# diffuStats: an R package to compute diffusion-based scores on biological networks

***Sergio Picart-Armada*** [*1,2], ***Wesley K. Thompson*** [3,4], ***Alfonso Buil*** [3], ***and Alexandre Perera-Lluna*** [1,2]

[1]B2SLab, Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial, Universitat Politècnica de Catalunya, CIBER-BBN, Barcelona, 08028, Spain
[2]Institut de Recerca Pediàtrica Hospital Sant Joan de Déu, Esplugues de Llobregat, Barcelona, 08950, Spain
[3]Mental Health Center Sct. Hans, 4000 Roskilde, Denmark
[4]Department of Family Medicine and Public Health, University of California, San Diego, La Jolla, California, USA

*sergi.picart@upc.edu

**November 24, 2026**

## 1 Abstract

Label propagation approaches are a standard and ubiquitous procedure in computational biology for giving context to molecular entities. Node labels, which can derive from gene expression, genome-wide association studies, protein domains or metabolomics profiling, are propagated to their neighbours, effectively smoothing the scores through prior annotated knowledge and prioritising novel candidates. However, there are several settings to tune when defining the diffusion process, including the diffusion kernel, the numeric codification of the labels and a choice of statistical normalisation of the scores. These settings can have a large impact on results, and there is currently no software implementing many of them in one place to screen their performance in the application of interest. This vignette presents *diffuStats*, an R package with a collection of diffusion kernels and scores, as well as a parallel permutation analysis for the normalised scores, that eases the analysis of several sets of molecular entities at once.

# 2 Introduction

The application of label propagation algorithms [1] is based on the guilt by association principle [2], which can be rephrased in the protein-protein interaction context as "proteins that interact are more likely to share biological functions". However, this principle is extremely general and has experienced success in numerous applications in bioinformatics.

HotNet [3] uses a diffusion process with mutated genes as seed nodes to find modules with a statistically high number of mutated genes in cancer. Another attempt to find relevant modules from gene expression and mutation data can be found in [4], where the authors propose a diffusion process followed by a statistical normalisation and an automatic process to extract a subnetwork. TieDIE [5] runs two diffusion processes to link perturbation in the genome with changes in the transcriptome, effectively linking two sets of genes. GeneMANIA [6] is a web server that predicts gene function using label propagation with a bias on the unlabelled nodes. Network-based learning sharing a background with diffusion has been also applied to protein classification using multiple networks [7]. Label propagation using graph kernels has been proven successful in gene-disease association [8, 9]. Equivalent formulations can be found under different terminology, like the electrical model applied to prioritise candidate genes in eQTL in [10].

The heterogeneity of applications hinders comparisons among approaches, therefore tools gathering the state of the art are highly needed. An existing solution is *RANKS*, an R package that contains a variety of diffusion kernels and kernelised scores for label propagation using a binary input vector. *RANKS* eases kernelised scores benchmarking and models it as a "one-class" classification semi-supervised learning problem, in which only some members of the class (positives) are known. Another possibility is to divide nodes into labelled positive, negative and unlabelled, like in [6], which poses questions like the effect of unlabelled nodes and possible numeric codifications of the labels, or the option to include quantitative data in the labels. In addition, statistical normalisations such as in [4] remove the effect of network structures like hubs and should be taken into account when choosing a diffusion scoring method. This motivates the introduction of our *diffuStats* R package, which collects widely adopted input codifications and explicitly accounts for unlabelled nodes. It also includes three statistically normalised scores, which can be obtained through Monte Carlo trials or a parametric alternative. The *diffuStats* package uses existent classes and provides high-level functions to screen the performance of several diffusion scores, in order to facilitate their integration in any computational biology study.

# 3 Methodology

One of the main purposes of *diffuStats* is to offer a battery of approaches to compute and compare diffusion scores. The diffusion scores $f$ using an input vector $y$ and a diffusion kernel $K$ are generally computed as

$$f = K \cdot y$$

possibly followed by further adjustments or a statistical normalisation.

The decisions taken in the definition of $K$, $y$ and the posterior normalisation generally give rise to different priorisations due to a different treatment of the balance between positive and negative examples, the unlabelled data and the network structure. The following sections cover the implemented choices for the kernel $K$ and the initial labels $y$.

## 3.1 Diffusion kernels and regularisation

The representation of any kind of data in a network model allows the definition of notions like distance or similarity based on the links in the network. This section will follow the notation in [11] and summarise the kernels proposed by the authors. In general, an undirected graph $G = (V, E)$ consists of a set of $n$ nodes $V$ and a set of edges $E$ of unordered pairs of nodes. This can be extended to weighted, undirected graphs, where each edge $i \sim j$ has a weight attribute $W_{ij} \in [0, \infty)$. The degree matrix of $G$ is defined as the $n \times n$ diagonal matrix so that $D_{ii} = \sum_{j=1}^{n} W_{ij}$. The (unnormalised) Laplacian of $G$ is defined as the $n \times n$ matrix $L = D - W$, whereas its normalised version is $\tilde{L} = D^{-\frac{1}{2}} \cdot L \cdot D^{-\frac{1}{2}}$.

The graph Laplacian is diagonalisable and can be written in terms of its eigenvalues $v_j$ and eigenvectors $\lambda_j$, as $L = \sum_{j=1}^{n} \lambda_j v_j v_j^T$. The proposed kernels stem from a family of regularisation functions $r(\lambda)$ on the spectrum of the graph Laplacian:

$$K = \sum_{j=1}^{n} r^{-1}(\lambda_j) v_j v_j^T$$

Well known graph kernels belong to this family because they can be written as transformations on the Laplacian spectrum. Table 1 summarises them, assuming the usage of the normalised Laplacian - the unnormalised Laplacian can also be used as long as the resulting kernel is still positive semidefinite. Further details about this family of kernels, all available in our package *diffuStats*, can be found in the original manuscript [11].

Additionally, the *diffuStats* package includes the commute time kernel, introduced in [12]. This kernel, also writable in terms of a regularisation function, is simply the pseudoinverse of the graph Laplacian, $K = L^{+}$.

| Kernel | Function |
|---|---|
| Regularised Laplacian | $r(\lambda) = 1 + \sigma^2\lambda$ |
| Diffusion process | $r(\lambda) = \exp(\frac{\sigma^2}{2}\lambda)$ |
| $p$-Step random walk | $r(\lambda) = (a - \lambda)^{-p}$ with $a \geq 2$, $p \geq 1$ |
| Inverse cosine | $r(\lambda) = (cos(\lambda\frac{\pi}{4}))^{-1}$ |

**Table 1:** Implemented diffusion kernels from [11]

| Score | $y^+$ | $y^-$ | $y^u$ | Normalised | Stochastic | Quantitative | Reference |
|---|---|---|---|---|---|---|---|
| raw | 1 | 0 | 0 | No | No | Yes | [3] |
| ml | 1 | -1 | 0 | No | No | No | [7, 1] |
| gm | 1 | -1 | $k$ | No | No | No | [6] |
| $ber_s$ | 1 | 0 | 0 | No | No | Yes | [4] |
| $ber_p$ | 1 | 0 | 0* | Yes | Yes | Yes | [4] |
| mc | 1 | 0 | 0* | Yes | Yes | Yes | [4] |
| z | 1 | 0 | 0* | Yes | No | Yes | [13] |

**Table 2:** Implemented diffusion scores

The default option in the *diffuStats* package is the regularised Laplacian kernel, as it is widely adopted and describes many physical models, for instance in [3, 5, 10].

## 3.2 Diffusion scores

Besides choosing a graph kernel, the codification of the input and the presence of a statistical normalisation can lead to important differences in the results. Table 2 gives an overview of the implemented scores, which will be detailed in the following sections. The argument *method* in the function `diffuse` can be set to the desired scores in table 2, which are described in the following sections. The numeric values of the positive, negative and unlabelled examples are respectively $y^+$, $y^-$ and $y^u$. Column "normalised" refers to the application of a statistical model involving permutations, whereas "stochastic" enumerates the normalised scores that need actual Monte Carlo permutations. The scores that also accept quantitative inputs instead of binary labels are listed in the "quantitative" column.

### 3.2.1 Scores without statistical normalisation

The base diffusion score `raw`, which has been used in algorithms like HotNet [3] and TieDIE [5], solves a diffusion problem in terms of the regularised Laplacian kernel [11].

$$f_{raw} = K \cdot y_{raw}$$

$K$ is the kernel matrix and $y_{raw}$ the vector of codified inputs. In general, the $i$-th component of $y$ equals $y^+$ if node $i$ is a positive, $y^-$ if $i$ is a negative and $y^u$ if $i$ is unlabelled. In the particular case of $y_{raw}$, the positively labelled nodes introduce one flow unit in the network ($y^+_{raw} = 1$), whereas the negative and unlabelled nodes are treated equivalently and do not introduce anything ($y^-_{raw} = y^u_{raw} = 0$). In the physical model using the regularised Laplacian kernel, the flow can be evacuated from the graph due to the presence of first-order leaking in every node, see [3] for further details on this. This formulation is proportional up to a scaling factor to the average score in *RANKS*.

On the other hand, the classical label propagation [1] treats positives as $y^+_{ml} = 1$ and negatives as $y^-_{ml} = -1$, while unlabelled nodes remain as $y^u_{ml} = 0$, thus making a distinction between the last two. A biological example can be found in protein classification [7]. This option is available as `ml`, and intuitively scores a node by counting if the majority of its neighbours vote positive or negative:

$$f_{ml} = K \cdot y_{ml}$$

The authors of GeneMANIA [6] propose a modification on the `ml` input - they adhere to $y^+_{gm} = 1$ and $y^-_{gm} = -1$, but introduce a bias term in the unlabelled nodes

$$y^u_{gm} = \frac{n^+ - n^-}{n^+ + n^- + n^u}$$

being $n^+$, $n^-$ and $n^u$ the number of positives, negatives and unlabelled entities. The `gm` score is then computed through

$$f_{gm} = K \cdot y_{gm}$$

The last option in this part, named `ber_s` [4], is a quantification of the relative change in the node score before and after the network smoothing. The score for a particular node $i$ can be written as

$$f_{ber_s,i} = \frac{f_{raw,i}}{y_{raw,i} + \epsilon}$$

where $\epsilon > 0$ is a parameter that regulates the importance of the relative change.

### 3.2.2 Scores with statistical normalisation

Recently, the combination of a permutation analysis with diffusion processes has been suggested [4]. This is a way to quantify how the diffusion score of a certain node compares to its score if the input was randomised - nodes that might have systematically high or low scores regardless of the input are normalised accordingly.

The cornerstone of normalised scores is the empirical p-value [14] that indicates, for a node $i$, the proportion of input permutations that led to a diffusion score as high or higher than the original diffusion score. Specifically, $f_{raw}$ is compared to scores from random trials $j$, $f_{raw}^{null,j} = K \cdot \pi_j(y_{raw})$, where $\pi_j(y_{raw})$ is a permutation of $y_{raw}$ on the labelled entities. The empirical p-value for node $i$ is therefore defined as in [14]:

$$p_i = \frac{r_i + 1}{n + 1}$$

being $r_i$ the number of trials $j$ in which $f_{raw,i}^{null,j} \geq f_{raw,i}$, and $n$ the total number of trials.

To be consistent with the increasing direction of the scores, the `mc` scores are defined as

$$f_{mc,i} = 1 - p_i$$

Importantly, the permutation has been applied only to the observed nodes. Therefore, any node outside the labelled background cannot receive a different input score in a permuted input. This implies that even if both negatives and unlabelled nodes are assigned the same input value ($y^- = y^u = 0$), the negatives are actually permuted and eventually exchanged for a $1$ in some permutations provided that the number of runs is large enough. For this reason, negatives and unlabelled nodes are not equivalent in these scores.

A parametric alternative to $f_{mc}$, are `z` scores, where each node $i$ is scored as:

$$f_{z,i} = \frac{f_{raw,i} - E(f_{raw,i}^{null,j})}{\sqrt{V(f_{raw,i}^{null,j})}}$$

The expectation and variance are computed in a closed form and these scores do not require running actual permutations, therefore saving on computational time and avoiding stochastic models. The analytical expression proven below also works for the general case when the input has quantitative labels.

First of all, note that all the unlabelled nodes will cancel out when substracting the expected value, so they can be excluded without loss of generality. Thus, let the row vector $\tilde{k}_i$ be the $i$-th row of the submatrix from $K$ including the columns

indexed by the labelled nodes only, and let $n_{lab}$ be the number of labelled nodes. Analogously, let $\tilde{y}$ be the (quantitative or binary) inputs for the observed nodes, with the same indexing as $\tilde{k}_i$. Consider the following scalar quantities:

$$Sk_i^I = \sum_{j=1}^{n_{lab}} (\tilde{k}_i)_j$$

$$Sk_i^{II} = \sum_{j=1}^{n_{lab}} (\tilde{k}_i)_j^2$$

$$Sy^I = \sum_{j=1}^{n_{lab}} \tilde{y}_j$$

$$Sy^{II} = \sum_{j=1}^{n_{lab}} \tilde{y}_j^2$$

Using these definitions and notation, the raw score for node $i$ is

$$f_i = \tilde{k}_i \cdot \tilde{y}$$

Its null score using a permuted input score is the random variable

$$f_i^{null} = \tilde{k}_i \cdot X$$

where $X$ is the random variable that arises from permuting $\tilde{y}$. In order to compute the z-scores, the expected value and variance of $f_i^{null}$ are needed. Starting with its expected value,

$$E_X(f_i^{null}) = E_X(\tilde{k}_i \cdot X) = \tilde{k}_i \cdot E_X(X) = \tilde{k}_i \cdot \frac{\sum_{j=1}^{n_{lab}} \tilde{y}_j}{n_{lab}} \cdot \mathbb{1} = \frac{Sk_i^I \cdot Sy^I}{n_{lab}}$$

where $\mathbb{1}$ is the $n_{lab}$-th dimensional column vector full of ones. Regarding its variance,

$$V_X(f_i^{null}) = V_X(\tilde{k}_i \cdot X) = \tilde{k}_i \cdot V_X(X) \cdot \tilde{k}_i^T$$

The covariance of $X$ can be written as

$$V_X(X) = \left[ \frac{\sum_{j=1}^{n_{lab}} \tilde{y}_j^2}{n_{lab}} - \left( \frac{\sum_{j=1}^{n_{lab}} \tilde{y}_j}{n_{lab}} \right)^2 \right] \left[ \frac{n_{lab}}{n_{lab}-1} Id - \frac{1}{n_{lab}-1} \mathbb{1}\mathbb{1}^T \right]$$

being $Id$ the $n_{lab} \times n_{lab}$ identity matrix. Operating, the variance of $f_i^{null}$ can be finally computed as

$$V_X(f_i^{null}) = \frac{1}{(n_{lab}-1)n_{lab}^2} \left[ n_{lab} Sy^{II} - (Sy^I)^2 \right] \left[ n_{lab} Sk_i^{II} - (Sk_i^I)^2 \right]$$

The z-score for node $i$ is, in terms of the new notation:

$$f_{z,i} = \frac{f_i - E_X(f_i^{null})}{\sqrt{V_X(f_i^{null})}}$$

This closes the `z` scoring option, which clearly does not require any actual permutations but normalises through the theoretical first and second statistical moments of the null distribution of the diffusion scores.

Finally, the authors in [4] also suggest a combination of a classical score with an statistically normalised one. Available as `ber_p`, the score of node $i$ is defined as

$$f_{ber_p,i} = -\log_{10}(p_i) \cdot f_{raw,i}$$

This approach corrects the original diffusion scores by the effect of the network, in order to mitigate the effect of structures like hubs.

### 3.2.3 Quantitative inputs

In its current release, *diffuStats* also accepts quantitative labels as input in the following scores: `raw`, `ber_s`, `z`, `ber_p` and `mc`. The scores `ml` and `gm` are naturally excluded from non-binary inputs due to their definitions. Currently `mc` scores (and therefore `ber_p` scores as well) accept quantitative inputs that are treated as sparse. Dense continuous inputs might take more time to compute, but this will be extended in future versions. Beware that quantitative inputs should be meaningful and one-tailed (monotonic) - the nature of diffusion processes involves averaging and strong effects with opposing signs cancel out instead of adding up.

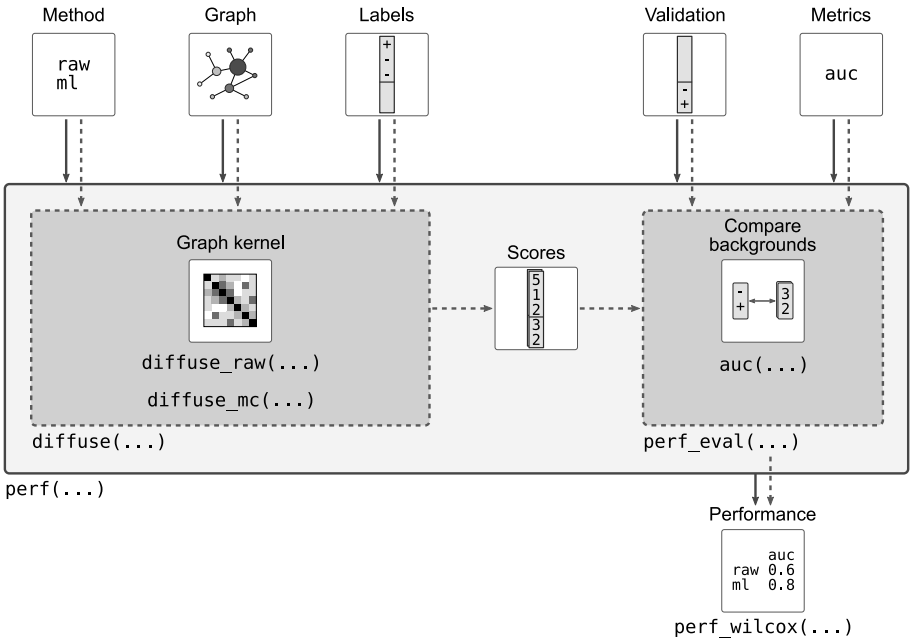



**Figure 1:** Overview of the main package functions.

## 3.3 Implementation, functions and classes

The package *diffuStats* is mainly implemented in R [15], but takes advantage of existing classes in *igraph* [16] and basic data types, thus not introducing any new data structure - this minimises the learning effort by the final user. Inputs and outputs are conceived to require minimal formatting effort in the whole analysis. The computationally intense stochastic part of the permutation analysis is implemented in C++ through the packages *Rcpp* [17], *RcppArmadillo* [18] and parallelised through *RcppParallel* [19]. Package *diffuStats* also contains documented functions and unit testing with small cases to spot potential bugs. Two vignettes facilitate the user experience: an introductory vignette showing the basic usage of the functions on a synthetic example and this vignette, which further describes the contents of the package and shows an application to real data.

A diagram containing the main functions in the R package *diffuStats* can be found in (Fig. 1). The main funcion is `diffuse`, a wrapper for computing diffusion scores from several categories at once, stemming from possibly different observed backgrounds. Function `diffuse` makes use of the deterministic `diffuse_raw` or the stochastic `diffuse_mc` implementations and combines them to give the desired scores for all the nodes in each of the observed backgrounds. The second wrapper `perf` compares the result of the diffusion scores to target validation scores. Validation scores might include only part of the nodes of the network and these nodes can be background-specific.

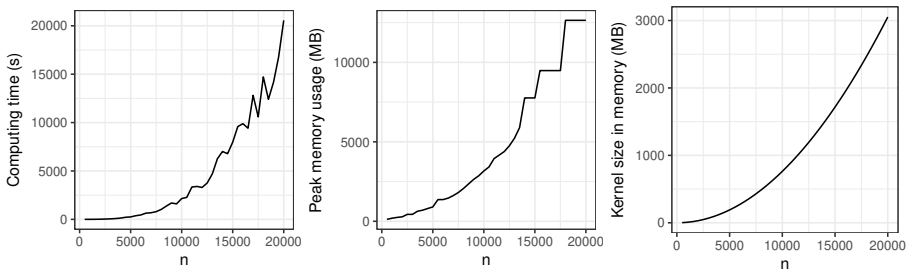



**Figure 2:** Profiling of the computation of the regularised Laplacian kernel on a synthetic n-th order network. Undirected networks are generated through `barabasi.game` in *igraph* using default parameters and $m = 6$ (each node adds 6 edges).

Regarding memory and processing power requirements, the analysis of networks with more than 10,000 nodes might need additional RAM memory and processing power. The main reason is the manipulation of dense graph kernel matrices that scale quadratically with the network order. To give a reference, a dense matrix of double-precision real numbers with 10,000 rows and columns uses roughly 800MB of memory. Computing a graph kernel on a large network can require -depending on the kernel- matrix diagonalisation, matrix products and matrix inversion operations that are likely to use a considerable amount of memory and time.

## 3.4 Limitations

The kernel framework is known to scale poorly with the number of nodes of the network when the kernel is explicitly computed and dense. Therefore, *diffuStats* is best suited for manipulating biological networks of a medium size - few tenths of thousands of nodes. Protein-protein interaction networks can have around 20,000 nodes, which is also the limit of the capabilities *diffuStats*, as it is now, in a standard workstation with 16GB of physical memory.

In particular, the explicit kernel computation is a demanding step, although it only has to be performed once with a given network. Figure 2 contains a profiling on the kernel computation using a Compaq q8100 workstation (intel core i5 650 @3.20GHz, 16GB physical memory). This should give an approximation of what to expect in terms of time and memory consumption. The same figure contains the memory usage of the kernel matrix per se.

On the other hand, the parallel implementation of the stochastic permutation analysis is another demanding task. It has been optimised assuming that the number of positives is usually low. Lists of scores with a very high amount of positives might slow down the permutation analysis, but not the parametric `z`.

# 4 Getting started

This vignette contains a classical example of label propagation on a biological network. The core tools for this analysis are the *igraph* R package [16] and the *diffuStats* package, whereas *ggplot2* [20] is a convenient tool to plot the results.

The data for this example is the `yeast` interactome with functional annotations, as found in the data package `igraphdata` [21].

```
> # Core
> library(igraph)
> library(diffuStats)
> # Plotting
> library(ggplot2)
> library(ggsci)
> # Data
> library(igraphdata)
> data(yeast)
> set.seed(1)
```

## 4.1 Data description

A summary of the network object can be obtained by just showing the object:

```
> summary(yeast)

IGRAPH 65c41bb UN-- 2617 11855 -- Yeast protein interactions, von Mering et al.
+ attr: name (g/c), Citation (g/c), Author (g/c), URL (g/c), Classes
| (g/x), name (v/c), Class (v/c), Description (v/c), Confidence (e/c)
```

For this analysis, only the largest connected component of this graph will be used, although the algorithms can handle graphs with several connected components.

```
> yeast <- diffuStats::largest_cc(yeast)
```

This yields to a graph with 2375 nodes and 11693 edges. There are several attributes that can be of interest. First of all, the `name` of the protein nodes:

```
> head(V(yeast)$name)

[1] "YLR197W" "YOR039W" "YDR473C" "YOR332W" "YER090W" "YDR394W"
```

Furthermore, the corresponding aliases and complete names can be found in `Description`

```
> head(V(yeast)$Description)

[1] "SIK1 involved in pre-rRNA processing"
[2] "CKB2 casein kinase II beta' chain"
[3] "PRP3 essential splicing factor"
[4] "VMA4 H+-ATPase V1 domain 27 KD subunit, vacuolar"
[5] "TRP2 anthranilate synthase component I"
[6] "RPT3 26S proteasome regulatory subunit"
```

The labels to perform network propagation are MIPS categories [22], which provide means to classify proteins regarding their function. These functions are coded as characters in the `yeast` object, in the node attribute `Class`

```
> table_classes <- table(V(yeast)$Class, useNA = "always")
> table_classes

   A    B    C    D    E    F    G    M    O    P    R    T    U <NA>
  51   98  122  238   95  171   96  278  171  248   45  240  483   39
```

The graph attribute `Classes` maps these abbreviations to the actual category:

```
> head(yeast$Classes)

  Category             Description
1        E       energy production
2        G    aminoacid metabolism
3        M        other metabolism
4        P             translation
5        T           transcription
6        B transcriptional control
                                            Original.MIPS.category
1                                                           energy
2                                             aminoacid metabolism
3                               all remaining metabolism categories
4                                                  protein synthesis
5 transcription, but without subcategory 'transcriptional control'
6                            subcategory 'transcriptional control'
```

Finally, the graph edges have a `Confidence` attribute that assesses the amount of evidence supporting the interaction. All the edges will be kept in this analysis, but different confidences can be weighted to favour diffusion in high confidence edges.

```
> table(E(yeast)$Confidence)

  high medium
  2395   9298
```

More on the yeast object can be found through `?yeast`.

## 4.2 First analysis: protein ranking

In this first case, the diffusion scores will be applied to the prediction of a single protein function. Let's assume that 50% of the labelled proteins in the graph as `transport and sensing` (category `A`) are actually unlabelled. Now, using the labels of the known positive and negative examples for `transport and sensing`, can we correctly label the remaining 50%? First of all, the list of known and unknown positives is generated. The function `diffuse` uses (row)names in the input scores so that unlabelled nodes are accounted as so.

```
> perc <- .5
> # Transport and sensing is class A
> nodes_A <- V(yeast)[Class %in% "A"]$name
> nodes_unlabelled <- V(yeast)[Class %in% c(NA, "U")]$name
> nodes_notA <- setdiff(V(yeast)$name, c(nodes_A, nodes_unlabelled))
> # Known labels
> known_A <- sample(nodes_A, perc*length(nodes_A))
> known_notA <- sample(nodes_notA, perc*length(nodes_notA))
> known <- c(known_A, known_notA)
> # Unknown target nodes
> target_A <- setdiff(nodes_A, known_A)
> target_notA <- setdiff(nodes_notA, known_notA)
> target <- c(target_A, target_notA)
> target_id <- V(yeast)$name %in% target
> # True scores
> scores_true <- V(yeast)$Class %in% "A"
```

Now that the input is ready, the diffusion algorithm can be applied to rank all the proteins. As a first approach, the vanilla diffusion scores will be computed through the *raw* method and the default regularised Laplacian kernel, which is calculated on the fly.

```
> # Vector of scores
> scores_A <- setNames((known %in% known_A)*1, known)
> # Diffusion
> diff <- diffuStats::diffuse(
```

```
+     yeast,
+     scores = scores_A,
+     method = "raw"
+ )
```

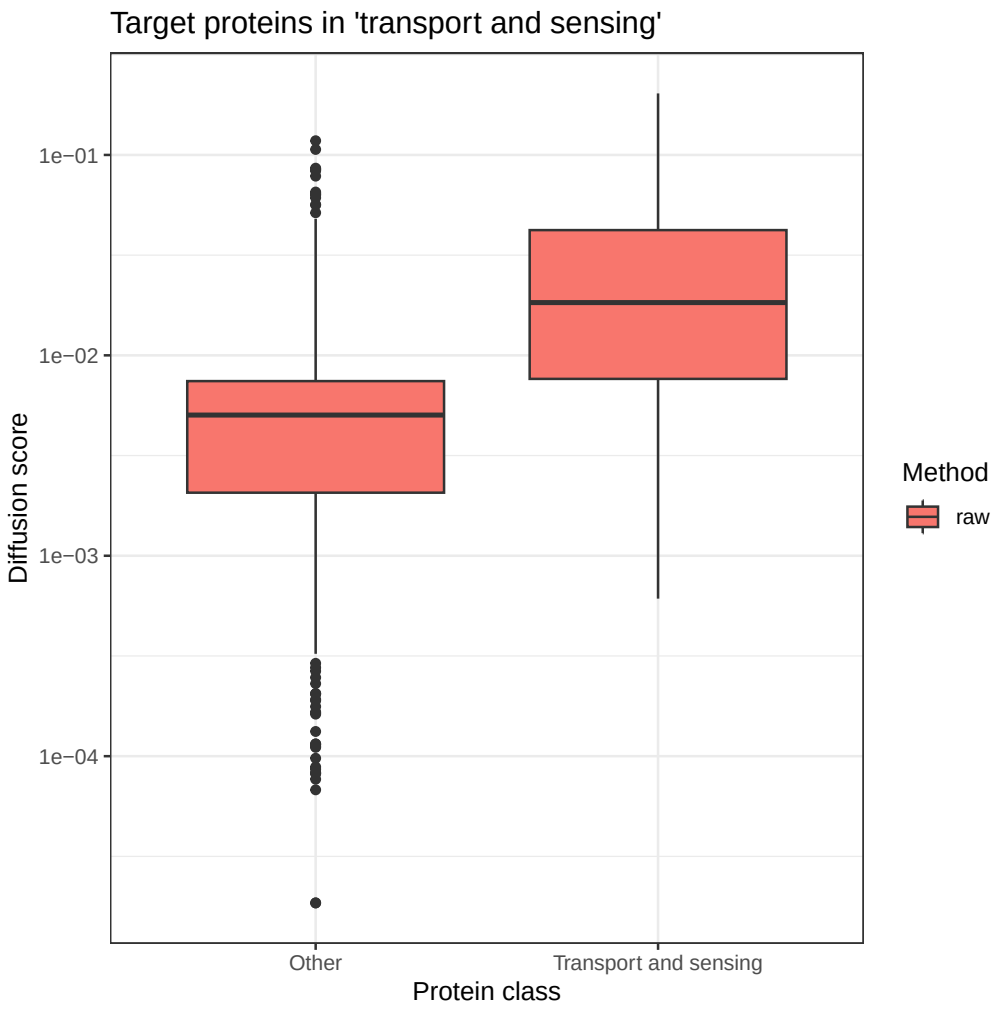
Diffusion scores are ready and in the same format they were introduced:

```
> head(diff)

    YLR197W     YOR039W     YDR473C     YOR332W     YER090W     YDR394W
0.004622066 0.003721601 0.003274780 0.085080780 0.009089522 0.006101765
```

Now, the scores obtained by the proteins actually belonging to `transport and sensing` can be compared to proteins with other labels.

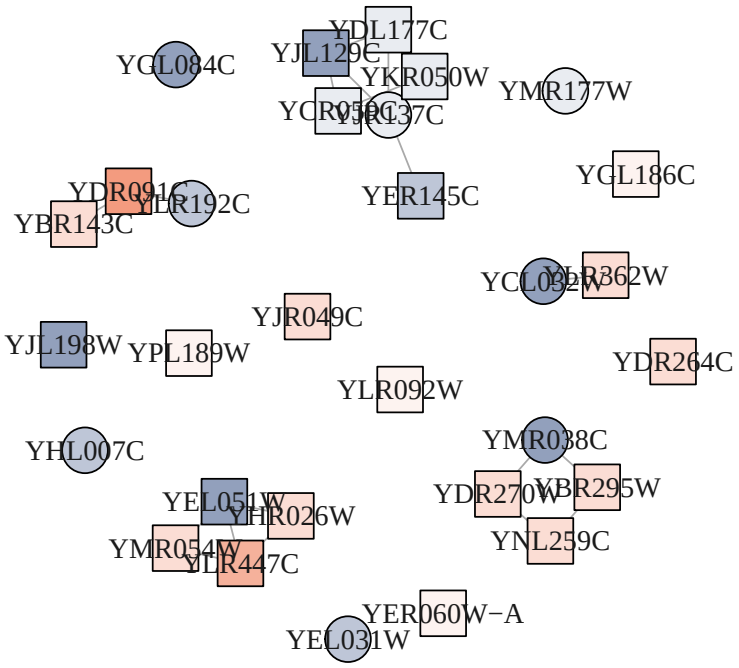
```
> # Compare scores
> df_plot <- data.frame(
+     Protein = V(yeast)$name,
+     Class = ifelse(scores_true, "Transport and sensing", "Other"),
+     DiffusionScore = diff,
+     Target = target_id,
+     Method = "raw",
+     stringsAsFactors = FALSE
+ )
> ggplot(subset(df_plot, Target), aes(x = Class, y = DiffusionScore)) +
+     geom_boxplot(aes(fill = Method)) +
+     theme_bw() +
+     scale_y_log10() +
+     xlab("Protein class") +
+     ylab("Diffusion score") +
+     ggtitle("Target proteins in 'transport and sensing'")
```




The last plot justifies the usefulness of label propagation, as proteins in `transport and sensing` obtain higher diffusion scores than the rest. The network analysis can be deepened by examining, for instance, the subnetwork containing the proteins with the top 30 diffusion scores, highlighting with squares the ones that were positive labels in the input. Notice the small clusters of proteins:

```
> # Top scores subnetwork
> vertex_ids <- head(order(df_plot$DiffusionScore, decreasing = TRUE), 30)
> yeast_top <- igraph::induced.subgraph(yeast, vertex_ids)
> # Overlay desired properties
> # use tkplot for interactive plotting
> igraph::plot.igraph(
+     yeast_top,
+     vertex.color = diffuStats::scores2colours(
+         df_plot$DiffusionScore[vertex_ids]),
```

```
+     vertex.shape = diffuStats::scores2shapes(
+         df_plot$Protein[vertex_ids] %in% known_A),
+     vertex.label.color = "gray10",
+     main = "Top 30 proteins from diffusion scores"
+ )
```




## 4.3 Second example: comparing scores with single protein ranking

The proposed diffusion scores can be easily applied and compared. The regularised Laplacian kernel will be used to compute all the implemented scores for the target nodes in `transport and sensing`.

```
> K_rl <- diffuStats::regularisedLaplacianKernel(yeast)
```

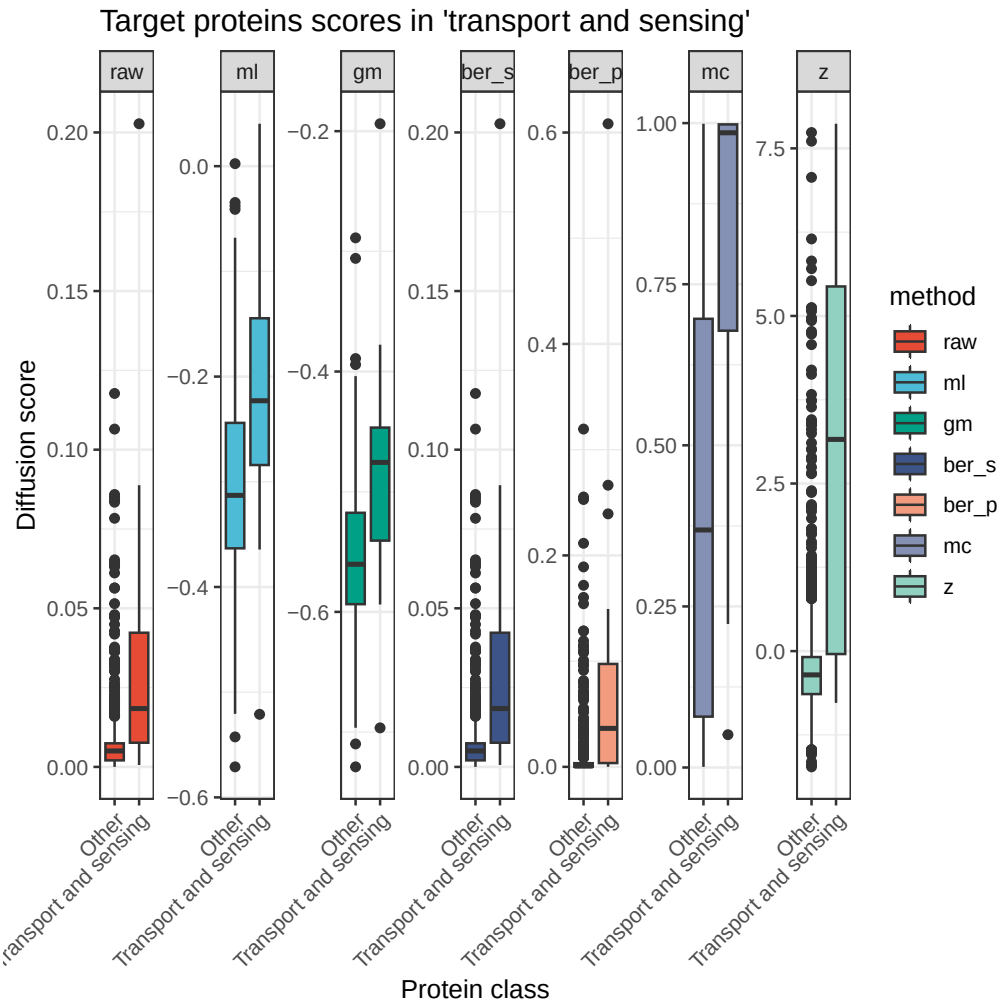
Functions `diffuse` and `perf` do accept, however, an `igraph` object as well, and compute the kernel automatically. For medium networks (10,000 nodes or more) the kernel computation can be computationally expensive in memory and time, so precomputing it avoids unnecessary recalculations.

The diffusion scores can be computed over a list of methods or sets of parameters. This can be achieved with instructions like `lapply`, but *diffuStats* contains a wrapper to facilitate this task. The function `diffuse_grid` takes the specified combinations of parameters -which can include the scoring method as well- and computes the diffusion scores for each one. The results are concatenated in a data frame that can be easily plotted:

```
> list_methods <- c("raw", "ml", "gm", "ber_s", "ber_p", "mc", "z")
> df_diff <- diffuse_grid(
+     K = K_rl,
+     scores = scores_A,
+     grid_param = expand.grid(method = list_methods),
+     n.perm = 1000
+ )
> df_diff$transport <- ifelse(
+     df_diff$node_id %in% nodes_A,
+     "Transport and sensing",
+     "Other"
+ )
```

The results can be directly plotted:

```
> df_plot <- subset(df_diff, node_id %in% target)
> ggplot(df_plot, aes(x = transport, y = node_score)) +
+     geom_boxplot(aes(fill = method)) +
+     scale_fill_npg() +
+     theme_bw() +
+     theme(axis.text.x = element_text(
+         angle = 45, vjust = 1, hjust = 1)) +
+     facet_wrap( ~ method, nrow = 1, scales = "free") +
+     xlab("Protein class") +
+     ylab("Diffusion score") +
+     ggtitle("Target proteins scores in 'transport and sensing'")
```




As expected, all the diffusion scores qualitatively show differences between positive and negative labels, but the quality of class separation will generally depend on the dataset and scoring method.

## 4.4 Third example: benchmarking scores with multiple protein functions

The package *diffuStats* is able to perform several screenings at once. To show its usefulness, we will generalise the procedure in the last section but screening all the categories in the yeast graph.

First of all, the input data must meet an adequate format - a straightforward approach is to populate a matrix with the input labels (one category per column).

```
> # All classes except NA and unlabelled
> names_classes <- setdiff(names(table_classes), c("U", NA))
> # matrix format
> mat_classes <- sapply(
+     names_classes,
+     function(class) {
+         V(yeast)$Class %in% class
+     }
+ )*1
> rownames(mat_classes) <- V(yeast)$name
> colnames(mat_classes) <- names_classes
```

The former 50% known / 50% unknown approach will be kept with the same split, although not all the 12 categories will be totally balanced in the splits now. All the methods will be compared using the area under the ROC curve (AUROC) as a performance index.

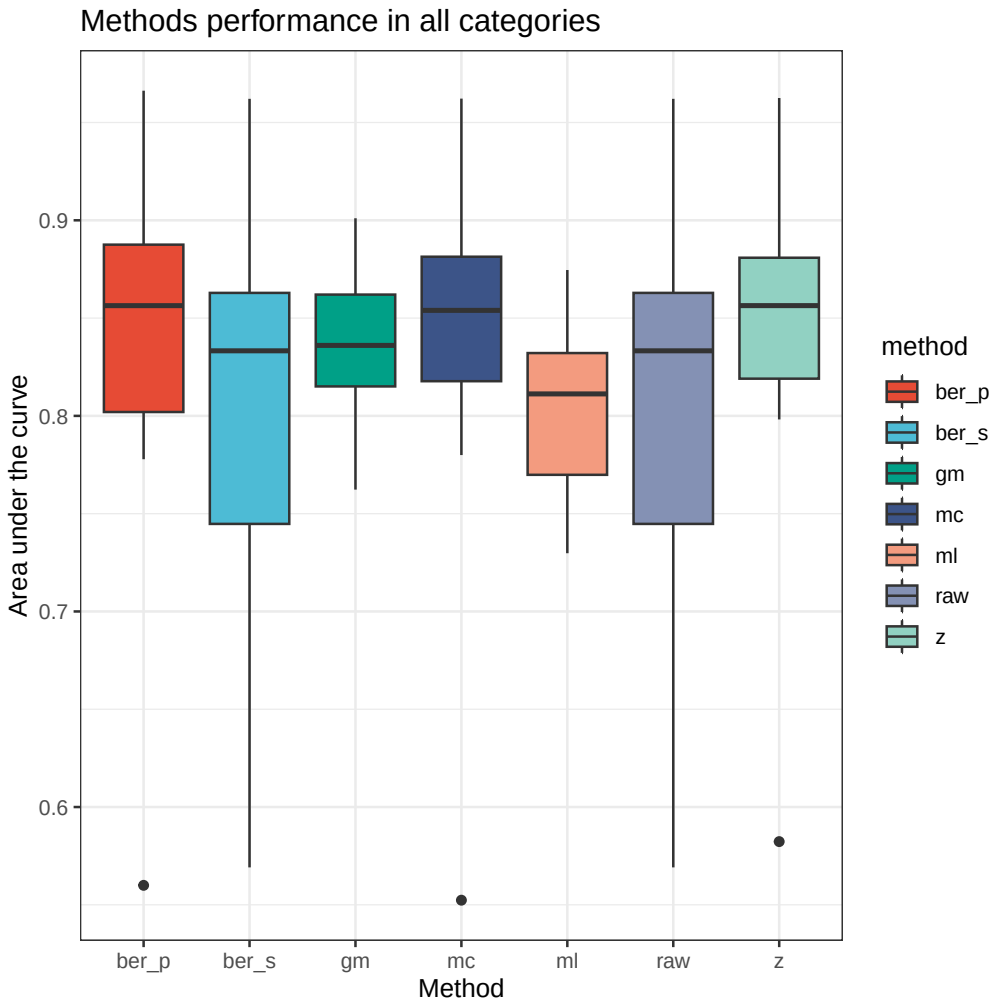
Please note that *diffuStats* is equipped with basic performance measures for rankers: the AUROC, the area under the Precision-Recall curve, or AUPRC, and their partial versions. These are available through the helper function `metric_fun` and can be passed in list format to `perf`. These measures are based on the *precrec* R package - further detail can be found in the original manuscript [23].

```
> list_methods <- c("raw", "ml", "gm", "ber_s", "ber_p", "mc", "z")
> df_methods <- perf(
+     K = K_rl,
+     scores = mat_classes[known, ],
+     validation = mat_classes[target, ],
+     grid_param = expand.grid(
+         method = list_methods,
+         stringsAsFactors = FALSE),
+     n.perm = 1000
+ )
```

This allows plotting of the AUCs over the categories for each method in one step:

```
> ggplot(df_methods, aes(x = method, y = auc)) +
+     geom_boxplot(aes(fill = method)) +
+     scale_fill_npg() +
+     theme_bw() +
+     xlab("Method") +
```

```
+     ylab("Area under the curve") +
+     ggtitle("Methods performance in all categories")
```




Scaling up the analysis can be useful for assessing how adequate a diffusion score is in the dataset of interest. These results suggest that, for the current `yeast` interactome and protein functions, the best priorisations are those obtained through a statistical normalisation, which might motivate its usage in other biological networks.

The user can also statistically compare the performance metrics through the function `perf_wilcox`. This generates a table with (i) the estimates on the differences on performance between the methods in rows and columns, with their confidence intervals and (ii) their associated p-value (Wilcoxon test). Positive and negative estimates respectively favour the method in the row and the column.

```
> # Format the data
> df_perf <- reshape2::acast(df_methods, Column~method, value.var = "auc")
> # Compute the comparison matrix
> df_test <- perf_wilcox(
+     df_perf,
+     digits_p = 1,
+     adjust = function(p) p.adjust(p, method = "fdr"),
+     scientific = FALSE)
> knitr::kable(df_test, format = "latex")
```

| | ber_p | ber_s | gm | mc | ml |
|---|---|---|---|---|---|
| ber_p | NA | 0.023(0.0097,0.036) | 0.02(-0.054,0.049) | 0.0015(-0.0076,0.0071) | 0.042(-0.047,0.0 |
| ber_s | 0.03 | NA | -0.0084(-0.082,0.025) | -0.022(-0.044,-0.0001) | 0.018(-0.075,0.0 |
| gm | 0.57 | 0.75 | NA | -0.02(-0.051,0.057) | 0.025(0.013,0.0 |
| mc | 0.77 | 0.13 | 0.58 | NA | 0.047(-0.043,0.0 |
| ml | 0.49 | 0.58 | 0.01 | 0.37 | NA |
| raw | 0.03 | 1.00 | 0.75 | 0.13 | 0.58 |
| z | 0.49 | 0.03 | 0.49 | 0.35 | 0.34 |

# 5 Conclusions

The *diffuStats* package is a new computational tool to compute and compare single-network diffusion scores that are object of active research in several bioinformatics areas. It is an effort to gather a collection of settings in the diffusion process like the graph kernel, the label codification and the choice of a statistical normalisation. The *diffuStats* package will help the end user in choosing and computing the best performing diffusion scores in the application of interest.

# 6 Funding

This work was supported by the Spanish Ministry of Economy and Competitiveness (MINECO) [TEC2014-60337-R to A.P.] and the National Institutes of Health (NIH) [R01GM104400 to W.T.]. AP. and S.P. thank for funding the Spanish Biomedical Research Centre in Diabetes and Associated Metabolic Disorders (CIBERDEM) and the Networking Biomedical Research Centre in the subject area of Bioengineering, Biomaterials and Nanomedicine (CIBER-BBN), both initiatives of Instituto de Investigación Carlos III (ISCIII). SP. thanks the AGAUR FI-scholarship programme.

# References


[1] Olga Zoidi, Eftychia Fotiadou, Nikos Nikolaidis, and Ioannis Pitas. Graph-Based Label Propagation in Digital Media: A Review. *ACM Computing Surveys*, 47(3):48:1–48:35, 2015. doi:10.1145/2700381.

[2] Stephen Oliver. Guilt-by-association goes global. *Nature*, 403(February):601–603, 2000.

[3] Fabio Vandin, Eli Upfal, and Benjamin J. Raphael. Algorithms for detecting significantly mutated pathways in cancer. *Lect. Notes Comput. Sci.*, 6044 LNBI(3):506–521, 2010. doi:10.1007/978-3-642-12683-3_33.

[4] Matteo Bersanelli, Ettore Mosca, Daniel Remondini, Gastone Castellani, and Luciano Milanesi. Network diffusion-based analysis of high-throughput data for the detection of differentially enriched modules. *Scientific Reports*, 6(August):34841, 2016. doi:10.1038/srep34841.

[5] Evan O. Paull, Daniel E. Carlin, Mario Niepel, Peter K. Sorger, David Haussler, and Joshua M. Stuart. Discovering causal pathways linking genomic events to transcriptional states using Tied Diffusion Through Interacting Events (TieDIE). *Bioinformatics*, 29(21):2757–2764, 2013. doi:10.1093/bioinformatics/btt471.

[6] Sara Mostafavi, Debajyoti Ray, David Warde-Farley, Chris Grouios, and Quaid Morris. GeneMANIA: a real-time multiple association network integration algorithm for predicting gene function. *Genome Biology*, 9 Suppl 1:S4, 2008. doi:10.1186/gb-2008-9-s1-s4.

[7] Koji Tsuda, HyunJung J. Shin, and Bernhard Schölkopf. Fast protein classification with multiple networks. *Bioinformatics*, 21(SUPPL. 2):59–65, 2005. doi:10.1093/bioinformatics/bti1110.

[8] Giorgio Valentini, Alberto Paccanaro, Horacio Caniza, Alfonso E. Romero, and Matteo Re. An extensive analysis of disease-gene associations using network integration and fast kernel-based gene prioritization methods. *Artificial Intelligence in Medicine*, 61(2):63–78, 2014. doi:10.1016/j.artmed.2014.03.003.

[9] Insuk Lee, U Martin Blom, Peggy I Wang, Jung Eun Shim, and Edward M Marcotte. Prioritizing candidate disease genes by network-based boosting of genome-wide association data. *Genome Research*, 21(7):1109–1121, 2011. doi:10.1101/gr.118992.110.

[10] Silpa Suthram, Andreas Beyer, Richard M Karp, Yonina Eldar, and Trey Ideker. eqed: an efficient method for interpreting eqtl associations using protein networks. *Molecular systems biology*, 4(1):162, 2008.

[11] Alexander J Smola and Risi Kondor. Kernels and regularization on graphs. pages 144–158, 2003. doi:10.1007/978-3-540-45167-9_12.

[12] Luh Yen, Francois Fouss, and Christine Decaestecker. Graph nodes clustering based on the commute-time kernel. *Advances in Knowledge Discovery and Data Mining*, pages 1037–1045, 2007. doi:10.1007/978-3-540-71701-0_117.

[13] Zaid Harchaoui, Francis Bach, Olivier Cappe, and Eric Moulines. Kernel-based methods for hypothesis testing: A unified view. *IEEE Signal Processing Magazine*, 30(4):87–97, 2013.

[14] Bernard V North, David Curtis, and Pak C Sham. A note on the calculation of empirical p values from monte carlo procedures. *The American Journal of Human Genetics*, 71(2):439–441, 2002.

[15] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2017. URL: https://www.R-project.org/.

[16] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 2006. URL: http://igraph.org.

[17] Dirk Eddelbuettel. *Seamless R and C++ integration with Rcpp*. Springer, 2013.

[18] Dirk Eddelbuettel and Conrad Sanderson. Rcpparmadillo: Accelerating r with high-performance c++ linear algebra. *Computational Statistics and Data Analysis*, 71:1054–1063, March 2014. doi:10.1016/j.csda.2013.02.005.

[19] JJ Allaire, Romain Francois, Kevin Ushey, Gregory Vandenbrouck, Marcus Geelnard, and Intel. *RcppParallel: Parallel Programming Tools for 'Rcpp'*, 2016. R package version 4.3.20. URL: https://CRAN.R-project.org/package=RcppParallel.

[20] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2009. URL: http://ggplot2.org.

[21] Gabor Csardi. igraphdata: A Collection of Network Data Sets for the 'igraph' Package. 2015. R package version 1.0.1. URL: https://CRAN.R-project.org/package=igraphdata.

[22] Hans-Werner Mewes, Dmitrij Frishman, Christian Gruber, Birgitta Geier, Dirk Haase, Andreas Kaps, Kai Lemcke, Gertrud Mannhaupt, Friedhelm Pfeiffer, C Schüller, et al. MIPS: a database for genomes and protein sequences. *Nucleic acids research*, 28(1):37–40, 2000.

[23] Takaya Saito and Marc Rehmsmeier. Precrec: fast and accurate precision–recall and roc curve calculations in r. *Bioinformatics*, 33(1):145–147, 2017.

# A Session info

Here is the output of `sessionInfo()` on the system that compiled this vignette:

- R version 4.6.1 (2026-06-24), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_GB`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Time zone: `America/New_York`
- TZcode source: `system (glibc)`
- Running under: `Ubuntu 24.04.5 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.24-bioc/R/lib/libRblas.so`
- LAPACK: `/usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0`
- Base packages: base, datasets, grDevices, graphics, methods, stats, utils
- Other packages: BiocStyle 2.42.0, diffuStats 1.34.0, ggplot2 4.0.4, ggsci 5.2.0, igraph 2.3.4, igraphdata 1.0.1
- Loaded via a namespace (and not attached): BiocManager 1.30.27, MASS 7.3-66, Matrix 1.7-6, R6 2.6.1, RColorBrewer 1.1-3, Rcpp 1.1.2, RcppArmadillo 15.8.0-1, RcppParallel 6.2.1, S7 1.0.0, backports 1.5.1, bookdown 0.48, bslib 0.12.0, cachem 1.1.0, checkmate 2.3.4, cli 3.6.6, compiler 4.6.1, data.table 1.18.6.1, dichromat 2.0-1, digest 0.6.39, dplyr 1.2.1, evaluate 1.0.5, expm 1.0-1, farver 2.1.2, fastmap 1.2.0, generics 0.1.4, glue 1.8.1, grid 4.6.1, gtable 0.3.6, htmltools 0.5.9, jquerylib 0.1.4, jsonlite 2.0.0, knitr 1.52, labeling 0.4.3, lattice 0.23-1, lifecycle 1.0.5, magick 2.9.1, magrittr 2.0.5, otel 0.3.0, pillar 1.11.1, pkgconfig 2.0.3, plyr 1.8.9, precrec 0.24.0, reshape2 1.4.5, rlang 1.3.0, rmarkdown 2.32, sass 0.5.0, scales 1.4.0, stringi 1.8.9, stringr 1.6.0, tibble 3.3.1, tidyselect 1.2.1, tinytex 0.61, tools 4.6.1, vctrs 0.7.3, withr 3.0.3, xfun 0.62, yaml 2.3.12