netDx use case
Integrate gene expression and CNV for binary classification of breast tumour

Shraddha Pai

Last updated: 20 June, 2016

1 Introduction

This tutorial shows the steps to build a breast tumour classifier using netDx (Ref 1) by integrating gene expression and DNA copy number variants (CNV). Based on the expression of 50 genes, breast tumours are traditionally classified into one of four broad categories: Luminal A, Luminal B, HER2-enriched, and basal-like (Ref 2). Each category of tumour has different prognostic value and response to chemotherapy and/or hormone therapy. To keep things simple, in this tutorial we build a binary classifier that discriminates between the Luminal A and other subtypes. The Luminal A subtype is a low-grade tumour with good prognosis; as it expresses the estrogen receptor, this type of tumour is a good candidate for hormone therapy (Ref 3).

Through this exercise, we will use the following capabilities of netDx:

- Perform feature selection on the training set
- Assess performance on the test set
- Generate patient similarity networks from more than one type of data

The workflow is shown in Figure 1. The algorithm proceeds in two steps:

1. *Feature selection:* Two-thirds of the samples from each class are designated as training samples. Feature selection is carried out twice, once for LumA samples and once for non-LumA samples. For details, see Ref 1.
2. *Predicting classes of test samples*: The other one-third of samples in each class are designed as test samples. For each class, a single integrated GeneMANIA network (or database) is constructed, comprising of those networks feature-selected in the previous step. This network should contain all patients in the database (train and test). A GeneMANIA query is then run against each database, using training samples from the respective class as query. This step obtains a class-similarity ranking for each test

sample. These ranks are then normalized and the patient is assigned to the class for which it has a higher rank.

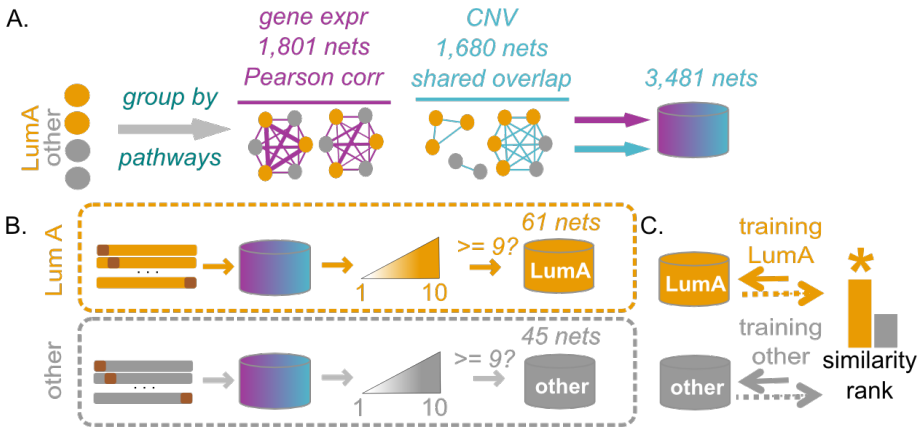

Figure 1: netDx workflow for a binary tumour classifier from gene expression and CNV data.
A. Two sets of patient similarity networks are generated: the first based on correlation of gene expression in cellular pathways (magenta), and the second based on shared overlap of CNVs in cellular pathways (teal). Each datatype generates 1,000-2,000 networks, and these are integrated into a single database by GeneMANIA.
B. Feature selection is separately carried out for the 'LumA' class for the 'other' class. A GeneMANIA query is run on the integrated database is queried 10 times; each time a different 9/10th of the training "+" samples is used as query. A network's score is the frequency with which GeneMANIA marks it as being informative. Networks scoring 9 or 10 out of 10 are feature selected. Before patient classification, two enriched databases are constructed (orange and grey cylinders); each contain feature selected nets and train as well as test samples.
C. Patient similarity to a class is ranked by running a query against the class database; this is done once per class. Test patients are assigned to the class for which they have the highest-ranking similarity.

2 Set up environment

A multi-core compute node is highly recommended.

```
rm(list=ls())

# Change this to a local directory where you have write permission
outDir <- "~/tmp/TCGA_BRCA"

numCores        <- 8L   # num cores available for parallel processing
GMmemory        <- 4L   # java memory in Gb
cutoff          <- 9L   # score cutoff for feature-selected networks
TRAIN_PROP <- 0.67      # fraction of samples to use for training

if (file.exists(outDir)) unlink(outDir,recursive=TRUE)
dir.create(outDir)
```

Load the netDx software and data packages. Finally, load the breast cancer dataset.

```
require(netDx)

## Loading required package:  netDx
## Loading required package:  bigmemory
## Loading required package:  bigmemory.sri
## Loading required package:  foreach
## foreach:  simple, scalable parallel programming from Revolution
Analytics
## Use Revolution R for scalability, fault tolerance and more.
## http://www.revolutionanalytics.com
## Loading required package:  doParallel
## Loading required package:  iterators
## Loading required package:  parallel
## Loading required package:  combinat
##
## Attaching package:  'combinat'
## The following object is masked from 'package:utils':
##
##    combn
## Loading required package:  GenomicRanges
## Loading required package:  BiocGenerics
##
## Attaching package:  'BiocGenerics'
## The following objects are masked from 'package:parallel':
##
##    clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
```

```
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
## The following object is masked from 'package:stats':
##
##     xtabs
## The following objects are masked from 'package:base':
##
##     Filter, Find, Map, Position, Reduce, anyDuplicated, append,
##     as.data.frame, as.vector, cbind, colnames, do.call,
##     duplicated, eval, evalq, get, intersect, is.unsorted, lapply,
##     mapply, match, mget, order, paste, pmax, pmax.int, pmin,
##     pmin.int, rank, rbind, rep.int, rownames, sapply, setdiff,
##     sort, table, tapply, union, unique, unlist, unsplit
## Loading required package:  S4Vectors
## Loading required package:  stats4
## Loading required package:  IRanges
## Loading required package:  GenomeInfoDb
## Loading required package:  ROCR
## Loading required package:  gplots
##
## Attaching package:  'gplots'
## The following object is masked from 'package:IRanges':
##
##     space
## The following object is masked from 'package:stats':
##
##     lowess
## Loading required package:  pracma
##
## Attaching package:  'pracma'
## The following object is masked from 'package:combinat':
##
##     fact
## Loading required package:  RColorBrewer
```

```
require(netDx.examples)
```

```
## Loading required package:  netDx.examples
```

```
data(TCGA_BRCA)
```

3 Split data into training and test sets

```
subtypes<- c("LumA")
pheno$STATUS[which(!pheno$STATUS %in% subtypes)] <- "other"
subtypes <- c(subtypes,"other") # add residual

pheno$TT_STATUS <- splitTestTrain(pheno,
    pctT = TRAIN_PROP,setSeed = 42,predClass = "LumA" )

## Setting seed for reproducibility: 42
##        IS_TRAIN
## STATUS  TRAIN TEST
##   LumA    103   51
##   other   129   65
## # training: N=232 (LumA = 103, (other) = 129)
## # test: N=116 (LumA = 51, (other) = 65)
```

4 Create patient similarity networks

Figure 1A shows the workflow for this step. The goal is to create input networks for all possible predictors, before proceeding to feature selection (next section). Note that as our goal is feature selection, only the training samples are used to generate input networks here.

We limit CNV and gene expression data to the samples for which we have labels and keep only the data for training samples.

```
pheno_FULL      <- pheno
xpr_FULL        <- xpr
cnv_FULL        <- cnv_GR
pheno           <- subset(pheno,TT_STATUS %in% "TRAIN")
xpr                     <- xpr[,which(colnames(xpr)%in% pheno$ID)]
cnv_GR          <- cnv_GR[which(cnv_GR$ID %in% pheno$ID)]
```

4.1 Lists of pathways

First we create a list of pathways using a `.gmt` file with pathway definitions from curated databases (Reactome, HumanCyc, Panther,etc.,)

```
pathFile <- sprintf("%s/extdata/Human_160124_AllPathways.gmt",
    path.package("netDx.examples"))
pathwayList <- readPathways(pathFile)
```

```
## ----------------------------------------
## File: Human_160124_AllPathways.gmt
##
## Read 2760 pathways in total, internal list has 2712 entries
##  FILTER: sets with num genes in [10, 500]
##    => 911 pathways excluded
##    => 1801 left
```

```
head(pathwayList)
```

```
## $GUANOSINE_NUCLEOTIDES__I_DE_NOVO__I__BIOSYNTHESIS
##  [1] "NME7"   "NME6"   "RRM2B"  "GMPS"   "NME2"   "NME3"   "NME4"
##  [8] "NME5"   "RRM2"   "NME1"   "GUK1"   "RRM1"   "IMPDH2" "IMPDH1"
##
## $RETINOL_BIOSYNTHESIS
##  [1] "RDH10" "DHRS4" "LRAT"  "LIPC"  "CES5A" "DHRS9" "RDH11" "DHRS3"
##  [9] "CES1"  "RBP1"  "CES4A" "RBP2"  "PNLIP" "RBP5"  "RBP4"  "CES2"
##
## $`MUCIN_CORE_1_AND_CORE_2__I_O__I_-GLYCOSYLATION`
##  [1] "GALNT1"  "GCNT4"   "GALNT7"  "GCNT3"   "GCNT7"   "GALNT6"  "GALNT4"
##  [8] "GALNT5"  "ST3GAL2" "ST3GAL1" "ST3GAL4" "GALNT10" "GALNT15" "GALNTL6"
## [15] "B3GNT3"  "GALNT16" "GALNT18" "GALNT11" "GALNT12" "GCNT1"   "C1GALT1"
## [22] "GALNT13" "GALNT14" "WBSCR17" "GALNT8"  "GALNT9"  "GALNT2"  "GALNT3"
##
## $`SUPERPATHWAY_OF_D-_I_MYO__I_-INOSITOL__1,4,5_-TRISPHOSPHATE_METABOLISM`
##  [1] "MINPP1" "INPP5A" "INPP5B" "INPP5K" "INPP5J" "ITPKA"  "PTEN"
##  [8] "INPP5D" "IPMK"   "SYNJ2"  "INPP5F" "IMPA1"  "INPP1"  "INPPL1"
## [15] "IMPA2"  "IMPAD1" "ITPKC"  "OCRL"   "SYNJ1"  "ITPKB"
##
## $`D-_I_MYO__I_-INOSITOL__1,4,5_-TRISPHOSPHATE_DEGRADATION`
##  [1] "INPP5A" "INPP5B" "INPP5K" "INPP5J" "SYNJ2"  "INPP5F" "IMPA1"
##  [8] "INPP1"  "INPPL1" "IMPA2"  "IMPAD1" "OCRL"   "SYNJ1"
##
## $MRNA_CAPPING
##  [1] "CCNH"   "RNGTT"  "MNAT1"  "GTF2F1" "GTF2F2" "SUPT5H" "ERCC2"
##  [8] "ERCC3"  "POLR2A" "POLR2B" "POLR2C" "POLR2D" "POLR2I" "POLR2J"
## [15] "POLR2K" "POLR2L" "POLR2E" "POLR2G" "POLR2F" "NCBP2"  "POLR2H"
## [22] "RNMT"   "NCBP1"  "GTF2H2" "GTF2H1" "CDK7"   "GTF2H3" "GTF2H5"
## [29] "GTF2H4"
```

4.2 Gene expression data

From gene expression data, we create one network per cellular pathway. Similarity between two patients is defined as the Pearson correlation of the expression vector; each network is limited to genes for the corresponding pathway.

The function that generates the networks from submatrices of the gene expression data is `makePSN_NamedMatrix()`. In this case, we are generating "profiles", or simply writing submatrices corresponding to the pathways (note the `writeProfiles=TRUE` argument). As these profiles will create completely connected networks with `(N choose 2)` edges, weaker edges will first be pruned for computational feasibility. We use GeneMANIA to "sparsify" the networks in the `GM_createDB()` subroutine. Note that `netList` contains the names of networks, rather than the contents; the profiles are written to `profDir`. Profile file names end with `.profile`.

```
profDir <- sprintf("%s/profiles",outDir)
netDir <- sprintf("%s/networks",outDir)

netList <- makePSN_NamedMatrix(xpr, rownames(xpr),
        pathwayList,profDir,verbose=FALSE,
        numCores=numCores,writeProfiles=TRUE)
netList <- unlist(netList)
head(netList)

## [1] "GUANOSINE_NUCLEOTIDES__I_DE_NOVO__I__BIOSYNTHESIS.profile"
## [2] "RETINOL_BIOSYNTHESIS.profile"
## [3] "MUCIN_CORE_1_AND_CORE_2__I_O__I_-GLYCOSYLATION.profile"
## [4] "SUPERPATHWAY_OF_D-_I_MYO__I_-INOSITOL__1,4,5_-TRISPHOSPHATE_METABOLISM.profile"
## [5] "D-_I_MYO__I_-INOSITOL__1,4,5_-TRISPHOSPHATE_DEGRADATION.profile"
## [6] "MRNA_CAPPING.profile"
```

4.3 Copy number variants

Similarly, we construct networks based on shared overlap of CNVs. For each cellular pathway, we create a network consisting of patients with CNVs in the member genes of that pathway (or gene-set). The edge weight here is binary; all patients in the network have an edge weight of one. Those not in the network implicitly have a weight of zero.

Genomic events need to first be mapped to unit variables before being grouped into sets or pathways; here, ranges of CNV "events" need to be labelled by the gene which these overlap. This mapping is achieved by `mapNamedRangesToSets()`. The function used to construct networks from genomic events is `makePSN_RangeSets()`. As with the gene-expression nets, CNV nets are written to `profDir`. All input networks must be in the same directory. Interaction network names end with `_cont.txt`.

```
data(genes)
gene_GR     <- GRanges(genes$chrom,
   IRanges(genes$txStart,genes$txEnd),
   name=genes$name2)
path_GRList <- mapNamedRangesToSets(gene_GR,pathwayList)
names(path_GRList) <- paste("CNV_",names(path_GRList),sep="")
## warning: this step can take 2-5 minutes depending on the
## number of processes running in parallel
netList2 <- makePSN_RangeSets(cnv_GR, path_GRList,profDir,verbose=F)

##  LOCUS_NAMES column not provided; computing overlap of patients
##  with regions
## * Preparing patient-locus matrix
##  232 unique patients, 9127 unique locus symbols
##
## ....................
## ................

cat(sprintf("CNV: Got %i networks\n",length(netList2)))

## CNV: Got 1622 networks
```

Let's take a look at CNV-based networks:

```
head(unlist(netList2))

## [1] "CNV_RETINOL_BIOSYNTHESIS_cont.txt"
## [2] "CNV_MUCIN_CORE_1_AND_CORE_2__I_O__I_-GLYCOSYLATION_cont.txt"
## [3] "CNV_SUPERPATHWAY_OF_D-_I_MYO__I_-INOSITOL__1,4,5_-TRISPHOSPHATE_METABOLISM_cont.txt"
## [4] "CNV_D-_I_MYO__I_-INOSITOL__1,4,5_-TRISPHOSPHATE_DEGRADATION_cont.txt"
## [5] "CNV_MRNA_CAPPING_cont.txt"
## [6] "CNV_GLUTATHIONE-MEDIATED_DETOXIFICATION_cont.txt"
```

4.4 Integrate input nets into GeneMANIA database

Once all our patient networks are constructed, these are integrated into a single GeneMANIA database for feature selection.

```
# now create database
dbDir   <- GM_createDB(profDir, pheno$ID, outDir,numCores=numCores)

## Got 3423 networks
##  * Creating placeholder files
##  * Populating database files, recoding identifiers
##  * Converting profiles to interaction networks
##    user  system elapsed
```

```
##   5.356   0.372 585.401
## Got 3423 networks from 3423 profiles
##  * Build GeneMANIA index
##  * Build GeneMANIA cache
##   * Cleanup
```

5 Feature selection

Figure 1B shows the schematic for feature selection. The goal of this step is to extract the networks that are most predictive of a given class. For each subtype, here "LumA" and "other", feature selection is performed once (the large outer `for` loop). The key functions are:

- `GM_runCV_featureSet()`, which runs the cross-validation with successive GeneMANIA queries
- `GM_networkTally()`, which loops over all network rank files (or NRANK files) and computes the network score

```
## repeat process for each class
for (g in subtypes) {
    pDir <- sprintf("%s/%s",outDir,g)
    if (file.exists(pDir)) unlink(pDir,recursive=TRUE)
        dir.create(pDir)

        cat(sprintf("\n******\nSubtype %s\n",g))
        pheno_subtype <- pheno

        ## label patients not in the current class as a residual
        pheno_subtype$STATUS[which(!pheno_subtype$STATUS %in% g)] <- "nonpred"
        ## sanity check
        print(table(pheno_subtype$STATUS,useNA="always"))

        resDir    <- sprintf("%s/GM_results",pDir)
        ## query for feature selection comprises of training
        ## samples from the class of interest
        trainPred <- pheno$ID[which(pheno$STATUS %in% g)]

        # Cross validation
        GM_runCV_featureSet(trainPred, resDir, dbDir$dbDir,
                nrow(pheno_subtype),verbose=T, numCores=numCores,
                GMmemory=GMmemory)

        # patient similarity ranks
```

```
        prank <- dir(path=resDir,pattern="PRANK$")
        # network ranks
        nrank <- dir(path=resDir,pattern="NRANK$")
        cat(sprintf("Got %i prank files\n",length(prank)))

    # Compute network score
        pTally          <- GM_networkTally(paste(resDir,nrank,sep="/"))
        head(pTally)
        # write to file
        tallyFile       <- sprintf("%s/%s_pathway_CV_score.txt",resDir,g)
        write.table(pTally,file=tallyFile,sep="\t",col=T,row=F,quote=F)
}
##
## ******
## Subtype LumA
##
##    LumA nonpred    <NA>
##     103     129       0
##  Writing GM queries: Setting seed for reproducibility: 42
## Read 103 IDs
##  10-fold CV
##  Each iter will sample 92 records, 10 will be test
## chunk 1: 10 test (1-10); 93 query
## chunk 2: 10 test (11-20); 93 query
## chunk 3: 10 test (21-30); 93 query
## chunk 4: 10 test (31-40); 93 query
## chunk 5: 10 test (41-50); 93 query
## chunk 6: 10 test (51-60); 93 query
## chunk 7: 10 test (61-70); 93 query
## chunk 8: 10 test (71-80); 93 query
## chunk 9: 10 test (81-90); 93 query
## chunk 10: 13 test (91-103); 90 query
## 1 2 3 4 5 6 7 8 9 10 Got 10 prank files
##
## ******
## Subtype other
##
## nonpred   other    <NA>
##     103     129       0
##  Writing GM queries: Setting seed for reproducibility: 42
## Read 129 IDs
##  10-fold CV
##  Each iter will sample 116 records, 13 will be test
## chunk 1: 13 test (1-13); 116 query
## chunk 2: 13 test (14-26); 116 query
```

```
## chunk 3: 13 test (27-39); 116 query
## chunk 4: 13 test (40-52); 116 query
## chunk 5: 13 test (53-65); 116 query
## chunk 6: 13 test (66-78); 116 query
## chunk 7: 13 test (79-91); 116 query
## chunk 8: 13 test (92-104); 116 query
## chunk 9: 13 test (105-117); 116 query
## chunk 10: 12 test (118-129); 117 query
## 1 2 3 4 5 6 7 8 9 10 Got 10 prank files
```

6 Rank test patients using trained model

Following feature selection (previous section), we have identified the networks that are predictive of our two classes of interest: LumA and other. For each of these classes, we now create a single GeneMANIA database comprising *only of the feature selected nets* ; this is equivalent to our trained model for each class. We rank the similarity of a test patient to each class via a GeneMANIA query; the query consists of training samples from the corresponding class. For example:

- $Rank_{LumA}$: GeneMANIA rank for similarity to *training 'LumA' samples*
- $Rank_{other}$: GeneMANIA rank for similarity to *training 'other' samples*
- Final rank = $max(Rank_{LumA}, Rank_{other})$

The following code block does all these steps:

1. `makePSN_NamedMatrix, makePSN_RangeSets`: Create patient nets for the feature-selected networks using both training and test samples
2. `GM_createDB`: Create the new database from the resulting nets
3. `runGeneMANIA`: Run the query with the training samples
4. `GM_getQueryROC`: Get patient rankings

```
# now create GM databases for each class
# should contain train + test patients
# and be limited to nets that pass feature selection
pheno <- pheno_FULL
predRes <- list()
for (g in subtypes) {
        pDir <- sprintf("%s/%s",outDir,g)
        # get feature selected net names
        pTally <- read.delim(
```

```
                sprintf("%s/GM_results/%s_pathway_CV_score.txt",pDir,g),
                sep="\t",h=T,as.is=T)
        pTally <- pTally[which(pTally[,2]>=cutoff),1]
        pTally <- sub(".profile","",pTally)
        pTally <- sub("_cont","",pTally)

        cat(sprintf("%s: %i pathways\n",g,length(pTally)))
        profDir <- sprintf("%s/profiles",pDir)

        # prepare nets for new db
        tmp <- makePSN_NamedMatrix(xpr_FULL,rownames(xpr),
                pathwayList[which(names(pathwayList)%in% pTally)],
                profDir,verbose=F,numCores=numCores,writeProfiles=TRUE)
        tmp <- makePSN_RangeSets(cnv_FULL,
                path_GRList[which(names(path_GRList)%in% pTally)],
                profDir,verbose=FALSE)
        # create db
        dbDir <- GM_createDB(profDir,pheno$ID,pDir,numCores=numCores)

        # query of all training samples for this class
        qSamps <- pheno$ID[which(pheno$STATUS %in% g & pheno$TT_STATUS%in%"TRAIN")]
        qFile <- sprintf("%s/%s_query",pDir,g)
        GM_writeQueryFile(qSamps,"all",nrow(pheno),qFile)

        resFile <- runGeneMANIA(dbDir$dbDir,qFile,resDir=pDir)

        predRes[[g]] <- GM_getQueryROC(sprintf("%s.PRANK",resFile),pheno,g)
}
```

```
## LumA: 57 pathways
##  LOCUS_NAMES column not provided; computing overlap of patients
##  with regions
## * Preparing patient-locus matrix
##  348 unique patients, 241 unique locus symbols
##
## ...Got 57 networks
##  * Creating placeholder files
##  * Populating database files, recoding identifiers
##  * Converting profiles to interaction networks
##    user  system elapsed
##   0.380   0.012  31.516
## Got 57 networks from 57 profiles
##  * Build GeneMANIA index
##  * Build GeneMANIA cache
##   * Cleanup[1] "java -d64 -Xmx6G -cp /home/spai/R/x86_64-pc-linux-gnu-library/3.2/netDx/j
## * Attempt 1 : LumA_query
```

```
## other: 67 pathways
##  LOCUS_NAMES column not provided; computing overlap of patients
##  with regions
## * Preparing patient-locus matrix
##  348 unique patients, 2442 unique locus symbols
##
## ...Got 67 networks
##  * Creating placeholder files
##  * Populating database files, recoding identifiers
##  * Converting profiles to interaction networks
##    user  system elapsed
##   0.100   0.016  23.270
## Got 67 networks from 67 profiles
##  * Build GeneMANIA index
##  * Build GeneMANIA cache
##   * Cleanup[1] "java -d64 -Xmx6G -cp /home/spai/R/x86_64-pc-linux-gnu-library/3.2/netDx/j
## * Attempt 1 : other_query
```

7 Assign labels to test patients

In the last section, we obtained two similarity ranks for each test patient. Here we use `GM_OneVAll_getClass()` to label patients by max rank.

```
predClass <- GM_OneVAll_getClass(predRes)

## *** 232 rows have an NA prediction

cat("Predicted classes\n")

## Predicted classes
```

Finally, we evaluate the performance of the classifier.

```
both <- merge(x=pheno,y=predClass,by="ID")
print(table(both[,c("STATUS","PRED_CLASS")]))

##         PRED_CLASS
## STATUS   LumA other
##   LumA     47     4
##   other     9    56

pos <- (both$STATUS %in% "LumA")
tp <- sum(both$PRED_CLASS[pos]=="LumA")
fp <- sum(both$PRED_CLASS[!pos]=="LumA")
tn <- sum(both$PRED_CLASS[!pos]=="other")
```

```r
fn <- sum(both$PRED_CLASS[pos]=="other")
cat(sprintf("Accuracy = %i of %i (%i %%)\n",tp+tn,nrow(both),
                         round(((tp+tn)/nrow(both))*100)))

## Accuracy = 103 of 116 (89 %)

cat(sprintf("PPV = %i %%\n", round((tp/(tp+fp))*100)))

## PPV = 84 %

cat(sprintf("Recall = %i %%\n", round((tp/(tp+fn))*100)))

## Recall = 92 %
```

8 sessionInfo

```
sessionInfo()

## R version 3.2.5 (2016-04-14)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Debian GNU/Linux 7 (wheezy)
##
## locale:
## [1] C
##
## attached base packages:
## [1] stats4    parallel  stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] netDx.examples_0.0.0.9000 netDx_0.9
##  [3] RColorBrewer_1.1-2        pracma_1.9.3
##  [5] ROCR_1.0-7                gplots_3.0.1
##  [7] GenomicRanges_1.18.4      GenomeInfoDb_1.2.4
##  [9] IRanges_2.0.1             S4Vectors_0.4.0
## [11] BiocGenerics_0.12.1       combinat_0.0-8
## [13] doParallel_1.0.10         iterators_1.0.7
## [15] foreach_1.4.2             bigmemory_4.5.8
## [17] bigmemory.sri_0.1.3       knitr_1.13
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.2        XVector_0.6.0      quadprog_1.5-5
##  [4] plyr_1.8.3         stringr_0.6.2      highr_0.6
##  [7] caTools_1.17.1     tools_3.2.5        KernSmooth_2.23-13
## [10] gtools_3.5.0       reshape2_1.4.1     formatR_1.4
## [13] bitops_1.0-6       codetools_0.2-9    evaluate_0.9
## [16] gdata_2.17.0       compiler_3.2.5
```

9 References

1. Pai et al. (2016). netDx: A patient classifier based on integration of patient similarity networks. *ms in prep* 2. Parker JS et al. (2009) J Clin Oncol. 27 (8):1160-7. 3. The Cancer Genome Atlas (2012). Comprehensive molecular portraits of human breast tumours *Nature* **490**:61-70.