Clustering Geo-data Cubes

4 October 2021

Introduction

Faced with the increasing ubiquity of large datasets across most scientific domains, data mining techniques have become essential to extracting patterns, i.e valorizing the data for knowledge development and/or decision making. Data clustering techniques - algorithmic prescriptions for identifying (sub-)groups of similar elements - play a key role in this regard.

In the case of multi-variate data matrices, it is often the sub-structures/sub-spaces characterized by simultaneous similarity along multiple/all dimensions which are of the greatest interest. It has, however, long been recognized that traditional 'single-sided' clustering techniques are, in general, inadequate for this purpose. Instead, techniques capable of clustering data along multiple axes simultaneously, often referred to as co- or bi-clustering in the case of two axes or tri-clustering in the case of three axes, are needed, and have seen significant development [] and adoption in fields ranging from bioinformatics [] to finance [] and natural language processing [].

At face value, geo-information science, with ever more and larger data sets of geo-referenced time-series (GTS), would appear to be a natural adopter of co- and tri-clustering. However, immediate adoption was stymied by the initial focus of co- and tri-clustering algorithms on 'significant' clusters, i.e. with values above a threshold [], rather than on a full partitioning of the data as required for clustering analysis of GTS. More recently, following the development of a general information-theoretical approach (Dhillon, Mallela, and Modha 2003) to partitional co-clustering (Banerjee et al. 2004), Wu et al. presented an application of co-clustering to GIS (Wu, Zurita-Milla, and Kraak 2015), as well as an extension of the method to tri-clustering (Wu et al. 2018). As they argue, with the explosion of Earth observation and remote sensing (3D)GTS, such methods will become increasingly essential in tackling the large data volumes becoming available.

In light of the eminent employability of partitional co- and tri-clustering to GIS, but also the transferability to other domains, this paper presents and publishes the implementation in the Clustering Geo-data Cubes (CGC) package of the co-clustering and tri-clustering methods originally developed in Wu, Zurita-Milla, and Kraak (2015) and Wu et al. (2018), respectively.

As outlined below, the package aims to meet the needs of the GIS community, including, in particular, the ability to handle 'big data' and to readily interface with the existing GIS software ecosystem. Nevertheless, the methods remain widely applicable and can easily be applied in other domains as well.

Statement of need

The CGC package targets a specific audience, that is researchers in the field of geo- and climate science. In particular, it aims to meet the community need for a tool able to accurately cluster geospatial data such as GTS by providing the following features and functionalities:

- **Partitional co- and tri-clustering algorithms, as suitable to work with (multi-band) spatiotemporal raster data.** CGC entails algorithms that are based on information theory and that are designed to simultaneously group elements into disjoint clusters along all the input array axes. These methods are advantageous over single-sided clustering in that they provide a strategy to identify patterns that unfold across multiple dimensions (e.g. space and time). In addition, CGC provides the functionality to perform a cluster refinement over the identified co- and tri-clusters. This post-processing step has the goal to reduce the number of clusters by grouping them based on similarity, facilitating the identification and visualization of patterns.
- **A scalable approach, able to efficiently perform cluster analyses on both small and big data sets.** CGC offers solutions for a wide range of data set sizes by providing co- and tri-clustering implementations designed to run on either a single machine or on a compute cluster. For the former, which tackles input data arrays that largely fit into memory, one can exploit parallelization of independent analysis runs, taking advantage of multi-core CPUs. For the latter, which tackles instead large datasets, the analysis is carried out using distributed data and computation.
- **A framework easy to integrate into geospatial analysis workflows.** CGC is written in Python, which is one of the top programming languages for GIS scripting and applications. In the implementation targeting distributed computing, CGC makes use of the Dask library (Dask Development Team 2016), which is widely employed in the field of big geo-data. Numpy and Dask arrays, which are the data structures employed in CGC, are also employed in the higher-level objects in the Xarray package (Hoyer and Hamman 2017) so that this versatile and popular tool can be used for data loading and manipulation before ingestion to CGC. Documentation and tutorials illustrate domain-science examples, applications, and use cases to facilitate community adoption.

Algorithms

Co-clustering

CGC implements the Bregman block average co-clustering (BBAC) algorithm from Banerjee et al. (2004) (see Algorithm 1 in the article), and it was inspired by the Matlab code by Srujana Merugu and Arindam Banerjee (Merugu and Banerjee 2004). The algorithm iteratively optimizes the clustering of rows and columns starting from a random initial assignment. The information loss from the original matrix to the clustered one, which is constructed as the matrix of the co-cluster means, is minimized using a loss function that is based on the I-divergence. To limit the influence of the initial conditions on the final clustering, which might represent a local minimum in the cluster space, multiple differently-initialized runs are carried out.

The iterative cluster optimization involves steps where the cluster-based means are calculated and the row- and column-cluster assignments updated to minimize the loss function. Note that in the CGC implementation of the algorithm, the update in the row- and column-cluster assignments is computed only from the previous iteration's row and column clusters (and the corresponding cluster-based means). The order in which the axes of the input data matrix are considered does not affect the outcome of the co-clustering. This differs from the implementation in the original Matlab code(Merugu and Banerjee 2004), where the new row-cluster assignments are employed to update the column-cluster assignments at the same step, leading to an asymmetry in how the clustering of rows and columns is handled.

Tri-clustering

For tri-clustering, CGC implements the so-called Bregman cube average tri-clustering (BCAT) algorithm, which is the natural generalization of the BBAC algorithm to three dimensions (Wu et al. 2018). Also for tri-clustering, a loss function based on the I-divergence is employed and multiple runs are carried out to avoid local minima as much as possible. At every iteration, the cluster-based means are calculated and the clusters in the three dimensions updated to minimize the information loss from the original data matrix to its clustered counterpart. As for the CGC co-clustering implementation, only the clusters from the previous iteration (and the corresponding cluster means) are employed to update the tri-cluster assignments. Thus, the tri-clustering outcome is not influenced by the order in which the three axes of the input data array are provided.

Cluster refinement

CGC implements an optional cluster refinement step based on the k-means method (Wu, Zurita-Milla, and Kraak 2016). For this, we exploit the implementation available in the scikit-learn package (Pedregosa et al. 2011). The co- and tri-clusters are classified into a pre-defined number of refined clusters and this pre-defined number is referred to as k. In this step, similar clusters are identified in the co-clustering or tri-clustering results. This identification is performed by computing certain pre-defined features over all elements belonging to the same co- or tri-cluster. CGC employs the following features in the k-means implementation:

- Mean value;
- Standard deviation;
- Minimum value;
- Maximum value;
- 5th percentile;
- 95th percentile;

CGC searches for the optimal k value within a given range, using the Silhouette coefficient (Rousseeuw 1987) as the measure. The optimal k is identified as the value with the highest Silhouette coefficient.

Software package overview

The CGC software is structured in the following main modules:

- `coclustering`, which offers access to all the available implementations of the same co-clustering algorithm:
    - An implementation based on Numpy, as suitable to run the algorithm on a single machine. In this implementation, vectorized operations between work arrays are exploited to efficiently calculate cluster-based means and to optimize the cluster assignments. Multiple differently-initialized runs can be run in parallel using the threading library, allowing to make optimal usage of machines with multi-core CPUs.
    - A second Numpy-based implementation that replaces some of the vectorized operations with looping to reduce the need for large work arrays, leading to a much-reduced memory footprint. While the reduced memory requirement comes at the cost of performance, the loss can be mitigated through a feature that activates Numba's just-in-time compilation for some of the critical steps.
    - An implementation based on the Dask library, targeting data sets whose size would prevent analyses on a single machine. In particular, Dask arrays are employed to process the data in chunks, which are distributed across the nodes of a compute cluster. The runs to sample initial conditions are executed sequentially to keep memory requirements to the minimum.
- `triclustering`, which offers access to all tri-clustering implementations:
    - A Numpy-based implementation analogous to the co-clustering one

    described above (note that the low-memory version is currently not available).
  - A Dask-based implementation, also analogous to the corresponding co-clustering version described above.

- `kmeans`, which implements the k-means cluster refinement step for both co- and tri-clustering.

- `utils`, which includes a collection of utility functions.

Tutorial

The software package is complemented by an online tutorial that illustrates how to perform cluster analysis of geospatial datasets using CGC. The tutorial exploits a format that is widely employed in the geoscience community, i.e. Jupyter notebooks(Kluyver et al. 2016). Notebooks are made directly available via a dedicated GitHub repository, but they are also published as static web pages for reference and linked to the CGC documentation via a 'Tutorials' tab. The tutorials are designed to run on systems with limited CPU/memory capacity, which, together with environment requirement specifications in a standardized format (`conda` YAML file) and binder badges, give users the possibility to easily run the notebooks live on `mybinder.org` (Jupyter Project et al. 2018).

Tutorials cover the following topics:

- Co- and tri-cluster analysis.
- K-means-based cluster refinement.
- Choice of the suitable implementation for a given problem size/infrastructure available.
- Loading of geospatial data, common data-manipulation tasks in a workflow involving CGC, visualization of the output.

Note they while the tutorial is aimed at geospatial uses cases, it illustrates some real-case applications that are likely to make it easier for users to carry out cluster analysis using CGC in other fields as well.

Acknowledgements

The authors acknowledge Dr. Yifat Dzigan for the helpful discussions and support and Dr. Romulo Goncalves for the preliminary work that lead to the development of the software package presented here.

Author contributions

The Netherlands eScience Center team (F.N., M.W.G., and O.K.) took care of most of the software implementation, contributed to the algorithm design and testing, and wrote most of the manuscript. E.I.V. S.G. R.Z.M.

References

Banerjee, Arindam, Inderjit Dhillon, Joydeep Ghosh, Srujana Merugu, and Dharmendra S. Modha. 2004. "A Generalized Maximum Entropy Approach to Bregman Co-Clustering and Matrix Approximation." In *Proceedings of the 2004 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '04*, 509. Seattle, WA, USA: ACM Press. https://doi.org/10.1145/1014052.1014111.

Dask Development Team. 2016. *Dask: Library for Dynamic Task Scheduling.* https://dask.org.

Dhillon, Inderjit S., Subramanyam Mallela, and Dharmendra S. Modha. 2003. "Information-Theoretic Co-Clustering." In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '03*, 89. Washington, D.C.: ACM Press. https://doi.org/10.1145/956750.956764.

Hoyer, Stephan, and Joseph J. Hamman. 2017. "Xarray: N-D Labeled Arrays and Datasets in Python." *Journal of Open Research Software* 5 (April): 10. https://doi.org/10.5334/jors.148.

Jupyter Project, Matthias Bussonnier, Jessica Forde, Jeremy Freeman, Brian Granger, Tim Head, Chris Holdgraf, et al. 2018. "Binder 2.0 - Reproducible, Interactive, Sharable Environments for Science at Scale." In, 113–20. Austin, Texas. https://doi.org/10.25080/Majora-4af1f417-011.

Kluyver, Thomas, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, et al. 2016. "Jupyter Notebooks - a Publishing Format for Reproducible Computational Workflows." In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, edited by Fernando Loizides and Birgit Scmidt, 87–90. IOS Press. https://eprints.soton.ac.uk/403913/.

Merugu, Srujana, and Arindam Banerjee. 2004. "Bregman Co-Clustering Code in Matlab." http://www.ideal.ece.utexas.edu/software.html.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, et al. 2011. "Scikit-Learn: Machine Learning in Python." *Journal of Machine Learning Research* 12: 2825–30.

Rousseeuw, Peter J. 1987. "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis." *Journal of Computational and Applied Mathematics* 20 (November): 53–65. https://doi.org/10.1016/0377-0427(87)90125-7.

Wu, Xiaojing, Raul Zurita-Milla, Emma Izquierdo-Verdiguier, and Menno-Jan Kraak. 2018. "Triclustering Georeferenced Time Series for Analyzing Patterns of Intra-Annual Variability in Temperature." *Annals of the American Association of Geographers* 108 (1): 71–87. https://doi.org/10.1080/24694452.2017.1325725.

Wu, Xiaojing, Raul Zurita-Milla, and Menno-Jan Kraak. 2015. "Co-Clustering Geo-Referenced Time Series: Exploring Spatio-Temporal Patterns in Dutch Temperature Data." *International Journal of Geographical Information Science* 29 (4): 624–42. https://doi.org/10.1080/13658816.2014.994520.

———. 2016. "A Novel Analysis of Spring Phenological Patterns over Europe Based on Co-Clustering: Co-Clustering European Spring Phenology." *Journal of Geophysical Research: Biogeosciences* 121 (6): 1434–48. https://doi.org/10.1002/2015JG003308.