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

18 November 2021

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, identifying 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 the cluster refinement and visualization functionality that support 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 data set, are becoming ever more popular.

Traditional clustering techniques focus on a single dimension and may therefore obfuscate relevant groups (Hartigan 1972; Cheng and Church 2000). Therefore, 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 and Church 2000) to finance (Shi et al. 2018) and natural language processing (Dhillon 2001).

The recent exponential growth of multidimensional data referring to geographical features (e.g., time series of satellite images) has resulted in a wide variety of geo-data cubes, which can benefit from co- and tri-clustering. Indeed, following

the development of a general information-theoretical approach (Dhillon, Mallela, and Modha 2003) to partitional co-clustering (Banerjee et al. 2007), Wu et al. presented an application of co-clustering to geo-information science (Wu, Zurita-Milla, and Kraak 2015), as well as an extension of that method to three dimensions (Wu et al. 2018), with these advanced clustering methods becoming increasingly essential to tackling complex multidimensional data cubes.

In light of the eminent employability of partitional co- and tri-clustering to geospatial disciplines like geo-information science and Earth observation, and the transferability to other (geo)scientific domains, this paper presents the Clustering Geo-data Cubes (CGC) package, which facilitates the efficient use of the co-clustering and tri-clustering methods presented in Wu, Zurita-Milla, and Kraak (2015) and Wu et al. (2018).

As outlined below, the package aims to meet the needs of the geospatial data scientists, including the ability to handle 'big (geo)-data' and to readily interface with the wider open-source software ecosystem specialized in geospatial data analysis. A case in point is the use of CGC in the (on-going) anlysis of spring on-set and bloom at high spatial resolution (1km) across continental scales in Europe and North America (RZM & EIV in prep.).
Nevertheless, the methods implemented in the CGC package remain widely applicable and can easily be applied in other domains.

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 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 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 data sets.** CGG offers functionality to efficiently utilize available computational resources (ranging from single machines to computing clusters) and to tackle a wide range of data set sizes. For single machine execution the package offers optimized support of multi-core CPUs and/or system memory. For large data sets GCC 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 and Dask (Dask Development Team 2016) arrays as input and output data types, guaranteeing seamless integration to the Python ecosystem and interoperability with the Dask library(Dask Development Team 2016) prevalent in the field of big geo-data. This furthermore ensures the interoperability of CGC with the Xarray package (Hoyer and 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 and 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. 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.

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 and Banerjee 2004), this makes the algorithm independent of the order in which the dimensions are considered, while still leading to a (locally) optimal clustering solution.

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). The algorithmic implementation with respect to the loss function and the update schema are analogous to that described above for the co-clustering. Importantly, 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, Zurita-Milla, and Kraak 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 the patterns hidden in the data.

Related Software

Within the Python ecosystem a number of co-/bi-clustering implementations based on a range of algorithms exist. However, prominent examples are often focused on very specific applications, such as the CoClust package (Role, Morbieu, and Nadif 2019), designed for term document clustering. More general packages such as, e.g. the spectral co-clustering implementation in scikit-learn (Pedregosa et al. 2011) are often focussed on specific classes of problems, i.e. those of block-diagonal form, or make use of Euclidean distances applied to the constructed cluster values in determining and evaluating these. In comparison, CGC makes use of an information theory based distance metric which considers the global information content of the matrix. Furthermore, CGC is also uniquely designed from the outset for performant use with 'big data.'

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 to mitigate performance loss.
    - The Dask-library-based implementation. This implementation provides support for clustering large, out-of-core data sets 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.

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 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 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.K. 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.K. 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

Banerjee, Arindam, Inderjit Dhillon, Joydeep Ghosh, Srujana Merugu, and Dharmendra S. Modha. 2007. "A Generalized Maximum Entropy Approach to Bregman Co-Clustering and Matrix Approximation." *The Journal of Machine Learning Research* 8 (December): 1919–86.

Cheng, Yizong, and George M Church. 2000. "Biclustering of Expression Data." In *Ismb*, 8:93–103. 2000.

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

Dhillon, Inderjit S. 2001. "Co-Clustering Documents and Words Using Bipartite Spectral Graph Partitioning." In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 269–74. KDD '01. New York, NY, USA: Association for Computing Machinery. https://doi.org/10.1145/502512.502550.

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.

Hartigan, J. A. 1972. "Direct Clustering of a Data Matrix." *Journal of the American Statistical Association* 67 (337): 123–29. https://doi.org/10.1080/01621459.1972.10481214.

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.

Role, François, Stanislas Morbieu, and Mohamed Nadif. 2019. "**CoClust** : A *Python* Package for Co-Clustering." *Journal of Statistical Software* 88 (7). https://doi.org/10.18637/jss.v088.i07.

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.

Shi, Guangwei, Liying Ren, Zhongchen Miao, Jian Gao, Yanzhe Che, and Jidong Lu. 2018. "Discovering the Trading Pattern of Financial Market Participants: Comparison of Two Co-Clustering Methods." *IEEE Access* 6: 14431–38.

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.