The Journal of Open Source Software

CGC: a scalable Python package for co- and tri-clustering of geodata cubes

Francesco Nattino[1], Ou Ku[1], Meiert W. Grootes[1], Emma Izquierdo-Verdiguier[2], Serkan Girgin[3], and Raul Zurita-Milla[3]

1 Netherlands eScience Center, Science Park 140, 1098 XG Amsterdam, The Netherlands **2** Institute of Geomatics, University of Natural Resources and Life Science (BOKU), 1190, Vienna, Austria **3** Faculty of Geo-Information Science and Earth Observation (ITC), University of Twente, PO Box 217, 7500 AE, Enschede, the Netherlands

DOI: 10.21105/joss.04032

Software

- Review
- Repository
- Archive

Editor: Hugo Ledoux

Reviewers:

- @subho07
- @narayana-rao

Submitted: 17 December 2021
Published: 10 March 2022

Summary

Multidimensional data cubes are increasingly ubiquitous, in particular in the geosciences. Clustering techniques encompassing their full dimensionality are necessary to identify patterns "hidden" within these cubes. Clustering Geodata Cubes (CGC) is a Python package designed for partitional clustering, which identifies groups of similar data across two (e.g., spatial and temporal) or three (e.g., spatial, temporal, and thematic) dimensions. CGC provides efficient and scalable co- and tri-clustering functionality appropriate to analyze both small and large datasets as well as a cluster refinement functionality that supports users in their quest to make sense of complex datasets.

Introduction

Faced with the increasing ubiquity of large datasets, data mining techniques have become essential to extracting patterns and generating insights. In this regard, clustering techniques, which aim to identify groups or subgroups with similar properties within a larger dataset, are becoming ever more popular.

Traditional clustering techniques focus on a single dimension and may therefore obfuscate relevant groups (Cheng & Church, 2000; Hartigan, 1972). Hence, clustering techniques capable of simultaneously grouping data along multiple dimensions are needed. These techniques – referred to as co- or bi-clustering and tri-clustering in the case of two and three dimensions, respectively – have seen significant development and adoption in fields ranging from bioinformatics (Cheng & Church, 2000) to finance (Shi et al., 2018) and natural language processing (Dhillon, 2001).

The exponential growth of multidimensional data referring to geographical features (e.g., time series of satellite images) has resulted in a wide variety of geodata cubes, which can benefit from co- and tri-clustering. Indeed, following the development of a general information-theoretical approach (Dhillon et al., 2003) to partitional co-clustering (Banerjee et al., 2007), Wu et al. presented an application of co-clustering to geodata (Wu et al., 2015), as well as its extension to three dimensions (Wu et al., 2018).

In light of the eminent employability of co- and tri-clustering approaches to geospatial disciplines like geo-information science and Earth observation, and the transferability to other (geo)scientific domains, this paper presents the Clustering Geodata Cubes (CGC) package. CGC provides efficient and scalable co- and tri-clustering methods to analyze both small and large datasets on single or multiple computing node systems. It also features a cluster

refinement functionality that allows users to make sense of complex datasets more easily. An example of its application is the ongoing work on the analysis of spring onset datasets at high spatial resolution and continental scale, a preview of which is presented in the CGC tutorial. Although the package aims to meet the needs of geospatial data scientists, the algorithms implemented remain widely applicable and can easily be applied in other domains that require performing partitional clustering of positive data matrices.

Statement of need

The CGC package focuses on the needs of geospatial data scientists who require tools to make sense of multi-dimensional data cubes by providing the following features and functionalities:

- **Partitional co- and tri-clustering methods suitable for spatiotemporal (multi-dimensional) data.** CGC includes clustering methods designed to simultaneously group elements into disjoint clusters along two or three dimensions. These methods are advantageous over one-dimensional clustering in that they provide a strategy to identify patterns that unfold across multiple dimensions, e.g. space and time. In addition, CGC provides functionality to refine the identified co- and tri-clusters. This post-processing step reduces the number of clusters to facilitate the identification and visualization of patterns.
- **Scalable clustering of small and big datasets.** CGC offers functionality to efficiently utilize available computational resources (ranging from single machines to computing clusters) and to tackle a wide range of dataset sizes. For single machine execution the package offers optimized support of multi-core CPUs and/or limited system memory. For large datasets CGC supports the use of distributed data and computation on computing clusters
- **Easy integration into geospatial analysis workflows.** CGC is written in Python, which is widely used for geospatial scripting and applications, and employs NumPy (Harris et al., 2020) and Dask (Dask Development Team, 2016) arrays as input and output data types, guaranteeing seamless integration to the Python ecosystem and interoperability with the libraries prevalent in the field of big (geo)data. This furthermore ensures the interoperability of CGC with the Xarray package (Hoyer & Hamman, 2017), so that this versatile and popular tool can be used for data loading and manipulation before and after analyses with CGC.
- **Ease of use and reproducibility.** To facilitate community use and adoption, documentation and tutorials illustrating domain-science examples, applications, and use cases are available via the publicly accessible repository, where development takes place, and which provides a platform for issue tracking. CGC is distributed via the Python Package Index (PyPI), and code-release snapshots archived on Zenodo facilitate reproducible analysis.

Algorithms

Co-clustering

CGC implements the Bregman block average co-clustering (BBAC) algorithm from Banerjee et al. (2007) as inspired by the MATLAB code of (Merugu & Banerjee, 2004). Briefly, the BBAC algorithm iteratively optimizes the clustering of rows and columns of a data matrix starting from a random initial assignment until convergence. 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 (Dhillon et al., 2003; Kullback & Leibler, 1951). CGC also supports a user-defined convergence threshold. To limit the influence of the initial conditions on the final clustering and to avoid local minima several

runs are carried out, with the cluster assignments from the lowest loss function value ultimately being selected. Number of runs can be altered by the user.

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. Contrarily to the original MATLAB implementation (Merugu & Banerjee, 2004), this makes the algorithm independent of the order in which the dimensions are considered, while still leading to an optimal clustering solution.

Tri-clustering

For tri-clustering CGC implements the so-called Bregman cube average tri-clustering (BCAT) algorithm, which is a generalization of the BBAC algorithm to three dimensions (Wu et al., 2018). The algorithmic implementation with respect to the loss function and the update schema are analogous to that described above for the co-clustering. Similarly, the tri-clustering outcome is independent of the order in which the three dimensions of the input array are provided.

Cluster refinement

The CGC package implements an optional, secondary cluster refinement step based on the k-means method (Wu et al., 2016) and optimized using the Silhouette metric (Rousseeuw, 1987) as implemented in the scikit-learn package (Pedregosa et al., 2011). This secondary grouping is based on statistical properties of the co- or tri-clusters (see the package documentation) and helps to better capture patterns in the data by combining clusters and going beyond strict checkerboard structures.

Related Software

A number of co-/bi-clustering implementations based on different algorithms exist. While some of the available packages focus on specific applications, like gene expression data (Barkow et al., 2006; Eren et al., 2012), generic co-clustering tools include biclust (R) and CoClust (Python) (Role et al., 2019), as well as two implementations from the scikit-learn Python library (Pedregosa et al., 2011). Most of the available algorithms target tabular data with a hidden blocked-diagonal structure, where each row and column of the input matrix is assigned to only one co-cluster. In contrast, in spatio-temporal data a set of spatial elements often exhibits the same behaviour within multiple time windows and vice versa. This type of data requires algorithms where a subset taken along one dimension can be associated to multiple subsets taken along the other dimension. Partitioning algorithms of this type, such as the BBAC and BCAT algorithms implemented in CGC, can discover checkerboard-like patterns in the input data matrix. The package additionally enables the user to recover more of the intrinsic structure of the data in a second step, going beyond the limits on structure imposed

Within the Python ecosystem, prominent implementations targeting checkerboard-like structures akin to CGC's intial step are the scikit-learn SpectralBiclustering algorithm, which implements the method from (Kluger et al., 2003), and the CoclustInfo algorithm from CoClust, also based on information-theoretic co-clustering (Dhillon et al., 2003; Govaert & Nadif, 2018). However, both approaches differ from CGC in the field of applicability. Spectral methods like the former are fast, but their accuracy has been shown to be limited (Ailem et al., 2015; Role et al., 2019). The latter focuses on datasets representing joint-probability distributions, preserving statistics other than the co-cluster average (the row and column averages) in the search for the optimal clustering solution (Banerjee et al., 2007). This requirement does not generally apply to the geospatial datasets which CGC is targeting. Neither, however, offers the ability to "break out" of the algorithmically-imposed checkerboard pattern.

Finally, to the best of our knowledge, CGC is unique in being designed from the outset for use with big data (e.g. by including an implementation that supports multi-node distributed computing), being able to analyze more complex data cubes via tri-clustering, and being able to perform secondary cluster refinement.

Software package overview

The CGC software is structured in the following main modules, details of which are described in the online package documentation:

- `coclustering`, containing the following implementations of the co-clustering algorithm:
  - The NumPy-based, vectorized single machine implementation with threading support for optimal usage of multi-core CPUs.
  - The NumPy-based single machine implementation with a reduced memory footprint. This implementation trades performance for low memory usage, but uses Numba's just-in-time compilation (Lam et al., 2015) to mitigate performance loss.
  - The Dask-based implementation. This implementation provides support for clustering large, out-of-core datasets by distributing the computation across multiple nodes using Dask arrays.
- `triclustering`, containing the following 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 e.g., memory consumption estimation and cluster averaging.

Performance comparisons between the various co-clustering and tri-clustering implementations are also briefly discussed in the package documentation.

Tutorial

The software package is complemented by an online tutorial (in the form of Jupyter notebooks (Kluyver et al., 2016)) that illustrates how to perform cluster analysis of geospatial datasets using CGC. Notebooks are made directly available via a dedicated GitHub repository, and 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 that 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 would like to thank Dr. Yifat Dzigan for the helpful discussions and support and Dr. Romulo Goncalves for the preliminary work that led to the development of the software package presented here. We also would like to thank SURF for providing computational resources to test the first versions of the CGC package via the e-infra190130 grant.

Author contributions

All co-authors contributed to the conceptualization of the work, which was led by R.Z.M. and E.I.V.. F.N., M.W.G., and O.K. prepared the first draft of the tutorials, and wrote the initial draft of this manuscript. F.N. suggested changes to the co- and tri-clustering algorithms. R.Z.M., E.I.V., and S.G. led the design of experiments to test and improve the CGC package. R.Z.M. and E.I.V. helped to improve the tutorials. S.G. provided the required computational resources to run the experiments and tutorials and made suggestions for code optimizations. All co-authors reviewed and edited the final document.

References

Ailem, M., Role, F., & Nadif, M. (2015). Co-clustering document-term matrices by direct maximization of graph modularity. *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 1807–1810. https://doi.org/10.1145/2806416.2806639

Banerjee, A., Dhillon, I., Ghosh, J., Merugu, S., & Modha, D. S. (2007). A generalized maximum entropy approach to bregman co-clustering and matrix approximation. *Journal of Machine Learning Research*, *8*(67), 1919–1986. http://jmlr.org/papers/v8/banerjee07a.html

Barkow, S., Bleuler, S., Prelić, A., Zimmermann, P., & Zitzler, E. (2006). BicAT: a biclustering analysis toolbox. *Bioinformatics*, *22*(10), 1282–1283. https://doi.org/10.1093/bioinformatics/btl099

Cheng, Y., & Church, G. M. (2000). Biclustering of expression data. *Proceedings of the Eighth International Conference on Intelligent Systems for Molecular Biology*, 93–103. ISBN: 1577351150

Dask Development Team. (2016). *Dask: Library for dynamic task scheduling*. https://dask.org

Dhillon, I. S. (2001). Co-clustering documents and words using bipartite spectral graph partitioning. *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 269–274. https://doi.org/10.1145/502512.502550

Dhillon, I. S., Mallela, S., & Modha, D. S. (2003). Information-theoretic co-clustering. *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '03*, 89. https://doi.org/10.1145/956750.956764

Eren, K., Deveci, M., Küçüktunç, O., & Çatalyürek, Ü. V. (2012). A comparative analysis of biclustering algorithms for gene expression data. *Briefings in Bioinformatics*, *14*(3), 279–292. https://doi.org/10.1093/bib/bbs032

Govaert, G., & Nadif, M. (2018). Mutual information, phi-squared and model-based co-clustering for contingency tables. *Advances in Data Analysis and Classification*, *12*(3), 455–488. https://doi.org/10.1007/s11634-016-0274-6

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hartigan, J. A. (1972). Direct Clustering of a Data Matrix. *Journal of the American Statistical Association*, *67*(337), 123–129. https://doi.org/10.1080/01621459.1972.10481214

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

Jupyter Project, Bussonnier, M., Forde, J., Freeman, J., Granger, B., Head, T., Holdgraf, C., Kelley, K., Nalvarte, G., Osheroff, A., Pacer, M., Panda, Y., Perez, F., Ragan-Kelley, B., & Willing, C. (2018). *Binder 2.0 - Reproducible, interactive, sharable environments for science at scale*. 113–120. https://doi.org/10.25080/Majora-4af1f417-011

Kluger, Y., Basri, R., Chang, J. T., & Gerstein, M. (2003). Spectral Biclustering of Microarray Data: Coclustering Genes and Conditions. *Genome Research*, *13*(4), 703–716. https://doi.org/10.1101/gr.648603

Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J., Grout, J., Corlay, S., Ivanov, P., Avila, D., Abdalla, S., Willing, C., & Jupyter development team. (2016). Jupyter notebooks - a publishing format for reproducible computational workflows. In F. Loizides & B. Scmidt (Eds.), *Positioning and power in academic publishing: Players, agents and agendas* (pp. 87–90). IOS Press. https://eprints.soton.ac.uk/403913/

Kullback, S., & Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79–86. https://doi.org/10.1214/aoms/1177729694

Lam, S. K., Pitrou, A., & Seibert, S. (2015). Numba: A LLVM-based python JIT compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. https://doi.org/10.1145/2833157.2833162

Merugu, S., & Banerjee, A. (2004). *Bregman Co-clustering code in Matlab*. http://www.ideal.ece.utexas.edu/software.html

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, *12*(85), 2825–2830. https://www.jmlr.org/papers/v12/pedregosa11a.html

Role, F., Morbieu, S., & Nadif, M. (2019). **CoClust** : A *python* Package for Co-Clustering. *Journal of Statistical Software*, *88*(7). https://doi.org/10.18637/jss.v088.i07

Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, *20*, 53–65. https://doi.org/10.1016/0377-0427(87)90125-7

Shi, G., Ren, L., Miao, Z., Gao, J., Che, Y., & Lu, J. (2018). Discovering the trading pattern of financial market participants: Comparison of two co-clustering methods. *IEEE Access*, *6*, 14431–14438. https://doi.org/10.1109/ACCESS.2018.2801263

Wu, X., Zurita-Milla, R., Izquierdo-Verdiguier, E., & Kraak, M.-J. (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, X., Zurita-Milla, R., & Kraak, M.-J. (2015). Co-clustering geo-referenced time series: Exploring spatio-temporal patterns in Dutch temperature data. *International Journal of Geographical Information Science*, *29*(4), 624–642. https://doi.org/10.1080/13658816.2014.994520

Wu, X., Zurita-Milla, R., & Kraak, M.-J. (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), 1434–1448. https://doi.org/10.1002/2015JG003308