Intelligent Data Analysis 1 (2020) 1–5
IOS Press

Modeling Data with Observers

Fares Meghdouri *Félix Iglesias Vázquez and Tanja Zseby
[a] *Telecommunication Institute, TU Wien, Gusshausstrasse 25-25a, 1040, Vienna, Austria*
E-mails: fares.meghdouri@tuwien.ac.at, felix.iglesias@tuwien.ac.at, tanja.zseby@tuwien.ac.at

Abstract. Compact data models have become more prominent due to the massive, ever-increasing generation of data. We propose Observers-based Data Modeling (ODM), a lightweight algorithm to extract low density models that are suitable for both static and stream data analysis. ODM is a convenient intermediate step that alleviates computational costs of machine learning during evaluation phases. We compare ODM with previous proposals in classification, clustering, and outlier detection. Results show the convenience of ODM in terms of accuracy, versatility, and speed.

Keywords: Big data, Low density model, Coreset

1. Introduction

Data is a key resource for Artificial Intelligence (AI) applications since generalization and prediction require considerable volumes of information to build reliable ground truth. In principle, and assuming a sufficient data quality, the more data available during Machine Learning (ML) training stages, the better the quality of the obtained models and classifications. This subsequently allows researchers to improve their frameworks and reach theoretical limits of learning. In this respect, nowadays researchers have access to more data than ever. In 2017, IBM disclosed that 90% of the data in human history has been produced in the previous two years, also that this trend would increase exponentially.

Managing, processing, and analyzing such ever-increasing volumes of data is a pressing challenge. Many algorithms either (1) cannot cope with massive raw data on-the-fly (freezing or collapsing) or (2) suffer from performance degradation due to noisy data, therefore requiring lightweight data summarization in between. For example, Zimek et al. show how subsampling in instance-based methods improves accuracy for outlier detection [1], introducing the counter-intuitive concept of "higher accuracy with less data". Data summaries become key middle boxes for future ML, either to understand data, avoid degradation, or to reduce computational costs. In this paper, a *low density data model* is equivalent to a *coreset*, which is a subset $\mathcal{C}$ of data samples from an original set $\mathcal{X}$ that retains some characteristic properties for a given objective or fitting function $o(.)$ such that $o(\mathcal{C}) \approx o(\mathcal{X})$, as introduced by Agarwal et al. in [2].

To this end, we propose ODM, a lightweight algorithm for extracting low density data models that scales properly and helps to simplify and understand huge volumes of both static and stream data, creating optimized models for clustering, outlier analysis, and classification. ODM retrieves the idea of *data observers* presented in [3], which is a subset of data points used for approximating the original distribution, hence a coreset. Noteworthy is the fact that observers do not have to necessarily be real data points. Moreover, ODM admits some parameterization to adjust the granularity of the desired model.

*Corresponding author. E-mail: fares.meghdouri@tuwien.ac.at.

The rest of the paper is organized as follows: previous research and methods are discussed in Section 2; in Section 3 we explain the algorithm and build its mathematical foundations; evaluation, experiments and used data are described in Section 4; results are shown in Section 5; finally, conclusions and a summary of paper contributions are given in Section 6.

2. Related Work

The intuitive approach to understand processes behind any kind of structural data generation is to find a simplified yet representative model that has an analogous distribution to the original one. Regardless of the type of analysis to perform, handling raw data is complex and time-demanding. In addition, modern data is often noisy and unstable, therefore low density models appear as a suitable solution.

For instance, developed by Kohonen et al. [4], self-organizing maps (SOM) are very useful tools for explaining multi-dimensional numerical data. They project complex input spaces into two-dimensional neural grids while trying to keep topological structures that are inherent to the original data. Based on SOM, Martinez et al. [5] propose Neural Gas and, later, Fritzke et al. present Growing Neural Gas [6]. These are algorithms that also learn typologies by means of iterative fitting (or "growing") processes. Intuitively, neurons behave like gas particles that extend and expand into the data, whose multidimensional shapes are seen as containers. This is equivalent to the idea of *low density models* pursued by ODM. In order to reduce the size of a given dataset, the Condensed Nearest Neighbor rule (CNN) was developed in [7, 8] as a solution for supervised training with fewer representative samples. The algorithm is based on the 1-Nearest Neighbor label to retain only necessary samples therefore eliminating redundancy. Similarly, bayesian coresets [9–12] are statistical tools that capture the data shape by extracting underlying probabilistic models. The KMeans algorithm can also be used to find coresets [13] by extracting the shape of the input with a set of centroids that reflect main clusters in data. Another way to build coresets are Gaussian Mixture Models [14, 15], which reconstruct the shape of the original data with a mixture of gaussian functions in which characteristic parameters are adjusted accordingly.

Finally, Iglesias et al. [3] mint the concept of "observers" to create a model of the data and use it later for detecting anomalies with the SDO (Sparse Data Observers) algorithm. Authors define observers as a set of sampled data points in which potential outliers have been removed according to a distance-based metric. These observers are later used as anchors to evaluate data points based on proximity calculations.

3. ODM

This section describes the ODM algorithm, its mathematical formulation, and possible adaptations based on the application domain. The algorithm framework is open-source and available in our git-hub repository.[1] Table 1 collects the notation and the algorithm parameters used throughout the paper.

3.1. Observers

In [3], $\mathcal{C}$ (the observers set) is initially formed by randomly subsampling $\mathcal{X}$. They are a set of points that reflect data shapes with a lower density (like a fading image). Applied as models, observers are intended to evaluate data points from the input space that are located in their vicinity.

[1] github.com/CN-TU/pyodm

*	Symbol	**	Description
	$\mathcal{X}$	–	Dataset
	$\mathcal{C}$	–	Set of observers (coreset)
	$\mathcal{Y}$	–	Random subset of $\mathcal{X}$
	$\mathbf{x_i}$	–	Vector of the i^{th} data point in $\mathcal{X}$
	$\mathbf{c_j}$	–	Vector of the j^{th} observer in $\mathcal{C}$
	$\mathbf{y_k}$	–	Vector of the k^{th} data point in $\mathcal{Y}$
	n	$\lvert\mathcal{X}\rvert$	Size of the data
✓	m	–	Size of the coreset
	N	–	Space dimensionality (number of features)
✓	R_o	–	Default radius of a new observer
	R_k	–	Radius of a k^{th} observer
	P_j	–	Number of points assigned to the j^{th} observer
✓	f	0.1	Expansion coefficient
✓	ρ	0.025	Sampling ratio for R_o estimation
✓	β	1	Correction factor for R_o estimation
✓	α	1	Observer inertia coefficient
	$d(.)$	–	Distance metric
	$O(.)$	–	Time complexity
	O_m	–	Time complexity of ODM with m-Trees core
	r	–	Size ratio (%) between $\mathcal{C}$ and $\mathcal{X}$
	Δu	–	Displacement of an observer after an update

* Algorithm parameter
** Default value

Table 1
Notation.

We take this intuitive idea as a basis and modify concepts by giving observers some new properties. Here observers are not required to be actual data points. Additionally, each observer $\mathbf{c_j}$ has a maximum observation radius R_j and a population P_j, which accounts for the number of observed data points.

3.2. Algorithm

The goal of the ODM algorithm is to build a set of proper observers (aka an optimal coreset that maximizes evaluation metrics depending on the given application). Intuitively, ODM starts taking a random data point as the first observer. Later, it processes the remaining data points sequentially. Given an i-data point ($\mathbf{x_i}$), if the distance to the closest observer ($\mathbf{c_j}$) is below the radius of the observer (R_j), $\mathbf{c_j}$ properties are modified to better fit the data; otherwise, $\mathbf{x_i}$ becomes a new observer. Some variables are worth mentioning:

- R_o (optional) is the default radius for any new observer.
- f (required) is an expansion factor that defines the growth rate of R.
- ρ (required if R_o is not given) is a sampling ratio used to estimate R_o in case that it is not externally set.
- $m = |\mathcal{C}|$ (optional) is the number of observers (size of the coreset).
- α (optional) is a inertia coefficient that determines how observers update their locations.
- β (optional) is a correction parameter for estimating R_o.

Algorithm 1: ODM

(Required) **Input:** f, R_o
(Optional) **Input:** ρ, m, α, β
Output: $\mathcal{C}$
Data: dataset $\mathcal{X}$
Shuffle $\mathcal{X}$
if *R_o is not specified* **then**
    Estimate $\widehat{R_o}$ from $\mathcal{Y}$, a fraction ρ of $\mathcal{X}$
    Set $R_o = \beta\widehat{R_o}$
Set a random sample $\mathbf{x_r}$ as $\mathbf{c_0}$
Set P_0 to 1
Set R_0 to R_o
Append $\mathbf{c_0}$ to $\mathcal{C}$
for *i in* $|\mathcal{X}| - 1$ **do**
    Get r the index of the nearest observer to $\mathbf{x_i}$
    if $d(\mathbf{c_r}, \mathbf{x_i}) \leqslant R_n$ **then**
        Shift $\mathbf{c_r}$'s coordinates by $\alpha \cdot \Delta u$
        Increment P_r by 1
        Subtract a fraction $f \cdot d(\mathbf{c_r}, \mathbf{x_i})$ from R_r
    else
        Add a fraction $f \cdot d(\mathbf{c_r}, \mathbf{x_i})$ to R_r
        Create a new observer $\mathbf{c_{new}} = \mathbf{x_i}$
        Set P_{new} to 1
        Set R_{new} to R_o
        Append $\mathbf{c_{new}}$ to $\mathcal{C}$
if *m is specified* **then**
    Sort $\mathcal{C}$ in descending order based on P values
    return first m elements of $\mathcal{C}$
return $\mathcal{C}$

Algorithm 1 shows the core routine of ODM and the mathematical formulation is discussed in Section 3.5. In particular, the algorithm performs the following steps:

(1) Shuffle the dataset to avoid bias.
(2) If R_o is not adjusted as an external parameter, a random subset of the dataset ($\mathcal{Y}$) is constructed with $|\mathcal{Y}| = \lceil n\rho \rceil$ elements and then R_o is calculated as the average distance between samples from $\mathcal{Y}$ and their corresponding centroid adjusted by a correction parameter β.
(3) The coreset is initialized with a first observer $\mathbf{c_0}$, which is a randomly chosen data point from $\mathcal{X}$. Hence, $R_0 = R_o$, and $P_0 = 1$.
(4) Later, the remaining data points are processed sequentially. For a given $\mathbf{x_i}$, the algorithm calculates the distance between $\mathbf{x_i}$ and its closest observer $\mathbf{c_j}$. If the distance is smaller than the observer radius R_j, $\mathbf{c_j}$ is moved closer to $\mathbf{x_i}$, its population P_j increases by one and its radius R_j is decreased by a

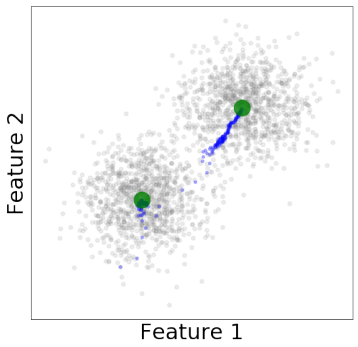

Fig. 1. Two observers (green) modeling a two-cluster dataset. Gradual observers locations are shown in blue.

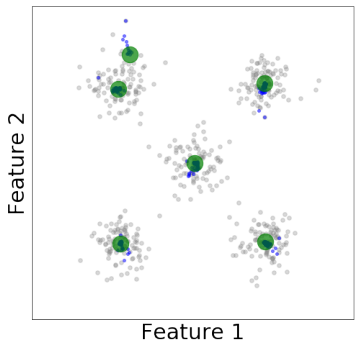

Fig. 2. Six observers (green) modeling a five-clusters dataset. Gradual observers locations are shown in blue.

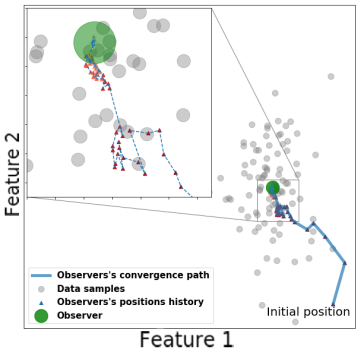

Fig. 3. Convergence path of an observer to the center of a sample cloud.

fraction f of the distance $d(\mathbf{c_j}, \mathbf{x_i})$. If, instead, the distance is larger, $\mathbf{x_i}$ becomes a new observer and $\mathbf{c_j}$ radius R_j is increased by a fraction f of the distance $d(\mathbf{c_j}, \mathbf{x_i})$. The population P increases over time and prevents some samples to become excessive due to the sequential processing.

The central loop of Algorithm 1 can be parallelized to enable faster extraction. Each data-batch is processed individually and the resulting subcoresets are simply joined together afterwards.If the size of the coreset is externally fixed as a parameter (m), the algorithm keeps only the m-most populous observers for the final coreset.

3.3. Examples

The ODM algorithm can be better understood with some visual examples. Fig. 1 and Fig. 2 represent two datasets [16, 17] with two and five clusters. For these two cases ODM has been applied to obtain coresets with two and six points respectively. Data points are shown in gray-color, the final positions of observers in green, and the displacement of observer is tracked with blue dots. In both cases we see how observers end up taking the respective clusters centers. If the number of observers is higher, the observers placement is set based on density variations. We zoom the observer movement in Fig. 3, which uses a different dataset [18]. Here we can see that the observer placement converges and its displacement is progressively smaller.

The previous examples use very few observers in order to show the natural tendency of observers to be attracted by density centers. In practical modeling, higher granularity is commonly required, especially in cases with complex data structures. Fig. 4 shows three datasets [19–21] for which ODM has been adjusted with low, medium, and high number of observers. As discussed above, denser coresets can be created by simply adjusting ODM parameters. Obviously, the size of a coreset cannot be larger than the size of the modeled dataset.

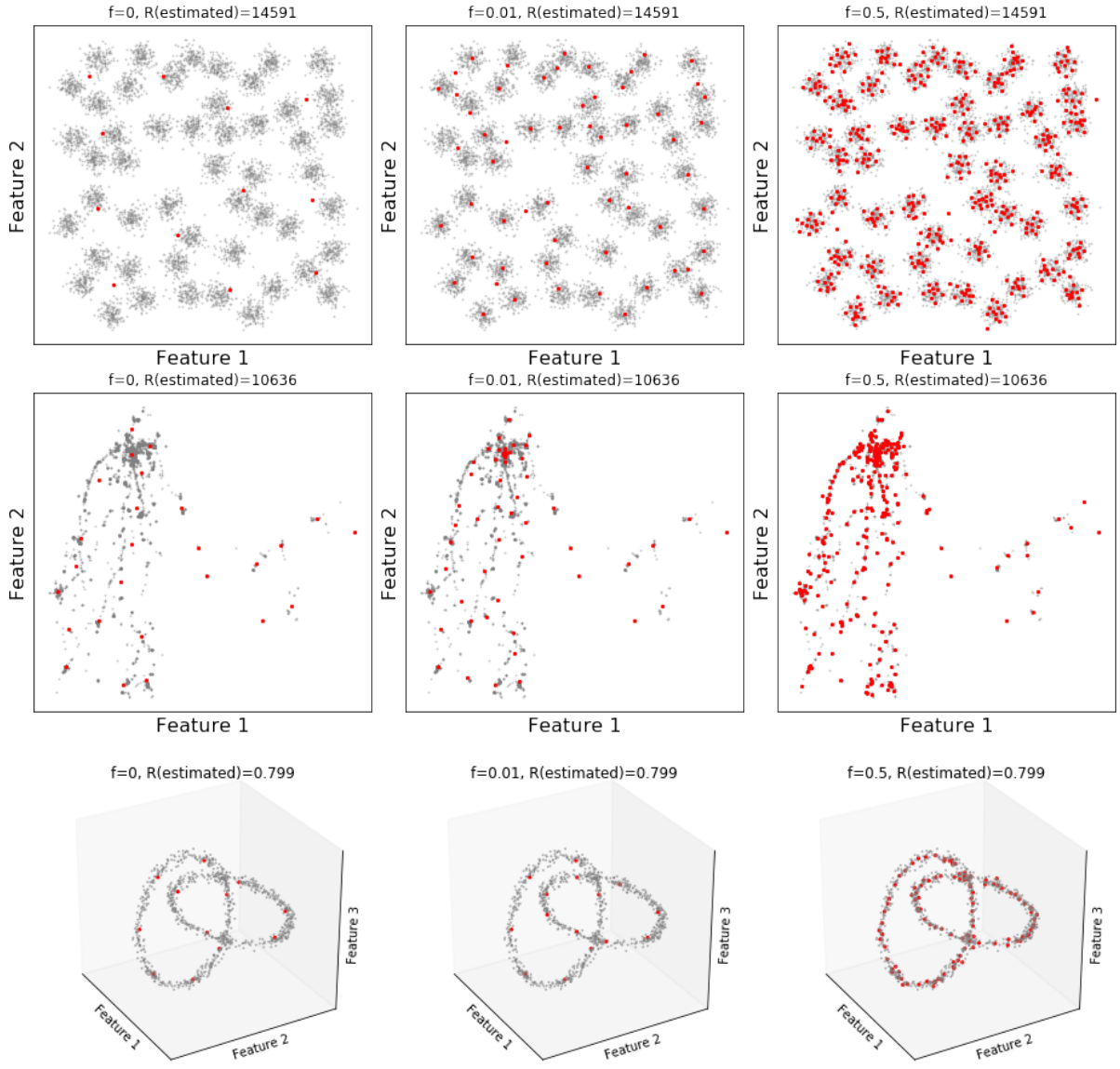

Fig. 4. ODM coresets extracted from three different datasets. Each column shows a parameterization with a different number of observers. Data points are shown in gray calor and observers in red color.

3.4. Complexity

The computational demand of ODM mainly appears when searching closest observers. Different algorithms can be used here, like kd-Trees [22] or m-Trees [23]. We opt for m-Trees in our implementation since they are robust, fast, and allow changes in the tree without need for reconstruction. The tree is built once at the beginning and keeps track of the observers. Based on Algorithm 1, each new data point involves: one k-nearest neighbor query, one insert (either a new observer or an updated one), and optionally one removal (remove an old observer if it has been updated and re-inserted). In the highly improbable

average worst case scenario, which happens if the number of observers increases with each iteration and approaches the number of data points, the complexity O_m with a m-Tree core can be estimated as follows: **number of data points** × **(k-NN query + removal + insertion)**, which gives:

$$\begin{aligned} O_m &= O(n) \cdot [O(logn) + O(logn) + O(logn)] \\ &= \mathbf{O(n \log n)} \end{aligned} \tag{3.1}$$

n being the number of processed data points (aka the cardinality of the dataset, $|\mathcal{X}|$),

3.5. Simplified Mathematical Formulation

In this section we address the mathematical formalism underlying ODM. ODM accepts different metrics for distance calculation (for instance, Aggrawal et al. [24] recommend using the Manhattan distances instead of Euclidean distances for high-dimensional spaces).

Initially, assuming that at least one observer exists in a coreset $\mathcal{C}$, let $\mathbf{x_i} \in \mathbb{R}^N$ be the i^{th} observation in an N dimensional dataset $\mathcal{X}$ and let $\mathbf{c_j} \in \mathbb{R}^N$ be the closest observer to $\mathbf{x_i}$:

$$\mathbf{x_i} = \begin{bmatrix} x_{i,0} \\ x_{i,1} \\ \vdots \\ x_{i,N} \end{bmatrix} \qquad \mathbf{c_j} = \begin{bmatrix} c_{j,0} \\ c_{j,1} \\ \vdots \\ c_{j,N} \end{bmatrix} \tag{3.2}$$

If the distance between $\mathbf{c_j}$ and $\mathbf{x_i}$ is smaller than R_j:

$$d(\mathbf{c_j}, \mathbf{x_i}) \leqslant R_j \tag{3.3}$$

The coordinates of $\mathbf{c_j}$ are updated such:

$$\mathbf{c_j} \leftarrow \mathbf{c_j} + \alpha \Delta \mathbf{u} \tag{3.4}$$

Where α is the observer's inertia coefficient that controls the displacement (set to 1 by default) and $\Delta\mathbf{u}$ is the displacement defined as:

$$\Delta \mathbf{u} = \frac{\mathbf{x_i} - \mathbf{c_j}}{P_j + 1} \tag{3.5}$$

P_j is the number of data points associated to the observer $\mathbf{c_j}$ (i.e., observations/population) and $\mathbf{x_i} - \mathbf{c_j}$ is a distance vector. An observer $\mathbf{c_a}$ with high population is called *lazy* observer since the displacement when observing a new data point tends to vanish, i.e.,:

$$\lim_{P_a \to \infty} \Delta \mathbf{u} = \lim_{P_a \to \infty} \frac{\mathbf{x} - \mathbf{c_a}}{P_a + 1} = \mathbf{0} \tag{3.6}$$

Equation 3.6 shows that after many iterations, more observers become lazy and their location tends to be bounded in a smaller region in the output space. Moreover, Equation 3.6 always holds since the distance vector $\mathbf{x} - \mathbf{c_a}$ cannot be arbitrarily large (in comparison to $P_a + 1$) nor exceeds the radius of the observer (Equation 3.3).

The radius of the closest observer is at the same time updated regardless of the distance check performed in Equation 3.3:

$$R_j \leftarrow \max\left(R_j \pm f \times d(\mathbf{c_j}, \mathbf{x_i}), 0\right) \tag{3.7}$$

Depending on the distance check, the radius R_j of an observer $\mathbf{c_j}$ is increased or decreased by a fraction f of the distance between the current sample ($\mathbf{x_i}$) and the closest observer ($\mathbf{c_j}$) itself. Conversely the $max(., 0)$ operator ensures that the radius is always positive.

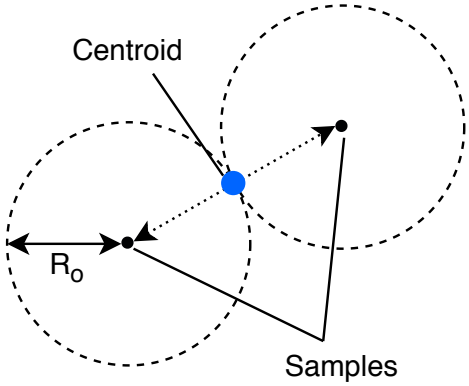

Fig. 5. By applying Equation 3.8, the initial radius R_o is estimated to be the radius of circles centered at each sample and optionally corrected using β (=1 in this case).

If not specified as an external parameter, the initial radius of new observers is estimated as follows:

$$R_o = \beta\left(\frac{1}{|\mathcal{Y}|}\sum_{j}^{|\mathcal{Y}|} d(\mathbf{y_j}, \bar{\mathbf{y}})\right) \tag{3.8}$$

Whereby $|\mathcal{Y}| = \lceil n\rho \rceil$. Equation 3.8 computes the average distance to the centroid of samples in a set $\mathcal{Y}$ sampled from $\mathcal{X}$. β is a tunable correction parameter (set to 1 by default). Fig. 5 shows how R_o is estimated in the case of a two-points subset.

4. Experiments

In this section we describe experiments conducted to evaluate ODM. They are applications related to anomaly detection, clustering, and supervised classification. In the experiments, ODM is compared with alternative coreset extraction algorithms (described in Section 2), namely:

- **Baseline**. No coreset, the whole dataset is used instead.
- **RS**. A coreset obtained by uniform Random Samping.
- **SDO**. Sparse Data Observers [3].
- **KMC**. k-Means Coreset [13].
- **BGM**. Bayesian Gaussian Mixture [15].
- **GNG**. Growing Neural Gas [6].
- **CNN**. Condensed Nearest Neighbours [7, 8] (only for supervised classification).

4.1. Datasets

For the experiments we used public datasets that are commonly applied for algorithm testing in the literature. A prerequisite when selecting datasets was that they should contain a considerable number of samples in order to make the application of coresets meaningful. They are:

- **MDCG-datasets**. 15 datasets created with MDCGen [25], a tool for the highly-customized generation of multi-dimensional, multi-cluster datasets for algorithm testing. Parameters were selected to obtain scenarios with different number of dimensions, clusters, sample sizes, cluster overlap, and noise.
- **OTDT-datasets**. A set of six popular datasets for outlier detection collected from the literature [26–31]. They are imbalanced datasets with two possible classes: normal and anomalies (or *inliers* and *outliers*). The positive class corresponds to anomalies, which account for less than 10% of the total dataset size.
- **MCC-datasets**. A set of five modern and balanced classification dataset with binary and multi-class targets (up to 10 classes) are used for the classification task [32–36].
- **Toy-datasets**. Some real and synthetic datasets (up to 3 dimensions) selected from the literature to illustrate characteristics of the algorithm under study [16–21].

4.2. Anomaly Detection

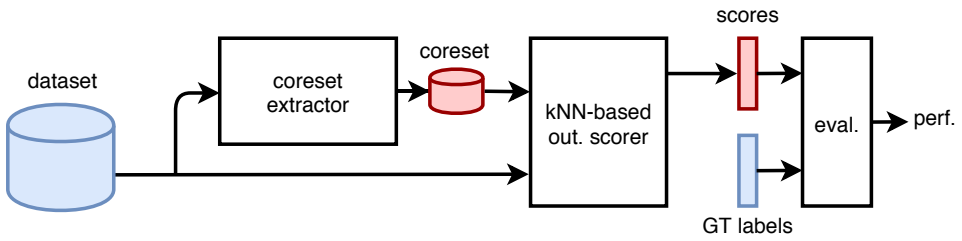

Fig. 6. Scheme of Anomaly Detection experiments.

Coresets can help anomaly detection to alleviate the computational cost of instance-based methods and avoid degradation [1]. The traditional way of measuring outlierness is by considering the whole dataset (or all previous data points) when evaluating each single data point. Instead, with coresets each single data point is contrasted only with the coreset to decide if it is considered as anomaly, therefore considerably reducing the computational burden. The experiment methodology is shown in Fig. 6, its elements are described as follows:

- *Datasets*. We used OTDT-datasets (Section 4.1).
- *Coreset extractors*. Competitor algorithms were tuned to create coresets with different r values (0.5%, 1%, 5% and 10%), where r is the size of the coreset relative to the dataset size, i.e.,

$$r = \frac{|C|}{|\mathcal{X}|} \cdot 100 = \frac{m}{n} \cdot 100 \tag{4.1}$$

  Since anomaly detection is unsupervised learning, additional parameters of each algorithm are tuned with the default values suggested in the respective references.
- *Outlierness scorer*. We use the kNN-based outlier detection algorithm proposed by Ramaswamy et al. [37] to obtain outlierness scores. The number of neighbors is set to five (k=5).
- *Evaluator*. We evaluate performance by computing the P@n, average precision, ROC-AUC and MaxF1 scores as suggested by [38]. We show *adjusted* metrics to allow fair comparisons regardless of the number of outliers. We additionally measure coreset extraction and k-NN-based outlierness scoring (empirical) average runtimes.

The setup shown in Fig. 6 covers 144 experimental runs (6 algorithms × 6 datasets × 4 r values) in addition to the baseline variant. Experiments results are summarized in Table 2 and discussed in Section 5.1.

4.3. Clustering

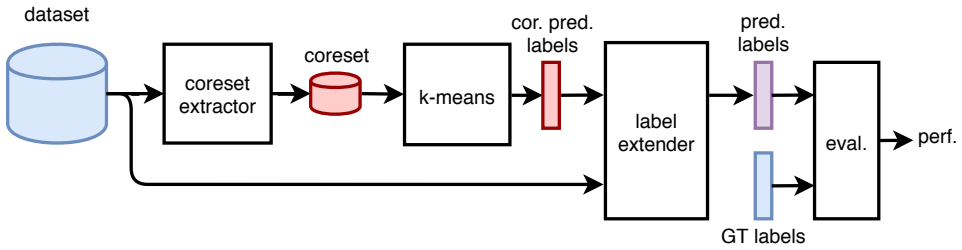

Fig. 7. Scheme of Clustering experiments.

Similarly to the anomaly detection case, in order to reduce memory and time requirements, label assignment in clustering can be performed by clustering a coreset and extending labels to the whole dataset based on proximity calculations. We reproduced this scheme in our experiments and used k-means as core algorithm. The methodology is shown in Fig. 7. Elements are:

- *Datasets*. We used MDCG-datasets (Section 4.1).
- *Coreset extractors*. Competitor algorithms were tuned with different r values (0.5%, 1%, 5% and 10%). Again, additional algorithm parameters were tuned based on the default values suggested in source references.
- *k-means*. Clustering is undertaken by the k-means++ algorithm [39]. The initial number of cluster k is assumed as a known value prior to the analysis.
- *Label extender*. This block calculates coreset cluster centers (centroids) and assigns final labels to the whole dataset by extending the label of the closest center (1-NN distance metric).

- *Evaluator*. We evaluate performances with the rand index [40] between predicted labels and GT labels. We additionally measure coreset extraction, k-means and label extender (empirical) average runtimes.

The setup shown in Fig. 7 covers 360 experimental runs (6 algorithms × 15 datasets × 4 r value) in addition to the baseline variant. Experiments results are summarized in Table 3 and discussed in Section 5.2.

4.4. Supervised Classification

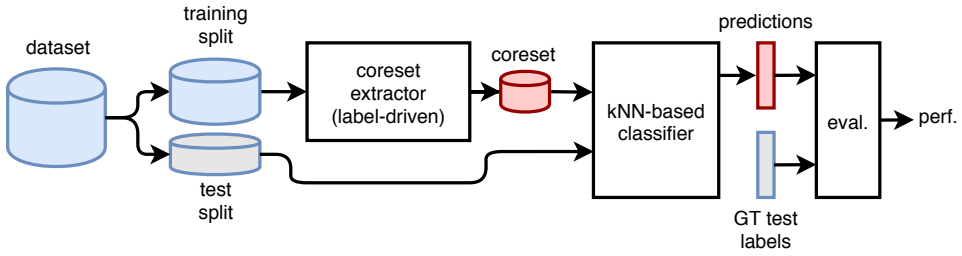

Fig. 8. Scheme of Supervised Classification experiments.

Coresets can also be used in supervised classification for data reduction, therefore suppressing noise, favoring generalization, and reducing training and/or evaluation complexity. Here coresets are intended to retain the essential set of training points for the subsequent classification. Within k-nearest neighbor classification, this method is known as *condensed nearest neighbor* [7]. Note that such data reduction is strongly recommended for instance-based (aka *lazy*) learners that deal with large datasets, in which the coreset would work as a model and make it closer to *eager* learning options. This approach would be consistent with the sampling recommendations given by Zimek et al. in [1] for the anomaly detection case. In our experiments we reproduce the setup with the following blocks (Fig. 8):

- *Datasets*. We used MCC-datasets (Section 4.1). We divided each dataset in training (70%) and test (30%) splits with stratified sampling. Datasets show different categories, therefore this becoming a multiclass classification experiment.
- *Coreset extractors*. Competitor algorithms were tuned again with different r values (0.5%, 1%, 5% and 10%). "label-driven" means that training data was divided into subsets based on GT labels and coresets were consequently extracted from each subset and later joined. This setup was used for all algorithms, except for CNN, which is already designed to properly deal with multiclass data. Also, vanilla CNN cannot be adjusted to create coresets with specific number of datapoints; therefore, random sampling was applied as a last step to equalize CNN with other algorithms and allow fair comparisons.
  Other algorithm parameters were adjusted by using Evolutionary search on training data.
- *k-NN based classifier*. We use a vanilla kNN-classifier to predict labels. The number of neighbors is set to five (k=5).
- *Evaluator*. We provide accuracy as well as micro and macro precision and recall metrics. We additionally measure coreset extraction and k-NN-based classification (empirical) average runtimes.

The setup shown in Fig. 8 covers 140 experimental runs (7 algorithms × 5 datasets × 4 r values) in addition to the baseline variant. Experiments results are summarized in Table 4 and discussed in Section 5.3.

5. Discussion

In this section, the results obtained from the conducted experiments are discussed.

5.1. Anomaly Detection

Results in Table 2 support the claims in [1] and show that, as a general rule, subsampling data improves the accuracy of outlier detection algorithms, since the baseline approach (i.e., using the whole dataset) performs worse than experiments that use coresets regardless of the used technique and the considered metric. Beyond that, all studied algorithms seem to obtain representative coresets for the analysis, KMC, SDO, and ODM slightly standing out over the rest.

Table 2
Performance of detecting anomalies.

Algorithm	r	MaxF1	Adj-MaxF1	P@tn	Adj-P@tn	AP	Adj-AP	ROC-AUC	Ext. Time (s)	k-NN Time (s)
Baseline	0%	0.32 ± 0.08	0.29 ± 0.10	0.14 ± 0.12	0.11 ± 0.10	0.19 ± 0.16	0.16 ± 0.15	0.83 ± 0.09	—	5.47 ± 5.44
ODM	0.5%	0.45 ± 0.23	0.42 ± 0.24	0.23 ± 0.25	0.21 ± 0.25	0.26 ± 0.25	0.23 ± 0.24	0.86 ± 0.12	2.26 ± 2.32	2.64 ± 2.77
RS		0.38 ± 0.17	0.35 ± 0.19	0.17 ± 0.16	0.14 ± 0.15	0.22 ± 0.19	0.19 ± 0.18	0.81 ± 0.12	0.00 ± 0.00	2.77 ± 2.98
SDO		0.43 ± 0.23	0.40 ± 0.24	0.23 ± 0.26	0.20 ± 0.26	0.27 ± 0.26	0.24 ± 0.26	0.85 ± 0.12	0.24 ± 0.32	2.64 ± 2.78
KMC		0.44 ± 0.21	0.41 ± 0.22	0.25 ± 0.27	0.22 ± 0.26	0.27 ± 0.27	0.25 ± 0.26	0.83 ± 0.12	23.27 ± 30.69	2.70 ± 2.81
WGM		0.38 ± 0.24	0.35 ± 0.25	0.23 ± 0.29	0.20 ± 0.29	0.25 ± 0.27	0.23 ± 0.27	0.78 ± 0.13	228.49 ± 331.62	2.67 ± 2.79
GNG		0.40 ± 0.19	0.37 ± 0.19	0.23 ± 0.25	0.20 ± 0.24	0.25 ± 0.24	0.22 ± 0.23	0.83 ± 0.12	433.36 ± 589.89	2.82 ± 2.89
ODM	1%	0.41 ± 0.17	0.38 ± 0.18	0.20 ± 0.19	0.17 ± 0.18	0.24 ± 0.21	0.22 ± 0.20	0.84 ± 0.10	2.43 ± 2.44	2.77 ± 2.86
RS		0.38 ± 0.17	0.35 ± 0.18	0.17 ± 0.14	0.14 ± 0.13	0.22 ± 0.18	0.19 ± 0.17	0.82 ± 0.11	0.00 ± 0.00	2.73 ± 2.81
SDO		0.41 ± 0.18	0.38 ± 0.19	0.19 ± 0.18	0.16 ± 0.17	0.22 ± 0.19	0.19 ± 0.18	0.84 ± 0.11	0.43 ± 0.56	2.70 ± 2.82
KMC		0.40 ± 0.18	0.37 ± 0.19	0.18 ± 0.16	0.15 ± 0.15	0.22 ± 0.19	0.19 ± 0.18	0.83 ± 0.10	47.00 ± 59.96	2.82 ± 2.95
WGM		0.38 ± 0.23	0.35 ± 0.24	0.22 ± 0.27	0.19 ± 0.27	0.23 ± 0.27	0.21 ± 0.27	0.79 ± 0.12	414.47 ± 637.57	2.75 ± 2.89
GNG		0.41 ± 0.18	0.38 ± 0.19	0.21 ± 0.20	0.18 ± 0.20	0.23 ± 0.21	0.21 ± 0.20	0.83 ± 0.11	796.74 ± 1109.50	2.71 ± 2.82
ODM	5%	0.40 ± 0.16	0.37 ± 0.18	0.17 ± 0.14	0.14 ± 0.13	0.22 ± 0.18	0.19 ± 0.17	0.84 ± 0.10	2.41 ± 2.45	3.26 ± 3.29
RS		0.39 ± 0.16	0.36 ± 0.18	0.18 ± 0.14	0.14 ± 0.13	0.23 ± 0.19	0.20 ± 0.18	0.83 ± 0.10	0.00 ± 0.00	3.05 ± 3.09
SDO		0.38 ± 0.15	0.35 ± 0.17	0.17 ± 0.14	0.14 ± 0.12	0.22 ± 0.18	0.19 ± 0.17	0.83 ± 0.10	1.82 ± 2.34	3.32 ± 3.35
KMC		0.41 ± 0.20	0.38 ± 0.21	0.16 ± 0.13	0.13 ± 0.12	0.21 ± 0.17	0.18 ± 0.16	0.81 ± 0.11	241.52 ± 325.97	3.49 ± 3.45
WGM		0.26 ± 0.14	0.23 ± 0.16	0.08 ± 0.07	0.04 ± 0.05	0.13 ± 0.12	0.10 ± 0.10	0.72 ± 0.12	2134.30 ± 3212.13	3.01 ± 3.25
GNG		0.28 ± 0.07	0.25 ± 0.07	0.16 ± 0.14	0.13 ± 0.12	0.22 ± 0.18	0.19 ± 0.17	0.81 ± 0.10	3344.25 ± 4983.34	2.87 ± 2.93
ODM	10%	0.39 ± 0.16	0.36 ± 0.18	0.16 ± 0.13	0.13 ± 0.12	0.22 ± 0.18	0.19 ± 0.17	0.84 ± 0.10	2.27 ± 2.34	3.07 ± 3.09
RS		0.39 ± 0.16	0.36 ± 0.18	0.16 ± 0.13	0.13 ± 0.12	0.21 ± 0.18	0.19 ± 0.16	0.84 ± 0.10	0.00 ± 0.00	3.02 ± 3.04
SDO		0.39 ± 0.16	0.36 ± 0.18	0.16 ± 0.13	0.13 ± 0.12	0.22 ± 0.19	0.20 ± 0.17	0.83 ± 0.10	2.56 ± 3.33	2.88 ± 2.94
KMC		0.38 ± 0.18	0.35 ± 0.19	0.16 ± 0.13	0.12 ± 0.11	0.21 ± 0.17	0.18 ± 0.16	0.81 ± 0.11	359.22 ± 497.37	3.15 ± 3.13
WGM		0.27 ± 0.14	0.24 ± 0.16	0.11 ± 0.09	0.07 ± 0.09	0.16 ± 0.13	0.12 ± 0.12	0.72 ± 0.12	3884.61 ± 6105.35	3.36 ± 3.86
GNG		0.27 ± 0.10	0.24 ± 0.09	0.16 ± 0.13	0.12 ± 0.11	0.19 ± 0.17	0.17 ± 0.15	0.80 ± 0.11	6075.36 ± 9054.07	3.01 ± 3.04

Ext. Time: Time needed to build a coreset.

On the other hand, analysis times (*k-NN time*) are considerably reduced when using coresets and such time reduction is expected to increase the larger the dataset is. If we additionally take coreset extraction costs into account (*Ext. Time*), only RS, SDO, and ODM seem to be light and fast enough for the undertaken task.

Weighting accuracy and time performances together, only SDO and ODM satisfactorily meet problem requirements. Fig. 9 shows a visual comparison of coresets by plotting accuracy (*Adjusted Average*

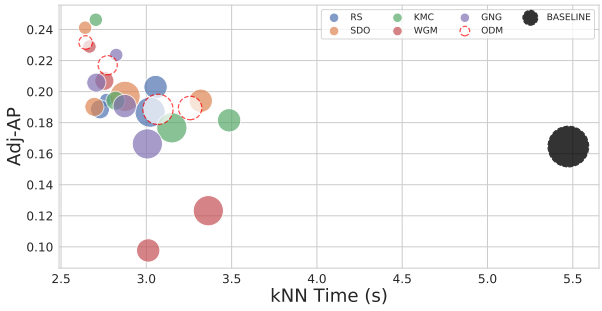

Fig. 9. Anomaly Detection (adj-AP vs k-NN time). Different sizes represent different r values.

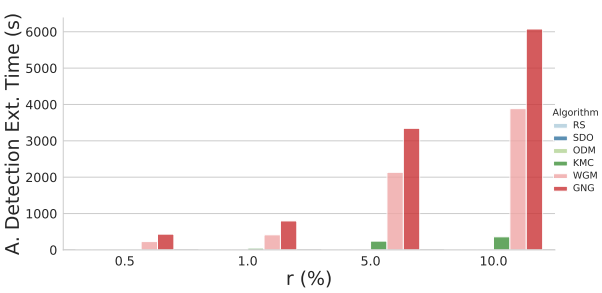

Fig. 10. Coreset Extraction Times.

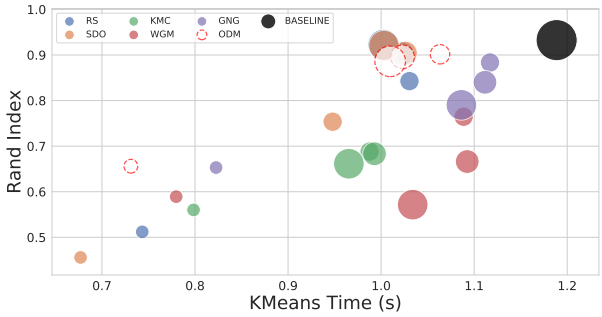

Fig. 11. Anomaly Detection (adj-AP vs k-NN time). Different sizes represent different r values.

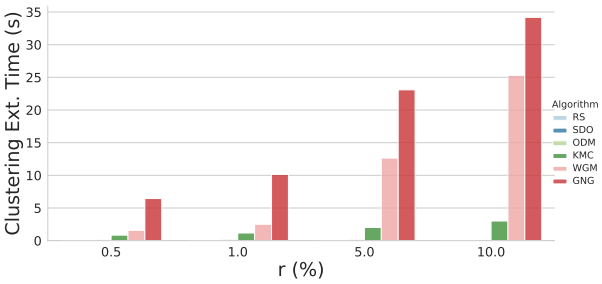

Fig. 12. Coreset Extraction Times.

Precision) versus analysis time (*k-NN time*). A comparison of coreset extraction times (*Ext. Time*) is shown in Fig. 10.

5.2. *Clustering*

In the clustering experiments (results in Table 3), all tested algorithms show poor performances for very low r (i.e., high dataset summarization). Note that the effect of r is hardly generalizable as it strongly depends on the specific distributions and geometries of the data under analysis. Except for the KMC and WGM algorithms, which obtain low performances even for high r, more quality performances are obtained the more datapoints we use for the coreset (i.e, higher r), getting progressively closer to the baseline. SDO and RS obtain almost ideal performances with $r = 10\%$.

Again, coreset extraction costs (*Ext.T.*) are considerably high for WGM and GNG. Weighting accuracy and time performances together, the best trade-off is obtained by SDO and ODM, or even RS. Fig. 11 shows a scatter plot comparing accuracies (*RAND index*) vs analysis time (*kMeans time*). A comparison of coreset extraction times (*Ext. Time*) is displayed in Fig. 12.

5.3. *Supervised Classification*

Similarly to the unsupervised classification experiments, in the supervised classification framework higher values of r tend to improve performances and get closer to the baseline (see experiment results

Table 3
Clustering performance.

Algorithm	r	Rand Ind.	Ext.T.(s)	kM.T.(s)	Lab.E.T.(s)
Baseline	—	0.93 ± 0.03	—	1.19 ± 0.04	—
ODM	0.5%	0.66 ± 0.40	0.14 ± 0.11	0.73 ± 0.44	0.09 ± 0.07
RS		0.51 ± 0.35	0.00 ± 0.00	0.74 ± 0.45	0.09 ± 0.06
SDO		0.46 ± 0.34	0.00 ± 0.00	0.68 ± 0.49	0.08 ± 0.07
KMC		0.56 ± 0.40	0.84 ± 0.54	0.80 ± 0.54	0.09 ± 0.07
WGM		0.59 ± 0.41	1.58 ± 2.07	0.78 ± 0.49	0.09 ± 0.07
GNG		0.65 ± 0.40	6.45 ± 5.47	0.82 ± 0.53	0.10 ± 0.08
ODM	1%	0.90 ± 0.04	0.21 ± 0.13	1.06 ± 0.15	0.11 ± 0.04
RS		0.84 ± 0.08	0.00 ± 0.00	1.03 ± 0.14	0.11 ± 0.04
SDO		0.75 ± 0.22	0.01 ± 0.00	0.95 ± 0.27	0.11 ± 0.06
KMC		0.69 ± 0.21	1.17 ± 0.26	0.99 ± 0.09	0.12 ± 0.06
WGM		0.76 ± 0.22	2.51 ± 1.96	1.09 ± 0.25	0.12 ± 0.06
GNG		0.88 ± 0.05	10.13 ± 6.38	1.12 ± 0.28	0.11 ± 0.06
ODM	5%	0.90 ± 0.06	0.20 ± 0.10	1.02 ± 0.07	0.11 ± 0.04
RS		0.90 ± 0.05	0.00 ± 0.00	1.02 ± 0.07	0.11 ± 0.03
SDO		0.90 ± 0.04	0.04 ± 0.03	1.03 ± 0.06	0.11 ± 0.04
KMC		0.68 ± 0.23	2.02 ± 1.03	0.99 ± 0.15	0.11 ± 0.05
WGM		0.67 ± 0.24	12.64 ± 10.52	1.09 ± 0.18	0.12 ± 0.05
GNG		0.84 ± 0.10	23.09 ± 17.53	1.11 ± 0.30	0.11 ± 0.06
ODM	10%	0.89 ± 0.12	0.19 ± 0.08	1.01 ± 0.03	0.11 ± 0.04
RS		0.92 ± 0.03	0.00 ± 0.00	1.00 ± 0.04	0.11 ± 0.04
SDO		0.92 ± 0.03	0.07 ± 0.05	1.00 ± 0.03	0.12 ± 0.05
KMC		0.66 ± 0.25	3.01 ± 1.62	0.97 ± 0.05	0.10 ± 0.04
WGM		0.57 ± 0.24	25.30 ± 18.58	1.03 ± 0.12	0.11 ± 0.04
GNG		0.79 ± 0.21	34.18 ± 26.41	1.09 ± 0.22	0.11 ± 0.06

Ext.T.: Time needed to build a coreset.
kM.T.: Time needed to build a kMeans model.
Lab.E.T.: Time needed to extend labels.

in Table 4). The performance reduction when using coresets compared with the baseline case is considerable, but coresets still work satisfactorily if we take into account that they are not dealing with binary classification experiments, but multi-class classification. Best accuracy performances are obtained by ODM, KMC, and RS. Even in spite of the fact that analysis runtimes (*k-NN time*) are not discriminating and more or less equally demanding for all cases, KMC is still a bit slower than other algorithms. On the other hand, if we examine coreset extraction costs (*Ext. Time*), there are two clearly separable groups: (a) light extraction: RS, SDO, and ODM; and (b) heavy extraction: KMC, WGM, GNG, and CNN. Weighting accuracy and time performances together, ODM and RS offer the best solutions with affordable complexities.

The CNN additionally appears in this set of experiments as it was specifically proposed for alleviating the computational costs of k-nn-based classification. However, it is a heavy algorithm in terms of extraction costs when compared to ODM or SDO, and accuracy performances are in line or below most of the competitors. Note that, in its original implementation, CNN automatically establishes the size of the coreset according to dataset properties and cannot be externally implemented. This fact might be affecting its capability of obtaining better performances.

Alike previous experiments, Fig. 13 compares coresets by using classification metrics (*Accuracy*, *Macro Precision*, *Macro Recall*) vs analysis time (*kNN time*). A comparison of coreset extraction times (*Ext. Time*) is displayed in Fig. 14.

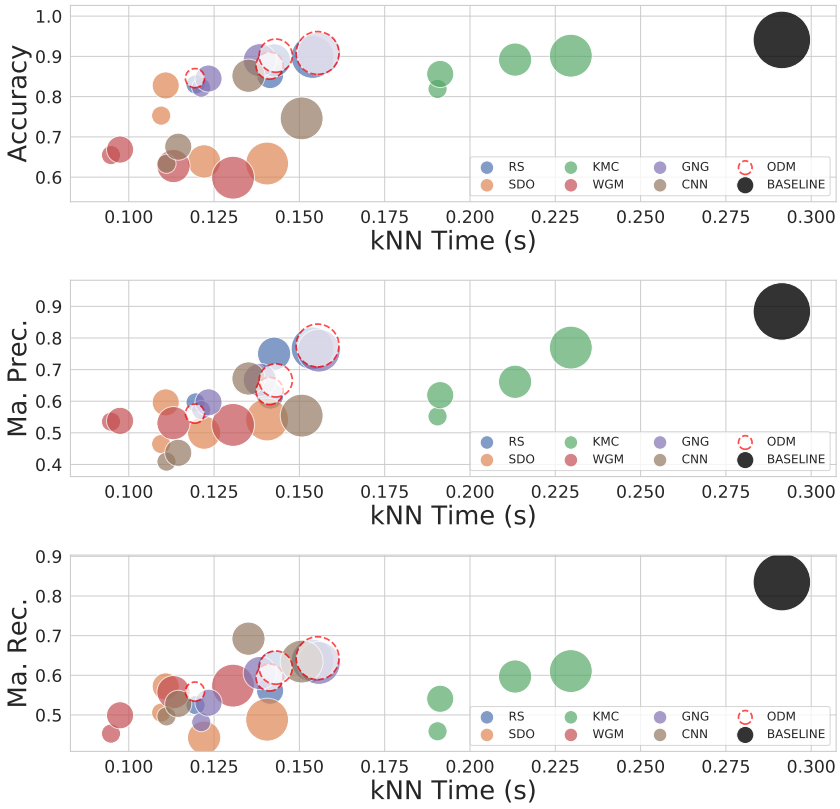

Fig. 13. Supervised Classification. Different sizes represent different r values.

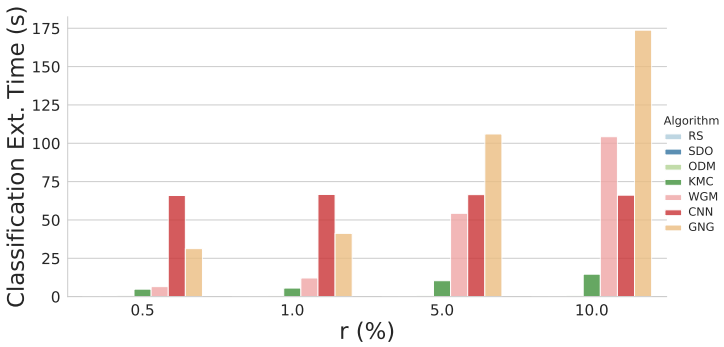

Fig. 14. Coreset Extraction Times for supervised classification experiments.

5.4. Results Overview

Evaluating all experiments together, ODM stands out as it is the only particularly stable method (i.e, its performance are among the top ones regardless of the application) and keeps considerable low time-complexity costs.

Table 4
Supervised learning performance.

Algorithm	r	Accuracy	Mi. Prec.	Ma. Prec.	Mi. Rec.	Ma. Rec.	Ext. Time (s)	k-NN Time (s)
Baseline	——	0.94 ± 0.09	0.94 ± 0.09	0.88 ± 0.09	0.94 ± 0.09	0.84 ± 0.10	——	0.29 ± 0.28
ODM	0.5%	0.85 ± 0.18	0.85 ± 0.18	0.56 ± 0.30	0.85 ± 0.18	0.56 ± 0.31	0.63 ± 0.30	0.12 ± 0.06
RS		0.83 ± 0.19	0.83 ± 0.19	0.60 ± 0.30	0.83 ± 0.19	0.52 ± 0.32	0.00 ± 0.00	0.12 ± 0.06
SDO		0.75 ± 0.24	0.75 ± 0.24	0.46 ± 0.29	0.75 ± 0.24	0.51 ± 0.33	0.02 ± 0.01	0.11 ± 0.05
KMC		0.82 ± 0.19	0.82 ± 0.19	0.55 ± 0.34	0.82 ± 0.19	0.46 ± 0.27	4.88 ± 3.40	0.19 ± 0.08
WGM		0.65 ± 0.39	0.65 ± 0.39	0.54 ± 0.38	0.65 ± 0.39	0.45 ± 0.34	6.52 ± 7.34	0.09 ± 0.07
GNG		0.82 ± 0.19	0.82 ± 0.19	0.57 ± 0.32	0.82 ± 0.19	0.48 ± 0.25	31.30 ± 16.28	0.12 ± 0.06
CNN		0.63 ± 0.29	0.63 ± 0.29	0.41 ± 0.29	0.63 ± 0.29	0.50 ± 0.23	65.94 ± 115.28	0.11 ± 0.05
ODM	1%	0.88 ± 0.17	0.88 ± 0.17	0.63 ± 0.26	0.88 ± 0.17	0.59 ± 0.28	0.73 ± 0.44	0.14 ± 0.09
RS		0.85 ± 0.19	0.85 ± 0.19	0.62 ± 0.27	0.85 ± 0.19	0.56 ± 0.29	0.01 ± 0.01	0.14 ± 0.09
SDO		0.83 ± 0.24	0.83 ± 0.24	0.60 ± 0.29	0.83 ± 0.24	0.57 ± 0.30	0.04 ± 0.02	0.11 ± 0.05
KMC		0.86 ± 0.18	0.86 ± 0.18	0.62 ± 0.29	0.86 ± 0.18	0.54 ± 0.30	5.59 ± 3.79	0.19 ± 0.09
WGM		0.67 ± 0.39	0.67 ± 0.39	0.54 ± 0.38	0.67 ± 0.39	0.50 ± 0.35	12.16 ± 13.37	0.10 ± 0.08
GNG		0.84 ± 0.17	0.84 ± 0.17	0.60 ± 0.30	0.84 ± 0.17	0.53 ± 0.25	41.26 ± 22.74	0.12 ± 0.06
CNN		0.68 ± 0.29	0.68 ± 0.29	0.44 ± 0.26	0.68 ± 0.29	0.53 ± 0.21	66.64 ± 116.62	0.11 ± 0.05
ODM	5%	0.90 ± 0.14	0.90 ± 0.14	0.67 ± 0.25	0.90 ± 0.14	0.62 ± 0.25	0.64 ± 0.31	0.14 ± 0.08
RS		0.89 ± 0.16	0.89 ± 0.16	0.75 ± 0.26	0.89 ± 0.16	0.62 ± 0.26	0.00 ± 0.00	0.14 ± 0.08
SDO		0.64 ± 0.37	0.64 ± 0.37	0.50 ± 0.41	0.64 ± 0.37	0.44 ± 0.32	0.18 ± 0.14	0.12 ± 0.06
KMC		0.89 ± 0.15	0.89 ± 0.15	0.66 ± 0.25	0.89 ± 0.15	0.60 ± 0.23	10.38 ± 6.30	0.21 ± 0.11
WGM		0.63 ± 0.37	0.63 ± 0.37	0.53 ± 0.35	0.63 ± 0.37	0.56 ± 0.35	54.34 ± 58.35	0.11 ± 0.10
GNG		0.89 ± 0.13	0.89 ± 0.13	0.67 ± 0.25	0.89 ± 0.13	0.61 ± 0.25	106.06 ± 69.87	0.14 ± 0.08
CNN		0.85 ± 0.15	0.85 ± 0.15	0.67 ± 0.16	0.85 ± 0.15	0.69 ± 0.23	66.50 ± 116.09	0.14 ± 0.07
ODM	10%	0.91 ± 0.13	0.91 ± 0.13	0.78 ± 0.24	0.91 ± 0.13	0.64 ± 0.24	0.65 ± 0.31	0.16 ± 0.10
RS		0.90 ± 0.15	0.90 ± 0.15	0.77 ± 0.23	0.90 ± 0.15	0.64 ± 0.25	0.00 ± 0.00	0.15 ± 0.09
SDO		0.63 ± 0.36	0.63 ± 0.36	0.54 ± 0.35	0.63 ± 0.36	0.49 ± 0.30	0.38 ± 0.30	0.14 ± 0.08
KMC		0.90 ± 0.13	0.90 ± 0.13	0.77 ± 0.25	0.90 ± 0.13	0.61 ± 0.22	14.65 ± 8.91	0.23 ± 0.13
WGM		0.60 ± 0.36	0.60 ± 0.36	0.53 ± 0.34	0.60 ± 0.36	0.57 ± 0.35	104.27 ± 112.55	0.13 ± 0.12
GNG		0.91 ± 0.13	0.91 ± 0.13	0.76 ± 0.25	0.91 ± 0.13	0.63 ± 0.21	173.76 ± 122.99	0.16 ± 0.10
CNN		0.75 ± 0.30	0.75 ± 0.30	0.55 ± 0.27	0.75 ± 0.30	0.63 ± 0.22	66.16 ± 115.79	0.15 ± 0.09

Ext.T.: Time needed to build a coreset.
Ma.: Macro.
Mi.: Micro.

6. Conclusion

We have presented ODM, a lightweight algorithm for the extraction of low-density data models (or data summarizations) in log-linear times. We have tested ODM and compared with state-of-the-art alternatives in principal machine learning applications, i.e., anomaly detection, clustering, and supervised classification. For the comparison experiments we have used multiple datasets and data generators that are publicly available and have been proposed and published for algorithm testing. Results place ODM as the most stable option and best trade-off between time-complexity and performance accuracy, becoming a strongly recommended option as an intermediate step to alleviate requirements in the analysis of big data.

References

[1] A. Zimek, M. Gaudet, R. J. Campello, J. Sander, Subsampling for efficient and effective unsupervised outlier detection ensembles, in: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data

Mining, 2013, pp. 428–436.
[2] P. K. Agarwal, S. Har-Peled, K. R. Varadarajan, Geometric approximation via coresets, Combinatorial and computational geometry 52 (2005) 1–30.
[3] F. I. Vázquez, T. Zseby, A. Zimek, Outlier detection based on low density models, in: 2018 IEEE International Conference on Data Mining Workshops (ICDMW), 2018, pp. 970–979.
[4] T. Kohonen, The self-organizing map, Proceedings of the IEEE 78 (9) (1990) 1464–1480.
[5] T. Martinetz, K. Schulten, et al., A" neural-gas" network learns topologies.
[6] B. Fritzke, A growing neural gas network learns topologies, in: Advances in neural information processing systems, 1995, pp. 625–632.
[7] P. Hart, The condensed nearest neighbor rule (corresp.), IEEE transactions on information theory 14 (3) (1968) 515–516.
[8] F. Angiulli, Fast condensed nearest neighbor rule, in: Proceedings of the 22nd international conference on Machine learning, 2005, pp. 25–32.
[9] J. Huggins, T. Campbell, T. Broderick, Coresets for scalable bayesian logistic regression, in: Advances in Neural Information Processing Systems, 2016, pp. 4080–4088.
[10] T. Campbell, T. Broderick, Bayesian coreset construction via greedy iterative geodesic ascent, arXiv preprint arXiv:1802.01737.
[11] T. Campbell, T. Broderick, Automated scalable bayesian inference via hilbert coresets, The Journal of Machine Learning Research 20 (1) (2019) 551–588.
[12] T. Campbell, B. Beronov, Sparse variational inference: Bayesian coresets from scratch, arXiv preprint arXiv:1906.03329.
[13] O. Bachem, M. Lucic, A. Krause, Scalable k-means clustering via lightweight coresets, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 1119–1127.
[14] H. Attias, A variational baysian framework for graphical models, in: Advances in neural information processing systems, 2000, pp. 209–215.
[15] M. Lucic, M. Faulkner, A. Krause, D. Feldman, Training gaussian mixture models at scale via coresets, The Journal of Machine Learning Research (2017) 5885–5909.
[16] P. Fränti, R. Mariescu-Istodor, C. Zhong, Xnn graph, in: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Springer, 2016, pp. 207–217.
[17] M. Rezaei, P. Fränti, Set matching measures for external cluster validity, IEEE Transactions on Knowledge and Data Engineering 28 (8) (2016) 2173–2186.
[18] P. Fränti, M. Rezaei, Q. Zhao, Centroid index: cluster level similarity measure, Pattern Recognition 47 (9) (2014) 3034–3045.
[19] I. Kärkkäinen, P. Fränti, Dynamic local search algorithm for the clustering problem, University of Joensuu, 2002.
[20] M. data, mopsi-finland, url = https://github.com/ deric/ clustering-benchmark/ blob/ master/ src/ main/ resources/ datasets/ artificial/ mopsi-finland.arff, urldate = 2019-08-19.
[21] A. Ultsch, Clustering with som: U^* c, in: Proceedings of the workshop on self-organizing maps, 2005, 2005.
[22] J. L. Bentley, Multidimensional binary search trees used for associative searching, Communications of the ACM 18 (9) (1975) 509–517.
[23] P. Ciaccia, M. Patella, P. Zezula, M-tree: An e cient access method for similarity search in metric spaces, in: Proceedings of the 23rd VLDB conference, Athens, Greece, Citeseer, 1997, pp. 426–435.
[24] C. C. Aggarwal, A. Hinneburg, D. A. Keim, On the surprising behavior of distance metrics in high dimensional space, in: International conference on database theory, Springer, 2001, pp. 420–434.
[25] F. Iglesias, T. Zseby, D. Ferreira, A. Zimek, Mdcgen: Multidimensional dataset generator for clustering, Journal of Classification 1–20.
[26] J.-M. Geusebroek, G. J. Burghouts, A. W. Smeulders, The amsterdam library of object images, International Journal of Computer Vision 61 (1) (2005) 103–112.
[27] N. Abe, B. Zadrozny, J. Langford, Outlier detection by active learning, in: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2006, pp. 504–509.
[28] J. A. Blackard, D. J. Dean, Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables, Computers and electronics in agriculture 24 (3) (1999) 131–151.
[29] I. W. Evett, E. J. Spiehler, Rule induction in forensic science, KBS in Goverment (1987) 107–118.
[30] F. Keller, E. Muller, K. Bohm, Hics: High contrast subspaces for density-based outlier ranking, in: 2012 IEEE 28th international conference on data engineering, IEEE, 2012, pp. 1037–1048.
[31] K. Yamanishi, J.-I. Takeuchi, G. Williams, P. Milne, On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms, Data Mining and Knowledge Discovery 8 (3) (2004) 275–300.
[32] C. De Stefano, M. Maniaci, F. Fontanella, A. S. di Freca, Reliable writer identification in medieval manuscripts through page layout features: The âĂIJavilaâĂİ bible case, Engineering Applications of Artificial Intelligence 72 (2018) 99–110.
[33] B. A. Johnson, R. Tateishi, N. T. Hoan, A hybrid pansharpening approach and multiscale object-based image analysis for mapping diseased pine and oak trees, International journal of remote sensing 34 (20) (2013) 6969–6982.

[34] M. Keith, A. Jameson, W. Van Straten, M. Bailes, S. Johnston, M. Kramer, A. Possenti, S. Bates, N. Bhat, M. Burgay, et al., The high time resolution universe pulsar survey–i. system configuration and initial discoveries, Monthly Notices of the Royal Astronomical Society 409 (2) (2010) 619–627.
[35] L. M. Candanedo, V. Feldheim, Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models, Energy and Buildings 112 (2016) 28–39.
[36] P. Cortez, A. Cerdeira, F. Almeida, T. Matos, J. Reis, Modeling wine preferences by data mining from physicochemical properties, Decision Support Systems 47 (4) (2009) 547–553.
[37] S. Ramaswamy, R. Rastogi, K. Shim, Efficient algorithms for mining outliers from large data sets, in: Proceedings of the 2000 ACM SIGMOD international conference on Management of data, 2000, pp. 427–438.
[38] G. O. Campos, A. Zimek, J. Sander, R. J. Campello, B. Micenková, E. Schubert, I. Assent, M. E. Houle, On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study, Data Mining and Knowledge Discovery 30 (4) (2016) 891–927.
[39] D. Arthur, S. Vassilvitskii, k-means++: The advantages of careful seeding, Tech. rep., Stanford (2006).
[40] L. Hubert, P. Arabie, Comparing partitions, Journal of classification 2 (1) (1985) 193–218.