Journal of Big Data

METHODOLOGY Open Access

DENCAST: distributed density-based clustering for multi-target regression

Roberto Corizzo[1,2*†], Gianvito Pio[1,2†], Michelangelo Ceci[1,2†] and Donato Malerba[1,2]

*Correspondence: roberto.corizzo@uniba.it
†Roberto Corizzo, Gianvito Pio and Michelangelo Ceci contributed equally to this work
[1] Department of Computer Science, University of Bari Aldo Moro, Via Orabona, 4, Bari, Italy
Full list of author information is available at the end of the article

Abstract
Recent developments in sensor networks and mobile computing led to a huge increase in data generated that need to be processed and analyzed efficiently. In this context, many distributed data mining algorithms have recently been proposed. Following this line of research, we propose the DENCAST system, a novel distributed algorithm implemented in Apache Spark, which performs density-based clustering and exploits the identified clusters to solve both single- and multi-target regression tasks (and thus, solves complex tasks such as time series prediction). Contrary to existing distributed methods, DENCAST does not require a final merging step (usually performed on a single machine) and is able to handle large-scale, high-dimensional data by taking advantage of locality sensitive hashing. Experiments show that DENCAST performs clustering more efficiently than a state-of-the-art distributed clustering algorithm, especially when the number of objects increases significantly. The quality of the extracted clusters is confirmed by the predictive capabilities of DENCAST on several datasets: It is able to significantly outperform (p-value < 0.05) state-of-the-art distributed regression methods, in both single and multi-target settings.

Keywords: Distributed clustering, Multi-target regression, Apache Spark

Introduction

The generation of massive amounts of data in different forms (such as activity logs and sensor measurements) has increased the need for novel data mining algorithms, which are capable of building accurate models efficiently and in a distributed fashion. In recent years, several researchers proposed novel approaches to distribute the workload among several machines for classical clustering, classification and regression tasks [1]. However, only a few of them tackle the specific problem of density-based clustering. This problem has received much attention in the last decades, because of many desirable properties of the extracted clusters (arbitrarily-shaped, noise-free, robustness to outliers) which turn out the be useful in many application domains (e.g., spatial data analysis).

Starting from the seminal work of DBSCAN [2], many algorithms have been proposed, but only a few of them are distributed. Unfortunately, existing distributed methods for density-based clustering suffer from several limitations. In particular, they are limited to data organized in a specific structure (e.g., they can analyze only low-dimensional feature spaces), or they suffer from overhead and scalability issues when the number of instances and attributes increase considerably [3–5]. These limitations depend from the

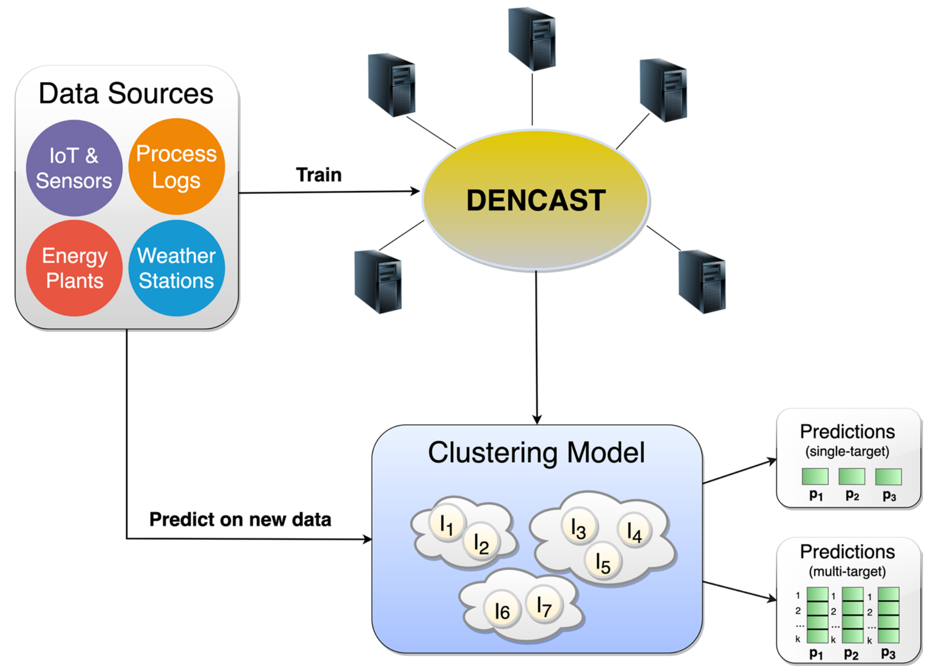

Fig. 1 Overview of the environment in which DENCAST works. In the figure, data coming from multiple sources are fed into DENCAST, which produces a clustering model that is exploited for predictive purposes on new data (single-target or multi-target regression). DENCAST runs on multiple computational nodes, in a distributed fashion

inherent difficulty in upgrading existing non-distributed density-based clustering algorithms towards their equivalent (or, at least, approximated) distributed counterpart. Finally, most of the existing methods are strictly tailored for pure clustering and do not exploit clusters to support predictive tasks, as in predictive clustering trees [6].

Therefore, our research focused on the following questions: can we perform density-based clustering on large-scale and high dimensional data, without incurring in computational bottlenecks? Can we profitably exploit these clusters for predictive purposes? To answer to these questions, we propose DENCAST, which simultaneously solves all the issues mentioned above. Specifically, it is a novel density-based clustering algorithm, implemented in the Apache Spark framework, which is able to handle large-scale, high-dimensional data. The proposed approach exploits the identified clusters, built on labeled data, to predict the value assumed by one or more target variables of unlabeled objects in an inductive, supervised learning setting. This characteristic allows the proposed method to solve any single- or multi-target predictive task. In this paper, we focus on single- and multi-target regression tasks, which are central in several real-world applications (see Fig. 1 for a graphical overview of the environment in which DENCAST works). For example, solving a multi-target regression task can be useful in energy planning and trading from renewable sources, such as photovoltaic or wind plants [7]. In this context, multi-step ahead forecasting (usually 24 h) is necessary to predict the energy produced by renewable sources, in order to minimize the production from polluting sources and possible money losses [8]. Other domains where multi-target regression finds application include traffic flow forecasting [9], air quality forecasting [10], bike demand forecasting [11, 12], life sciences (e.g., predicting the toxicity of molecules) and ecology (e.g., analysis of remotely sensed data, habitat modelling) [13]. The peculiarities of data in such application domains further motivate the adoption of the predictive clustering framework in this paper. Indeed, not only several studies in the literature proved the effectiveness of predictive clustering frameworks [6, 14–16], but it has shown to be particularly appropriate when data exhibit different forms of autocorrelation [17], i.e., objects which are close to each other (spatially, temporally, or in a network) appear more related than distant objects. Such phenomena are commonly present in data regarding the cited domains and approaches based on clustering can naturally detect them.

Therefore, the main contribution of this paper consists in a method for distributed density-based clustering which, contrary to existing works (see "Distributed methods for density-based clustering" and "Distributed methods for multi-target regression" sections), simultaneously shows all the following key features:

- It works on the neighborhood graph. In this way, the algorithm needs only object IDs and their neighborhood relationships (instead of their initial, possibly high-dimensional, representation) and thus it requires limited space resources. We build such a neighborhood graph efficiently from high-dimensional data through the locality-sensitive hashing (LSH) method [18].
- It is implemented in the Apache Spark framework and it is fully distributed. Therefore, it does not require pre-processing or post-processing steps, usually performed on a single machine (see " Distributed methods for density-based clustering" section for details about this aspect in other methods). This aspect allows our method to analyze large-scale datasets without incurring in computational bottlenecks.
- The identified density-based clusters can be exploited to predict the value of one or more target variables, by means of a density- and distance-based approach. The result is that the proposed method can be adopted to solve single-target and multi-target regression tasks in a distributed setting.

Overall, we propose a distributed density-based clustering algorithm that is capable to (i) handle large-scale data; (ii) deal with the high dimensionality of data; (iii) exploit the identified clusters to perform predictions in both single-target and multi-target settings. To the best of our knowledge, existing methods are limited in one or more of these aspects, or are not able to address all of them simultaneously.

In "Background" section, we introduce some background notions and briefly review existing methods that are related to this paper. In "Method" section, we propose our distributed density-based (predictive) clustering method, while in "Time complexity analysis" section we analyze its time complexity. In "Results and discussion" section, we describe the experimental evaluation, showing that our method obtains accurate predictions and appears efficient in dealing with massive amounts of high-dimensional data. Finally, in "Conclusion" section we draw some conclusions and outline future work.

Background

The pioneer density-based clustering approach in the literature is DBSCAN [2]. This approach is able to identify arbitrarily shaped clusters (i.e., not only spherical) without requiring the number of clusters to be extracted as an input parameter. However, it requires two other parameters, i.e., *eps* and *minPts*.

Since several concepts that characterize density-based algorithms are in common with those adopted in this paper, we recall some useful notions:

- The neighborhood $N(p)$ of an object p is defined as the set of objects whose distance from p, according to a given measure, is within the threshold *eps*. Formally, $N(p) = \{q \mid dist(p, q) < eps\}$.
- An object p is a *core object* w.r.t. *eps* and *minPts* if it has at least *minPts* objects in its neighborhood $N(p)$. Formally, p is a core object if $|N(p)| \geq minPts$.
- An object p is *directly density-reachable* from an object q if $p \in N(q)$ and q is a core object.
- An object p_w is *density-reachable* from an object p_1 if there exists a chain of objects $p_1, p_2, \ldots, p_w$, such that for each pair of objects $\langle p_i, p_{i+1}\rangle$, p_{i+1} is directly density-reachable from p_i w.r.t. *eps* and *minPts*.
- An object p is *density-connected* to an object q if there exists an object o, such that both p and q are density-reachable from o w.r.t. *eps* and *minPts*.
- A *cluster* is a non-empty subset of objects, where each pair of objects $\langle p, q\rangle$ is density-connected.
- Non-core objects belonging to at least one cluster are called *border objects*, whereas objects not belonging to any cluster are considered *noise objects*.

Specifically, DBSCAN starts with an arbitrary object o and, if this is a core object, retrieves all the objects which are density-reachable from it w.r.t. *eps* and *minPts*, returning a cluster. The algorithm then proceeds with the next unclustered object. Other density-based methods follow a slightly different approach. For example, Density Peaks Clustering (DPC) [19] follows a hybrid workflow which takes inspiration from both centroid-based and density-based methods. In particular, it selects some objects as cluster centroids, and subsequently assigns other objects to centroids according to the fact that they locally show density peaks. The algorithm identifies density peaks according to two indicators: a local density indicator, which corresponds to the concept of *eps*-neighborhood in DBSCAN, and the maximum similarity (or minimum distance) indicator, computed between the current object and any object with higher local density. Similarly, Mean Shift [20] selects an object, identifies a circle of a pre-defined radius, computes the centroid of objects which fall within the radius, and it moves its center towards it. This process, which works iteratively, allows the algorithm to find a local maximum (in terms of density) for each object, and to group objects that appear to be tied to the same local maximum. Mean Shift does not require to specify the number of clusters to be extracted, coherently with other density-based clustering algorithms. However, it requires to specify the radius, and its time complexity is $O(n^2 \cdot I)$, where n is the number of objects and I is the number of iterations, which can be considered high for high-dimensional and large-scale data.

These density-based methods, even if they follow different approaches, are able to identify accurate and arbitrary shaped clusters, and are almost independent of the order of the analysis of objects. Moreover, many variants available in the literature aim to adapt density-based clustering algorithms (in particular DBSCAN) to specific contexts or to overcome limitations on time and space complexity. Regarding this aspect, in "Distributed methods for density-based clustering" section we briefly review existing works focusing on novel strategies to make density-based approaches applicable to large datasets. Moreover, since this paper has its roots also in methods for multi-target regression, in "Distributed methods for multi-target regression" section we briefly describe some related works in this field.

Distributed methods for density-based clustering

Although in the literature we can find some existing clustering algorithms which are able to handle large-scale and/or high-dimensional data [21–23], only few of them are density-based. The first attempts focused on extensions or variations of the well-known DBSCAN algorithm. The first extensions of DBSCAN concern the estimation of the optimal value of the input parameters *eps* and *minPts* [24] and its applicability to different contexts, such as data streams [25] and spatio-temporal data [26]. Due to the necessity to process large, high-dimensional datasets, more recent works have focused on the optimization of the time complexity, which is originally $O(n^2 \cdot m)$, where n is the number of objects and m is the number of features. Since the dominating phase is the identification of the neighborhood of all nodes (which time complexity is, thus $O(n^2 \cdot m)$), in [27] the authors proposed to adopt the locality-sensitive hashing (LSH) [18] to perform this phase in $O(n \cdot m)$. Although this method is able to process high-dimensional data, it cannot distribute the workload both in time and space to multiple machines and, consequently, cannot scale in the presence of distributed architectures, thus limiting the possibility of handling large datasets.

In [3], the authors proposed a distributed variant of DBSCAN for MapReduce, which exploits an R-Tree-based index to compute the distance among objects. However, R-Tree-based indexes are not efficient with high-dimensional data [28], due to a high overlap among bounding boxes. For this reason, experiments are limited to 2D datasets. In [4] and [5], the authors proposed a variant of DBSCAN, named RDD-DBSCAN, implemented in Apache Spark. RDD-DBSCAN consists of three phases: data partitioning, local clustering and global labeling. The algorithm takes as input the same parameters as DBSCAN, and defines a bounding rectangle for the whole dataset. Subsequently, the algorithm splits this rectangle into two parts, containing approximately the same number of data points. The resulting partitions are clustered locally on executor nodes. In the global labeling phase, RDD-DBSCAN examines all the points that are within a specified distance (*eps*) of the borders of the bounding rectangle of each partition. If two clusters contain some common objects, the algorithm assumes they are the same cluster. Given these characteristics, these two works show the same limitations of the method proposed in [3], i.e., experiments are limited to 2D data. Moreover, one common limitation of [3–5], also present in the density-based approach proposed in [29], is the necessity of a merging phase which aggregates partial results obtained by the worker machines. This phase usually takes place on a single (driver) machine and can, in principle, require a complexity $\mathcal{O}(n^2)$, possibly leading to a significant increase of the overall running time.

Our method faces all the issues raised by large-scale, high-dimensional datasets. In particular, we propose an approach which is computationally efficient and distributed in all its steps, leading to the easy handling of large-scale datasets, and that, inspired by the work in [27], adopts locality-sensitive hashing to handle high-dimensional data.

Distributed methods for multi-target regression

In the literature, and specifically in the field of Structured Output Prediction, researchers paid much attention to the multi-target regression task [13], i.e., to the learning of regression models for multiple target attributes. The easiest way to solve this task consists in the application of methods for single-target regression for each target attribute, independently (local models). In this way, almost all the existing approaches allow to perform multi-target regression, even if they require an adaptation step. Focusing on methods for processing large-scale datasets, in the literature we can find some distributed approaches for single-target regression (elastic net regularized linear regression [30] and isotonic regression [31]) that can be adapted to multi-target regression tasks.

A recent survey [32] highlighted the advantages and disadvantages of prediction algorithms in parallel multicore systems. Some simple algorithms, such as AutoRegressive Integrated Moving Average (ARIMA) [33], k-nearest neighbors and linear regression, show a moderate computational cost and good prediction performances in many scenarios when the task is that of prediction or forecasting with a limited time horizon. Other algorithms, such as Neural Networks and deep neural networks [34], show a higher predictive accuracy and the ability to consider nonlinearity in the in data, at the cost of a higher computational complexity.

More complex approaches for multi-target regression learn a global model which is able to predict the value of all the target attributes as a whole. Since these approaches specifically perform multi-target regression, i.e., they can exploit possible dependencies among the target attributes, usually lead to better predictive performance (see [35] for an example showing the superiority of global methods in the case of predictive clustering trees).

Statistical approaches can be considered as the first attempt to deal with the simultaneous prediction of multiple real-valued target attributes [36]. Subsequent attempts focused on extending support vector regression (SVR) models in order to allow them to manage multiple target variables. For example, in [37] the authors developed a vector-valued SVR (i.e., able to predict a vector of numeric values) by adapting the concepts of estimator, loss function and regularization function from the scalar-valued case to the vector-valued case. Another recent approach [38] proposed to extend the least squares SVR to the multi-target setting.

Alternative approaches (see [35] and [39]) proposed a multi-target variant of regression trees, which exploits possible correlations among the different target attributes. Moreover, it is noteworthy that neural networks and deep learning algorithms can naturally be applied to the multi-target setting, by defining the output layer with multiple neurons. In this class of methods, it is worthy to mention the long short-term neural

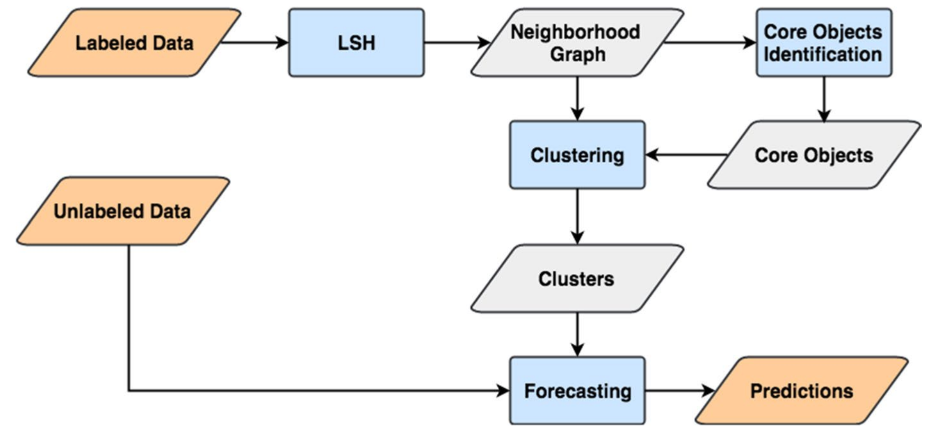

Fig. 2 Workflow of the proposed method. Orange blocks represent inputs (labeled and unlabeled objects) and outputs (predictions); grey blocks represent intermediate results; blue blocks represent the different phases of the proposed method

networks [40], which are particularly powerful when data describe seasonal and recurrent phenomena characterized by temporal correlations.

However, to the best of our knowledge, global methods for multi-target regression that are distributed, and therefore able to process large-scale, high-dimensional datasets, are still scarcely available in the literature. An exception is the implementation of the ARIMA models [33], available in the Spark-TS library,[1] which, however, is tailored for the analysis of time series. In particular, the different target variables regard the same feature predicted in different time instants in the future.

Moreover, recently, researchers put a significant effort to the adaptation of deep learning algorithms towards distributed frameworks, such as Apache Spark. Important examples are *DeepLearning4J*,[2] *Elephas*[3] and *TensorFlowOnSpark*[4] which provide straightforward approaches to distribute: (i) the data during the training phase, (ii) the workload in the hyper-parameter optimization, or (iii) the learning of ensembles of models.

Method

On the basis of the notions introduced in "Background" section, in this section we describe our method, which general workflow is depicted in Fig. 2 and formalized in Algorithm 1. Note that, since our method is implemented in the Apache Spark framework, we adopt the Resilient Distributed Dataset (RDD) data structure and its operations (see [41] for details).

[1] github.com/sryza/spark-timeseries.

[2] deeplearning4j.org/docs/latest/deeplearning4j-scaleout-intro.

[3] github.com/maxpumperla/elephas.

[4] github.com/yahoo/TensorFlowOnSpark.

```
Algorithm 1: Main Algorithm
Data:
    · $m$: Integer
        number of (non-target) descriptive attributes
    · $k$: Integer
        number of target attributes
    · $A_L$: RDD[Node, Vector]
        labeled objects represented by a node and an $(m+k)$-dimensional feature vector
    · $A_U$: RDD[Node, Vector]
        unlabeled objects represented by a node and an $m$-dimensional feature vector
    · $minPts$: Integer $\in [1, |A_L|]$
        min number of neighboring objects for core objects
    · $labelChangeRate$: Double $\in [0, 1]$
        min percentage of propagations to perform an iteration
Result:
    · $pred$: RDD[Node, Vector]
        unlabeled objects associated with predictions for the $k$ target attributes
1 begin
        /* Neighborhood graph identified by LSH. Edges are represented as pairs of
           nodes $\langle src, dst \rangle$                                          */
2       $\langle V : RDD[Node], E : RDD[Node, Node] \rangle = LSH(A_L)$;
        /* Identify the neighbours of each node                                      */
3       $objNeig$: RDD[Node, Set[Node]] = $E$.aggregateByKey();
        /* Identify core objects (i.e., those having at least minPts neighbours)    */
4       $cores$: RDD[Node] = $objNeig$.filter{case$(node, neig) \rightarrow neig$.size $\geq minPts$}.keys;
        /* Identify the clusters                                                     */
5       $clusters$: RDD[Node, ClusterID] = $findClusters(V, E, cores, labelChangeRate)$;
        /* Remap each node to its features                                           */
6       $model$: RDD[Node, ⟨Vector, ClusterID⟩] = $A_L$.join(clusters)
        /* Exploit the clusters to predict the value of non-target attributes for
           unlabeled objects                                                         */
7       $pred$: RDD[Node, Vector] = $predict(m, k, model, A_U)$
8       return $pred$
9 end
```

Given the dataset A_L consisting of n labeled objects represented by $m + k$ attributes (m descriptive attributes and k target attributes), we first apply a distributed variant of locality-sensitive hashing—LSH [42] (line 2) to identify an approximate neighborhood graph. The obtained graph consists of a node for each labeled object and an undirected edge for each pair of nodes $\langle u, v \rangle$, which appear similar enough according to the representation obtained after the application of the LSH algorithm and a threshold *minSim*. This step and the specific details about the distributed variant adopted are described in "Identification of the neighborhood graph" section. From this point, the algorithm only uses the neighborhood graph, which can be considered an approximate representation of the objects and their distances, instead of objects represented in the original feature space. This design choice, which has been conveniently adopted by several clustering algorithms (see for example [43]), reduces significantly the space and the time necessary for the next steps: identification of the neighbours of each node (line 3) and identification of core objects (line 4), i.e., those having at least *minPts* nodes in their neighborhood.

Our method for density-based clustering then maps each labeled node to a cluster (line 5), by propagating cluster IDs from core objects through their neighbors.

As we will describe in "Density-based clustering" section, our approach is iterative and requires a stopping criterion, based on a threshold (*labelChangeRate*), aiming to avoid unnecessary iterations, which would lead to slight changes in cluster assignments. It is noteworthy that not all the objects will be necessarily assigned to a cluster, i.e., similarly to existing density-based clustering algorithms, our algorithm is able to discard objects that can be considered noise or outliers, since they are too far in the feature space from the identified clusters.

Finally, we re-associate all the nodes with the original features of the corresponding objects (line 6) and exploit the identified clusters to predict the value of the target attributes for all the nodes of a set of unlabeled objects A_U (line 7). The prediction step is described in detail in "Exploiting clusters for multi-target regression" section.

Identification of the neighborhood graph

As we mentioned in "Method" section, we adopt a distributed variant of the locality-sensitive hashing (LSH) method to efficiently identify an approximate neighborhood graph, which will then be exploited by our clustering algorithm. LSH hashes objects so that similar objects map to the same buckets with a high probability (where the number of buckets is much smaller than the number of analyzed objects). Contrary to conventional hash functions, LSH maximizes the probability of collision for similar objects [44]. LSH exploits some properties of the cosine similarity: given two objects represented as $(m + k)$-dimensional vectors, the probability of a random hyperplane to correctly separate them increases as the angle between them increases [42]. Accordingly, the computation of the neighborhood of each node in A_L through LSH is reformulated as follows:

- Generate r random $(m + k)$-dimensional hyperplanes, where $r \ll (m + k)$;
- Represent each object $p \in A_L$ as an r-dimensional bit stream p_r, where the i-th feature is 0 or 1 according to the side of the i-th hyperplane which p falls into;
- Generate *numPerm* random permutations of r elements. For each permutation, permute the bit stream of all the objects in A_L (each object is represented by *numPerm* bitstreams). Bitstreams, for each permutation, are then sorted lexicographically.
- Find the set $\tilde{N}(p)$ of the B nearest neighbors of each object p in every sorted list and compute the Hamming distance between the bitstream of p and the bitstream of the objects in $\tilde{N}(p)$. Every object $q \in \tilde{N}(p)$ having a Hamming distance with p smaller than a given threshold *minSim* is included in $N(p)$.

The implementation we adopt[5] is the distributed variant proposed in [42]. Such a variant identifies $N(p)$ by replacing the Hamming distance with the exact cosine similarity, which avoids the presence of false positives (objects detected as neighbors, that actually are not). Formally: $N(p) = \{q | q \in \tilde{N}(p) \wedge cosine(p, q) \geq minSim\}$. Although other variants of LSH, based on different similarity/distance measures are available in the literature [45], they mainly exploit the Euclidean distance on the unit sphere, which actually corresponds to the cosine similarity. Moreover, their adoption would require an

[5] https://github.com/soundcloud/cosine-lsh-join-spark.

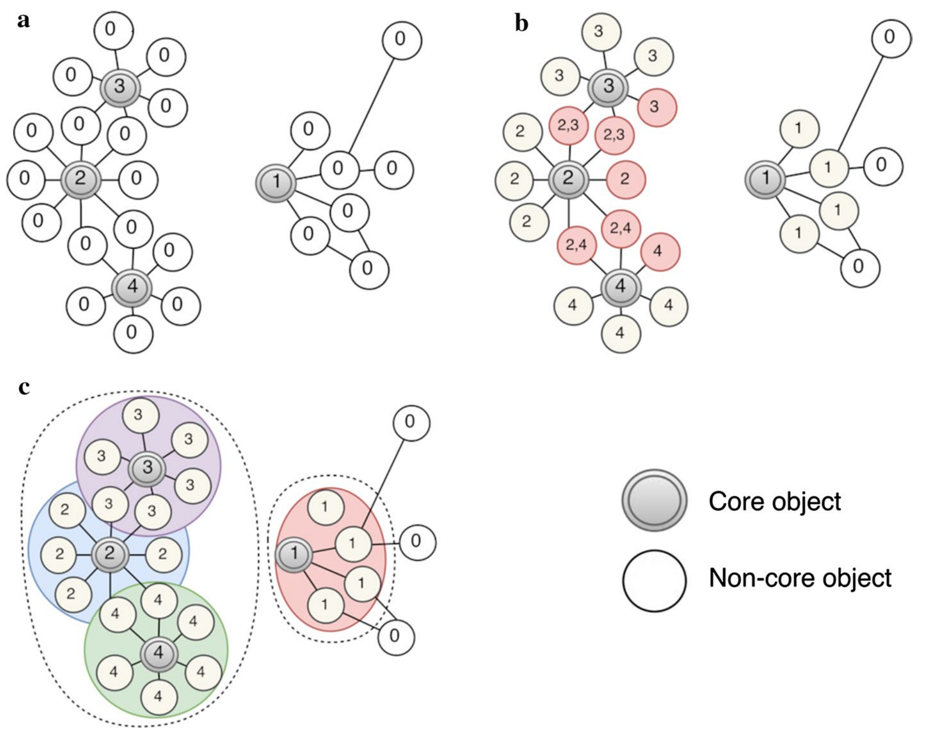

Fig. 3 Graphic representation of the clustering approach. **a** Initialization of cluster IDs. **b** Core objects propagate their cluster ID to their neighbors (*map* phase). Nodes highlighted in red receive multiple messages. **c** Multiple cluster IDs received by the same node are aggregated (*reduce* phase). Dotted lines represent the final clustering result, after some iterations

additional step to normalize the values in [0,1], leading to introduce possible approximation errors.

Density-based clustering

In this section, supported by the pseudo-code of Algorithm 2 and by Fig. 3, we describe our distributed density-based clustering method. Our implementation exploits GraphX APIs[6] of Apache Spark to analyze the neighborhood graph identified by LSH. GraphX [47] internally represents graphs through a collection of vertices and a collection of edges, built on top of the Spark RDD. The *vertex collection* is hash-partitioned by vertex IDs and supported by a local hash index in each partition, which facilitates frequent joins across vertex collections. The *edge collection* is horizontally partitioned and supported by a routing table that enables the efficient lookup of edges according to their source and target vertices. GraphX also adopts specific strategies to reduce network costs and to avoid unnecessary movements of unchanged data in subsequent iterations. Additional details can be found in [47]. Interestingly, GraphX provides the counterpart of the most common primitives of Spark RDDs for graph processing, such as *map*, *filter* and *aggregate*, which are exploited by our method in order to manipulate distributed graphs.

[6] Although other APIs for graph analysis have been recently proposed [46], they can only improve the efficiency when there is an unidirectional value propagation. Since, in our case, propagation can happen in both directions, GraphX appeared the most appropriate approach, since directly integrated within Apache Spark.

Algorithm 2: *findClusters(V,E,cores,labelChangeRate)*. Distributed density-based clustering.

```
Data:
    · $V$: RDD[Node]
        nodes of the neighborhood graph
    · $E$: RDD[Node, Node]
        edges of the neighborhood graph (as pairs $\langle src, dst \rangle$)
    · $cores$: RDD[Node]
        core objects
    · $labelChangeRate$: Double $\in [0, 1]$
        min percentage of propagations for a new iteration
Result:
    · $clusters$: RDD[Node, ClusterID]
        labeled objects associated to their cluster ID
1  begin
       /* Initialization of the cluster IDs */
2      $clusterID = 0$
3      $clusters$: RDD[Node, ClusterID] = $V$.map
4          {$v \rightarrow$ if $v \in cores$ then $\langle v, (clusterID \mathrel{+}= 1) \rangle$ else $\langle v, 0 \rangle$}
       /* Propagation of cluster IDs */
5      $threshold = labelChangeRate \cdot |E|$
6      repeat
7          $propagations = 0$
           /* Map phase */
8          $clusters = E$.map
9            {case $(\langle src, srcID \rangle, \langle dst, dstID \rangle) \rightarrow$
10             if $src \in cores$ and $srcID > dstID$ then
11                 $propagations = propagations + 1$
12                 $\langle dst, srcID \rangle$
13             else
14                 $\langle dst, dstID \rangle$}
           /* Reduce phase */
15         $clusters = clusters$.reduceByKey{case$(ID1, ID2) \rightarrow max(ID1, ID2)$}
16     until $propagations < threshold$
17     return $clusters$
```

The novelty of the proposed method relies on the formulation of a density-based clustering method that performs the exploration of the neighborhood graph, through the GraphX programming primitives, in a fully distributed way. We stress this last aspect, since, contrary to existing methods [3–5], it does not require a merging phase at the end, usually performed on a single driver machine.

We recall that the original DBSCAN implementation [2] identifies a cluster starting from an arbitrary core object and retrieving all objects which are density-reachable from it w.r.t. *eps* and *minPts*. Our approach aims at identifying all the reachable nodes of all the core objects simultaneously. This is performed by propagating the cluster assignment of all the core objects to their neighbors, until the cluster assignment appears stable enough.

The first step consists in assigning a different cluster ID to each core object. Non-core objects are associated with 0 (lines 2–4). Then we start a process which, as mentioned in "Method" section, iterates until a criterion based on the number of propagations is not satisfied. In particular, we stop the iterative process when the number of propagated IDs is below a given percentage (*labelChangeRate*) of the number of edges of the neighborhood graph (line 5). This strategy avoids the execution of additional iterations that would only lead to slight changes in cluster assignments.

Algorithm 3: *findClusters(V,E,cores,labelChangeRate)*. Pseudo-code description.

```
Data:
    · $V$: nodes of the neighborhood graph
    · $E$: edges of the neighborhood graph (as pairs $\langle src, dst \rangle$)
    · $cores$: core objects
    · $labelChangeRate$: min percentage of propagations to perform a new iteration
Result:
    · $clusters$ labeled objects associated with their cluster ID
1  begin
       /* Initialization of cluster IDs */
2      $clusters \leftarrow V$
3      $clusterID \leftarrow 1$
4      foreach $v \in clusters$ do
5          if $v \in cores$ then
6              $v.clusterID \leftarrow clusterID$
7              $clusterID \leftarrow clusterID + 1$
8          else
9              $v.clusterID \leftarrow 0$
       /* Propagation of cluster IDs */
10     $threshold \leftarrow labelChangeRate \cdot |E|$
11     repeat
12         $propagations \leftarrow 0$
           /* Reset messages for all the nodes */
13         foreach $v \in clusters$ do
14             $v.messages \leftarrow \emptyset$
           /* Each node receives multiple messages from (core, clustered)
              neighboring nodes */
15         foreach $e = \langle src, dst \rangle \in E$ do
16             if $src \in cores$ and $src.clusterID > dst.clusterID$ then
17                 $dst.messages \leftarrow dst.messages \cup \{src.clusterID\}$
18                 $propagations \leftarrow propagations + 1$
           /* Aggregate messages received by the nodes */
19         foreach $v \in V$ do
20             $v.clusterID \leftarrow max(v.messages)$
21     until $propagations < threshold$
22     return $clusters$
```

Each iteration consists in the propagation of the cluster ID from all the core objects towards their neighbors. To this aim, we perform a *map* phase which works on the set of edges of the neighborhood graph (lines 8–14). In particular, for each edge $\langle src, dst \rangle$,[7] we propagate the cluster ID of the node *src* towards the node *dst* if *src* is a core object and if its current cluster ID is higher than the cluster ID of the object *dst*.[8] This choice guarantees a deterministic behaviour of our approach, as well as its convergence. Moreover, similarly to [46], it leads to a reduction of the number of messages possibly exchanged among different machines in the cluster.

After propagation, each node receives multiple cluster IDs from its neighboring core objects. Therefore, the final step of each iteration consists of a *reduce* phase (line 15),

[7] We remind that the neighborhood graph is undirected. The identifiers *src* and *dst* are used only to distinguish between the two nodes involved in the link. This means that, in the algorithms, $\forall p, q \in V$ the edge $\langle p, q \rangle$ is interchangable with (equivalent to) the edge $\langle q, p \rangle$.

[8] It is noteworthy that this is only an implementation choice. Indeed, we could propagate the lowest cluster IDs without any change in the final clustering result, since the density-connection property that we catch from the neighborhood graph is symmetric and transitive.

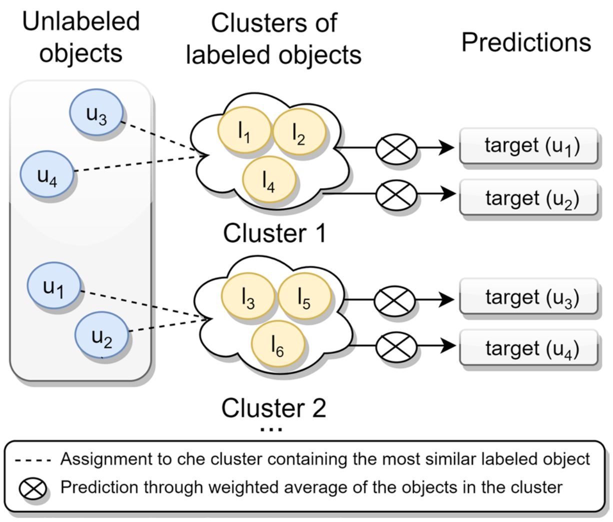
Unlabeled objects
Clusters of labeled objects
Predictions
u3
u4
u1
u2
l1
l2
l4
l3
l5
l6
Cluster 1
Cluster 2
target (u1)
target (u2)
target (u3)
target (u4)
Assignment to che cluster containing the most similar labeled object
Prediction through weighted average of the objects in the cluster

Fig. 4 Graphical representation of the prediction phase of our approach. Each unlabeled object is assigned to the cluster containing the most similar labeled object (dotted lines) and the value of its target attributes are predicted according to a weighted average of the values assumed by the labeled objects in such a cluster (crossed circle)

which aggregates the set of received cluster IDs into a single cluster ID. Coherently with the approach adopted during the *map* phase, each node will be assigned to the highest cluster ID received (see footnote 8). This leads, after some iterations, to collapse neighboring clusters in a single cluster, that is, the cluster with the highest ID. This makes the final merging phase, typically performed by existing distributed clustering methods on a single driver node, not necessary.

In Algorithm 3, we report a higher-level (non-parallel) pseudo-code description of Algorithm 2, from which it is possible to observe the performed steps also without going into details about the Apache Spark primitives.

An example of an iteration performed by our density-based clustering method can be observed in Fig. 3.

Exploiting clusters for multi-target regression

In this section, we describe the strategy we adopt to exploit the identified clusters to solve both single-target and multi-target regression tasks (see Fig. 4). Formally, given the set of unlabeled objects A_U, we aim at predicting the value of the target attributes for each object in A_U. Inspired by other solutions which exploit clusters for predictive purposes [15] we estimate, for each unlabeled object $u \in A_U$, the cluster $c(u)$ to which u ideally belongs: that is, the cluster to which u would have been assigned, if it had been known during the clustering process. We identify the most similar labeled object l and assign u to its cluster, i.e. $c(u) = c(l)$. Formally:

$$c(u) = c\big(argmin_{l \in A_L}\ cosineSim\big(u, l_{[1:m]}\big)\big), \tag{1}$$

where $cosineSim(\cdot,\cdot)$ is the cosine similarity between two vectors and $l_{[1:m]}$ is the sub-vector of the object l, consisting of only the descriptive attributes. Finally, we predict the value of the target attributes of each unlabeled object u by computing the average of the values associated to the target attributes of all the objects falling in the cluster $c(u)$, weighted according to their similarity with u. Formally:

$$u_{[m+1:m+k]} = \frac{\sum_{l \in A_L(c(u))} cosineSim(u, l_{[1:m]}) \cdot l_{[m+1:m+k]}}{\sum_{l \in A_L(c(u))} cosineSim(u, l_{[1:m]})}, \tag{2}$$

where $l_{[m+1:m+k]}$ is the sub-vector of the object l, consisting of only the target attributes, $A_L(c(u))$ is the subset of objects of A_L falling in $c(u)$ and $u_{[m+1:m+k]}$ is the part of the vector u reserved for target values.

Note that, this approach is coherent with the philosophy of density-based clustering because it extends the concept of density-connection to unlabeled examples.

More details about the implementation of the prediction phase are formalized in Algorithm 4. In particular, we first compute the cosine similarity between all the unlabeled objects in A_U and all the labeled objects in A_L (lines 2–8). Then we associate each unlabeled object to the cluster in which its most similar labeled object falls (lines 9–17). Finally, we predict the value of the target attributes of each unlabeled object according to Eq. 2 (lines 18–33).

As for the clustering phase, also this phase is fully distributed. Here, the idea is to perform an incremental computation of the weighted average, which mainly consists of a *map* phase (lines 25–27) and a *reduce* phase (lines 28–30). Such a computation also exploits the operators: $+$ between two vectors, which distributedly computes their element-wise sum and $*$ (resp. $/$) between a vector and a scalar, which distributedly computes the multiplication (resp. division) of each element of the vector by the scalar.

Time complexity analysis

In this section, we analyze the time complexity of the proposed method. First, we consider the time complexity of the training phase, by following the steps of the main algorithm (Algorithm 1) and of the clustering algorithm (Algorithm 2).

In particular, the first step performed by our method is the identification of the neighborhood graph through LSH (Algorithm 1 , line 2), which has a time complexity of $O(|V| \cdot m)$ [18], where $|V|$ is the number of nodes (i.e., objects) and m is the number of features. Next, the identification of the neighborhood of each object (Algorithm 1, line 3) requires a scan of the whole neighborhood graph, i.e., $O(|V| \cdot B)$ operations, where B is the maximum number of neighboring objects identified by LSH for each node. Given the neighborhood of each object, the identification of core objects (Algorithm 1, line 4) only requires a single scan of all the objects, leading to a complexity of $O(|V|)$. Finally, after executing the density-based clustering algorithm, a final join operation (Algorithm 1, line 6) is performed. This join operation, since the used data structures (i.e., paired RDD) are indexed on the node identifier, has a time complexity of $O(2 \cdot |V|)$. Therefore, the pre-processing and the post-processing steps of our main clustering algorithm require an overall time complexity of:

$$O(|V| \cdot m) + O(|V| \cdot B) + O(|V|) + O(2 \cdot |V|) \tag{3}$$

that, since B is a constant, can be approximated to

$$O(|V| \cdot m). \tag{4}$$

```
Algorithm 4: $predict(m, k, model, A_U)$
Data:
    · $m$: Integer
        number of (non-target) descriptive attributes
    · $k$: Integer
        number of target attributes
    · $model$: RDD[Node, ⟨Vector, ClusterID⟩]
        clustering result: labeled objects associated with their $(m+k)$-dimensional feature
        vector and their cluster ID
    · $A_U$: RDD[Node, Vector]
        unlabeled objects represented by a node and an $m$-dimensional feature vector
Result:
    · $pred$: RDD[Node, Vector]
        unlabeled objects associated with predictions for the $k$ target attributes
1  begin
       /* Compute similarities between unlabeled objects and labeled objects,
          keeping the cluster ID and the value of target attributes              */
2      $sims$: RDD[Node, ⟨ClusterID, Double, Vector⟩] = $A_U$
3          .cartesian($model$).map
4            {case(⟨$unNode, unVec$⟩, ⟨$node$, ⟨$vec, cluster$⟩⟩) →
               /* Non-target attributes                                          */
5              $nontarget = vec$.take($m$)
               /* Cosine similarity                                              */
6              $sim$ = cosineSim($unVec, nontarget$)
               /* Value of target attributes                                     */
7              $target = vec$.takeRight($k$)
8              ⟨$unNode$, ⟨$cluster, sim, target$⟩⟩}
       /* Associate unlabeled objects to the cluster of the most similar labeled
          object                                                                 */
9      $assocClusters$: RDD[Node, ClusterID] = $sims$
10         .reduceByKey
11           {case(⟨$clust1, sim1, val1$⟩, ⟨$clust2, sim2, val2$⟩) →
12             if ($sim1 > sim2$) then
13                 ⟨$clust1, sim1, val1$⟩
14             else
15                 ⟨$clust2, sim2, val2$⟩}
16         .map
17           {case($unNode$, ⟨$cluster, sim, target$⟩) → ⟨$unNode, cluster$⟩}
           /* Compute predictions through a weighted average of the values of
              target attributes of labeled objects belonging to the associated
              cluster                                                            */
18         $pred$: RDD[Node,Vector] = $sims$
19             .map
20               {case ($unNode$, ⟨$cluster, sim, target$⟩) →
21                   ⟨⟨$unNode, cluster$⟩, ⟨$sim, target$⟩⟩}
22             .filter
23               {case(⟨$unNode, cluster$⟩, ⟨$sim, target$⟩) →
24                   $cluster == assocClusters$.lookup($unNode$)}
25             .map
26               {case (⟨$unNode, cluster$⟩, ⟨$sim, target$⟩) →
27                   ⟨⟨$unNode, cluster$⟩, ⟨$(sim * target), sim$⟩⟩}
28             .reduceByKey
29               {case(⟨$target1, sim1$⟩, ⟨$target2, sim2$⟩) →
30                   ⟨$(target1 + target2), (sim1 + sim2)$⟩}
31             .map
32               {case(⟨$unNode, cluster$⟩, ⟨$sumTarget, sumSim$⟩) →
33                   ⟨$unNode, (sumTarget / sumSim)$⟩}
34         return $pred$
```

Focusing on the main clustering algorithm (Algorithm 2), we can observe that the initialization (Algorithm 2, lines 2–4) requires $O(|V|)$ operations, since it performs an assignment for each object. Then, the algorithm performs u iterations of the main loop (Algorithm 2, lines 6–15), each of which consisting of a *map* and a *reduceByKey* performed on all the links of the neighborhood graphs. This means that, since the neighborhood graph has $O(|V| \cdot B)$ links, the main clustering algorithm has an overall time complexity of:

$$O(u \cdot 2 \cdot |V| \cdot B) = O(|V| \cdot u). \tag{5}$$

The number of performed iterations u can be (pessimistically) estimated as the average number of steps required to propagate the cluster ID of a core object to all the other objects. Since objects in the neighborhood graph have at most B neighbors, we can observe that, starting from a given core object, in u iterations we are able to propagate its cluster ID to $B + O(B^2) + O(B^3) + \cdots + O(B^u)$ objects,[9] that can be approximated to $O(B^u)$. This means that, in order to reach all the objects (we have this guarantee when $B^u \geq |V|$), we need at least $u = \log_B |V|$ iterations. In fact:

$$B^u \geq |V| \Rightarrow \log_B B^u \geq \log_B |V| \Rightarrow u \geq \log_B |V| \tag{6}$$

Therefore, by assuming $u = O(\log_B |V|)$ and by combining Eqs. 4 and 5, we can conclude that the time complexity of the training phase is:

$$O(|V| \cdot m) + O(|V| \cdot \log_B |V|). \tag{7}$$

As regards the prediction phase (Algorithm 1, line 7 and Algorithm 4), we compute the cosine similarity between the unlabeled object and all the labeled objects (Algorithm 4, lines 2–8), leading to a time complexity of $O(|V| \cdot m)$. Then, the identification of the cluster containing the most similar labeled object (Algorithm 4, lines 9–17) requires a scan of all the labeled objects, which time complexity is $O(|V|)$. Finally, the weighted average computed to make the predictions (Algorithm 4, lines 18–33), requires the scan of the objects belonging to the selected cluster that, in the worst case scenario (i.e., all the objects belong to one single cluster), requires $O(|V|)$ operations. Therefore, the overall complexity of the prediction phase is:

$$O(|V| \cdot m) + O(|V|) + O(|V|) = O(|V| \cdot m) \tag{8}$$

Results and discussion

In the following, we first describe the experimental setting, the competitor systems, and the adopted datasets. Finally, we show and discuss the obtained results.

Experimental setting and competitor systems

Our experiments focused on two aspects: scalability and regression performances. It is noteworthy that the accuracy achieved in solving the regression task is a clear indicator of the quality of the identified clusters.

Regarding scalability, we compared DENCAST with a highly optimized clustering method available in Apache Spark, i.e., K-means, on a large-scale dataset. The adoption of the K-means implementation available in Apache Spark is motivated by its high

[9] This assumes that, during the exploration of the graph, the propagation always happens towards new (not already visited) objects. However, if an object receives the same cluster ID multiple times during the execution of different iterations, the propagation does not happen and, therefore, it is not counted (Algorithm 2, lines 10–12).

popularity, as well as by its native presence as an official, stable implementation since the early versions of the Spark MLLib machine learning library.

For a fair comparison, we let K-means extract the same number of clusters identified by DENCAST. We run the experiments on a cluster of five machines, each equipped with a 4-cores (8 threads) CPU at 3.40 GHz, 32 GB of RAM and a 750 GB SSD hard drive. Moreover, we measured DENCAST running times on a single machine, with an increasing number of instances, and compared them with those obtained on the cluster of machines, in order to directly evaluate the performance gain due to the distributed environment. By exploiting such results, we also evaluated the speedup factor, i.e., the ratio between the running time on a single machine and the running time on the cluster. Finally, we measured the scaleup factor, which shows the ability of DENCAST to exploit the computational power of multiple CPUs to process an increasing number of instances.

Regression performances were evaluated in a forecasting setting since all the considered datasets contain measurements of a target variable at different time stamps (see "Datasets" section). Moreover, we performed the experiments in both the single-target (ST) setting, to predict a single target value in the future, and the multi-target (MT) setting to predict a time series in the future. We performed the evaluation in terms of Root Mean Square Error (RMSE) and, for the time series, on the average RMSE over the time series.

We compared DENCAST with: (i) a baseline strategy, which predicts the value of the target attributes using the average value in the training set (AVG); (ii) four distributed regression algorithms, i.e., linear regression (LR), isotonic regression (ISO), ARIMA and a distributed implementation, based on DeepLearning4J, of long short-term memory neural networks (LSTM) for regression; (iii) the K-means clustering algorithm, extended to solve regression tasks.[10]

Methodologically, LR trains an elastic net regularized linear regression model [30], which overcomes the limitations of the LASSO method by combining the L1 and L2 penalties of the LASSO and ridge methods. ISO is capable of fitting a non-decreasing free-form line to a set of points, without assuming the linearity of the target function [31]. LSTM learns neural networks that effectively model temporal dependencies in sequential data, by introducing loops in the structure of the network that allow the information to persist [40]. For K-means, we adopted a prediction strategy similar to that described in "Exploiting clusters for multi-target regression" section, except for the cluster assignment that was performed according to the closest cluster centroid (coherently with the way it performs clustering).

We clarify that, although several additional methods for single- and multi-target regression exist in the literature, we focus on those for which a distributed implementation is available. Non-distributed algorithms have been widely investigated before, and a comparison with them is out of the scope of our analysis.

We run LR and ISO only in the single target setting since they do not perform multi-target regression. On the other hand, we adapted K-means and AVG to perform

[10] The adopted implementation of the ARIMA algorithm is available at https://github.com/sryza/spark-timeseries. LR, ISO, and K-means are available in the Apache Spark MLlib library.

Table 1 Experimental results on LSH showing RMSE and execution time obtained with different configurations of *r* and *numPerm*

r	numPerm	RMSE	Time (s)
1	20	0.1619	27
1	30	0.1619	36
1	40	0.1619	45
3	20	0.1198	28
3	30	0.1198	40
3	40	0.1198	60
5	20	0.0991	45
5	30	0.0991	93
5	40	0.0991	74
7	20	0.0876	46
7	30	0.0875	74
7	40	0.0865	119
9	20	0.0762	57
9	30	0.0758	111
9	40	0.0755	190
11	20	0.0622	57
11	30	0.0634	131
11	40	0.0634	160

multi-target regression, following the same principle adopted by DENCAST to solve regression tasks.

We optimized the input parameters of all the methods on an independent split of each dataset as follows:

- *DENCAST* We set $labelChangeRate = 5\%$ and performed a grid search to identify the best values of the parameters $minPts \in \{3, 5, 10\}$ and $minSim \in \{0.8, 0.9, 0.95, 0.97, 0.98, 0.99\}$. The other LSH parameters were optimized in a separate preliminary experiment (see Table 1 for some results) and were accordingly set as follows: $r = 5$ (number of hyperplanes), $B = minPts \cdot 2$, (number of nearest neighbors to consider), $numPerm = 20$ (number of random permutations). These values provided a good trade-off between accuracy and running times.
- *AVG* This approach does not need any tuning.
- *K-means* We performed a grid search to choose the best value of k from the set $\{\sqrt{n}/8, \sqrt{n}/4, \sqrt{n}/2, \sqrt{n}, \sqrt{n} \cdot 2, \sqrt{n} \cdot 4, \sqrt{n} \cdot 8\}$.
- *ARIMA* The best values of its parameters (i.e., the parameters p, d, q [33]) were automatically optimized by a tuning procedure available in the Spark-TS library (based on the AUTO-ARIMA algorithm [48]);
- *Linear regression (LR)* We performed a grid search to optimize the regularization parameter in $\{0.15, 0.3, 0.45\}$.
- *Isotonic regression (ISO)* The specific implementation in Spark does not require any optimization.

Table 2 Quantitative information on the datasets used for the scalability analysis and for the single-target (ST) and multi-target (MT) regression tasks

	#instances (ST)	#features (ST)	#instances (MT)	#features (MT)
Scalability				
PVNREL-Full	20,000,000	11	–	–
Regression				
PVItaly	254,486	16	13,455	214
PVNREL	331,968	16	17,520	214
LightSource	331,968	14	2550	212
WindNREL	48,545	14	2650	221
Bike sharing	17,379	15	731	199
Burlington	16,464	11	686	218

- *Long short-term memory neural networks (LSTM)* We performed a grid search to optimize the values of different hyperparameters: learning rate $lr \in \{10^{-1}, 10^{-2}, 10^{-3}\}$, dropout $d \in \{0.1, 0.3, 0.5\}$, batch size $bs \in \{64, 128, 256, 512, 1024\}$.

We set the remaining (secondary) parameters to their default values.

Datasets

In our experiments, we considered the following datasets (also see Table 2):

- *PVItaly* data on energy production, aggregated hourly, collected from Jan 1st, 2012, to May 4th, 2014, by sensors located on 17 photovoltaic plants in Italy (see details in [7]).
- *PVNREL* simulated photovoltaic data from 6000 plants, aggregated hourly, for the year 2006. In the scalability test, we considered a dataset of 20 M instances, which allowed us to deeply assess the efficiency of the proposed approach compared to K-means. In the evaluation of the regression performances, we also used a reduced version, consisting of the 48 plants obtained by a stratified sampling which selected 3 plants for each of the 16 States with the highest global horizontal irradiation (GHI). The reduced version allowed us to perform an extensive comparison with the approaches mentioned above since most of them were not able to process the full dataset in a reasonable time.
- *LightSource*:[11] solar energy production data from 7 plants based in the United Kingdom. We enriched production data with irradiance data from PVGIS and weather data from Forecast.io, and aggregated spot values (1 min data) hourly.
- *WindNREL* measurements of wind power plants from more than 30,000 sites. We selected five plants with the highest production, obtaining the time series of wind speed, production and climatic data (extracted from Forecast.io), aggregated hourly, from Jan 1st, 2005, to Dec 31st, 2006.
- *Bike sharing* data from the *Capital bikeshare* system on the rental of bikes, aggregated hourly and daily, from/to different positions, collected in 2011 and 2012 [49]. Data include the count of rented bikes and weather information.

[11] Not publicly available, even if anonymized, due to legal reasons.

- *Burlington* data from 51 kW DC rooftop photovoltaic installations owned by Dealer.com with 216 modules. The data span a period between 2nd Nov, 2012 and 18th Sep, 2014, aggregated hourly, and include the average measurements of temperature, irradiance, energy production and weather conditions extracted from Forecast.io.

Given the nature and temporal granularity of all the datasets considered, for each dataset, we select five random splits containing 10% of the days and adopt them as five different test sets. For each day of the test set, we consider the measurements of each hour of the previous 30, 60 and 90 days as training set, predict the value assumed by the target variable for each hour of the day and collect the average RMSE computed over the splits and the days. In the MT setting, each object represents one day, therefore: (i) the descriptive attributes correspond to the time series of the measurements and (ii) we predict the time series of the target attribute. The time series represent the hours of the day.

Discussion

In Table 3 we show the average RMSE obtained by DENCAST and by all the considered competitor systems. As anticipated, the MT results are not available for LR and ISO. Additionally, ISO was not able to provide the results within 20 days of execution for some configurations of the PVItaly and PVNREL datasets. The results obtained by ISO were also poor in terms of RMSE and almost in line with the results obtained by the baseline (AVG).

From the results obtained with different sizes of the training set we can observe that, in general, there is no significant variation. An important exception resides in the results obtained by ARIMA, which leads to more errors when the size of the training set increases. This behavior is reasonable since ARIMA only observes the time series described by the target variables. Therefore, its predictions are negatively affected by objects that are too distant, in terms of timestamp, from the unlabeled objects in the test set. On the contrary, in some configurations, other methods took advantage of larger training sets. An example can be seen in the Bike sharing dataset (single-target setting), for which DENCAST obtains the best results only with the largest training set (90 days).

Focusing on the two different settings (MT vs. ST), we observe that the MT setting provides advantages in most cases. In some datasets such a difference is significant. It is noteworthy that DENCAST is the system that benefits most from the MT setting (Table 3, last column). This result confirms that it is reasonable in many domains to combine the MT setting (which takes into account dependency between the values of the same time series), with a density-based predictive clustering solution.

Comparing DENCAST with other methods in terms of RMSE, we can see (Table 3) that it shows the best results in almost all the configurations, in both the ST and MT settings. We can observe some exceptions in the datasets LightSource and Bike Sharing, where K-means shows the best performances in the ST setting, and in the Burlington dataset, where LSTM outperforms the other competitors in the MT setting. The only dataset in which DENCAST obtains worse results than K-means in the MT setting (but not in ST) is WindNREL. This behavior is possibly motivated by the highly variable nature of wind that makes long-term prediction challenging for density-based clustering methods. Here density-based clustering tends to extract bigger clusters when compared

Table 3 Forecasting results in terms of average RMSE

	Single-target (ST)			Multi-target (MT)			Avg MT
	30 days	60 days	90 days	30 days	60 days	90 days	Improv (%)
PVItaly							
DENCAST	*0.1416*	*0.1384*	*0.1416*	*0.1002*	*0.1030*	*0.0950*	*29.2*
K-means	0.1471	0.1474	0.1469	0.1803	0.1744	0.1726	− 19.5
ARIMA	0.1508	0.1704	0.1925	0.1508	0.1704	0.1925	0.0
AVG	0.2032	0.2058	0.2065	0.2490	0.2490	0.2490	− 21.4
LR	0.1516	0.1521	0.1532	N/A	N/A	N/A	N/A
ISO	0.2026	0.2000	–	N/A	N/A	N/A	N/A
LSTM	0.2296	0.2296	0.2296	0.2280	0.2280	0.2280	0.70
PVNREL							
DENCAST	*0.1519*	*0.1535*	*0.1529*	*0.1140*	*0.1166*	*0.1193*	*23.7*
K-means	0.2366	0.2385	0.2397	0.2337	0.2198	0.2150	4.8
ARIMA	0.2736	0.2843	0.3001	0.2736	0.2843	0.3001	0.0
AVG	0.2635	0.2640	0.2647	0.3324	0.3324	0.3324	− 25.9
LR	0.2265	0.2274	0.2284	N/A	N/A	N/A	N/A
ISO	0.2621	–	–	N/A	N/A	N/A	N/A
LSTM	0.3367	0.3367	0.3367	0.2867	0.2867	0.2867	14.85
LightSource							
DENCAST	0.1680	0.1618	0.1680	*0.1222*	*0.1225*	*0.1263*	25.45
K-means	*0.1338*	*0.1380*	*0.1332*	0.1658	0.1668	0.1672	− 23.43
ARIMA	0.1596	0.1729	0.1920	0.1596	0.1729	0.1920	0.0
AVG	0.1891	0.1910	0.1940	0.1267	0.1299	0.1355	*31.71*
LR	0.1651	0.1667	0.1687	N/A	N/A	N/A	N/A
ISO	0.1989	0.1977	0.1968	N/A	N/A	N/A	N/A
LSTM	0.2027	0.2027	0.2027	0.2123	0.2123	0.2123	− 4.73
WindNREL							
DENCAST	*0.2992*	*0.2813*	*0.2853*	0.3169	0.3220	0.3343	− 12.5
K-means	0.3763	0.3844	0.3929	*0.2037*	*0.1761*	*0.1682*	*53.1*
ARIMA	0.3131	0.3460	0.3757	0.3131	0.3460	0.3757	0.0
AVG	0.3263	0.3452	0.3480	0.4939	0.4939	0.4939	− 45.5
LR	0.3072	0.3146	0.3226	N/A	N/A	N/A	N/A
ISO	0.3494	0.3517	0.3627	N/A	N/A	N/A	N/A
LSTM	0.4858	0.4858	0.4858	0.3913	0.3913	0.3913	19.45
Bike sharing							
DENCAST	0.1117	0.1114	*0.1089*	*0.0848*	*0.0894*	*0.0926*	*19.6*
K-means	*0.1096*	*0.1096*	0.1136	0.2335	0.2301	0.2244	− 87.4
ARIMA	0.1646	0.1739	0.2240	0.1646	0.1739	0.2240	0%
AVG	0.1625	0.1638	0.1656	0.0926	0.0957	0.0992	41.6
LR	0.1260	0.1278	0.1308	N/A	N/A	N/A	N/A
ISO	0.1759	0.1905	0.1996	N/A	N/A	N/A	N/A
LSTM	0.2424	0.2424	0.2424	0.2144	0.2155	0.2155	11.52
Burlington							
DENCAST	*0.1263*	*0.1280*	*0.1260*	0.1205	0.1320	0.1196	2.2
K-means	0.1408	0.1450	0.1485	0.2128	0.2047	0.2059	− 43.7
ARIMA	0.1886	0.2100	0.2263	0.1886	0.2100	0.2263	0.0%
AVG	0.1985	0.2017	0.2051	0.2246	0.2246	0.2246	− 11.3
LR	0.1440	0.1485	0.1538	N/A	N/A	N/A	N/A
ISO	0.4521	0.4388	0.4218	N/A	N/A	N/A	N/A
LSTM	0.1972	0.1980	0.1975	*0.1157*	*0.1157*	*0.1157*	41.43

The best results for each configuration are highlighted in italics. The last column represents the improvement of MT w.r.t. ST

Table 4 Number of clusters extracted by K-means and DENCAST in their best performing configuration for all the datasets and window sizes

	30 days		60 days		90 days	
	DENCAST	K-means	DENCAST	K-means	DENCAST	K-means
Single-target (ST)						
PVItaly	618	50	1356	50	1905	25
PVNREL	1441	21	3396	21	4386	21
LightSource	268	16	490	11	722	109
WindNREL	306	8	491	8	911	8
Bike sharing	72	216	119	216	179	108
Burlington	61	27	121	27	171	13
Multi-target (MT)						
PVItaly	31	192	85	192	91	192
PVNREL	103	304	205	304	315	304
LightSource	17	120	31	120	37	120
WindNREL	13	96	28	96	37	96
Bike sharing	3	5	6	2	9	2
Burlington	2	20	7	5	9	20

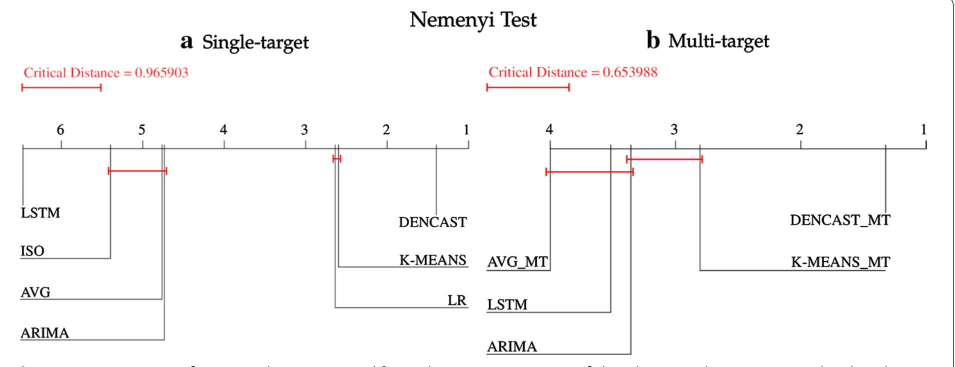

Fig. 5 Nemenyi test for **a** single-target and **b** multi-target settings. If the distance between methods is less than the critical distance (at *p*-value = 0.05), there is no statistically significant difference between them

to highly-fragmented centroid-based ones. This hypothesis is confirmed by the fact that, in this dataset, the optimal value of k for K-means is the highest ($\sqrt{n} \cdot 8$). In Table 4, where we report the number of clusters extracted by K-means and DENCAST in their best-performing configurations, we can observe that this behavior is confirmed for all the datasets: DENCAST generally extracts a lower amount of clusters than K-means in the MT setting, while it extracts a significantly higher number of clusters in the ST setting. This result confirms that, although in the MT setting the number of clusters is generally lower for both the approaches (due to the lower number of instances), the density-based approach leads to a number of clusters which is not strictly dependent on the number of instances, and that better adapts to the data distribution.

However, in most of the cases in which DENCAST obtains worse results with respect to the competitors, the difference is marginal. In order to statistically confirm this conclusion, we used the Friedman test with the Nemenyi post-hoc test at $\alpha = 0.05$. In Fig. 5,

Table 5 Standard deviations of the predictions measured for K-means and DENCAST and the result of the Wilcoxon signed-rank tests

	30 days		60 days		90 days	
	DENCAST	K-means	DENCAST	K-means	DENCAST	K-means
Single-target (ST)						
PVItaly	0.0419	0.0497	0.0388	0.0510	0.0389	0.0520
PVNREL	0.0217	0.0379	0.0196	0.0385	0.0213	0.0386
LightSource	0.0079	0.0055	0.0061	0.0068	0.0082	0.0090
WindNREL	0.1063	0.1948	0.1086	0.2113	0.1081	0.2188
Bike sharing	0.0399	0.0417	0.0398	0.0422	0.0391	0.0425
Burlington	0.0655	0.0820	0.0636	0.0867	0.0665	0.0862
Average	0.0472	0.0686	0.0461	0.0727	0.0470	0.0745
Winner	DENCAST		DENCAST		DENCAST	
p-value	*0.0374*		*0.0139*		*0.0139*	
Multi-target (MT)						
PVItaly	0.0360	0.0743	0.0363	0.0740	0.0335	0.0753
PVNREL	0.0184	0.0416	0.0185	0.0380	0.0197	0.0353
LightSource	0.0082	0.0080	0.0064	0.0070	0.0041	0.0025
WindNREL	0.1139	0.1247	0.1137	0.0926	0.1119	0.0794
Bike sharing	0.0402	0.0918	0.0433	0.0895	0.0459	0.0885
Burlington	0.0678	0.1427	0.0726	0.1339	0.0673	0.1404
Average	0.0474	0.0805	0.0485	0.0725	0.0470	0.0702
Winner	DENCAST		DENCAST		DENCAST	
p-value	*0.0232*		0.0579		0.0865	

Statistically significant results are marked in italics

we depict the result of the test, which shows that in the ST setting ISO, ARIMA and LSTM do not appear statistically better than the baseline AVG, while LR, K-means and DENCAST significantly outperform it. Moreover, DENCAST significantly outperforms all the competitors. Looking at the results in the MT setting, the difference is more evident, showing a clear dominance of DENCAST.

Moreover, we performed the Wilcoxon test on the average standard deviations which showed a p-value < 0.05 for all the single-target configurations and the 30-days configuration of the multi-target setting. Another aspect we want to emphasize is that DENCAST generally provides more stable predictions than our version of K-means (which, as stated above, we extended with the same prediction step proposed for DENCAST). To show this aspect, in Table 5 we report the results in terms of the standard deviation of the predictions, which clearly show that DENCAST always leads to lower standard deviations than K-means.

In order to evaluate the efficiency of the proposed approach, we compared the performance of DENCAST, on the full version of the PVNREL dataset (20 M objects–400 million edges), with the predictive K-means, which is highly-optimized in terms of efficiency in Apache Spark. In Fig. 6, we show the running times observed when the number of objects (and, thus, the number of edges in the neighborhood graph) increases. DENCAST appears much more efficient than the predictive K-means and scales better

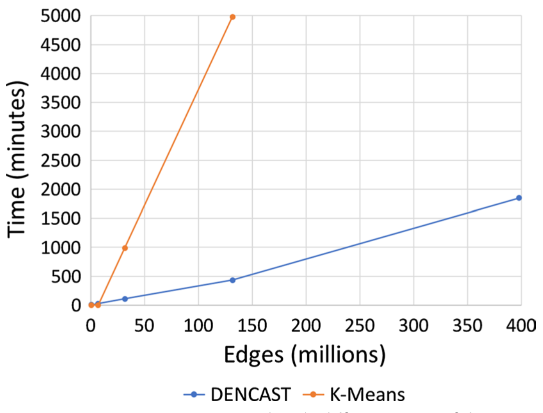

Fig. 6 DENCAST vs K-means. Running times measured with different sizes of the PVNREL dataset (100 K, 1 M, 5 M, 10 M, 20 M objects and 0.7 M, 7 M, 30 M, 130 M, 400 M edges in the neighborhood graph)

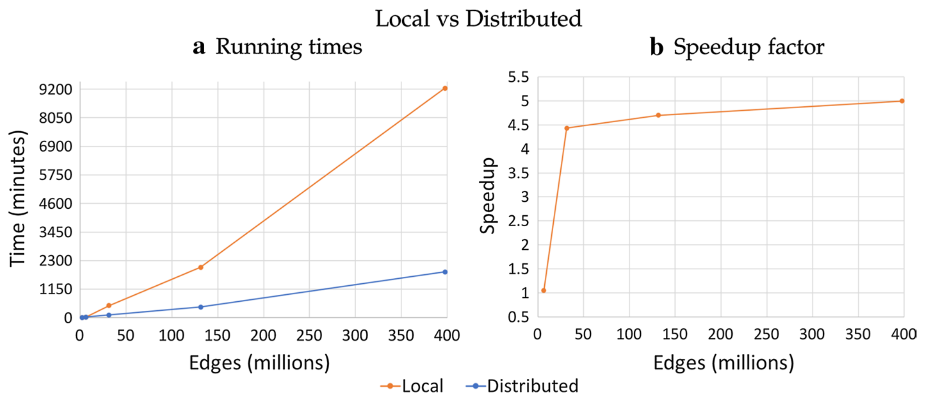

Fig. 7 DENCAST-local vs DENCAST-distributed. **a** Running times and **b** speedup factor, with different sizes of PVNREL (100 K, 1 M, 5 M, 10 M, 20 M objects and 0.7 M, 7 M, 30 M, 130 M, 400 M edges in the graph)

when the number of edges increases significantly. On the other hand, K-means was not able to analyze the full version of the dataset within 20 days.

Moreover, as introduced in "Results and discussion" section, we evaluated the DENCAST speedup factor. In particular, we first measured the running times on a single machine and on the cluster of machines for the analysis of the dataset PVNREL, with different sizes (in terms of the number of objects and, accordingly, of edges in the neighborhood graph). In Fig. 7a, it is possible to observe a comparison in terms of running times, which shows that the distributed approach scales much more efficiently on larger datasets with respect to the local variant. This improvement is confirmed by the speedup factor plotted in Fig. 7b, which shows that, with the largest version of the PVNREL dataset, DENCAST boosts the performance up to a 5× factor, which is the ideal speedup factor with our cluster consisting of five machines.

Finally, we measured the scaleup factor, in order to evaluate the ability of DENCAST to exploit the computational power of multiple CPUs when dealing with datasets with a linearly increasing size. In particular, we considered four different sizes of the PVNREL dataset with a proportional increase in the number of cores used by DENCAST. The results plotted in Fig. 8 show that the scaleup factor is very close to 1 for almost all

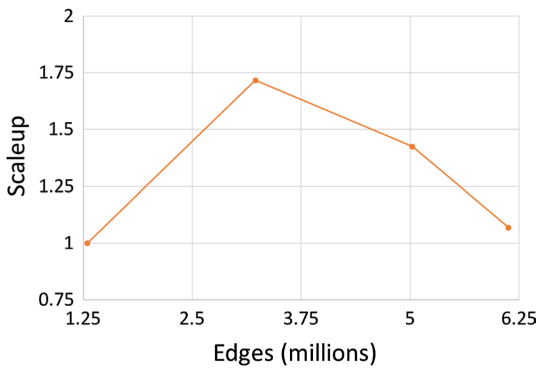

Fig. 8 DENCAST-local vs DENCAST-distributed. Scaleup factor measured with different sizes of the PVNREL dataset (1.25 M, 3,75 M, 5 M, 6.25 M edges) and increasing number of cores (8, 16, 24, 32)

the configurations, which means that the overhead introduced by DENCAST in the distribution of the workload to multiple machines is very low. Moreover, processing even larger datasets through DENCAST would only require a linear increase in the number of available CPUs in the cluster of machines.

Conclusion

In this paper, we proposed DENCAST, a density-based clustering algorithm implemented in Apache Spark, which is able to handle large-scale and high-dimensional data. We also exploited the clusters identified by DENCAST to predict the value assumed by one or more target variables of unlabeled objects, i.e., for regression purposes in both single and multi-target settings.

Our experimental evaluation, performed on several datasets, demonstrated the ability of DENCAST to obtain predictions with a higher accuracy than existing distributed regression approaches. Such competitive regression results also confirm the quality of the extracted clusters. A further analysis showed that DENCAST clearly benefits from the multi-target setting. In particular, the combination of the density-based predictive clustering solution with the multi-target setting led DENCAST to dominate over all the considered competitors. This is an important result, since it confirms that catching possible dependencies among the target attributes provides a great margin of improvement in the regression task.

Moreover, an analysis focused on the efficiency emphasized the ability of DENCAST to significantly outperform the distributed version of K-means in Apache Spark, in terms of running times. Finally, a scalability analysis has shown that DENCAST exhibits optimal speedup and scaleup performances. In particular, it reached a 5× speedup factor with five machines, corresponding to the ideal speedup factor, and a scaleup factor close to 1, which emphasizes a very low overhead due to the distribution of the workload. This relevant result is due to the advantage of performing all the stages in a fully distributed manner, without incurring in any computational bottleneck.

For future work, we aim to introduce the possibility to handle mixed-types attributes (i.e., not only numerical attributes). Moreover, we plan to extend the proposed approach in order to make it able to solve classification tasks as well to measure and explicitly model spatio-temporal autocorrelation phenomena during the clustering and the prediction phases.

Abbreviations

API: Application Programming Interface; ARIMA: AutoRegressive Integrated Moving Average; DPC: density peaks clustering; GHI: global horizontal irradiation; IoT: Internet of Things; ISO: isotonic regression; LR: linear regression; LSH: locality-sensitive hashing; LSTM: long short-term neural networks; MT: multi-target; RDD: resilient distributed dataset; RMSE: root mean square error; ST: single-target; SVR: support vector regression.

Acknowledgements
We thank Lynn Rudd for her help in reading the manuscript.

Authors' contributions

RC, GP and MC collaborated in the design of the method. RC and MC took care of the study of the related work. RC and GP implemented the system and ran the experiments. MC, RC and GP performed the analysis of the results. RC, GP and MC contributed to the manuscript drafting. GP, MC, RC and DM contributed to the manuscript finalization. MC and DM supervised the study. All authors read and approved the final manuscript.

Funding
We acknowledge the support of the European Commission through the projects MAESTRA—Learning from Massive, Incompletely annotated, and Structured Data (Grant No. ICT-2013-612944) and TOREADOR—Trustworthy Modelaware Analytics Data Platform (Grant No. H2020-688797). We also acknowledge the support of Ministry of Education, Universities and Research (MIUR) through the PON project ComESto—Community Energy Storage: Gestione Aggregata di Sistemi d'Accumulo dell'Energia in Power Cloud (Grant No. ARS01 01259).

Availability of data and materials

The system DENCAST and all the datasets are available at http://www.di.uniba.it/~gianvitopio/systems/dencast/.

Competing interests

The authors declare that they have no competing interests.

Author details
[1] Department of Computer Science, University of Bari Aldo Moro, Via Orabona, 4, Bari, Italy. [2] Big Data Laboratory, National Interuniversity Consortium for Informatics (CINI), Rome, Italy.

Received: 18 April 2019 Accepted: 24 May 2019
Published online: 03 June 2019

References

1. Cannataro M, Congiusta A, Pugliese A, Talia D, Trunfio P. Distributed data mining on grids: services, tools, and applications. IEEE Trans Syst Man Cybern B. 2004;34(6):2451–65.
2. Ester M, Kriegel H-P, Sander J, Xu X, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. Kdd. 1996;96:226–31.
3. He Y, Tan H, Luo W, Mao H, Ma D, Feng S, Fan J. MR-DBSCAN: an efficient parallel density-based clustering algorithm using MapReduce. In: Proceeding of ICPADS. 2011. p. 473–80.
4. Cordova I, Moh T-S. DBSCAN on resilient distributed datasets. In: High performance computing & simulation. 2015. p. 531–40.
5. Han D, Agrawal A, Liao WK, Choudhary A. A novel scalable DBSCAN algorithm with Spark. In: International parallel and distributed processing symposium workshops. 2016. p. 1393–402.
6. Blockeel H, Raedt LD, Ramon J. Top–down induction of clustering trees. In: Shavlik JW, editor. Proceeding of ICML. Madison: Morgan Kaufmann; 1998. p. 55–63.
7. Ceci M, Corizzo R, Fumarola F, Malerba D, Rashkovska A. Predictive modeling of PV energy production: how to set up the learning task for a better prediction? IEEE Trans Ind Inform. 2017;13(3):956–66.
8. Ceci M, Corizzo R, Malerba D, Rashkovska A. Spatial autocorrelation and entropy for renewable energy forecasting. Data Mining Knowl Discov. 2019;33:698–729.
9. Chen X, Cai X, Liang J, Liu Q. Ensemble learning multiple lssvr with improved harmony search algorithm for short-term traffic flow forecasting. IEEE Access. 2018;6:9347–57.
10. Liu B-C, Binaykia A, Chang P-C, Tiwari MK, Tsao C-C. Urban air quality forecasting based on multi-dimensional collaborative support vector regression (svr): a case study of beijing-tianjin-shijiazhuang. PLoS ONE. 2017;12(7):0179763.
11. Liu J, Sun L, Li Q, Ming J, Liu Y, Xiong H. Functional zone based hierarchical demand prediction for bike system expansion. In: Proceeding of ACM SIGKDD 2017. New York: ACM; 2017. p. 957–66.
12. Li Y, Zheng Y, Zhang H, Chen L. Traffic prediction in a bike-sharing system. In: SIGSPATIAL. New York: ACM; 2015. p. 33.
13. Xioufis ES, Tsoumakas G, Groves W, Vlahavas IP. Multi-target regression via input space expansion: treating targets as inputs. Mach Learn. 2016;104(1):55–98.
14. Dincer NG, Akkuş Ö. A new fuzzy time series model based on robust clustering for forecasting of air pollution. Ecol Inform. 2018;43:157–64.
15. Stojanova D, Ceci M, Appice A, Dzeroski S. Network regression with predictive clustering trees. Data Mining Knowl Discov. 2012;25(2):378–413.

16. Pio G, Serafino F, Malerba D, Ceci M. Multi-type clustering and classification from heterogeneous networks. Inform Sci. 2018;425:107–26.
17. Stojanova D, Ceci M, Appice A, Malerba D, Džeroski S. Dealing with spatial autocorrelation when learning predictive clustering trees. Ecol Inform. 2013;13:22–39.
18. Charikar MS. Similarity estimation techniques from rounding algorithms. In: Proceeding of the 34th annual ACM symposium on theory of computing. New York: ACM; 2002. p. 380–8.
19. Rodriguez A, Laio A. Clustering by fast search and find of density peaks. Science. 2014;344(6191):1492–6.
20. Comaniciu D, Meer P. Mean shift: a robust approach toward feature space analysis. IEEE Trans Pattern Anal Mach Intell. 2002;24(5):603–19.
21. Sreedhar C, Kasiviswanath N, Reddy PC. Clustering large datasets using k-means modified inter and intra clustering (km-i2c) in hadoop. J Big Data. 2017;4(1):27.
22. Zhang H, Raitoharju J, Kiranyaz S, Gabbouj M. Limited random walk algorithm for big graph data clustering. J Big Data. 2016;3(1):26.
23. Kaur A, Datta A. A novel algorithm for fast and scalable subspace clustering of high-dimensional data. J Big Data. 2015;2(1):17.
24. Ankerst M, Breunig MM, Kriegel H-P, Sander J. Optics: ordering points to identify the clustering structure. SIGMOD Rec. 1999;28(2):49–60.
25. Aggarwal CC, Han J, Wang J, Yu PS. A framework for clustering evolving data streams. In: VLDB. 2003. p. 81–92.
26. Birant D, Kut A. ST-DBSCAN: an algorithm for clustering spatial-temporal data. Data Knowl Eng. 2007;60(1):208–21.
27. Wu Y-P, Guo J-J, Zhang X-J. A linear DBSCAN algorithm based on LSH. In: International conference on machine learning and cybernetics, vol. 5. IEEE. 2007. p. 2608–14.
28. Berchtold S, Keim DA, Kriegel H-P. The X-tree: an index structure for high-dimensional data. In: Proceedings of VLDB '96, San Francisco, CA, USA. 1996. p. 28–39.
29. Huang F, Zhu Q, Zhou J, Tao J, Zhou X, Jin D, Tan X, Wang L. Research on the parallelization of the DBSCAN clustering algorithm for spatial data mining based on the Spark platform. Rem Sens. 2017;9:12.
30. Zou H, Hastie T. Regularization and variable selection via the elastic net. J R Stat Soc. 2005;67(2):301–20.
31. Barlow R, Brunk H. The isotonic regression problem and its dual. J Am Stat Assoc. 1972;67(337):140–7.
32. Ababei C, Moghaddam MG. A survey of prediction and classification techniques in multicore processor systems. IEEE Trans Parallel Distrib Syst. 2018;30:5.
33. Box GE, Jenkins GM, Reinsel GC, Ljung GM. Time series analysis: forecasting and control. 5th ed. Oxford: Wiley; 2015.
34. Corizzo R, Ceci M, Japkowicz N. Anomaly detection and repair for accurate predictions in geo-distributed Big Data. Big Data Res. 2019;16:18–35.
35. Kocev D, Vens C, Struyf J, Džeroski S. Tree ensembles for predicting structured outputs. Pattern Recogn. 2013;46(3):817–33.
36. Borchani H, Varando G, Bielza C, Larrañaga P. A survey on multi-output regression. Wiley Interdiscip Rev. 2015;5(5):216–33.
37. Brudnak M. Vector-valued support vector regression. In: IJCNN'06. IEEE international joint conference on neural networks. 2006. p. 1562–9.
38. Xu S, An X, Qiao X, Zhu L, Li L. Multi-output least-squares support vector regression machines. Pattern Recogn Lett. 2013;34(9):1078–84.
39. Appice A, Džeroski S. Stepwise induction of multi-target model trees. In: European conference on machine learning. Berlin: Springer; 2007. p. 502–9.
40. Hochreiter S, Schmidhuber J. Long short-term memory. Neural Comput. 1997;9:1735–80.
41. Zaharia M, Chowdhury M, Franklin MJ, Shenker S, Stoica I. Spark: cluster computing with working sets. In: Proceeding of HotCloud'10. 2010. p. 10.
42. Ravichandran D, Pantel P, Hovy E. Randomized algorithms and NLP: using locality sensitive hash function for high speed noun clustering. In: Meeting on association for computational linguistics. ACL '05. 2005. p. 622–9.
43. Ferreira LN, Zhao L. Time series clustering via community detection in networks. Inform Sci. 2016;326:227–42.
44. Leskovec J, Rajaraman A, Ullman JD. Mining of massive datasets. 2nd ed. New York: Cambridge University Press; 2014.
45. Andoni A, Indyk P, Laarhoven T, Razenshteyn I, Schmidt L. Practical and optimal lsh for angular distance. In: Proceedings of the 28th international conference on neural information processing systems, volume 1. NIPS'15. Cambridge: MIT Press. 2015. p. 1225–33.
46. Tian X, Guo Y, Zhan J, Wang L. Towards memory and computation efficient graph processing on spark. In: International conference on Big Data (Big Data). 2017. p. 375–82.
47. Gonzalez JE, Xin RS, Dave A, Crankshaw D, Franklin MJ, Stoica I. Graphx: graph processing in a distributed dataflow framework. OSDI. 2014;14:599–613.
48. Hyndman RJ, Khandakar Y, et al. Automatic time series for forecasting: the forecast package for r. Technical report. Monash University, Department of Econometrics and Business Statistics. 2007.
49. Fanaee TH, Gama J. Event labeling combining ensemble detectors and background knowledge. Progr Artif Intell. 2013;2:1–15.

Publisher's Note