Unsupervised consensus for clustering and network problems

Mark D. Humphries[1,2*]

Version 0.1

1. School of Psychology, University of Nottingham, UK
2. Faculty of Biology, Medicine, and Health, University of Manchester, UK

* Corresponding author: mark.humphries@nottingham.ac.uk

Abstract

Clustering algorithms are inherently stochastic, and so are highly likely to return different clusters on each run on the same data-set. Choosing which is best is a hard problem. A solution is to find agreement between these repeated clustering. Here I introduce such a consensus algorithm that is fully unsupervised, having a self-determined convergence to a final answer, and is agnostic to the basic algorithm used to generate the initial set of partitions. I outline how this approach could self-determine the number of clusters, find clusters at multiple-scales, and create a form of discrete hierarchical clustering. This note ends with outstanding problems to solve to achieve all the above robustly.

1 Introduction

Clustering data is a hard problem. Given n objects, the number of possible unique partitions of those objects into between 1 and n clusters is the Bell number. For $n = 100$, the Bell number is $\sim 4.75 \times 10^{115}$. Hence clustering algorithms are inherently stochastic (and we should be suspicious of any that are not), with high probability returning a different partition of the objects on each run. How then do we pick the "best" partition to use as the basis for further analyses?

Consensus approaches attempt to solve this problem by discerning agreement across the multiple partitions of the same data-set (Strehl et al., 2002; Monti et al., 2003; Nguyen and Caruana, 2007; Goder and Filkov, 2008; Lancichinetti and Fortunato, 2012; Jeub et al., 2018). These approaches are now routinely used in genomics analysis pipelines (Senbabaoglu et al., 2014; Kiselev et al., 2017). But knowing when a consensus has been reached is difficult, and is often determined with a user-defined parameter, or by further rounds of user-specified clustering, or both.

Here I introduce a simple self-determined convergence condition for consensus algorithms. This unsupervised consensus algorithm is agnostic to the chosen algorithm used to generate the initial partitions, and I outline how this consensus approach applies to both data clustering and network theory problems, using standard algorithms (spectral clustering, and modularity maximisation, respectively). I outline how this condition could allow a consensus algorithm to infer the number of clusters unsupervised; to automatically uncover multi-scale clusters in a single data-set; and to be naturally extended to hierarchical clustering. We conclude here with outstanding issues to be solved to reach this full potential.

All code is provided at `https://github.com/mdhumphries/HierarchicalConsensusClustering`.

2 An unsupervised consensus algorithm

We assume that our data is expressed as a $n \times n$ affinity matrix $\mathbf{A}$ between n objects, whose entry A_{ij} defines the affinity between objects i and j. Affinity is any measure of similarity (e.g. cosine similarity) or correlation (e.g. Pearson's R) that is not a distance metric (e.g. Euclidean distance). If a network, then A_{ij} is the weight between nodes i and j, and we expect many entries to be zero.

Given any clustering algorithm that can operate on an affinity matrix (see below for specific examples), we obtain an initial partition of n objects into k clusters; repeating the clustering p times gives us p partitions that we denote as the set $\mathbb{P}(k)$. We expect to have a set of partitions $\mathbb{P}(k)$, per tested value of k: $\mathbb{P}(2), \mathbb{P}(3), \ldots, \mathbb{P}(K)$ up to the maximum tested value K.

2.1 The consensus algorithm

Given an initial set of partitions $\mathbb{P}(k)$, the basic consensus clustering algorithm iteratively proceeds as follows:

1. (Optional) From set $\mathbb{P}$ remove all poor-quality partitions by some quality metric. For example, if A is a network, we might reject any partitions that have modularity $Q \leq 0$ as these provide little information. We implemented this check in (Bruno et al., 2015), but not in the present toolbox of code.
2. Make the *consensus matrix* $\mathbf{C}(k)$: entry $C_{ij} = n_{ij}/p$ is the proportion of times each pair of objects (i, j) are placed in the same cluster.
3. Check for convergence of the consensus matrix (see below):
   (a) If converged, stop; return the single clustering defined by the consensus matrix (see below)
   (b) If not converged, cluster the consensus matrix using the chosen clustering algorithm, giving a new set of clusterings $\mathbb{P}(k)$; repeat from step (1).
   (c) If not converged after T iterations, exit with current best answer.

Note we can repeat this for every set of partitions $\mathbb{P}(k)$, over all tested k; or we can pick the best set of partitions $\mathbb{P}(k)$ to take forward – these options are discussed further in section 2.2. Of course, we may have picked a single k to create the initial partitions, in which case we will have a single consensus matrix.

This form of consensus clustering has some simple but pleasing features:

- the values C_{ij} are always in the range $[0, 1]$
- the consensus matrix thus defines another affinity matrix
- thus we have great flexibility in how we tackle clustering $\mathbb{C}(k)$: can use the same algorithm for all steps; or different algorithms for the initial partitions and for $\mathbb{C}(k)$ (i.e. we can choose one known to work well on dense matrices in $[0, 1]$); or the initial partitions could be an ensemble over multiple algorithms.

Convergence Our main contribution here is the following ansatz. The values of the consensus matrix will become bimodal in the presence of any detectable clusters: the distribution around the upper mode will comprise entries corresponding to pairs of objects consistently clustered together because of their shared membership of a common group; the distribution around the lower mode will comprise entries corresponding to pairs of objects clustered together at random (Figure 1). Thus, we propose an unsupervised convergence condition for consensus clustering: that the upper mode will contain a unique clustering when the consensus matrix has converged.

In practice, we check this criterion as follows:

- We divide the entries of $\mathbf{C}(k)$ into two groups $\mathbb{C}^{lower}$ and $\mathbb{C}^{upper}$ corresponding to the two modes. We do this here using Otsu's method for defining two modes in an arbitrary histogram (Otsu, 1979). We bin the values of $\mathbf{C}(k)$ into fixed widths bins (here 0.01), then apply Otsu's method to the histogram to find the threshold $\theta \in [0, 1]$ that splits the two modes. We assign all values of $\mathbf{C}(k) \leq \theta$ to $\mathbb{C}^{lower}$, and all other values to $\mathbb{C}^{upper}$. (Previously, instead of Otsu's method we have also used k-means with $k = 2$ directly on $\mathbf{C}(k)$ to split into the lower and upper mode groups, with initial centroid values determined from $\mathbf{C}(k)$ as the 5th and 95th percentiles of its values).

- We then check for the presence of a single clustering in the upper distribution $\mathbb{C}^{upper}$. We first create two lists: an empty list of assigned objects, and a list of unassigned objects that initially contains all objects.

    1. Picking any objects at random from the unassigned list, we create a cluster containing that object and all other objects with which it has a non-zero entry in $\mathbb{C}^{upper}$.
    2. If any of these objects are already in the assigned list, then the upper distribution does not define a unique clustering. We exit and continue the consensus algorithm.
    3. Otherwise these objects are moved to the assigned list.
    4. Repeat from step (1) until either we have exited (at step 2) or all objects are assigned to a cluster.

Thus, if each object is assigned, then we have produced a single consensus clustering $\mathbb{C}(k)$ of all objects into clusters.

2.2 Multi-scale clustering

If we repeat the above consensus algorithm on each $\mathbb{P}(k)$ separately, then in principle this approach will automatically generate multi-scale clustering, with one consensus clustering $\mathbb{C}(k)$ per k that converged. Note that this then forms an unsupervised method for choosing the appropriate number of clusters k: only those that converge.

There are options for multi-scale clustering here we have yet to explore. In tests on real data, we have always found a consensus partition for at least one k on the first pass. But partitions at other k may exist. Thus instead of exiting as soon as we find at least one consensus clustering, we could instead specify a depth parameter for how many iterations of the consensus clustering to repeat before exiting with all found converged k. In practice, we would remove all k that converged after the initial partitions, then iterate the consensus algorithm down to the chosen depth, removing all k that converged as we go.

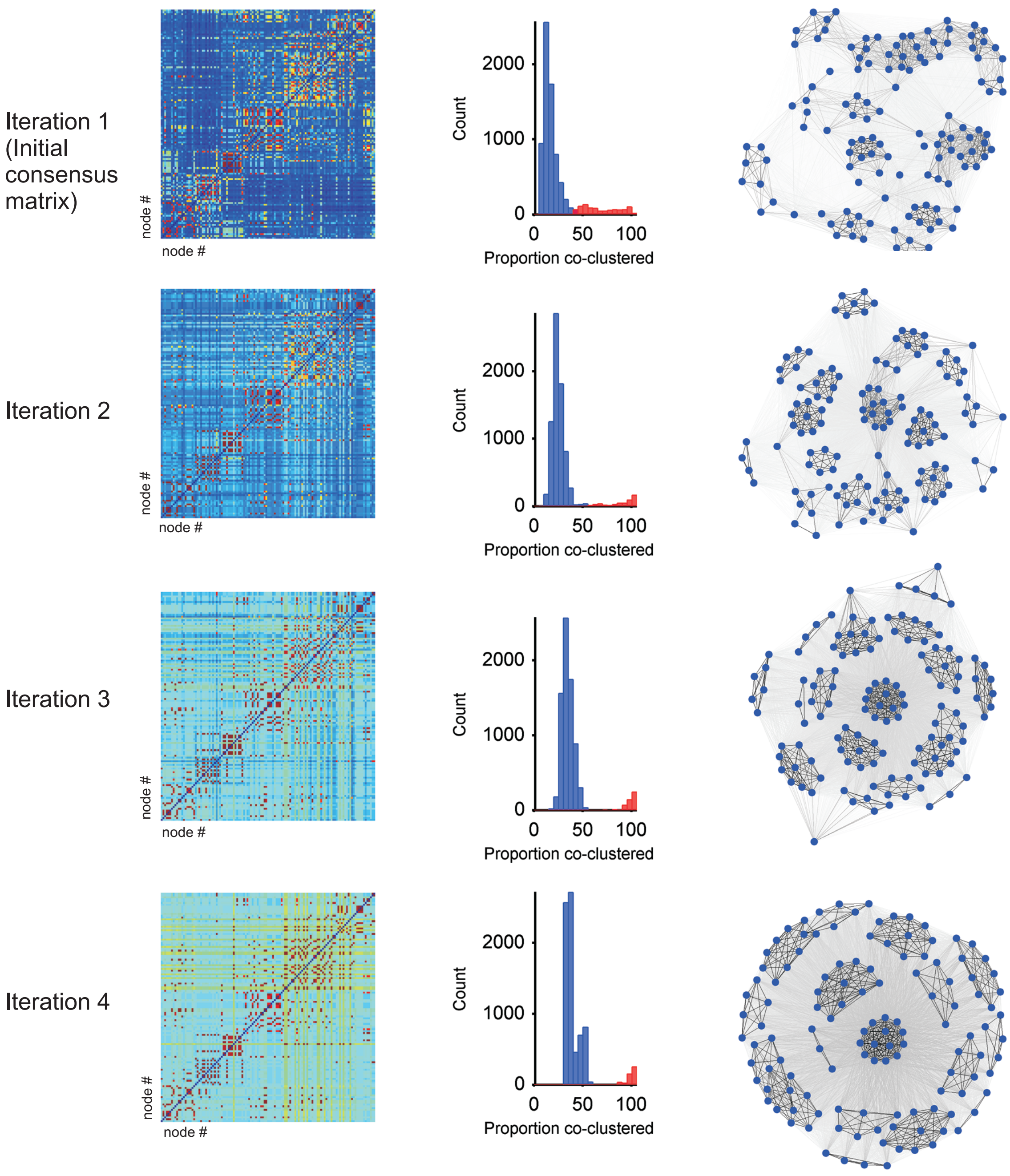

Figure 1: Example output of the consensus algorithm, applied to community detection. Each row shows properties of the consensus matrix **C** for one iteration of the algorithm, from the initial consensus matrix built from the clustering of affinity matrix A, to the final converged matrix after four iterations. The rows depict: left, the consensus matrix visualised as a pseudo-colour plot, colour intensity (blue-red) of each square coding the proportion of times that pair of nodes (i, j) are placed in the same module; middle: histogram of entries in the consensus matrix, colour-coded to show the two distributions ($\mathbb{C}^{lower}$,$\mathbb{C}^{upper}$) found by the k-means clustering of those entries; right: network visualisation of the consensus matrix, with each link's greyscale intensity proportional to the consensus matrix entry for that pair of nodes. The network visualisation clearly shows how the increasing bimodality of the consensus matrix over the iterations corresponds to the separation of the original data-set into a single set of modules.

Note this is either memory hungry (we store every $n \times n$ consensus matrix for each k) or computationally expensive (we regenerate each consensus matrix when needed).

There are also alternatives for creating single-scale clustering:

- Take a single consensus: compute a single consensus matrix $\mathbf{C}_{all}$ by combining partitions $\mathbb{P}(k)$ over all $k = \{2, 3, \ldots, K\}$. We did just this in (Bruno et al., 2015), where the number of partitions was self-determined, and small (typically 12-15, given datasets of between 80 and 180 objects); however this approach may well fail when the n is larger, as the range of tested k may stretch over orders of magnitude, and make no sense to form a single consensus (as they would be dominated by solutions with small k). As the value of C_{ij} almost certainly falls monotonically with increasing k, so a full version of this approach should correct for k used for each partition submitted to the consensus process.

- Pick the best consensus matrix $\mathbf{C}(k)$: we find a cost function that allows us to pick the most informative consensus matrix across all $k = \{2, 3, \ldots, K\}$ to carry forward, and use that as the new A to cluster. For example, we might use the separation of the distribution of values in $\mathbf{C}(k)$ (Senbabaoglu et al., 2014).

2.3 Hierarchical consensus clustering

Data often contain a hierarchical structure of sub-divisions, with each ascending level indicating weaker but meaningful groupings of the data. We can also extend our consensus algorithm ideas to hierarchical clustering.

The simplest approach is if we obtain l multi-scale clusters $\mathbb{C}(k)$ at $k_1 < k_2 < \ldots < k_l$. Then it is possible these form a natural hierarchy, such that the clusters at k_l are the bottom layer, those at k_{l-1} are the next layer, and so on. Determining this would mean checking that the clusters at layer k_{l-c} are either exactly those at layer k_{l-c+1} below it, or are an exact merge of two or more clusters at k_{l-c+1}. Given the stochasticity of clustering, exact matches cannot be guaranteed, so we leave for future work the best way to assess partial matches of merged clusters (using e.g. the Rand index, or variational information).

A second approach is to build a hierarchy by repeatedly clustering from some initial consensus partition $\mathbb{C}_0$ that acts as the bottom layer. We could use here the highest k (i.e. k_l) found by the multi-scale clustering above; or we could force a high k by only choosing to cluster into many groups. Figure 2 schematically illustrates the hierarchical algorithm, which is similar in spirit to the well-known Louvain algorithm (Blondel et al., 2008) for community detection. Beginning with $\mathbb{C}_0$, we form a super-graph by aggregating each cluster into a node, and weighting the links between each pair of nodes by the aggregate affinity between those clusters. Aggregates here could simply be the mean or median affinity between the pair of clusters Applying our consensus clustering algorithm to the super-graph then generates a consensus partition $\mathbb{C}_1$ of the initial clusters. This new partition is the next level of the hierarchy. Iterations of this process recover any hierarchy present, stopping when only two clusters remain.

2.4 Clustering algorithms

Throughout we cluster data in A by first projecting it into a low-dimensional space, then using k-means to identify clusters in that space.

However we project the affinity matrix A, we cluster it using a k-means sweep, with k taking values between a specified lower l and upper u bound. We repeat k-means p times, typically 100, for each value of k, giving a set of partitions $\mathbb{P}(k)$ for each k. Here

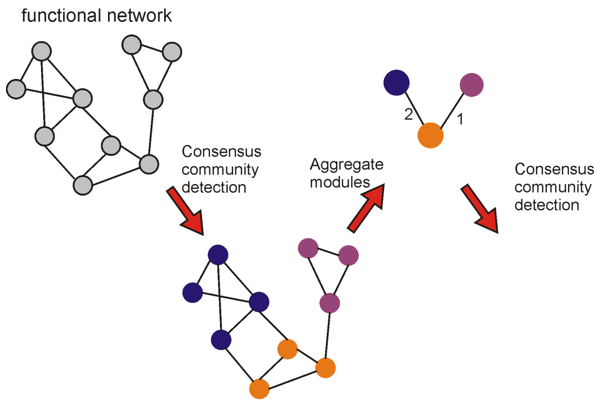

Figure 2: Schematic of hierarchical algorithm. We illustrate the hierarchical algorithm here on the problem of clustering a network A into modules. The initial run of the consensus clustering algorithm divides the original network into modules. We then aggregate the modules into a super-graph describing the network between modules, and apply the consensus clustering algorithm to that super-graph. Any modules detected then indicate groups of modules, giving the next level of the hierarchy. This sequence can be repeated to reveal multiple levels of hierarchy, stopping either when only two modules remain or when no combination of modules is detected.

we use as many dimensions d as clusters k; we leave to future work to explore choosing d independently from k, and to use exhaustive sweeps of every (k, d) combination within the lower and upper bounds (Kiselev et al., 2017). As k-means is the example form of stochastic clustering we use here, it is the p repeats of the k-means clustering at a given k that we submit to the consensus algorithm.

Data clustering For clustering problems, we implemented two spectral approaches. We tested the random-walk Laplacian transform L_{rw} of $\mathbf{A}$ (Luxburg, 2007). We use the top d eigenvectors of L_{rw} for k-means clustering. In the second approach, we also tested simply taking the d lead eigenvectors of $\mathbf{A}$ directly.

Community detection in networks For community detection problems where A is a weighted network, we form the modularity matrix $\mathbf{B} = \mathbf{A} - \gamma\mathbf{P}$, where $P_{ij} = k_i k_j / m$ is the null configuration model for the expected edges between nodes, given the degrees (k_i, k_j) of nodes (i, j) and the total number m of unique edges in the network (Newman, 2006). The resolution parameter γ determines the scale of modules recoverable from $\mathbf{B}$ (Reichardt and Bornholdt, 2006), with $\gamma > 1$ finding smaller modules and $\gamma = 1$ giving the standard modularity matrix (Newman, 2006). We then take the d leading eigenvectors of $\mathbf{B}$ as our input to k-means clustering (Humphries, 2011).

What then should we use as the null model $\mathbf{P}$ for the consensus matrix $\mathbf{C}(k)$? We could use the configuration model, but note that the consensus matrix has a different structure to a network A in the absence of clusters: that the probability of objects (i, j) being co-clustered is uniform for a given k. So we form the consensus null model $P_{ij}^{\text{consensus}} = 1/k$, and construct $\mathbf{B}$ from that.

2.5 Data and code

MATLAB code implementing the consensus algorithms, clustering (projections and k-means sweeps), and scripts for this paper are available at `https://github.com/mdhumphries/HierarchicalConsensusClustering`.

This repository also contains all data used to test the above algorithms.

Data on abstract co-authorship at the annual Computational and Systems Neuroscience (COSYNE) conference were shared with us by Adam Calhoun (personal communication). These data contained all co-authors of abstracts in each of the years 2005 to 2014. From these we constructed a single network, with nodes as authors, and weights between nodes indicating the number of co-authored abstracts in that period.

We obtained the Mouse Brain Atlas of gene co-expression (Ng et al., 2009) from the Allen Institute for Brain Sciences website (`http://mouse.brain-map.org/`), using their API. The Brain Atlas is the expression of 2654 genes in 1299 labelled brain regions. However, these regions are arranged in a hierarchy; we used the 625 individual brain regions at the bottom of the hierarchy as the finest granularity contained in the Atlas. We constructed the gene co-expression network by calculating Pearson's correlation coefficient between the gene expression vectors for all pairs of these 625 brain regions; all correlations were positive.

I have also tested data on gene expression in retinal ganglion cells from (Rheaume et al., 2018); and on 12 properties (e.g. pH) of red and white wines from Portugal (Cortez et al., 2009).

Toy model I implemented a toy model for a hierarchical affinity matrix, to test the accuracy of the clustering algorithms. The model has two layers: the first layer has g_1 groups, each of size t, giving $t \times g_1$ objects in total. The second layer has two groups, of $g_1/2$ objects each (i.e. lower-level merged into two equal size clusters).

Values for the affinity between objects (i, j) were drawn from a truncated Gaussian distribution G^*, limited to the range $[0, 1]$. Note that affinity is symmetric, so we sample for (i, j) and assign the same value for (j, i). Affinities between objects within the g_1 groups are sampled from $G^*(m_1, s_1)$; the affinities within the g_2 groups (i.e. between the merged g_1 groups of that cluster) are sampled from $G^*(m_2, s_2)$. Finally, the remaining affinities between the two second layer groups are sampled from $G^*(m_b, s_b)$. By choosing means such that $m_1 > m_2 > m_b$ we obtain a clear hierarchy. Currently we choose the same standard deviation $s_1 = s_2 = s_b$; but could explore scaling them to the mean.

3 Future work

We finish with a short list of outstanding issues and future directions.

To solve:

- How to pick the upper limit K? We have tried using prior knowledge; pure guessing; setting the upper limit as a fixed percentage of the number of objects n (say 2%); and obtaining a single k by the eigengap heuristic (Luxburg, 2007), of the Laplacian's eigenvalues with the greatest jump between them needless to say, this did not give great results on real data.

- What should the clustering algorithm be? Running the k-means sweep version on the toy model returned the right answer at $k = 2$ but the wrong answer at $k = 4$ (it reliably returned 3 groups). This is partly because k-means returns many rubbish answers at $k = 4$, often returning one much larger group than t, as it tries to meld two within the bottom layer. Consequently, while the four groups are visible in the consensus matrix, there are also high consensus scores between two of the first layer groups and the convergence condition finds these as a single group. (Trying to filter the best k-means clustering at $k = 4$ using the clustering with minimum sum of object distances to the cluster centroids did not work this also returned merged groups). One avenue to explore is thus an alternative to doing a sweep using k-means, or return to more complex algorithms for self-determining a single k.

- How best to choose the modes in the consensus matrix histogram? We use here fixed small bins for histogram, taking advantage of the restricted range to $[0, 1]$. But this could be contributing to poor performance on the toy model, because the bins are not optimal description of distribution of values. This we could try optimal histogram techniques for $\mathbf{C}(k)$, which picks bin sizes according to data concentration (from Wasserman (2004), as implemented in `optimalhistogram.m`).

To try:

- Choosing the best consensus matrix: the most consistent clusterings should produce the most concentrated consensus matrix. This suggests that computing the Shannon entropy of the distribution of the consensus values would be a good guide i.e. $H = -\sum_C p(C) \log_2 p(C)$. Here the lower the entropy H, the more concentrated the distribution. Question remains on how to best discretise the distribution of consensus values.
- Implement the hierarchical clustering properly in code
- Test hierarchical algorithm on data clustering and networks
- Network from toy model: obtaining an undirected weighted hierarchical network A from the toy model could be achieved by building the full model and then making it sparse by e.g. deleting ρ proportion of the (i, j) entries (and corresponding (j, i)) uniformly at random.

Acknowledgements

We thank Adam Calhoun for permission to use his collated COSYNE submission dataset, and Mat Evans for formatting and computing the Allen Gene Expression data. This work was supported by the Medical Research Council [grant numbers MR/J008648/1 and MR/P005659/1].

Author Contributions

MDH: wrote paper; developed conceptual framework; coded algorithms; ran analyses

References

Blondel, V. D., Guillaume, J.-L., Lambiotte, R., and Lefebvre, E. (2008). Fast unfolding of communities in large networks. *J. Stat. Mech*, page P10008.

Bruno, A. M., Frost, W. N., and Humphries, M. D. (2015). Modular deconstruction reveals the dynamical and physical building blocks of a locomotion motor program. *Neuron*, 86:304–318.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., and Reisa, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision Support Systems*, 47:547–553.

Goder, A. and Filkov, V. (2008). Consensus clustering algorithms: Comparison and refinement. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, pages 109–117. Society for Industrial and Applied Mathematics.

Humphries, M. D. (2011). Spike-train communities: finding groups of similar spike trains. *J Neurosci*, 31:2321–2336.

Jeub, L. G. S., Sporns, O., and Fortunato, S. (2018). Multiresolution consensus clustering in networks. *Sci Rep*, 8:3259.

Kiselev, V. Y., Kirschner, K., Schaub, M. T., Andrews, T., Yiu, A., Chandra, T., Natarajan, K. N., Reik, W., Barahona, M., Green, A. R., and Hemberg, M. (2017). Sc3: consensus clustering of single-cell rna-seq data. *Nature Methods*, 14:483–486.

Lancichinetti, A. and Fortunato, S. (2012). Consensus clustering in complex networks. *Scientific Reports*, 2:336. How to make good use of each re-clustering of a network...

Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and Computing*, 17(4):395–416.

Monti, S., Tamayo, P., Mesirov, J., and Golub, T. (2003). Consensus clustering: A resampling-based method for class discovery and visualization of gene expression microarray data. *Mach. Learn.*, 52:91–118.

Newman, M. E. J. (2006). Finding community structure in networks using the eigenvectors of matrices. *Phys Rev E*, 74:036104.

Ng, L., Bernard, A., Lau, C., Overly, C. C., Dong, H.-W., Kuan, C., Pathak, S., Sunkin, S. M., Dang, C., Bohland, J. W., Bokil, H., Mitra, P. P., Puelles, L., Hohmann, J., Anderson, D. J., Lein, E. S., Jones, A. R., and Hawrylycz, M. (2009). An anatomic gene expression atlas of the adult mouse brain. *Nat Neurosci*, 12:356–362.

Nguyen, N. and Caruana, R. (2007). Consensus clusterings. In *Proceedings of the 2007 Seventh IEEE International Conference on Data Mining*, ICDM '07, pages 607–612, Washington, DC, USA. IEEE Computer Society.

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Trans Sys Man Mach*, 9:62–66.

Reichardt, J. and Bornholdt, S. (2006). Statistical mechanics of community detection. *Phys Rev E*, 74:016110.

Rheaume, B. A., Jereen, A., Bolisetty, M., Sajid, M. S., Yang, Y., Renna, K., Sun, L., Robson, P., and Trakhtenberg, E. F. (2018). Single cell transcriptome profiling of retinal ganglion cells identifies cellular subtypes. *Nat Commun*, 9:2759.

Senbabaoglu, Y., Michailidis, G., and Li, J. Z. (2014). Critical limitations of consensus clustering in class discovery. *Sci Rep*, 4:6207.

Strehl, A., Ghosh, J., and Cardie, C. (2002). Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617. Cool, but objective function for finding clusters in consensus matrix requires supplying k.

Wasserman, L. (2004). *All of Statistics: A Concise Course in Statistical Inference.* Springer, New York.