lpNet

Bettina Knapp, Marta R. A. Matos, Johanna Mazur and Lars Kaderali

November 8, 2022

Contents

1 Introduction

This package infers networks using perturbation data, either steady-state or time-series, with a linear optimization program as published in [KK13] and [Mat13]. In section 2, the idea of the network inference method and the type of data that can be used is explained. Section 3 shows how to use the package on a quick example. In section 4 and 5, Leave-One-Out cross-validation (LOOCV) and k-fold cross-validation are used to find the best parameter λ which minimizes the mean squared error (MSE) of the used objective function as described in [KK13]. Section 6 exemplifies how prior knowledge can be included in the network inference process. Finally, in section 7 we applied the network inference method on a real data set studying ERBB signaling in trastuzumab resistant breast cancer cells [SFL$^+$09].

2 Data

The method proposed in this package aims at reconstructing intracellular signal transduction networks from perturbation data, either steady-state or time-series. We consider signal transduction as an information flow through a network where nodes represent genes (or proteins) and edges represent the continuous strengths of interactions between the nodes (genes or proteins). By using experimental interventions the flow in the network can be perturbed and this leads to a problem which is an inverse to the well-known maximum-flow / minimum cut problem in graph theory. The flow through the network is perturbed by different cuts and the measured effect can be used to reconstruct the underlying graph. We formulate this problem as a linear program which can be solved efficiently using the simplex algorithm.

The perturbation experiments can be performed for example using RNA interference (RNAi) which allows to specifically silence one or several genes at the same time. To quantify the effect of a perturbation on the remaining genes, for instance, flow cytometry or gene expression measurements [SPP+05, SFL+09] can be used.

3 Starting

This section shows a first small example to infer signaling networks from perturbation data using the linear programming (LP) approach. First load the lpNet package.

```
> library("lpNet")
```

Then, define details about the perturbation data which is used for the network inference. When using steady-state data, define the number of genes and the number of perturbation experiments which are given. Thereafter, we define a random observation matrix and define threshold δ_i as the mean of the observations of each gene $i \in \{1, \ldots, n\}$. Although here we define a δ value per gene, these values can also be defined for each gene and perturbation experiment, for each gene and time-point, etc.

```
> n <- 5 # number of genes
> K <- 7 # number of perturbation experiments
> obs <- matrix(rnorm(n*K), nrow=n, ncol=K)
> delta <- apply(obs, 1, mean)
> delta_type <- "perGene"
```

```
> # perturbation vector (0 if gene inactivated and 1 otherwise)
> b <- c(0,1,1,1,1,    # perturbation exp1
+        1,0,1,1,1,     # perturbation exp2
+        1,1,0,1,1,     # perturbation exp3...
+        1,1,1,0,1,
+        1,1,1,1,0,
+        1,0,0,1,1,
+        1,1,1,1,1)
```

Using this, the network can be inferred with the LP method for a specific penalty parameter λ with the function "doILP". To convert the result of the "doILP" function into an adjacency matrix use "getAdja".

```
> res1 <- doILP(obs, delta, lambda=1, b, n, K, T_=NULL,
+               annot=getEdgeAnnot(n), delta_type)
> adja1 <- getAdja(res1, n)
```

If the inferred network is assumed to have activating edges only, this can be used during the network reconstruction.

```
> res2 <- doILP(obs, delta, lambda=1, b, n, K, T_=NULL,
+               annot=getEdgeAnnot(n,allpos=TRUE), delta_type,
+               all.pos=TRUE)
> adja2 <- getAdja(res2,n)
```

If using time-series data, the number of time points in the data needs to be defined, and the observation matrix becomes a 3-dimensional array, where the third dimension represents time-points:

```
> T_ <- 4 # number of time points
> obs_ts <- array(rnorm(n*K*T_), c(n,K,T_))
```

Besides, flag_time_series needs to be set to TRUE when executing doILP:

```
> res1 <- doILP(obs_ts, delta, lambda=1, b, n, K, T_,
+               annot=getEdgeAnnot(n), delta_type,
+               flag_time_series=TRUE)
> adja1 <- getAdja(res1, n)
```

4 Leave-One-Out cross-validation

If the penalty parameter λ is not given, it can be determined using a stratified LOOCV of the data. First, the parameters (mean and standard deviation) for the two Gaussian distributions representing activation and deactivation have to be defined. Also, these parameters values can be different for each gene, gene + perturbation experiment, gene + time point, etc.

```
> active_mu <- 1.1
> inactive_mu <- 0.9
> active_sd <- inactive_sd <- 0.01
> mu_type <- "single"
```

These parameters are used in each step of the LOOCV to compute the MSE. For this, the learned network (of the respective step) is used to predict if the removed observation is supposed to be in active or inactive state and thus, is sampled from the normal distribution of activated or inactivated genes, respectively.

```
> times <- 5 # number of times the removed observation is sampled
> annot_node <- seq(1,n) # annotation of the nodes
> loocv_res <- loocv(kfold=NULL, times, obs, delta, lambda=1,
+                    b, n, K, T_=NULL, annot=getEdgeAnnot(n),
+                    annot_node, active_mu, active_sd,
+                    inactive_mu, inactive_sd, mu_type,
+                    delta_type)
> loocv_res$MSE
```

The "loocv" function results in a list giving the MSE and the edge weights learned in each LOOCV step. The weights are finally summarized to compute an adjacency matrix.

```
> adja_loocv <- getSampleAdjaMAD(loocv_res$edges_all, n,
+                                annot_node)
```

If the MSE is computed for several values of λ, the minimal MSE determines the best λ. To restrict the range of possible lambda the function "calcRangeLambda" can be used.

```
> lambda <- calcRangeLambda(obs, delta, delta_type)
> MSE <- Inf
> for (lamd in lambda) {
+   loocv_res <- loocv(kfold=NULL, times, obs, delta, lambda=lamd,
```

```
+                       b, n, K, T_=NULL, annot=getEdgeAnnot(n),
+                       annot_node, active_mu, active_sd,
+                       inactive_mu, inactive_sd, mu_type,
+                       delta_type)
+   if (loocv_res$MSE < MSE) {
+                   MSE <- loocv_res$MSE
+                   edges_all <- loocv_res$edges_all
+                   bestLambda <- lamd
+   }
+ }
> adja_bestLambda <- getSampleAdjaMAD(edges_all, n, annot_node)
```

If using time-series data, only the flag_time_series needs to be set to TRUE when executing loocv and calcRangeLambda.

5 K-fold cross-validation

Instead of using LOOCV a k-fold cross-validation can be used to compute the λ which minimizes the MSE.

```
> kfold <- 5
> MSE <- Inf
> for (lamd in lambda) {
+   kcv_res <- kfoldCV(kfold, times, obs, delta, lambda=lamd,
+                       b, n, K, T_=NULL, annot=getEdgeAnnot(n),
+                       annot_node, active_mu, active_sd,
+                       inactive_mu, inactive_sd, mu_type,
+                       delta_type)
+   if (kcv_res$MSE < MSE) {
+                   MSE <- kcv_res$MSE
+                   edges_all <- kcv_res$edges_all
+                   bestLambda <- lamd
+   }
+ }
> adja_bestLambda <- getSampleAdjaMAD(edges_all, n, annot_node)
```

If using time-series data, only the flag_time_series needs to be set to TRUE when executing kfoldCV.

6 Integrating prior knowledge

For the integration of prior knowledge, such as the information of source and sink nodes, this can be used in the LP network inference approach by setting the corresponding parameters.

```
> res3 <- doILP(obs, delta, lambda=1, b, n, K, T_=NULL,
+               annot=getEdgeAnnot(n), delta_type,
+               sourceNode=1, sinkNode=5)
> adja3 <- getAdja(res3,n)
```

Similary, additional constraints defining prior knowledge on individual edges can be formulated and added as a list to the "doILP" function. The prior of each individual edge consists of a vector of four elements with the first element being the annotation of the edge (see function "getEdgeAnnot") and the second being the coefficient of the objective function. The third, respectively the fourth elements correspond to the direction, respectively the right-hand side of the constraint. For example, assume that we know that the edge weight between node 1 and node 2 (w_{12}^{+}) is greater than 1, it is defined as follows:

```
> prior <- list(c("w+_1_2", 1, ">", 1))
> res4 <- doILP(obs,delta, lambda=1, b, n, K, T_=NULL,
+               annot=getEdgeAnnot(n), delta_type,
+               prior=prior)
> adja4 <- getAdja(res4, n)
```

The prior knowledge can be extracted from databases such as KEGG or Reactome. Using, for example, the "KEGGgraph" package [ZW09] in combination with lpNet allows to import already known activations or deactivations between two genes. Thus, this information can be used to define the positive (w^{+}) or the negative weight (w^{-}) of the corresponding edges to be bigger than 1 in the prior constraints of the LP method.
If there is additionally a confidence score given for the edge, this can be taken into account by defining the second element in the prior knowledge list accordingly. Assuming that the edge between node 1 and node 2 has a confidence score of 0.9, this translates into:

```
> prior <- list(c("w+_1_2", 1/0.9, ">", 1))
> res5 <- doILP(obs, delta, lambda=1, b, n, K, T_=NULL,
+               annot=getEdgeAnnot(n), delta_type,
+               prior=prior)
> adja5 <- getAdja(res5, n)
```

Considering a small example based on KEGG where we 1) load a network from KEGG, 2) include non-null edges as prior knowledge, and 3) solve the LP problem:

```
> library("KEGGgraph")
> toyKGML <- system.file("extdata/kgml-ed-toy.xml", package="KEGGgraph")
> toyGraph <- parseKGML2Graph(toyKGML, genesOnly=FALSE)
> adja <- as(toyGraph,"matrix")
> entries <- which(adja!=0, arr.ind=TRUE)
> ### use apply to set the prior from a given adjacency matrix
> myFun <- function(el, sign, confidence, rhs) {
+   prior <- c(sprintf("w+_%s_%s", el[[1]][1], el[[1]][2],
+              adja[el[[1]][1],el[[1]][2]]), confidence, sign, rhs)
+ }
> prior <- lapply(apply(entries,1,list), myFun, ">", 1, 1)
> res5 <- doILP(obs, delta, lambda=1, b, n, K, T_=NULL,
+               annot=getEdgeAnnot(n), delta_type,
+               prior=prior)
> adja5 <- getAdja(res5, n)
```

7 Example on Real Data

In the following we use data from the "nem" package which consists of combinatorial knockdowns in the ERBB signaling pathway [SFL$^+$09, FMT$^+$]. Given are 17 experiments of 16 proteins in the ERBB signaling pathway of trastuzumab resistant breast cancer cells. The experiments include 13 single knockdowns, 3 double knockdowns and one experiment without any perturbation. After the knockdowns, Reverse Phase Protein Array (RPPA) measurments were performed for ten signaling intermediates before and after EGF stimulation. For each experiment, four technical and three biological replicates are given. First, we load the data and preprocess it. For this, we separate it into unstimulated (time=0) and stimulated (time=1) experiments and summarize the different replicate measurements of each experiment by taking the mean.

```
> data("SahinRNAi2008")
> dataStim <- dat.normalized[dat.normalized[ ,17] == 1,-17]
> dataUnstim <- dat.normalized[dat.normalized[ ,17] == 0,-17]
> # summarize replicates; transpose: rows=genes, cols=experiments
> dataSt <- t(summarizeRepl(dataStim, type=mean))
> dataUnst <- t(summarizeRepl(dataUnstim, type=mean))
```

Next, the parameters which are needed for the network inference need to be defined. The perturbation vector describes the given knockdown experiments.

```
> n <- 16 # number of genes
> K <- 16 # number of experiments
> annot <- getEdgeAnnot(n)
> # perturbation vector;       kd of:
> b <- c(0,rep(1,15),          # erbb1
+        0,0,rep(1,14),           # erbb1 & 2
+        0,rep(1,14),0,           # erbb1 & 3
+        1,0,rep(1,13),0,         # erbb2 & 3
+        rep(1,10),0,1,1,1,1,1,   # IGFR
+        rep(1,11),0,1,1,1,1,     # ERalpha
+        rep(1,12),0,1,1,1,       # MYC
+        rep(1,7),0,rep(1,8),     # AKT1
+        rep(1,8),0,rep(1,7),     # MEK1
+        rep(1,5),0,rep(1,10),    # CDK2
+        rep(1,6),0,rep(1,9),     # CDK4
+        rep(1,13),0,1,1,         # CDK6
+        1,1,0,rep(1,13),         # p21
+        1,1,1,0,rep(1,12),       # p27
+        rep(1,4),0,rep(1,11),    # Cyclin D1
+        rep(1,14),0,1)           # Cyclin E1
```

For the computation of the parameters δ_i, we use the unstimulated data of the MOCK control (experiment without any perturbation). For four proteins no measurements are given, and no δ can be computed from the MOCK control. Here, we use the average of all measurements of MOCK in the unstimulated setting.

```
> delta <- as.double(dataUnst[ ,1])
> delta[11:16] <- mean(dataUnst[ ,1], na.rm=T)
```

We infer the network using the LP method with source nodes ERBB1, ERBB2 and ERBB3, and sink node pRB1. The parameter $\lambda = 1.83$ has been identified using LOOCV (not shown due to run time reasons) to be the one with minimum MSE.

```
> resERBB <- doILP(dataSt[ ,-1], delta, lambda=1.83,
+                  b, n, K, T_=NULL, annot,
+                  delta_type, all.pos=FALSE,
```

```
+                   sourceNode=c(1,2,16), sinkNode=10)
> adjaERBB <- getAdja(resERBB,n)
```

References

[FMT+] H. Froehlich, F. Markowetz, A. Tresch, T. Niederberger, C. Bender, M. Maneck, C. Lottaz, and T. Beissbarth, *nem: Nested effects models to reconstruct phenotypic hierarchies*, R package version 2.32.1.

[KK13] Bettina Knapp and Lars Kaderali, *Reconstruction of cellular signal transduction networks using perturbation assays and linear programming*, PLoS ONE **8** (2013), no. 7, e69220.

[Mat13] Marta R. A. Matos, *Network inference : extension of linear programming model for time-series data*, Master's thesis, Department of Informatics, University of Minho, Campus Gualtar, 4710-057 Braga, Portugal, October 2013.

[SFL+09] O. Sahin, H. Froehlich, C. Lobke, U. Korf, S. Burmester, M. Majety, J. Mattern, I. Schupp, C. Chaouiya, D. Thieffry, A. Poustka, S. Wiemann, T. Beissbarth, and D. Arlt, *Modeling ERBB receptor-regulated G1/S transition to find novel targets for de novo trastuzumab resistance*, BMC Syst Biol **3** (2009), 1.

[SPP+05] K. Sachs, O. Perez, D. Pe'er, D. A. Lauffenburger, and G. P. Nolan, *Causal protein-signaling networks derived from multiparameter single-cell data*, Science **308** (2005), no. 5721, 523–529.

[ZW09] J. D. Zhang and S. Wiemann, *KEGGgraph: a graph approach to KEGG PATHWAY in R and bioconductor*, Bioinformatics **25** (2009), no. 11, 1470–1471.