BIOINFORMATICS APPLICATIONS NOTE Vol. 30 no. 6 2014, pages 896–898
doi:10.1093/bioinformatics/btt626

Data and text mining

Advance Access publication October 31, 2013

pyGCluster, a novel hierarchical clustering approach

Daniel Jaeger, Johannes Barth, Anna Niehues and Christian Fufezan*

Institute of Plant Biology and Biotechnology, Department of Biology, University of Muenster, Münster 48143, Germany

Associate Editor: Jonathan Wren

ABSTRACT

Summary: pyGCluster is a clustering algorithm focusing on noise injection for subsequent cluster validation. The reproducibility of a large amount of clusters obtained with agglomerative hierarchical clustering is assessed. Furthermore, a multitude of different distance-linkage combinations are evaluated. Finally, highly reproducible clusters are meta-clustered into communities. Graphical illustration of the results as node and expression maps is implemented.

Availability and implementation: pyGCluster requires Python 2.7, it is freely available at http://pyGCluster.github.io and published under MIT license. Dependencies are NumPy, SciPy and optionally fastcluster and rpy2.

Contact: christan@fufezan.net

Supplementary information: Supplementary data is available at *Bioinformatics* online and at http://pyGCluster.github.io

Received on August 9, 2013; revised on October 21, 2013; accepted on October 26, 2013

1 INTRODUCTION

'Omics' technologies yield large data sets, which are commonly subjected to cluster analysis to organize them into comprehensible, i.e. coregulated groups, which might be functionally related (Si *et al*., 2011). A critical step in cluster analysis is cluster validation (Handl *et al*., 2005), the most stringent form of validation being the assessment of exact reproducibility of a cluster in the light of the uncertainty of the data.

This issue is addressed by pyGCluster, an algorithm working in two steps. First, it creates many agglomerative hierarchical clusterings (AHCs) of the input data by injecting noise based on the uncertainty of the data and clusters them using different distance linkage combinations (DLCs). Second, pyGCluster creates a meta-clustering, i.e. clustering of the resulting highly reproducible clusters into communities to gain a most complete representation of common patterns in the data. Communities are defined as sets of clusters with a specific pairwise overlap.

The first key concept is based on the importance of the uncertainty in the data set, which can be assessed by biological and/or technical repetitions. Classical bootstrapping offers a way to include the repetitions to estimate the uncertainties of the data points. For example, one could cluster each repetition separately, and clusters consistently formed would be highly reliable. However, a more refined approach would be to use the repetitions to derive the shape of the value distributions. These shapes and their describing functions can be used to define the noise injecting function in pyGCluster that allows a new data set to be generated during each iteration.

The second key concept is the evaluation of different DLCs. Although all clustering algorithms require a method to calculate the distance, i.e. a similarity metric between objects, the linkage method is specific for AHC. In a classical AHC approach, one has to define a specific distance metric (e.g. correlation, aiming at relative distances) and linkage method, albeit other distance metrics (e.g. Euclidean, aiming at absolute distances) and linkage methods may also yield interesting clusters. Using a variety of different DLCs results in a broader result spectrum and in a reduction of user bias in data evaluation.

These key concepts coupled with high numbers of iterations and meta-clustering of highly reproducible clusters into communities make pyGCluster a novel hierarchical clustering approach.

2 ALGORITHM

The workflow of pyGCluster can be divided into two steps. First, cluster reproducibility is assessed in the resampling routine. Second, highly reproducible clusters are meta-clustered into communities.

For each iteration, a new data set is generated evoking the resampling routine. The pyGCluster uses by default a noise injection function that generates a new data set in which each object o in condition l is drawn from a normal distribution defined by its mean μ_{ol} and standard deviation σ_{ol}. Alternatively, user-defined noise injection routines can be defined with which pyGCluster can also be initialized (see Supplementary Material or online documentation). The new data set is then subjected to several AHCs using different DLCs. In each hierarchical tree, all clusters are examined and their reproducibility is stored separately for each DLC in a cluster-count matrix. In this matrix, each row r is assigned to one cluster based on its exact identity, i.e. the objects within the clusters, and each column c corresponds to a specific DLC. Thus, the value_{rc} is equal to the number of times this exact cluster was found using a specific DLC.

The resampling routine may be performed either for a fixed number of iterations or until convergence is reached (see Supplementary Material for its implementation). After this point, clusters that show at least one relative frequency (i.e. count/number of iterations) for one given DLC that is higher than a user-defined threshold are retained for further analysis. Because each of these clusters has one frequency per DLC, we introduced what we have called cFreq to merge those values into one number. A normalization procedure was implemented because the number of DLCs based on Euclidean distance was larger than the ones using correlation when, for example, default parameters are used. Thus, the cFreq of a cluster is defined as the sum of the median frequencies for each group of DLCs with the DLCs grouped according to their common distance metric. All distance metrics and linkage methods currently available are listed in the SciPy of fastcluster (Müllner, 2013) documentation.

Communities are created after the iterations by a meta-clustering of the most frequent clusters, i.e. top X% or top Y number of clusters. Community construction is performed iteratively through an AHC

*To whom correspondence should be addressed.

approach with a specifically developed distance metric (1) and complete linkage. Complete linkage was chosen because it insures that all clusters or meta-clusters have overlapping objects. The distance between two objects i and j is defined as follows:

$$dist(i,j) = \begin{cases} |root| - 1, \text{if } i \text{ or } j \equiv root \\ |root|, \text{if } \frac{i \cap j}{\max(|i|,|j|)} < minimal\_required\_overlap \\ \max(|i|,|j|) - |i \cap j| \end{cases} \quad (1)$$

The root is the cluster that holds all objects, and all clusters have the distance | root | − 1 to the root. If the overlap between i and j (relative to the largest of the two clusters) is smaller than the minimal overlap required, then the distance is | root |. Otherwise the distance is the size of the larger object (i or j) minus the number of overlapping elements. This ensures that (i) smaller clusters are merged earlier in the hierarchy (closer to the bottom) and (ii) clusters that have a smaller overlap to each other than the threshold will merge after the root, i.e. never into the same branch. After this first iteration, closely related clusters (in terms of their object content) are merged in the hierarchy forming one branch or community starting from the root. In the next iteration each of those branches defines a meta-cluster, which is clustered using the procedure above with an increased required minimal overlap. This cycle is repeated until no change in the hierarchy is observed. Finally, each branch starting from the root node defines one community. The starting threshold is by default 10% plus 5% for each iteration. These values were empirically determined and yielded best results with our data sets (e.g. Höhner *et al*., 2013), i.e. minimizing the merging of unrelated clusters with no direct overlap. However, these values can be defined by the user.

Communities can be visualized as expression maps and profiles. Briefly, expression maps are similar to classical heat maps, except σ is included as box size. Furthermore, the maps include what is called the object community frequency, obCoFreq, which is the sum of all cFreqs for a given object within a community. Finally, pyGCluster allows the results of the meta-clustering to be exported as a DOT file (Gansner and North, 2000). For more details and examples please refer to http://pyGCluster.github.io.

3 CONCLUSIONS

Classical hierarchical clustering requires the selection of a distance metric and linkage method, which imposes a user bias. Furthermore, classical approaches do not include the uncertainty of the values measured, which is an important part of the data.

The pyGCluster overcomes those limitations by evaluating a multitude of DLCs together, whereas performing a high number of iterations during which noise is injected into the data set according to the uncertainty of the data at hand. Therefore, pyGCluster allows (i) the overall resulting data size to be reduced, as only well-defined clusters are retained, and (ii) communities of coregulated objects to be identified in a broader fashion with less user bias.

Recently, the iron homeostasis in *Chlamydomonas reinhardtii* was investigated using quantitative proteomics, and pyGCluster was used to identify coregulated communities with two distance metrics (Euclidean and correlation) and six linkage methods (Höhner *et al*., 2013). Figure 1 shows an alternative visualization of their data highlighting the strength of pyGCluster. Clusters found with purely Euclidean or correlation distance are shown as squares or circles, respectively. Clusters that were found with both distance metrics are shown as triangles. As highlighted with the arrow in Figure 1, some communities are a mix of both distance metrics and would have not been resolved in the same way using classical approaches. This advantage increased the quality of Höhner's *et al*. data analysis.

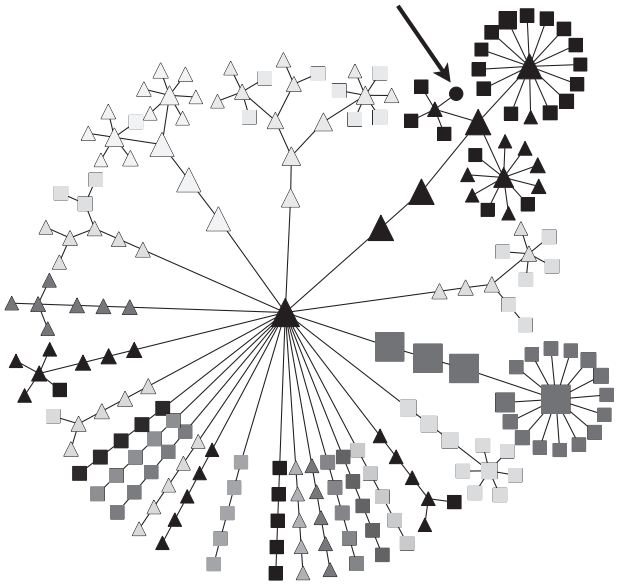

Fig. 1. Node map showing the community assignment of the top 1% clusters from Höhner *et al*. (2013). Nodes represent communities at the internal nodes and clusters at the leaves, their color the community assignment and their shape the distance metric (Euclidean: squares; correlation: circles; both: triangles), which was used to identify the cluster. The arrow highlights a cluster within the black community that was found by correlation distance only, thus highlighting the importance of an unbiased user input, which is provided by pyGCluster

To compare pyGCluster with other stability-based algorithms, the data set of Höhner *et al*. was analyzed with (i) clValid (*cran.r-project.org/package=clValid*), using AHC of the mean data with average linkage and $k = 5$ as determined using stability and internal indices, (ii) the algorithm of Bréhélin *et al*. (2008), where all clusters inside the AHC tree obtained by using the mean data are validated by bootstrapping on the level of repetitions and (iii) the MultiExperiment Viewer (MEV, Saeed *et al*., 2003) that uses a 'Support Tree', in which the clusters of the mean AHC tree are validated by bootstrapping on the level of the mean data only (classical bootstrap). The respective parameters and the argumentation of their choice are summarized in the Supplementary Material. The comparison between those approaches shows that the majority of the objects in clusters obtained by Bréhélin and clValid are covered by pyGCluster (MEV yielded no stable cluster). Moreover, pyGCluster identified overall 107 clusters in 23 communities, which is far more than any of the algorithms mentioned earlier in the text.

Thus pyGCluster offers a novel approach using AHC, whereas removing user bias and overcoming several limitations, in particular exploiting the uncertainty of the data via the noise injection. This makes pyGCluster a novel tool to identify e.g. coregulated proteins or genes in large 'omics' studies.

Funding: By the DFG (to C.F.) is gratefully acknowledged (fu-780/2).

Conflict of Interest: none declared.

REFERENCES

Bréhélin,L. *et al.* (2008) Using repeated measurements to validate hierarchical gene clusters. *Bioinformatics*, **24**, 682–688.

Gansner,E.R. and North,S.C. (2000) An open graph visualization system and its applications to software engineering. *Softw. Pract. Exp.*, **30**, 1203–1233.

Handl,J. *et al.* (2005) Computational cluster validation in post-genomic data analysis. *Bioinformatics*, **21**, 3201–3212.

Höhner,R. *et al.* (2013) The metabolic status drives acclimation of iron deficiency responses in *Chlamydomonas reinhardtii* as revealed by proteomics based hierarchical clustering and reverse genetics. *Mol. Cell. Proteomics*, **12**, 2774–2790.

Müllner,D. (2013) fastcluster: fast hierarchical agglomerative clustering routines for R and Python. *J. Stat. Softw.*, **53**, 1–18.

Saeed,A.I. *et al.* (2003) TM4: a free, open-source system for microarray data management and analysis. *Biotechniques*, **34**, 374–378.

Si,J. *et al.* (2011) Model-based clustering for RNA-seq data. In: *Joint Statistical Meeting*. FL.