progenyClust: an R package for Progeny Clustering

by Chenyue W. Hu and Amina A. Qutub

Abstract Identifying the optimal number of clusters is a common problem faced by data scientists in various research fields and industry applications. Though many clustering evaluation techniques have been developed to solve this problem, the recently developed algorithm *Progeny Clustering* is a much faster alternative and one that is relevant to biomedical applications. In this paper, we introduce an R package **progenyClust** that implements and extends the original *Progeny Clustering* algorithm for evaluating clustering stability and identifying the optimal cluster number. We illustrate its applicability using two examples: a simulated test dataset for proof-of-concept, and a cell imaging dataset for demonstrating its application potential in biomedical research. The **progenyClust** package is versatile in that it offers great flexibility for picking methods and tuning parameters. In addition, the default parameter setting as well as the plot and summary methods offered in the package make the application of *Progeny Clustering* straightforward and coherent.

Introduction

Clustering is a classical and widely-used machine learning technique, yet the field of clustering is constantly growing. The goal of clustering is to group objects that are similar to each other and separate objects that are not similar to each other based on common features. Clustering can, for example, be applied to distinguishing tumor subclasses based on gene expression data (Sørlie et al., 2001; Budinska et al., 2013), or dividing sport fans based on their demographic information (Ross, 2007). One critical challenge in clustering is identifying the optimal number of groups. Despite some advanced clustering algorithms that can automatically determine the cluster number (e.g. Affinity Propagation (Frey and Dueck, 2007)), the commonly used algorithms (e.g. k-means (Hartigan and Wong, 1979) and hierarchical clustering (Johnson, 1967)) unfortunately require users to specify the cluster number before performing the clustering task. However, most often than not, the users do not have prior knowledge of the number of clusters that exist in their data.

To solve this challenge of finding the optimal cluster number, quite a few clustering evaluation techniques (Arbelaitz et al., 2013; Charrad et al., 2014a) as well as R packages (e.g. **cclust** (Dimitriadou et al., 2015), **clusterSim** (Walesiak et al., 2015), **cluster** (Maechler et al., 2015), **Nbclust** (Charrad et al., 2014b), **fpc** (Hennig, 2015)) were developed over the years to objectively assess the clustering quality. The problem of identifying the optimal cluster number is thus transformed into the problem of clustering evaluation. In most of these solutions, clustering is first performed on the data with each of the candidate cluster numbers. The quality of these clustering results is then evaluated based on properties such as cluster compactness (Tibshirani et al., 2001; Rousseeuw, 1987) or clustering stability (Ben-Hur et al., 2001; Monti et al., 2003). In particular, stability-based methods have been well received and greatly promoted in recent years (Meinshausen and Bühlmann, 2010). However, these methods are generally slow to compute because of the repetitive clustering process mandated by the nature of stability assessment. Recently, a new method *Progeny Clustering* was developed by Hu et al. (2015) to assess clustering quality and to identify the optimal cluster number based on clustering stability. Compared to other clustering evaluation methods, *Progeny Clustering* requires fewer samples for clustering stability assessment, thus it is able to greatly boost computing efficiency. However, this advantage is based on the assumption that features are independent for each cluster, thus users need to either transform data and create independent features or consult other methods if this assumption does not hold for the data of interest.

Here, we introduce a new R package, **progenyClust**, that performs *Progeny Clustering* for continuous data. The package consists of a main function `progenyClust()` that requires few parameter specifications to run the algorithm on any given dataset, as well as a built-in function `hclust.progenyClust` to use hierarchical clustering as an alternative to using `kmeans`. Two example datasets `test` and `cell`, used in the original publication of *Progeny Clustering*, are provided in this package for testing and sharing purposes. In addition, the **progenyClust** package includes an option to invert the stability scores, which is not considered in the original algorithm. This additional capability enables the algorithm to produce more interpretable and easier-to-plot results. The rest of the paper is organized as follows: We will first describe how *Progeny Clustering* works and then go over the implementation of the **progenyClust** package. Following the description of functions and datasets provided by the package, we will provide one proof-of-concept example of how the package works and a real world example where the package is used to identify cell phenotypes based on imaging data.

Progeny Clustering

In this section, we briefly review the algorithm of *Progeny Clustering* (Hu et al., 2015). *Progeny Clustering* is a clustering evaluation method, thus it needs to couple with a stand-alone clustering method such as *k-means*. The framework of *Progeny Clustering* is similar to other stability based methods, which select the optimal cluster number that renders the most stable clustering. The evaluation of clustering stability usually starts with an initial clustering of the full or sometimes partial dataset, followed by bootstrapping and repetitive clustering, and then uses certain criterion to assess the stability of clustering solutions. *Progeny Clustering* uses the same workflow, but innovates at the bootstrapping method and improves on the stability assessment.

Consider a finite dataset $\{x_{ij}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$ that contains M variables (or features) for N independent observations (or samples). Given a number K (a positive integer) for clustering, a clustering method partitions the dataset into K clusters. Each cluster is denoted as C_k, $k = 1, \ldots, K$. Inspired by biological concepts, each cluster is treated as a subpopulation and the bootstrapped samples as progenies from that subpopulation. The uniqueness of *Progeny Sampling* during the bootstrapping step is that it randomly samples feature values with replacement to construct new samples rather than directly sampling existing samples. Let $\tilde{N}$ be the number of progenies we generate from each cluster C_k. Combining these progenies, we have a validation dataset $\{y_{ij}^{(k)}\}$, $i = 1, \ldots, \tilde{N}$, $j = 1, \ldots, M$, $k = 1, \ldots, K$, containing $K \times \tilde{N}$ observations with M features. Using the same number K and the same method for clustering, we partition the progenies $\{y_{ij}^{(k)}\}$ into K progeny clusters, denoted by C'_k, $k = 1, \ldots, K$. A symmetric co-occurrence matrix Q records the clustering memberships of each progeny as follows:

$$Q_{ab} = \begin{cases} 1, & \text{if the } a^{th} \text{ progeny and the } b^{th} \text{ progeny are in the same cluster } C'_k \\ 0, & \text{otherwise} \end{cases}. \tag{1}$$

The progenies in Q were ordered by the initial cluster (C_k) they were generated from, such that $Q_a, \ldots, Q_{a+\tilde{N}} \in C_k$, $a = (k-1)\,\tilde{N}$. After repeating the above process (from generating *Progenies* to obtaining Q) R times, we can get a series of co-occurrence matrices $Q^{(r)}$, $r = 1, \ldots, R$. Averaging $Q^{(r)}$ results in a stability probability matrix P, i.e.

$$P_{ab} = \sum_r Q_{ab}^{(r)} / R \ . \tag{2}$$

From this probability matrix P, we compute the stability score for clustering the dataset $\{x_{ij}\}$ into K clusters as

$$S = \frac{\sum_k \sum_{a,b \in C_k, b \neq a} P_{ab} / \left(\tilde{N} - 1\right)}{\sum_k \sum_{a \in C_k, b \notin C_k} P_{ab} / \left(K\tilde{N} - \tilde{N}\right)}. \tag{3}$$

A schematic for this process and the pseudocode are shown in Figure 1 and Figure 2.

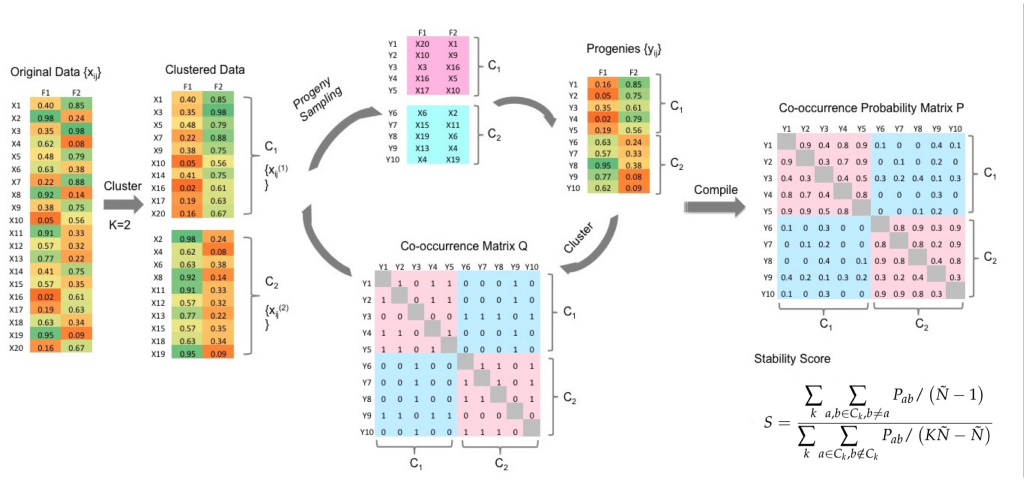

Figure 1: The schematic of the core steps in *Progeny Clustering*, illustrated using an example of clustering a 20×2 matrix into two groups. Schematic reproduced from Hu et al. (2015) under a Creative Commons License.

Input $\{x_{ij}\}$, K_{min}, K_{max}, $\tilde{N}$, R
for $K = K_{min}$ to K_{max} **do**
  Cluster $\{x_{ij}\}$ into K clusters
  Initiation Q = Zero Matrix of size $N \times N$
  for $r = 1$ to R **do**
    for $k = 1$ to K **do**
      Construct $\{y_{ij}^{(k)}\}$ of size $\tilde{N}$ by *Progeny Sampling* from $\{x_{ij}^{(k)}\}$
    end for
    Cluster $\{y_{ij}\}$ into K clusters
    $Q = Q + Q^{(r)}$
  end for
  $P^{(K)} = Q/R$
  Compute $\{S^{(K)}\}$ by Equation 3
end for
Output $\{S^{(K)}\}$

Figure 2: The pseudo code of the *Progeny Clustering* algorithm, from Hu et al. (2015).

After computing the stability score for each cluster number, we can then pick the optimal number using a 'greatest score' criterion or a 'greatest gap' criterion or both. The 'greatest score' criterion selects the cluster number that produces the highest stability score compared to reference datasets, similar to what's used in *Gap Statistics* (Tibshirani et al., 2001). T reference datasets are first generated from a uniform distribution over the range of each feature using Monte Carlo simulation. Each reference dataset is then treated as an input dataset, and stability scores are computed respectively using the same process as in Figure 1. Let $\{\tilde{S}^{(K)(t)}\}$, $t = 1, \ldots, T$, be the stability score for clustering the t^{th} reference dataset into K clusters. The stability score difference between the original dataset and reference datasets are obtained by

$$D^{(K)} = S^{(K)} - \sum_t \tilde{S}^{(K)(t)}/T, \tag{4}$$

where $K = K_{min}, \ldots, K_{max}$. The optimal cluster number with the greatest score difference is then selected, i.e.

$$K_o = \arg\max D^{(K)}. \tag{5}$$

While the 'greatest score' criterion requires computing stability scores from random datasets, the 'greatest gap' criterion does not, due to the fact that the stability score linearly increases with an increase in cluster number among reference datasets. The 'greatest gap' criterion therefore searches for peaks in the stability score curve and selects the cluster number that has the highest stability score compared to those of its neighboring numbers, i.e.

$$K_o = \arg\max \left(2S^{(K)} - S^{(K-1)} - S^{(K+1)}\right). \tag{6}$$

Compared to other stability-based evaluation methods, the major benefits of using *Progeny Clustering* include less re-use of the same samples and faster computation. The progenies sampled from the original data resemble but are hardly the same as the original samples. Thanks to this unique feature, a small number of progenies are sufficient to evaluate the clustering stability. The reduction of sample size for evaluation in turn saves substantial computing time, because the complexity of most clustering algorithms is dependent on the sample size (Andreopoulos et al., 2009). The proposal of the 'greatest gap' criterion further boosts computation speed of clustering evaluation by eliminating the step of generating reference scores. The comparison of computation speed between *Progeny Clustering* and other commonly used algorithms can be found in Hu et al. (2015).

The progenyClust package

The **progenyClust** package was developed with the aim of enabling and promoting the usage of the *Progeny Clustering* algorithm in the R community. This package implements the *Progeny Clustering* algorithm with an additional feature to invert stability scores. The package includes a main function `progenyClust()`, plot and summary methods for "progenyClust" objects, a function

`hclust.progenyClust` for hierarchical clustering, and two example datasets. To perform *Progeny Clustering* using the **progenyClust** package, users should first run the main function `progenyClust()` on their dataset, then use plot and summary methods to check the stability score curves, review the clustering results, and check the recommended cluster number. The `progenyClust()` function allows flexible plug-ins of various clustering algorithms into *Progeny Clustering*, and directly couples with *k-means* clustering algorithm as a default as well as hierarchical clustering as an alternative. Since the clustering memberships are returned in addition to the optimal cluster number, the package integrates the clustering process and the cluster number selection process into one, and it saves users additional efforts that are required to complete clustering tasks. In the following sections, we will first explain the motivation to provide score inversion, then go over the main `progenyClust()` function, the "S3" methods for "progenyClust" objects, the built-in function `hclust.progenyClust()`, and describe the background of the included datasets.

Inversion of the stability scores

In the original *Progeny Clustering* algorithm, the optimal cluster number was chosen based on stability scores, which capture the true classification rate over the false classification rate. The higher the score is, the more stable the clustering is, and the more desirable the cluster number is. The computation of stability scores works well in general, except for when the false classification rate is equal to zero. The zero false classification rate indicates a perfectly stable clustering, that is when all progenies are correctly clustered with progenies coming from the same initial cluster. The perfectly stable clustering will produce a positive infinite stability score, which is not ideal for plotting or for further computing to select the optimal cluster number. Therefore, we offer a choice of inverting the stability scores in this package to mitigate the risk of generating an infinite score. The inverted stability scores can be interpreted as a measure of instability, calculated by false classification rate over true classification rate. In the case of a perfectly stable clustering, the inverted stability score is equal to zero, thus is much easier for comparison and visualization. Meanwhile, the chances of a perfectly unstable clustering are much rarer. If the inversion of stability score is chosen when running *Progeny Clustering*, users should select the cluster number with the smallest score instead of the greatest score.

The `progenyClust()` function

The `progenyClust()` function takes in a data matrix, performs *Progeny Clustering*, and outputs a "progenyClust" object. The clustering is performed on rows, thus the input data matrix needs to be formatted accordingly. A number of input arguments were offered by `progenyClust()` to allow users to specify the clustering algorithm, cluster number selection criterion and parameter values they want to use for *Progeny Clustering*. The output "progenyClust" object contains information on the clustering memberships and stability scores at each cluster number, and it can work with the `plot` and `summary` methods. Since the default values for most of the input arguments are provided, `progenyClust()` can be run without any tuning. The function is used as follows:

```
progenyClust(data, FUNclust = kmeans, method = 'gap', score.invert = F,
        ncluster = 2:10, size = 10, iteration = 100, repeats = 1, nrandom = 10, ...)
```

Here, we group the input arguments into three categories, and highlight the meaning and usage of each argument.

- **Input Data:** `data` is a matrix, the rows of which are of interest to cluster. `ncluster` is a sequence of candidate cluster numbers to evaluate.
- **Method:** Since `progenyClust()` is a clustering evaluation algorithm, it needs to work together with a clustering algorithm. `FUNclust` is where the clustering function is specified. The input and output of `FUNclust` is required to be similar to the default `kmeans()` function from **stat**, or the alternative `hclust.progenyClust()` function for hierarchical clustering as provided in **progenyClust**. `FUNclust` should be able to accept `data` as its first argument, accept the number for clustering as its second argument, and return a list containing a component `cluster` which is a vector of integers denoting the clustering assignment for each sample. `method` is the stability score comparison criterion being selected. `score.invert` can be used to flip the stability scores to instability scores when specified to be `TRUE`. The values of `method` can be 'gap' which represents the 'greatest gap' criterion, 'score' which represents the 'greatest score' criterion, or 'both' which represents using both the 'greatest gap' and the 'greatest score' criteria. In cases when optimal cluster numbers determined by the 'greatest gap' and the 'greatest score' do not agree, we suggest users to either review the stability score plots from both criteria and pick the most preferred one or use the cluster number suggested by the 'greatest score' criterion.

- **Tuning Parameters:** `size` specifies the number of progenies to generate from each initial cluster for stability evaluation. `iteration` denotes how many times the progenies are generated for calculating the stability score. `repeats` is the number of times the entire algorithm should be repeated from the initial clustering to obtaining the stability scores. If `repeats` is greater than one, the standard deviation of the stability score at each cluster number will be produced. `nrandom` specifies the number of random datasets to generate when computing the reference scores, if the 'greatest score' method is chosen. All these tuning parameters, if specified inappropriately, can affect the accuracy and computing efficiency of the *Progeny Clustering* algorithm. In general, the greater the values of `size`, `iteration`, `repeats` and `nrandom` are, the slower the computing will be.

The output of the `progenyClust()` function is an object of the "progenyClust" class, which contains information on the clustering results, the stability scores computed and the calls that were made. Specifically, `cluster` is a matrix of clustering memberships, where rows are samples and columns are cluster numbers; `mean.gap` and `mean.score` are the scores computed at each given cluster number and normalized based on the 'greatest gap' and the 'greatest score' criteria; `score` and `random.score` are the initial stability scores computed before using any criteria to normalize; `sd.gap` and `sd.score` are the standard deviations of the scores when the input argument `repeats` is specified to be greater than one; `call`, `ncluster`, `method` and `score.invert` return the call that was made and input arguments specified.

The `plot` and `summary` methods for "progenyClust" objects

To identify the optimal cluster number, we provide the S3 `plot` and `summary` methods for "progenyClust" objects. The `plot` method enables users to visualize stability scores for cluster number selection and to visualize the clustering results. The `plot` function is as follows:

```
plot(x, data = NULL, k = NULL, errorbar = FALSE, xlab = '', ylab = '', ...)
```

If `data` is not provided, the function will visualize the stability score at each investigated cluster number to give users an overview of the clustering stability. When `data` is provided, the function will visualize `data` in scatter plots and represent each cluster membership by a distinct color. `data` can be the orginal data matrix used for clustering or a subset of the original data with fewer variables but the same number of samples. Additional graphical arguments can be passed to customize the plot. The only extra input argument we added here is `errorbar`, which will render error bars when plotting stability scores if `errorbar = TRUE`. The `errbar` function from **Hmisc** (Harrell Jr and Harrell Jr, 2015) was used to generate the error bars. In addition, the `summary` method of the "progenyClust" object produces a quick summary of what number of clusters is the best to use for the given data.

The `hclust.progenyClust()` function

The `hclust.progenyClust()` function performs hierarchical clustering by combining three existing R functions `dist()`, `hclust()` and `cutree()` from **stat** into one. The input and output are formatted such that they can be directly plugged into the `progenyClust()` function as an option for `FUNclust`, similar to the default `kmeans()` function. The function is as follows:

```
hclust.progenyClust(x, k, h.method = 'ward.D2', dist = 'euclidean', p = 2, ...)
```

To ensure consistency between similar R functions and allow users to easily use this function, the input arguments are largely kept the same as the ones used in fucntions `dist()`, `hclust()`, `cutree()`. The function returns clustering memberships, an `hclust` object of the tree, and a `dist` object of the distance matrix.

The `test` and `cell` datasets

A couple of datasets from the original paper on *Progeny Clustering* (Hu et al., 2015) were included in the **progenyClust** package for testing and sharing purposes. As a proof-of-concept example, `test` was a simulated dataset to help users quickly test the algorithm and see how it works. The dataset was generated by randomly drawing 50 samples from bivariate normal distributions with a common identity covariance matrix and a mean at (-1,2), (2,0) and (-1,-2) respectively. Thus, `test` is a 150 by 2 matrix that contains three clusters.

The dataset `cell`, generated experimentally from Slater et al. (2015), contains 444 cell samples and the first three principal components of their morphology metrics. Since the cells were engineered into 4 distinct morphological phenotypes, this dataset in theory should contain 4 clusters. More experimental details of this dataset can be found in Slater et al. (2015) and Hu et al. (2015).

Examples

In this section, we demonstrate the use of the **progenyClust** package in two examples. The first example is a proof-of-concept of how **progenyClust** works on a simulated test dataset. The second example demonstrates the biomedical application of **progenyClust** to identify the number of cell phenotypes based on cell imaging data.

Proof-of-concept example

To show how the `progenyClust()` function works, we use the dataset `test` included in the **progenyClust** package as the input dataset. The goal here is to find the inherent number of clusters present in this dataset, which is known to be three. Since most of the parameters have default values, we can run the `progenyClust()` function for this dataset with the default setting. The R code is as follows:

```
require(`progenyClust')
data(test)
set.seed(1)

## run Progeny Clustering with default parameter setting
test.progenyClust<-progenyClust(test)

## plot stability scores computed by Progeny Clustering
plot(test.progenyClust)

## plot clustering results at the optimal cluster number (default)
plot(test.progenyClust, test)

## report the optimal cluster number
summary(test.progenyClust)

## output from the summary
Call:
progenyClust(data = test)

Optimal Number of Clusters:
gap criterion - 3
```

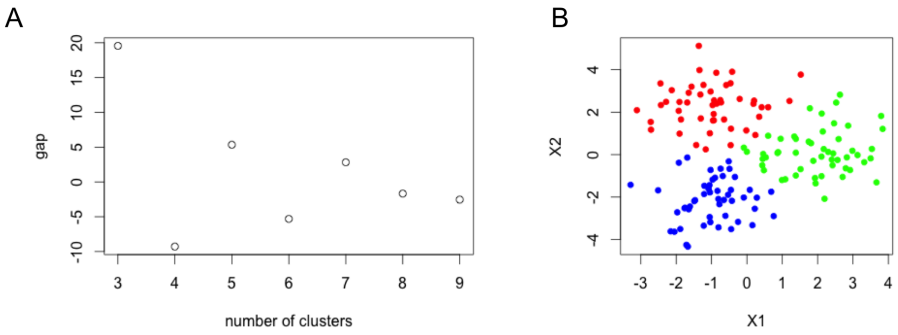

Figure 3: Plots of the "progenyClust" object from clustering the `test` dataset under the default setting. (A) Normalized stability scores based on the 'greatest gap' method were shown at each cluster number. The greater the stability score is, the closer the cluster number matches the true cluster number. (B) The clustered `test` data is shown with the optimal number of clusters.

The summary of the "progenyClust" object concludes that the optimal number for clustering this `test` dataset is three, which agrees with the fact that the dataset was generated from three centers. The plot result of the "progenyClust" object alone is shown in Figure 3A, displaying a curve of normalized stability scores for all candidate numbers of clusters except for the minimum and maximum. This score curve can provide us with insights of clustering quality at all cluster numbers, and help us identify the second preferred number of clusters if needed. Using the `test` data as input, the `plot()`

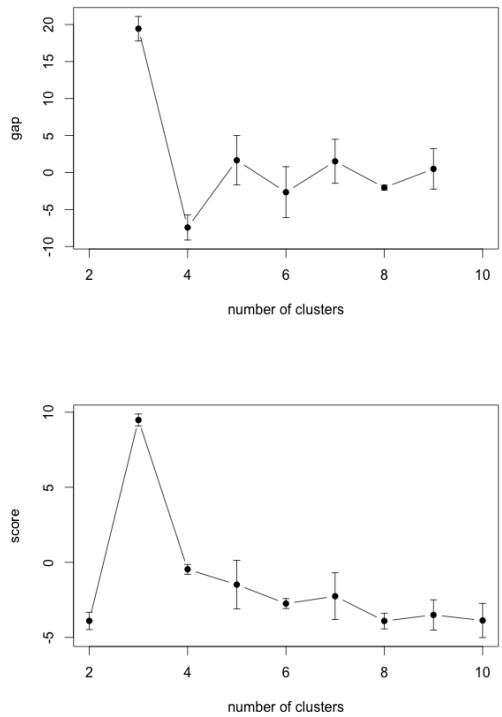

Figure 4: Plot of the "progenyClust" object from running `progenyClust()` on the `test` dataset three times with both evaluation methods, 'greatest gap' (top) and 'greatest score' (bottom). The score curves from both methods estimated that the number of three clusters is best for this dataset. The plot was customized to display the error bars.

function visualizes the data in a scatter plot with three colors, where each corresponds to a cluster (Figure 3B).

Though the default setting of `progenyClust()` function works well in this example, for the purpose of illustrating the capabilities of the function, we will change the input argument values and tune the algorithms slightly. For example, due to the theoretical shortage of the 'greatest gap' criterion, the user might want to obtain estimation from both the 'greatest gap' and the 'greatest score' methods. Though the 'greatest score' method will slow down the algorithm because of the laborious process of generating reference scores, it can evaluate clustering stabilities at the minimum and maximum potential cluster numbers which are ignored by the 'greatest gap' method. The R code for the altered version is shown below. Here, we also change the input argument `repeats` to repeat the algorithm three times instead of one time to obtain standard deviations of the stability scores.

```
set.seed(1)

## run Progeny Clustering with both methods and repeated three times
test2.progenyClust<-progenyClust(test, method = 'both', repeats = 3)

## plot with error bars and summarize the output progenyClust object
plot(test2.progenyClust, errorbar = TRUE, type = 'b')
summary(test2.progenyClust)

## output from the summary
Call:
progenyClust(data = test, method = "both", repeats = 3)

Optimal Number of Clusters:
gap criterion - 3
score criterion -  3
```

It is clear from both the summary and the score curve plots (Figure 4) that both methods agree on the optimal cluster number being three. Specifically, the S3 `plot` method automatically plots two

score curves if the "progenyClust" object was generated with `method = 'both'`. Using the `errbar()` function from **Hmisc**, the S3 `plot` method is able to display error bars with `errorbar = TRUE`.

Application to identifying cell phenotypes

Clustering is a useful technique for the biomedical community, and it can be widely applied to various data-driven research projects. As a second example, we illustrate here how the **progenyClust** package can be used to identify the number of cell phenotypes based on the morphology metrics derived from cell images. In this experiment, biomedical researches used a special technique called "Image Guided Laser Scanning Lithography (LG-LSL)" (Slater et al., 2011) to pattern cells into four shapes. Images of all patterned cells were taken, and morphology metrics were derived to study cytoskeletal and nuclei features of patterned cells. Finding the cell clusters based on their imaging data is of particular interest in this case, and *Progeny Clustering* can help estimate the optimal number for clustering.

Similar to the first example, applying *Progeny Clustering* to the `cell` dataset using the **progenyClust** package is straightforward. The R code is shown below. Here, we use the built-in function `hclust.progenyClust` as `FUNclust` to run the algorithm with hierarchical clustering instead of the default `kmeans`, and we select the optimal cluster number based on the 'greatest gap' criterion. The `plot` and `summary` methods are used to show the output scores and the estimated optimal cluster number. From the output result (Figure 5A), we can see that clustering the cells into four groups has the highest stability, which matches the four patterned cell shapes included in this dataset. The clustering results are shown in Figure 5B in a table of scatter plots for each pairing of variables. Since the cell patterns were engineered, we are fortunate in this example to have prior knowledge of the true number of clusters and to easily test clustering algorithms. However, in a lot of similar biological experiments (e.g. collected tumor cells), we do not possess the knowledge of the true cluster number. In these cases, **progenyClust** can come in handy to identify the optimal cluster number to divide the cells into, and subsequent analyses are then possible for characterizing each cell cluster and discovering its biological or clinical impact.

```
data(cell)
set.seed(1)

## run Progeny Clustering with hierarchical clustering
cell.progenyClust<-progenyClust(cell, hclust.progenyClust)

## plot stability scores, clustering results at optimal cluster number, and summarize results
plot(cell.progenyClust, type = 'b')
plot(cell.progenyClust, cell)
summary(cell.progenyClust)

## output from the summary
Call:
progenyClust(data = cell, FUNclust = hclust.progenyClust)

Optimal Number of Clusters:
gap criterion - 4
```

Summary

This paper introduces the R package **progenyClust**, which identifies the optimal cluster number for any given dataset based on the *Progeny Clustering* algorithm. Improving on the original algorithm, **progenyClust** provides the option to invert stability scores to instability scores, thus preventing the generation of infinite scores in a perfectly stable clustering solution. A variety of parameters (including the clustering method, the evaluation method and the size of progenies) are offered by the package and can be easily adjusted for *Progeny Clustering*. In addition, the default parameter setting specified by the package allows users to perform the algorithm with little background knowledge and parameter tuning. Thanks to the superior computing efficiency of *Progeny Clustering*, this package is a faster alternative to traditional clustering evaluation methods, and it can benefit R communities in biomedicine and beyond.

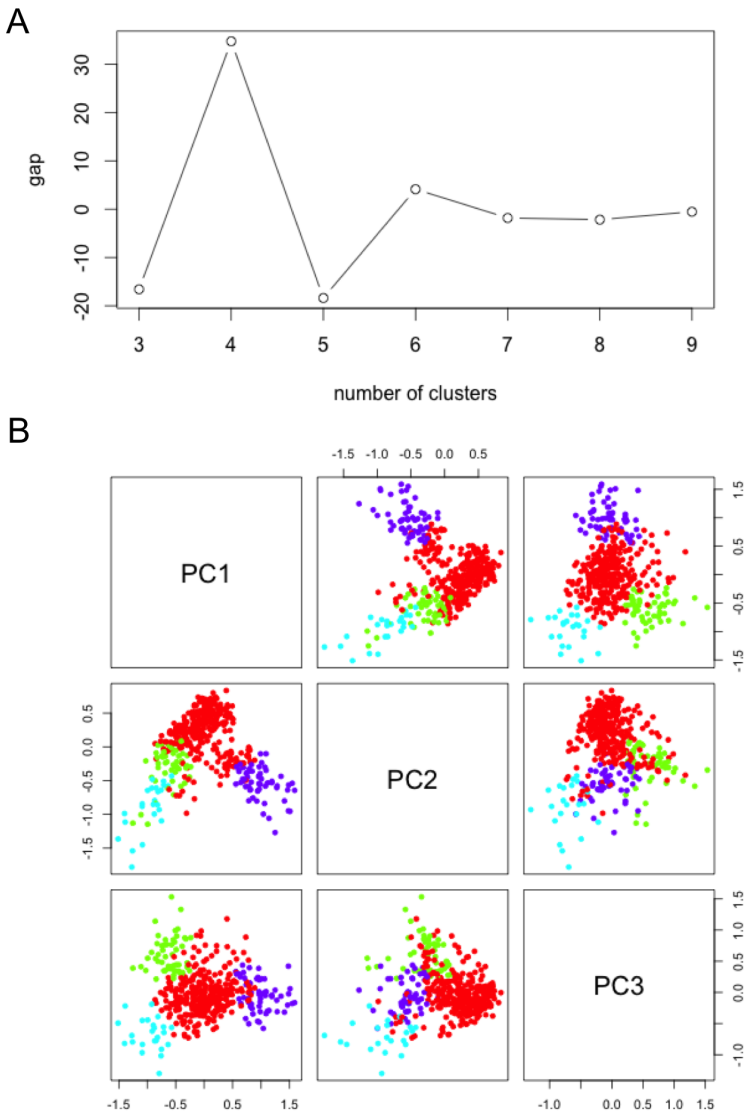

Figure 5: Plots of the "progenyClust" object from running `progenyClust()` on the `cell` dataset with hierarchical clustering. (A) The score curve shows that the `cell` data is best clustered with four clusters. (B) The clustering results with four clusters are shown in a table of scatter plots.

Bibliography

B. Andreopoulos, A. An, X. Wang, and M. Schroeder. A roadmap of clustering algorithms: finding a match for a biomedical application. *Briefings in Bioinformatics*, 10(3):297–314, 2009. [p3]

O. Arbelaitz, I. Gurrutxaga, J. Muguerza, J. M. Pérez, and I. Perona. An extensive comparative study of cluster validity indices. *Pattern Recognition*, 46(1):243–256, 2013. [p1]

A. Ben-Hur, A. Elisseeff, and I. Guyon. A stability based method for discovering structure in clustered data. In *Pacific Symposium on Biocomputing*, volume 7, pages 6–17, 2001. [p1]

E. Budinska, V. Popovici, S. Tejpar, G. D'Ario, N. Lapique, K. O. Sikora, A. F. Di Narzo, P. Yan, J. G. Hodgson, S. Weinrich, et al. Gene expression patterns unveil a new level of molecular heterogeneity in colorectal cancer. *The Journal of Pathology*, 231(1):63–76, 2013. [p1]

M. Charrad, N. Ghazzali, V. Boiteau, and A. Niknafs. Nbclust: an R package for determining the relevant number of clusters in a data set. *Journal of Statistical Software*, 61(6):1–36, 2014a. [p1]

M. Charrad, N. Ghazzali, V. Boiteau, A. Niknafs, and M. M. Charrad. Package 'nbclust'. *J. Stat. Soft*, 61:1–36, 2014b. [p1]

E. Dimitriadou, K. Hornik, and M. K. Hornik. Package 'cclust'. 2015. [p1]

B. J. Frey and D. Dueck. Clustering by passing messages between data points. *Science*, 315(5814): 972–976, 2007. [p1]

F. E. Harrell Jr and M. F. E. Harrell Jr. Package 'hmisc'. 2015. [p5]

J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979. [p1]

C. Hennig. Package 'fpc'. 2015. [p1]

C. W. Hu, S. M. Kornblau, J. H. Slater, and A. A. Qutub. Progeny clustering: A method to identify biological phenotypes. *Scientific reports*, 5, 2015. [p1, 2, 3, 5]

S. C. Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967. [p1]

M. Maechler, P. Rousseeuw, A. Struyf, M. Hubert, K. Hornik, M. Studer, and P. Roudier. Package 'cluster', 2015. [p1]

N. Meinshausen and P. Bühlmann. Stability selection. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(4):417–473, 2010. [p1]

S. Monti, P. Tamayo, J. Mesirov, and T. Golub. Consensus clustering: a resampling-based method for class discovery and visualization of gene expression microarray data. *Machine Learning*, 52(1-2): 91–118, 2003. [p1]

S. D. Ross. Segmenting sport fans using brand associations: A cluster analysis. *Sport Marketing Quarterly*, 16(1):15, 2007. [p1]

P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987. [p1]

J. Slater, J. C. Culver, B. L. Long, C. W. Hu, J. Hu, T. F. Birk, A. A. Qutub, M. E. Dickinson, and J. L. West. Recapitulation and modulation of the cellular architecture of a user-chosen cell-of-interest using cell-derived, biomimetic patterning. *ACS nano*, 2015. [p5]

J. H. Slater, J. S. Miller, S. S. Yu, and J. L. West. Fabrication of multifaceted micropatterned surfaces with laser scanning lithography. *Advanced Functional Materials*, 21(15):2876–2888, 2011. [p8]

T. Sørlie, C. M. Perou, R. Tibshirani, T. Aas, S. Geisler, H. Johnsen, T. Hastie, M. B. Eisen, M. van de Rijn, S. S. Jeffrey, et al. Gene expression patterns of breast carcinomas distinguish tumor subclasses with clinical implications. *Proceedings of the National Academy of Sciences*, 98(19):10869–10874, 2001. [p1]

R. Tibshirani, G. Walther, and T. Hastie. Estimating the number of clusters in a data set via the gap statistic. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(2):411–423, 2001. [p1, 3]

M. Walesiak, A. Dudek, and M. A. Dudek. Package 'clustersim', 2015. [p1]

Chenyue W. Hu
Rice University
Suite 610, BioScience Research Collaborative, 6500 Main St, Houston, TX 77030
U.S.A
wendyhu001@gmail.com

Amina A. Qutub
Rice University
Suite 610, BioScience Research Collaborative, 6500 Main St, Houston, TX 77030
U.S.A
aminaq@gmail.com