Chromosomal Instability Determines Taxane Response

Swanton *et. al.*

July 4, 2008

The following R code describes the microarray meta-anlaysis to derive a Taxane response gene signature and its relationship to chromosomal instability (CIN). The presented R code is only exemplary for one dataset. The listed meta-analysis methods start after the preprocessing step of the microarray data. The original meta-analysis includes six microarray datasets prepared on different microarray platforms. The meta-analysis is peformed via a binomial test with probability correction.

1 Description of the algorithm for the binomial test with probability correction

Step1: Select S similar differential expression analyses.

Step2: Use genes of all datasets for the meta-analysis.

Step3:
Genes are sorted by the number of datasets in which they are present.

Step4:
Binomial test to filter for genes significantly up or down regulated on a significance level α.

Step5:
Calculate weighted mean across experiments. The weight is the inter-quartile range of the q values in each experiment. The weighted mean is calculated by

$$\bar{q}_{w,i} = \frac{\sum_{j=1}^{n} w_j q_{i,j}}{\sum_{j=1}^{n} w_j}$$

with

$$w_j = IQR_j.$$

The weighted q value is calculated for each gene i in dataset j. $q_{i,j}$ is the q value of gene i in dataset j. The weighted q value is calculated for each gene i. To filter the genes of the signature one chooses a threshold for the weighted q values.

2 Meta-analysis

The preprocessing step is performed via the bioconductor R-package *limma*. To obtain the significant genes of the preprocessing step the customized function *sigGenesFkt* is used. The input parameter *fit2Bani* is the output of the preprocessing of one microarray dataset.
The customized function *setUpTableInfo* creates a table with all information of the microarray analysis. The information is stored in *infoBani*.

```
BanisigGenesMethods <- sigGenesFkt(fit2Bani,2,2,0.05)
infoBani <- list()
infoBani[[1]] <- setUpTableInfo(BanisigGenesMethods[[1]]$topTableFC,MABani$platform)
infoBani[[2]] <- setUpTableInfo(BanisigGenesMethods[[2]]$topTableFC,MABani$platform)
```

The obtained information of all datasets (for late, early timepoints and high, low concentrations) is stored in in the list *pvalueFCListLate*.

```
pvalueFCListLate <- list()
pvalueFCListLate[[1]]  <- infoLi[[2]]      [,c(1,2,5,6,7,8,10,12,13,15)]
pvalueFCListLate[[2]]  <- infoChenE[[1]]  [,c(1,2,4,5,6,7,9,11,12,14)]   # PTX
pvalueFCListLate[[3]]  <- infoChenE[[2]]  [,c(1,2,4,5,6,7,9,11,12,14)] # Epo
pvalueFCListLate[[4]]  <- infoKim[[1]]     [,c(1,2,10,6,7,8,11,12,13,15)]
pvalueFCListLate[[5]]  <- infoHVG[[2]]   [,c(1,2,15,9,12,13,16,17,18,20)]
pvalueFCListLate[[6]]  <- infoHVG[[3]]   [,c(1,2,15,9,12,13,16,17,18,20)]
pvalueFCListLate[[7]] <- infoHVG[[5]]   [,c(1,2,15,9,12,13,16,17,18,20)]
pvalueFCListLate[[8]] <- infoHVG[[6]]   [,c(1,2,15,9,12,13,16,17,18,20)]
pvalueFCListLate[[9]] <- infoBani[[2]]   [,c(1,2,13,8,10,11,14,16,17,19)]#late


columnlabels
for(i in 1:(dim(summary(pvalueFCListLate))[1])){
colnames(pvalueFCListLate[[i]]) <-
  c("rank","rankPercent","EntrezGeneID","GeneSymbol","HUGOSymbol"}
,"HGNCID","logFC","t","P.Value","platform")
}
```

The names of the datasets correspond to the following cancer cell lines:

Li	Human prostate cancer cell line - PC3
ChenE	Human lung carcinoma cell line - A549
Kim	Human lung carcinoma cell line - H460
HVG	Human breast carcinoma cell lines - MCF7 and MDA-MB-231
Bani	Human ovarian carcinoma xenografts - 1A9

The customized function *averageProbIdentifierZScoreListFktA* calculates the average of multiple values for one gene linked by an identifier (e.g. Entrez Gene ID). The result is stored in *averagepvalueFCListLateA*.

```
averagepvalueFCListLateA <-
averageProbIdentifierZScoreListFktA(pvalueFCListLate,"EntrezGeneID")

# average Chen again, because separation of concentration and treatment
pValFCListChen <- list()
```

```
pValFCListChen[[1]]  <- infoChenTC[[1]]  [,c(1,2,4,5,6,7,9,11,12,14)]    # PTX high
pValFCListChen[[2]]  <- infoChenTC[[2]]  [,c(1,2,4,5,6,7,9,11,12,14)]    # PTX low
pValFCListChen[[3]]  <- infoChenTC[[3]]  [,c(1,2,4,5,6,7,9,11,12,14)]    # Epo high

# columnlabels
for(i in 1:(dim(summary(pValFCListChen))[1])){
  colnames(pValFCListChen[[i]]) <-
   c("rank","rankPercent","EntrezGeneID","GeneSymbol","HUGOSymbol",
  "HGNCID","logFC","t","P.Value","platform")
}

# average multiple values again for Chen data
averageChenTCA <- averageProbIdentifierZScoreListFktA(pValFCListChen,"EntrezGeneID")
```

The list *averagePValGCListLateConcA* includes the combined data of all averaged datasets.

```
averagePValGCListLateConcA <- list()
averagePValGCaveragePValGCListLateConcAListLateConcA[[1]] <- averageChenTCA[[1]]
 # Chen PTX high (18 and 45)
averagePValGCListLateConcA[[2]] <- averageChenTCA[[2]]
 #Chen PTX low  (4 to 8)
averagePValGCListLateConcA[[3]] <- averageChenTCA[[3]]
  # Chen Epo high  (40)
averagePValGCListLateConcA[[4]] <- averagepvalueFCListLateA[[4]]
 # Kim   24h 5nM
averagePValGCListLateConcA[[5]] <- averagepvalueFCListLateA[[5]]
 # HV  MCF7 100nM 24 and 48h
averagePValGCListLateConcA[[6]] <- averagepvalueFCListLateA[[6]]
 # HV  MCF7 4 nM 24-48h
averagePValGCListLateConcA[[7]] <- averagepvalueFCListLateA[[8]]
 # HV  MDA 100nM 24 and 48h
averagePValGCListLateConcA[[8]] <- averagepvalueFCListLateA[[9]]
 # Bani 24h
```

The vector *expNamePvalueFCListLateConcRank* includes the labels of all datasets:

```
expNamePvalueFCListLateConcRank  <- c("ChenPTX16A45nM18h","ChenPTX4A8nM18h",
"ChenEpo40nM18h","Kim5nM24h","MCF7100nM24A48h","MCF74nM24A48h",
"MDA100nM24A48h","Bani60mg24h")
```

The vector *numbRepPvalueFCListLateConcRank* includes the inter-quartile range calculated by *IQR* .

```
IQR(as.numeric(averagePValGCListLateConcA[[5]][,"qValueMean"]),na.rm=TRUE)
# generate IQR vector}
numbRepPvalueFCListLateConcRank <- c(0.45,0.46,0.5,0.56,0.45,0.39,0.44,0.6)
```

metaAnaListLateConclogFC100EGIDRank is the result of the meta-analysis function. The signature of up- and down-regulated genes is stored in *sigGenes.logFC.100.EGID.ListLateConcRank.up* and *sigGenes.logFC.100.EGID.ListLateConcRank.down*. The significance level α of the binomial test which filters the significantly up or down regulated genes is set to 0.05.

```
metaAnaListLateConclogFC100EGIDRank <-
metaAnalysisFkt(averagePValGCListLateConcA,100,"logFC","EntrezGeneID",
expNamePvalueFCListLateConcRank,numbRepPvalueFCListLateConcRank,0)

metaAnaListLateConclogFC100EGIDRank$overlapFC[[1]]<-
 as.matrix(metaAnaListLateConclogFC100EGIDRank$overlapFC[[1]])

sigGenes.logFC.100.EGID.ListLateConcRank.down <-
 metaAnaListLateConclogFC100EGIDRank$overlapFC[[1]][metaAnaListLateConclogFC100EGIDRank$
 overlapFC[[1]][,"binomTestPValueDown"]<=0.05,]

sigGenes.logFC.100.EGID.ListLateConcRank.up <-
 metaAnaListLateConclogFC100EGIDRank$overlapFC[[1]][metaAnaListLateConclogFC100EGIDRank$
 overlapFC[[1]][,"binomTestPValueUp"]<=0.05,]
```

In *sigGenes.logFC.100.EGID.ListLateConcRank.down.QVal04Rank* and *sigGenes.logFC.100.EGID.ListLateConcRank.up.QVal04Rank* are the weighted mean q values for each gene obtained by the binomial test with probability correction. The threshold for the weighted q values is set to 0.4.

```
sigGenes.logFC.100.EGID.ListLateConcRank.down.QVal04Rank <-
  sigGenes.logFC.100.EGID.ListLateConcRank.down
  [sigGenes.logFC.100.EGID.ListLateConcRank.down[,"weightedMeanQVal"]<0.4,]

sigGenes.logFC.100.EGID.ListLateConcRank.up.QVal04Rank <-
sigGenes.logFC.100.EGID.ListLateConcRank.up
[sigGenes.logFC.100.EGID.ListLateConcRank.up[,"weightedMeanQVal"]<0.4,]
```

3 Customized functions used for the meta-analysis

3.1 setUpTableInfo

Input:
The input variable *tableOutput* is the output of the table function in the *limma* package. *platf* is the name of the used platform. The length of the *platf* vector must be the same as the number genes.

Output:
sigGenesFkt delivers a table with all information from analysis.

```
setUpTableInfo <- function(tableOutput,platf){
   n <- nrow(tableOutput)$
    $rankNu <- 1:nrow(tableOutput)$
    rankPercent <- c(1:n)*0

    for(k in 1:n){
      rankPercent[k] <- (rankNu[k]/n)*100
    }
```

```
    tableInfo <- cbind(rankNu,rankPercent,tableOutput,platf)
    colnames(tableInfo) <- c("rank","rankPercent",colnames(tableOutput),"platform")
print(c("function: setUpTableInfo() ok!"))
return(tableInfo )
}
```

3.2 sigGenesFkt

Input:
The input variable *fit2HVConc* is the output of the microarray analysis. *pVal* is the significance level α and *fc* is the value for the threshold.

Output:
The output of *sigGenesFkt* is the number of significant genes obtained with three different methods (one-sample t-test, moderated t-test with adjusted p-values, moderated t-test with unadjusted p-values)

```
sigGenesFkt <- function(fit2HVConc,numb,fc,pVal){
   fit2HVConcMethodsSigGenes <- list()
   for(i in 1:numb){
     nafilter <- apply(as.matrix(fit2HVConc$coefficients[,i]),1,function(x) is.na(x))
     num <- length(fit2HVConc$coefficients[!nafilter,i])
     print(num)
     # BH adjusted
     fit2HVConcMethodsSigGenes[[i]] <- list()
     fit2HVConcMethodsSigGenes[[i]]$topTableBH <-
     topTable(fit2HVConc[!nafilter,],coef=i,number=num,genelist=fit2HVConc$genes[!nafilte
     adjust.method="BH",sort.by="P")

     fit2HVConcMethodsSigGenes[[i]]$pValBHAdj <- fit2HVConcMethodsSigGenes[[i]]$
     topTableBH[fit2HVConcMethodsSigGenes[[i]]$topTableBH$P.Value <= pVal,]

     fit2HVConcMethodsSigGenes[[i]]$pValBHAdjUp <-
     fit2HVConcMethodsSigGenes[[i]]$pValBHAdj[fit2HVConcMethodsSigGenes[[i]]$pValBHAdj$M

     fit2HVConcMethodsSigGenes[[i]]$pValBHAdjDown <-
     fit2HVConcMethodsSigGenes[[i]]$pValBHAdj[fit2HVConcMethodsSigGenes[[i]]$pValBHAdj$M
     print(dim(fit2HVConcMethodsSigGenes[[i]]$pValBHAdjUp))
     print(dim(fit2HVConcMethodsSigGenes[[i]]$pValBHAdjDown))

     # not adjusted
     fit2HVConcMethodsSigGenes[[i]]$topTableNotAd <-

   topTable(fit2HVConc[!nafilter,],coef=i,number=num,
   genelist=fit2HVConc$genes[!nafilter,],adjust.method="none",sort.by="P")

     fit2HVConcMethodsSigGenes[[i]]$pValNotAd <-
     fit2HVConcMethodsSigGenes[[i]]$
     topTableNotAd[fit2HVConcMethodsSigGenes[[i]]$topTableNotAd$P.Value <= pVal,]

     fit2HVConcMethodsSigGenes[[i]]$pValNotAdUp <-
     fit2HVConcMethodsSigGenes[[i]]$pValNotAd[fit2HVConcMethodsSigGenes[[i]]$pValNotAd$M
```

```
      fit2HVConcMethodsSigGenes[[i]]$pValNotAdDown <-
      fit2HVConcMethodsSigGenes[[i]]$pValNotAd[fit2HVConcMethodsSigGenes[[i]]$pValNotAd$M
      print(dim( fit2HVConcMethodsSigGenes[[i]]$pValNotAdUp))
      print(dim( fit2HVConcMethodsSigGenes[[i]]$pValNotAdDown))
      #FC
      fit2HVConcMethodsSigGenes[[i]]$topTableFC <-
      topTable(fit2HVConc[!nafilter,],coef=i,number=num,genelist=fit2HVConc$genes[!nafilte
      adjust.method="none",sort.by="M")

      fit2HVConcMethodsSigGenes[[i]]$FC2Up <-
      fit2HVConcMethodsSigGenes[[i]]$
      topTableFC[fit2HVConcMethodsSigGenes[[i]]$topTableFC$M > fc,]

      print(dim( fit2HVConcMethodsSigGenes[[i]]$FC2Up))

      fit2HVConcMethodsSigGenes[[i]]$FC2Down <-
      fit2HVConcMethodsSigGenes[[i]]$
      topTableFC[fit2HVConcMethodsSigGenes[[i]]$topTableFC$M < -fc,]
      print(dim( fit2HVConcMethodsSigGenes[[i]]$FC2Down))
      rm(nafilter,num)
      }
    return(fit2HVConcMethodsSigGenes)
  }
```

3.3 averageProbIdentifierZScoreListFkt

Input:
The input variable *identifiername* is the identifier (e.g. "EntrezGeneID") in character-format (""). *valueTableList* is a list of tablse with different values and information on probes.

Output:
The output is a matrix with one row per gene and the columns: "Identifier","target","madTarget" ,"medianTarget", "minTarget","maxTarget","zScoreMedian","numberProbes"

```
averageProbIdentifierZScoreFktA <- function(valueTable,identifierName){

   # simplify input data by set attribut in function:
   targetName <-  "logFC"

   # set up output frame
   outputData <- as.data.frame(matrix(0,dim(valueTable)[1],14))
   colnames(outputData) <- c(identifierName,
    paste("mad",targetName,sep="_"),
    paste("median",targetName,sep="_"),
                    paste("min",targetName,sep="_"),
                    paste("max",targetName,sep="_"),
                    paste("zScoreMean",targetName,sep="_"),
                    "numberProbes","minRankPercent","minPValue","maxPValue",
                    "fisherStatistic","qValueMean","minQValue","maxQValue")

# do each identifier merge only one time and therefore set up
```

```
# a memoryvector
  memoryvector <- c(1:nrow(valueTable))*0

  # i = probes
for(i in 1:nrow(valueTable)){
# for(i in 1:5){

  # only if not used before
    if(memoryvector[i]==0){
    if(!is.na(valueTable[i,identifierName])){
      identifier <- as.character(valueTable[i,identifierName])
      # table with multiple values to one identifier
      findProbes <- na.omit(valueTable[valueTable[,identifierName]==identifier,])
      print(findProbes)

    # ask if rows in findProbes in more than one
       if(nrow(findProbes)>1){

         # calculate statistical values for the target
         madTarget <- mad(as.numeric(findProbes[,targetName]),na.rm=TRUE)
         medianTarget <- median(as.numeric(findProbes[,targetName]),na.rm=TRUE)
         minTarget <- min(as.numeric(findProbes[,targetName]),na.rm=TRUE)
         maxTarget <- max(as.numeric(findProbes[,targetName]),na.rm=TRUE)
         # mean and sd for z score
         meanTarget <- mean(as.numeric(findProbes[,targetName]),na.rm=TRUE)
         sdTarget <- sd(as.numeric(findProbes[,targetName]),na.rm=TRUE)
         number <- nrow(findProbes)

         # additional info about intra experimental rank
         minIntraAverageRankPercent <- min(as.numeric(findProbes[,"rankPercent"]),na.rm=TR
         # info about pvalue range
         minPValue <- min(as.numeric(findProbes[,"P.Value"]),na.rm=TRUE)
         maxPValue <- max(as.numeric(findProbes[,"P.Value"]),na.rm=TRUE)

         # calculate q value for each p value
         # calculate z score for each fold change expression
         zScore <- 1:number*0
         qValue <- 1:number*0
         for(j in 1:number){
          zScore[j] <-  ((as.numeric(findProbes[j,targetName])-meanTarget)/sdTarget)
          #print(zScore[j])
          qValue[j] <-
          ((as.numeric(findProbes[j,"P.Value"])*nrow(valueTable))/
          as.numeric(findProbes[j,"rank"]))
          #print(qValue[j])
         }
         # summarize q values
         qValueMean <- mean(qValue,na.rm=TRUE)
         #print(qValueMean)
         minQValue <- min(as.numeric(qValue),na.rm=TRUE)
         maxQValue <- max(as.numeric(qValue),na.rm=TRUE)
```

```
        # summarize z score
        zScoreMean <- mean(as.numeric(zScore),na.rm=TRUE)
        #print(zScoreMean)
        # calculate fisher's combination test (p value can be read up in
        # chisq table with df = number p values)
        fisherStatistic <- -2*(sum(log(findProbes[,"P.Value"])))
        # log(x, base = exp(1))


        # save info in file
        outputData[i,] <- c(identifier,madTarget,
                            medianTarget,minTarget,
                            maxTarget,zScoreMean,number,
                          minIntraAverageRankPercent,minPValue,maxPValue,
                          fisherStatistic,qValueMean,minQValue,maxQValue)
        print(outputData[i,])

        # set memoryvector
        for(k in 1:nrow(findProbes)){
        memoryvector[findProbes[k,"rank"]] <- 1
        }

      } else {
      # if only one row than only one qvalue
      # qvalue
       qValueMean <-
         ((as.numeric(findProbes["P.Value"]))*nrow(valueTable))/(as.numeric(findProbes["ra
       minQValue <- as.numeric(qValueMean)
       maxQValue <- as.numeric(qValueMean)

      # save info in file
       outputData[i,] <- c(identifier,0,valueTable[i,targetName],
                           valueTable[i,targetName],valueTable[i,targetName],0,1,
                           valueTable[i,"rankPercent"],valueTable[i,"P.Value"],
                           valueTable[i,"P.Value"],0,qValueMean,minQValue,maxQValue)
      print(outputData[i,])
      memoryvector[i] <- 1
      }
    }
    memoryvector[i] <- 1
    }  # memoryvector
  }
outputData <- outputData[outputData[,1]!=0,]
print(c("function: averageProbIdentifierFktA ok!"))
return(outputData)
}
```

3.4 metaAnalysisFkt

Input:

valuesAverage : list where each entry is one dataset in matrix format
(output function averageProbFkt ( ))
limit : threshold to prefilter the genes
targetName : if method : rank , BT,BTwPC: logFC ,
if method : Rhodes : qValueMean
identifierName: i. e. EntrezGeneID or other identifier
expName : vector with experiment names, length is number experiments
numbRep : weighting (i. e. IQR or number replicates)
direction : 0=no direction
1=only down-regulated genes
2=only up-reglated genes

Output:
The output are matrices with information to overlapping genes across all datasets (step3 in meta analysis).

```
metaAnalysisFkt <- function(valuesAverage,limit,targetName,identifierName,
expName,numbRep,direction){

  fcRank <- list()

  #----------------------------------
  # set up colnames for output table
  colnamesExp <- c(as.character(identifierName),
  paste(expName[1],"mad_logFC",sep="_"),
  paste(expName[1],"median_logFC",sep="_"),
  paste(expName[1],"zScoreMean_logFC",sep="_"),
  paste(expName[1],"numberProbes",sep="_"),
  paste(expName[1],"RankIndExp",sep="_"),
  paste(expName[1],"fisherStatistic",sep="_"),
  # paste(expName[1],"qValueMedian",sep="_"),
  paste(expName[1],"qValueMean",sep="_"),
  paste(expName[1],"minQValue",sep="_"),
  paste(expName[1],"maxQValue",sep="_"),
  paste(expName[1],"minPValue",sep="_"),
  paste(expName[1],"maxPValue",sep="_"),
  paste(expName[1],"numberReplicates",sep="_"))


  # set up colnamesvector to later find only repressed
  listCoeff <- c(paste(expName[1],"median_logFC",sep="_"))

  # set up colname sfor q value calculation
  qValMean <- c(paste(expName[1],"qValueMean",sep="_"))

  ##set up colnames for numberProbes
  #nuProColnames <- c(paste(expName[1],"numberProbes",sep="_"))

  for(k in 2:nrow(summary(valuesAverage))){
```

```
  colnamesExp <-  c(colnamesExp,
  paste(expName[k],"mad_logFC",sep="_"),
  paste(expName[k],"median_logFC",sep="_"),
  paste(expName[k],"zScoreMean_logFC",sep="_"),
  paste(expName[k],"numberProbes",sep="_"),
  paste(expName[k],"RankIndExp",sep="_"),
  paste(expName[k],"fisherStatistic",sep="_"),
  #paste(expName[k],"qValueMedian",sep="_"),
  paste(expName[k],"qValueMean",sep="_"),
  paste(expName[k],"minQValue",sep="_"),
  paste(expName[k],"maxQValue",sep="_"),
  paste(expName[k],"minPValue",sep="_"),
  paste(expName[k],"maxPValue",sep="_"),
  paste(expName[k],"numberReplicates",sep="_"))

  listCoeff <- c(listCoeff,paste(expName[k],"median_logFC",sep="_"))

  qValMean <- c(qValMean,paste(expName[k],"qValueMean",sep="_"))
}
#---------------------------------
if(direction==0){

if(targetName=="logFC"){
# add a rank number to each gene in each experiment
fcRank$fcData <- rankDecreasingFktN(valuesAverage,targetName,identifierName)
  # define signatures by rank cut at limit
fcRank$CutValue <- selectGenesRankN(fcRank$fcData,limit)
}
if(targetName=="qValueMean"){
# define signatures by q value cut at limit
fcRank$CutValue <- selectGenesTargetN(valuesAverage,identifierName,limit)
 }
 } else {
 # if direction ==1 for "yes consider sign"
 # if method fold change
if(targetName=="logFC"){
fcRank$fcData <- rankDecreasingFktN(valuesAverage,targetName,identifierName)
fcRank$CutValue <- selectGenesSignRankN(fcRank$fcData,limit,direction)
 }
 # if method q value
if(targetName=="qValueMean"){
fcRank$CutValue <- selectGenesSignTargetN(valuesAverage,identifierName,limit,direction)
 }
 }

# how many genes in each signature?
fcRank$CutNumberGenesInSignatures <- numberGenesInEachSignature(fcRank$CutValue)

# count in how many experiments each gene appears
fcRank$CutCountGenes <- countGenesFkt(fcRank$CutValue,identifierName)
#test <- fcRank$CutCountGenes[order(fcRank$CutCountGenes[,2],decreasing=TRUE),]
```

```
   # number of genes which appear in each possible Signature number
  fcRank$CutNumberGenes <- askNumSign(fcRank$CutCountGenes)
  # Symbols of genes in overlap
  fcRank$CutGenesInSign <- askGenesSign(fcRank$CutCountGenes)

  # generate lists
  fcRank$overlapFC <- generateMetaList(fcRank$CutGenesInSign,fcRank$CutNumberGenes,
              fcRank$CutValue,identifierName,targetName,colnamesExp,listCoeff,
              qValMean,numbRep)

  if(identifierName=="EntrezGeneID"){
    #query HUGO library for Symbol
    #source("queryLibrary.R")
    fcRank$HUGOInfo <- list()
    for(i in 1: nrow(summary(fcRank$overlapFC))){
      print(i)

      fcRank$HUGOInfo[[i]] <-
      getInfoFromEGID(as.character(fcRank$overlapFC[[i]][,"EntrezGeneID"]))
      #print(dim(fcRank$HUGOInfo[[i]]))
    }

  for(j in 1:nrow(summary(fcRank$overlapFC))){
  # case: more than one identifier
  if(!is.null(dim(fcRank$overlapFC[[j]]))){
    HUGOSymbol <- as.character(fcRank$HUGOInfo[[j]][,"HUGOSymbol"])
   fcRank$overlapFC[[j]] <- data.frame(HUGOSymbol,fcRank$overlapFC[[j]])
   #print(fcRank$overlapFC[[j]][1:2,1:5])
   rm(HUGOSymbol)
   # case: only one identifier
   } else {
   HUGOSymbol <- as.character(fcRank$HUGOInfo[[j]]["HUGOSymbol"])
   fcRank$overlapFC[[j]] <- data.frame( HUGOSymbol,fcRank$overlapFC[[j]])
    fcRank$overlapFC[[j]] <- as.matrix( fcRank$overlapFC[[j]])
   #print(fcRank$overlapFC[[j]][1,1:5])
   rm(HUGOSymbol)
   }
  }
  }

rm(colnamesExp,qValMean,listCoeff,targetName,identifierName)

print(c("function: metaAnalysisFkt ok!"))
return(fcRank)
}
```