Forecasting via Distributed Density-Based Clustering (Discussion Paper)

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

[1]University of Bari Aldo Moro
Department of Computer Science - Via Orabona, 4 - 70125 Bari, Italy

[2]CINI - Consorzio Interuniversitario Nazionale per l'Informatica - Bari
{roberto.corizzo,gianvito.pio,michelangelo.ceci,donato.malerba}@uniba.it

1 Introduction

The generation of massive amounts of data in different forms (such as activity logs and measurements produced by sensor networks) increased the need of novel data mining algorithms which are capable to build accurate models efficiently and in a distributed fashion. In the recent years, several approaches able to distribute the workload to several machines have been proposed for the classical clustering, classification and regression tasks. However, they often suffer from strong limitations in their applicability, i.e. they are limited to data organized in a specific structure, are able to analyze only low-dimensional data, or suffer from overhead and scalability issues when the number of instances and attributes increase considerably. For example, to the best of our knowledge, in the literature there are very few density-based clustering algorithms (mostly inspired by the well known algorithm DBSCAN) able to work with an arbitrary number of features and with good scalability performances. Moreover, most of them are exploitable only for pure clustering purposes. At this respect, in this paper we propose a novel density-based clustering algorithm, implemented in the Apache Spark framework, which is able to handle large-scale and high-dimensional data by exploiting the Locality Sensitive Hashing (LSH). Moreover, the proposed approach is able to exploit the identified clusters, built on historical data, to forecast the values assumed by a target variable in the future. Therefore, the proposed method can be adopted also for forecasting purposes.

The rest of the paper is structured as follows: in Section 2, we briefly review existing methods to solve the classical density-based clustering task as well as recently proposed clustering methods which are able to handle large-scale data in parallel on multiple processors. In Section 3, we propose our distributed density-based clustering solution, which is able to work on high-dimensional and large-scale data, and describe our approach to exploit the identified clusters for forecasting purposes. In Section 4, we report the results of our experimental evaluation, showing that the proposed method is able to obtain accurate predictions and appears very efficient in dealing massive amounts of high-dimensional data. Finally, in Section 5 we draw some conclusions and outline some future work.

2 Background on Density-based Clustering

Density-based clustering was introduced in [9] with the algorithm DBSCAN. This approach is able to identify arbitrarily shaped clusters (not only spherical, as partitioning-based approaches) 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 about density-based algorithms are common to those adopted in this paper, here we recall some useful basic notions:

- The neighborhood of an object p is defined as $N(p) = \{q \mid dist(p, q) < eps\}$;
- An object p is a *core object* w.r.t. *eps* and *minPts* 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 is 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, such that for each pair of objects $\langle p, q\rangle$, p is density-connected to q;
- Non-core objects belonging to at least one cluster are called *border objects*, whereas objects not belonging to any cluster are called *noise objects*.

The algorithm DBSCAN starts with an arbitrary object o and, if it is a core object, retrieves all the objects which are density-reachable from o w.r.t. *eps* and *minPts*. This procedure returns a cluster and the algorithm proceeds with the next unclustered object. This algorithm has been proved to identify accurate and arbitrary shaped clusters and is almost independent of the order of the analysis of the objects. Several variants have been proposed in the literature, aiming at facing different limitations of the original DBSCAN (e.g., [2, 5] for the estimation of the value of the input parameters) or at extending its applicability, for example, to streams of data [1] or to spatio-temporal data [3].

In order to be able to process large datasets, other works focused on the computational complexity, which is originally $O(n^2 \cdot m)$ (where n is the number of objects and m is the number of features), dominated by the computation of the neighborhood of each node. An example can be found in [14], where the authors proposed an approximated variant of DBSCAN, based on Locality Sensitive Hashing (LSH) [7] which requires $O(n \cdot m)$ steps to compute the neighborhood of the nodes. Still focusing on the possibility to process large datasets, in [12] a parallel implementation of DBSCAN for MapReduce has been proposed. This method is based on the partitioning of objects according to each dimension, thus it suffers from a computational viewpoint with high-dimensional data. For this reason, the authors limit their experiments to 2D datasets. The same limitation affects other existing methods implemented in the Apache Spark framework [8].

In this context, the algorithm proposed here attacks the issues raised by high-dimensional and large-scale datasets through an approach which is computationally efficient and distributed in all its steps and that, inspired by works on predictive clustering [13], can be exploited for forecasting purposes.

3 The Proposed Method

In this Section, we describe our distributed density-based clustering method, which is able to work with high-dimensional and large-scale datasets and that can be exploited for forecasting purposes. In Figure 1 we show the general workflow of the proposed approach, while in the following subsections we describe in detail each step of the method, briefly analyzing their computational complexity and showing that they can be easily parallelized in the Apache Spark framework.

3.1 Computation of the Neighborhood Graph and Core Objects

Given a dataset D of n objects described by m features ($m-1$ descriptive attributes and one target attribute), the first goal is to identify groups of similar objects according to their features. Most clustering algorithms, including density-based algorithms, strongly rely on the computation of similarity/distance measures among pairs of objects. This task has an inherent cost of $O(n^2 \cdot m)$, which can be reduced significantly only by resorting to approximated strategies. In this paper, inspired by [14], we adopt the Locality Sensitive Hashing (LSH), which is able to identify an approximation of the neighborhood graph (where nodes are linked when their similarity is greater than 1-*eps*) among the instances in $O(n \cdot m)$, preserving the neighborhood relationships. In particular, LSH adopts a hash function which maps similar objects (according to cosine similarity) to the same *bucket* with high probability. Objects falling in the same bucket of a given object are considered as belonging to its neighborhood. Here, we exploit a parallel implementation of LSH available for the Apache Spark framework[1].

Once the neighborhood graph has been computed, we identify the set of core objects, according to the definitions provided in the previous section, that is, we identify the set $cores = \{o \text{ s.t. } |N(o)| \geq minPts\}$. Let E be the set of edges of the neighborhood graph identified by LSH, represented as a distributed data structure of pairs of nodes. This step can be implemented as an *aggregateByKey* operation (to find the set of neighbors for each node) followed by a *filter* operation (to select only those nodes o such that $|N(o)| \geq minPts$). Therefore, this step is fully parallelizable and, since each edge of the neighborhood graph is analyzed once, its computational complexity is $O(|E|)$.

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

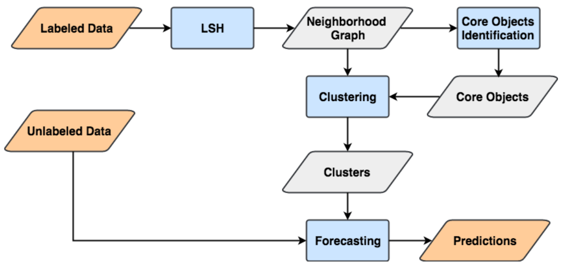

Fig. 1. Workflow of the proposed method.

3.2 Distributed Density-based Clustering

Given the set of core nodes and the neighborhood graph, our density-based clustering algorithm exploits the property of *density-connection*, described in Section 2, to simultaneously identify multiple density-based clusters. It is noteworthy that in this phase we do not need node attributes, therefore we can distribute over the available machines only the neighborhood graph, making the proposed algorithm efficient also in terms of memory requirements. An intuitive pseudo-code description of our clustering method is shown in Algorithm 1.

In detail, the algorithm initially assigns a different cluster ID to each core node, and the $ID = 0$ to boundary and noise nodes (lines 1-10). This step can be implemented in a distributed fashion by a single *map* operation over the set of nodes, therefore its complexity is $O(n)$. Subsequently, each core node *sends a message* containing its cluster ID to its neighbouring nodes having a lower cluster ID (lines 15-27). Each node receives multiple *messages* from its adjacent

Algorithm 1 Density-based clustering exploiting the neighborhood graph.

Require:
· $G = \langle V, E \rangle$: the neighborhood graph, where V are the nodes and E are the edges representing the neighborhood relationship between pairs of nodes;
· *cores*: set of core nodes from which to start the propagation of cluster IDs;
· *labelChangeRate*: minimum percentage of label changes to perform a new iteration.

Ensure:
· the updated graph G, where nodes V are associated to their cluster ID.

```
1: {Initialization of the cluster ID for core objects}
2: $clusterid \leftarrow 1$
3: for all $v \in V$ do
4:     if $v \in cores$ then
5:         $v.clusterID \leftarrow clusterID$
6:         $clusterID \leftarrow clusterID + 1$
7:     else
8:         $v.clusterID \leftarrow 0$
9:     end if
10: end for
11:
12: {Propagation of cluster IDs according to reachability of nodes}
13: $threshold \leftarrow labelChangeRate * |E|$
14: repeat
15:     $changes \leftarrow 0$
16:     {Reset messages for all the nodes}
17:     for all $v \in V$ do
18:         $v.messages \leftarrow \emptyset$
19:     end for
20:
21:     {Each node receives multiple messages from (core, clustered) neighboring nodes}
22:     for all $e = \langle src, dst \rangle \in E$ do
23:         if $src \in cores$ and $src.clusterID > dst.clusterID$ then
24:             $dst.messages \leftarrow dst.messages \cup \{src.clusterID\}$
25:             $changes \leftarrow changes + 1$
26:         end if
27:     end for
28:
29:     {Aggregate multiple messages received by each node}
30:     for all $v \in V$ do
31:         $v.clusterID \leftarrow max(v.messages)$
32:     end for
33: until ($changes < threshold$)
34: return $G$
```

nodes and aggregates them by taking the highest cluster ID (lines 29-32)[2]. We repeat this process until the cluster assignments appear stable, i.e. when the messages exchanged among nodes through the edges of the neighborhood graph are less then a given percentage $labelChangeRate$ of the whole set of edges. This process is coherent with the general concepts at the basis of classical density-based clustering approaches, since clusters can only be built from core nodes and since the propagation of cluster IDs is performed only from core nodes.

This iterative process can be easily parallelized, since each edge of the graph can be analyzed independently of the other to evaluate whether a message has to be sent. Moreover, the final aggregation step of the received messages can be performed by a single *reduce* operation. Therefore, the computational cost of this phase is dominated by $O(u \cdot |E|)$, where u is the number of performed iterations. We also emphasize the fact that, contrary to existing methods (e.g., [8]) our approach does not require a final merging procedure necessary to aggregate the results computed on different machines, which is usually executed on a single *driver* machine.

3.3 Exploiting Clusters for Forecasting Purposes

Inspired by predictive clustering approaches [13], we exploit the k identified clusters for forecasting purposes (see Figure 2). In particular, we re-associate each node to its features through a *join* operation and compute a representative m-dimensional vector (including the target attribute[3]) for each cluster, by performing a column-wise average of the vectors associated to objects falling in the cluster. This operation can be easily parallelized by aggregating the clustering result over the cluster IDs and by performing a single *map* operation to obtain the representative vectors for each cluster. Since each object can fall in at most one cluster, the complexity of this operation is $O(n \cdot m)$. The prediction of the value of the target attribute for unlabeled instances can then be performed by

[2] This is only an implementation choice. Indeed, we could keep the lowest cluster ID without any change in the results, since the *density-connection* is symmetric.

[3] Here we assume that the target attribute is numerical. However, we can handle also categorical attributes by adopting a strategy based on majority voting.

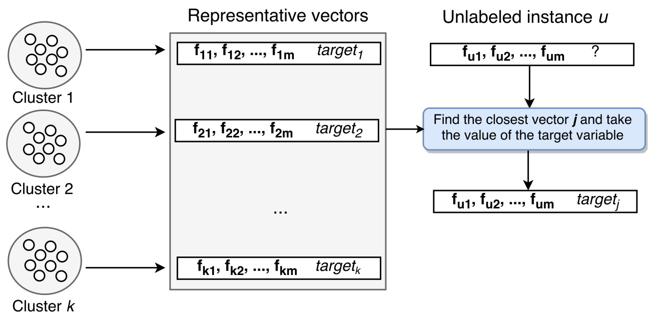
Representative vectors
Unlabeled instance u
Cluster 1
Cluster 2
...
Cluster k
$f_{11}, f_{12}, ..., f_{1m}$ $target_1$
$f_{21}, f_{22}, ..., f_{2m}$ $target_2$
...
$f_{k1}, f_{k2}, ..., f_{km}$ $target_k$
$f_{u1}, f_{u2}, ..., f_{um}$?
Find the closest vector j and take the value of the target variable
$f_{u1}, f_{u2}, ..., f_{um}$ $target_j$

Fig. 2. Outline of the forecasting step.

comparing their $(m-1)$-dimensional feature vector (excluding the target attribute, since it is unknown for unlabeled instances) to all the representative vectors associated to the clusters, in order to find the closest cluster. The value assumed by the target attribute of the closest representative vector is finally associated to the unlabeled instance. Also this step can be easily parallelized through a *map* operation over the set of unlabeled instances which compares their vector to all the k representative vectors. Therefore, the computational complexity of the prediction phase is $O(k * m)$ for each unlabeled instance.

4 Experiments

We performed the experiments by adopting the self-adaptive online training strategy, where data are represented as a time-based sliding window [11] of size d (past d days of historical data). We considered different values of d $(30, 60$ and $90)$, and set the forecasting horizon as one-day-ahead. Each configuration was run five times, with different random picks of test days. Results are evaluated in terms of the average Root Mean Square Error (RMSE).
The considered approaches, each run in its best configuration, are:
• **Our method**, optimized by means of a grid search on a separate set of days, in order to identify the best values of the parameters *eps* and *minPts*;
• **K-Means** algorithm, implemented in the Apache Spark framework, extended with our forecasting strategy. A grid search has been performed using a separate set of days, in order to identify the best value of the parameter k;
• **ARIMA** [4], a pure forecasting method which automatically selects the best forecasting model, on the basis of the Akaike Information Criterion (AIC);
• **AVG**, a baseline approach which predicts the value of the target attribute as the average value assumed over the training instances.

The considered datasets are described in the following:
• **PVItaly.** This dataset contains data about the energy production of photovoltaic power plants, aggregated hourly, collected at regular intervals by sensors located on 17 plants in Italy, from January 1st 2012 to May 4th 2014.
• **PVNREL.** This dataset originally consists of simulated photovoltaic data of 6,000 plants, aggregated hourly, for the year 2006. The experiments were performed on a reduced version of the dataset, consisting of 48 plants belonging to the 16 States with the highest Global Horizontal Irradiation.
• **Bike Sharing (BS).** This dataset[4] contains data of the *Capital bikeshare* system, aggregated hourly, about the process of rental and returning of bikes, from/to possibly different positions, collected in 2011 and 2012 [10].

Descriptive attributes of *PVItaly* and *PVNREL* regard environmental and meteorological factors, whereas the target attribute is the hourly energy production (pre-processing steps applied to these datasets can be found in [6]). Descriptive attributes of *Bike Sharing* regard weather and seasonal information, whereas the target attribute is the hourly count of rented bikes. See Table 1 for additional information about the dataset characteristics and the considered parameters.

[4] `https://archive.ics.uci.edu/ml/datasets/Bike+Sharing+Dataset`

Table 1. Characteristics and parameter values for each dataset. LSH parameters are set as follows: *dimensions*=5, *numPermutations*=20, *maxNeighbors*=2·*minPts*.

				Window size (days)								
				30			60			90		
	instances	**test set**	**features**	**min Pts**	**eps**	**training set**	**min Pts**	**eps**	**training set**	**min Pts**	**eps**	**training set**
PVItaly	254,486	323	19	3	0.03	9690	3	0.03	19,380	5	0.02	29,070
PVNREL	331,968	912	17	10	0.97	27,360	10	0.95	54,720	10	0.95	82,080
BS	17,379	24	15	3	0.02	720	3	0.02	1,440	3	0.02	2,160

Table 2. Forecasting results (RMSE) with different training window sizes. Best results for each dataset and configuration are highlighted in bold.

	PVItaly			**PVNREL**			**BS**		
	Window size (days)			Window size (days)			Window size (days)		
	30	60	90	30	60	90	30	60	90
Our Method	**0.1371**	**0.1341**	**0.1379**	**0.1525**	**0.1488**	**0.1499**	0.1204	0.1142	0.1188
K-Means (w/forecasting)	0.1471	0.1474	0.1469	0.2338	0.2335	0.2349	**0.1096**	**0.1096**	**0.1136**
ARIMA	0.1508	0.1704	0.1925	0.2736	0.2843	0.3001	0.1646	0.1739	0.2240
AVG	0.2032	0.2058	0.2065	0.2635	0.2640	0.2647	0.1625	0.1638	0.1656

In the results shown in Table 2, we can observe that the simple baseline approach *AVG* actually appears comparable to *ARIMA* in the datasets PVNREL and BS. However, they are strongly outperformed by our method and the extended version of *K-Means*. Our density-based method leads to the best results in PVItaly and PVNREL, whereas it appears comparable to *K-Means* in BS. Moreover, it can be observed that, on PVNREL and BS, the proposed method is able to exploit the possible availability of more instances in the training phase. At this respect, in order to evaluate whether we can handle a massive amount of data, without incurring in time complexity issues, we performed a scalability test with the large version of the dataset PVNREL, on a cluster of 4 worker machines and one driver machine, each with 4 cores and 32GB of RAM, by progressively increasing the number of edges in the neighborhood graph up to 130 millions. In Figure 3, we can observe that *i)* the proposed algorithm scales linearly with respect to the number of edges, therefore it is able to deal massive amounts of data, and *ii)* the possible overhead introduced by the distribution of data to (and by the collection of the results from) different machines appears negligible.

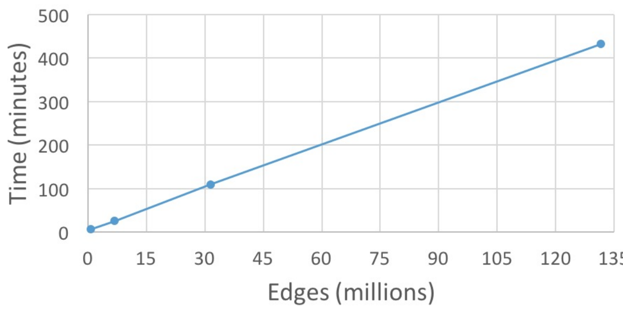

Fig. 3. Scalability test with increasing number of edges in the neighborhood graph.

5 Conclusions

In this paper, we proposed a distributed density-based clustering approach, which can be fruitfully exploited for forecasting purposes. The algorithm, implemented in the Apache Spark framework, is fully parallel, does not require any final merging procedure and is computationally efficient. Experimental results show that it is able to obtain good predictive performance and that it scales linearly with respect to the number of edges in the neighborhood graph. For future work, we intend to extend the forecasting capabilities to the multi-target setting and to perform more extensive comparisons with existing parallel solutions.

Acknowledgments

We would like to acknowledge the support of the European Commission through the projects MAESTRA - Learning from Massive, Incompletely annotated, and Structured Data (Grant Number ICT-2013-612944) and TOREADOR - Trustworthy Model-aware Analytics Data Platform (Grant Number H2020-688797).

References

1. Aggarwal, C.C., Han, J., Wang, J., Yu, P.S.: A framework for clustering evolving data streams. In: Proc. of VLDB - Volume 29. pp. 81–92. VLDB Endowment (2003)
2. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: OPTICS: Ordering Points to Identify the Clustering Structure. In: ACM SIGMOD '99. pp. 49–60 (1999)
3. Birant, D., Kut, A.: ST-DBSCAN: An algorithm for clustering spatial–temporal data. Data & Knowledge Engineering 60(1), 208–221 (2007)
4. Box, G.E.P., Jenkins, G.: Time Series Analysis, Forecasting and Control. Holden-Day, Incorporated (1990)
5. Campello, R.J.G.B., Moulavi, D., Zimek, A., Sander, J.: Hierarchical density estimates for data clustering, visualization, and outlier detection. ACM Trans. Knowl. Discov. Data 10(1), 5:1–5:51 (Jul 2015)
6. 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 Transactions on Industrial Informatics PP(99), 1–1 (2016)
7. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: Proc. of ACM symposium on Theory of computing. pp. 380–388. ACM (2002)
8. Cordova, I., Moh, T.S.: Dbscan on resilient distributed datasets. In: High Performance Computing & Simulation (HPCS). pp. 531–540. IEEE (2015)
9. Ester, M., Kriegel, H.P., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: Kdd. vol. 96(34), pp. 226–231 (1996)
10. Fanaee-T, H., Gama, J.: Event labeling combining ensemble detectors and background knowledge. Progress in Artificial Intelligence pp. 1–15 (2013)
11. Gama, J.: Knowledge Discovery from Data Streams. Chapman and Hall / CRC Data Mining and Knowledge Discovery Series, CRC Press (2010)
12. 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: Parallel and Distributed Systems (ICPADS). pp. 473–480. IEEE (2011)
13. Stojanova, D., Ceci, M., Appice, A., Dzeroski, S.: Network regression with predictive clustering trees. Data Min. Knowl. Discov. 25(2), 378–413 (2012)
14. Wu, Y.P., Guo, J.J., Zhang, X.J.: A linear DBSCAN algorithm based on LSH. In: Int. Conf. on Machine Learning and Cybernetics. vol. 5, pp. 2608–2614 (2007)