http://research.mssm.edu/tulab/software/irafnet.html

iRafNet Tutorial

Contents

1 Introduction

In this tutorial, we describe how to implement iRafNet [1] in R Cran. iRafNet (Integrative Random Forest based Network Inference) is a random forest based algorithm which integrates heterogeneous data to construct gene regulatory networks (GRN). As shown in figure 1, iRafNet is a two steps algorithm where, first, prior biological information is used to weight regulatory relationships and, then, this information and gene-expression data are utilized by a random-forest based algorithm to derive the final network.

Specifically, for each target gene g_j, regulators are selected via iRafNet. First, for each prior biological data $d \in \{1, \ldots, D\}$, we derive weights measuring the regulatory relationships $\{g_k \rightarrow g_j\}$; then, using expression data, we run random forest to find genes regulating gene g_j. Contrary to the usual random forest algorithm which, at each node, samples a random subset of genes from the entire set of genes; we randomly choose an integer I and we sample genes according to weights $\{w^I_{g_k \rightarrow g_j}\}$. Once that the model is estimated, the final network is derived by ranking potential regulators based on the importance score resulting from iRafNet (further details on weights computation and algorithm implementation can be found in Petralia et al. [1]).

Our package is freely available for download at `http://research.mssm.edu/tulab/Codes.html`. iRafNet package is an extension of the original package Random-Forest available in R Cran [2]. Specifically, the original random-forest's code was modified to sample potential regulators according to weights computed according to other available data rather than randomly. Similarly to the original random forest algorithm, our code is particularly efficient due to the use of C routines called by the main code written in R.

2 System requirements

In order to implement iRafNet it is required R Cran (version $\geq$ 2.5.0) which can be freely downloaded at `http://cran.r-project.org/`.

3 Functions specification

For each target gene g_j, iRafNet models the expression value of g_j as a function of the expression value of potential regulators via random-forest. Let r be the number of potential regulators and n the number of samples. Generally, potential regulators of a target gene g_j consist of any other gene g_k with $k \neq j$; in some cases, the set of potential regulators may be set as a smaller subset based on certain prior knowledge. Let x_j be the vector of expression levels of gene g_j and let z_j be a matrix $(n \times r)$ containing expression levels of potential regulators. Let N be the number of potential regulators to be sampled at each node and T the number of random forest trees. As discussed in Petralia et al. [1], it is standard choice to consider a large number of trees (about $1,000$ trees)

and set N equal to the square root of the number of predictors (in our case potential regulators)[3].

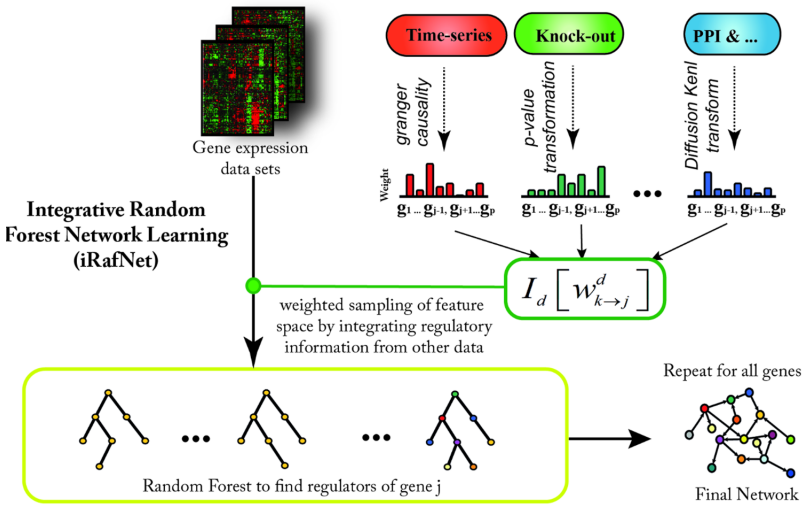

Figure 1: iRafNet algorithm schematic: for each target gene g_j, potential regulators are selected via iRafNet. First, for each prior biological data $d \in \{1, \ldots, D\}$, we derive weights measuring the regulatory relationship $\{g_k \rightarrow g_j\}$; then, using expression data, we run random forest to find genes regulating gene g_j. Contrary to the usual random forest algorithm which, at each node, samples a random subset of genes from the entire set of genes; we randomly choose an integer I and we sample genes according to weights $\{w^I_{g_k \rightarrow g_j}\}$. Once that the model is estimated, the final network is derived by ranking potential regulators based on the importance score resulting from random forest.

3.1 Function `RF`

Description:

`RF` implements iRafNet algorithm. For each target gene g_j, iRafNet models the expression value of g_j as a function of the expression value of potential regulators via random-forest. At each node, potential regulators are sampled based on weights computed considering other D prior biological data.

Usage:

```
Rout<-RF(z_j,x_j,importance=TRUE,mtry=N,ntree=T,
        sw=as.double(w_j),numsource=D)
```

Input:

z_j:	$(n \times r)$ matrix containing expression of potential regulators.
x_j:	$(n \times 1)$ matrix containing expression of target gene g_j.
`importance`:	TRUE if importance scores need to be stored.
`mtry`:	number of potential regulators to be sampled at each node.
`ntree`:	number of trees.
`sw`:	$(r \times D)$ matrix containing sampling weights for each data.
`numsource`:	number of prior data available to compute sampling weights.

3.2 Function `Importance`

Description:

`Importance` computes importance scores resulting from function `RF`. This function provides two importance scores, the mean decrease in accuracy and the mean decrease in node impurity. The outcome of this function will be a matrix $(r \times 2)$ containing the two importance measures for each potential regulator. In particular, we will use the decrease in node impurity as importance score and, therefore, we will only consider the second columns of the resulting matrix. Further details about this function can be found in the package randomForest available in R Cran [2].

Usage:

```
imp<-importance(Rout)[,2]
```

As input, we will need to specify only the outcome of function `RF`. The function returns variable `imp`, a r dimensional vector containing importance score for each potential regulator.

4 Software implementation

This section provides details about all steps necessary to implement iRafNet.

4.1 Compile C libraries

From console execute the following commands:

```
cd .../Code/Random.Forest.multiple.weights/src
R CMD SHLIB regrf.c classTree.c regTree.c rf.c rfutils.c
```

4.2 Load libraries into R Cran

4.2.1 C libraries

From R Cran load C libraries using the command `dyn.load`:

```
setwd("../Code/Random.Forest.multiple.weights/src")
dyn.load("regrf.so")
```

Check if C libraries have been successfully uploaded using the command `is.loaded`:

```
is.loaded("regTree")
```

When `is.loaded` returns `TRUE`, C libraries have been correctly uploaded.

4.2.2 R libraries

Execute the following commands in R Cran to load functions `RF` and `importance`.

```
setwd("../Code/")
source("RF.R")
source("importance.R")
```

4.3 Estimate network using iRafNet

4.3.1 Sampling weights based on one data

Let w_j be a r dimensional vector containing sampling weights corresponding to regulatory relationships $\{g_k \rightarrow g_j\}$. Please, refer to the main manuscript for more information about sampling weights computation. The main algorithm can be summarized via the following steps:

1. Derive w_j the vector of sampling weights based on prior biological data
2. Sort elements of w_j in increasing order
3. Sort columns of matrix z_j to match sampling weights
4. Estimate iRafNet by executing the following command in R Cran

   `Rout<-RF(`z_j`,`x_j`,importance=TRUE,mtry=`N`,ntree=`T`,`
   `sw=as.double(`w_j`),numsource=1)`

5. Compute importance scores executing the following command in R Cran

   ```
   imp<-importance(Rout)[,2]
   ```

The hth element of vector `imp` is the final weight given to the regulatory relationship $g_h \rightarrow g_j$. This score can be used to compute receiving-operating characteristic (ROC) curve or other measures of accuracy.

4.3.2 Sampling weights based on multiple data

Let D be the number of prior biological data available. Let w_j be a $(r \times D)$ matrix of sampling weights based on the D prior biological data. The main algorithm can be summarized in the following steps:

1. Derive w_j the matrix of sampling weights based on the prior biological data.
2. Estimate iRafNet by executing the following command in R Cran

   `Rout<-RF(`z_j`,`x_j`,importance=TRUE,mtry=`N`,ntree=`T`,`
   `sw=as.double(`w_j`),numsource=`D`)`
3. Compute importance scores executing the following command in R Cran

   `imp<-importance(Rout)[,2]`

The hth element of vector `imp` is the final weight given to the regulatory relationship $g_h \rightarrow g_j$.

5 Application: DREAM4 challenge

As an example, we implement iRafNet to construct networks involved in the DREAM4 in-silico size 100 challenge. This challenge consists of five networks involving $p = 100$ genes. For each network, knockout data and time-series gene expression are provided. In particular, time-series data consists of ten different experiments with 21 time points each; while knockout includes wild-type gene expression and gene expression of knocking out each one of the p genes. Computation of sampling weights are derived following the procedure described in Petralia et al. [1].

The R code used to implement iRafNet on DREAM 4 data is available in the `Code` folder as `iRafNetdream4`. Figure 2 and 3 shows the code used to load all necessary packages and the data, consisting of time-series gene expression and expression from knock-out experiments. For each target gene, the number of potential regulators was set to the total number of genes minus one, i.e. 99, and consequently N was set to $\sqrt{99}$. The number of trees was set equal to $1,000$. Figure 4 shows the computation of importance scores for each target gene j. The final outcome `imp` is a $(p \times p)$ matrix containing importance scores for each regulation. In particular, the (h, j) element of `imp` contains the importance score for regulation $g_h \rightarrow g_j$.

Once that importance scores are computed, the resulting weighted networks can be validated via the computation of ROC and precision recall curves. To this end, we use the package ROCR available in R CRAN. As shown in figure 5, we load the set of true regulations and then we arrange the importance scores resulting from iRafNet in the same format. Function `predict.network` is used to construct `imp.final`, a matrix containing for each possible regulation its true status and importance score resulting from iRafNet. In particular, `imp.final` is a $(M \times 4)$ matrix with M being the total number of all possible

regulations. Each row correspond to one particular regulation, e.g. $g_k \rightarrow g_j$, and contains the ID of both gene g_k and g_j, the regulation status (equal to one when `TRUE` and equal to 0 when `FALSE`), and the importance score resulting from iRafNet. Finally, we compute the area under the ROC and precision-recall curves using the package ROCR. In particular, this is done by using function `prediction` which takes as input importance scores and regulation status from matrix `imp.final`.

```
# ----- Load Software    ----------------------------------- #

library(matrixStats)
library(ROCR)

setwd('.../Code/')
source("RF.R")
source("importance.R")

setwd('.../Code/Random.Forest.multiple.weights/src/')
dyn.load("regrf.so") # -- load C library
is.loaded("regTree") # -- check if loaded
```

Figure 2: DREAM4 challenge: load all necessary sofwares to implement iRafNet as described in section 4.

```
# ---------- Load time-series Data     ----------------------------------------------------------#
  net=1; # -- network to infer
  setwd(paste('.../DREAM4 in-silico challenge/Size 100/DREAM4 training data/insilico_size100_',net,sep=""))
  Data<-read.table("data_dream4.csv",sep="\t",header=F)
  yy<-data[,seq(1,100)]
  xx<-data[,seq(101,200)]

# ---------- load Knock-Out data   ----------------------------------------------------------#
  file=paste("insilico_size100_",net,"_knockouts.tsv",sep="")
  KO<-read.table(file,header=T)

  file=paste("insilico_size100_",net,"_wildtype.tsv",sep="")
  w.f<-read.table(file,header=T)

  wild<-kronecker(matrix(1,dim(xx)[2],1), as.matrix(w.f), FUN = "*")
  sigma<-(colVars(KO))
  ko.diff<-abs(KO-wild)/(matrix(1,dim(KO)[1],1)%*%sqrt(sigma));
  p<-dim(KO)[2]; pko<-dim(KO)[1]

  sim.un<-2*(1-pnorm(as.matrix(ko.diff))) #--- weights based on KO data
  sim.un<-(1/sim.un-1); sim.un[sim.un=="Inf"]=max(sim.un[sim.un!="Inf"])
```

Figure 3: DREAM4 challenge: load time-series and knockout data and derive weights based on knockout data as described in Petralia et al. [1].

```
# ----- Standardize data      ------------------------------------------------------------ #
  xx.matrix <- apply(xx, 2, function(x) { (x - mean(x)) / sd(x) } )
  yy.matrix <- apply(yy, 2, function(x) { (x - mean(x)) / sd(x) } )

# ---- for loop over genes ------------------------------------------------------------- #
  imp<-matrix(0,p,p); gene.ord<-imp; trans.id<-colnames(xx[,seq(1,p)])

  for (j in 1:p){
    weights.rf<-rep(0,p)
    y<-yy.matrix[,j]; x<-xx.matrix[,seq(1,p)];

    weights.rf<-as.matrix(sim.un[,j]); weights.rf[j]<-0
    weights.rf<-weights.rf/sum(weights.rf);

    w.sorted<-sort(weights.rf,decreasing = FALSE,index.return=T)
    index<-w.sorted$ix
    x.sorted<-x[,index]
    w.sorted<-w.sorted$x

    rout<-RF(x=x.sorted,y=as.double(y),importance=TRUE,mtry=round(sqrt(p)),ntree=1000,
             sw=as.double(w.sorted),numsource=1L)
    imp[index,j]<-c(importance(rout)[,2])
}
```

Figure 4: DREAM4 challenge: for-loop over target genes. For each target gene g_j, we derive importance scores for regulations $\{g_k \rightarrow g_j\}_{k \neq j}$.

6 Parallelization

iRafNet is based on p independent regression problems where, for each target gene g_j with $j = \{1, ..., p\}$, importance scores for regulations $\{g_k \rightarrow g_j\}_{k \neq j}$ are derived. For this reason, iRafNet can be easily parallelized. Specifically, we need to parallelize the for-loop over target genes in figure 4.

References

[1] Petralia F., Wang P., Yang J. and Tu Z., *A General Framework of Integrative Random Forest for Gene Regulatory Network Inference.*

[2] Liaw A., Wiener M. (2002), *Classification and Regression by randomForest.* R news, 2: 18-22.

[3] Shi T., Horvath S. (2006), *Unsupervised learning with random forest predictors.* Journal of Computational and Graphical Statistics, 15.

```
# ---- Load true network    ------------------------------------------------------------------ #
setwd('.../Codes/DREAM4 in-silico challenge/Size 100/DREAM4 gold standards')
truth<-read.table(paste("insilico_size100_",net,"_goldstandard.tsv",sep=""),header=F)
genes<-paste("G",seq(1,p),sep="");
truth.unique.caused<-unique(truth[,2]);
m.m<-match(truth.unique.caused,genes)

# ---- Rearrange importance scores from iRafNet in the format of the true network ------------- #
colnames(imp)<-colnames(xx)
vec.imp <- c(as.matrix(imp))
g.id1<-kronecker(rep(1,dim(imp)[2]), seq(1,dim(imp)[1]), FUN = "*");
g.id1<-paste("G",g.id1,sep="")
g.id2<-kronecker(match(colnames(imp),genes), rep(1,dim(imp)[1]), FUN = "*");
g.id2<-paste("G",g.id2,sep="")
i<-sort(vec.imp,index.return = TRUE,decreasing=TRUE);
vec.imp<-cbind(g.id1[i$ix],g.id2[i$ix],i$x)

# ---- Validation ------------------------------------------------------------------------------ #
  # -- 1) match true value for each edge with corresponding importance score
setwd('.../Code/')
source("predict_network.R")
imp.final<-predict.network(vec.imp,truth,genes,p)

  # -- 2) ROC computation
pred <- prediction(as.numeric(imp.final[,4]), as.numeric(imp.final[,3]))
perf <- performance(pred,"tpr","fpr") # --- ROC
auc <- performance(pred,"auc")@y.values[[1]]   # --- area under ROC

  # -- 3) Precision-Recall computation
perf <- performance(pred,"prec","rec") # --- precision-recall (PR) curve
prec<-perf@x.values[[1]]; rec<-perf@y.values[[1]];
prec[is.na(prec)]=0; rec[is.na(rec)]<-0
auc.p<-trapz(prec,rec);     # --- area under PR curve
```

Figure 5: DREAM4 challenge: Construct ROC and Precision-Recall cuves to validate networks.