PPInfer: Inferring functionally related proteins using protein interaction networks

Dongmin Jung, Xijin Ge

April 7, 2020

1 Introduction

Interactions between proteins occur in many, if not most, biological processes. Most proteins perform their functions in networks associated with other proteins and other biomolecules. This fact has motivated the development of a variety of experimental methods for the identification of protein interactions. This variety has in turn ushered in the development of numerous different computational approaches for modeling and predicting protein interactions. Sometimes an experiment is aimed at identifying proteins closely related to some interesting proteins. A network-based statistical learning method is used to infer the putative functions of proteins from the known functions of its neighboring proteins on a PPI network. This package identifies such proteins often involved in the same or similar biological functions.

2 Graph

Graph data is ubiquitous and graph mining is the study that aims to discover novel and insightful knowledge from data that is represented as a graph. Graph mining differs from traditional data mining in a number of critical ways. For example, the topic of classification in data mining is often introduced in relation to vector data; however, these techniques are often unsuitable when applied to graphs, which require an entirely different approach such as the use of graph kernels (Samatova *et al.*, 2013).

A support vector machine only applies to datasets in the real space. Often, however, we want to use a SVM on a dataset that is not a subset of the real space. This occurs in the case of biology and chemistry problems to describe our data. Fortunately, there is a ready solution to this problem, formalized in the use of kernel functions (Werther & Seitz, 2008). We employ the kernel support vector machine (KSVM) based on the regularized Laplacian matrix (Smola & Kondor, 2003) for a graph. The kernel matrix K can now be used with a classification algorithm for predicting the class of vertices in the given dataset,

$$K = (I + \gamma L)^{-1},$$

where K is $N \times N$, I is an identity matrix, L is the normalized Laplacian matrix, and γ is an appropriate decay constant. The decay constant is typically regarded as an arbitrary constant that is less than one.

3 Support Vector Machine

We focus on the application of computational method using a support vector machine. Suppose we have a dataset in the real space and that each point in our dataset has a corresponding class label. Our goal is to separate the points in our dataset according to their class label. A SVM is a linear binary classifier. The idea behind nonlinear SVM is to find an optimal separating hyperplane in high-dimensional feature space just as we did for the linear SVM in original space. At the heart of kernel methods is the notion of a kernel function. Broadly speaking, kernels can be thought of as functions that produce similarity matrices (Kolaczyk & Csardi, 2014). One of the advantages of support vector machines is that we can improve performance by properly selecting kernels. In most applications, RBF kernels are widely used but kernels suited for specific applications are developed. Here, we select the graph kernel K for PPI.

Data in many biological problems are often compounded by imbalanced class distribution, known as the imbalanced data problem, in which the size of one class is significantly larger than that of the other class. Many classification algorithms such as a SVM are sensitive to data with imbalanced class distribution, and result in a suboptimal classification. It is desirable to compensate the imbalance effect in model training for more accurate classification. One possible solution to the imbalanced data problem is to use one-class SVMs by learning from the target class only, instead of traditional binary SVMs. In one-class classification, it is assumed that only information of one of the classes, the target class, is available, and no information is available from the other class, known as the background. The task of one-class classification is to define a boundary around the target class such that it accepts as much of the targets as possible and excludes the outliers as much as possible (Ma, 2014).

However, one-class classifiers seldom outperform two-class classifiers when the data from two class are available (Ma, 2014). So the OCSVM and classical SVM are sequentially used in this package. First, we apply the OCSVM by training a one-class classifier using the data from the known class only. Let n be the number of proteins in the target class. This model is used to identify distantly related proteins among remaining $N - n$ proteins in the background. Proteins with zero similarity with the target class are extracted. Then they are potentially defined as the other class by pseudo-absence selection methods (Senay *et al.*, 2013) from spatial statistics. The target class can be seen as real presence data. For the data to be balanced, assume that two classes contain the same number of proteins. Next, by the classical SVM, these two classes are used to identify closely related proteins among remaining $N - 2n$ proteins. Those found by this procedure can be functionally linked to the known class or interesting proteins.

Semi-supervised learning can be applied to make use of large unlabeled data and small labeled data. Some of these methods directly try to label the unlabeled data. Self-training is a commonly used semi-supervised learning technique (Zhu, 2006). Self-training is an incremental algorithm that initially builds a classifier using a small amount of labeled data. So it iteratively predicts the labels of the unlabeled data and then predicted labels are added to the labeled data. Here, the function `net.infer.ST` is the self-training method for SVM. Also, the function `net.infer` is the special case of `net.infer.ST` where a single iteration is conducted.

4 Example

Consider a simple example about a graph representing the curated set of literature predicted protein-protein interactions, containing 2885 nodes, named using yeast standard names.

```
 library(PPInfer)
 data(litG)
 litG <- igraph.from.graphNEL(litG)
 summary(litG)
```

```
IGRAPH 07d2b2f UN-- 2885 315 --
+ attr: name (v/c)
```

```
 sg <- decompose(litG, min.vertices = 50)
 sg <- sg[[1]]           # largest subgraph
 summary(sg)
```

```
IGRAPH aa6ed78 UN-- 88 107 --
+ attr: name (v/c)
```

We use only the largest subnetwork in this example. There are 88 proteins and 107 interactions.

```
 V(sg)$color <- "green"
 V(sg)$label.font <- 3
 V(sg)$label.cex <- 1
 V(sg)$label.color <- "black"
 V(sg)[1:10]$color <- "blue"
```

```
plot(sg, layout = layout.kamada.kawai(sg), vertex.size = 10)
```

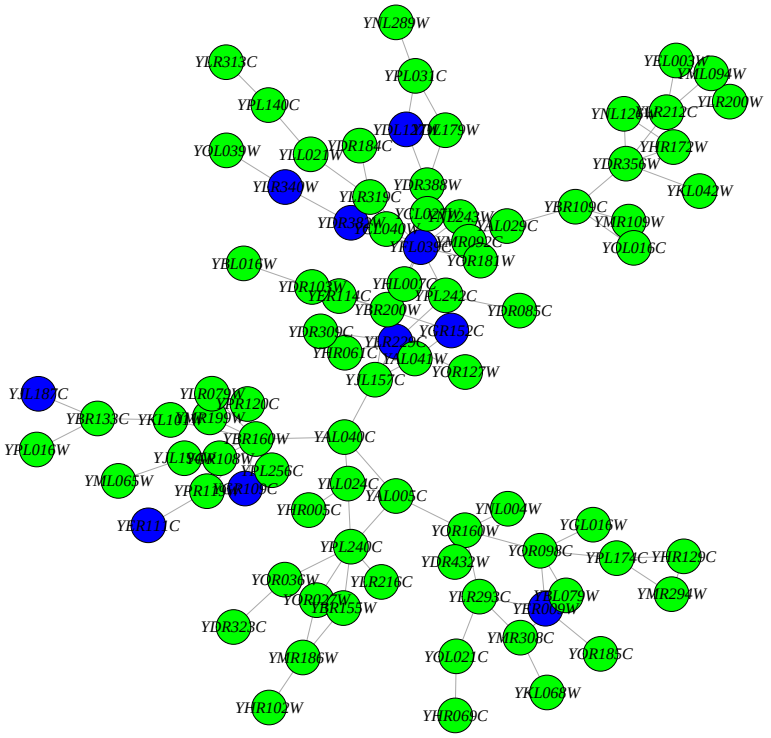

Figure 1: Network among yeast proteins with target class in blue and remaining proteins in green.

First, calculate the kernel matrix and choose 10 proteins as a target class. Then we can find proteins closely related to the target class by using the KSVM for a graph (Samatova *et al.*, 2013; Kolaczyk & Csardi, 2014). Network of interactions among proteins with target class in blue and backgrounds in green. Red vertices represent the top 20 proteins which are most closely related to the target class.

```
K <- net.kernel(sg)
set.seed(123)
litG.infer <- net.infer(names(V(sg))[1:10], K, top = 20, cross = 10)
litG.infer$CVerror
```

```
[1] 0.45
```

```
index <- match(litG.infer$top,names(V(sg)))
V(sg)[index]$color <- "red"
```

```
plot(sg, layout = layout.kamada.kawai(sg), vertex.size = 10)
```

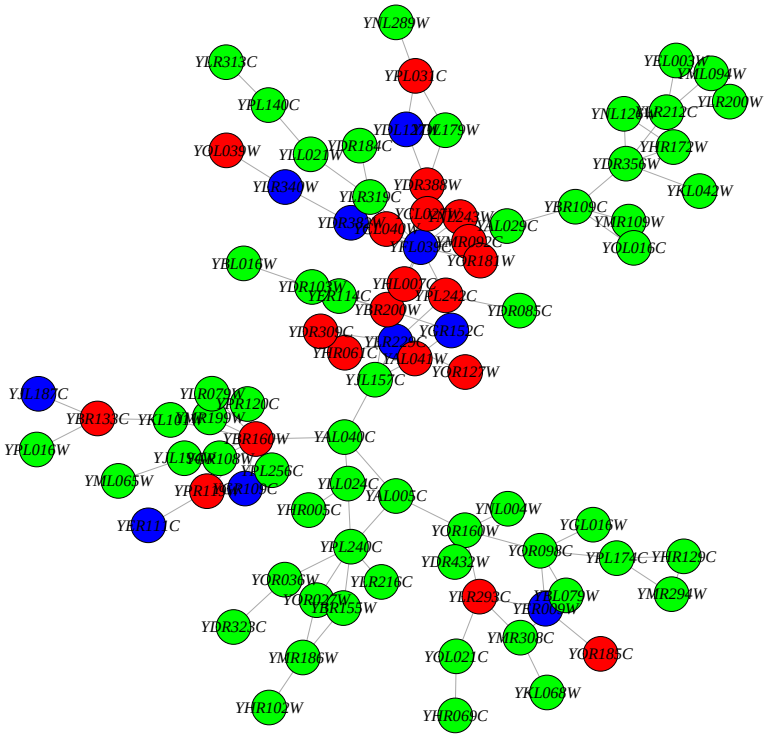

Figure 2: Red vertices denote the top 20 yeast proteins which are most closely related to 10 proteins of the target class.

Note that the number of proteins is not greater than a half of total proteins in a kernel matrix due to $N - 2n > 0$. Also, the number of top proteins to be inferred is less than or equal to $N - 2n$. If we use 50 proteins as a target class, then there is an error since $N - 2n = -12$. If we use 40 proteins as a target, and want to find top 20 proteins, then the number of available top proteins are only 8, which is the minimum of $N - 2n = 8$ and 20.

```
litG.infer <- try(net.infer(names(V(sg))[1:50], K, top = 20))
cat(litG.infer)
```

```
Error in net.infer(names(V(sg))[1:50], K, top = 20) :
  size of list is too large
```

```
litG.infer <- net.infer(names(V(sg))[1:40], K, top = 20)
litG.infer$top
```

```
[1] "YBR160W" "YDL179W" "YAL041W" "YDR323C" "YBR133C" "YPL174C" "YPL140C"
[8] "YDR356W"
```

Next, consider the functional enrichment analysis. Here, we use the same kernel but different gene names. For the ORA, we use top 10 proteins among 88 proteins.

```
data(examplePathways)
data(exampleRanks)
geneNames <- names(exampleRanks)
set.seed(1)
gene.names <- sample(geneNames, length(V(sg)))
rownames(K) <- gene.names
myInterestingGenes <- sample(gene.names, 10)
infer <- net.infer(myInterestingGenes, K)
gene.id <- infer$top

# ORA
result.ORA <- ORA(examplePathways, gene.id[1:10])

ORA.barplot(result.ORA, category = "Category", size = "Size",
            count = "Count", pvalue = "pvalue", sort = "pvalue") +
            scale_colour_gradient(low = 'red', high = 'gray', limits=c(0, 0.1))
```

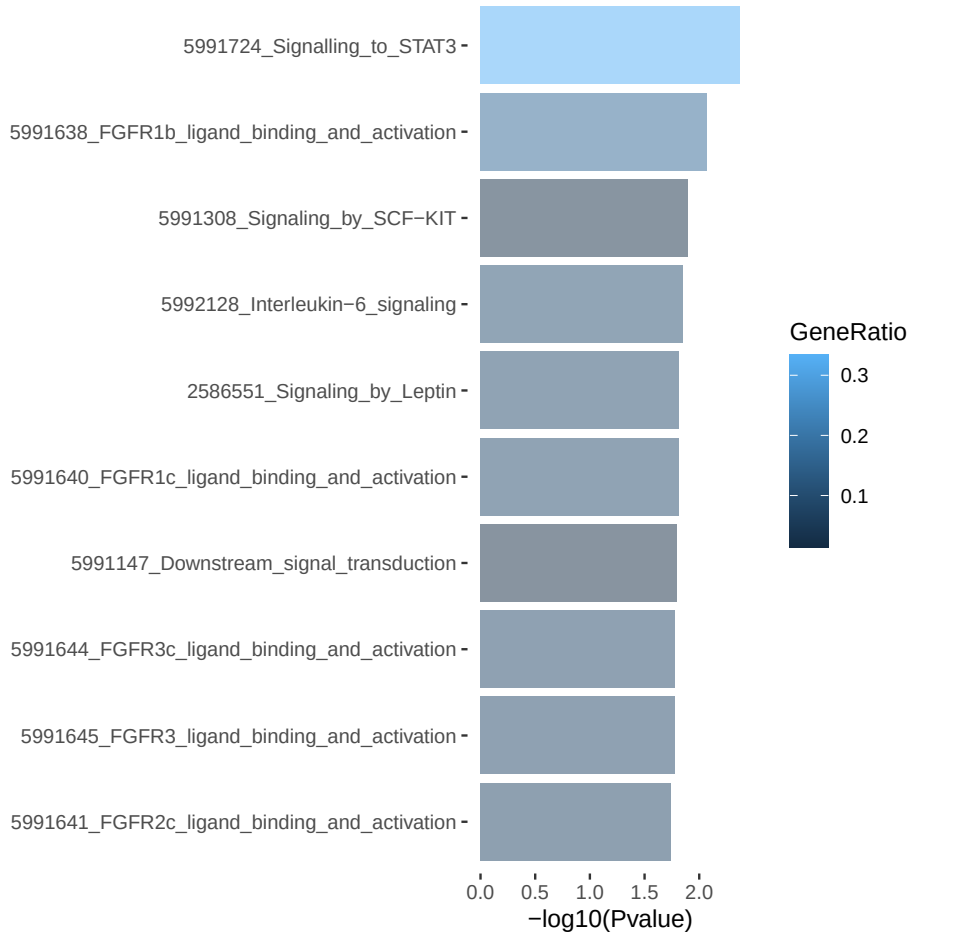

Figure 3: Result from the over-representation analysis with Gene Ontology

```
# GSEA
index <- !is.na(infer$score)
gene.id <- infer$top[index]
scores <- infer$score[index]
scaled.scores <- as.numeric(scale(scores))
names(scaled.scores) <- gene.id
set.seed(1)
result.GSEA <- fgsea(examplePathways, scaled.scores, nperm=1000)
```

```
GSEA.barplot(result.GSEA, category = 'pathway', score = 'NES', pvalue = 'pval',
             numChar = 50, sort = 'NES', decreasing = TRUE)
```

	pathway	NES	pval
13	5991071_Signal_Transduction	1.375176	0.07070707
1	2586551_Signaling_by_Leptin	1.315986	0.03198294
2	5334727_Mus_musculus_biological_processes	1.315986	0.03198294
62	5991310_Signaling_by_Leptin	1.315986	0.03198294
101	5991724_Signalling_to_STAT3	1.315986	0.03198294
118	5992038_Growth_hormone_receptor_signaling	1.315986	0.03198294
129	5992128_Interleukin-6_signaling	1.315986	0.03198294
18	5991145_NGF_signalling_via_TRKA_from_the_plasma_me...	1.301882	0.12248996
19	5991146_Signalling_by_NGF	1.301882	0.12248996
20	5991147_Downstream_signal_transduction	1.301882	0.12248996

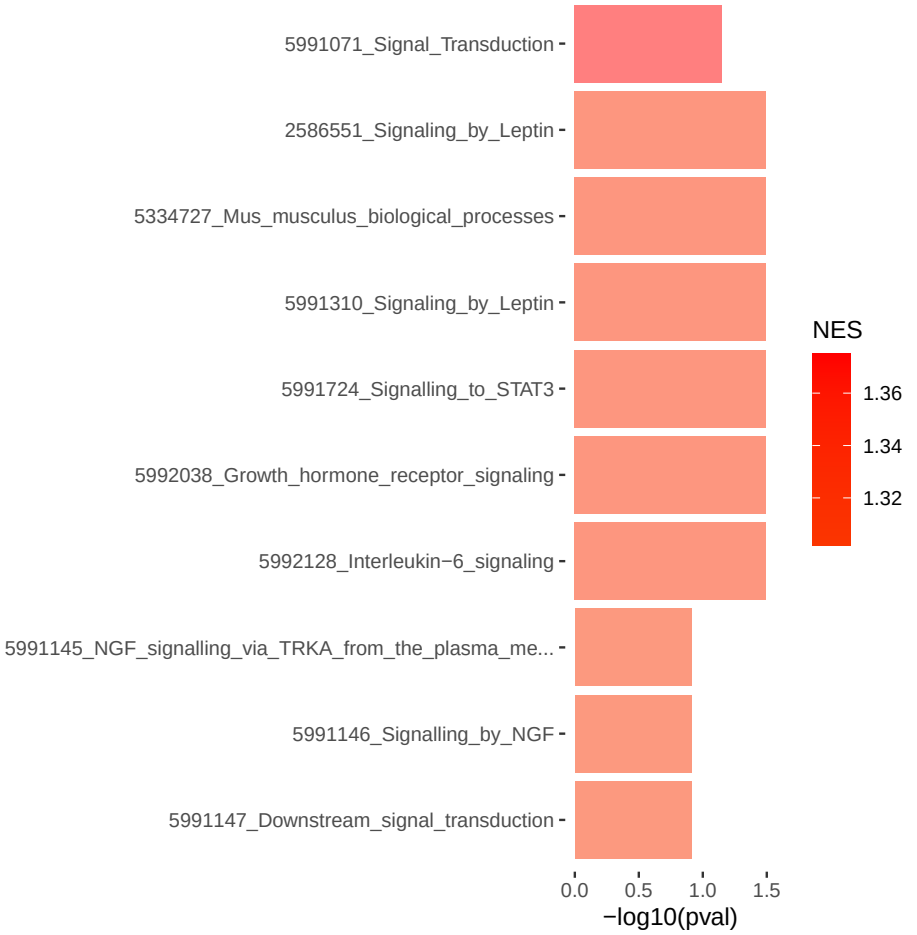

Figure 4: Result from the gene set enrichment analysis

```
enrich.net(result.GSEA, examplePathways, node.id = 'pathway',
           pvalue = 'pval', pvalue.cutoff = 0.25, degree.cutoff = 0,
           n = 100, vertex.label.cex = 0.75, show.legend = FALSE,
           edge.width = function(x) {5*sqrt(x)},
           layout=igraph::layout.kamada.kawai)
```

```
IGRAPH 00778f7 UN-- 43 190 --
+ attr: name (v/c), size (v/n), shape (v/c), color (v/c), width (e/n)
+ edges from 00778f7 (vertex names):
[1] 2586551_Signaling_by_Leptin              --5334727_Mus_musculus_biological_processes
[2] 2586551_Signaling_by_Leptin              --5991310_Signaling_by_Leptin
[3] 5334727_Mus_musculus_biological_processes--5991310_Signaling_by_Leptin
[4] 2586551_Signaling_by_Leptin              --5991724_Signalling_to_STAT3
[5] 2586551_Signaling_by_Leptin              --5992038_Growth_hormone_receptor_signaling
[6] 5991310_Signaling_by_Leptin              --5992038_Growth_hormone_receptor_signaling
[7] 2586551_Signaling_by_Leptin              --5992128_Interleukin-6_signaling
+ ... omitted several edges
```

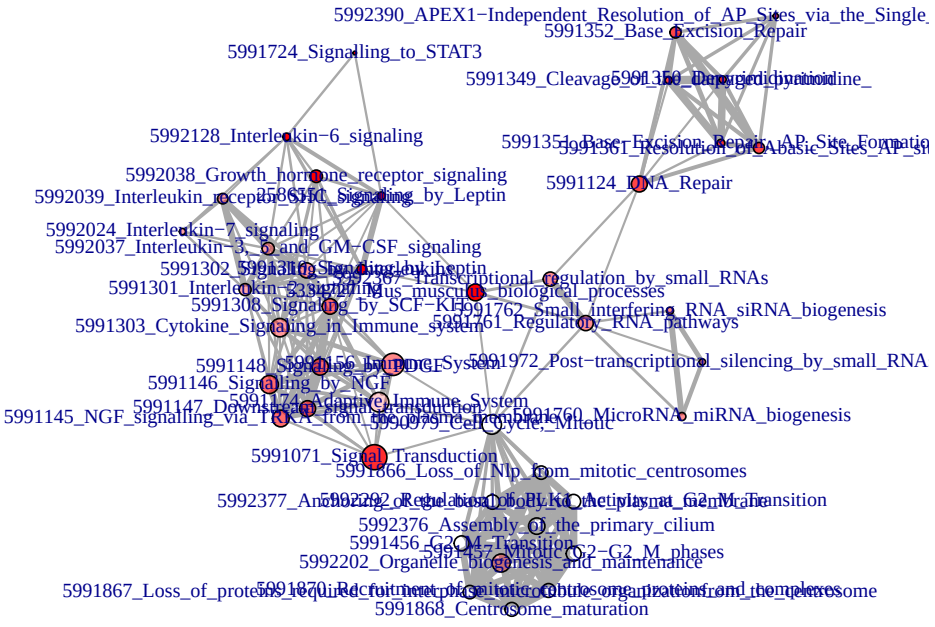

Figure 5: Network from the gene set enrichment analysis

In the network, the connection between nodes depends on the proportion of overlapping genes between two categories. The size of nodes is proportional to the size of gene sets. The more significant categories are, the less transparent their nodes are.

5 Protein-protein interaction

5.1 Download the kernel matrix

We need a list of proteins and the kernel matrix to infer functionally related proteins. For the database for kernel matrix, we use the STRING data for protein-protein interactions for human and mouse. You do not have to calculate these kernel matrices. To save significant time, you can download kernel matrices at `https://zenodo.org/record/1066236` for human and mouse.

5.2 PPI for human

Consider two examples for human. First, we can find proteins related to apoptosis. Then, load the kernel matrix for human.

```
library(GOstats)
# load kernel matrix
K.9606 <- readRDS("K9606.rds")
# remove prefix
rownames(K.9606) <- sub(".*\\.", "", rownames(K.9606))
# load target class from KEGG apoptosis pathway
data(apopGraph)
list.proteins <- nodes(apopGraph)
head(list.proteins)
```

There are many types of protein ID or gene ID. By using `getBM` in `biomaRt`, we can change the format. For this reason, input and output formats must be available for `getBM`. Note that the number of proteins used as a target may be different from the number of proteins in the input since mapping between formats is not always one-to-one in `getBM`.

```
# find top 100 proteins
apoptosis.infer <- ppi.infer.human(list.proteins, K.9606, output="entrezgene", 100)
gene.id <- apoptosis.infer$top
head(gene.id)
# GO terms
params <- new("GOHyperGParams", geneIds = gene.id,annotation = "org.Hs.eg.db",
              ontology = "BP", pvalueCutoff = 0.001, conditional = FALSE,
              testDirection = "over")
(hgOver <- hyperGTest(params))
```

```
# Top 10 biological functions related to apoptosis by using GO terms
ORA.barplot(summary(hgOver), category = "Term", size = "Size",
            count = "Count", pvalue = "Pvalue", p.adjust.methods = 'fdr')
# KEGG pathway
params <- new("KEGGHyperGParams", geneIds = gene.id, annotation = "org.Hs.eg.db",
              pvalueCutoff = 0.05, testDirection = "over")
(hgOver <- hyperGTest(params))
# Top 10 biological functions related to apoptosis by using KEGG pathways
ORA.barplot(summary(hgOver), category = "Term", size = "Size",
            count = "Count", pvalue = "Pvalue", p.adjust.methods = 'fdr')
```

We found functionally related top 100 proteins in terms of Entrez-ID by `ppi.infer.human`. However, we are often interested in biological functions about such inferred proteins. This is the top 10 categories from gene ontology. As we expected, inferred proteins have similar biological functions with the target, apoptosis. Thus this example supports that this model is reliable. Also, we get the similar result from KEGG pathway. Next, take another example. The protein p53 is known for inhibition of cancer. From KEGG pathway, we can find proteins for p53 signaling pathway. The procedure is the same to the previous example, but for the target class.

```
library(KEGG.db)
library(limma)
# load target class for p53
mget('p53 signaling pathway', KEGGPATHNAME2ID)
kegg.hsa <- getGeneKEGGLinks(species.KEGG='hsa')
index <- which(kegg.hsa[,2] == 'path:hsa04115')
path.04115 <- kegg.hsa[index,1]
head(path.04115)
# find top 100 proteins
hsa04115.infer <- ppi.infer.human(path.04115, K.9606, input = "entrezgene",
                                  output = "entrezgene", nrow(K.9606))
gene.id <- data.frame(hsa04115.infer$top)[,1]
head(gene.id)
rm(K.9606)
index <- !is.na(hsa04115.infer$score)
gene.id <- hsa04115.infer$top[index]
scores <- hsa04115.infer$score[index]
scaled.scores <- as.numeric(scale(scores))
names(scaled.scores) <- gene.id
# GO terms
library(org.Hs.eg.db)
xx <- as.list(org.Hs.egGO2EG)
```

```
set.seed(1)
fgseaRes <- fgsea(xx, scaled.scores, nperm = 1000)
# Top 10 biological functions related to p53 signaling pathway by using GO terms
GSEA.barplot(data.frame(fgseaRes, select(GO.db, fgseaRes$pathway, "TERM")),
            category = 'TERM', score = 'NES', pvalue = 'padj',
            sort = 'NES', decreasing = TRUE)
# KEGG pathways
pathway.id <- unique(kegg.hsa[,2])
yy <- list()
for(i in 1:length(pathway.id))
{
  index <- which(kegg.hsa[,2] == pathway.id[i])
  yy[[i]] <- kegg.hsa[index,1]
}
library(Category)
names(yy) <- getPathNames(sub("[[:alpha:]]+....", "", pathway.id))
yy[which(names(yy) == 'NA')] <- NULL
set.seed(1)
fgseaRes <- fgsea(yy, scaled.scores, nperm=1000)
# Top 10 biological functions related to p53 signaling pathway by using KEGG pathways
GSEA.barplot(fgseaRes, category = 'pathway', score = 'NES', pvalue = 'padj',
             sort = 'NES', decreasing = TRUE)
```

5.3 PPI for mouse

For mouse, we can infer functionally related proteins by `ppi.infer.mouse` with the kernel matrix for mouse. The first example is Acute myeloid leukemia.

```
# load kernel matrix
K.10090 <- readRDS("K10090.rds")
# remove prefix
rownames(K.10090) <- sub(".*\\.", "", rownames(K.10090))
# load target class
mget('Acute myeloid leukemia', KEGGPATHNAME2ID)
kegg.mmu <- getGeneKEGGLinks(species.KEGG='mmu')
index <- which(kegg.mmu[,2] == 'path:mmu05221')
path.05221 <- kegg.mmu[index,1]
head(path.05221)
# find top 100 proteins
path.05221.infer <- ppi.infer.mouse(path.05221, K.10090, input="entrezgene",
                                    output="entrezgene", nrow(K.10090))
```

```
gene.id <- path.05221.infer$top
head(gene.id)
# ORA
params <- new("GOHyperGParams", geneIds = gene.id[1:100],
              annotation = "org.Mm.eg.db",
              ontology = "BP", pvalueCutoff = 0.001,
              conditional = FALSE, testDirection = "over")
(hgOver <- hyperGTest(params))
# Top 10 biological functions related to Acute myeloid leukemia
ORA.barplot(summary(hgOver), category = "Term", size = "Size",
            count = "Count", pvalue = "Pvalue", p.adjust.methods = 'fdr')
# GSEA
library(org.Mm.eg.db)
xx <- as.list(org.Mm.egGO2EG)
index <- !is.na(path.05221.infer$score)
gene.id <- path.05221.infer$top[index]
scores <- path.05221.infer$score[index]
scaled.scores <- as.numeric(scale(scores))
names(scaled.scores) <- gene.id
set.seed(1)
fgseaRes <- fgsea(xx, scaled.scores, nperm=1000)
# Top 10 biological functions related to Acute myeloid leukemia
GSEA.barplot(na.omit(data.frame(fgseaRes, select(GO.db, fgseaRes$pathway, 'TERM'))),
              category = 'TERM', score = 'NES', pvalue = 'padj',
              sort = 'NES', decreasing = TRUE)
```

The second example is Ras signaling pathway. The Ras proteins are GTPases that function as molecular switches for signaling pathways regulating cell proliferation, survival, growth, migration, differentiation or cytoskeletal dynamism.

```
# load target class
mget('Ras signaling pathway', KEGGPATHNAME2ID)
kegg.mmu <- getGeneKEGGLinks(species.KEGG='mmu')
index <- which(kegg.mmu[,2] == 'path:mmu04014')
path.04014 <- kegg.mmu[index,1]
head(path.04014)
# find top 100 proteins
path.04014.infer <- ppi.infer.mouse(path.04014, K.10090,
                                    input="entrezgene",output="entrezgene", nrow(K.10090))
gene.id <- data.frame(path.04014.infer$top)[,1]
head(gene.id)
```

```
rm(K.10090)
# ORA
params <- new("KEGGHyperGParams", geneIds = gene.id[1:100],
              annotation = "org.Mm.eg.db", pvalueCutoff = 0.05, testDirection = "over")
(hgOver <- hyperGTest(params))
# Top 10 biological functions related to Ras signaling pathway
ORA.barplot(summary(hgOver), category = "Term", size = "Size",
            count = "Count", pvalue = "Pvalue", p.adjust.methods = 'fdr')
# GSEA
kegg.mmu <- getGeneKEGGLinks(species.KEGG='mmu')
head(kegg.mmu)
pathway.id <- unique(kegg.mmu[,2])
yy <- list()
for(i in 1:length(pathway.id))
{
  index <- which(kegg.mmu[,2] == pathway.id[i])
  yy[[i]] <- kegg.mmu[index,1]
}
names(yy) <- getPathNames(sub("[[:alpha:]]+....", "", pathway.id))
yy[which(names(yy) == 'NA')] <- NULL
index <- !is.na(path.04014.infer$score)
gene.id <- path.04014.infer$top[index]
scores <- path.04014.infer$score[index]
scaled.scores <- as.numeric(scale(scores))
names(scaled.scores) <- gene.id
set.seed(1)
fgseaRes <- fgsea(yy, scaled.scores, nperm = 1000)
# Top 10 biological functions related to Ras signaling pathway
GSEA.barplot(fgseaRes, category = 'pathway', score = 'NES', pvalue = 'padj')
```

We discussed about how to infer functionally related proteins for human and mouse. Two functions `ppi.infer.human` and `ppi.infer.mouse` are specially designed because popular organisms are human and mouse. However, other kinds of species are also available in `net.infer` if kernel matrices are given.

5.4 PPI for other organisms

```
############################## E. coli
string.db.511145 <- STRINGdb$new(version = '11', species = 511145)
string.db.511145.graph <- string.db.511145$get_graph()
K.511145 <- net.kernel(string.db.511145.graph)
rownames(K.511145) <- sub("[[:digit:]]+.", "", rownames(K.511145))
```

```
# load target class (DNA replication)
kegg.eco <- getGeneKEGGLinks(species.KEGG='eco')
index <- which(kegg.eco[,2] == 'path:eco03030')
path.03030 <- kegg.eco[index,1]
head(path.03030)
sce03030.infer <- net.infer(path.03030, K.511145, top = 100)
gene.id <- data.frame(sce03030.infer$top)[,1]
head(gene.id)
rm(K.511145)
############################## yeast
# string.db.4932 <- STRINGdb$new(version = '11', species = 4932)
# string.db.4932.graph <- string.db.4932$get_graph()
# K.4932 <- net.kernel(string.db.4932.graph)
# saveRDS(K.4932, 'K4932.rds')
K.4932 <- readRDS("K4932.rds")
dim(K.4932)
rownames(K.4932) <- sub("[[:digit:]]+.", "", rownames(K.4932))
# load target class (Cell cycle)
kegg.sce <- getGeneKEGGLinks(species.KEGG='sce')
index <- which(kegg.sce[,2] == 'path:sce04111')
path.04111 <- kegg.sce[index,1]
head(path.04111)
sce04111.infer <- net.infer(path.04111, K.4932, top = 100)
gene.id <- data.frame(sce04111.infer$top)[,1]
head(gene.id)
rm(K.4932)
# functional enrichment
params <- new("GOHyperGParams", geneIds = gene.id, annotation = "org.Sc.sgd.db",
              ontology = "BP",pvalueCutoff = 0.001, conditional = FALSE,
              testDirection = "over")
(hgOver <- hyperGTest(params))
# Top 10 biological functions related to Cell cycle
ORA.barplot(summary(hgOver), category = "Term", size = "Size",
            count = "Count", pvalue = "Pvalue", p.adjust.methods = 'fdr')
############################## C. elegans
# string.db.6239 <- STRINGdb$new(version = '11', species = 6239)
# string.db.6239.graph <- string.db.6239$get_graph()
# K.6239 <- net.kernel(string.db.6239.graph)
# saveRDS(K.6239, 'K6239.rds')
```

```
K.6239 <- readRDS("K6239.rds")
dim(K.6239)
rownames(K.6239) <- sub("[[:digit:]]+.", "", rownames(K.6239))
# load target class (DNA replication)
kegg.cel <- getGeneKEGGLinks(species.KEGG='cel')
index <- which(kegg.cel[,2] == 'path:cel03030')
path.03030 <- kegg.cel[index,1]
path.03030 <- sub('.*\\_', '', path.03030)
head(path.03030)
cel03030.infer <- net.infer(path.03030, K.6239, top=100)
gene.id <- data.frame(cel03030.infer$top)[,1]
head(gene.id)
rm(K.6239)
library(org.Ce.eg.db)
gene.id2 <- as.vector(na.omit(select(org.Ce.eg.db,
                        keys=as.character(gene.id), "ENTREZID",
                        keytype = 'SYMBOL')[,2]))
# functional enrichment
params <- new("GOHyperGParams", geneIds = gene.id2, annotation = "org.Ce.eg.db",
              ontology = "BP", pvalueCutoff = 0.001, conditional = FALSE,
              testDirection = "over")
(hgOver <- hyperGTest(params))
# Top 10 biological functions related to DNA replication
ORA.barplot(summary(hgOver), category = "Term", size = "Size",
            count = "Count", pvalue = "Pvalue", p.adjust.methods = 'fdr')
############################## Drosophila melanogaster
# string.db.7227 <- STRINGdb$new(version = '11', species = 7227)
# string.db.7227.graph <- string.db.7227$get_graph()
# K.7227 <- net.kernel(string.db.7227.graph)
# saveRDS(K.7227, 'K7227.rds')
K.7227 <- readRDS("K7227.rds")
dim(K.7227)
rownames(K.7227) <- sub("[[:digit:]]+.", "", rownames(K.7227))
# load target class (Proteasome)
kegg.dme <- getGeneKEGGLinks(species.KEGG='dme')
index <- which(kegg.dme[,2] == 'path:dme03050')
path.03050 <- kegg.dme[index,1]
path.03050 <- sub('.*\\_', '', path.03050)
head(path.03050)
```

```
library(org.Dm.eg.db)
path2.03050 <- select(org.Dm.eg.db, keys = path.03050,
                      "FLYBASEPROT", keytype = 'ALIAS')[,2]
dme03050.infer <- net.infer(path2.03050, K.7227, top = 100)
gene.id <- data.frame(dme03050.infer$top)[,1]
head(gene.id)
rm(K.7227)
gene.id2 <- as.vector(na.omit(select(org.Dm.eg.db,
                          keys=as.character(gene.id), "ENTREZID",
                          keytype = 'FLYBASEPROT')[,2]))
# functional enrichment
params <- new("GOHyperGParams", geneIds = gene.id2, annotation = "org.Dm.eg.db",
              ontology = "BP", pvalueCutoff = 0.001, conditional = FALSE,
              testDirection="over")
(hgOver <- hyperGTest(params))
# Top 10 biological functions related to Proteasome
ORA.barplot(summary(hgOver), category = "Term", size = "Size",
            count = "Count", pvalue = "Pvalue", p.adjust.methods = 'fdr')
############################### Arabidopsis thaliana
# string.db.3702 <- STRINGdb$new(version = '11', species = 3702)
# string.db.3702.graph <- string.db.3702$get_graph()
# K.3702 <- net.kernel(string.db.3702.graph)
# saveRDS(K.3702, 'K3702.rds')
K.3702 <- readRDS("K3702.rds")
dim(K.3702)
rownames(K.3702) <- sub("[[:digit:]]+.", "", rownames(K.3702))
rownames(K.3702) <- gsub("\\..*", "", rownames(K.3702))
# load target class (Photosynthesis)
kegg.ath <- getGeneKEGGLinks(species.KEGG = 'ath')
index <- which(kegg.ath[,2] == 'path:ath00195')
path.00195 <- kegg.ath[index,1]
path.00195 <- sub('.*\\_', '', path.00195)
head(path.00195)
ath00195.infer <- net.infer(path.00195, K.3702, top = 100)
gene.id <- data.frame(ath00195.infer$top)[,1]
head(gene.id)
rm(K.3702)
# functional enrichment
params <- new("GOHyperGParams", geneIds = gene.id, annotation = "org.At.tair.db",
```

```
              ontology = "BP",pvalueCutoff = 0.001,conditional = FALSE,
              testDirection="over")
(hgOver <- hyperGTest(params))
# Top 10 biological functions related to Photosynthesis
ORA.barplot(summary(hgOver), category = "Term", size = "Size",
            count = "Count", pvalue = "Pvalue", p.adjust.methods = 'fdr')
############################### Zebra fish
# string.db.7955 <- STRINGdb$new(version = '11', species = 7955)
# string.db.7955.graph <- string.db.7955$get_graph()
# K.7955 <- net.kernel(string.db.7955.graph)
# saveRDS(K.7955, 'K7955.rds')
K.7955 <- readRDS("K7955.rds")
dim(K.7955)
rownames(K.7955) <- sub("[[:digit:]]+.", "", rownames(K.7955))
# load target class (ErbB signaling pathway)
kegg.dre <- getGeneKEGGLinks(species.KEGG = 'dre')
index <- which(kegg.dre[,2] == 'path:dre04012')
path.04012 <- kegg.dre[index,1]
path.04012 <- sub('.*\\_', '', path.04012)
head(path.04012)
library(org.Dr.eg.db)
path2.04012 <- select(org.Dr.eg.db, path.04012, c("ENSEMBLPROT"))[,2]
dre04012.infer <- net.infer(path2.04012, K.7955, top = 100)
gene.id <- data.frame(dre04012.infer$top)[,1]
head(gene.id)
rm(K.7955)
gene.id2 <- as.vector(na.omit(select(org.Dr.eg.db,
                                     keys = as.character(gene.id), "ENTREZID",
                                     keytype = 'ENSEMBLPROT')[,2]))
# functional enrichment
params <- new("GOHyperGParams", geneIds = gene.id2, annotation = "org.Dr.eg.db",
              ontology = "BP",pvalueCutoff = 0.001, conditional = FALSE,
              testDirection = "over")
(hgOver <- hyperGTest(params))
# Top 10 biological functions related to ErbB signaling pathway
ORA.barplot(summary(hgOver), category = "Term", size = "Size",
            count = "Count", pvalue = "Pvalue", p.adjust.methods = 'fdr')
```

6 Session Information

sessionInfo()

```
R version 4.0.0 alpha (2020-04-05 r78150)
Platform: x86_64-apple-darwin17.0 (64-bit)
Running under: macOS Mojave 10.14.6

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRblas.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.0/Resources/lib/libRlapack.dylib

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] parallel  stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] PPInfer_1.13.2      yeastExpData_0.33.0 graph_1.65.2
 [4] BiocGenerics_0.33.3 STRINGdb_1.99.10    igraph_1.2.5
 [7] ggplot2_3.3.0       kernlab_0.9-29      fgsea_1.13.4
[10] Rcpp_1.0.4.6        biomaRt_2.43.5

loaded via a namespace (and not attached):
 [1] Biobase_2.47.3       httr_1.4.1           bit64_0.9-7
 [4] gsubfn_0.7           gtools_3.8.2         assertthat_0.2.1
 [7] askpass_1.1          stats4_4.0.0         BiocFileCache_1.11.4
[10] blob_1.2.1           progress_1.2.2       pillar_1.4.3
[13] RSQLite_2.2.0        lattice_0.20-41      glue_1.4.0
[16] chron_2.3-55         digest_0.6.25        RColorBrewer_1.1-2
[19] colorspace_1.4-1     Matrix_1.2-18        plyr_1.8.6
[22] XML_3.99-0.3         pkgconfig_2.0.3      purrr_0.3.3
[25] scales_1.1.0         gdata_2.18.0         BiocParallel_1.21.2
[28] tibble_3.0.0         openssl_1.4.1        farver_2.0.3
[31] IRanges_2.21.8       sqldf_0.4-11         ellipsis_0.3.0
[34] withr_2.1.2          cli_2.0.2            proto_1.0.0
[37] magrittr_1.5         crayon_1.3.4         memoise_1.1.0
[40] fansi_0.4.1          gplots_3.0.3         tools_4.0.0
```

```
[43] hash_2.2.6.1          data.table_1.12.8    prettyunits_1.1.1
[46] hms_0.5.3             lifecycle_0.2.0      stringr_1.4.0
[49] S4Vectors_0.25.15     munsell_0.5.0        plotrix_3.7-7
[52] AnnotationDbi_1.49.1  compiler_4.0.0       caTools_1.18.0
[55] rlang_0.4.5           grid_4.0.0           RCurl_1.98-1.1
[58] rappdirs_0.3.1        labeling_0.3         bitops_1.0-6
[61] gtable_0.3.0          DBI_1.1.0            curl_4.3
[64] R6_2.4.1              gridExtra_2.3        dplyr_0.8.5
[67] bit_1.1-15.2          fastmatch_1.1-0      KernSmooth_2.23-16
[70] stringi_1.4.6         vctrs_0.2.4          png_0.1-7
[73] dbplyr_1.4.2          tidyselect_1.0.0
```

7 References

Kolaczyk, E. D. & Csardi, G. (2014). *Statistical analysis of network data with R.* Springer.

Ma, Y. (2014). *Support vector machines applications.* G. Guo (Ed.). Springer.

Samatova, *et al.* (Eds.). (2013). *Practical graph mining with R.* CRC Press.

Senay, S. D. *et al.* (2013). Novel three-step pseudo-absence selection technique for improved species distribution modelling. *PLOS ONE.* **8(8)**, e71218.

Smola, A. J. & Kondor, R. (2003). Kernels and regularization on graphs. *In Learning theory and kernel machines.* 144-158. Springer Berlin Heidelberg.

Werther, M., & Seitz, H. (Eds.). (2008). *Protein-protein interaction.* Springer.

Zhu, X. (2006). Semi-supervised learning literature survey. *Computer Science, University of Wisconsin-Madison.* 2(3), 4.