Systems biology

**RANKS**: a flexible tool for node label ranking and classification in biological networks:

Supplementary Information

Giorgio Valentini\(^1\)\(^*,\) Giuliano Armano\(^2\), Marco Frasca\(^1\), Jianyi Lin\(^1\), Marco Mesiti\(^1\) and Matteo Re\(^1\)

\(^{1}\)Anacletolab, Department of Computer Science, University of Milan, 20139 Milan, Italy and
\(^{2}\)Department of Electric and Electronic Engineering, University of Cagliari, 09129 Cagliari, Italy.

* To whom correspondence should be addressed.

Associate Editor: XXXXXXX

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Abstract

The Supplementary Information includes a detailed description of **RANKS – RAnking of Nodes with Kernelized Score functions**, and some examples of its application to the Gene Function Prediction, Drug repositioning and Human Phenotype Ontology prediction problems. All the examples include R code that shows how to use the **RANKS** package in the context of node label ranking problems. A final section explains how to expand the functionalities of the package through user defined score functions and kernels.

1 The **RANKS** method

Most of the node label ranking algorithms proposed for the analysis of biomolecular networks exploit local or global learning strategies to properly rank nodes [25, 22, 19, 5, 6]. In this context nodes represent usually genes or proteins, and edges functional similarities between nodes, according to the biological property under investigation.

**RANKS – RAnking of Nodes with Kernelized Score functions** is a very fast semi-supervised network method that combines local and global learning strategies to exploit both “local” similarities between nodes and “global” similarities embedded in the topology of the biomolecular network. From this standpoint **RANKS** can be considered a generalization of both guilt-by-association methods [20], and kernel based algorithms for semi-supervised network analysis [11].

Indeed, the guilt-by-association (GBA) approach [20] is generalized through fast and efficient local learning strategies based on an extended notion of functional distance between nodes. Global learning strategies are introduced by using kernel functions able to exploit the relationships and the overall topology of the underlying biological network. This approach can be also seen as a general algorithmic scheme: by introducing different local score functions and by choosing different kernels to model the similarity between nodes, we can derive different network-based algorithms. For instance, by adopting graph kernels [11], both direct and indirect relationships between genes can be exploited, thus taking into account the overall topology of the network.

Let \(G = (V, E)\) be an undirected graph with weighted adjacency matrix \(W\) representing a biomolecular network \(W\) (e.g. a gene or protein network), where \(V\) denotes the set of nodes (corresponding, for instance, to genes or proteins), and \(V_C \subseteq V\) denotes a subset of nodes having a specific property \(C\) (\(C\) could be, for instance, a Gene Ontology term, or a genetic disease). For the sake of simplicity, we can represent the nodes of the graph with natural numbers \(1, 2, \ldots, n\). Moreover a set of features \(\mathbf{x}_i \in X\) can be associated to a node \(i\). For instance, \(\mathbf{x}_i\) could represent the expression or the phylogenetic profile of a node \(i\), or whatever available data for a given gene/node \(i\).

The **node label ranking problem** consists of finding a score function \(S : V \rightarrow \mathbb{R}^+\) by which we can directly rank nodes according to their likelihood to belong to a specific category \(C\) (the higher the score, the higher the likelihood that a node belongs to \(C\)). It is worth noting that **node label ranking** can be seen as a “one-class” semi-supervised learning problem in biomolecular networks \(G\), since we can exploit the labeling of the known “positive” vertices \(v \in V_C\) belonging to the category \(C\), but also the similarity relationships between labeled and unlabeled vertices \(v \in V\).

In **RANKS** score functions are based on distance measures defined in a suitable Hilbert space \(\mathcal{H}\). More precisely, let \(\phi : X \rightarrow \mathcal{H}\) be a mapping to a given Reproducing Kernel Hilbert Space \(\mathcal{H}\), and \(K : X \times X \rightarrow \mathbb{R}\) its associated kernel function, such that \(< \phi(x), \phi(y) >_{\mathcal{H}} = K(x, y)\), where \(< \cdot, \cdot >_{\mathcal{H}}\) represents an internal product in \(\mathcal{H}\).
A distance measure $D(i, V_{C})$ in the Hilbert space between a given node $i$ and the set of nodes $V_{C}$ having a specific property $C$ can be thus introduced by using a proper kernel function [32]. Indeed, we can embed any valid kernel into the distance measure itself, thus resulting in a modular approach by which existing graph kernels, or in perspective new graph kernels properly designed for node label ranking problems in biomolecular networks, can be applied to rank nodes according to the property $C$ under study. By choosing different distance measures, diverse score functions can be derived. As examples we consider the following score functions, each one based on a different notion of distance between nodes:

1. Nearest Neighbours score
2. $K$-Nearest Neighbours score
3. Average score

### 1.1 Nearest-Neighbours Score

We can define a distance measure $D_{NN}(i, V_{C})$ of a vertex $i \in V$ w.r.t. to a set of nodes $V_{C}$, as the minimum squared distance in the Hilbert space between $i$ and $V_{C}$:

$$D_{NN}(i, V_{C}) = \min_{j \in V_{C}} \left\| \phi(x_{i}) - \phi(x_{j}) \right\|^2$$

(1)

By developing the square (1) we can obtain:

$$D_{NN}(i, V_{C}) = \min_{j \in V_{C}} \left\{ \phi(x_{i}) \cdot \phi(x_{j}) + \phi(x_{j}) \cdot \phi(x_{j}) - 2 \phi(x_{i}) \cdot \phi(x_{j}) \right\}$$

(2)

where $\langle \cdot, \cdot \rangle$ is the inner product in the feature space $H$. We can achieve a similarity measure by simply changing the sign of equation 2 and recalling that $\phi(x_{j}) \cdot \phi(x_{j}) = K(x_{j}, x_{j})$:

$$Sim_{NN}(i, V_{C}) = \min_{j \in V_{C}} \left\{ K(x_{i}, x_{j}) - 2K(x_{i}, x_{j}) + K(x_{j}, x_{j}) \right\}$$

(3)

If $K(x_{i}, x_{j})$ is equal for all $j \in V$, we can disregard these terms, thus achieving the nearest neighbours score $S_{NN}$:

$$S_{NN}(i, V_{C}) = \min_{j \in V_{C}} -2K(x_{i}, x_{j}) = 2 \max_{j \in V_{C}} K(x_{i}, x_{j})$$

(4)

### 1.2 $K$-Nearest-Neighbours Score

If we consider the $k$-nearest neighbours, i.e. $I_{k}(i) = \{ j \in V_{C} \}$ is ranked among the first $k$ nearest neighbours of $i$, then we can easily extend the $S_{NN}$ score by introducing the $k$-nearest neighbours distance:

$$D_{kNN}(i, V_{C}) = \sum_{j \in I_{k}(i)} \left\| \phi(x_{i}) - \phi(x_{j}) \right\|^2$$

(5)

By expanding the square in (5) and inverting the sign we can obtain a similarity measure:

$$Sim_{kNN}(i, V_{C}) = -\sum_{j \in I_{k}(i)} \left( K(x_{i}, x_{j}) - 2K(x_{i}, x_{j}) + K(x_{j}, x_{j}) \right)$$

(6)

This similarity measure can be directly used as a $k$-nearest neighbours score $S_{kNN}$, but in the case that $K(x_{i}, x_{j})$ is equal for all $j \in V$, we can disregard constant terms and hence obtain the simplification:

$$S_{kNN}(i, V_{C}) = 2 \sum_{j \in I_{k}(i)} K(x_{i}, x_{j})$$

(7)

### 1.3 Average Score

Another distance can be simply defined as the average distance in the Hilbert space $H$ between $i$ and the set of nodes $V_{C}$:

$$D_{AV}(i, V_{C}) = \left\| \phi(x_{i}) - \frac{1}{|V_{C}|} \sum_{j \in V_{C}} \phi(x_{j}) \right\|^2$$

(8)

By expanding the square (8) we obtain:

$$D_{AV}(i, V_{C}) = \left\langle \phi(x_{i}), \phi(x_{j}) \right\rangle > -\frac{1}{|V_{C}|} \sum_{j \in V_{C}} \left\langle \phi(x_{i}), \phi(x_{j}) \right\rangle$$

(9)

Also in this case a similarity measure can be obtained by changing the sign:

$$Sim_{AV}(i, V_{C}) = -K(x_{i}, x_{j}) + \frac{2}{|V_{C}|} \sum_{j \in V_{C}} K(x_{i}, x_{j})$$

(10)

We can finally obtain the average score $S_{AV}$, by observing that the third term of (10) is equal for all $i \in V$:

$$S_{AV}(i, V_{C}) = -K(x_{i}, x_{j}) + \frac{2}{|V_{C}|} \sum_{j \in V_{C}} K(x_{i}, x_{j})$$

(11)

By using the proposed kernelized score functions we can rank nodes with respect to their likelihood to belong to a given category $C$ simply by evaluating the selected kernel function. If the kernel matrix is computed in advance, the time complexity of $RANKS$ is $O(|V_{C}|^{2}|V|)$, that is approximately linear in respect to the number of nodes when $|V_{C}| << |V|$.

### 1.4 Random Walk kernels

We can obtain different node ranking algorithms by embedding different kernels in eq. 4, 7 and 11. In principle, any valid kernel can be used (e.g. linear, polynomial, gaussian, Laplacian, Cauchy and inverse multiquadric kernels), but in the context of biomolecular networks it is often meaningful to use a random walk kernel [34] constructed from the weighted adjacency matrix $W$ of the graph under study. Indeed it can capture not only relationships coming from direct neighborhoods between nodes, similarly to graph kernels, but also relationships coming from shared and more in general indirect neighborhoods between nodes. For instance, in the context of network-based Gene Function Prediction (GFP), while it is quite obvious that functional relationships are coded into direct neighborhoods, important functional relationships between genes can also be coded through indirect neighborhoods [3]. For instance, enzymes belonging to the same biological process may not share the same links, since their catalyzed reactions can be linked through other intermediate reactions belonging to the same pathway, but of course the involved enzymes do belong to the same biological process.

Random walk kernels represent the kernelized version of Markov Random Walks, by which random trajectories across graphs can be exploited to investigate the relationships between nodes and to score or label each node with respect to a specific property of the nodes [18].

The one-step random walk kernel matrix $K$ can be obtained from the adjacency matrix $W$ of the graph in the following way:

$$K = (d_{i} - 1)I + D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$$

(12)

where $D$ is the “degree” diagonal matrix with elements $d_{i} = \sum_{j} w_{ij}$, $I$ is the identity matrix and $\alpha$ is a value larger than 2. The $q$-step random
walk kernel can be directly obtained from (12):

\[ K^q = [(a-1)I + D^{-\frac{1}{2}}WD^{-\frac{1}{2}}]^q \]  

(13)

where \( q \geq 2 \) is an integer representing the number of steps of the random walk across the graph and can be easily computed by a recursive chain of matrix multiplications:

\[ K^q = K^{q-1}K \]  

(14)

Fig. 1 provides a derivation of the random walk kernel:

By setting \( q = 2 \), the random walks consider indirect neighbours, that is two nodes are similar if either they are directly connected or they share common nodes in their neighborhood. It is worth noting that if \( q = 1 \) we obtain the one-step random walk kernel, by which only the direct neighbours of each node are visited. More in general, by setting \( q \geq 2 \) two vertices are considered similar if they are directly connected or if they are indirectly connected through a path including from 1 to \( q - 1 \) intermediate vertices. Large values of \( q \) may introduce remote similarities between nodes, in a way similar to diffusion kernels [14]. It can be shown that (13) is up to scaling terms equivalent to a \( q \)-step random walk on the graph with random restarts, a well-known algorithm used for scoring web pages in the Google search engine [2].

2 Application examples

RANKS has been successfully applied to gene function prediction, gene disease prioritization and drug repositioning ([24]) problems, comparing favourably with state-of-the-art network based methods [27, 28, 38].

In this section we provide several application examples of RANKS, including the R code that shows how to use the RANKS R package in the context of node label ranking problems. More precisely, we at first consider an example of application to two classical prediction problems, Gene Function Prediction [26] and Drug repositioning [33]. Then we describe an application to the Human Phenotype Ontology prediction problem [13], a complex prediction task proposed in the recent Critical Assessment of Functional Annotation 2 (CAFA2) international challenge, where RANKS was one of the top ranked method in this specific prediction task (http://biofunctionprediction.org/node/8) [9].

The RANKS package is freely downloadable from CRAN (https://cran.r-project.org/) for Unix, Windows and Mac operating systems. The data used in these examples are downloadable from http://homes.di.unimi.it/valentini/DATA/RANKS. Detailed explanations about the syntax and the semantic of each function and method implemented in the package are available in the RANKS reference manual: (https://cran.r-project.org/web/packages/RANKS/RANKS.pdf).

2.1 Gene Function Prediction

Gene Function Prediction (GFP) can be formalized as a node label ranking problem, where nodes represent genes and edges relationships weighted according to the evidence of co-functionality implied by data sources [40]. By exploiting the labeling of a subset of genes annotated with a specific function, and the topology of the network, the prediction task consists of ranking unlabeled nodes with respect to the function under study [23, 4, 21].

In this section we present a simple application for ranking genes with respect to FunCat (Functional Catalogue) classes with the yeast model organism [31]. Here we limited the experiments to a relatively low number of genes and FunCat classes to allow to easily experiment with RANKS. We combined 6 bio-molecular data sets previously used for the related task of gene classification [35]. The data sets include pairwise sequence similarity data, protein-protein interaction, protein domain and gene expression data. We considered only yeast genes common to all data sets. Moreover, in order to get a not too small set of positive examples for training, for each data set we selected only the FunCat-annotated genes, and classes with at least 20 positive examples, using the HGgene R package [36]. This selection process yielded 1901 yeast genes annotated to 168 FunCat classes (see [27] for more details about the construction of the integrated network).

Let us now illustrate, step-by-step, how to use the package in an R script to predict gene functions in the yeast using 5-fold cross-validation, repeated 10 times, in order to evaluate the scores for each of the 1901 yeast genes relative to all the considered FunCat classes. At first we need to load the necessary packages:

```
library(RANKS);
library(PerMeas);
```

PerMeas is another package available from CRAN to measure the performance of supervised and semi-supervised learning methods (i.e. accuracy, AUC, etc). Then network data (in the form of a weighted adjacency matrix) and annotation data (another matrix \( T \), where \( T[i,j] = 1 \) if gene \( i \) is annotated to the FunCat category \( j \), otherwise \( T[i,j] = 0 \)) are loaded:

```
# loading network data
datafile <- "data/yeast.data.matrix.rda";
load(datafile); # avgEF.matrix loaded
N <- avgEF.matrix; # 1901 X 1901
re(avgEF.matrix);

# filtering negative values
N[NG]<-0;

# read labels matrix with gene annotations
labels.file <- "data/yeast.label.matrix.rda";
load(labels.file);
T <- Yeast.Funcat.Table.intersection.filtered2DormoreGenes;
rm(Yeast.Funcat.Table.intersection.filtered2DormoreGenes);
T <- as.matrix(T[-1]); # root category deleted
n.classes <- ncol(T);
n.genes <- nrow(T);
```

Our experiments build on annotations coded in the FunCat-2.1 scheme, and FunCat-2.1_data_20070316 data, available from the MIPS web site (http://mips.gsf.de/projects/functat).
classnames <- colnames(T);
Then the score matrix to collect the final scores and the random walk kernel are constructed using the \texttt{rw.kernel} \texttt{RANKS} method:

\begin{verbatim}
# construction of the matrix of scores.
SI <- matrix(numeric(nclasses * ngenes), ncol=nclasses);
rownames(SI) <- rownames(T);
colnames(SI) <- classnames;
# Construction of a 1-step RW kernel
RW <- rw.kernel(RW);
\end{verbatim}

Then the scores are computed by 5-fold cross-validation repeated 10 times by calling the \texttt{multiple.ker.score.cv} \texttt{RANKS} method. Note that the \texttt{RANKS} score function is used, considering \( k = 19 \) nearest neighbours.

\begin{verbatim}
for (i in 1:nclasses) {
    # 10-fold CV with 1 step RW kernel
    res <- multiple.ker.score.cv(RW, ind.pos, m=5, p=10,
        init.seed=4, fun=RW.score, k=19);
    SI[,i] <- res$av.scores;
    cat("Class ", i, ": ", classnames[i], "n;
}
\end{verbatim}

By simply changing the argument \texttt{fun} you can experiment with other scores, e.g. by setting \texttt{fun = eav.score} we can use the average score. Finally we can estimate the performance by computing the Area Under the ROC curve and precision at different levels of recall by calling the corresponding functions of the \texttt{PerfMeas} package:

\begin{verbatim}
# saving scores
save(SI, file="Results/Scores.RANKS.GFP.rda");
# Computing precision at different levels of recall
recall.levels <- seq(0,0.5, by=0.1);
res <- precision.at.multiple.recall.level.over.classes
    (T, SI, rec.levels = recall.levels);
# Computing AUC
auc <- AUC.single.over.classes(T, S1);
# saving results
save(auc, res, file="Results/AUC.PFR.RANKS.GFP.rda");
\end{verbatim}

\texttt{RANKS} is very fast: on a notebook with an Intel i7 2.20 GHz with 4 GB RAM the entire 5 fold cross validation procedure repeated 10 times for all the 168 considered FunCat classes required less than 50 seconds, including also the I/O and the computation of the performance measures.

Note that \texttt{RANKS} is basically a ranker, since it provides scores for each gene/node, but can also be used as a classifier through the \texttt{RANKS} functions \texttt{ker.classifier.classify}, \texttt{ker.classifier.classify.holdout} and \texttt{multiple.ker.score.threshold}, by which an \"optimal\" threshold is applied to the score to obtain the label associated to each node. The optimal threshold is obtained by internal cross-validation on training data (see the Reference Manual for more details).

Table 1 reports cross-validated results on the FunCat-yeast prediction task, stratified across the 5 levels of the FunCat taxonomy. In particular, level 1 is the highest level, which includes the most general terms, level 2 is less general, till to level 5, which includes the most specific FunCat categories. Results show that \texttt{RANKS} methods (the first 3 columns) are competitive with other state of the art network-based algorithm such as GeneMania \cite{23}, or the classical random walk and random walk with restart algorithm, as well as with inductive supervised algorithms such as Support Vector Machines.

### 2.2 Drug repositioning

Drug repositioning consists of predicting novel therapeutic indications for existing drugs. In this context nodes represent drugs and edges their structural or functional similarities. The task consists of scoring and ranking unlabeled nodes/drugs with respect to a given therapeutic indication starting from a small set of labeled drugs/nodes \cite{8, 33}.

Here we show how to predict the therapeutic category of drugs according to the annotations provided by DrugBank 3.0 \cite{12} using \texttt{RANKS}. We considered 51 Therapeutic Categories (TC) from DrugBank (Table 2), that is the TC having more than 15 drugs annotated. We constructed three drug networks involving 1253 drugs: the first one is based on the similarity of chemical structures according to their SMILES molecular fingerprint; the second is obtained through network projections between drugs and their molecular targets and the third one has been constructed by network projections of the chemical-chemical interactions in the STITCH database \cite{15}. Then we constructed three networks \texttt{U1}, \texttt{U2} and \texttt{U3}, by progressively integrating the first, the second and the third networks described above \cite{29} for more details about the construction of the drug networks.

In the rest of this section we show how to process data through cross-validation techniques using a simple call to the high-level function \texttt{do.RANKS} of the \texttt{RANKS} package.

After loading \texttt{RANKS} and \texttt{PerfMeas} packages, we call \texttt{do.RANKS} to perform a 5-fold cross validation (\( kk = 5 \)) using the average score (\texttt{score = eav.score}) and a 1-step random walk kernel (\texttt{kernel = rw.kernel, p = 1}) to score the drugs with respect to 51 TCS. The network data are in the directory \texttt{data} (\texttt{data.dir = "data"}) and the name of the file of the network data is \texttt{U1.rda}, corresponding to a network constructed by structural similarity between each pair of drugs. In the same way the directory where the labels of the drugs are stored is \texttt{data} (\texttt{labels.dir = "labels"}) and the file name of the drug labels is \texttt{"U1"} (\texttt{labels = "U1"}), where \( T[i, j] = 1 \) if drug \texttt{i} is annotated with the Therapeutic Category \texttt{j}, otherwise \( T[i, j] = 0 \). The corresponding results (computed scores, as well as AUC results and precision at different recall levels) are stored in the \texttt{Results} directory (\texttt{output.dir = "Results"}).

\begin{verbatim}
library(RANKS);
library(PerfMeas); do.RANKS(score=eav.score, kernel=rw.kernel, a=2, p=1,
    sparsify=TRUE, kk=5, rep=1, seed=0, data.dir="data","U1", labels.dir="U1");
\end{verbatim}

Note that this interface is conceived for batch experiments, where both input data and the results are automatically stored in \texttt{R} compressed \texttt{rda} files. In this way we can experiment with different score functions or kernels by only changing the arguments of the high-level function \texttt{do.RANKS}. In a similar way experiments with different data can be easily performed by only changing the input data themselves, just using only 1 line of code.

The following lines of code show an example of cross-validations with different score functions and kernels, and different input networks:

\begin{verbatim}
# The same task as above, but using a different network
U2, constructed by integrating molecular fingerprints
# of the drugs and drug-target interactions:
do.RANKS(score=eav.score, kernel=rw.kernel, a=2, p=1, sparsify=TRUE, kk=5, rep=1, seed=0, data.dir="data", labels.dir="data", output.dir="Results", data.dir="U2", labels.dir="U2");
# Here also chemical-chemical interaction form STITCH
# have been integrated (data="U3"):
do.RANKS(score=eav.score, kernel=rw.kernel, a=2, p=1, sparsify=TRUE, kk=5, rep=1, seed=0, data.dir="data", labels.dir="data", output.dir="Results", data.dir="U3", labels.dir="U3");
\end{verbatim}

A different score function, i.e.

- Weighted Sum Linear Decay was used with the same data:
  \begin{verbatim}
do.RANKS(score=MELD.score, kernel=rw.kernel, a=2, p=1, sparsify=TRUE, kk=5, rep=1, seed=0,\end{verbatim}
Table 1. Average AUC at the five levels of the FunCat taxonomy. Numbers in boldface refer to the best results for a given level. $S_{KNN}$ stands for K-nearest-neighbours score, $S_{AV}$ stands for average score, $S_{NN}$ nearest-neighbours score. $GM$ GeneMania, $RW$ random walk with restart, $RW^2$ random walk till to convergence, $RW^2*5$ random walk limited to 2 steps, $GBA$ guilt-by-assocation, and $SV M$ is a linear Support Vector Machine.

<table>
<thead>
<tr>
<th>Level</th>
<th>$S_{KNN}$</th>
<th>$S_{AV}$</th>
<th>$S_{NN}$</th>
<th>$GM$</th>
<th>$RW$</th>
<th>$RW^2$</th>
<th>$RW^2*5$</th>
<th>$GBA$</th>
<th>$SV M$</th>
<th>n.class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.8206</td>
<td>0.8909</td>
<td>0.7920</td>
<td>0.7466</td>
<td>0.5194</td>
<td>0.7686</td>
<td>0.7006</td>
<td>0.8782</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.8620</td>
<td>0.8335</td>
<td>0.8424</td>
<td>0.8379</td>
<td>0.8071</td>
<td>0.7492</td>
<td>0.7649</td>
<td>0.7849</td>
<td>49</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.9086</td>
<td>0.8983</td>
<td>0.8874</td>
<td>0.8807</td>
<td>0.8614</td>
<td>0.7364</td>
<td>0.7712</td>
<td>0.8279</td>
<td>65</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.9052</td>
<td>0.9030</td>
<td>0.8880</td>
<td>0.8866</td>
<td>0.8717</td>
<td>0.5210</td>
<td>0.7858</td>
<td>0.8419</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.9206</td>
<td>0.9206</td>
<td>0.9164</td>
<td>0.9087</td>
<td>0.9064</td>
<td>0.5733</td>
<td>0.8173</td>
<td>0.8682</td>
<td>495</td>
<td></td>
</tr>
</tbody>
</table>

The same experiment with the $S_{KNN}$ score but using a leave-one-out technique to assess the generalization performances could be in principle attained by using the same function do.RANKS and by setting $kk=1253$ (i.e. the total number of nodes/drugs), but a more efficient version is implemented in the RANKS package through the function do.loo.RANKS:

<table>
<thead>
<tr>
<th>Level</th>
<th>$S_{KNN}$</th>
<th>$S_{AV}$</th>
<th>$S_{NN}$</th>
<th>$GM$</th>
<th>$RW$</th>
<th>$RW^2$</th>
<th>$RW^2*5$</th>
<th>$GBA$</th>
<th>$SV M$</th>
<th>n.class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9276313</td>
<td>0.8782</td>
<td>0.8535</td>
<td>0.7649</td>
<td>0.9064</td>
<td>0.7466</td>
<td>0.8419</td>
<td>0.7246</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0.9276313</td>
<td>0.8782</td>
<td>0.8535</td>
<td>0.7649</td>
<td>0.9064</td>
<td>0.7466</td>
<td>0.8419</td>
<td>0.7246</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0.9276313</td>
<td>0.8782</td>
<td>0.8535</td>
<td>0.7649</td>
<td>0.9064</td>
<td>0.7466</td>
<td>0.8419</td>
<td>0.7246</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0.9276313</td>
<td>0.8782</td>
<td>0.8535</td>
<td>0.7649</td>
<td>0.9064</td>
<td>0.7466</td>
<td>0.8419</td>
<td>0.7246</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0.9276313</td>
<td>0.8782</td>
<td>0.8535</td>
<td>0.7649</td>
<td>0.9064</td>
<td>0.7466</td>
<td>0.8419</td>
<td>0.7246</td>
<td>30</td>
<td></td>
</tr>
</tbody>
</table>

automatically generates in the directory Results the files:

- Scores.RW.5step.fTRUE.U3.T.rda: Score results
- AUC.RNK.score19.p2.a5.fTRUE.U3.T.rda: AUC results
- PXR.RNK.score19.p2.a5.fTRUE.U3.T.rda: Precision/recall

For instance by looking at AUC results:

- load("AUC.RNK.score19.p2.a5.fTRUE.U3.T.rda")
- AUCsave: AUC averaged across the 51 TC categories
- AUCper.class: per class results
- only the first 10 visualized:
  - Adrenergic_Agents
  - Anti.Allergic_Agents
  - Anti.Bacterial_Agents
  - Anti.Carbohydrate_Agents
  - Anti.Infections_Agents
  - Anti.Inflammatory_Agents
  - Anti.Innervation_Agents
  - Anti.IVX_Agents
  - Anti.MVX_Agents
  - Anti.Pathogenic_Agents

2.3 Human Phenotype Ontology prediction

The Human Phenotype Ontology (HPO) project [30] provides a comprehensive and well-structured set of more than 10000 terms (classes) that represent human phenotypic abnormalities annotated to more than 7000 hereditary syndromes listed in OMIM, Orphanet and DECIPHER databases [1]. An important computational task is represented by the prediction or ranking of genes with respect to HPO terms [10, 37]. HPO prediction has been recently proposed in the international CAF2 challenge, and RANKS was one of the top-scoring methods participating to the challenge.

Here we show an application of RANKS to HPO term ranking using integrated networks of human genes obtained from two previous
Table 2: DrugBank Therapeutic Categories (TC) with more than 15 drugs considered in the experiments. The first column reports the abbreviated name, the second the full DrugBank name and the third the cardinality of the TC.

<table>
<thead>
<tr>
<th>Abbreviated name</th>
<th>Full DrugBank name name</th>
<th>Card.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aden.A.</td>
<td>Adrenergic_Agents</td>
<td>25</td>
</tr>
<tr>
<td>Aden.In.</td>
<td>Adrenergic_Uptake_Inhibitors</td>
<td>20</td>
</tr>
<tr>
<td>Adren.</td>
<td>Strategic_alpha_Agonists</td>
<td>27</td>
</tr>
<tr>
<td>Adren.b.</td>
<td>Adrenergic_beta_Antagonists</td>
<td>25</td>
</tr>
<tr>
<td>Analges.</td>
<td>Analgesics</td>
<td>40</td>
</tr>
<tr>
<td>Anti.Alv.</td>
<td>Anti-Alzheimer_Agents</td>
<td>32</td>
</tr>
<tr>
<td>Anti.Aub.</td>
<td>Anti-Anxiety_Agents</td>
<td>42</td>
</tr>
<tr>
<td>Anti.Bac.</td>
<td>Anti-Bacterial_Agents</td>
<td>103</td>
</tr>
<tr>
<td>Anti.HIV.</td>
<td>Anti-HIV_Agents</td>
<td>22</td>
</tr>
<tr>
<td>Anti.Inf.</td>
<td>Anti-Infective_Agents</td>
<td>20</td>
</tr>
<tr>
<td>Anti.Inf.</td>
<td>Anti-Infectives</td>
<td>99</td>
</tr>
<tr>
<td>Anti.Uker</td>
<td>Anti.Ute_Agents</td>
<td>19</td>
</tr>
<tr>
<td>Anti.ux.</td>
<td>Anti.Anxiety_Agents</td>
<td>35</td>
</tr>
<tr>
<td>Anti.infl.</td>
<td>Anti-Inflammatory_Agents</td>
<td>69</td>
</tr>
<tr>
<td>Anti.Ant.</td>
<td>Anti-Behavioral_Agents</td>
<td>26</td>
</tr>
<tr>
<td>Anti.Osv.</td>
<td>Anti-Oxidants</td>
<td>86</td>
</tr>
<tr>
<td>Antiost.</td>
<td>Anti-Osteo_Agents</td>
<td>23</td>
</tr>
<tr>
<td>Antimet.</td>
<td>Antimetabolites</td>
<td>34</td>
</tr>
<tr>
<td>Antifungal</td>
<td>Antifungal_Agents</td>
<td>22</td>
</tr>
<tr>
<td>Antidiab.</td>
<td>Antidiabetic_Agents</td>
<td>24</td>
</tr>
<tr>
<td>Antihypert.</td>
<td>Antihypertensive_Agents</td>
<td>105</td>
</tr>
<tr>
<td>Anto metab.</td>
<td>Antimetabolites</td>
<td>25</td>
</tr>
<tr>
<td>Antineopl.</td>
<td>Antineoplastic_Agents</td>
<td>66</td>
</tr>
<tr>
<td>Antineopl.H</td>
<td>Antineoplastic_Agents_Horizontal</td>
<td>27</td>
</tr>
<tr>
<td>Antipark.</td>
<td>Antiparkinson_Agents</td>
<td>27</td>
</tr>
<tr>
<td>Antip.</td>
<td>Antipsychotics</td>
<td>39</td>
</tr>
<tr>
<td>Antipyc.</td>
<td>Antipsychotics</td>
<td>27</td>
</tr>
<tr>
<td>Antirenal.</td>
<td>Antirenal_Agents</td>
<td>26</td>
</tr>
<tr>
<td>Bronchodil.</td>
<td>Bronchodilator_Agents</td>
<td>13</td>
</tr>
<tr>
<td>CaCh.Block.</td>
<td>Calcium_Channel_Blockers</td>
<td>22</td>
</tr>
<tr>
<td>Cephal.</td>
<td>Cephalosporins</td>
<td>30</td>
</tr>
<tr>
<td>Cyclooxy.</td>
<td>Cyclooxygenase_Inhibitors</td>
<td>24</td>
</tr>
<tr>
<td>Dietary.sup.</td>
<td>Dietary_supplement</td>
<td>34</td>
</tr>
<tr>
<td>Diuretics.</td>
<td>Diuretics</td>
<td>23</td>
</tr>
<tr>
<td>Dopam. Ant.</td>
<td>Dopamine_Antagonists</td>
<td>20</td>
</tr>
<tr>
<td>Enzyme Inf.</td>
<td>Enzyme_Inhibitors</td>
<td>64</td>
</tr>
<tr>
<td>GABA Med.</td>
<td>GABA_Mediations</td>
<td>28</td>
</tr>
<tr>
<td>Glucocort.</td>
<td>Glucocorticoids</td>
<td>21</td>
</tr>
<tr>
<td>Hist.H1 Ant.</td>
<td>Histamine_H1_Antagonists</td>
<td>28</td>
</tr>
<tr>
<td>Hypost.Sed.</td>
<td>Hypothesis_and_Sedatives</td>
<td>41</td>
</tr>
<tr>
<td>Hypogly.</td>
<td>Hypoglycemic_Agents</td>
<td>22</td>
</tr>
<tr>
<td>Immunosup.</td>
<td>Immunosuppressive_Agents</td>
<td>30</td>
</tr>
<tr>
<td>Macromot.</td>
<td>Macromotors</td>
<td>45</td>
</tr>
<tr>
<td>Muc.Ant.</td>
<td>Mucamin_Antagonists</td>
<td>23</td>
</tr>
<tr>
<td>Narcotics</td>
<td>Narcotics</td>
<td>22</td>
</tr>
<tr>
<td>Penicillins</td>
<td>Penicillins</td>
<td>30</td>
</tr>
<tr>
<td>Sympathol.</td>
<td>Sympathomimetics</td>
<td>24</td>
</tr>
<tr>
<td>Sympathomim.</td>
<td>Sympathomimetics</td>
<td>42</td>
</tr>
<tr>
<td>Vasodilator</td>
<td>Vasodilator_Agents</td>
<td>25</td>
</tr>
<tr>
<td>Vasodilator</td>
<td>Vasodilator_Agents</td>
<td>55</td>
</tr>
</tbody>
</table>

Studies [41, 17]. The Functional Interaction (FI) network [41] has been constructed by using functional interactions predicted by a Naive Bayes classifier trained on pairwise relationships extracted from Reactome [59] and other curated pathways databases, and from uncurated pairwise relationships derived from physical protein-protein interactions (PPI) in human and other species, from gene co-expression data, proteins domain-domain interactions, protein interactions obtained via biomedical text mining, and Gene Ontology annotations. The network presented in [17] integrates diverse lines of evidence in order to produce a functional human gene network (HumanNet) that has then been used in several tests to predict causal genes for human diseases and to increase the power of genome-wide association studies. The most significant difference between HumanNet and FI networks consists of including in the former functional interactions borrowed from other species (yeast, fly and worm) through comparative genomics techniques.

As an example of application of RANKS we integrated the two networks through a simple unweighted integration method, thus obtaining a net with 16505 nodes (genes) and about one million and two hundred thousands edges. We randomly selected 100 HPO terms having more than 20 annotated genes and applied RANKS to HPO term ranking and prediction. Please, note that this task requires a computer with 16 GB RAM to be comfortably executed (other machines with less available memory can be used, but in this case it is likely that a certain computational burden will occur, due to memory swapping problems).

After loading the RANKS and PerfMera packages, we load the HPO annotations, randomly selecting 100 HPO terms having more than 20 annotated genes:

```r
# the network data (net) and the corresponding labels are automatically stored in .rda files in the current directory:
net = "hponet-finet.UA.net";
labels <- hpo.hnnet.finet.ann.20;
load("data/hpo.hnnet.finet.ann.rda");
# elimination of terms having less than 20 annotations
x <- apply(hpo.hnnet.finet.ann,2,sum);
y <- which (x<=20);
5066 terms removed, 1464 terms remained
hpo.hnnet.finet.ann.20 <- hpo.hnnet.finet.ann[y,-y];
# random selection of 100 HPO terms
n <- ncol(hpo.hnnet.finet.ann.20);
selected <- sample(1:n, 100);
labels.sel <- hpo.selected.labels;
save(hpo.hnnet.finet.ann.20[selected],
    file="data/hpo.selected.labels.rda");
labels.sel <- hpo.selected.labels;
```

With a single call to the high level function do.loo.RANKS we can perform a leave-one-out run on the entire network using the average score and a 1-step random walk kernel; scores, AUC and precision/recall results are automatically stored in .rda files in the current directory:

```r
# RM kernel with 1 step
output-dir <- "/r/";
labels.sel <- hpo.selected.labels;
do.los.RANKS(score=wv.score, compute.kernel=TRUE, kernel=wn.kernel, a=2, p=1, sparsify=TRUE, data=net, labels=labels.sel, output-dir=output-dir, output.name="hpo.sel");
```

The same task, using different combination of kernels, can be easily performed with the following lines of code:

```r
# RM kernel with 3 steps
output-dir
labels.sel
kernel-linear.kernel, p=2, sparsify=TRUE, data=net, labels=labels.sel, output-dir=output-dir, output.name="hpo.sel");
```

```
```
3 User defined score functions and kernels

One of the strengths of the RANKS is its modularity: it offers an algorithmic scheme where the specific choice of a score function and a kernel leads to a different semi-supervised learning algorithm (Section 1).

RANKS offers a set of different score functions implemented in the package: Nearest-Neighbour (\(S_N\)), K-Nearest-Neighbour (\(S_{KNN}\)) and average (\(S_{AV}\)) score – Section 1), as well as Weighted Sum with Linear Decay (\(S_{AV,WSLD}\)), which represents a generalization of the score introduced in [16]. Moreover several kernels are just implemented in the library, such as the Cauchy, the Laplacian, the Gaussian, the inverse multiquadric, the linear and Polynomial and the random walk kernels (Section 1.4). In addition, a user can easily devise and implement her/his own score functions and kernels, which can be used by any high-level function of the software library, such as do.RANKS, as arguments to any high level function of the software library, such as do.RANKS.

Once a new score function and/or a new kernel function have been defined as specified above, they can be used in any high-level RANKS functions where a generic score or kernel function can be used. For instance, if we consider the high level function do.RANKS, to perform a 5-fold cross-validation with the network “net.rda” and the labels “labels.rda”, both stored in the directory “data”, one should call My.score and My.kernel as follows:

```r
my.kernel <- function(N, a) { ... }  
my.kernel <- function(K, x, x.pos) { ... }
```

where

- \(K\): numeric matrix.
- \(N\): numeric vector. Rows are examples and columns are features.
- \(a\): parameter for the kernel.

My.score and My.kernel should return a vector embodying the scores of all nodes of the network. For the sake of clarity, let us provide the declaration of the method for single.My.score:

```r
# generic function
setGeneric("single.My.score");
# method for class graph
setMethod("single.My.score", signature(K="graph"), function(K, x, x.pos) [...]);
# method for class graph - package graph
setMethod("single.My.score", signature(K="graph"), function(K, x, x.pos) [...]);
```

whose arguments are:

- \(K\): a matrix. It must be a kernel matrix or a symmetric matrix expressing the similarity between nodes.
- \(x\): integer. Index corresponding to the element of the \(K\) matrix for which the score must be computed.
- \(x.pos\): vector of integer. Indices of the positive elements of the \(K\) matrix.

The single.My.Score is expected to return the score of a single node, while My.score should return a vector embodying the scores of all nodes of the network.

Adding novel kernel functions is even easier. For instance a user-defined kernel My.kernel can be added by implementing a function as follows:

```r
my.kernel <- function(W, a) { ... }
```

where

- \(W\): numeric matrix.
- \(a\): parameter for the kernel.

(If necessary other kernel parameters can be added.)

My.kernel should return a square symmetric kernel matrix representing the similarities between the examples (rows of \(W\)), as specified in the My.kernel function.

As already pointed out, custom score functions and kernels can be passed as arguments to any high level function of the software library, such as do.RANKS or do.loo.RANKS.

The following additional example considers a hold-out classification of a kernel matrix \(K\) using the user-defined My.score function:

```r
do.loo.RANKS(score = My.score, kernel = My.kernel, kk = 5,  
data = "data", labels = "labels");  
data.dir = "data/", labels.dir = "labels/",  
data = "results/", data = "net", labels = "labels")
```

The following additional example considers a hold-out classification of a kernel matrix \(K\) using the user-defined My.score function:

```r
ker.score.classifier.holdout(K, ind.pos, ind.test,  
fun = My.score)
```

As already pointed out, custom score functions and kernels can be passed as arguments to any high level function of the software library, such as do.RANKS or do.loo.RANKS.

```
References


The distribution of the AUC results across the 100 considered HPO terms, for the identity (Id), linear (Lin) and random walk 1-step (RW1) and random walk 3 steps (RW3) kernels are summarized in Fig. 2.