De-sparsifying patient mutations by smoothing over gene interaction networks

Luca Giudice,Shraddha Pai

2020-07-23

# TL;DR

This code block is not evaluated. Need a breakdown? Look at the following sections.

# Introduction

In this example, we classify Testicular Germ Cell Tumors (TGCT) by pathologic stage of the tumor. We first reduce sparsity by using prior knowledge of a gene interaction network. Conceptually, the logic is that if a patient has a mutation in a given gene, the mutation indirectly impacts interacting genes. The indirect mutation is inferred by label propagation over a gene-gene interaction network. The resulting smoothed network is then used for downstream applications. Hofree et al. showed that indirect mutations inferred by such smoothing improved patient clustering in 4 types of cancer.

This workflow is identical to that of using unsmoothed somatic mutations (see vignette 3: XXXXX); the only difference is an additional step to smooth binary mutations.

# Setup

set.seed(8)  
suppressWarnings(suppressMessages(require(netDx)))  
suppressWarnings(suppressMessages(require(MultiAssayExperiment)))

# Data

Both the patient mutation profiles and the associated pathologic stage of their tumor are downloaded from TCGA with the curatedTCGAData R package. Mutations are represented as a binary matrix with rows corresponding to genes and columns to patients; entry [i,j] is set to one if gene i has a somatic mutation at gene level SNV detected with the algorithm MutSig2CV, while zero otherwise.

genoFile <- paste(path.package("netDx"),"extdata",  
 "TGCT\_mutSmooth\_geno.txt",  
 sep=getFileSep())  
print(genoFile)

## [1] "/usr/local/lib/R/site-library/netDx/extdata/TGCT\_mutSmooth\_geno.txt"

geno <- read.delim(genoFile,sep="\t",header=TRUE,as.is=TRUE)  
  
phenoFile <- paste(path.package("netDx"),"extdata",  
 "TGCT\_mutSmooth\_pheno.txt",  
 sep=getFileSep())  
pheno <- read.delim(phenoFile,sep="\t",header=TRUE,as.is=TRUE)

We apply minor modification to finish the matrix with sample names and clinical information:

rownames(pheno) <- pheno$ID  
table(pheno$STATUS)

##   
## EARLY LATE   
## 66 14

# Smooth mutations over a gene interaction network

Load the gene-gene interaction network, downloaded from the NBS python implementation of Huang JK and Ideker T <doi:10.1093/bioinformatics/bty186>: This network is a compact cancer reference network (CRN) that contains only high-confidence interactions specific to cancer. Huang JK et al. showed that the CRN effectively clusters tumor samples of patients distinguishing them by tumor type and time of survival.

netFile <- paste(path.package("netDx"),"extdata",  
 "CancerNets.txt",  
 sep=getFileSep())  
cancerNets <- read.delim(netFile,sep="\t",header=T,as.is=T)  
head(cancerNets[,1:5])

## HSPA2 RPN1 GK2 HSPA6 PPP3R1  
## HSPA2 0 1 1 1 1  
## RPN1 1 0 0 1 0  
## GK2 1 0 0 1 0  
## HSPA6 1 1 1 0 1  
## PPP3R1 1 0 0 1 0  
## DLG1 1 0 0 1 0

Use smoothMutations\_LabelProp() to run label propagation over the mutation matrix, thereby smoothing it. The result of using this strategy on a patient?s binary somatic mutation profile is a non-sparse network-smoothed profile in which the state of each gene is a continuous value which reflects its network proximity to the original mutations.

message("\* Running label prop")

## \* Running label prop

require(doParallel)

## Loading required package: doParallel

## Loading required package: foreach

## Loading required package: iterators

# Start the node clusters for parallel propagation  
cl <- makeCluster(2L)  
registerDoParallel(cl)  
prop\_net <- smoothMutations\_LabelProp(geno,cancerNets,cl,  
 no\_cores=2L)  
stopCluster(cl)

Finally, binarize the smoothed matrix. Only the genes which got a high propagation value are set to one, the others are set to zero. This step ensures that genes which got a low propagation value are not used. More a gene is distant from the original mutations, more its propagation value is low and more is considered unreliable and not involved from them.

#Set the name of the project and future resulting directory  
#Apply binarization  
genoP <- thresholdSmoothedMutations(  
 prop\_net,geno,"TGCT\_CancerNets",c(20)  
 )

# Create pathway-level features with binary patient similarity

Smoothed mutations are now grouped at the level of biological pathways. As with other examples, pathways are downloaded from a compilation of curated pathway databases (.gmt format). Thereafter, we define pathway-level patient similarity to be binary; i.e. if two patients share a mutation in genes from the same pathway, their mutual similarity is one; else it is zero.

For more details on the individual steps below, see the first vignette related to building a binary predictor.

#Setup to build the predictor  
pathwayList <- readPathways(  
 fetchPathwayDefinitions("January",2018)  
 )

## ---------------------------------------

## Fetching http://download.baderlab.org/EM\_Genesets/January\_01\_2018/Human/symbol/Human\_AllPathways\_January\_01\_2018\_symbol.gmt

## File: 2e35f27571\_Human\_AllPathways\_January\_01\_2018\_symbol.gmt

## Read 3028 pathways in total, internal list has 3009 entries

## FILTER: sets with num genes in [10, 200]

## => 971 pathways excluded  
## => 2038 left

# limit num pathways to speed example  
# pathwayList <- pathwayList[sample(1:length(pathwayList),100,replace=FALSE)]  
exprdat <- SummarizedExperiment(genoP, colData=pheno)  
objList <- list(genetic=exprdat)

Now we define functions for patient similarity:

makeNets <- function(dataList,groupList,netDir,numCores,...) {  
 netList <- c(); netList2 <- c()  
   
 # create genetic nets  
 if (!is.null(groupList[["genetic"]])) {  
 netList <- makeMutNets(dataList[["genetic"]],  
 groupList[["genetic"]],  
 netDir,numC=numCores)  
 }  
 cat(sprintf("\t%i genetic-pathway nets\n", length(netList)))  
 cat(sprintf("Total of %i nets\n", length(netList)))  
   
 return(netList)  
}  
  
# g geno matrix, genes by patients (columns) - binary  
# pList list of genesets  
# outDir - dir where nets are to be written  
makeMutNets <- function(g,pList,oDir,numC) {  
 g <- t(g) # transpose to have genes as columns  
 cl <- makeCluster(numC)  
 registerDoParallel(cl)  
   
 numPat <- c()  
 netList <- foreach(k=1:length(pList)) %do% {  
 idx <- which(colnames(g) %in% pList[[k]])  
   
 if (length(idx)>0) {  
 has\_mut <- rowSums(g[,idx,drop=FALSE])  
 has\_mutp <- names(has\_mut)[which(has\_mut>0)]  
   
 if (length(has\_mutp)>=6) {  
 ##cat(sprintf("%s: %i patients\n", names(pList)[k],  
 ## length(has\_mutp)))  
 #numPat <- c(numPat, length(has\_mutp))  
 pat\_pairs <- t(combinat::combn(has\_mutp,2));  
 pat\_pairs <- cbind(pat\_pairs,1);  
 outFile <- sprintf("%s/%s\_cont.txt",oDir,names(pList)[k])  
 write.table(pat\_pairs, file=outFile,sep="\t",  
 col=FALSE,row=FALSE,quote=FALSE)  
 basename(outFile)  
 } else NULL  
 } else {  
 NULL  
 }  
 }  
 stopCluster(cl)  
 unlist(netList)  
}

# Build predictor

Finally, we compile all the data into a MultiAssayExperiment object and as before, run the predictor.

exprdat <- SummarizedExperiment(genoP, colData=pheno)  
objList <- list(genetic=exprdat)  
  
groupList <- list()  
groupList[["genetic"]] <- pathwayList #names for groupList and objList now match  
  
dataList <- MultiAssayExperiment(objList,pheno)

The predictor call is essentially the same as with other simpler designs:

outDir <- paste(tempdir(),"pred\_output",sep=getFileSep())  
#Run the predictor as usual  
out <- suppressMessages(  
 buildPredictor(dataList=dataList,groupList=groupList,  
 makeNetFunc=makeNets, ### custom network creation function  
 outDir=outDir, ## absolute path  
 numCores=1L, featScoreMax=2L, featSelCutoff=2L,  
 numSplits=2L,logging="none"  
))

## Warning in normalizePath(outDir): path[1]="/tmp/Rtmp6pbeiY/pred\_output": No such  
## file or directory

## 1915 genetic-pathway nets  
## Total of 1915 nets  
## 5 genetic-pathway nets  
## Total of 5 nets  
## 205 genetic-pathway nets  
## Total of 205 nets  
## 1909 genetic-pathway nets  
## Total of 1909 nets  
## 12 genetic-pathway nets  
## Total of 12 nets  
## 203 genetic-pathway nets  
## Total of 203 nets

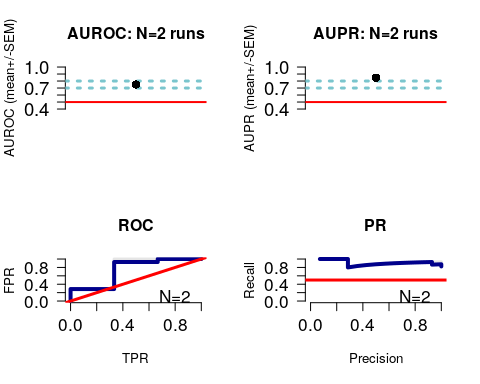
# Examine output

This code collects different components of model output to examine the results.

numSplits <- 2L  
st <- unique(colData(dataList)$STATUS)  
acc <- c() # accuracy  
predList <- list() # prediction tables  
  
featScores <- list() # feature scores per class  
for (cur in unique(st)) featScores[[cur]] <- list()  
  
for (k in 1:numSplits) {   
 pred <- out[[sprintf("Split%i",k)]][["predictions"]];  
 # predictions table  
 tmp <- pred[,c("ID","STATUS","TT\_STATUS","PRED\_CLASS",  
 sprintf("%s\_SCORE",st))]  
 predList[[k]] <- tmp   
 # accuracy  
 acc <- c(acc, sum(tmp$PRED==tmp$STATUS)/nrow(tmp))  
 # feature scores  
 for (cur in unique(st)) {  
 tmp <- out[[sprintf("Split%i",k)]][["featureScores"]][[cur]]  
 colnames(tmp) <- c("PATHWAY\_NAME","SCORE")  
 featScores[[cur]][[sprintf("Split%i",k)]] <- tmp  
 }  
}

Plot the AUROC and AUPR curves:

predPerf <- plotPerf(predList, predClasses=st)



Examine features with the highest scores. Here, these are pathways with somatic mutations that best predict vital status:

featScores2 <- lapply(featScores, getNetConsensus)  
summary(featScores2)

## Length Class Mode  
## EARLY 3 data.frame list  
## LATE 3 data.frame list

featSelNet <- lapply(featScores2, function(x) {  
 callFeatSel(x, fsCutoff=1, fsPctPass=0)  
})  
print(head(featScores2[["LATE"]]))

## PATHWAY\_NAME  
## 1 1D-\_I\_MYO\_\_I\_-INOSITOL\_HEXAKISPHOSPHATE\_BIOSYNTHESIS\_II\_\_MAMMALIAN\_\_cont.txt  
## 2 3-PHOSPHOINOSITIDE\_BIOSYNTHESIS\_cont.txt  
## 3 3-PHOSPHOINOSITIDE\_DEGRADATION\_cont.txt  
## 4 ABC\_TRANSPORTERS\_IN\_LIPID\_HOMEOSTASIS\_cont.txt  
## 5 ABORTIVE\_ELONGATION\_OF\_HIV-1\_TRANSCRIPT\_IN\_THE\_ABSENCE\_OF\_TAT\_cont.txt  
## 6 ACTIVATED\_PKN1\_STIMULATES\_TRANSCRIPTION\_OF\_AR\_\_ANDROGEN\_RECEPTOR\_\_REGULATED\_GENES\_KLK2\_AND\_KLK3\_cont.txt  
## Split1 Split2  
## 1 1 NA  
## 2 2 2  
## 3 NA 2  
## 4 1 NA  
## 5 1 NA  
## 6 2 2

# Session Info

sessionInfo()

## R version 4.0.0 (2020-04-24)  
## Platform: x86\_64-pc-linux-gnu (64-bit)  
## Running under: Ubuntu 18.04.4 LTS  
##   
## Matrix products: default  
## BLAS/LAPACK: /usr/lib/x86\_64-linux-gnu/libopenblasp-r0.2.20.so  
##   
## locale:  
## [1] LC\_CTYPE=C LC\_NUMERIC=C   
## [3] LC\_TIME=C LC\_COLLATE=C   
## [5] LC\_MONETARY=C LC\_MESSAGES=C   
## [7] LC\_PAPER=en\_US.UTF-8 LC\_NAME=C   
## [9] LC\_ADDRESS=C LC\_TELEPHONE=C   
## [11] LC\_MEASUREMENT=en\_US.UTF-8 LC\_IDENTIFICATION=C   
##   
## attached base packages:  
## [1] stats4 parallel stats graphics grDevices utils datasets   
## [8] methods base   
##   
## other attached packages:  
## [1] doParallel\_1.0.15 iterators\_1.0.12   
## [3] foreach\_1.5.0 MultiAssayExperiment\_1.14.0  
## [5] SummarizedExperiment\_1.18.1 DelayedArray\_0.14.0   
## [7] matrixStats\_0.56.0 GenomicRanges\_1.40.0   
## [9] GenomeInfoDb\_1.24.2 IRanges\_2.22.2   
## [11] S4Vectors\_0.26.1 netDx\_1.1.4   
## [13] bigmemory\_4.5.36 Biobase\_2.48.0   
## [15] BiocGenerics\_0.34.0 BiocManager\_1.30.10   
##   
## loaded via a namespace (and not attached):  
## [1] uuid\_0.1-4 BiocFileCache\_1.12.0   
## [3] NMF\_0.22.0 plyr\_1.8.6   
## [5] igraph\_1.2.5 RCy3\_2.8.0   
## [7] lazyeval\_0.2.2 splines\_4.0.0   
## [9] entropy\_1.2.1 BiocParallel\_1.22.0   
## [11] rncl\_0.8.4 ggplot2\_3.3.2   
## [13] gridBase\_0.4-7 scater\_1.16.2   
## [15] digest\_0.6.25 htmltools\_0.5.0   
## [17] viridis\_0.5.1 magrittr\_1.5   
## [19] memoise\_1.1.0 cluster\_2.1.0   
## [21] ROCR\_1.0-11 limma\_3.44.3   
## [23] annotate\_1.66.0 R.utils\_2.9.2   
## [25] prettyunits\_1.1.1 colorspace\_1.4-1   
## [27] blob\_1.2.1 rappdirs\_0.3.1   
## [29] xfun\_0.15 dplyr\_1.0.0   
## [31] crayon\_1.3.4 RCurl\_1.98-1.2   
## [33] bigmemory.sri\_0.1.3 graph\_1.66.0   
## [35] genefilter\_1.70.0 phylobase\_0.8.10   
## [37] survival\_3.2-3 ape\_5.4   
## [39] glue\_1.4.1 registry\_0.5-1   
## [41] gtable\_0.3.0 zlibbioc\_1.34.0   
## [43] XVector\_0.28.0 BiocSingular\_1.4.0   
## [45] kernlab\_0.9-29 Rhdf5lib\_1.10.0   
## [47] shape\_1.4.4 SingleCellExperiment\_1.10.1  
## [49] HDF5Array\_1.16.1 scales\_1.1.1   
## [51] DBI\_1.1.0 edgeR\_3.30.3   
## [53] rngtools\_1.5 bibtex\_0.4.2.2   
## [55] Rcpp\_1.0.4.6 viridisLite\_0.3.0   
## [57] xtable\_1.8-4 progress\_1.2.2   
## [59] bit\_1.1-15.2 rsvd\_1.0.3   
## [61] glmnet\_4.0-2 netSmooth\_1.8.0   
## [63] httr\_1.4.1 RColorBrewer\_1.1-2   
## [65] ellipsis\_0.3.1 pkgconfig\_2.0.3   
## [67] XML\_3.99-0.3 R.methodsS3\_1.8.0   
## [69] dbplyr\_1.4.4 locfit\_1.5-9.4   
## [71] RJSONIO\_1.3-1.4 howmany\_0.3-1   
## [73] tidyselect\_1.1.0 rlang\_0.4.6   
## [75] softImpute\_1.4 reshape2\_1.4.4   
## [77] AnnotationDbi\_1.50.1 munsell\_0.5.0   
## [79] tools\_4.0.0 generics\_0.0.2   
## [81] RSQLite\_2.2.0 ade4\_1.7-15   
## [83] evaluate\_0.14 stringr\_1.4.0   
## [85] yaml\_2.2.1 knitr\_1.29   
## [87] bit64\_0.9-7 purrr\_0.3.4   
## [89] nlme\_3.1-148 R.oo\_1.23.0   
## [91] pracma\_2.2.9 xml2\_1.3.2   
## [93] compiler\_4.0.0 beeswarm\_0.2.3   
## [95] curl\_4.3 tibble\_3.0.1   
## [97] RNeXML\_2.4.5 stringi\_1.4.6   
## [99] RSpectra\_0.16-0 lattice\_0.20-41   
## [101] Matrix\_1.2-18 vctrs\_0.3.1   
## [103] pillar\_1.4.4 lifecycle\_0.2.0   
## [105] combinat\_0.0-8 zinbwave\_1.10.0   
## [107] BiocNeighbors\_1.6.0 data.table\_1.12.8   
## [109] bitops\_1.0-6 irlba\_2.3.3   
## [111] R6\_2.4.1 gridExtra\_2.3   
## [113] vipor\_0.4.5 codetools\_0.2-16   
## [115] MASS\_7.3-51.6 assertthat\_0.2.1   
## [117] rhdf5\_2.32.1 pkgmaker\_0.31.1   
## [119] withr\_2.2.0 GenomeInfoDbData\_1.2.3   
## [121] locfdr\_1.1-8 hms\_0.5.3   
## [123] grid\_4.0.0 tidyr\_1.1.0   
## [125] rmarkdown\_2.3 DelayedMatrixStats\_1.10.0   
## [127] clusterExperiment\_2.8.0 ggbeeswarm\_0.6.0