netDx: Building interpretable patient classifiers by multi-'omic data integration and patient similarity networks

Shraddha Pai1, Philipp Weber2, Ruth Isserlin1, Hussam Kaka1, Shirley Hui1, Muhammad Ahmad Shah1, Anne Krogh Nøhr3, Jan Baumbach2,4, Gary D. Bader1,5,6,7

**Affiliations**

1. The Donnelly Centre, University of Toronto, Toronto, Canada
2. Department of Mathematics and Computer Science, University of Southern Denmark, Odense, Denmark
3. Anne affiliation?
4. TUM School of Life Sciences Weihenstephan, Technical University of Munich, Munich, Germany.
5. Department of Molecular Genetics, University of Toronto, Toronto, Canada
6. Department of Computer Science, University of Toronto, Toronto, Canada
7. The Lunenfeld-Tanenbaum Research Institute, Mount Sinai Hospital, Toronto, Canada

**Contributions**

SP developed the R implementation and wrote the paper

PW altered the GeneMANIA implementation to provide memory improvements for netDx

AKN wrote the Monte Carlo resampling code

RI, HK and SH implemented initial versions of the software

JB supervised PW

GDB supervised the project and wrote the paper

# Abstract

Patient classification based on diagnostics, prognostics, and treatment response, will further the goal of precision medicine. Interpretability is of particular relevance for models based on genomic data, where samples sizes are relatively small (hundreds) and the risk of ungeneralizable models is higher1. netDx is an algorithm that uses machine learning to integrate multi-modal patient data and build a patient classifier2. Patient data are converted into networks of patient similarity, a paradigm conceptually analogous to medical diagnosis. Features passing selection are integrated, and new patients are assigned to the class with the greatest profile similarity. netDx has excellent performance, outperforming most machine-learning methods in binary cancer survival prediction2. Importantly, it handles missing data – a common problem in real-world data – without requiring imputation. netDx also has excellent intepretability, with native support to group genes into pathways for mechanistic insight into predictive features.

The netDx BioConductor package provides multiple workflows for users to build custom patient classifiers. It provides turnkey functions for one-step predictor generation from multi-modal data, including feature selection over multiple train/test data splits. Workflows offer versatility and speed with custom feature design, custom choice of similarity metric, and parallel execution. Built-in functions and examples allow users to compute model performance metrics such as AUROC, AUPR, and accuracy. netDx integrates with Cytoscape and RCy3 for network-based visualization of top-scoring pathways and the final integrated patient network. Advanced users can build more complex predictor designs with functional building blocks used in the simple design. Finally, netDx provides a novel workflow for pathway-based patient classification from sparse genetic data.

# Introduction

[Need broader intro sentence] . netDx is a supervised learning algorithm that classifies patients by integrating multimodal patient data1. It is notable among machine learning methods for handling missing data, and excels at interpretability by allowing users to create biologically-meaningful grouping of features, such as grouping genes into pathway-level features. netDx integrates multi-modal data by converting each layer into a patient similarity network (Figure 1a). Patient similarity networks are conceptually analogous to medical diagnosis, and may help preserve patient privacy{Pai, 2018 #14}.

The details of the netDx algorithm and performance have been discussed in a previous publication{Pai, 2019 #13}; only a conceptual summary is briefly provided here (Figure 1b). As input, netDx takes one table per data layer, known patient labels, and rules used to group data into features. Consider an application of predicting good or poor survival from gene expression, DNA methylation and proteomic data. In this instance, the software is provided with three data tables, one per ‘omic data type, a table with patient ID and known labels, and the grouping rule that one feature is to be created per data layer. Patients are then split into training and test samples and feature selection is performed using training samples. netDx uses grouping rules to convert patient data into a common space of patient similarity networks1,2. A feature may comprise of an entire data layer, of a single variable, or of specified groupings, such as that of genes into pathways. Feature selection is performed once per patient label, and features passing selection are used to classify patients from the held-out test sample. Performance robustness is evaluated by repeating this feature selection and classification exercise on multiple train/test samples. The final model is created by choosing features that consistently score highly.

# Software Workflow

The current version of netDx is integrated into the BioConductor system, a high-quality computational biology software framework for genomic data analysis in R3. Figure 2 shows the workflow for building a model using the netDx software package. netDx also now uses BioConductor data structures and mechanisms for fetching and storing data, and representation of input data. Patient data is now provided to netDx using the *MultiAssayExperiment* class, a data structure and class used to represent multi-’omics experiments associated with a set of samples. In our usage, each data layer is an entry in the named *assays()* slot of the object (e.g. “mRNA”, “proteomics”); the sole exception is clinical data, which is provided as part of the sample metadata, in the *colData()* slot. Grouping rules are provided as a nested *list* object - or list-of-lists (*groupList*). The outer list consists of data layers and inner list, of groupings for the corresponding layer. *assays()* and *groupList* must have matching names.

Binary classifiers using the above design are built using the wrapper function, *buildPredictor()*. This function runs feature selection and classification over a specified number of train/test splits, and returns all associated feature scores and detailed classification results in a *list* object. This function allows the user to provide an arbitrary number of data layers as well as custom rules for defining patient similarity and feature generation, in addition to allowing the user to set parameters that affect degree of overfitting. Advanced users can create custom predictor designs by combining the individual steps used in *buildPredictor()* (Table 1).

# Use Cases

This section shows three use cases for building predictors with netDx. The first uses pathway-level features from gene expression to generate a binary classifier. The second performs three-way classification by integrating gene expression, DNA methylation and proteomic data. The third builds a binary classifier from sparse genetic data.

## Use Case 1: Build a binary classifier with pathway-level features and visualize results

In this example, we will build a binary breast tumour classifier from clinical data and gene expression data. For this we will use breast cancer profiles generated as part of the Cancer Genome Atlas4 and provided in BioConductor as part of the curatedTCGAData package. We will use different rules to create features for each data layer. Specifically:

* Clinical data: Features are defined directly at the level of *variables*; similarity is defined by normalized difference.
* Gene expression data: Features are defined at the level of *pathways*; similarity is defined by pairwise Pearson correlation.

Feature scoring is performed over multiple random splits of the data into train and blind test partitions. Feature selected networks are those that consistently score highly across the multiple splits (e.g. those that score 9 out of 10 in >=70% of splits).

Conceptually, this is what the higher-level logic looks like for a cross-validation design. In the pseudocode example below, the predictor runs for 100 train/test splits. Within a split, features are scored from 0 to 10. Features scoring >=9 are used to predict labels on the held-out test set (20%).

*(Note: these aren’t real function calls; this block just serves to illustrate the concept of the design for our purposes)*

numSplits <- 100 # num times to split data into train/blind test samples  
featScoreMax <- 10 # num folds for cross-validation, also max score for a network  
featSelCutoff <- 9  
netScores <- list() # collect <numSplits> set of netScores  
perf <- list() # collect <numSplits> set of test evaluations  
  
for k in 1:numSplits  
 [train, test] <- splitData(80:20) # split data using RNG seed  
 featScores[[k]] <- scoreFeatures(train, featScoreMax)  
 topFeat[[k]] <- applyFeatCutoff(featScores[[k]])  
 perf[[k]] <- collectPerformance(topFeat[[k]], test)  
end

### Setup

suppressWarnings(suppressMessages(require(netDx)))

### Data

The goal is to classify a breast tumour into either a Luminal A subtype or otherwise. The predictor will integrate clinical variables selected by the user, along with gene expression data. Here we load the required packages and download clinical and gene expression data.

suppressMessages(library(curatedTCGAData))

## Warning: package 'SummarizedExperiment' was built under R version 3.6.1

## Warning: package 'BiocParallel' was built under R version 3.6.1

Take a look at the available data without downloading any:

curatedTCGAData(diseaseCode="BRCA", assays="\*",dry.run=TRUE)

## Title DispatchClass  
## 31 BRCA\_CNASeq-20160128 Rda  
## 32 BRCA\_CNASNP-20160128 Rda  
## 33 BRCA\_CNVSNP-20160128 Rda  
## 35 BRCA\_GISTIC\_AllByGene-20160128 Rda  
## 36 BRCA\_GISTIC\_Peaks-20160128 Rda  
## 37 BRCA\_GISTIC\_ThresholdedByGene-20160128 Rda  
## 39 BRCA\_Methylation\_methyl27-20160128\_assays H5File  
## 40 BRCA\_Methylation\_methyl27-20160128\_se Rds  
## 41 BRCA\_Methylation\_methyl450-20160128\_assays H5File  
## 42 BRCA\_Methylation\_methyl450-20160128\_se Rds  
## 43 BRCA\_miRNASeqGene-20160128 Rda  
## 44 BRCA\_mRNAArray-20160128 Rda  
## 45 BRCA\_Mutation-20160128 Rda  
## 46 BRCA\_RNASeq2GeneNorm-20160128 Rda  
## 47 BRCA\_RNASeqGene-20160128 Rda  
## 48 BRCA\_RPPAArray-20160128 Rda

We will work only with the mRNA data in this example:

brca <- suppressMessages(curatedTCGAData("BRCA",c("mRNAArray"),FALSE))

This next code block prepares the TCGA data. In practice you would do this once, and save the data before running netDx, but we run it here to see an end-to-end example.

staget <- sub("[abcd]","",sub("t","",colData(brca)$pathology\_T\_stage))  
staget <- suppressWarnings(as.integer(staget))  
colData(brca)$STAGE <- staget  
  
pam50 <- colData(brca)$PAM50.mRNA  
pam50[which(!pam50 %in% "Luminal A")] <- "notLumA"  
pam50[which(pam50 %in% "Luminal A")] <- "LumA"  
colData(brca)$pam\_mod <- pam50  
  
idx <- union(which(pam50 == "Normal-like"), which(is.na(staget)))  
pID <- colData(brca)$patientID  
tokeep <- setdiff(pID, pID[idx])  
brca <- brca[,tokeep,]  
  
# remove duplicate assays mapped to the same sample  
smp <- sampleMap(brca)  
samps <- smp[which(smp$assay=="BRCA\_mRNAArray-20160128"),]  
notdup <- samps[which(!duplicated(samps$primary)),"colname"]  
brca[[1]] <- suppressMessages(brca[[1]][,notdup])

## harmonizing input:  
## removing 63 sampleMap rows with 'colname' not in colnames of experiments

The important thing is to create ID and STATUS columns in the sample metadata table. netDx uses these to get the patient identifiers and labels, respectively.

pID <- colData(brca)$patientID  
colData(brca)$ID <- pID  
colData(brca)$STATUS <- colData(brca)$pam\_mod

### Design custom patient similarity networks (features)

netDx allows the user to define a custom function that takes patient data and variable groupings as input, and returns a set of patient similarity networks (PSN) as output. The user can customize what datatypes are used, how they are grouped, and what defines patient similarity for a given datatype.

When running the predictor (next section), the user simply passes this custom function as an input variable; i.e. the makeNetFunc parameter when calling buildPredictor().

***Note:*** While netDx provides a high degree of flexibility in achieving your design of choice, it is up to the user to ensure that the design, i.e. the similarity metric and variable groupings, is appropriate for your application. Domain knowledge is almost likely required for good design.

netDx requires that this function take some generic parameters as input. These include:

* dataList: the patient data, provided as a MultiAssayExperiment object. Refer to the [tutorials for MultiAssayExperiment](https://bioconductor.org/packages/release/bioc/html/MultiAssayExperiment.html) to see how to construct those objects from data.
* groupList: sets of input data that would correspond to individual networks (e.g. genes grouped into pathways)
* netDir: the directory where the resulting PSN would be stored.

#### dataList

Here the BRCA data is already provided to us as a MultiAssayExperiment object, so there is nothing more to do.

summary(brca)

#### groupList

This object tells the predictor how to group units when constructing a network. For examples, genes may be grouped into a network representing a pathway. This object is a list; the names match those of dataList while each value is itself a list and reflects a potential network.

groupList <- list()  
  
# genes in mRNA data are grouped by pathways  
pathList <- readPathways(fetchPathwayDefinitions("January",2018))

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

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

## File: 182107f6006ac\_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

groupList[["BRCA\_mRNAArray-20160128"]] <- pathList[1:3]  
# clinical data is not grouped; each variable is its own feature  
groupList[["clinical"]] <- list(  
 age="patient.age\_at\_initial\_pathologic\_diagnosis",  
 stage="STAGE"  
)

So the groupList variable has one entry per data *layer*:

summary(groupList)

## Length Class Mode  
## BRCA\_mRNAArray-20160128 3 -none- list  
## clinical 2 -none- list

Each entry contains a list, with one entry per feature. Here we have 3 pathway-level features for mRNA and two variable-level features for clinical data.

For example, here are the networks to be created with RNA data. Genes corresponding to pathways are to be grouped into individual network. Such a groupList would create pathway-level networks:

groupList[["BRCA\_mRNAArray-20160128"]][1:3]

## $UREA\_CYCLE  
## [1] "SLC25A15" "CPS1" "ASL" "ARG2" "SLC25A2" "OTC"   
## [7] "NMRAL1" "NAGS" "ASS1" "ARG1"   
##   
## $`CDP-DIACYLGLYCEROL\_BIOSYNTHESIS\_I`  
## [1] "AGPAT1" "GPD2" "ABHD5" "GPAT2" "CDS1" "LPCAT3" "LPCAT4"  
## [8] "CDS2" "AGPAT6" "AGPAT5" "MBOAT7" "AGPAT9" "LCLAT1" "MBOAT2"  
## [15] "AGPAT4" "GPAM" "AGPAT3" "AGPAT2"  
##   
## $`SUPERPATHWAY\_OF\_D-\_I\_MYO\_\_I\_-INOSITOL\_\_1,4,5\_-TRISPHOSPHATE\_METABOLISM`  
## [1] "IPMK" "INPP5B" "INPP5F" "INPP5D" "MINPP1" "INPP5A" "ITPKA"   
## [8] "OCRL" "ITPKC" "ITPKB" "SYNJ2" "INPP5J" "INPP5K" "PTEN"   
## [15] "IMPA2" "INPP1" "SYNJ1" "INPPL1" "IMPA1" "IMPAD1"

For clinical data, we want to keep each variable as its own network:

head(groupList[["clinical"]])

## $age  
## [1] "patient.age\_at\_initial\_pathologic\_diagnosis"  
##   
## $stage  
## [1] "STAGE"

### Define patient similarity for each network

This function is defined by the user and tells the predictor how to create networks from the provided input data.

This function requires dataList,groupList, and netDir as input variables. The residual ... parameter is to pass additional variables to makePSN\_NamedMatrix(), notably numCores (number of parallel jobs).

In this particular example, the custom similarity function does the following:

1. Creates *pathway-level networks from RNA* data using the default Pearson correlation measure makePSN\_NamedMatrix(writeProfiles=TRUE,...)
2. Creates *variable-level networks from clinical* data using a custom similarity function of normalized difference: makePSN\_NamedMatrix(writeProfiles=FALSE,simMetric="custom",customFunc=normDiff).

makeNets <- function(dataList, groupList, netDir,...) {  
 netList <- c() # initialize before is.null() check  
 # make RNA nets (NOTE: the check for is.null() is important!)  
 # (Pearson correlation)  
 if (!is.null(groupList[["BRCA\_mRNAArray-20160128"]])) {   
 netList <- makePSN\_NamedMatrix(dataList[["BRCA\_mRNAArray-20160128"]],  
 rownames(dataList[["BRCA\_mRNAArray-20160128"]]),  
 groupList[["BRCA\_mRNAArray-20160128"]],  
 netDir,verbose=FALSE,   
 writeProfiles=TRUE,...)   
 }  
   
 # make clinical nets (normalized difference)  
 netList2 <- c()  
 if (!is.null(groupList[["clinical"]])) {  
 netList2 <- makePSN\_NamedMatrix(dataList$clinical,   
 rownames(dataList$clinical),  
 groupList[["clinical"]],netDir,  
 simMetric="custom",customFunc=normDiff, # custom function  
 writeProfiles=FALSE,  
 sparsify=TRUE,verbose=TRUE,...)  
 }  
 netList <- c(unlist(netList),unlist(netList2))  
 return(netList)  
}

**Note:** dataList and groupList are generic containers that can contain whatever object the user requires to create PSN. **The custom function gives the user complete flexibility in feature design**.

### Build predictor

Finally we call the function that runs the netDx predictor. We provide:

* number of train/test splits over which to collect feature scores and average performance: numSplits,
* maximum score for features in one round of feature selection (featScoreMax, set to 10)
* threshold to call feature-selected networks for each train/test split (featSelCutoff); only features scoring this value or higher will be used to classify test patients, and
* the information to create the PSN, including patient data (dataList), how variables are to be grouped into networks (groupList) and the custom function to generate features (makeNetFunc).

Change numCores to match the number of cores available on your machine for parallel processing.

The call below runs 2 train/test splits. Within each split, it:

* splits data into train/test using the default split of 80:20
* score2 networks between 0 to 2 (i.e. featScoreMax=2)
* uses networks that score >=1 out of 2 (featSelCutoff) to classify test samples for that split.

These are unrealistically low values set so the example will run fast. In practice a good starting point is featScoreMax=10, featSelCutoff=9 and numSplits=100, but these parameters depend on the sample sizes in the dataset and heterogeneity of the samples.

set.seed(42) # make results reproducible  
outDir <- sprintf("%s/pred\_output",tempdir()) # location for intermediate work  
# set keepAllData=TRUE to not delete at the end of the predictor run.  
# This can be useful for debugging.  
out <- suppressMessages(buildPredictor(dataList=brca,groupList=groupList,  
 makeNetFunc=makeNets,outDir=outDir,  
 numSplits=2L,featScoreMax=2L, featSelCutoff=1L,  
 numCores=1L))

### Examine output

The results are stored in the list object returned by the buildPredictor() call. This list contains:

* inputNets: all input networks that the model started with.
* Split<i>: a list with results for each train-test split
  + predictions: real and predicted labels for test patients
  + accuracy: percent accuracy of predictions
  + featureScores: feature scores for each label (list with g entries, where g is number of patient labels). Each entry contains the feature selection scores for the corresponding label.
  + featureSelected: vector of features that pass feature selection. List of length g, with one entry per label.

summary(out)

## Length Class Mode   
## inputNets 10 -none- character  
## Split1 4 -none- list   
## Split2 4 -none- list

summary(out$Split1)

## Length Class Mode   
## featureScores 2 -none- list   
## featureSelected 2 -none- list   
## predictions 2692 data.frame list   
## accuracy 1 -none- numeric

### Reformat results for further analysis

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

numSplits <- 2  
st <- unique(colData(brca)$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  
 }  
}

### Compute model performance

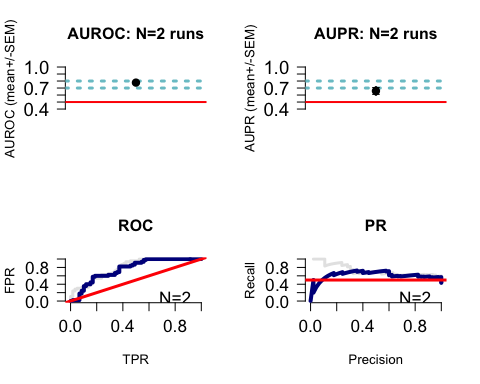
After compiling the data above, plot accuracy for each train/test split:

print(acc)

## [1] 0.6952381 0.7019231

Create a ROC curve, a precision-recall curve, and plot average AUROC and AUPR:

predPerf <- plotPerf(predList, predClasses=st)



### Examine feature scores and consistently high-scoring features

Use getNetConsensus() to convert the list data structure into a single table, one per patient label. The rows show train/test splits and the columns show features that consistently perform well.

We then use callFeatSel() to identify features that consistently perform well across the various train/test splits. Because this is a toy example, we set the bar very low to get some features. Here we accept a feature if it scores 1 or higher (fsCutoff=1) in even one split (fsPctPass=0.05), setting the latter to a low positive fraction.

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

## Length Class Mode  
## LumA 3 data.frame list  
## notLumA 3 data.frame list

head(featScores2[["LumA"]])

## PATHWAY\_NAME  
## 1 CDP-DIACYLGLYCEROL\_BIOSYNTHESIS\_I.profile  
## 2 SUPERPATHWAY\_OF\_D-\_I\_MYO\_\_I\_-INOSITOL\_\_1,4,5\_-TRISPHOSPHATE\_METABOLISM.profile  
## 3 UREA\_CYCLE.profile  
## 4 age\_cont.txt  
## 5 stage\_cont.txt  
## Split1 Split2  
## 1 2 2  
## 2 2 2  
## 3 2 2  
## 4 NA 1  
## 5 1 NA

In practice, a recommended setting is fsCutoff=9 and fsPctPass=0.7 to get features that score at least 9 (out of 10) in at least 70% of the train/test splits.

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

## PATHWAY\_NAME  
## 1 CDP-DIACYLGLYCEROL\_BIOSYNTHESIS\_I.profile  
## 2 SUPERPATHWAY\_OF\_D-\_I\_MYO\_\_I\_-INOSITOL\_\_1,4,5\_-TRISPHOSPHATE\_METABOLISM.profile  
## 3 UREA\_CYCLE.profile  
## 4 age\_cont.txt  
## 5 stage\_cont.txt  
## Split1 Split2  
## 1 2 2  
## 2 2 2  
## 3 2 2  
## 4 NA 1  
## 5 1 NA

print(head(featScores2[["NotLumA"]]))

## NULL

### Visualize EnrichmentMap

An EnrichmentMap5 is a network-based visualization of pathway connectivity and is used in netDx to visualize themes in predictive pathway-based features. It is used in conjunction with AutoAnnotate6 to identify clusters, and apply auto-generated labels to these. For more information, see the [EnrichmentMap](https://www.baderlab.org/Software/EnrichmentMap) website at baderlab.org.

Use getEMapInput\_many() to create the input that helps generate the EnrichmentMap in Cytoscape.

Emap\_res <- getEMapInput\_many(featScores2,pathList,  
 minScore=1,maxScore=2,pctPass=0,out$inputNets,verbose=FALSE)

Write the results to files that Cytoscape can read in:

gmtFiles <- list()  
nodeAttrFiles <- list()  
  
for (g in names(Emap\_res)) {  
 outFile <- sprintf("%s/%s\_nodeAttrs.txt",outDir,g)  
 write.table(Emap\_res[[g]][["nodeAttrs"]],file=outFile,  
 sep="\t",col=TRUE,row=FALSE,quote=FALSE)  
 nodeAttrFiles[[g]] <- outFile  
  
 outFile <- sprintf("%s/%s.gmt",outDir,g)  
 conn <- base::file(outFile,"w")  
 tmp <- Emap\_res[[g]][["featureSets"]]  
 gmtFiles[[g]] <- outFile  
  
 for (cur in names(tmp)) {  
 curr <- sprintf("%s\t%s\t%s", cur,cur,  
 paste(tmp[[cur]],collapse="\t"))  
 writeLines(curr,con=conn)  
 }  
close(conn)  
}

## Found more than one class "file" in cache; using the first, from namespace 'BiocGenerics'

## Also defined by 'RJSONIO'

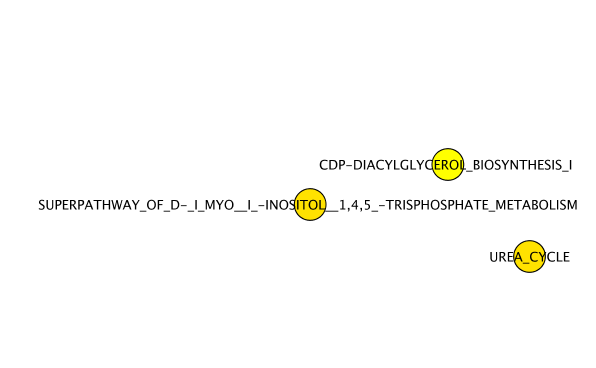
## Found more than one class "file" in cache; using the first, from namespace 'BiocGenerics'

## Also defined by 'RJSONIO'

Finally, plot the EnrichmentMap. This step requires Cytoscape to be installed, along with the EnrichmentMap and AutoAnnotate apps. It also requires the Cytoscape application to be open and running on the machine running the code. This block is commented out for automatic builds on BioConductor, but a screenshot of the intended result is shown below.

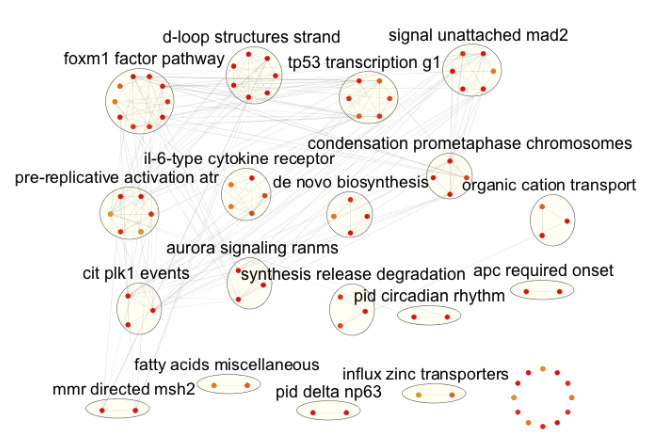
###plotEmap(gmtFiles[[1]],nodeAttrFiles[[1]],  
### groupClusters=TRUE, hideNodeLabels=TRUE)

This example EnrichmentMap isn’t terribly exciting because of the low number of pathway features permitted, the upper bound on feature selection scores and low number of train/test splits. But hopefully it serves its purpose to be illustrative.



EnrichmentMap generated from example in this vignette. The small number of nodes reflects the limited number of pathways provided to the model, and also reduced parameter values for model building.

Here is an example of an EnrichmentMap generated by running the above predictor with more real-world parameter values, and all available pathways:



This function call generates an EnrichmentMap, clusters pathways and applies WordCloud-generated labels using AutoAnnotate, and applies a network layout that arranges nodes by clusters. In the figure above, the nodes have been reorganized to improve label visibility and fit the image on a page.

## Use Case 2: Build three-way classifier with multimodal data

### Introduction

In this example, we will use clinical data and three types of ’omic data - gene expression, DNA methylation and proteomic data - to classify breast tumours as being one of three types: Luminal A, Luminal B, or Basal. This example is nearly identical to the one used to build a binary classifier and the data are from the Cancer Genome Atlas4.

We also use several strategies and definitions of similarity to create features:

* Clinical variables: Each *variable* is its own feature (e.g. age); similarity is defined as *normalized difference*.
* Gene expression: Features are defined at the level of ***pathways***; i.e. a feature groups genes corresponding to the pathway. Similarity is defined as pairwise *Pearson correlation*
* Proteomic and methylation data: Features are defined at the level of the entire *data layer*; a single feature is created for all of proteomic data, and the same for methylation. Similarity is defined by pairwise *Pearson correlation*

### Setup

Load the netDx package.

suppressWarnings(suppressMessages(require(netDx)))

### Data

For this example we pull data from the The Cancer Genome Atlas through the BioConductor curatedTCGAData package. The fetch command automatically brings in a MultiAssayExperiment object.

suppressMessages(library(curatedTCGAData))

We use the curatedTCGAData() command to look at available assays in the breast cancer dataset.

curatedTCGAData(diseaseCode="BRCA", assays="\*",dry.run=TRUE)

## Title DispatchClass  
## 31 BRCA\_CNASeq-20160128 Rda  
## 32 BRCA\_CNASNP-20160128 Rda  
## 33 BRCA\_CNVSNP-20160128 Rda  
## 35 BRCA\_GISTIC\_AllByGene-20160128 Rda  
## 36 BRCA\_GISTIC\_Peaks-20160128 Rda  
## 37 BRCA\_GISTIC\_ThresholdedByGene-20160128 Rda  
## 39 BRCA\_Methylation\_methyl27-20160128\_assays H5File  
## 40 BRCA\_Methylation\_methyl27-20160128\_se Rds  
## 41 BRCA\_Methylation\_methyl450-20160128\_assays H5File  
## 42 BRCA\_Methylation\_methyl450-20160128\_se Rds  
## 43 BRCA\_miRNASeqGene-20160128 Rda  
## 44 BRCA\_mRNAArray-20160128 Rda  
## 45 BRCA\_Mutation-20160128 Rda  
## 46 BRCA\_RNASeq2GeneNorm-20160128 Rda  
## 47 BRCA\_RNASeqGene-20160128 Rda  
## 48 BRCA\_RPPAArray-20160128 Rda

In this call we fetch only the gene expression, proteomic and methylation data; setting dry.run=FALSE initiates the fetching of the data.

brca <- suppressMessages(  
 curatedTCGAData("BRCA",  
 c("mRNAArray","RPPA\*","Methylation\_methyl27\*"),  
 dry.run=FALSE))

## Warning: package 'rhdf5' was built under R version 3.6.1

This next code block prepares the TCGA data. In practice you would do this once, and save the data before running netDx, but we run it here to see an end-to-end example.

# prepare clinical variable - stage  
staget <- sub("[abcd]","",sub("t","",colData(brca)$pathology\_T\_stage))  
staget <- suppressWarnings(as.integer(staget))  
colData(brca)$STAGE <- staget  
  
# exclude normal, HER2 (small num samples)  
pam50 <- colData(brca)$PAM50.mRNA  
idx <- union(which(pam50 %in% c("Normal-like","HER2-enriched")),   
 which(is.na(staget)))  
idx <- union(idx, which(is.na(pam50)))  
pID <- colData(brca)$patientID  
tokeep <- setdiff(pID, pID[idx])  
brca <- brca[,tokeep,]  
  
pam50 <- colData(brca)$PAM50.mRNA  
colData(brca)$pam\_mod <- pam50  
  
# remove duplicate names  
smp <- sampleMap(brca)  
for (nm in names(brca)) {  
 samps <- smp[which(smp$assay==nm),]  
 notdup <- samps[which(!duplicated(samps$primary)),"colname"]  
 brca[[nm]] <- suppressMessages(brca[[nm]][,notdup])  
}

## harmonizing input:  
## removing 59 sampleMap rows with 'colname' not in colnames of experiments

## harmonizing input:  
## removing 19 sampleMap rows with 'colname' not in colnames of experiments

## harmonizing input:  
## removing 26 sampleMap rows with 'colname' not in colnames of experiments

The important thing is to create ID and STATUS columns in the sample metadata slot. netDx uses these to get the patient identifiers and labels, respectively.

pID <- colData(brca)$patientID  
colData(brca)$ID <- pID  
colData(brca)$STATUS <- gsub(" ","\_",colData(brca)$pam\_mod)

### Rules to create features (patient similarity networks)

Our plan is to group gene expression data by pathways and clinical data by single variables. We will treat methylation and proteomic data each as a single feature, so each of those groups will contain the entire input table for those corresponding data types.

In the code below, we fetch pathway definitions for January 2018 from http://download.baderlab.org/EM\_Genesets and group gene expression data by pathways. To keep the example short, we limit to only three pathways, but in practice you would use all pathways meeting a size criterion; e.g. those containing between 10 and 500 genes.

Grouping rules are accordingly created for the clinical, methylation and proteomic data.

groupList <- list()  
  
# genes in mRNA data are grouped by pathways  
pathList <- readPathways(fetchPathwayDefinitions("January",2018))

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

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

## File: 182107f6006ac\_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

groupList[["BRCA\_mRNAArray-20160128"]] <- pathList[1:3]  
# clinical data is not grouped; each variable is its own feature  
groupList[["clinical"]] <- list(  
 age="patient.age\_at\_initial\_pathologic\_diagnosis",  
 stage="STAGE"  
)  
# for methylation generate one feature containing all probes  
# same for proteomics data  
tmp <- list(rownames(experiments(brca)[[2]]));  
names(tmp) <- names(brca)[2]  
groupList[[names(brca)[2]]] <- tmp  
  
tmp <- list(rownames(experiments(brca)[[3]]));  
names(tmp) <- names(brca)[3]  
groupList[[names(brca)[3]]] <- tmp

#### Define patient similarity for each network

We provide netDx with a custom function to generate similarity networks (i.e. features). The first block tells netDx to generate correlation-based networks using everything but the clinical data. This is achieved by the call:

makePSN\_NamedMatrix(..., writeProfiles=TRUE,...)`

The second block makes a different call to makePSN\_NamedMatrix() but this time, requesting the use of the normalized difference similarity metric. This is achieved by calling:

makePSN\_NamedMatrix(,...,   
 simMetric="custom", customFunc=normDiff,  
 writeProfiles=FALSE)

normDiff is a function provided in the netDx package, but the user may define custom similarity functions in this block of code and pass those to makePSN\_NamedMatrix(), using the customFunc parameter.

makeNets <- function(dataList, groupList, netDir,...) {  
 netList <- c() # initialize before is.null() check  
 # correlation-based similarity for mRNA, RPPA and methylation data  
 # (Pearson correlation)  
 for (nm in setdiff(names(groupList),"clinical")) {  
 # NOTE: the check for is.null() is important!  
 if (!is.null(groupList[[nm]])) {  
 netList <- makePSN\_NamedMatrix(dataList[[nm]],  
 rownames(dataList[[nm]]),  
 groupList[[nm]],netDir,verbose=FALSE,  
 writeProfiles=TRUE,...)   
 }  
 }  
   
 # make clinical nets (normalized difference)  
 netList2 <- c()  
 if (!is.null(groupList[["clinical"]])) {  
 netList2 <- makePSN\_NamedMatrix(dataList$clinical,   
 rownames(dataList$clinical),  
 groupList[["clinical"]],netDir,  
 simMetric="custom",customFunc=normDiff, # custom function  
 writeProfiles=FALSE,  
 sparsify=TRUE,verbose=TRUE,...)  
 }  
 netList <- c(unlist(netList),unlist(netList2))  
 return(netList)  
}

### Build predictor

Finally we make the call to build the predictor.

set.seed(42) # make results reproducible  
  
# location for intermediate work  
# set keepAllData to TRUE to not delete at the end of the   
# predictor run.  
# This can be useful for debugging.  
outDir <- sprintf("%s/pred\_output",tempdir())   
  
numSplits <- 2L  
out <- suppressMessages(  
 buildPredictor(dataList=brca,groupList=groupList,  
 makeNetFunc=makeNets,outDir=outDir,  
 numSplits=numSplits, featScoreMax=2L, featSelCutoff=1L,  
 numCores=1L)  
)

### Examine output

Compute accuracy for three-way classification.

# Average accuracy  
numSplits <- 2  
st <- unique(colData(brca)$STATUS)   
acc <- matrix(NA,ncol=length(st),nrow=numSplits)   
colnames(acc) <- st   
for (k in 1:numSplits) {   
 pred <- out[[sprintf("Split%i",k)]][["predictions"]];  
 tmp <- pred[,c("ID","STATUS","TT\_STATUS","PRED\_CLASS",  
 sprintf("%s\_SCORE",st))]  
 for (m in 1:length(st)) {  
 tmp2 <- subset(tmp, STATUS==st[m])  
 acc[k,m] <- sum(tmp2$PRED==tmp2$STATUS)/nrow(tmp2)  
 }  
}  
print(round(acc\*100,2))

## Luminal\_A Basal-like Luminal\_B  
## [1,] 57.14 100 28.57  
## [2,] 58.62 100 45.00

Also, examine the confusion matrix. We can see that the model perfectly classifies basal tumours, but performs poorly in distinguishing between the two types of luminal tumours.

res <- out$Split1$predictions  
print(table(res[,c("STATUS","PRED\_CLASS")]))

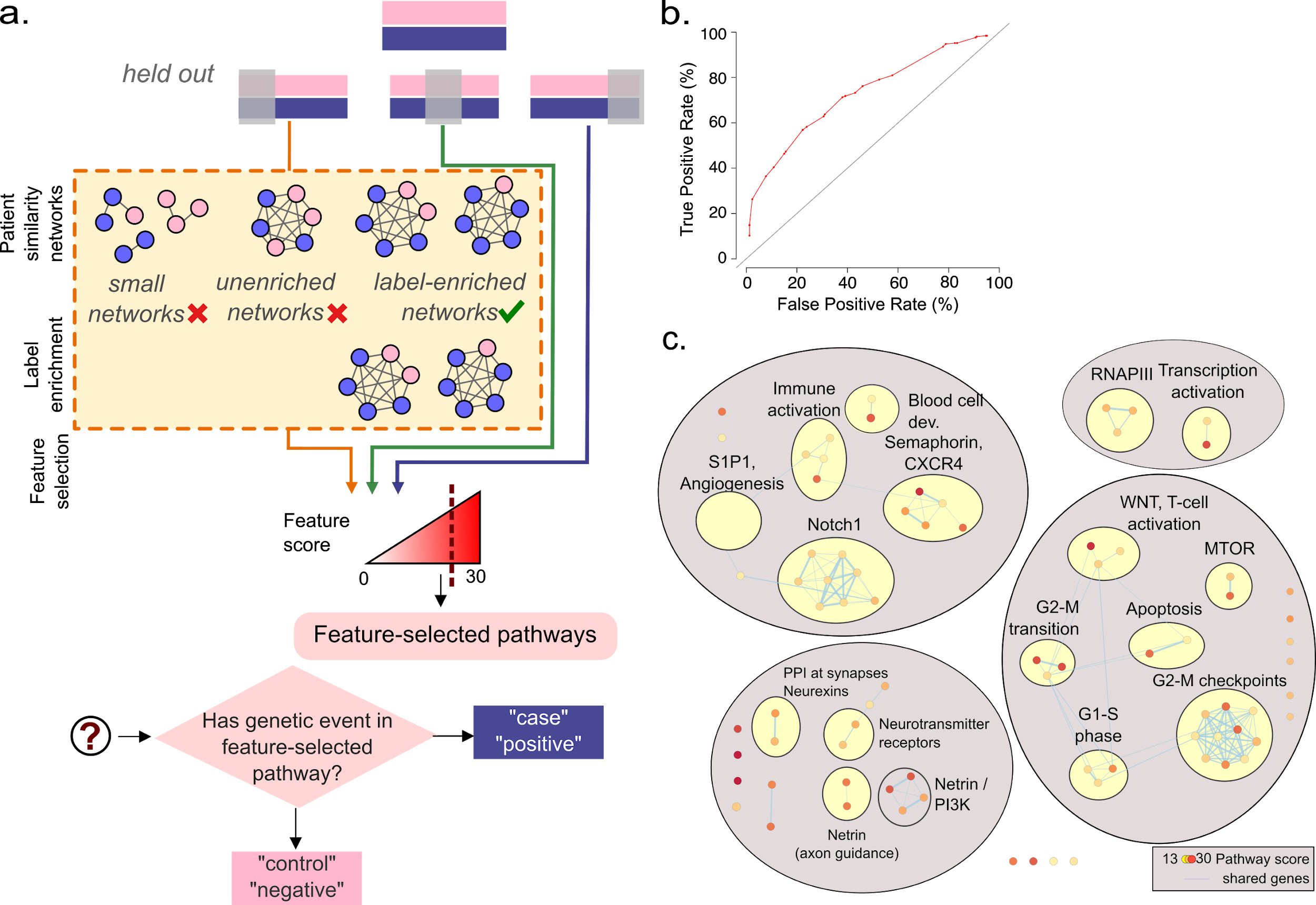
## PRED\_CLASS  
## STATUS Basal-like Luminal\_A Luminal\_B  
## Basal-like 14 0 0  
## Luminal\_A 4 16 8  
## Luminal\_B 4 6 4

## Use Case 3: Classification with sparse genetic data - AGP1

netDx natively handles missing data, making it suitable to build predictors with sparse genetic data such as somatic DNA mutations, frequently seen in cancer, and from DNA Copy Number Variations. This example demonstrates how to use netDx to build a predictor from sparse genetic data. Here we build a case/control classifier for Autism Spectrum Disorder (ASD) diagnosis, starting from rare CNVs; for this, we use data from Pinto *et al.*7.

### Design and Adapting the Algorithm for Sparse Event Data

In this design, we group CNVs by pathways. The logic behind the grouping is prior evidence showing that genetic events in diseases tend to converge on cellular processes of relevance to the pathophysiology of the disease. For example, see the Pinto et al. paper referenced in the previous section.



**Figure 2.** Autism sparse genetic data binary similarity in pathways.

#### Binary Similarity and Label enrichment

In this design, similarity is defined as a binary function, a strategy that has advantages and drawbacks. In plain terms, ***if two patients share a mutation in a pathway, their similarity for that pathway is 1.0 ; otherwise it is zero.*** This binary definition, while conceptually intuitive, increases the false positive rate in the netDx feature selection step. That is, networks with even a single case sample will get a high feature score, regardless of whether that network is enriched for case samples.

To counter this problem, we introduce a ***label-enrichment*** step in the feature selection. A bias measure is first computed for each network, such that a network with only cases has +1; one with only controls has a score of -1; and one with an equal number of both has a score of zero. Label-enrichment compares the bias in each real network, to the bias in that network in label-permuted data. It then assigns an empirical p-value for the proportion of times a label-permuted network has a bias as high as the real network. Only networks with a p-value below a user-assigned threshold pass label-enrichment, and feature selection is limited to these networks. In netDx, label-enrichment is enabled by setting enrichLabels=TRUE in the call to buildPredictor\_sparseGenetic().

#### Cumulative feature scoring

The other difference between this design and those with non-sparse data, is the method of feature scoring. The user specifies a parameter which indicates the number of times to split the data and run feature selection. The algorithm then runs feature selection numSplits times, each time leaving 1/numSplits of the samples out. In each split, features are scored between 0 and featScoreMax, using the same approach as is used for continuous-valued input. Feature scores are then added across the splits so that a feature can score as high as numSplits\*featScoreMax.

#### Evaluating model performance

For a given cutoff for features, a patient is called a “case” if they have a genetic event in pathways that pass feature selection at that cutoff; otherwise, at that cutoff, they are labelled a “control”. These calls are used to generate the false positive and true positive rates across the various cutoffs, which ultimately generates an ROC curve.

### Setup

suppressMessages(require(netDx))  
suppressMessages(require(GenomicRanges))  
suppressMessages(require(biomaRt)) # for fetching gene coordinates

## Warning: package 'biomaRt' was built under R version 3.6.1

### Data

CNV coordinates are read in, and converted into a GRanges object. As always, the sample metadata table, here the pheno object, must have ID and STATUS columns.

outDir <- sprintf("%s/200129\_threeWay",tempdir())  
if (file.exists(outDir)) unlink(outDir,recursive=TRUE); dir.create(outDir)  
  
cat("\* Setting up sample metadata\n")

## \* Setting up sample metadata

phenoFile <- sprintf("%s/extdata/AGP1\_CNV.txt",path.package("netDx"))  
pheno <- read.delim(phenoFile,sep="\t",header=TRUE,as.is=TRUE)  
colnames(pheno)[1] <- "ID"  
head(pheno)

## ID seqnames start end Gene\_symbols Pathogenic STATUS  
## 3 1020\_4 chr3 4110452 4145874 no case  
## 4 1030\_3 chr10 56265896 56361311 no case  
## 5 1030\_3 chr7 64316996 64593616 ZNF92,LOC441242 no case  
## 7 1045\_3 chr3 83206919 83239473 no case  
## 11 1050\_3 chr6 57021412 57062509 KIAA1586 no case  
## 16 1116\_4 chr1 30334653 30951250 no case

cnv\_GR <- GRanges(pheno$seqnames,IRanges(pheno$start,pheno$end),  
 ID=pheno$ID,LOCUS\_NAMES=pheno$Gene\_symbols)  
pheno <- pheno[!duplicated(pheno$ID),]

### Group CNVs by pathways

The fetchPathwayDefinitions() function downloads pathway definitions from baderlab.org but users may provide custom .gmt files as well. In the example below, gene coordinates for the hg18 genome build are also automatically fetched from BioMart, using the biomaRt package, and converted to a GRanges object. The function mapNamedRangesToSets() is used to group this GRanges object into pathway-level sets.

pathFile <- fetchPathwayDefinitions("February",2018,verbose=TRUE)

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

pathwayList <- readPathways(pathFile)

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

## File: f72c2f3fae\_Human\_AllPathways\_February\_01\_2018\_symbol.gmt

## Read 3199 pathways in total, internal list has 3163 entries

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

## => 1044 pathways excluded  
## => 2119 left

# get gene coordinates, use hg18  
ensembl <- suppressMessages(suppressWarnings(  
 useMart("ENSEMBL\_MART\_ENSEMBL",  
 dataset="hsapiens\_gene\_ensembl",  
 host="may2009.archive.ensembl.org",  
 path="/biomart/martservice",archive=FALSE)  
))  
genes <- getBM(attributes=c("chromosome\_name",  
 "start\_position",  
 "end\_position",  
 "hgnc\_symbol"),  
 mart=ensembl)  
genes <- genes[which(genes[,4]!=""),]  
gene\_GR <- GRanges(genes[,1],IRanges(genes[,2],genes[,3]),  
 name=genes[,4])

Group gene extents into pathway-based sets, which effectively creates grouping rules for netDx. The function mapNamedRangesToSets() does this grouping, generating a GRangesList object.

path\_GRList <- mapNamedRangesToSets(gene\_GR,pathwayList)

### Run predictor

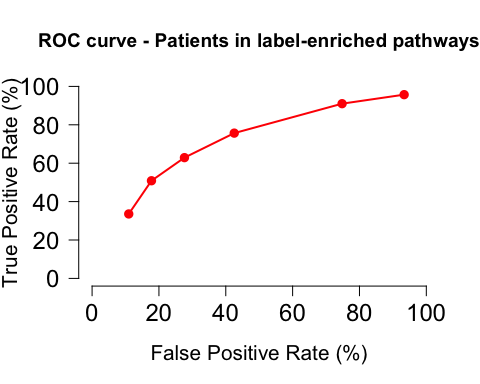
Once the phenotype matrix and grouping rules are set up, the predictor is called using buildPredictor\_sparseGenetic(). Note that unlike with non-sparse data, the user does not provide a custom similarity function in this application; currently, the only option available is the binary similarity defined above. As discussed above, setting enrichLabels=TRUE to enable label-enrichment is highly recommended to reduce false positive rate.

predictClass <- "case"  
out <- suppressMessages(  
 buildPredictor\_sparseGenetic(pheno, cnv\_GR, predictClass,  
 path\_GRList,outDir,  
 numSplits=3L, featScoreMax=3L,  
 enrichLabels=TRUE,numPermsEnrich=20L,  
 numCores=2L)  
)

### Plot results

Feature selection identifies pathways that are consistently enriched for the label of interest; here, “case” status. From the diagnostic point of view, a patient with a genetic event in a selected feature - here, a CNV in a feature-selected pathway - is labelled a “case”. “True positives” are therefore cases with CNVs in feature-selected pathways, while “false positives” are controls with CNVs in feature-selected pathways. These definitions are used to compute the ROC curve below.

dat <- out$performance\_denEnrichedNets  
plot(0,0,type="n",xlim=c(0,100),ylim=c(0,100),  
 las=1, xlab="False Positive Rate (%)",   
 ylab="True Positive Rate (%)",  
 bty='n',cex.axis=1.5,cex.lab=1.3,  
 main="ROC curve - Patients in label-enriched pathways")  
points(dat$other\_pct,dat$pred\_pct,  
 col="red",type="o",pch=16,cex=1.3,lwd=2)



We can also compute the AUROC and AUPR from scratch.

tmp <- data.frame(   
 score=dat$score,  
 tp=dat$pred\_ol,fp=dat$other\_ol,  
 # tn: "-" that were correctly not called  
 tn=dat$other\_tot - dat$other\_ol,  
 # fn: "+" that were not called   
 fn=dat$pred\_tot - dat$pred\_ol)   
  
stats <- netDx::perfCalc(tmp)  
tmp <- stats$stats  
message(sprintf("PRAUC = %1.2f\n", stats$prauc))

## PRAUC = 0.41

message(sprintf("ROCAUC = %1.2f\n", stats$auc))

## ROCAUC = 0.71

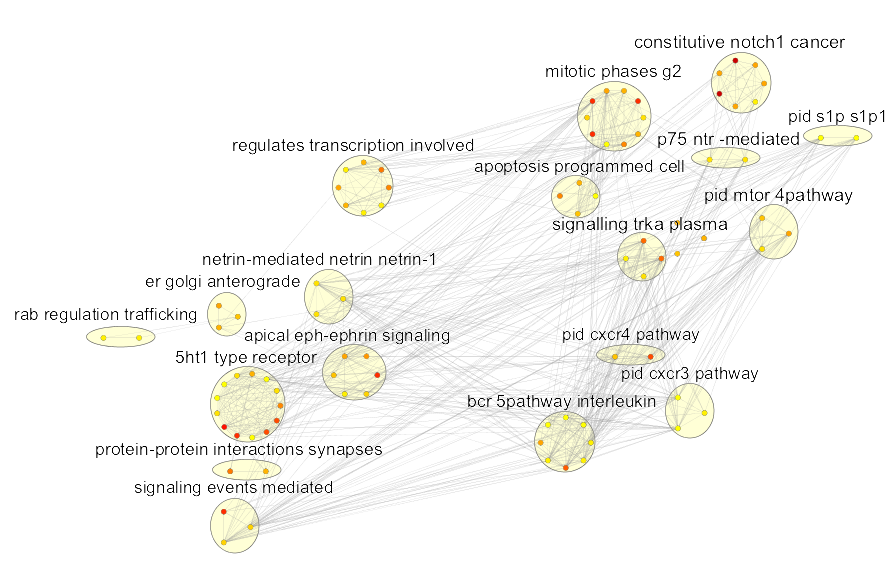
Pathway scores are also added across the splits, for a total of 9 across the 3 splits (3 + 3 + 3).

# now get pathway score  
print(head(out$cumulativeFeatScores))

## PATHWAY\_NAME  
## METABOTROPIC\_GLUTAMATE\_RECEPTOR\_GROUP\_III\_PATHWAY\_cont.txt METABOTROPIC\_GLUTAMATE\_RECEPTOR\_GROUP\_III\_PATHWAY  
## HALLMARK\_PROTEIN\_SECRETION\_cont.txt HALLMARK\_PROTEIN\_SECRETION  
## PROTEIN-PROTEIN\_INTERACTIONS\_AT\_SYNAPSES\_cont.txt PROTEIN-PROTEIN\_INTERACTIONS\_AT\_SYNAPSES  
## NEUROTRANSMITTER\_RECEPTORS\_AND\_POSTSYNAPTIC\_SIGNAL\_TRANSMISSION\_cont.txt NEUROTRANSMITTER\_RECEPTORS\_AND\_POSTSYNAPTIC\_SIGNAL\_TRANSMISSION  
## NEUREXINS\_AND\_NEUROLIGINS\_cont.txt NEUREXINS\_AND\_NEUROLIGINS  
## S\_PHASE\_cont.txt S\_PHASE  
## SCORE  
## METABOTROPIC\_GLUTAMATE\_RECEPTOR\_GROUP\_III\_PATHWAY\_cont.txt 6  
## HALLMARK\_PROTEIN\_SECRETION\_cont.txt 6  
## PROTEIN-PROTEIN\_INTERACTIONS\_AT\_SYNAPSES\_cont.txt 6  
## NEUROTRANSMITTER\_RECEPTORS\_AND\_POSTSYNAPTIC\_SIGNAL\_TRANSMISSION\_cont.txt 6  
## NEUREXINS\_AND\_NEUROLIGINS\_cont.txt 6  
## S\_PHASE\_cont.txt 6

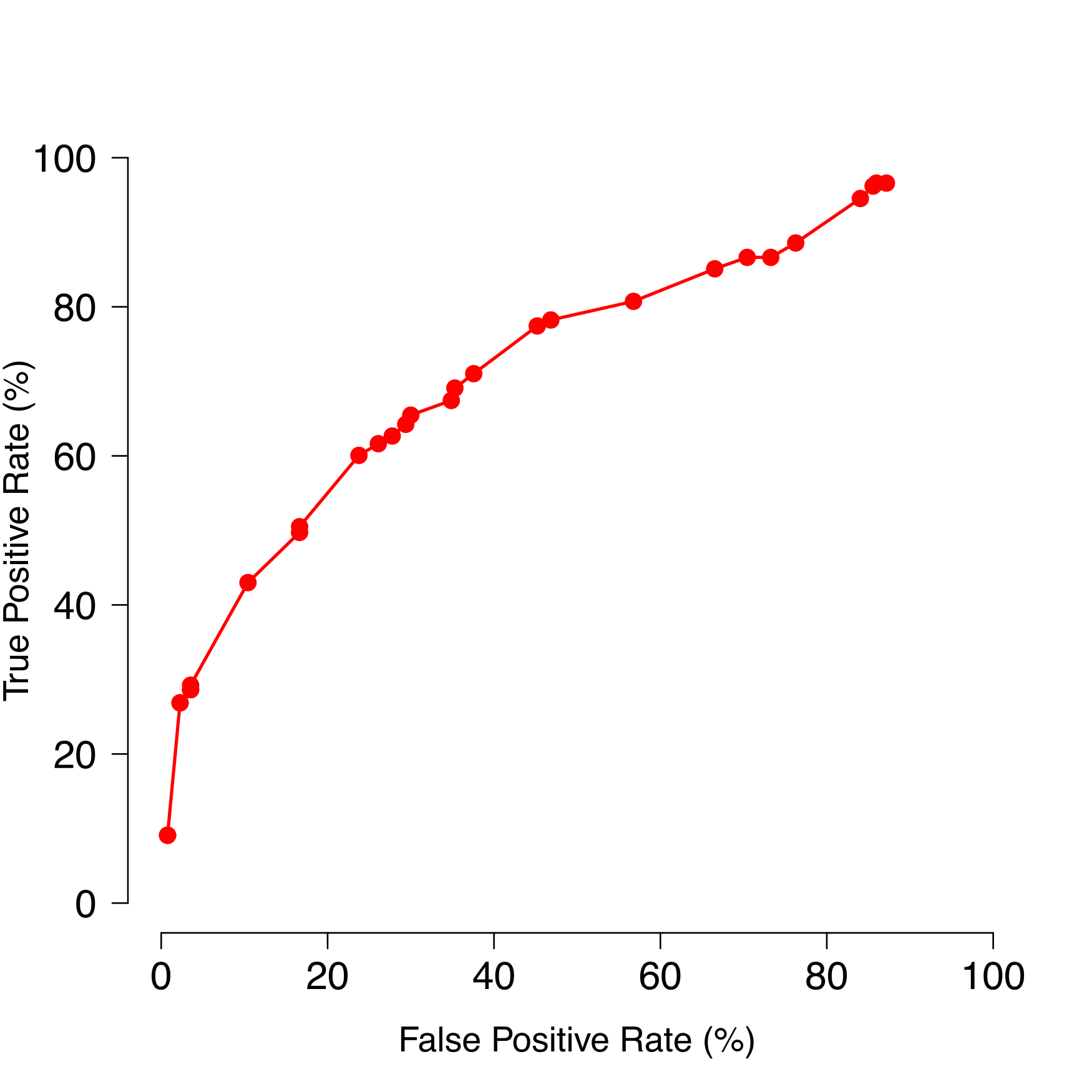
|  |
| --- |
| predictClass <- "case" buildPredictor\_sparseGenetic(pheno, cnv\_GR, predictClass,  path\_GRList,outDir,  numSplits=3L, featScoreMax=10L,  enrichLabels=TRUE,numPermsEnrich=200L,  numCores=8L) |

As before, running the predictor with all possible pathway-related features and realistic training parameters, such as numPermsEnrich=200L, featScoreMax=10L, numSplits=3L identifies a much richer set of themes themes related to synaptic transmission and cell proliferation, consistent with those identified in the original publication8.



The nodes in the image above have been reorganized to group clusters sharing a broader theme. Above, terms related to neurotransmission and synaptic plasticity are in the bottom left; those related to the cell cycle and proliferation are in the top-right; and those related to immune function are in the bottom right.

The dynamic range of feature scores is much larger as well, here ranging from 0 to 30, here as is evident from the ROC curve.



# Software Updates

netDx vXXX has several updates relative to the version released with the netDx paper (v1.0.23).

A major change includes a renaming of most functions to reflect their role rather than implementation; Table 1 provides a reference of previous and corresponding updated function names. The updated package also includes the ability to build a classifier from sparse genetic data (Use Case 3), using the function buildPredictor\_sparseGenetic(). Finally, the feature selection step now permits a Monte Carlo resampling strategy for selecting samples for iterative feature scoring. The previous version of the software required a fraction of samples to be held out, the fraction being directly related to the maximum feature score. The resampling approach should allow users to increase the upper-bound of feature scores, even in smaller samples.

A number of software updates were made as part of the netDx-BioConductor integration. The new netDx package works on Windows machines, in addition to OS X and Unix, and all built-in vignettes successfully run on these platforms. However, where possible, users are advised to run netDx on Mac and Unix, as the software has not been extensively tested on the Windows platform. Several changes were made to conform to BioConductor standards, notably the direction of all output to a temporary directory unless otherwise specified, automatic logging, and the dependency of plotting functions on an expected output directory structure. Users now set random number generator seeds at the outset, before calling the buildPredictor() function as all parameters allowing seed setting have been removed. Separately, the companion R package *netDx-examples*, previously used to store example data, is now deprecated. All examples are now either contained within the *netDx* package or are fetched from BioConductor using local file-caching (FileCache package). Automated network visualization in Cytoscape now uses RCy3, for programmatic access of Cytosape from R.

Memory improvements were made to the underlying GeneMANIA Java implementation, creating a modified version specifically for netDx. netDx incurs a higher memory footprint than other machine learning methods because each feature in netDx is a similarity network with pairwise similarity measures. Network integration, a step in feature seleciton, requires keeping all these networks in memory. Certain grouping rules also incur a greater memory footprint than others. For instance, when starting with one matrix of gene expression data, the design generating ~2,000 pathway-level features would be less scalable than the design generating one feature generated from the entire expression matrix. Memory optimization was achieved by customizing the underlying Java application used for network integration, an implementation of the GeneMANIA network integration algorithm. netDx now uses a custom version of this Java jar, which bypasses the identifier conversion and file-writing steps used in GeneMANIA but not required for netDx. Memory and computational time improvements were benchmarked by building binary classifiers for a breast cancer dataset [cite TCGA] and schizophrenia case-control classification. The CommonMind Consortium [CITE paper; CITE synapse] dataset included 279 controls and 258 cases, with a total of 537 patients, with gene expression data from the prefrontal cortex organized into pathway level features (1,735 pathways).The breast cancer data was part of the TCGA project, with tumour gene-expression for 348 patients, including 154 Luminal A and 194 tumours of other subtypes, also organized into pathway-level features (1,706 pathways). All tests were performed on an Intel XXX machine with XGB RAM and using X cores. Following improvements, memory use dropped to one-third of the original amount. With the updated software, the CommonMind dataset also required two-third of the time to build the predictor, as compared to with the original version (Table 2).

# Conclusions

We expect this tool to be useful for clinical research in applications pertaining to risk stratification, treatment response, and to identify biomarkers associated with patient subtypes. Method extensions will be required to predict continuous outcome. Classification of borderline patients could be better controlled, perhaps by a user-specified margin. Similar to pathway-level grouping for gene expression data, other grouping strategies will be required for other types of genomic data, such as miRNA, single nucleotide polymorphisms, and brain imaging data.

# Data Availability

**Autism example.**

Data for the autism case/control classification is provided as part of the netDx package.

**Breast cancer example**

Data for the breast cancer examples is fetched from the curatedTCGAData package which is maintained by the BioConductor repository

# Software Availability

The source code for netDx is available from BioConductor: <http://bioconductor.org/packages/devel/bioc/html/netDx.html>

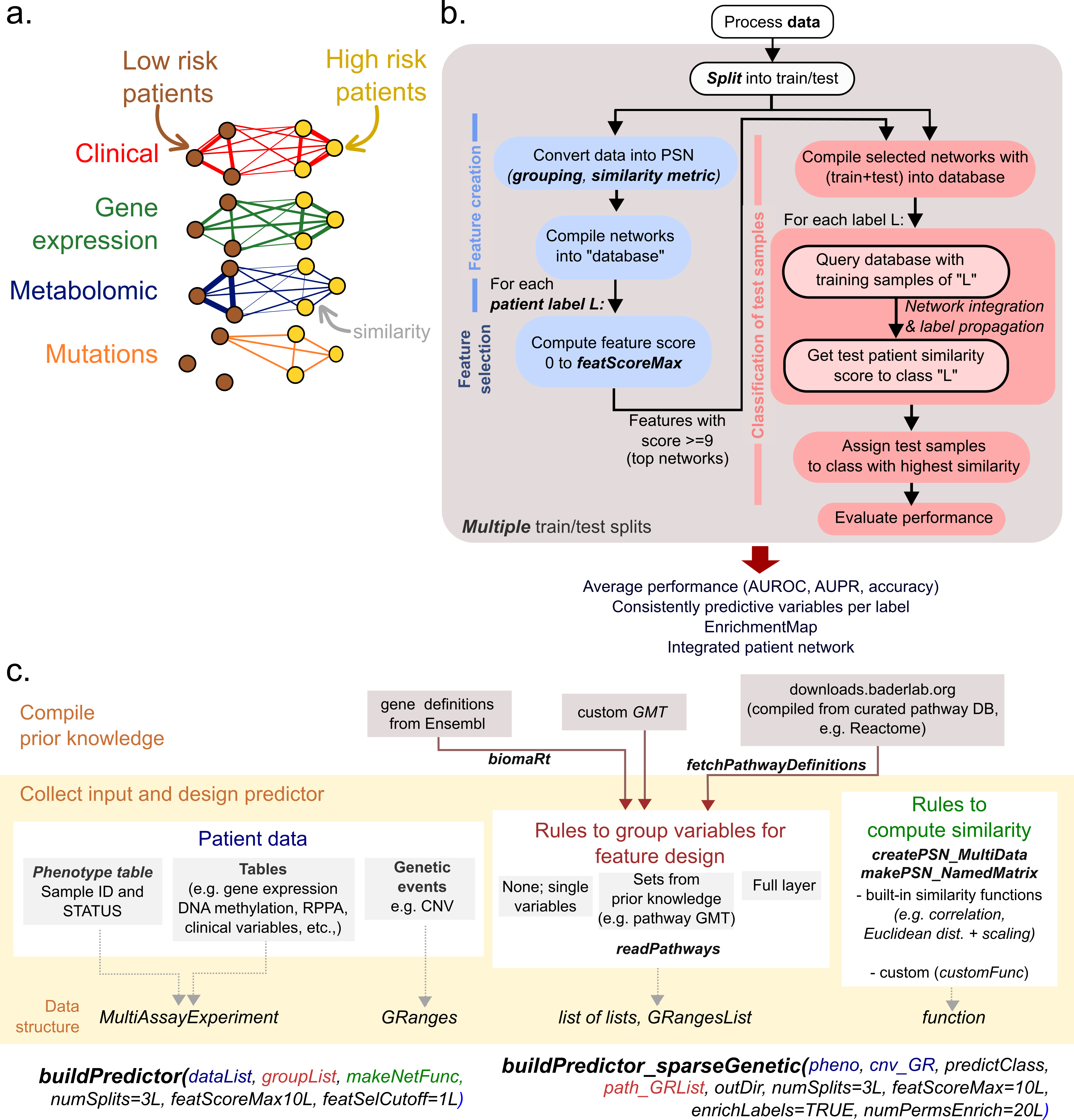
Archived source code at time of publication: Zenodo link  
Issue tracker: <https://github.com/BaderLab/netDx/issues>

License: MIT

# Acknowledgements

We greatly appreciate the input from Marcel Ramos in furthering the integration of netDx with BioConductor. We appeciate the work of Guodong Xu and acknowledge Google Summer of Code in helping prototype a previous solution for improved memory use in netDx.

# Figures

****

**Figure 1.** netDx concepts and software workflow. a. Conceptual visualization of patient similarity networks. Nodes are patients and edge weights measure pairwise similarity for a given problem. The example shows a two-class problem (with high risk and low risk patients), with four features (here, patient similarity networks): those for clinical (red), gene expression (green), metabolomic (blue) and mutation data (orange). b. Conceptual workflow for netDx predictor. Text in italicized bold font indicate parameters provided by the user. Samples are split into train and test samples, and training samples are subjected to feature selection (blue flow). Feature selection involves feature creation from provided data and rules, and calculation of feature-level scores. Each feature is assigned one score per patient label. Features passing a user-specified threshold are used to classify held-out samples. Using network integration of top features, followed by label propagation, the similarity of held-out patients is computed to each label in turn. Test patients are assigned to the class to which they are most similar. This process of feature selection and classification is repeated for mulitple train/test splits, and the model is then evaluated based on average performance, and the nature of features that are consistently predictive of labels.

# Tables

|  |  |
| --- | --- |
| Function name  (vXXX) | Purpose |
| Basic model design |  |
| *buildPredictor()* | Turnkey function to build predictor |
| *buildPredictor\_sparseGenetic()* | Turnkey |
| *makePSN\_NamedMatrix()* | Create PSN from a single data layer (matrix representation) |
| *createPSN\_MultiData()* | Create PSN from multiple data layers. May include calls to *makePSN\_NamedMatrix()* |
| *plotPerf()* | Plot ROC and PR curves, compute AUROC and AUPR |
| *plotEMap()* | Plot EnrichmentMap in Cytoscape, annotating main themes. Used in pathway-based feature design |
| *plotIntegratedPatientNetwork()* | Plot patient network by integrating predictive features for all patient labels |
| Advanced control of model design |  |
| *splitTrainTest()* | Sets up feature control |
| *setupFeatureDB()* | Collects all created features into a database in preparation for feature selection. |
| *compileFeatures()* | Feature selection process. Iterative scoring of input networks based on network integration and regularized regression. Each scoring step is called a “query”. A pre-specified number of queries is run with different subsamples of the training set. |
| *runFeatureSelection()* | Run network integration and label propagation. Used in feature selection for unit increase in network weights. Also used for classification of test samples, where it returns similarity scores. |
| *writeQueryFile()*  *runQuery()* | Feature selection: Collects feature scores from |
| *compileFeatureScores()* | Gets patient similarity scores fol |
| *getPatientRankings()* | *Fill this out* |
| *predictPatientLabels()* | *Fill this out* |

**Table 1.** Major functions for basic and advanced model-building, and result evaluation. Intermediate functions useed to prepare data are not shown, but are illustrated in the use cases.

|  |  |  |
| --- | --- | --- |
| JavaMemory setting | Previous runtime (s) | Current runtime (s) |
| Breast cancer (Luminal A / not): 154 cases, 194 controls  1,706 pathway-based networks from gene expression data | | |
| 4GB | 37.52 +- 2.0 | 2.08 +- 0.01 |
| 6GB | 35.62 +- 2.0 | 1.98 +- 0.20 |
| 8GB | 36.84 +- 1.0 | 2.04 +- 0.20 |
| Schizophrenia (case / control): 279 controls, 258 cases 1,735 pathway-based networks from gene expression | | |
| 4GB | 604.36 +- 13.0 | 390.69 +- 8.0 |
| 6GB | 592.41 +- 20.0 | 397.52 +- 12.0 |
| 8GB | 580.44 +- 14 | 394.55 +- 8.0 |

**Table 2:** Benchmarking memory improvement for netDx

# References

1 Pai, S. *et al.* netDx: interpretable patient classification using integrated patient similarity networks. *Molecular systems biology* **15**, e8497, doi:10.15252/msb.20188497 (2019).

2 Pai, S. & Bader, G. D. Patient Similarity Networks for Precision Medicine. *Journal of molecular biology* **430**, 2924-2938, doi:10.1016/j.jmb.2018.05.037 (2018).

3 Huber, W. *et al.* Orchestrating high-throughput genomic analysis with Bioconductor. *Nature methods* **12**, 115-121, doi:10.1038/nmeth.3252 (2015).

4 Comprehensive molecular portraits of human breast tumours. *Nature* **490**, 61-70, doi:10.1038/nature11412 (2012).

5 Merico, D., Isserlin, R., Stueker, O., Emili, A. & Bader, G. D. Enrichment map: a network-based method for gene-set enrichment visualization and interpretation. *PloS one* **5**, e13984, doi:10.1371/journal.pone.0013984 (2010).

6 Kucera, M., Isserlin, R., Arkhangorodsky, A. & Bader, G. D. AutoAnnotate: A Cytoscape app for summarizing networks with semantic annotations. *F1000Research* **5**, 1717, doi:10.12688/f1000research.9090.1 (2016).

7 Pinto, D. *et al.* Convergence of genes and cellular pathways dysregulated in autism spectrum disorders. *American journal of human genetics* **94**, 677-694, doi:10.1016/j.ajhg.2014.03.018 (2014).

# Supplementary Tables

|  |  |
| --- | --- |
| Function name  (vXXX) | Function name in Pai et al. (2019). (v1.0.23) |
| *buildPredictor()* | *runPredictor\_nestedCV()* |
| *buildPredictor\_sparseGenetic()* |  |
| *makePSN\_NamedMatrix()* | *makePSN\_NamedMatrix()* |
| *createPSN\_MultiData()* | *createPSN\_MultiData()* |
| *splitTrainTest()* | *splitTrainTest()* |
| *setupFeatureDB()* | - |
| *compileFeatures()* | *GM\_createDB()* |
| *runFeatureSelection()* | *GM\_runCV\_featureSet()* |
| *writeQueryFile()*  *runQuery()* | *GM\_writeQueryFile()*  *runGeneMANIA()* |
| *compileFeatureScores()* | *GM\_networkTally()* |
| *getPatientRankings()* | *GM\_getQueryROC()* |
| *predictPatientLabels()* | *GM\_OneVAll\_getClass()* |

**Supplementary Table 1.** Function names in the current and original version of netDx