As a second step all the core points connected by an edge shorter than *Eps* are collected together in the same cluster. Finally the border points are merged in the cluster of the closest core point.

2) *HDBSCAN:* As mentioned above DBSCAN is very sensitive to its parameters, moreover it is not suited to varying densities in the data due to the fixed notion of "density" once specified the parameters of the algorithm. For these reasons we decided to implement a novel algorithm called HDBSCAN [6].

We will not delve into the finer details of this algorithm, that is fully presented in the linked paper. The main steps are briefly listed:

- build a complete graph, with points as nodes and predefined pair-wise distances as edges weights;
- build over this graph the minimum spanning tree;
- recursively cut edges with higher weights by labelling, at each step, connected components as belonging to the same cluster.

A new stability measure is then defined in order to stop the algorithm when a reliable cluster solution is reached.

By defining $\lambda_{\text{birth}}(C)$ as the reciprocal of the weight relative to the removed edge that created the cluster *C* and λ_p as the reciprocal of the distance that cuts out the point *p* from the cluster (or that splits the cluster), HDBSCAN measures the stability of a cluster C as:

$$\text{stability}(C) = \sum_{p \in C} \big(\lambda_p - \lambda_{\text{birth}}(C)\big) \tag{5}$$

When a split is proposed, if the sum of the stabilities of the two children cluster is lower than the one of the original cluster then the split is not performed. HDBSCAN marks the remaining isolated nodes as noise points exactly as DBSCAN. In this algorithm the only parameter to set is the *MinPts* parameter, analogous to DBSCAN and used in the definition of the initial distance.

Scikit-library offers an easy implementation of HDBSCAN algorithm [7].

C. K-Nearest-Neighbor (KNN)

K-Nearest-Neighbors approaches aim to detect anomalies by processing, for each point *p*, the K closest points to *p*.

At this stage we will have found a consistent clustering solution. We decided to detect outlying samples using a KNN approach by injecting some information from the clustering solution.

The procedure works as follows:

- for each point the distance from the K-nearest neighbors has been calculated;
- the median of these values is taken and compared among the points.

The choice of the median is motivated by its higher robustness with respect to the mean when in the K selected neighbors some outliers are included.

On the one hand if less than $\frac{K}{2}$ outliers are included in the neighbors is less likely to mark a normal point as anomaly. On the other hand outlying data will be the ones farther from normal data, but it may happen that few outlying points are relatively close one to the other (not close enough to form a cluster), this would affect the mean by decreasing it, something the median does not suffer from.

The clustering previously obtained helps us in the choice of the number of neighbors K. The selected K should be a little lower than twice the size of the smallest cluster found applying HDBSCAN. By setting K in this way it is reasonable to think that, taken a point *p*, we have:

- if *p* is an isolated point the K-neighborhood will most likely contain points from at least one of the clusters, hence from the above-mentioned reasons the median distance will be higher;

- if *p* is an inner point of a cluster the K-neighborhood will contain mostly cluster points, resulting in a lower median distance.
- if *p* is a border point of a given cluster its K-neighborhood will be mostly composed by cluster points.

As last step we will analyze the obtained values, for instance by plotting the histogram, and set a cutting point to divide normal points from anomalies.

D. ***Local Outlier Factor (LOF)***

The *Local Outlier Factor* approach is a density-based outlier detection algorithm. The main idea of this method is to assign a score for each data point *q* based on the local reachability density *lrd(q)* compared with its k-nearest neighbors local reachability densities. More in detail, we can first define the *reachability distance* between an object *q* and an object *p* as:

$$reach - dist(p, q) = \max\{d_k^q, d(p, q)\} \tag{6}$$

then, with (6) we compute the *local reachability density* as

$$lrd(p) = \frac{MinPts}{\sum_{q \in \text{ k-NN}} \text{reach} - \text{dist}_{MinPts}(p, q)} \tag{7}$$

Finally, we obtain the LOF as:

$$LOF(p) = \frac{1}{MinPts} \sum_{q} \frac{lrd(q)}{lrd(p)} \tag{8}$$

In order to implement this method, we used the *scikit-klearn* library [8]. As a parameter we will set the number of neighbors and the contamination (percentage of expected anomalies), we will use the results from the previous methods to set these values. If the parameter contamination is given, the algorithm automatically defines a threshold score used to classify the precise amount of points as outliers.

E. ***One-Class SVM***

One Class SVM (Support Vector Machine) is an anomaly detection algorithm particularly used when dealing with unsupervised problems. It differs from classical SVM because it does not involve target labels during the training process, but learns the boundary for the normal data points and identifies the data outside this boundary as anomalies. Indeed, the algorithm works by learning a decision function for a single class (the normal data) by building a hyperplane that separates the normal data points from the origin in a transformed features space obtained by a kernel function.
More in detail, this is done by using an embedding function ϕ that maps each data objects x into the transformed space, where the instances can be separated using a linear hyperplane defined as

$$\langle w, \phi(x) \rangle = b \tag{9}$$

where b is the bias term and w the normal vector to the hyperplane. In order to separate the normal instances from the anomalies, the aim is to maximize the distance of the separating plane from the origin and thus maximizing

$$\frac{b}{\|w\|} \tag{10}$$

Therefore, as the article [9] shows, the solution is given by the following quadratic programming problem:

$$\min_{w,b,\xi} \frac{1}{2} \; \|w\|^2 - b + \frac{1}{nv} \sum_{i=1}^{n} \xi_i \tag{11}$$

under the constraints

$$\langle w, \phi(x) \rangle \geq b - \xi_i,$$
$$\xi_i \geq 0,$$
$$i = 1, ..., n.$$

The parameter v in (11) is crucial and describes the fraction of outliers in the data.
In our work, we implemented the algorithm in the *scikit-learn* framework [10] and encoded the binary attributes with one hot encoding.

V. Anomaly detection: Results

As explained in the previous section, we started by applying the Isolation Forest algorithm in order to get a first estimate of the percentage of anomalies.

A. *Isolation Forest*

Using the Isolation Forest algorithm with default parameters, beside the number of iTrees that was set to 2000, the percentage of anomalies detected in the dataset is equal to $3, 74\%$. We did not apply further validation than the tSNE plot.

B. *Cluster-based Anomaly Detection*

For what concerns the clustering solutions, we ran DBSCAN and HDBSCAN with the parameter *MinPts* varying in the range $[2, 50]$ with step 2.
For what concerns DBSCAN, the *Eps* parameter is obtained by identifying the knee on the *MinPts*-nearest neighbor distance plot thanks to the *kneed* library [11], which gives the point of maximum curvature of a line.
Once obtained the clustering solutions for the two algorithms, in order to validate our solutions, we performed the *silhouette score* [12] between DBSCAN and HDBSCAN for each value of *MinPts*.

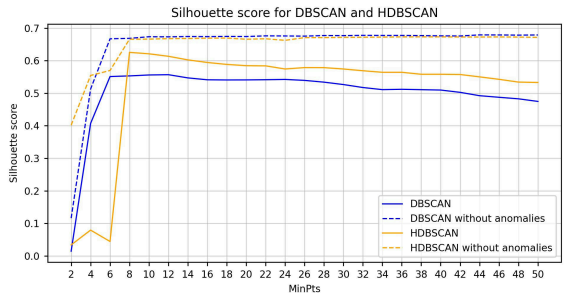

Figure 3: Silhouette score for DBSCAN and HDBSCAN algorithms for the range selected for *MinPts*; the dashed line curve presents the score obtained by removing the anomalies from the clustering solutions.

From the plots showed in Figure 3 we can see that the two algorithms start to get a meaningful score from $MinPts = 6$ for DBSCAN and $MinPts = 8$ for HDBSCAN. The Silhouette score tends to decrease if we don't remove the anomalies from the clustering solutions: this is due to the increasing number of detected anomalies along with the *MinPts* value. With only this initial analysis we are not able to discriminate the clustering solutions and to take the most suitable for our purpose, therefore we decided to analyze the percentage of anomalies detected by each clustering solution and to make a comparison with the one found with the Isolation Forest, which, as described above, uses a completely different approach.

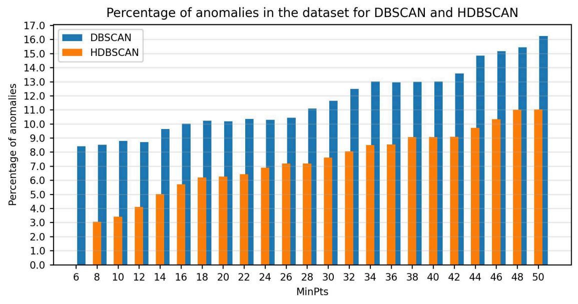

Figure 4: Percentage of anomalies for each clustering solution for DBSCAN and HDBSCAN with respect to the parameter *MinPts*. The *MinPts* values under 6-8 have been removed for DBSCAN-HDBSCAN due to previous considerations.

The barplot in Figure 4 shows that the DBSCAN algorithm, with an *Eps* parameter chosen automatically with the knee method, tends to overestimate the percentage of anomalies with respect to the result obtained with Isolation Forest. The HDBSCAN, on the other hand, presents some results under and over that estimation, that in general seems to better agree with Isolation Forest.

Since the HDBSCAN is a more robust version of DBSCAN that already performs a search over potential candidates of *Eps*, we decided to take on the analysis only considering HDBSCAN.

We selected the best clustering solutions for HDBSCAN by looking at the cluster sizes, intra-cluster and inter-cluster distances for each detected cluster in various solutions, we also made use of graphical representation obtained via *tSNE* [13]. The idea is to partially validate the clusters by weighting clusters' sizes with the intra-cluster distance. We concluded that the best solution is given by running HDBSCAN with 14 *MinPts*. The percentage of anomalies found in this case is 5.01%.

C. K-Nearest-Neighbors

For what concerns the KNN algorithm, as said in the previous section we set the K parameter based on the clustering solution obtained with HDBSCAN. We decided to take $K = 20$. For this method, we used the Gower matrix to find the K-neighbors. In the next figure we show the standardized distances obtained with the algorithm.

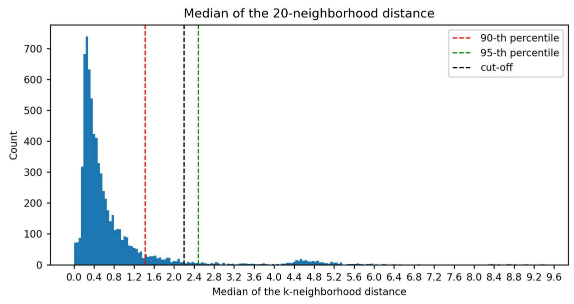

Figure 5: Histogram of the median distances of the K-neighborhood.

In order to find the cut-off point showed in Figure 5, we sorted the distances and looked for knee point(s).

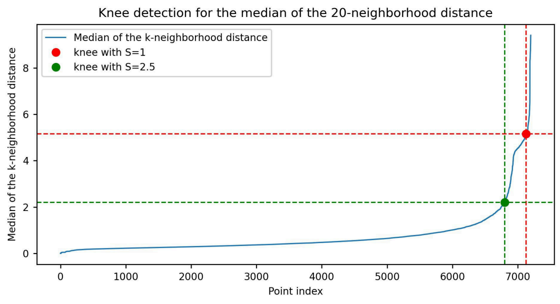

Figure 6: Knee method for the K-neighborhood algorithm.

We can see in Figure 6 that there are two knee points in the graph. These two knees values were obtained by tuning the *sensitivity S* of *KneeLocator* [11]. The cutoff in Figure 5 corresponds to the first knee (green) detecting a percentage of 5.56% of anomalies.

D. Local Outlier Factor

For the LOF algorithm, we used the same number of neighbors as for *MinPts* parameter for the HDBSCAN and set the parameter *contamination* (the expected percentage of anomalies) equal to 4%. This value was set by looking at the *tSNE* plot, by increasing it we noticed that some evident cluster present detected anomalies in their interior. By observing graphically the cut we deemed it reasonable on the tail of the distribution (Figure 7). This method has been implemented by *scikit-learn* [8] via the Gower's matrix.

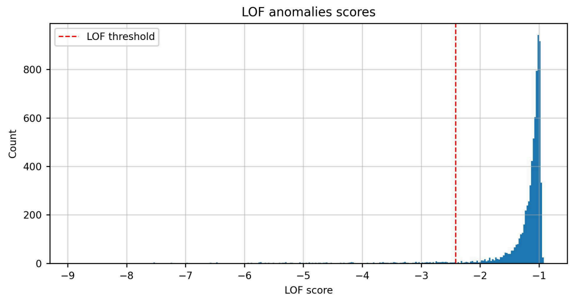

Figure 7: Histogram of the LOF score distribution obtained by setting the number of neighbors = 14 and contamination = 4%. The red dashed line representing the cut is at LOF score = −2.236

E. One-Class SVM

For the *One-Class SVM* algorithm, we used a gaussian kernel and set the parameter $\nu = 0.05$ (this parameter has the same scope of *contamination* for the previous algorithms). In this case, we applied *One Hot Encoding* to the binary attributes before using the algorithm. This encoding allows for a treatment of the binary variable as if they were belonging to an euclidean space, hence the algorithm can be correctly applied. The only partial validation we applied was by looking at tSNE plot.

F. Final results

In order to conclude our analysis, we decided to assign a score to outlying data. We calculated this score by counting, for each observation *p*, how many methods marked *p* as an anomaly and finally dividing by the number of algorithms applied. Since we implemented 5 methods, we obtained 6 outlying scores in the range [0,1], with 0 marking normal data.

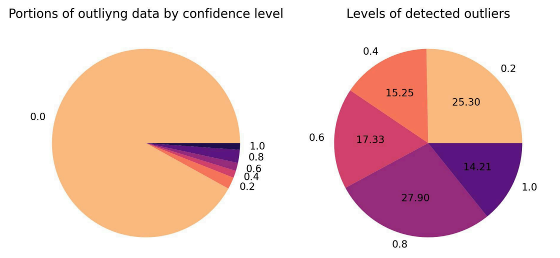

Figure 8:
(left): Pie plot of the anomaly scores in the dataset.
(right): Detail of the (left) plot for anomalies with score > 0. In this case also the percentage of each score obtained is shown.

As the pie chart Figure 8 shows, the amount points that are classified as inliers by all the methods is 92%. Since we used five different methods, we may consider a point as an effective outlier if it is detected by at least 3 algorithms. Therefore, we could set a threshold for the anomaly score at 0.5.
With this threshold, the percentage of anomalies detected is 4.76%. As a reference we reported also the *tSNE* plot (that we used throughout the whole analysis without showing it), that shows a reasonable outcome for the anomaly scores.

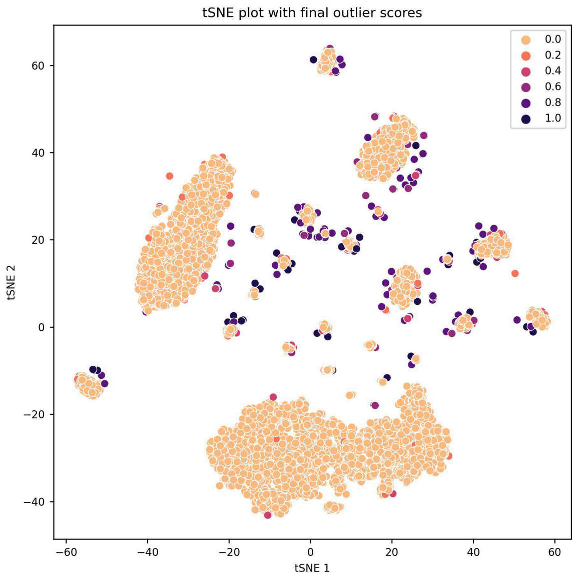

Figure 9: Final outlier scores on the dataset by using tSNE representation. The legend refers to the fraction of methods that detect a certain point as an outlier.

In order to validate the final results we calculated the agreement on the detection among the various methods. Since we are interested in the agreement on the samples detected as anomalies and not also on the inliers we implemented a Jaccard index [14] as measure: only matches in the anomaly detection are counted.

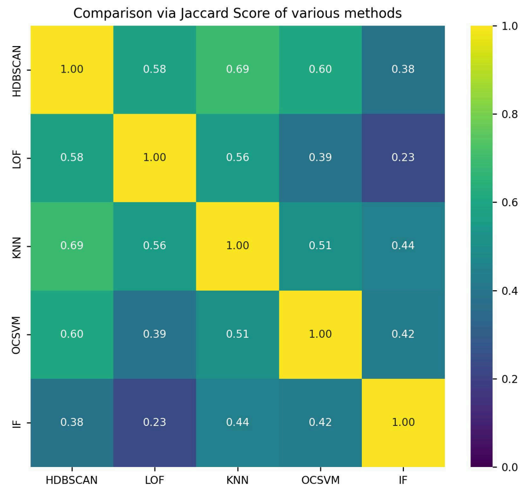

Figure 10: Confusion matrix for each pair of methods solution with Jaccard score measure.

As the Figure 10 shows, the Jaccard scores obtained are quite different: while it can be seen that algorithms like HDBSCAN and KNN got a score over 0.50 with almost the other methods, The Isolation Forest algorithm seems to be in disagreement with all the other algorithms.

Since this algorithm was our starting point to estimate the percentage of anomalies, we analyzed deeper this situation.

Model	Number of outliers	% of anomalies
Isolation Forest	269	3.74 %
HDBSCAN	361	5.04 %
KNN	400	5.56 %
LOF	288	4.00 %
OC-SVM	358	5.00 %

Table 1: Summary table for the methods presented in the report. For *LOF* and OC_SVM the % of anomalies was set as a parameter.

We can see from Table 1 that Isolation Forest results consider quite a smaller number of data with respect to the majority of the methods applied. This suggests that the reason of why the Jaccard scores are quite small with this method is that Isolation Forest detects less anomalies than other methods.

The Figure 11 shows the confusion matrix of the anomalies detected between Isolation Forest and each other method.

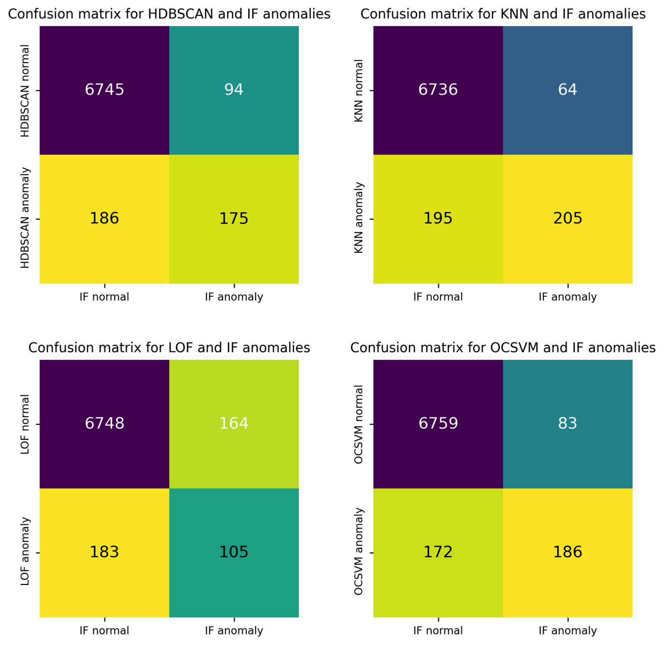

Figure 11: Anomalies found by Isolation Forest compared with the other methods

It is evident that, beside LOF, all the discrepancy in the anomaly matches lies in the bottom-left corner, namely where samples are detected as anomalies for the second method but not from Isolation Forest. This means that Isolation Forest has low jaccard score because it detects fewer outliers with respect to other methods and not because of some lower internal quality.

For the LOF result, a possible reason for the lower agreement with respect to the other results, could be due to varying density combined with the high dimensionality of the data.

As final analysis we plotted in Figure 12 the anomaly score along with the various continuous attributes, this was done to show that not every samples highly outlying in one of the features is not necessarely marked as strong anomalies. This is striking in the case of "Dim_16". We reported here just the continuous variables but the same holds true for the categorical ones.

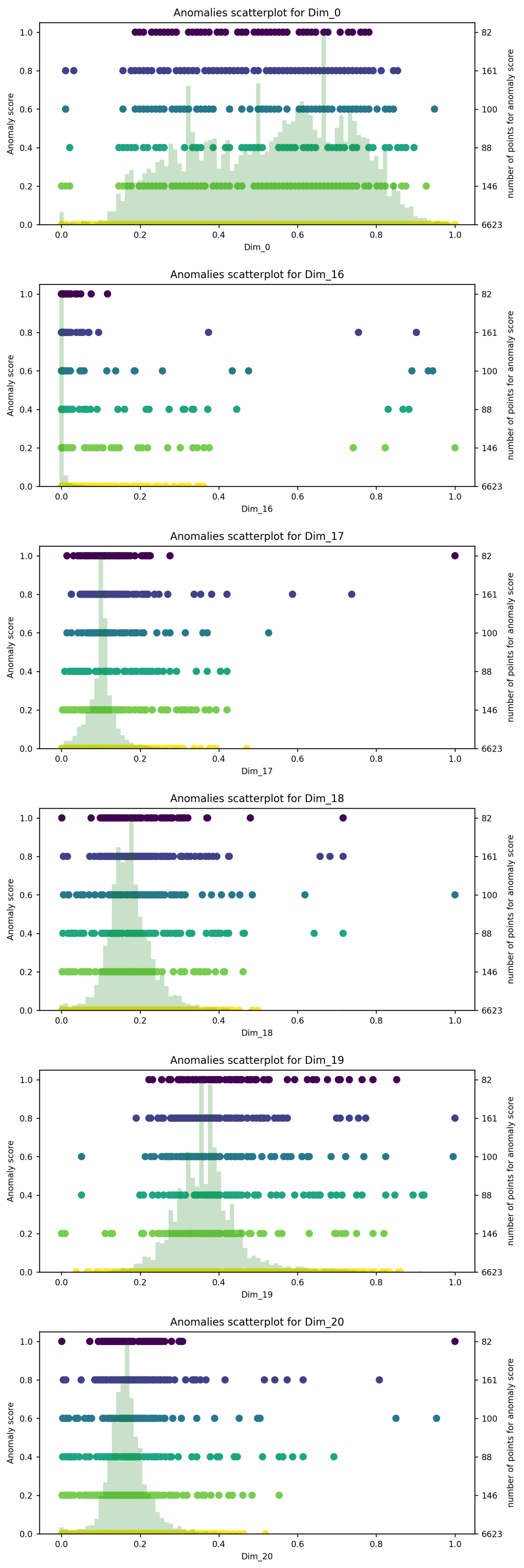

Figure 12: Anomaly scores by continuous attributes value.

G. *Conclusions*

As final recap, we applied several methods by tackling the mixed attributes problem. Some of the methods can deal with both categorical and numerical attributes, others need a specific encoding for the binary features or the definition of a metric suited for mixed data types.

At the end of this study we conclude that the methods are mostly in good agreement and that their individual performances are satisfying. Thanks to this property we are confident in a reliable anomaly detection.

We obtained multiple outlying levels that allows for a more detailed final anomaly detection provided the interpretation of the data. In detail, our analysis concludes that in the dataset 92% of the data points are sure inliers, while 4.76% of the data points are marked as outliers if a standard threshold of 0.5 is set.

One possible improvement is to substitute the LOF method with COF (Connectivity Outliers Factor), that should lead to better performance in case of varying densities in the data.

VI. Appendix

A. *Peaks in the distance histogram*

In this section we would like to present a very short test to show that the peaks present in the Gower's pairwise distance are likely to be associated with a clustering structure

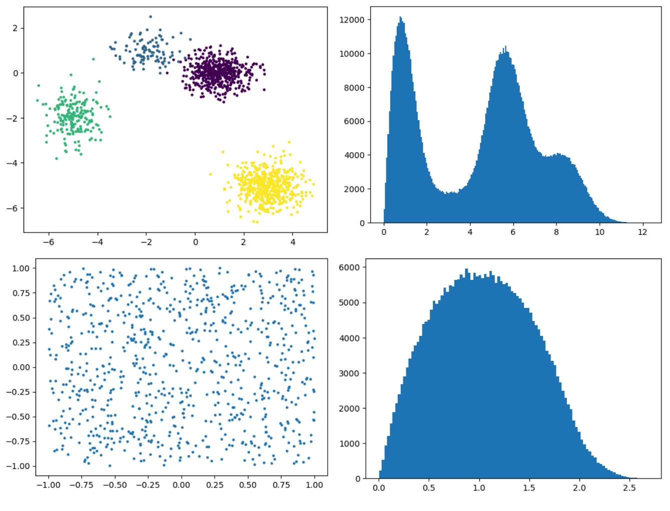

Figure 13:
(up) Histogram for clustered data.
(down) Histogram for unclustered data.

As it can be seen in Figure 13 the clustered synthetic dataset presents different peaks in the pairwise distances histogram. In the case of uniformly generated data this does not hold true, a single bell-shaped distribution is generated. Of course the same reasoning is valid in higher dimensions and with other reasonable metrics.

With this final statement we want to claim that everything we have written in this paper is a result of our personal knowledge and experience, every reference used to carry out this work has been properly cited and no form of plagiarism has been made. Moreover, none of this text has been generated using natural language processing models.

References

[1] J. C. Gower, “A general coefficient of similarity and some of its properties,” *Biometric*, vol. 27, pp. 857–871, 1971.

[2] F. T. Liu, K. M. Ting, and Z.-H. Zhou, “Isolation Forest,” vol. 0, no. , pp. 413–422, 2008, doi: 10.1109/ICDM.2008.17.

[3] “Isolation Forest algorithm from Scikit-learn,” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html

[4] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, “A density-based algorithm for discovering clusters in large spatial databases with noise,” pp. 226–231, 1996.

[5] “DBSCAN algorithm from scikit-learn.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

[6] R. J. G. B. Campello, D. Moulavi, and J. Sander, “Density-Based Clustering Based on Hierarchical Density Estimates,” pp. 160–172, 2013, [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-642-37456-2_14

[7] “HDBSCAN algorithm from scikit-learn.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.cluster.HDBSCAN.html

[8] “Local Outlier Factor algorithm from Scikit-learn,” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor

[9] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt, “Support Vector Method for Novelty Detection,” *NIPS*, vol. 12, pp. 582–588, 1999.

[10] “One Class SVM algorithm from scikit-learn.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html

[11] “Knee Locator algorithm from kneed.” [Online]. Available: https://pypi.org/project/kneed/

[12] “Silhouette score implementation from Scikit-learn.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.silhouette_score.html

[13] “tSNE implementation from scikit-learn.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

[14] “Jaccard score from Scikit-learn.” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.jaccard_score.html