GSSTDA: Implementation in an R Package of the Progression of Disease with Survival Analysis (PAD-S) that Integrates Information on Genes Linked to Survival in the Mapper Filter Function

by Miriam Esteve, Raquel Bosch-Romeu, Antonio Falco, Jaume Fores, and Joan Climent

Abstract GSSTDA is a new package for R that implements a new analysis for trascriptomic data, the Progression Analysis of Disease with Survival (PAD-S) by Fores-Martos et al. (2022), which allows to identify groups of samples differentiated by both survival and idiosyncratic biological features. Although it was designed for transcriptomic analysis, it can be used with other types of continuous omics data. The package implements the main algorithms associated with this methodology, which first removes the part of expression that is considered physiological using the Disease-Specific Genomic Analysis (DSGA) and then analyzes it using an unsupervised classification scheme based on Topological Data Analysis (TDA), the Mapper algorithm. The implementation includes code to perform the different steps of this analysis: data preprocessing by DSGA, the selection of genes for further analysis and a new filter function, which integrates information about genes related to survival, and the Mapper algorithm for generating a topological invariant Reeb graph. These functions can be used independently, although a function that performs the entire analysis is provided. This paper describes the methodology and implementation of these functions, and reports numerical results using an extract of real data base application.

1 Introduction

This paper presents the implementation of Progression Analysis of Disease with Survival (PAD-S) (Fores-Martos et al. (2022)) a new analysis, based on Progression Analysis of Disease (PAD), whose main novelty is that it integrates information on genes linked to survival. The PAD, that was developed by Nicolau et al. (2011), allows a set of transcriptomic data samples to be summarised in a combinatorial graph whose nodes are subsets of the samples. In this analysis, data pre-processed using the Disease-Specific Genomic Analyses (DSGA) is subjected to the Mapper algorithm. Although both DSGA and PAD were initially used with microarray data, they can be used on other types of continuous omics data. This extends to PAD-S as well.

This DSGA is a mathematical analysis for trascriptomic data that isolates the component of data relevant to disease by defining a transformation that measures the extent to which diseased tissue samples deviates from healthy tissue samples (Nicolau et al. (2007)). On the other hand, Mapper is a Topological Data Analysis tool (Carlsson (2009)). TDA is intended to find the topological structure of the data using tools for algebraic topology and computational geometry. Two of its approaches is persistent homology (Edelsbrunner and Harer (2022)) and Mapper (Singh et al. (2007); Lum et al. (2013)). Persistent homology adopts concepts from abstract algebra in order to extract characteristics in the data, such as the presence of higher-dimensional holes and the number of connected components. Mapper however was designed as a visualization method although it can also be considered as an unsupervised classification methodology. It allows to simplify and visualize the information of high dimensional data sets into a combinatorial or complex simplicial graph, which is called the dataset skeleton. To do so, it employs preassigned guiding functions called filters. Mapper has been applied in biological and biomedical research. It has been used successfully to study aortic stenosis, where through the construction of Mapper graph is able to identify disease subtypes with a higher resolution than standard approaches (Casaclang-Verzosa et al. (2019)). Another application in the clinic has been its use in asthma, where clinical examination and biochemical and cellular parameters in biological samples are introduced as input for Mapper in order to identify subgroup of healthy patients and subgroups of asthmatic patients (Hinks et al. (2015)). In addition, it has also been used in other cases to analyze omics data, such as single-cell RNA sequencing data to study cell differentiation and development (Rizvi et al. (2017)) or to study breast cancer using RNA sequencing data (Mathews et al. (2019)).

The PAD, using DSGA and Mapper and developing an own filter function and a way to select the most relevant genes for Mapper, puzzles out the geometry characteristics of the data that are obscured when using cluster analysis and deliver a simple representation of the trascriptomic data set. Our new analysis, the PAD-S, integrates in its new filter function information about the relationship between gene expression level and survival and also uses this information to select the genes to be used in Mapper. This adapted filter function aims to capture the expected survival associated with each patient from the information obtained from the survival analyses.

Mapper performs clustering by strata of different value ranges of this filter function, an then, it is determined if there is a relationship between clusters of different strata, obtaining a topological invariant Reeb graph. As PAD-S filtering function captures the expected survival of each individual, we obtain a graph that captures the shape of the data along the values of the "survival". In this graph, in addition to possible clusters of interest, it is possible to study structures that appear in the graph, such as branches or loops, that can uncover underlying biological patterns. This could show new relevant groups of patients with good or bad prognosis and specific biological characteristics not detected by other types of analysis.

In PAD-S, this ability of Mapper is strengthened by DSGA preprocessing, as it facilitates only the portion of gene expression that is not present in healthy tissue to be used in clustering, thus streamlining data processing and enhancing the extraction of biologically relevant information. In turn, gene selection allows the results to meet the proposed objectives in the most efficient way by selecting the most relevant survival-related genes or that have greater variability. This could lead to the identification of potential biomarkers.

In the landscape of omics data analysis methods, PAD-S stands out for its unique approach rooted in TDA. Mapper can be defined as an atypical unsupervised classification method and, like PCA and traditional clustering, it is not necessary to have prior knowledge of the data and the possible subgroups that form it. However, the Mapper's approach provides a holistic view of the data, allowing for the identification of data subsets, non-linear relationships and intricate patterns reflected in the structures of the graph that may be missed by these methods.

Many methods for analysis of omics data are nowadays available in Bioconductor, a collection of almost 1000 packages for the analysis and comprehension of high-throughput biological data, in R packages such as: **rtracklayer** (Lawrence et al., 2009), for interacting with multiple genome browsers (UCSC) and manipulating annotation tracks in various formats (GFF, BED, bedGraph, BED15, WIG, BigWig and 2bit); **Rsubread** (Liao et al., 2019), for read mapping, read counting, SNP calling, structural variant detection and gene fusion discovery; or **survcomp** (Schroder et al., 2011), for performance assessment and comparison of survival models; **scMappR** (Sokolowski et al., 2021), for providing experimentally relevant cell-type specific information to a list of differentially expressed genes (DEG). Regarding the availability of the R implementation of the analyses used in this study, in GitHub, we can find implementations of the Mapper algorithm, **Mapper** [`https://github.com/peekxc/Mapper`] or **TDAMapper** [`https://github.com/paultpearson/TDAmapper`]. No implementations found in CRAN repository. In relation to the DSGA, the implementation provided by the developers is no longer available.

In this paper, we introduce a new package in R, called **GSSTDA** which is the implementation of the already presented Progression Analysis of Disease with Survival (PAD-S). In particular, **GSSTDA** includes a complete set of baseline functions, covering DSGA method and Mapper tool applying the filter function with information on survival-related genes. We make PAD-S available as a R package, with options for DSGA only, filtering function with selection of genes, Mapper only, or a combination of the three. **GSSTDA** is available as free software, under the GNU General Public License version 3, and can be downloaded from the Comprehensive R Archive Network (CRAN) at `https://cran.r-project.org/web/packages/GSSTDA`, including supplementary material as data sets or vignettes to replicate all the results presented in this paper. In addition, **GSSTDA** is hosted on an open source repository on GitHub at `https://github.com/MiriamEsteve/GSSTDA`.

The functions included in the **GSSTDA** package are summarized in 1. This table comprises two columns divided into four subsections: three of them correspond to the three stages of the Gene Structure Survival using Topological Data Analysis and one to the full analysis. The first column is the name of the functions and the second one is the description of the functions.

The paper is organized as follows. The following subsection Background summaries the three parts of the methodology in the **GSSTDA** package: DSGA, gene selection and mapper process. Section Data structure describes the structure of the functions, the results and briefly explains which data are used to illustrate the package. Section Basic functions of the package presents the basic methods explained in Background. Finally, Section Conclusion concludes.

Table 1: GSSTDA package functions

Function	Description
‘dsga‘	For ‘dsga object‘. It allows the calculation of the ‘disease component‘ of a expression matrix which consists of, through linear models, eliminating the part of the data that is considered normal or healthy and keeping only the component that is due to the disease. It is intended to precede other techniques like classification or clustering. For more information see @Nicolau2007.
‘results_dsga‘	For ‘dsga object‘. It calculates the 100 genes with the highest variability in the matrix disease component between samples and use them to draw the heat map.
‘gene_selection‘	For ‘gene_selection object‘. It fittings a Cox proportional hazard model to each gene, then it makes a selection of genes according to both: their variability within the database and their relationship with survival. Subsequently, with the genes selected, it calculates the values of the filtering functions for each patient. The filter function allows to summarise each vector of each individual in a single data. This function takes into account the survival associated with each gene. In particular, the implemented filter function performs the vector magnitude in the Lp-norm (as well as k powers of this magnitude) of the vector resulting of weighting each element of the column vector by the Z score obtained in the cox proportional hazard model.
‘mapper‘	For ‘mapper object‘. It condenses the information of high-dimensional data sets into a combinatory graph or simplicial complex that is referred to as the skeleton of the data set. This implementation is the mapper of one dimension, i.e. using only one filter function value.
‘plot_mapper‘	For ‘mapper object‘. It produces an interactive network plot using visNetwork function from the mapper results.
‘gsstda‘	For ‘gsstda object‘. It integrate the three parts of the process: the preprocessing of the data [dsga process], the gene selection and the filter function [gene selection process], and the mapper algorithm [mapper process].

2 Background

2.1 DSGA: Disease-Specific Genomic Analysis

In PAD-S analysis the first step is use the DSGA method to transform the original data. DSGA was first developed for transcriptome array data although it can be used for other continuous omics data such as methylation data. It is intended to precede other techniques like clustering. DSGA methods obtains the amount of deviation that is present in a diseased sample compared to healthy control tissues, by isolating and separating the disease component (D_c) from the normal-component (N_c) portion of the data. This method involves three steps: (i) Flat construction (ii) Healthy State Model (HSM) construction, and (iii) Disease component computation for the original data set.

Flat constuction

The flat construction reduces or smooth characteristics of the data that are idiosyncratic to each normal or healthy tissue sample. The data used in this step is filtering with the healthy or normal tissue samples and is organized by columns. This matrix is denote as N. The R normal tissue gene expression vectors, denoted as $N_1, N_2, ..., N_R$, are computed as flattened vectors of the form $\hat{N}_1, \hat{N}_2, ..., \hat{N}_R$. The flattened vector $\hat{N}_i$ is calculated by fitting a 0-intercept least-squares linear model from all other N_i-normal tissue vectors $N_1, N_2, ..., N_{i-1}, N_{i+1}, ..., N_R$ as predictor variables, that is,

$$\hat{N}_i = \sum_{\substack{j=1 \\ j \neq i}}^{R} \beta_j N_j.$$

In this equation, $\hat{N}_i$ is the i-th flat vector and β_j is the coefficient associated with normal tissue sample N_j. Then, after carrying out this step separately for each healthy tissue sample, we obtain the flat matrix in which each column represents a flatted healthy tissue sample as:

$$\hat{N} = \left[\hat{N}_1, \hat{N}_2, ..., \hat{N}_R\right].$$

This flat data matrix $\hat{N}$ could be expressed as the sum of a signal matrix $\hat{N}_{true}$ and a noise matrix. In the next step, a singular value decomposition will be applied to denoise it. The estimated matrix $\hat{N}_{true}$ is the so-called Healthy State Model.

Healthy State Model (HSM) generation

Healthy State Model (HSM) computation describes, predicts, and quantitatively captures the functioning of health systems. In this step, the singular value decomposition (SVD) of the flat data matrix $\hat{N}$ is performed. In the implementation presented in this article, a different method developed by Gavish and Donoho (2014) is used for the selection of the number of singular values than in the original DSGA, which allows the process to be automated.

In Gavish and Donoho (2014), it is assumed that the number of rows in the matrix to be decomposed must be smaller than the number of columns. This is not true for the flat data matrix $\hat{N}$ obtained in the previous step, an ℓ-by-R matrix, as we assume a larger number of genes (ℓ) than healthy tissue samples (R). For this reason, in this section we work with the transpose matrix $\hat{N}^T$, an R-by-ℓ matrix, and so $R < \ell$.

Following this method, the matrix $\hat{N}^T$ can be expressed as the sum of a signal matrix and a noise matrix:

$$\hat{N}^T = \hat{N}^T_{true} + \gamma \hat{N}^T_{noise},$$

where $\hat{N}^T$ is the transpose flat matrix; $\hat{N}^T_{true}$ is an underlying low-rank matrix that contains the true signal; $\hat{N}^T_{noise}$ is a noise matrix in which entries are assumed to be a sample of i.i.d. random variables extracted from a Gaussian distribution, with zero mean and unit variance; and γ is a parameter that indicates the magnitude of the noise.

In the DSGA, the estimation of $\hat{N}_{true}$ corresponds to the previously introduced HSM. Our aim is to estimate it using a truncated SVD of $\hat{N}^T$ which is the default way of estimating it.

The singular value decomposition of the transpose of the flat matrix is denoted as:

$$\hat{N}^T = UDV^T,$$

where D is a diagonal matrix in which the elements of the diagonal are the singular values of $\hat{N}^T$

$\sigma_1, \sigma_2, ...\sigma_R$, with R being the number of rows of $\hat{N}^T$, ordered from high to low; and U and V^T are matrices containing the left and right singular vectors in columns and rows, respectively.

As mentioned above, $\hat{N}^T_{true}$ can be estimated by truncated SVD. One option to carry out this truncated SVD is to determine a hard threshold from which singular values in D are selected. This threshold depends on γ:

$$\hat{N}^T_{true} \simeq HSM^T = UD_\gamma V^T.$$

For rectangular matrices with known γ the optimal hard threshold for singular value selection is:

$$\tau = \lambda(\beta)\sqrt{\ell}\gamma,$$

where:

- ℓ is the number of columns of the matrix $\hat{N}^T$.
- β is the aspect ratio of our input matrix $\hat{N}^T$, that is, $\beta = \frac{R}{\ell}$, with R and ℓ being the number of rows and colums respectively.
- $\lambda(\beta)$ is obtained through the following expression:

$$\lambda(\beta) = \left(2(\beta+1) + \frac{8\beta}{(\beta+1)+(\beta^2+14\beta+1)^{1/2}}\right)^{1/2}.$$

On the other hand, for rectangular matrices with unknown γ the optimal hard threshold for singular value selection is:

$$\tau = \omega(\beta)\sigma_{med},$$

where σ_{med} is the median of the values $\{\sigma_1, \sigma_2, ...\sigma_R\}$ and $\omega(\beta)$ is defined by:

$$\omega(\beta) = \frac{\lambda(\beta)}{\mu_\beta}.$$

$\lambda(\beta)$ has already been defined above. μ_β is found by computing numerically the upper bound of the following definite integral in the range $[(1-\beta)^2, (1+\beta)^2]$:

$$\int_{(1-\beta)^2}^{\mu_\beta} \frac{\left[\left((1+\sqrt{\beta})^2 - t\right)\left(t-(1-\sqrt{\beta})^2\right)\right]^{1/2}}{2\pi t\beta} dt = \frac{1}{2}.$$

Only those singular values larger than the threshold obtained by the respective method will be kept. It is important to note that γ is not known or predetermined in standard analyses. Nevertheless, our package is intentionally designed to enable users to assess the impact of varying γ values, guided by external insights or theoretical assumptions about noise levels.

As already introduced, the estimation of $\hat{N}_{true}$, the new de-noised matrix, is the Healthy State Model (HSM). To construct it assume we need a diagonal matrix termed $D_\gamma = \text{diag}(\sigma_1^*, \ldots, \sigma_R^*)$ which is obtained from $D = \text{diag}(\sigma_1, \ldots, \sigma_R)$ by using that equation:

$$\sigma_i^* = \begin{cases} \sigma_i & \text{if } \sigma_i > \tau, \\ 0 & \text{otherwise.} \end{cases}$$

So, the `HSM` is defined as:

$$\hat{N}^T_{true} \simeq UD_\gamma V^T,$$
$$HSM \simeq \hat{N}_{true}.$$

Disease component

The computation of the disease vectors for each sample in the original data set is applied. The original matrix including healthy and disease samples is expressed as O. Then, for each O_i data vector present in O, a zero-intercept linear model is fitted to the columns of the HSM matrix, i.e,

$$O_i = \sum_k \eta_k^{(i)} HSM_k + \varepsilon_i$$

where $\eta_k^{(i)}$ are the coefficients obtained from the best approximation of the column space of the matrix HSM to O_i and ε_i is its corresponding residual. The estimation of $\eta_k^{(i)}$ aims to minimize the error between disease vectors and the projection of the observed disease vectors onto the HSM. Least squares regression is used in our package but other options such as ridge regression could also be employed adapting the estimation method based on data complexity and considerations like multicollinearity and regularization needs.

Now, considering the matrix

$$O_{DSGA} = [\varepsilon_1 \cdots \varepsilon_p],$$

we remove the portion of each sample that best mimics the expression patterns of healthy tissues and remain with the vector of residuals that carry out the information about how much each gene of a particular sample deviates from the values observed in healthy tissues.

2.2 Selection of the genes

Once the disease component matrix O_{DSGA} has been constructed, the next step consists of the selection of genes for downstream analysis. The gene selection procedure is based on the variability of the expression of each gene across the O_{DSGA} matrix and the degree of association of each gene with either disease-free or overall survival.

Associations between the levels of the expression of each gene with survival are computed employing univariate Cox proportional hazard models using the vectors corresponding to the pathological tissue samples of the original expression matrix. The Z-scores derived from the Cox proportional hazard models fits representing the degree of association of each gene with survival are then stored in the vector Z_{cox}. While it is often essential to check compliance with the proportional hazards assumption, we believe it is not for the PAD-S analysis, as the Z scores are merely used as weights.

The standard deviation of each gene in the pathological tissue samples is then calculated using the O_{DSGA} matrix. The vector of standard deviations is then stored as O_{sd}. To avoid values between 1 and -1, $+1$ is added to all positive values in vectors Z_{cox} and O_{sd}, and -1 is added to all negative values of vector Z_{cox}. The element-wise product between Z_{cox} and O_{sd} matrices is computed. Two methods of gene selection are proposed. In the first of them, the top and bottom n genes of the distribution of this product are selected for further analysis. In the other option, the n genes with the highest absolute value of this product are chosen.

Then, the O_{DSGA} matrix is filtered keeping the selected genes. This matrix constitute the input for the Mapper algorithm.

2.3 Mapper

Mapper (Singh et al. (2007), Lum et al. (2013)) is a tool derived from TDA that allows to condense high-dimensional data sets into a combinatorial graph capturing the shape of the data. Some of its properties is that it is insensitivity to metric, noise robustness and it allows multiscale representations (Carlsson (2009)).

This section explains (i) the PAD-S filter function that needs to be applied before Mapper, (ii) the the One Dimensional Mapper Algorithm itself and (iii) the different options for selecting the optimal number of clusters that can be used in the cluster step.

Filter function in PAD-S

To use the Mapper, a filter function denoted as f is required. This function summarizes each point in our data set denoted by X to $\mathbb{R}$.

$$f : X \rightarrow \mathbb{R}.$$

The selection of this filter function is of particular relevance and must be adapted to the nature of the problem under study. While more than one filter function can be used, resulting in a multi-dimensional mapper, the PAD-S methodology employs a single filter function to focus analysis on survival-associated gene expressions, simplifying the computational model, enhancing interpretability for clinical decision-making, and ensuring robustness through theoretical and empirical validation. This choice ensures a targeted analysis that is directly relevant to the biological and clinical questions at hand. The mapper developers suggest using density estimators, eccentricity, or graph Laplacians as filter functions (Singh et al. (2007)), although any function deemed appropriate can be used. In PAD analysis the filter function of choice was the vector of magnitude in the L^p norm, as well as k powers of this magnitude:

$$f(O^i_{DSGA}) = f(O^i_{DSGA}; p, k) = \left[\sum_r |g_r^{(i)}|^p \right]^{k/p}.$$

where O^i_{DSGA} denotes the i-th column vector of our disease component matrix (corresponding to individual i), which contains the values of each selected gene in that sample with coordinates $g_r^{(i)}$. Note that if $k = 1$ and $p = 2$, the function simply computes the standard (Euclidean) vector magnitude of each column.

In PAD-S analysis the filter function takes account the magnitude of the association between the expression level of a particular gene and survival. In particular the filter function is defined as follows:

$$f(O^i_{DSGA}; p, k) = \left[\sum_r |z_r \cdot g_r^{(i)}|^p\right]^{k/p},$$

where z_r is the z-value derived from the Cox proportional hazard models analysis and $g_r^{(i)}$ the i-th disease component value (corresponding to individual i) for r-th feature.

Note that in this case, to avoid values between 1 and -1, $+1$ is added to all positive values in vectors Z_{cox} and O^i_{DSGA}, and -1 is added to all negative values of both vectors. Therefore, the amount of deviation of each gene to the HSM is multiplied by the degree of association of this particular gene with survival.

The One Dimensional Mapper Algorithm

In PAD-S methodology, the Mapper is subsequently applied on O_{DSGA} using its filter function defined in the previous section Filter function in PAD-S.

In addition, Mapper requires the use of a specific distance metric and clustering type for the clustering step. To this end, the **GSSTDA** package implementation allows choosing between correlation and Euclidean distances as distance metrics and among single linkage, average linkage, complete linkage or k-medioids as clustering methods.

Once the values of the filter function have been calculated, the range of filter function values is divided into overlapping intervals. Subsequently, for each interval, the clustering of the individuals that have a value of the filter function that is within that interval is performed. After clustering each interval, the graph is constructed. Each cluster is a node and those that share at least one individual are joined by an edge. This is possible because the intervals are overlapping.

The output graph $G = G(V, E)$ is then defined putting each cluster as a node (or vertex) of the nodes that share samples are connected with an edge.

Methods for the identification of the optimal number of clusters

Two different options for the selection of the optimal number of clusters are offered in the package presented. One of them is the method originally used in Mapper, which is also the one used by the PAD analysis. It selects the number of clusters by constructing a histogram of the cluster edge lengths using k bins. An empty interval is usually generated in this histogram. The edge length of the start of this interval is chosen. The clusters with a greater edge length than this one are chosen. In addition to this method, the option of using the Silhouettes (Rousseeuw (1987)) method is also available.

In this method, the first step is as follows. For a particular data point x included in cluster C_i we first define $a(x)$ as the average distance of data point x to all other points y in the cluster. Thus,

$$a(x) = \frac{1}{|C_i| - 1} \sum_{\substack{y \in C_i \\ j \neq i}} d(x, y).$$

Then, $b(x)$ is defined as the minimum mean difference of point x to any other cluster C of which x is not a member, as the average distance from x to all points y in C where $C \neq C_i$.

$$b(x) = \min_{\substack{1 \le k \le \ell \\ k \neq i}} \frac{1}{|C_k|} \sum_{y \in C_k} d(x, y).$$

The cluster which has the minimum average distance to point x is said to be the neighbor cluster of x.

In the context of our analysis, the function $d(x, y)$ represents the distance measure used to quantify the similarity between data points x and y in our dataset. This distance metric is critical for clustering and analyzing the structure of the data. Specifically, $d(x, y)$ could be any metric that suits the nature of the data and the specific requirements of the analysis, such as Euclidean, Manhattan, or cosine similarity. In this manuscript, unless otherwise specified, we use the Euclidean distance, which is defined as

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2},$$

where x_i and y_i are the components of vectors x and y respectively. This choice is motivated by its geometric interpretability and computational efficiency in handling numerical data typical in omics studies.

The concept of Silhouette for point x is defined by

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}}, \quad \text{if} \quad |C_i > 1|,$$

that is, if the number of elements in the cluster is larger than 1. When a particular cluster C_i contains only a single object, it is unclear how $a(x)$ should be defined, and therefore we set the silhouette to 0.

$$s(x) = 0, \quad \text{if} \quad |C_i| = 1.$$

The average of the $s(x)$ values of all points grouped in a particular cluster C_i, i.e.

$$\bar{s}_i = \frac{1}{|C_i|}\sum_{x \in C_i} s(x)$$

indicates how well grouped are the members of this cluster, whereas the average of all data points

$$\bar{s} = \frac{1}{|\cup_i C_i|}\sum_{x \in \cup_i C_i} s(x)$$

indicates how well the available data points have been clustered in general.

To select the optimum number of clusters within each interval of the filter function, the average silhouette values $\bar{s}$ are computed for all possible partitions from 2 to $n-1$, where n is the number of samples within a specific interval. Then the n that produces the highest value of $\bar{s}$ and that exceeds a specific threshold is selected as the optimum number of clusters. The threshold of 0.25 for $\bar{s}$ has been chosen based on standard practice, recognizing it as a moderate value that reflects adequate separation and cohesion within clusters, which is crucial for ensuring both statistical significance and biological or clinical relevance of the clusters. If a different threshold is considered, it should be adjusted based on the study's specific objectives and the dataset's characteristics, with higher thresholds used for stronger delineation between clusters and lower thresholds suitable for exploratory analyses or overlapping data categories. To implement an alternative threshold, one should: 1) Analyze the distribution of silhouette scores for various thresholds to understand the impact on cluster structures, 2) Validate the stability and validity of these clusters using additional internal metrics, 3) Consult with domain experts to align the threshold with biological or clinical importance, and 4) Conduct a sensitivity analysis to ensure robustness of the results. If no partition produces an $\bar{s}$ exceeding the chosen threshold, all samples are then assigned to a unique cluster, facilitating the clear identification of distinct groupings within the data.

3 Data structure

Data are managed as a regular R `matrix` in the **GSSTDA** functions. The main functions of the GSSTDA package are `dsga()`, `gene_selection()`, `mapper()` and `gsstda()`, which return structured objects named `dsga_object`, `gene_selection_object`, `mapper_object` and `GSSTDA_object`, respectively. These objects contain fields with relevant information such as the genes selected for the mapper algorithm or the arguments introduced by the user in the function call.

The main fields of the all objects are the following:

- `full_data`: Matrix containing normalized gene expression data. The columns correspond to the patients and the rows to the genes.
- `survival_time`: Numerical vector of the same length as the number of columns of `full_data`. In addition, the patients must be in the same order as in `full_data`. For the patients whose sample is pathological should be indicated the time between the disease diagnosis and event (death, relapse or other). If the event has not occurred it should be indicated the time until the end of follow-up. Patients whose sample is from healthy tissue must have an `NA` value.
- `survival_event`: Numerical vector of the same length as the number of columns of `full_data`. Patients must be in the same order as in `full_data`. For the the patients with pathological sample should be indicated whether the event has occurred (1) or not (0). Only these values are

valid and healthy patients must have an NA value.
- case_tag: Character vector of the same length as the number of columns of full_data. Patients must be in the same order as in full_data. It must be indicated for each patient whether its sample is from pathological or healthy tissue. One value should be used to indicate whether the patient's sample is healthy and another value should be used to indicate whether the patient's sample is pathological. The user will then be asked which one indicates whether the patient is healthy. Only two values are valid in the vector in total.

3.1 Data set

```
# We load the data
data("full_data")
data("survival_time")
data("survival_event")
data("case_tag")
```

We illustrate all the functions presented in this paper resorting to a single data set (full_data) with its corresponding vectors (survival_time, survival_event and case_tag) available in the **GSSTDA** package. The original data are from the study GSE42568 available in https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE42568. This data was processed normalizing the genes expression. We carry out background correction, summarizing, and quantile normalization of data set using the fRMA method implemented in the **fRMA** package. Our data set consists of 20,825 genes and 121 patients. Compliant with CRAN publication requirements, which necessitate demonstrating our package's functionality on a manageable subset of data, we have strategically reduced the number of genes in our dataset to 4165. This ensures that our analysis remains computationally feasible while maintaining a representative sample. The examples were executed in the posit (RStudio Cloud) environment, which has a system configuration including 3.75 GB RAM, Intel Xeon E5-2673 v3 @ 2.40 GHz processor, and Ubuntu 20.04 operating system. The calculations were completed in 78 seconds, demonstrating the package's efficiency. The dataset thus includes gene expression profiling from 104 breast cancer and 17 normal breast biopsies, exemplifying the package's utility across typical clinical samples without compromising the integrity and representativeness of the results. The first four patients and their ten genes are shown below for the four data sets:

	GSM1045191	GSM1045192	GSM1045193	GSM1045194
A1BG	5.769	5.912	5.829	5.868
A1BG-AS1	5.340	5.417	4.986	5.059
A1CF	4.732	4.780	4.870	5.780
A2M	11.053	10.794	5.897	11.029
A2M-AS1	5.410	5.223	5.063	4.740
A2ML1	4.375	4.680	4.888	4.629
A2MP1	4.712	6.083	6.943	6.452
A4GALT	6.791	7.086	7.562	7.604
A4GNT	4.456	4.331	4.720	4.966
AA06	4.990	5.903	5.741	5.886

Vectors	Extract of the values
'survival_time'	NA NA NA 99.417 24.805 99.023 13.339 73.101 8.279 70.242 80.46
'survival_event'	NA NA NA 0 1 0 1 0 1 0 0
'case_tag'	NT NT NT T T T T T T T T

4 Basic functions of the package

In this section, we introduce the main functions of the library related to Gene Structure Survival using Topological Data Analysis.

4.1 DSGA object

The basic model of Disease-Specific Genomic Analysis that we explained in subsection DSGA: Disease-Specific Genomic Analysis can be implemented in R using the function dsga():

```
dsga_object <- dsga(full_data, survival_time, survival_event, case_tag, gamma)
```

The minimum arguments of this function are `full_data`, `survival_time`, `survival_event` and `case_tag`, which were explained in the previous section Data structure. The parameter `gamma` is optional. It indicates the magnitude of the noise assumed in the flat data matrix for the generation of the Healthy State Model. If it takes the value `NA` the magnitude of the noise is assumed to be unknown. We recommend using the default value `NA`. Additionally, the function requests the following information from the user at runtime:

- Enter "yes" if the columns are the patients and the row the genes or "no" in other cases. In the "no" case, the library will automatically change the columns and rows to fulfill this condition.
- Enter which is the tag of the patients whose sample is from healthy tissue that is stored in the `case_tag` vector. It is also possible to introduce it in the function as a parameter.

This function returns an object composed of:

- `normal_space`: The matrix with the normal space (linear space generated from normal tissue samples).
- `matrix_disease`: The disease component matrix that contains the disease component of all patients (`dsga_object[["matrix_disease_component"]]`).

As an example, using data and vectors from subsection Data structure, we next process the genes expression using the suitable code as follows. Results are returned as an `dsga object`, as explained in section Data structure.

```
dsga_object <- dsga(full_data = full_data,
                    survival_time = survival_time,
                    survival_event = survival_event,
                    case_tag = case_tag)

dsga_information <- results_dsga(
                  matrix_disease_component = dsga_object[["matrix_disease_component"]],
                    case_tag = case_tag)
```

The DSGA information are plotted using `results_dsga()` function (see Figure 1).

```
print(dsga_information)

#>   [1] "CPB1"      "AGR3"      "BMPR1B"    "ANKRD30A" "CP"        "CEACAM6"
#>   [7] "ADH1B"     "CXCL13"    "COL11A1"   "CHI3L1"    "CYP4Z1"    "AGR2"
#>  [13] "CD36"      "CLIC6"     "CYP4X1"    "C19orf33" "AREG"      "APOD"
#>  [19] "ADIPOQ"    "BEX1"      "AGTR1"     "CALML5"    "CLDN8"     "CYP4B1"
#>  [25] "CRISP3"    "DACH1"     "CXCL14"    "AFF3"      "CYP2B7P"   "CAPN8"
#>  [31] "COMP"      "CLSTN2"    "CCL19"     "CFB"       "CYP2T1P"   "CSTA"
#>  [37] "AZGP1"     "CRISPLD1" "CLGN"      "ANXA3"     "CGA"       "CHGB"
#>  [43] "CXCL9"     "AKR1C2"    "ACTG2"     "ALOX15B"   "AQP3"      "CLCA2"
#>  [49] "COL2A1"    "CXCL11"    "CA12"      "C15orf48" "CA2"       "CYP4Z2P"
#>  [55] "ASPN"      "AR"        "CHRDL1"    "AKR1C1"    "BBOX1"     "ABCA8"
#>  [61] "CYP4F8"    "CT83"      "CXCL10"    "ABAT"      "CEACAM5"   "ADAMTS15"
#>  [67] "ANLN"      "CLDN1"     "CPE"       "DCLK1"     "CELSR1"    "COL10A1"
#>  [73] "CFD"       "CNTNAP2"   "CLDN11"    "APOBEC3B" "ALDH3B2"   "C16orf54"
#>  [79] "CDO1"      "ANKRD30B" "COL14A1"   "ARHGAP36" "CECR2"     "BAMBI"
#>  [85] "CCND1"     "ADGRG6"    "AKR1C3"    "ABCC13"    "C16orf89" "CCL8"
#>  [91] "CNKSR3"    "DCD"       "CCL5"      "CEP55"     "CST6"      "ARNT2"
#>  [97] "BEX5"      "COL4A5"    "CLEC3A"    "ARMT1"
```

4.2 Gene selection object

The basic model of gene selection that we explained in subsection Selection of the genes can be implemented in R using the function `gene_selection()`. Furthermore, this function for convenience calculates the filter function values for mapper. It uses the filter function developed for the PAD-S:

```
gene_selection_object <- gene_selection(data, gen_select_type, percent_gen_select)
```

The fields of the `gene_selection()` are the following:

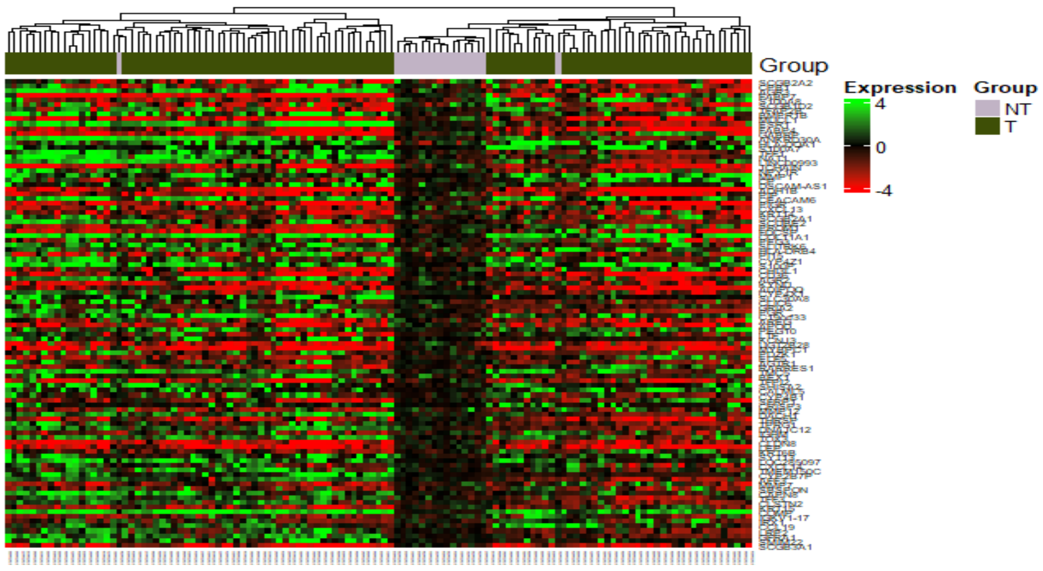

Figure 1: Heatmap of the disease component matrix (result of the 'dsga' function) after selecting the 100 genes with the highest variability between samples. Each column represents a patient sample and each row represents a gene. The color scale reflects the values of the disease component, with warmer colors indicating higher expression values than in healthy tissue and cooler colors indicating lower expression. Key: 'NT' stands for 'Non-Tumor' – representing normal breast biopsy samples; 'T' stands for 'Tumor' – representing breast cancer biopsy samples. In addition, the result of the hierarchical clustering of the samples using the euclidean distance and the complete method is included. This visualization aids in distinguishing between the disease component profiles of tumor and non-tumor samples, highlighting the difference between the two types after preprocessing with DSGA.

- `gen_select_type`: Options on how to select the genes to be used in the mapper:
  - `Abs`: The genes with the highest absolute value are chosen.
  - `Top_Bot`: Half of the selected genes are those with the highest value (positive value, i.e. worst survival prognosis) and the other half are those with the lowest value (negative value, i.e. best prognosis). "Top_Bot" default option.
- `percent_gen_select`: Percentage (from zero to one hundred) of genes to be selected to be used in mapper. 10 is the default option.
- `data`: The `data` argument could be two options:
  - `dsga_object`: This is use if `dsga()` function was previously executed.
  - `data_object`: Create a object `data_object` with the require information (see following R example).

This function returns a `gene_selection object` composed of:

- `cox_all_matrix`: A matrix with the results of the application of proportional hazard models: the regression coefficients, the odds ratios, the standard errors of each coefficient, the Z values and the p-values for each Z value)
- `genes_selected`: A vector with the name of the selected genes,
- `genes_disease_ component`: The matrix of disease components with only the rows of the selected genes.
- `filter_values`: The vector of the values of the filter function.

We introduce two distinct options for data input when utilizing our analytical package. These input methods are designed to accommodate different data structures and user preferences, ensuring flexibility and accessibility for a variety of research needs.

First option: dsga_object

The first option for data input involves using a predefined object type, `dsga_object`. This object is specifically tailored for users who have pre-processed their data using the Disease-Specific Genomic Analysis (DSGA) method. The `dsga_object` contains all necessary attributes and methods for subsequent analysis steps within our package, ensuring that users can seamlessly integrate their DSGA-processed data.

As an example, we next create a `gene_selection object` for the first option of data input:

```
dsga_object <- dsga(full_data = full_data,
                    survival_time = survival_time,
                    survival_event = survival_event,
                    case_tag = case_tag)
gene_selection_object <- gene_selection(data =  dsga_object,
                                      gen_select_type = "Top_Bot",
                                      percent_gen_select = 10)
```

Second option: data_object

The second option allows users to input their data as a generic data_object. This approach is intended for users who may have their data prepared in different formats or who require a more general input method that does not depend on the specific preprocessing steps like those required by the DSGA. The data_object should contain all the necessary data fields, such as gene expression data, survival data, and any other relevant clinical parameters, formatted in a way that can be directly utilized by our package.

The example of the second option of data input is:

```
# Create data object
data_object <- list(full_data = full_data,
                    survival_time = survival_time,
                    survival_event = survival_event,
                    case_tag = case_tag)
class(data_object) <- "data_object"

#Select gene from data object
gene_selection_object <- gene_selection(data = data_object,
                                      gen_select_type = "Top_Bot",
                                      percent_gen_select = 10)
```

Each of these input options is designed to provide the flexibility needed to handle diverse data types and preprocessing techniques. This ensures that our package can be effectively used in various scenarios, catering to both advanced users with specific preprocessing needs and those seeking more general data input methods.

4.3 Mapper object

The basic model of mapper that we explained in subsection Mapper can be implemented in R using the function `mapper()`:

```
mapper_object <- mapper(data,
                        filter_values,
                        num_intervals,
                        percent_overlap,
                        distance_type,
                        clustering_type,
                        num_bins_when_clustering,
                        linkage_type,
                        optimal_clustering_mode,
                        silhouette_threshold)
```

The fields of the mapper object are the following:

- `data`: Input matrix with which the analysis is performed. The user is asked whether the columns correspond to patients or features. In GSSTDA, this matrix corresponds to the matrix of disease components with only the rows of the selected genes. It can be any other matrix to which mapper is to be applied.

- filter_values: Vector obtained after applying the filtering function to the input matrix, i.e, a vector with the filtering function values for each included sample.
- num_intervals: Number of intervals used to create the first sample partition based on filtering values. "5" is the default option.
- percent_overlap: Percentage of overlap between intervals. Expressed as a percentage. "40" is the default option.
- distance_type: Type of distance to be used for clustering.
  - correlation: "correlation" is the default option.
  - euclidean
- clustering_type: Type of clustering method.
  - hierarchical: "hierarchical" is the default option.
  - PAM: "PAM" ("partition around medoids") option.
- num_bins_when_clustering: Number of bins to generate the histogram employed by the standard optimal number of cluster finder method. Parameter not necessary if the optimal_clustering_mode option is "silhouette" or the clustering_type is "PAM". "10" is the default option.
- linkage_type: Linkage criteria used in hierarchical clustering. Only necessary for hierarchical clustering.
  - single: Single-linkage clustering. "single" is the default option.
  - complete: Complete-linkage clustering.
  - average: Average linkage clustering (or UPGMA).
- optimal_clustering_mode: Method for selection optimal number of clusters. It is only necessary if the chosen type of algorithm is "hierarchical". In this case, choose between "standard" (the method used in the original mapper article) or "silhouette". In the case of the "PAM" algorithm, the method will always be "silhouette".
- silhouette_threshold: Minimum value of $\bar{s}$ that a set of clusters must have to be chosen as optimal. Within each interval of the filter function, the average silhouette values $\bar{s}$ are computed for all possible partitions from 2 to $n-1$, where n is the number of samples within a specific interval. The n that produces the highest value of $\bar{s}$ and that exceeds a specific threshold is selected as the optimum number of clusters. If no partition produces an $\bar{s}$ exceeding the chosen threshold, all samples are then assigned to a unique cluster. We recommend to use the default value of 0.25.

This function returns a mapper object which contains:

- interval_data: The values of the intervals.
- sample_in_level: The samples included in each interval.
- clustering_all_levels: The information about the cluster to which the individuals in each interval belong.
- node_samples: A list including the individuals contained in each detected node.
- node_sizes: Their size.
- node_average_filt: The average of the filter function values of the individuals of each node.
- adj_matrix: The adjacency matrix linking the nodes.

As an example, using results of dsga() and gene_selection(), we next process the mapper() using the suitable code as follows:

The information obtained from the mapper object are showed using print:

```
print(mapper_object)
```

```
#> $interval_data
#> $interval_data$Level_1
#> [1] 612.7152 627.1164
#>
#> $interval_data$Level_2
#> [1] 621.3959 635.6971
#>
#> $interval_data$Level_3
#> [1] 629.9766 644.2778
#>
#> $interval_data$Level_4
#> [1] 638.5573 652.8585
```

```
#>
#> $interval_data$Level_5
#> [1] 647.1381 661.4393
#>
#> $interval_data$Level_6
#> [1] 655.7188 670.0200
#>
#> $interval_data$Level_7
#> [1] 664.2995 678.6007
#>
#> $interval_data$Level_8
#> [1] 672.8802 687.1814
#>
#> $interval_data$Level_9
#> [1] 681.4609 695.7622
#>
#> $interval_data$Level_10
#> [1] 690.0417 704.4429
#>
#>
#> $sample_in_level
#> $sample_in_level$Level_1
#> [1] "GSM1045193" "GSM1045217" "GSM1045302"
#>
#> $sample_in_level$Level_2
#> [1] "GSM1045192" "GSM1045217" "GSM1045288"
#>
#> $sample_in_level$Level_3
#> [1] "GSM1045192" "GSM1045194" "GSM1045221"
#>
#> $sample_in_level$Level_4
#> [1] "GSM1045194" "GSM1045211" "GSM1045225" "GSM1045243" "GSM1045250"
#> [6] "GSM1045282"
#>
#> $sample_in_level$Level_5
#>  [1] "GSM1045211" "GSM1045213" "GSM1045225" "GSM1045242" "GSM1045243"
#>  [6] "GSM1045250" "GSM1045282" "GSM1045285" "GSM1045293" "GSM1045301"
#>
#> $sample_in_level$Level_6
#>  [1] "GSM1045203" "GSM1045206" "GSM1045209" "GSM1045212" "GSM1045213"
#>  [6] "GSM1045214" "GSM1045224" "GSM1045227" "GSM1045237" "GSM1045241"
#> [11] "GSM1045242" "GSM1045248" "GSM1045255" "GSM1045264" "GSM1045266"
#> [16] "GSM1045274" "GSM1045285" "GSM1045293" "GSM1045301"
#>
#> $sample_in_level$Level_7
#>  [1] "GSM1045200" "GSM1045201" "GSM1045203" "GSM1045206" "GSM1045209"
#>  [6] "GSM1045218" "GSM1045222" "GSM1045227" "GSM1045229" "GSM1045232"
#> [11] "GSM1045233" "GSM1045237" "GSM1045239" "GSM1045240" "GSM1045241"
#> [16] "GSM1045244" "GSM1045252" "GSM1045255" "GSM1045257" "GSM1045259"
#> [21] "GSM1045264" "GSM1045269" "GSM1045270" "GSM1045276" "GSM1045278"
#> [26] "GSM1045294" "GSM1045298" "GSM1045305" "GSM1045306"
#>
#> $sample_in_level$Level_8
#>  [1] "GSM1045191" "GSM1045195" "GSM1045196" "GSM1045197" "GSM1045198"
#>  [6] "GSM1045199" "GSM1045200" "GSM1045201" "GSM1045204" "GSM1045205"
#> [11] "GSM1045207" "GSM1045210" "GSM1045216" "GSM1045218" "GSM1045219"
#> [16] "GSM1045222" "GSM1045228" "GSM1045231" "GSM1045232" "GSM1045233"
#> [21] "GSM1045238" "GSM1045239" "GSM1045240" "GSM1045245" "GSM1045246"
#> [26] "GSM1045252" "GSM1045254" "GSM1045257" "GSM1045258" "GSM1045259"
#> [31] "GSM1045260" "GSM1045261" "GSM1045263" "GSM1045267" "GSM1045270"
#> [36] "GSM1045271" "GSM1045275" "GSM1045280" "GSM1045283" "GSM1045284"
#> [41] "GSM1045287" "GSM1045296" "GSM1045298" "GSM1045300" "GSM1045303"
#> [46] "GSM1045306" "GSM1045309"
#>
```

```
#> $sample_in_level$Level_9
#>  [1] "GSM1045191" "GSM1045195" "GSM1045196" "GSM1045197" "GSM1045198"
#>  [6] "GSM1045199" "GSM1045202" "GSM1045204" "GSM1045205" "GSM1045208"
#> [11] "GSM1045215" "GSM1045220" "GSM1045228" "GSM1045231" "GSM1045235"
#> [16] "GSM1045236" "GSM1045246" "GSM1045249" "GSM1045253" "GSM1045256"
#> [21] "GSM1045258" "GSM1045260" "GSM1045262" "GSM1045265" "GSM1045267"
#> [26] "GSM1045268" "GSM1045271" "GSM1045273" "GSM1045277" "GSM1045279"
#> [31] "GSM1045280" "GSM1045281" "GSM1045286" "GSM1045289" "GSM1045290"
#> [36] "GSM1045291" "GSM1045292" "GSM1045295" "GSM1045297" "GSM1045299"
#> [41] "GSM1045304" "GSM1045307" "GSM1045309" "GSM1045310"
#>
#> $sample_in_level$Level_10
#>  [1] "GSM1045202" "GSM1045215" "GSM1045220" "GSM1045223" "GSM1045226"
#>  [6] "GSM1045230" "GSM1045234" "GSM1045235" "GSM1045236" "GSM1045247"
#> [11] "GSM1045249" "GSM1045251" "GSM1045256" "GSM1045262" "GSM1045265"
#> [16] "GSM1045268" "GSM1045272" "GSM1045273" "GSM1045277" "GSM1045279"
#> [21] "GSM1045286" "GSM1045289" "GSM1045291" "GSM1045292" "GSM1045297"
#> [26] "GSM1045304" "GSM1045307" "GSM1045308" "GSM1045310" "GSM1045311"
#>
#>
#> $clustering_all_levels
#> $clustering_all_levels$Level_1
#> GSM1045193 GSM1045217 GSM1045302
#>          1          2          2
#>
#> $clustering_all_levels$Level_2
#> GSM1045192 GSM1045217 GSM1045288
#>          1          1          1
#>
#> $clustering_all_levels$Level_3
#> GSM1045192 GSM1045194 GSM1045221
#>          1          1          1
#>
#> $clustering_all_levels$Level_4
#> GSM1045194 GSM1045211 GSM1045225 GSM1045243 GSM1045250 GSM1045282
#>          1          1          1          1          1          1
#>
#> $clustering_all_levels$Level_5
#> GSM1045211 GSM1045213 GSM1045225 GSM1045242 GSM1045243 GSM1045250 GSM1045282
#>          1          1          1          1          1          1          1
#> GSM1045285 GSM1045293 GSM1045301
#>          1          1          1
#>
#> $clustering_all_levels$Level_6
#> GSM1045203 GSM1045206 GSM1045209 GSM1045212 GSM1045213 GSM1045214 GSM1045224
#>          1          1          2          3          4          5          6
#> GSM1045227 GSM1045237 GSM1045241 GSM1045242 GSM1045248 GSM1045255 GSM1045264
#>          7          4          2          7          3          2          3
#> GSM1045266 GSM1045274 GSM1045285 GSM1045293 GSM1045301
#>          5          8          4          3          9
#>
#> $clustering_all_levels$Level_7
#> GSM1045200 GSM1045201 GSM1045203 GSM1045206 GSM1045209 GSM1045218 GSM1045222
#>          1          1          1          1          1          1          1
#> GSM1045227 GSM1045229 GSM1045232 GSM1045233 GSM1045237 GSM1045239 GSM1045240
#>          1          1          1          1          1          1          1
#> GSM1045241 GSM1045244 GSM1045252 GSM1045255 GSM1045257 GSM1045259 GSM1045264
#>          1          1          1          1          1          1          1
#> GSM1045269 GSM1045270 GSM1045276 GSM1045278 GSM1045294 GSM1045298 GSM1045305
#>          1          1          1          1          1          1          1
#> GSM1045306
#>          1
#>
#> $clustering_all_levels$Level_8
#> GSM1045191 GSM1045195 GSM1045196 GSM1045197 GSM1045198 GSM1045199 GSM1045200
#>          1          1          1          1          1          1          1
#> GSM1045201 GSM1045204 GSM1045205 GSM1045207 GSM1045210 GSM1045216 GSM1045218
#>          1          1          1          1          1          1          1
#> GSM1045219 GSM1045222 GSM1045228 GSM1045231 GSM1045232 GSM1045233 GSM1045238
#>          1          1          1          1          1          1          1
#> GSM1045239 GSM1045240 GSM1045245 GSM1045246 GSM1045252 GSM1045254 GSM1045257
#>          1          1          1          1          1          1          1
#> GSM1045258 GSM1045259 GSM1045260 GSM1045261 GSM1045263 GSM1045267 GSM1045270
#>          1          1          1          1          1          1          1
#> GSM1045271 GSM1045275 GSM1045280 GSM1045283 GSM1045284 GSM1045287 GSM1045296
#>          1          1          1          1          1          1          1
#> GSM1045298 GSM1045300 GSM1045303 GSM1045306 GSM1045309
#>          1          1          1          1          1
#>
#> $clustering_all_levels$Level_9
#> GSM1045191 GSM1045195 GSM1045196 GSM1045197 GSM1045198 GSM1045199 GSM1045202
#>          1          1          1          1          1          1          1
#> GSM1045204 GSM1045205 GSM1045208 GSM1045215 GSM1045220 GSM1045228 GSM1045231
#>          1          1          1          1          1          1          1
#> GSM1045235 GSM1045236 GSM1045246 GSM1045249 GSM1045253 GSM1045256 GSM1045258
#>          1          1          1          1          1          1          1
#> GSM1045260 GSM1045262 GSM1045265 GSM1045267 GSM1045268 GSM1045271 GSM1045273
#>          1          1          1          1          1          1          1
#> GSM1045277 GSM1045279 GSM1045280 GSM1045281 GSM1045286 GSM1045289 GSM1045290
#>          1          1          1          1          1          1          1
#> GSM1045291 GSM1045292 GSM1045295 GSM1045297 GSM1045299 GSM1045304 GSM1045307
#>          1          1          1          1          1          1          1
#> GSM1045309 GSM1045310
#>          1          1
#>
#> $clustering_all_levels$Level_10
#> GSM1045202 GSM1045215 GSM1045220 GSM1045223 GSM1045226 GSM1045230 GSM1045234
#>          1          1          1          1          1          1          2
#> GSM1045235 GSM1045236 GSM1045247 GSM1045249 GSM1045251 GSM1045256 GSM1045262
#>          1          1          1          1          1          1          1
#> GSM1045265 GSM1045268 GSM1045272 GSM1045273 GSM1045277 GSM1045279 GSM1045286
#>          1          1          1          1          1          1          1
#> GSM1045289 GSM1045291 GSM1045292 GSM1045297 GSM1045304 GSM1045307 GSM1045308
#>          1          1          1          1          1          1          1
#> GSM1045310 GSM1045311
#>          1          1
#>
#>
#> $node_samples
#> $node_samples$Node_1
#> [1] "GSM1045193"
#>
#> $node_samples$Node_2
#> [1] "GSM1045217" "GSM1045302"
#>
#> $node_samples$Node_3
#> [1] "GSM1045192" "GSM1045217" "GSM1045288"
#>
#> $node_samples$Node_4
#> [1] "GSM1045192" "GSM1045194" "GSM1045221"
#>
#> $node_samples$Node_5
#> [1] "GSM1045194" "GSM1045211" "GSM1045225" "GSM1045243" "GSM1045250"
#> [6] "GSM1045282"
#>
#> $node_samples$Node_6
#>  [1] "GSM1045211" "GSM1045213" "GSM1045225" "GSM1045242" "GSM1045243"
#>  [6] "GSM1045250" "GSM1045282" "GSM1045285" "GSM1045293" "GSM1045301"
```

```
#>
#> $node_samples$Node_7
#> [1] "GSM1045203" "GSM1045206"
#>
#> $node_samples$Node_8
#> [1] "GSM1045209" "GSM1045241" "GSM1045255"
#>
#> $node_samples$Node_9
#> [1] "GSM1045212" "GSM1045248" "GSM1045264" "GSM1045293"
#>
#> $node_samples$Node_10
#> [1] "GSM1045213" "GSM1045237" "GSM1045285"
#>
#> $node_samples$Node_11
#> [1] "GSM1045214" "GSM1045266"
#>
#> $node_samples$Node_12
#> [1] "GSM1045224"
#>
#> $node_samples$Node_13
#> [1] "GSM1045227" "GSM1045242"
#>
#> $node_samples$Node_14
#> [1] "GSM1045274"
#>
#> $node_samples$Node_15
#> [1] "GSM1045301"
#>
#> $node_samples$Node_16
#>  [1] "GSM1045200" "GSM1045201" "GSM1045203" "GSM1045206" "GSM1045209"
#>  [6] "GSM1045218" "GSM1045222" "GSM1045227" "GSM1045229" "GSM1045232"
#> [11] "GSM1045233" "GSM1045237" "GSM1045239" "GSM1045240" "GSM1045241"
#> [16] "GSM1045244" "GSM1045252" "GSM1045255" "GSM1045257" "GSM1045259"
#> [21] "GSM1045264" "GSM1045269" "GSM1045270" "GSM1045276" "GSM1045278"
#> [26] "GSM1045294" "GSM1045298" "GSM1045305" "GSM1045306"
#>
#> $node_samples$Node_17
#>  [1] "GSM1045191" "GSM1045195" "GSM1045196" "GSM1045197" "GSM1045198"
#>  [6] "GSM1045199" "GSM1045200" "GSM1045201" "GSM1045204" "GSM1045205"
#> [11] "GSM1045207" "GSM1045210" "GSM1045216" "GSM1045218" "GSM1045219"
#> [16] "GSM1045222" "GSM1045228" "GSM1045231" "GSM1045232" "GSM1045233"
#> [21] "GSM1045238" "GSM1045239" "GSM1045240" "GSM1045245" "GSM1045246"
#> [26] "GSM1045252" "GSM1045254" "GSM1045257" "GSM1045258" "GSM1045259"
#> [31] "GSM1045260" "GSM1045261" "GSM1045263" "GSM1045267" "GSM1045270"
#> [36] "GSM1045271" "GSM1045275" "GSM1045280" "GSM1045283" "GSM1045284"
#> [41] "GSM1045287" "GSM1045296" "GSM1045298" "GSM1045300" "GSM1045303"
#> [46] "GSM1045306" "GSM1045309"
#>
#> $node_samples$Node_18
#>  [1] "GSM1045191" "GSM1045195" "GSM1045196" "GSM1045197" "GSM1045198"
#>  [6] "GSM1045199" "GSM1045202" "GSM1045204" "GSM1045205" "GSM1045208"
#> [11] "GSM1045215" "GSM1045220" "GSM1045228" "GSM1045231" "GSM1045235"
#> [16] "GSM1045236" "GSM1045246" "GSM1045249" "GSM1045253" "GSM1045256"
#> [21] "GSM1045258" "GSM1045260" "GSM1045262" "GSM1045265" "GSM1045267"
#> [26] "GSM1045268" "GSM1045271" "GSM1045273" "GSM1045277" "GSM1045279"
#> [31] "GSM1045280" "GSM1045281" "GSM1045286" "GSM1045289" "GSM1045290"
#> [36] "GSM1045291" "GSM1045292" "GSM1045295" "GSM1045297" "GSM1045299"
#> [41] "GSM1045304" "GSM1045307" "GSM1045309" "GSM1045310"
#>
#> $node_samples$Node_19
#>  [1] "GSM1045202" "GSM1045215" "GSM1045220" "GSM1045223" "GSM1045226"
#>  [6] "GSM1045230" "GSM1045235" "GSM1045236" "GSM1045247" "GSM1045249"
#> [11] "GSM1045251" "GSM1045256" "GSM1045262" "GSM1045265" "GSM1045268"
#> [16] "GSM1045272" "GSM1045273" "GSM1045277" "GSM1045279" "GSM1045286"
#> [21] "GSM1045289" "GSM1045291" "GSM1045292" "GSM1045297" "GSM1045304"
#> [26] "GSM1045307" "GSM1045308" "GSM1045310" "GSM1045311"
#>
#> $node_samples$Node_20
#> [1] "GSM1045234"
#>
#>
#> $node_sizes
#>  Node_1  Node_2  Node_3  Node_4  Node_5  Node_6  Node_7  Node_8  Node_9 Node_10
#>       1       2       3       3       6      10       2       3       4       3
#> Node_11 Node_12 Node_13 Node_14 Node_15 Node_16 Node_17 Node_18 Node_19 Node_20
#>       2       1       2       1       1      29      47      44      29       1
#>
#> $node_average_filt
#> $node_average_filt$Node_1
#> [1] 612.8152
#>
#> $node_average_filt$Node_2
#> [1] 619.2167
#>
#> $node_average_filt$Node_3
#> [1] 628.0636
#>
#> $node_average_filt$Node_4
#> [1] 637.1298
#>
#> $node_average_filt$Node_5
#> [1] 649.4243
#>
#> $node_average_filt$Node_6
#> [1] 654.6199
#>
#> $node_average_filt$Node_7
#> [1] 667.8046
#>
#> $node_average_filt$Node_8
#> [1] 669.0753
#>
#> $node_average_filt$Node_9
#> [1] 663.2821
#>
#> $node_average_filt$Node_10
#> [1] 662.3884
#>
#> $node_average_filt$Node_11
#> [1] 662.0506
#>
#> $node_average_filt$Node_12
#> [1] 662.4023
#>
#> $node_average_filt$Node_13
#> [1] 663.1383
#>
#> $node_average_filt$Node_14
#> [1] 663.8281
#>
#> $node_average_filt$Node_15
#> [1] 658.3845
#>
#> $node_average_filt$Node_16
#> [1] 672.296
#>
#> $node_average_filt$Node_17
#> [1] 680.2087
```

```
#>
#> $node_average_filt$Node_18
#> [1] 688.619
#>
#> $node_average_filt$Node_19
#> [1] 693.9702
#>
#> $node_average_filt$Node_20
#> [1] 697.2491
#>
#>
#> $adj_matrix
#>         Node_1 Node_2 Node_3 Node_4 Node_5 Node_6 Node_7 Node_8 Node_9 Node_10
#> Node_1       1      0      0      0      0      0      0      0      0       0
#> Node_2       0      1      1      0      0      0      0      0      0       0
#> Node_3       0      0      1      1      0      0      0      0      0       0
#> Node_4       0      0      0      1      1      0      0      0      0       0
#> Node_5       0      0      0      0      1      1      0      0      0       0
#> Node_6       0      0      0      0      0      1      0      0      1       1
#> Node_7       0      0      0      0      0      0      1      0      0       0
#> Node_8       0      0      0      0      0      0      0      1      0       0
#> Node_9       0      0      0      0      0      0      0      0      1       0
#> Node_10      0      0      0      0      0      0      0      0      0       1
#> Node_11      0      0      0      0      0      0      0      0      0       0
#> Node_12      0      0      0      0      0      0      0      0      0       0
#> Node_13      0      0      0      0      0      0      0      0      0       0
#> Node_14      0      0      0      0      0      0      0      0      0       0
#> Node_15      0      0      0      0      0      0      0      0      0       0
#> Node_16      0      0      0      0      0      0      0      0      0       0
#> Node_17      0      0      0      0      0      0      0      0      0       0
#> Node_18      0      0      0      0      0      0      0      0      0       0
#> Node_19      0      0      0      0      0      0      0      0      0       0
#> Node_20      0      0      0      0      0      0      0      0      0       0
#>         Node_11 Node_12 Node_13 Node_14 Node_15 Node_16 Node_17 Node_18 Node_19
#> Node_1        0       0       0       0       0       0       0       0       0
#> Node_2        0       0       0       0       0       0       0       0       0
#> Node_3        0       0       0       0       0       0       0       0       0
#> Node_4        0       0       0       0       0       0       0       0       0
#> Node_5        0       0       0       0       0       0       0       0       0
#> Node_6        0       0       1       0       1       0       0       0       0
#> Node_7        0       0       0       0       0       1       0       0       0
#> Node_8        0       0       0       0       0       1       0       0       0
#> Node_9        0       0       0       0       0       1       0       0       0
#> Node_10       0       0       0       0       0       1       0       0       0
#> Node_11       1       0       0       0       0       0       0       0       0
#> Node_12       0       1       0       0       0       0       0       0       0
#> Node_13       0       0       1       0       0       1       0       0       0
#> Node_14       0       0       0       1       0       0       0       0       0
#> Node_15       0       0       0       0       1       0       0       0       0
#> Node_16       0       0       0       0       0       1       1       0       0
#> Node_17       0       0       0       0       0       0       1       1       0
#> Node_18       0       0       0       0       0       0       0       1       1
#> Node_19       0       0       0       0       0       0       0       0       1
#> Node_20       0       0       0       0       0       0       0       0       0
#>         Node_20
#> Node_1        0
#> Node_2        0
#> Node_3        0
#> Node_4        0
#> Node_5        0
#> Node_6        0
#> Node_7        0
#> Node_8        0
#> Node_9        0
#> Node_10       0
#> Node_11       0
#> Node_12       0
#> Node_13       0
#> Node_14       0
#> Node_15       0
#> Node_16       0
#> Node_17       0
#> Node_18       0
#> Node_19       0
#> Node_20       1
#>
#> $n_sizes
#> [1] 20
#>
#> $average_nodes
#> [1] 9.7
#>
#> $standard_desviation_nodes
#> [1] 14.78655
#>
#> $number_connections
#> [1] 16
#>
#> $proportion_connections
#> [1] 0.08421053
#>
#> $number_ramifications
#> [1] 7
#>
#> attr(,"class")
#> [1] "mapper_object"
```

In addition, the mapper information are plotted using `plot_mapper()` function (see Figure 2). Hovering the mouse over each node in the interactive graph displays the number of samples that form the node.

To show more clearly the resizing options provided by *plot_mapper*(), the same graph is shown by varying the values of the function parameters (see Figure 3):

```
plot_mapper(mapper_object, trans_node_size = TRUE, exp_to_res = 1/1.5)
```

4.4 GSSTDA object

The analysis of `gsstda()` function consists of the three previous parts, i.e., a preprocessing of the data {`dsga()`], the gene selection and the filter function [`gene_selection()`], and the mapper algorithm [`mapper()`]. So, this function integrates the explained functions above. Results are returned as an `gsstda object`, which is composed by all the results of `dsga()`, `gene_selection()` and `mapper()` functions.

The basic model of GSSTDA (dsga + gene selection + mapper) can be implemented in R using the function `gsstda()`:

```
gsstda_object <- gsstda(full_data, survival_time, survival_event, case_tag,
                        gamma, gen_select_type, percent_gen_select,
                        num_intervals, percent_overlap, distance_type,
                        clustering_type, num_bins_when_clustering,
                        linkage_type, optimal_clustering_mode,
                        silhouette_threshold)
```

This function returns a `gsstda object` composed of:

- `normal_space`: The matrix with the normal space (linear space generated from normal tissue samples).

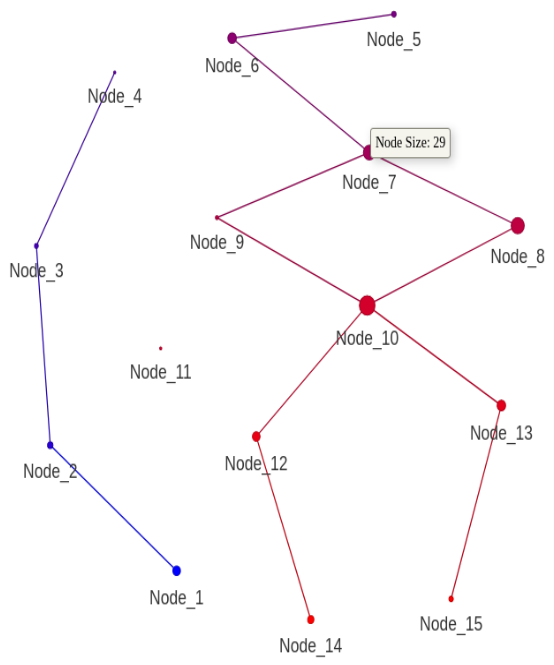

Figure 2: Mapper result graph. Mapper visualization depicting clusters of samples based on gene expression profiles along the value of the filter function, with nodes representing clusters and edges indicating overlap, facilitating the identification of nodes of different gene expression patterns and survival. The intensity of the color within each node reflects the average level of filter function within the cluster, aiding in the characterization of biological significance. Low filter function values, represented in blue, are associated with better survival. High values, in red, are associated with worse survival. Additionally, annotation allows for the identification of a cluster of interest, assisting in the extraction of insight into potential biomarkers or gene expression patterns associated with specific sample groups. The default option used in this case resizes the node sizes as $No.samples^{1/2}$.

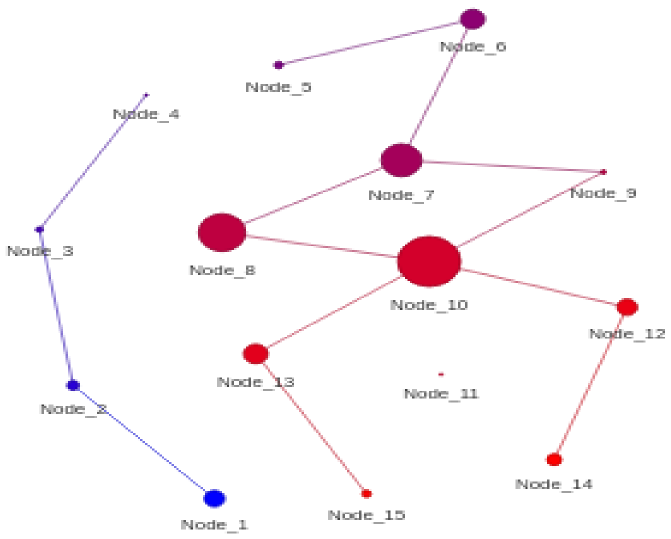

Figure 3: Mapper result graph after varying the parameter exp_to_res to 1/1.5 ($No.samples^{1/1.5}$). This way the resizing of the node size is smoother.

- `matrix_disease_components`: The matrix of the disease components (the transformed `full_data` matrix from which the normal component has been removed).
- `cox_all_matrix`: A matrix with the results of the application of proportional hazard models for each gene.
- `genes_selected`: The genes selected for `gene_selection()`.
- `genes_disease_component`: The matrix of the disease components with information from these genes only.
- `mapper_object`: The mapper object explained in the subsection Mapper object.

As an example, using arguments of `dsga()` and `gene_selection()`, we next process the `gsstda()` using the suitable code as follows:

```
gsstda_object <- gsstda(full_data = full_data,
                        survival_time = survival_time,
                        survival_event = survival_event,
                        case_tag = case_tag,
                        gen_select_type = "Top_Bot",
                        percent_gen_select = 10,
                        num_intervals = 10,
                        percent_overlap = 40,
                        distance_type = "correlation",
                        clustering_type = "hierarchical",
                        linkage_type = "single")
```

The information obtained from the GSSTDA object are showed using `print` for `dsga` and `mapper` information. In addition, the mapper information obtained by GSSTDA object are plotted using `plot_mapper()` function (see Figure 2).

5 Conclusions

The **GSSTDA** package allows transcriptomic data to be analysed (although it could be used for other types of omics data) integrating the information related on the degree of association of each gene with survival through the use of a specific feature selection procedure and a new adapted filter function. This allows to obtain a graph reflecting groups of samples that are not only differentiated by similarity in expression pattern but also by similarity in survival.

Specifically, this package is the implementation of the Progression Analysis of Disease with Survival. This methodology pre-processes the data using the DSGA which isolates the part of the expression that is considered pathological. This allows only this part to be used in further analysis. The genes to be used in the Mapper analysis are then selected on the basis of their association with survival. This information is also integrated into the filter function. Using the values of this filter function and the data processed through the DSGA, this information is condensed into a graph using the Mapper algorithm. One of the advantages of using Mapper over classical clustering techniques is that it has been shown in numerous applications to reveal aspects of the data that classical clustering does not detect.

In the broad field of oncology, the study of survival is essential, and omics sciences are increasingly important. GSSTDA allows the integration of both, so it could be useful in any oncological pathology and the results obtained can contribute to the development of personalized medicine. It can aid in the identification of prognostic biomarkers, therapeutic targets, or molecular subtypes. As has been done in this work with survival, any function of the data that reflects a clinical or biological characteristic of interest could be used as a filtering function so that the applications of possible adaptations of this package could be numerous.

Additionally, the combined use of our package with other analytical tools could show its versatility and potential for interdisciplinary research collaborations. Concrete examples or case studies demonstrating the application of the package in the analysis of real-world data sets and the discovery of novel biological insights can further illustrate its practical value and impact on biomedical research.

Several functions have been implemented in the **GSSTDA** package: for processing the data using DSGA method, selecting the genes and calculating the values of the new filter function taking into account the association of each gene with survival, and generating a topological invariant Reeb graph applying the Mapper algorithm. The package provides a function to perform the full PAD-S analysis but also allows the DSGA and Mapper analyses, which can have numerous applications, to be performed independently.

Throughout the paper, we have also shown how to organize the data, use the available functions, and interpret the results. In particular, to illustrate the different functions implemented in the package, we applied all of them on a common empirical example so that results can easily be compared. In this way, we believe that the **GSSTDA** package is a valid self-contained R package for grouping transcriptomics data samples according to their survival and gene expression by integrating information on genes linked to survival in the process of gene selection and in the Mapper filter function from the popular topological technique: Topological Data Analysis.

Diving into potential future directions for expanding this R package, an array of promising avenues beckon. Initially, existing functionalities could be improved by refining data preprocessing algorithms, optimizing parameter selection methods, and bolstering computational efficiency to accommodate larger datasets more adeptly. Another possible parallel option is to explore new alternatives to carry out these processes and adapt them to other types of omics data. To complement its applications, functions that allow exploring the results obtained could be integrated into the package.Exploring integrative strategies for analyzing multi-omics data stands out as another compelling trajectory, offering deeper insights into biological mechanisms and disease pathology. Adapting the method to other problems by modifying the feature selection process and the filtering function represents another possible direction for future work. Additionally, fostering community engagement and collaboration will be pivotal for nurturing ongoing growth and evolution, inviting users to contribute feedback, suggestions, and innovative applications to enrich the package's capabilities and impact. Through these concerted efforts, the aim is to equip researchers with a versatile and robust toolkit for exploring complex biological datasets and advancing biomedical research.

6 Acknowledgments

R. Bosch-Romeu, A. Falco and M. Esteve thank the grant TED2021-129347B-C22 funded by Ministerio de Ciencia e Innovación/ Agencia Estatal de Investigación. Additionally, R. Bosch-Romeu and A. Falco gratefully acknowledges the financial support from CEU Cardenal Herrera University and Santander under a predoctoral grant, from Scientific Foundation of the Spanish Association Against Cancer under a predoctoral grant, and the Spanish Ministry of Science, Innovation and Universities under grant *Ayudas para la Formación de Profesorado Universitario FPU 2022* (FPU 22/02810).

References

G. Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009. [p90, 95]

G. Casaclang-Verzosa, S. Shrestha, M. J. Khalil, J. S. Cho, M. Tokodi, S. Balla, M. Alkhouli, V. Badhwar, J. Narula, J. D. Miller, et al. Network tomography for understanding phenotypic presentations in aortic stenosis. *JACC: Cardiovascular Imaging*, 12(2):236–248, 2019. [p90]

H. Edelsbrunner and J. L. Harer. *Computational topology: an introduction*. American Mathematical Society, 2022. [p90]

J. Fores-Martos, B. Suay-Garcia, R. Bosch-Romeu, M. C. Sanfeliu-Alonso, A. Falco, and J. Climent. Progression analysis of disease with survival (pad-s) by survmap identifies different prognostic subgroups of breast cancer in a large combined set of transcriptomics and methylation studies. *bioRxiv*, pages 2022–09, 2022. [p90]

M. Gavish and D. L. Donoho. The optimal hard threshold for singular values is $4/\sqrt{3}$. *IEEE Transactions on Information Theory*, 60(8):5040–5053, 2014. [p93]

T. S. Hinks, X. Zhou, K. J. Staples, B. D. Dimitrov, A. Manta, T. Petrossian, P. Y. Lum, C. G. Smith, J. A. Ward, P. H. Howarth, et al. Innate and adaptive t cells in asthmatic patients: relationship to severity and disease mechanisms. *Journal of allergy and clinical immunology*, 136(2):323–333, 2015. [p90]

M. Lawrence, R. Gentleman, and V. Carey. rtracklayer: an r package for interfacing with genome browsers. *Bioinformatics*, 25(14):1841, 2009. [p91]

Y. Liao, G. K. Smyth, and W. Shi. The r package rsubread is easier, faster, cheaper and better for alignment and quantification of rna sequencing reads. *Nucleic acids research*, 47(8):e47–e47, 2019. [p91]

P. Y. Lum, G. Singh, A. Lehman, T. Ishkanov, M. Vejdemo-Johansson, M. Alagappan, J. Carlsson, and G. Carlsson. Extracting insights from the shape of complex data using topology. *Scientific reports*, 3 (1):1236, 2013. [p90, 95]

J. C. Mathews, S. Nadeem, A. J. Levine, M. Pouryahya, J. O. Deasy, and A. Tannenbaum. Robust and interpretable pam50 reclassification exhibits survival advantage for myoepithelial and immune phenotypes. *NPJ Breast Cancer*, 5(1):30, 2019. [p90]

M. Nicolau, R. Tibshirani, A.-L. Børresen-Dale, and S. S. Jeffrey. Disease-specific genomic analysis: identifying the signature of pathologic biology. *Bioinformatics*, 23(8):957–965, 02 2007. ISSN 1367-4803. doi: 10.1093/bioinformatics/btm033. URL `https://doi.org/10.1093/bioinformatics/btm033`. [p90]

M. Nicolau, A. J. Levine, and G. Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *10.1073/pnas.1102826108*, 2011. [p90]

A. H. Rizvi, P. G. Camara, E. K. Kandror, T. J. Roberts, I. Schieren, T. Maniatis, and R. Rabadan. Single-cell topological rna-seq analysis reveals insights into cellular differentiation and development. *Nature biotechnology*, 35(6):551–560, 2017. [p90]

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. [p96]

M. S. Schroder, A. C. Culhane, J. Quackenbush, and B. Haibe-Kains. survcomp: an r/bioconductor package for performance assessment and comparison of survival models. *Bioinformatics*, 27(22): 3206–3208, 2011. [p91]

G. Singh, F. Mémoli, G. E. Carlsson, et al. Topological methods for the analysis of high dimensional data sets and 3d object recognition. *PBG@ Eurographics*, 2:091–100, 2007. [p90, 95]

D. J. Sokolowski, M. Faykoo-Martinez, L. Erdman, H. Hou, C. Chan, H. Zhu, M. M. Holmes, A. Goldenberg, and M. D. Wilson. Single-cell mapper (scmappr): using scrna-seq to infer the cell-type specificities of differentially expressed genes. *NAR Genomics and Bioinformatics*, 3(1):lqab011, 2021. [p91]

Miriam Esteve
Universidad Cardenal Herrera-CEU
Department of Matematicas, Fisica y Ciencias Tecnologicas, 03203 Carmelitas, 3 (Elche), Spain
ORCiD: 0000-0002-5908-0581
`miriam.estevecampello@uchceu.es`

Raquel Bosch-Romeu
Universidad Cardenal Herrera-CEU
Department of Matematicas, Fisica y Ciencias Tecnologicas, San Bartolome 55, Alfara del Patriarca (Valencia), Spain
ORCiD: 0000-0001-9126-3241
`raquel.boschromeu@uchceu.es`

Antonio Falco
Universidad Cardenal Herrera-CEU
Department of Matematicas, Fisica y Ciencias Tecnologicas, ESI International Chair at CEU UCH, 03203 Carmelitas, 3 (Elche) Spain
ORCiD: 0000-0001-6225-0935
`afalco@uchceu.es`

Jaume Fores
Universidad Cardenal Herrera-CEU
Department of Matematicas, Fisica y Ciencias Tecnologicas, San Bartolome 55, Alfara del Patriarca (Valencia), Spain
ORCiD: 0000-0002-9025-4877
`fores.martos.jaume@gmail.com`

Joan Climent
Universidad Cardenal Herrera-CEU

Departamento de Producción y Sanidad Animal, Salud Pública Veterinaria y Ciencia y Tecnología de los Alimentos (PASAPTA), C/ Tirant lo Blanc, 7. 46115, Valencia
ORCiD: 0000-0002-8927-6614
joan.climentbataller@uchceu.es