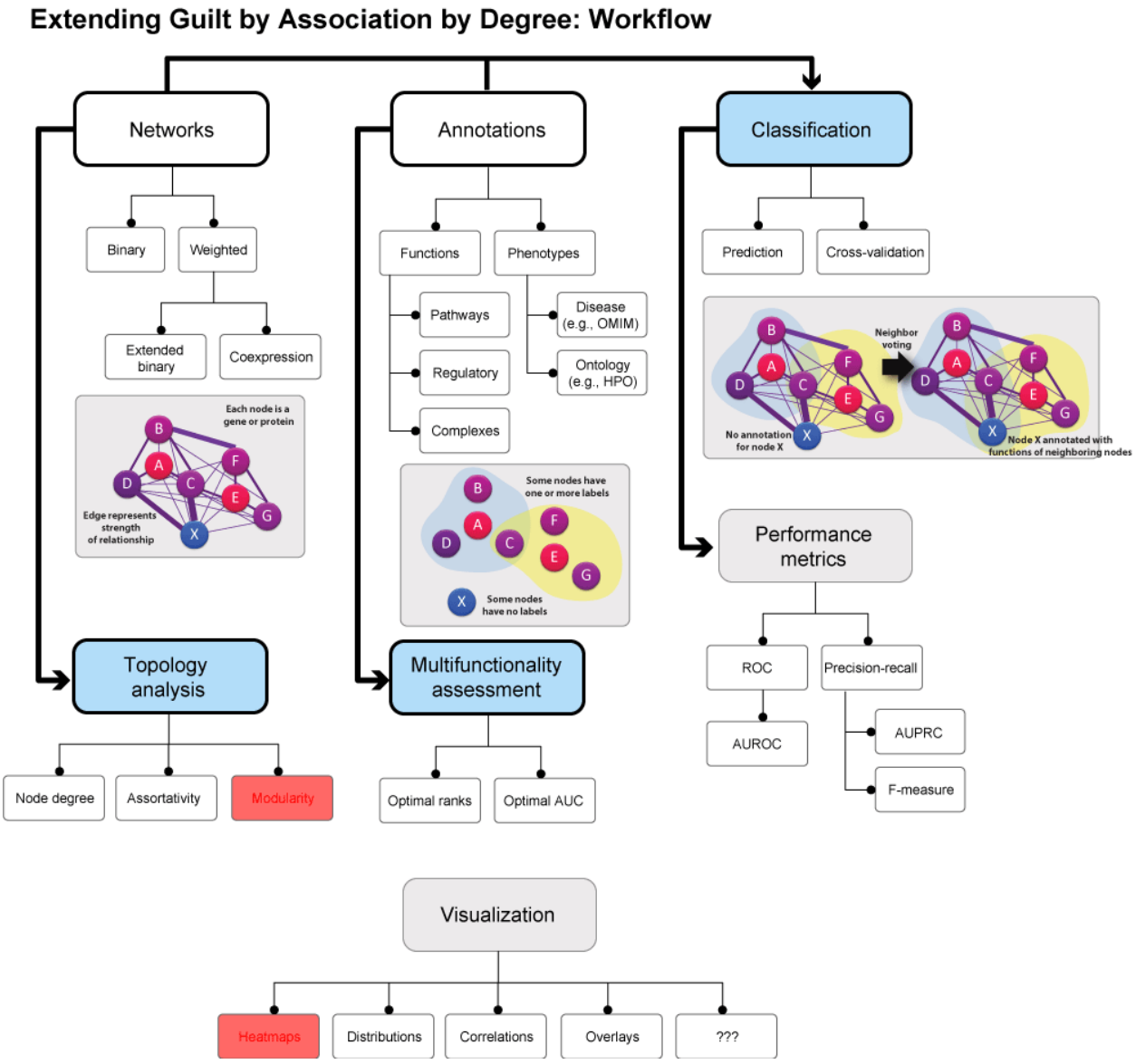
1 Quick tutorial

With the **EGAD** (**E**xtending '**G**uilt-by-**A**ssociation' by **D**egree) package we present a series of highly efficient tools to calculate functional properties in networks based on the guilt-by-association principle (Gillis and Pavlidis 2011, Gillis and Pavlidis 2011). Our basic toolset is inspired by an observation made repeatedly in machine learning; simple methods are often surprisingly efficacious (Hand 2006).



1.1 Using EGAD to evaluate protein-protein interactions with GO

This is a quick analysis for evaluating a protein-protein interaction network using gene labels from GO.

1.1.1 Prepare data

We begin first by loading in the data. Both *biogrid.rda* and *GO.human.rda* are provided as an example in this R package, but you can use other data interchangeably.

For the network, load the set of interacting pairs from BIOGRID:

```
load("~/data/biogrid.rda") ###
```

and load the ontology you want to use (e.g. Gene Ontology)

```
load("~/data/GO.human.rda") ###
```

1.1.2 Build interaction network (e.g., gene-by-gene matrix)

Using the pairs of genes, extract all the genes into a gene list. Then build the network using the gene pairs and the gene list.

```
genelist <- make_genelist(biogrid)
gene_network <- make_gene_network(biogrid,genelist)
```

```
> genelist[1:10]
 [1] "6416"  "84665" "90"    "2624"  "6118"  "375"   "377"   "54464"
"351"
[10] "333"
```

```
> gene_network[1:10,1:10]
```

```
      6416 84665 90 2624 6118 375 377 54464 351 333
6416     1     0  0    0    0   0   0     0   1   0
84665    0     1  0    0    0   0   0     0   0   0
90       0     0  1    0    0   0   0     0   0   0
2624     0     0  0    1    0   0   0     0   0   0
6118     0     0  0    0    1   0   0     1   1   0
375      0     0  0    0    0   1   1     0   0   0
377      0     0  0    0    0   1   1     0   0   0
54464    0     0  0    0    1   0   0     1   0   0
351      1     0  0    0    1   0   0     0   1   0
333      0     0  0    0    0   0   0     0   0   1
```

1.1.3 Build annotation matrix

The next step is to build the annotations matrix, using the set of genes in common with the network (genelist), and the subset of annotation terms (goterms).

```
annotations <- make_annotations(GO.human[,c(2,3)],genelist,goterms)
```

This matrix should be a sparse matrix, with a 1 if the gene (row) is annotated to the term (column).

```
> annotations[1:10, 1:6]
      GO:0000002 GO:0000003 GO:0000009 GO:0000010 GO:0000012
GO:0000014
6416           0          0          0          0          0
0
84665          0          0          0          0          0
0
90             0          1          0          0          0
0
2624           0          0          0          0          0
0
6118           0          0          0          0          0
0
375            0          0          0          0          0
0
377            0          0          0          0          0
0
54464          0          0          0          0          0
0
351            0          1          0          0          0
0
333            0          0          0          0          0
0
```

1.1.4 Run neighbor-voting algorithm

Using the binary gene network from biogrid and the GO annotations as input, we can run the neighbor voting algorithm.

```
GO_groups_voted <- run_GBA(gene_network, annotations)
```

1.1.5 Run multifunctionality assessment for the ontology used

```
GO_multifunc_assessment <- calculate_multifunc(annotations)
```

1.1.6 Create optimal lists

Once the scores of the genes are calculated, we can use this score as the values for the predictions.

for genes

```
ord <- order(as.numeric(GO_multifunc_assessment[,2]),decreasing=T)
GO_multifunc_assessment_s <- GO_multifunc_assessment[ord,c(1,2)]
optimallist_genes <- as.vector(unlist(GO_multifunc_assessment_s[,1]))
```

for GO groups

```
ord <- order(as.numeric(GO_groups_voted[[1]][,2]),decreasing=T)
GO_groups_voted_s <- GO_groups_voted[[1]][ord,]
optimallist_GO <- cbind(GO.term=rownames(GO_groups_voted_s), node.degre
e=GO_groups_voted_s[,2], roc=GO_groups_voted_s[,1])
```

1.1.7 Use optimal lists to determine multifunctionality AUC

```
AUC <- auc_multifunc(annotations, optimallist_genes)
```

1.1.8 Visualizing results

We give examples for visualizing results for two commonly used ontologies (human): The Gene Ontology 'GO' (*GO.human.Rdata*) and Phenocarta (*Pheno.human.Rdata*).

1.1.8.1 Gene Ontology

To visualize the distribution of neighbor-voting AUC scores over all GO groups (from running *run_GBA*), use

```
plot_distribution(auc_GO_nv, xlab=" Neighbor Voting AUC ",ylab="# of fu
nctional terms", b=30, xlim=c(0.4,1), ylim=c(0, 440), col="gray64", den
sity=F, avg=F, bars=T)
```

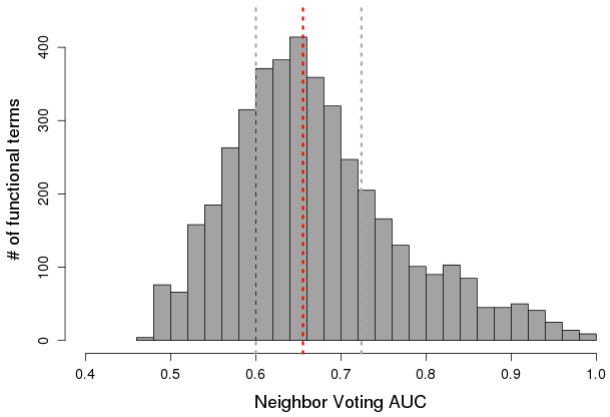


Figure 1: Distribution of AUC scores for neighbor voting using GO

This code also plots the statistical properties of the distribution (red: median, grey: inter quartile ranges) and incorporates them in the figure.

Similarly, the distribution of multifunctionality AUCs (use *calculate_multifunc* and *auc_multifunc*) can be visualized running

```
plot_distribution(auc_GO_mf, xlab="Optimal GO Ranking AUC",ylab="# of f
unctional terms", b=20, xlim=c(0.2,1), ylim=c(0,4400), col="gray64", de
nsity=F, avg=F, bars=T)
```

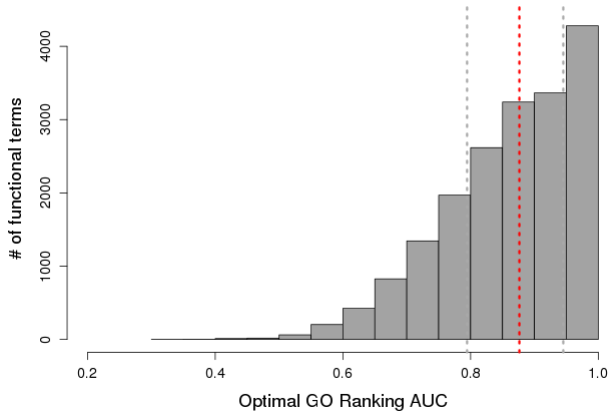


Figure 2: Distribution of AUC scores for multifunctionality assessment using GO.

To analyze node degree biases it is useful to evaluate the correlation between node degree rank and the rank in terms of the number of annotated GO groups per gene (use *run_GBA* and create and optimal ranked list as shown above for node degree rank and *calculate_multifunc* GO term rank or see example below):

```
# determine (scaled) ranks
numbers <- rowSums(annotations)
ord_labels <- order(numbers,decreasing=F)

node_degree <- node_degree(gene_network)
ord_nodes <- order(node_degree,decreasing=F)
ord_nodes <- ord_nodes/max(ord_nodes)

# # plot correlation, define bins (size: 200)
conv_smoother(ord_labels, ord_nodes, 200, "# of functional terms ranked
",”Node degree rank")
```

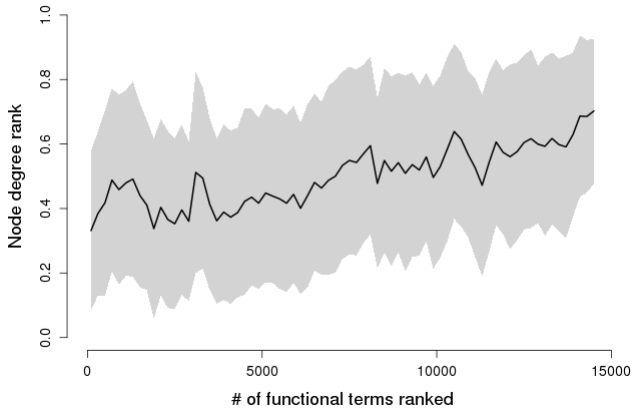


Figure 3: Relation between node degree and annotation based ranking using GO.

1.1.8.2 Phenocarta

In this manual we walked the user through a complete analysis using GO. The syntax can be used for any gene annotations including the commonly used *Phenocarta*. Here we provide visualizations of the results when running the demonstrated analysis in Phenocarta.

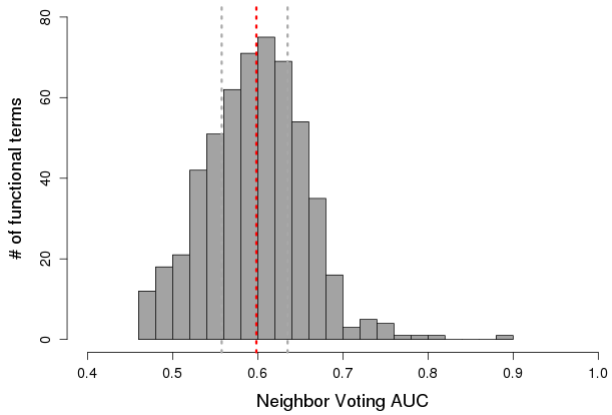


Figure 4: Distribution of AUC scores for neighbor voting using Phenocarta.

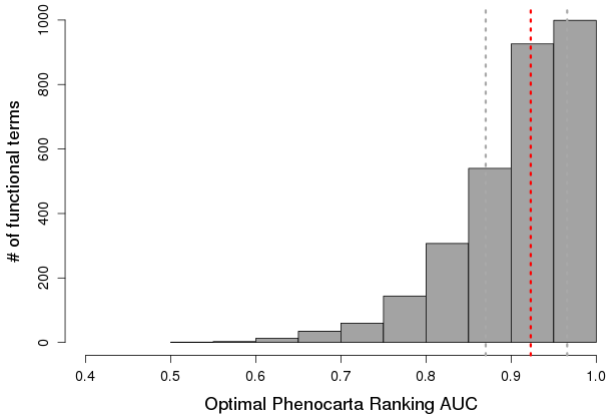


Figure 5: Distribution of AUC scores for multifunctionality assessment using Phenocarta.

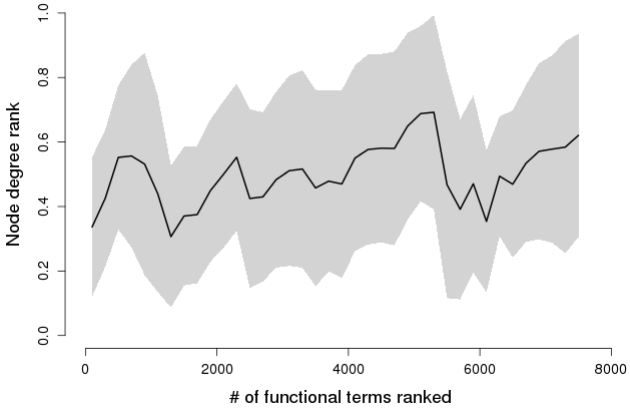


Figure 6: Correlation between node degree and annotation based ranking using Phenocarta.

1.2 Using EGAD to generate and analyze dense co-expression networks

1.2.1 Prepare data

We begin by loading in the data.

```
load("~/9606.gene_ids.rda")
```

And only selecting protein coding genes with entrez IDs:

```
genelist <- unique( attr$entrezID[attr$type == "protein_coding" & !is.na(attr$entrezID)] )
```

We can generate a co-expression network from publically available data.

For the network, get the expression data set.

```
exprs <- get_expression_data(GEOID="GSE3XXXX")
```

If this doesn't work as some files are missing, use a method you have to extract data. ExpressionSets from the biobase package can be used here instead.

1.2.2 Build co-expression network

```
network <- build_coexp_network(exprs, genelist)
```

Or if using the expressionSet object:

```
network <- build_coexp_expressionSet(exprsSet, genelist)
```

The default parameters are to use the spearman correlation and then to rank and standardize the network.

1.2.3 Calculating and visualizing network topological properties

We can calculate the assortativity of the network through:

```
assort <- assortativity(network)
```

A useful topological property would be the node degrees of the genes.

```
nd <- node_degrees(network)
```

We can visualize the node degree distribution by plotting the density using:

```
plot_density(nd)
```

1.2.4 Run neighbor-voting algorithm

As in the previous example, we need a gene annotations matrix.

```
annotations <- make_annotations(GO.human[,c(2,3)],genelist,goterms)
```

In this case, if we want to use sets between 20 and 300, we would filter the annotations object:

```
annotations_sub <- filter_network_cols(annotations, min=20, max=300)

GO_groups_voted <- run_GBA(network, annotations_sub)
```