R code to run gene expression signatures

Document written by Nick Tobin, code by Nick Tobin and Kristian Wennmalm

Background

- This report contains the code we used to run the gene expression signatures in the paper "Multi-level gene expression signatures but not binary outperform Ki67 for the long term prognostication of breast cancer patients"
- We feel obliged to provide the disclaimer that whilst we are confident of this code in our hands and on our datasets, we cannot be responsible for ongoing trouble-shooting and error-messages when it is applied by others. Having said that, we are always happy to collaborate so please don't hesitate to contact me with queries/proposals here: nick.tobin@ki.se or tobin.nick@gmail.com. Note that all files used in this document can be retrieved from this public repository https://bitbucket.org/tobinn3/
- It is important to note that we used version 2.4.1 of the hgu133a.db and 2.5.0 of hgu133b.db packages from the Bioconductor website for mapping between the affy probes in the Uppsala and Stockholm datasets and gene names in the original signature annotations of the 70-gene, Sorlie, Hu and Parker signatures. Using updated annotation packages for the U133A and U133B arrays will invariably change the mapping of certain probes, which has a downstream effect on signature classifcations. It is of course encouraged to use the updated annotation files, but worth noting that the classifications will not be identical to those outlined below.

1. Genomic Grade Index (GGI)

- Here, we demonstrate our code using the Uppsala cohort, the same code was applied to Stockholm cohort. All tumours were profiled on both the Affymetrix U133A and U133B chips, but we only apply the GGI signature to the U133A data.

```r
# load in the RMA normalised Uppsala data
load("your.file.path/uppsalaA.RData")
uppA = uppsalaA

# Use the sig.gene GGI object from the R genefu package to get the list of
# genes used for the signature
library(genefu)
data(sig.ggi)
ggi_sig_upp = uppA[rownames(sig.ggi), ]
dim(ggi_sig_upp)
# [1] 128 253

# Download the weighting for each probe from the above repository, these can
# also be obtained from the 'grade' column of the sig.gene object, just
# replace '3' with 1 and '1' with -1.
setwd("your.file.path")
weights = read.table("genesandweightsU.txt", header = T, sep = "\t", row.names = 1)

# Matching probes?
identical(rownames(ggi_sig_upp), rownames(weights))
# [1] TRUE

# Multiply by probe weighting
probe_weighting = weights$Weights
```

```r
posnegdata = ggi_sig_upp * probe_weighting

# Add each column to get the unscaled and offset GGI score
rawggi = colSums(posnegdata)
rawggi_df = data.frame(rawggi)

# Calculate scale and offset. First download and read in the clincial
# grading information for this dataset and bind in the grade columns to the
# raw GGI score
setwd("your.file.path")
gradeinfo = read.table("gradeinfo.txt", header = T, sep = "\t", row.names = 1)
identical(rownames(rawggi_df), rownames(gradeinfo))
# [1] FALSE

rownames(gradeinfo) <- paste("X", rownames(gradeinfo))
rownames(gradeinfo) <- sub(" ", "", rownames(gradeinfo))
identical(rownames(rawggi_df), rownames(gradeinfo))
# [1] TRUE

rawggigrade = cbind(rawggi_df, gradeinfo)

# Get out means of each grade
meansbygrade = tapply(rawggigrade$rawggi, rawggigrade$grade, mean)

# Divide the data by half the difference between the means (scaling step)
division = (meansbygrade[3] - meansbygrade[1])/2

# Do the division
rawggigrade$beforesubtract = rawggigrade$rawggi/division

# Find the new means- note the difference of 2 between grade 1 and grade 3
meansbeforesubtract = tapply(rawggigrade$beforesubtract, rawggigrade$grade,
    mean)

# Offset step, subtract the grade 1 score (+1) from all values
subtract = meansbeforesubtract[1] + 1

# Subtract values to get scaled and offset ggi scores
rawggigrade$final_ggi_scores = rawggigrade$beforesubtract - subtract

# make a list of which tumours are 1's and which are 3's based on a cutoff
# of 0
rawggigrade$tf = rawggigrade$final_ggi_scores >= 0
w <- rep(1, 253)
rawggigrade$ggi_call = ifelse(rawggigrade$tf == TRUE, 3, 1)
table(rawggigrade$ggi_call)
```

- The above code is useful for understanding how the GGI signature should be run, but as the GGI calls from the original manuscript are based on a leave-one-out cross validation (LOOCV) a further step is required.

```r
# Scale LOOCV
scaleggi = function(data, weights, grade) {
    x = data
    y = weights
    z = y[1:128, 1]
    posneg = x * z
    rawggi = colSums(posneg)
    meansbygrade = tapply(rawggi, grade, mean)
    division = (meansbygrade[3] - meansbygrade[1])/2
}

# Create empty scale vector
scalev <- rep(NA, 253)
gradeinfov = gradeinfo$grade

# Loop scaling step over all samples
for (i in 1:253) {
    scalev[i] <- scaleggi(ggi_sig_upp[, -i], weights, gradeinfov[-i])
}

# Offset LOOCV
offggi = function(data, weights, grade) {
    x = data
    y = weights
    z = y[1:128, 1]
    posneg = x * z
    rawggi = colSums(posneg)
    meansbygrade = tapply(rawggi, grade, mean)
    division = (meansbygrade[3] - meansbygrade[1])/2
    beforesubtract = rawggi/division
    meansbeforesubtract = tapply(beforesubtract, grade, mean)
    subtract = meansbeforesubtract[1] + 1
}

# Create empty offset vector
offsetv <- rep(NA, 253)

# Loop offset step over all samples
for (i in 1:253) {
    offsetv[i] <- offggi(ggi_sig_upp[, -i], weights, gradeinfov[-i])
}

# raw ggi scores
rggi = function(data, weights, grade) {
    x = data
    y = weights
    z = y[1:128, 1]
    posneg = x * z
    rawggi = colSums(posneg)
}

rawggi = rggi(ggi_sig_upp, weights, gradeinfo)  # Raw GGI scores
LOOCV = data.frame(cbind(rawggi, scalev, offsetv))
LOOCV$division = (LOOCV$rawggi/LOOCV$scalev)
LOOCV$final_ggi_score_LOOCV = (LOOCV$division - LOOCV$offsetv)
LOOCV$tf = LOOCV$final_ggi_score_LOOCV >= 0
LOOCV$ggi_call = ifelse(LOOCV$tf == TRUE, 3, 1)  # Final GGI calls object used in the manuscript
```

```
save(LOOVC, file = "LOOCV.RData")
```

2. 70-gene

- As some of the gene signatures were not produced on the affymetrix platform, we need an annotation object to map through which includes gene symbols and ids. The 70-gene signature is the first example of this- it was produced on the Aligent Hu25K microarray. As such, we initially produce an annotation object with the mapping identifiers we're interested in for both the U133A and U133B chips using the bioconductor .db objects for both arrays.

```
library("annaffy")
library("hgu133a.db")
library("hgu133b.db")

# u133a first
Probe <- ls(hgu133aSYMBOL)
ll <- aafLocusLink(Probe, "hgu133a.db")
GeneID <- getText(ll)

gs <- aafSymbol(Probe, "hgu133a.db")
Symbol <- getText(gs)

gb <- aafGenBank(Probe, "hgu133a.db")
GenBank <- getText(gb)

ug <- aafUniGene(Probe, "hgu133a.db")
UniGene <- getText(ug)

cc <- aafChromosome(Probe, "hgu133a.db")
Chromosome <- getText(cc)

cl <- aafChromLoc(Probe, "hgu133a.db")
ChromLoc <- getText(cl)

cb <- aafCytoband(Probe, "hgu133a.db")
Cytoband <- getText(cb)

u133a <- cbind(Probe, GeneID, Symbol, GenBank, UniGene, Chromosome, ChromLoc,
    Cytoband)

save(u133a, file = "u133a.RData")

# Now u133b
Probe <- ls(hgu133bSYMBOL)
ll <- aafLocusLink(Probe, "hgu133b.db")
GeneID <- getText(ll)

gs <- aafSymbol(Probe, "hgu133b.db")
Symbol <- getText(gs)

gb <- aafGenBank(Probe, "hgu133b.db")
GenBank <- getText(gb)

ug <- aafUniGene(Probe, "hgu133b.db")
```

```
UniGene <- getText(ug)

cc <- aafChromosome(Probe, "hgu133b.db")
Chromosome <- getText(cc)

cl <- aafChromLoc(Probe, "hgu133b.db")
ChromLoc <- getText(cl)

cb <- aafCytoband(Probe, "hgu133b.db")
Cytoband <- getText(cb)

u133b <- cbind(Probe, GeneID, Symbol, GenBank, UniGene, Chromosome, ChromLoc,
    Cytoband)

save(u133b, file = "u133b.RData")
```

- Next read in the Aligent probe annotation for the 70-gene signature and map to Affymetrix identifiers

```
# Load in orinigal signature
load("your.file.path/original_70_annotation.RData")
NCBI70 = original_70_annotation

# Make affymetrix verions of signature, so we can pull these out from the
# Uppsala dataset
rownames(u133a) <- u133a[, "Probe"]
rownames(u133b) <- u133b[, "Probe"]

createTable2 <- function(sigob, sigobCN, chipob, chipobCN, chip) {

    ct <- function(sigobrow, sigobCN, chipob, chipobCN) {

        m <- sigobrow[sigobCN]
        rws <- chipob[, chipobCN] %in% m
        psets <- chipob[rws, "Probe"]
        if (length(psets) == 0) {
            psets <- NA
        }

        o <- matrix(sigobrow, nrow = length(psets), ncol = length(sigobrow),
            byrow = T)
        o <- cbind(o, psets)
        colnames(o) <- c(names(sigobrow), "ProbeSet")
        if (!is.na(psets[1])) {
            o
        }
    }

    output <- c()

    for (i in 1:dim(sigob)[1]) {

        nl <- ct(sigob[i, ], sigobCN, chipob, chipobCN)
        output <- rbind(output, nl)
    }
    cbind(output, Match = rep(chip, dim(output)[1]))
}

NCBI70m <- as.matrix(NCBI70)  #just creating an object that will agree with the function 'createTable2'
NCBI70m[, "LLID"] <- gsub(" ", "", NCBI70m[, "LLID"])
NCBI70m[, "Chromosome"] <- gsub(" ", "", NCBI70m[, "Chromosome"])

partAprobesLLID <- createTable2(NCBI70m, "LLID", u133a, "GeneID", "A")
partBprobesLLID <- createTable2(NCBI70m, "LLID", u133b, "GeneID", "B")
partAprobesGB <- createTable2(NCBI70m, "Acc", u133a, "GenBank", "A")
partBprobesGB <- createTable2(NCBI70m, "Acc", u133b, "GenBank", "B")
partAprobesUG <- createTable2(NCBI70m, "UGCluster", u133a, "UniGene", "A")
partBprobesUG <- createTable2(NCBI70m, "UGCluster", u133b, "UniGene", "B")

allparts <- rbind(partAprobesLLID, partBprobesLLID, partAprobesGB, partBprobesGB,
    partAprobesUG, partBprobesUG)
dv <- duplicated(allparts[, c("ORIGINALID", "ProbeSet", "Match")])
allparts <- allparts[!dv, ]
u133ab <- rbind(u133a, u133b)

MappableByLLID <- u133ab[allparts[, "ProbeSet"], "GeneID"] == allparts[, "LLID"]
MappableByGenBank <- u133ab[allparts[, "ProbeSet"], "GenBank"] == allparts[,
    "Acc"]
MappableByUniGene <- u133ab[allparts[, "ProbeSet"], "UniGene"] == allparts[,
    "UGCluster"]

NCBI70Affy <- cbind(allparts, MappableByLLID, MappableByGenBank, MappableByUniGene)
NCBI70Affy <- data.frame(NCBI70Affy, stringsAsFactors = F)  # Get expected warning message
NCBI70Affy <- NCBI70Affy[order(NCBI70Affy$ORIGINALID), ]

rownames(NCBI70Affy) <- 1:dim(NCBI70Affy)[1]
save(NCBI70Affy, file = "NCBI70Affy.RData")
```

- Now read in the Uppsala U133A and U133B files and run the signature

```
# UppsalaA
load("your.file.path/uppA.RData")

# UppsalaB
load("your.file.path/uppB.RData")

annot = NCBI70Affy

# Create a TF vector denoting which of the probes (from the U133A 1st, then
# U133B) are present in the uppA and B datasets respectively
uppANKI = rownames(uppA) %in% annot[(annot[, 12] == "A"), 11]
uppBNKI = rownames(uppB) %in% annot[(annot[, 12] == "B"), 11]

a = uppA[uppANKI, ]
b = uppB[uppBNKI, ]

dim(a)
# [1] 80 253
```

```r
dim(b)
# [1] 44 253

rmA = rowMeans(a)
rmB = rowMeans(b)
mcA = a - rmA
mcB = b - rmB

colnames(mcB) <- paste("X", colnames(mcB))
colnames(mcB) <- sub(" ", "", colnames(mcB))
mcB <- mcB[, colnames(mcA)]

# Are the column names identical in A and B?
identical(colnames(mcA), colnames(mcB))
# [1] TRUE

# Everything matches, so row bind to combine the probes
probes = rbind(mcA, mcB)
dim(probes)
# [1] 124 253

# Merge and concat
con = merge(annot, probes, by.x = "ProbeSet", by.y = "row.names", all.x = T)
dim(con)
# [1] 136 268

final = aggregate(con, list(con$ORIGINALID), mean)  # get warning, this is expected
values = final[, -(2:16)]
dim(values)
# [1] 67 254

rownames(values) <- values[, "Group.1"]
values = values[, -1]
values_df = data.frame(values)

# Now we use the original signature file again
# Load('your.file.path/annotation.RData') NCBI70=original_70_annotation

rownames(NCBI70) = NCBI70[, "ORIGINALID"]
original_correlations = subset(NCBI70, select = c("Averagegood"))
original_correlations$tf = rownames(original_correlations) %in% rownames(values_df)
matched_correlation_ob = subset(original_correlations, tf == TRUE)
dim(matched_correlation_ob)
# [1] 67 2

identical(rownames(matched_correlation_ob), rownames(values_df))
# [1] TRUE

seventy_gene = t(data.frame(cor(matched_correlation_ob[, "Averagegood"], values_df[1:253],
    use = "everything", method = c("pearson"))))
colnames(seventy_gene) = "seventy_gene_correlations"
```

- Finally, we have to determine the cutoff for good vs. poor prognosis. In the original manuscript the prognostic cutoff was set at a correlation value of 0.3. As detailed in the Supplementary materials of this manuscript, this is not appropriate in new datasets, so we need to find Uppsala data specific cutoff

```
# Load in the patient seventy gene correlations from the original paper
setwd("your.file.path")
NKI_pts_correlations = read.table("NKI_pts_correlations.txt", header = T, sep = "\t",
    row.names = 1)
NKI_pts = as.matrix(NKI_pts_correlations)
UPP_pts = seventy_gene[, "seventy_gene_correlations"]

# Rescale the NKI_pts correlation and Uppsala
library("genefu")
scaledNKIcor = rescale(NKI_pts, q = 0.05)
scaledUppcor = rescale(UPP_pts, q = 0.05)

# plot scaled and unscaled underneath each other
par(mfrow = c(2, 2))
plot(density(UPP_pts), col = "red", xlim = c(-1, 1.5), main = "Uppsala unscaled density")
abline(v = 0.3, col = "blue")
plot(density(NKI_pts), col = "black", xlim = c(-1, 1.5), main = "NKI unscaled density")
abline(v = 0.3, col = "blue")
plot(density(scaledUppcor), col = "red", xlim = c(-1, 1.5), main = "Uppsala scaled density")
abline(v = 0.3, col = "blue")
plot(density(scaledNKIcor), col = "black", xlim = c(-1, 1.5), main = "NKI scaled density")
abline(v = 0.3, col = "blue")

# Looking at the NKI scaled density plot, clearly the 0.3 cutoff is no
# longer appropriate, calculate a new cutoff for NKI and Uppsala datasets

lm(scaledNKIcor ~ NKI_pts)$coeff[2]
# NKI_pts 0.770

lm(scaledNKIcor ~ NKI_pts)$coeff[1]
# (Intercept) 0.38204

# y=mx+c
y = (0.77 * 0.3) + 0.38204
# y=0.61304

# Work out Uppsala cutoff for the unscaled data
lm(scaledUppcor ~ UPP_pts)$coeff[2]
# UPP_pts 0.8992891

lm(scaledUppcor ~ UPP_pts)$coeff[1]
# (Intercept) 0.5205539

# y=mx+c
0.61304 = (0.8992891 * x) + 0.5205539
x = (0.61304 - 0.5205539)/0.8992891
# x= 0.1028436

final_calls = ifelse((seventy_gene[, "seventy_gene_correlations"] > 0.1028),
    "Good", "Poor")
save(final_calls, file = "final_calls.RData")
```

3. p53 signature

- As the p53 signature was derived in the Uppsala cohort, we will show the code here to run the signature on the Stockholm cohort, with the Uppsala cohort as the data training set.

```r
# Make an object with the 32 signature genes
p53_at <- c("217889_s_at", "243929_at", "229975_at", "223864_at", "227081_at",
    "215014_at", "206869_at", "221585_at", "205440_s_at", "228969_at", "212949_at",
    "226067_at", "232855_at", "221520_s_at", "205472_s_at", "205186_at", "221275_s_at",
    "229030_at", "233413_at", "203438_at", "230378_at", "238581_at", "235343_at",
    "229150_at", "205734_s_at", "214079_at", "238746_at", "204623_at", "230863_at",
    "215047_at", "201710_at", "205009_at")

# Divide the probes according to whether they are from the 'A' or 'B' chip
AB <- c("A", "B", "B", "B", "B", "A", "A", "A", "A", "B", "A", "B", "B", "A",
    "A", "A", "A", "B", "B", "A", "B", "B", "B", "B", "A", "A", "B", "A", "B",
    "A", "A", "A")

p53_atA <- p53_at[AB == "A"]
p53_atB <- p53_at[AB == "B"]

# Load Uppsala data

# UppsalaA
load("your.file.path/uppA.RData")
# UppsalaB
load("your.file.path/uppB.RData")

# Same column order for A and B chips:
mcA = uppA
mcB = uppB
colnames(mcB) <- paste("X", colnames(mcB))
colnames(mcB) <- sub(" ", "", colnames(mcB))
mcB <- mcB[, colnames(mcA)]

# Are the column names identical in A and B?
identical(colnames(mcA), colnames(mcB))
# [1] TRUE

upp32 <- rbind(mcA[p53_atA, ], mcB[p53_atB, ])
upp32 <- t(scale(t(upp32)))
dim(upp32)
# [1] 32 253

# Load in the original Uppsala p53 calls
load("your.file.path/p53.RData")
rownames(p53) <- paste("X", rownames(p53))
rownames(p53) <- sub(" ", "", rownames(p53))
upp32 <- upp32[, rownames(p53)]

# source('http://www.bioconductor.org/biocLite.R'); biocLite('supclust')
library("supclust")
p53status <- p53$p53status
p53status <- as.integer(p53status == "p53+")
```

```r
# Apply p53 signature to Stockholm data, load stockholm U133A and U133B
# chips
load("your.file.path/stockholm_hgu133a.RData")
load("your.file.path/stockholm_hgu133b.RData")

stocka = stockholm_hgu133a
stocka = stocka[, order(colnames(stocka))]
stockb = stockholm_hgu133b
stockb = stockb[, order(colnames(stockb))]

identical(colnames(stocka), colnames(stockb))
# [1] TRUE

# Extract signature genes
stock32 <- rbind(stocka[p53_atA, ], stockb[p53_atB, ])
stock32 <- t(scale(t(stock32)))
identical(rownames(stock32), rownames(upp32))
# [1] TRUE

# Run DLDA, a part of the 'supclust' package
stockDLDA <- dlda(t(upp32), t(stock32), p53status)
p53_sig_sthlm = cbind(stockDLDA, colnames(stocka))
colnames(p53_sig_sthlm) = (c("DLDA", "patID"))
rownames(p53_sig_sthlm) = p53_sig_sthlm[, "patID"]
p53_sig_sthlm = p53_sig_sthlm[, -2]
p53_sig_sthlm = data.frame(p53_sig_sthlm)
p53_sthlm_calls = p53_sig_sthlm
save(p53_sthlm_calls, file = "p53_sthlm_calls.RData")
```

4. Recurrence Score

```r
# Make a recurrence score function
oneToFifteen <- function(x) {

    x <- x - min(x)
    x <- x * (15/max(x))
    x
}

GRB7groupscore <- function(arr) {

    arr["GRB7", ] <- arr["GRB7", ] * 0.9
    arr["HER2", ] <- arr["HER2", ] * 0.1

    gs <- apply(arr[c("GRB7", "HER2"), ], 2, sum)
    gs[gs < 8] <- 8
    gs
}

ERgroupscore <- function(arr) {

    arr["ER", ] <- arr["ER", ] * 0.2
    arr["PGR", ] <- arr["PGR", ] * 0.3
    arr["BCL2", ] <- arr["BCL2", ] * 0.25
    arr["SCUBE2", ] <- arr["SCUBE2", ] * 0.25

    gs <- apply(arr[c("ER", "PGR", "BCL2", "SCUBE2"), ], 2, sum)
    gs
}

proliferationgroupscore <- function(arr) {

    arr["Survivin", ] <- arr["Survivin", ] * 0.2
    arr["Ki67", ] <- arr["Ki67", ] * 0.2
    arr["MYBL2", ] <- arr["MYBL2", ] * 0.2
    arr["CCNB1", ] <- arr["CCNB1", ] * 0.2
    arr["STK15", ] <- arr["STK15", ] * 0.2

    gs <- apply(arr[c("Survivin", "Ki67", "MYBL2", "CCNB1", "STK15"), ], 2,
        sum)
    gs[gs < 6.5] <- 6.5
    gs
}

invasiongroupscore <- function(arr) {

    arr["CTSL2", ] <- arr["CTSL2", ] * 0.5
    arr["MMP11", ] <- arr["MMP11", ] * 0.5

    gs <- apply(arr[c("CTSL2", "MMP11"), ], 2, sum)
    gs
}

RS <- function(arr) {

    arr["GRB7gs", ] <- arr["GRB7gs", ] * 0.47
    arr["ERgs", ] <- arr["ERgs", ] * -0.34
    arr["proliferationgs", ] <- arr["proliferationgs", ] * 1.04
    arr["invasiongs", ] <- arr["invasiongs", ] * 0.1
    arr["CD68", ] <- arr["CD68", ] * 0.05
    arr["GSTM1", ] <- arr["GSTM1", ] * -0.08
    arr["BAG1", ] <- arr["BAG1", ] * -0.07

    RSu <- apply(arr, 2, sum)
    RS <- 20 * (RSu - 6.7)
    RS[RS < 0] <- 0
    RS[RS > 100] <- 100

    class <- rep("intermediate", length(RS))
    class[RS >= 31] <- "high"
    class[RS < 18] <- "low"
    rbind(RSu, RS, class)
}

paikfunction <- function(mat, group) {
    reference <- apply(mat[group, ], 2, mean)
    matr <- t(apply(mat, 1, function(x) x - reference))
    # print(matr[,1:5])

    matr <- apply(matr, 1, function(x) oneToFifteen(x))
    matr <- t(matr)
    # print(matr[,1:25])

    GRB7gs <- GRB7groupscore(matr)
    ERgs <- ERgroupscore(matr)
    proliferationgs <- proliferationgroupscore(matr)
    invasiongs <- invasiongroupscore(matr)

    mat2 <- rbind(GRB7gs, ERgs, proliferationgs, invasiongs, CD68 = matr["CD68",
        ], GSTM1 = matr["GSTM1", ], BAG1 = matr["BAG1", ])

    # print(mat2)[1:5,1:5]
    o <- RS(mat2)

    rbind(mat2[c("GRB7gs", "ERgs", "proliferationgs", "invasiongs"), ], o)
}

# Load Uppsala data

# Uppsala A
load("your.file.path/uppA.RData")
# Uppsala B
load("your.file.path/uppB.RData")

# same column order for A and B chips:
colnames(uppB) <- paste("X", colnames(uppB))
colnames(uppB) <- sub(" ", "", colnames(uppB))
uppB <- uppB[, colnames(uppA)]

# Are the column names identical in A and B?
identical(colnames(uppA), colnames(uppB))
# [1] TRUE

uppab = rbind(uppA, uppB)
dim(uppab)
# [1] 44928 253

# load in annotation object
load("your.file.path/paik110504.RData")

paikRMA <- uppab[paik$probes, ]
paikRMA <- apply(paikRMA, 2, function(x) tapply(x, paik$symbol, mean))
gr <- paik[match(rownames(paikRMA), paik$symbol), "group"] == "ref"

RS_upp_calls <- t(data.frame(paikfunction(paikRMA, gr)))
save(RS_upp_calls, file = "RS_upp_calls.RData")
```

5. Sorlie subtypes

```
# Make Sorlie annotation object
setwd("/your.file.path/")
sorlie <- read.delim("SorlieAnnotations.txt", sep = "\t", header = T)

load("your.file.path/Array Annotation/u133a.RData")
load("your.file.path/Array Annotation/u133b.RData")

rownames(u133a) <- u133a[, "Probe"]
rownames(u133b) <- u133b[, "Probe"]

createTable2 <- function(sigob, sigobCN, chipob, chipobCN, chip) {

    ct <- function(sigobrow, sigobCN, chipob, chipobCN) {

        m <- sigobrow[sigobCN]
        rws <- (chipob[, chipobCN] %in% m) & (chipob[, chipobCN] != "")
        psets <- chipob[rws, "Probe"]
        if (length(psets) == 0) {
            psets <- NA
        }

        o <- matrix(sigobrow, nrow = length(psets), ncol = length(sigobrow),
            byrow = T)
        o <- cbind(o, psets)
        colnames(o) <- c(names(sigobrow), "ProbeSet")
        if (!is.na(psets[1])) {
            o
        }
    }

    output <- c()

    for (i in 1:dim(sigob)[1]) {

        nl <- ct(sigob[i, ], sigobCN, chipob, chipobCN)
        output <- rbind(output, nl)
    }

    cbind(output, Match = rep(chip, dim(output)[1]))
}

sorliem <- as.matrix(sorlie)
sorliem[, "LLID"] <- gsub(" ", "", sorliem[, "LLID"])  # creating agreement with 'createTable2'

partAprobesLLID <- createTable2(sorliem, "LLID", u133a, "GeneID", "A")
partBprobesLLID <- createTable2(sorliem, "LLID", u133b, "GeneID", "B")
partAprobesGB <- createTable2(sorliem, "Acc", u133a, "GenBank", "A")
partBprobesGB <- createTable2(sorliem, "Acc", u133b, "GenBank", "B")
partAprobesUG <- createTable2(sorliem, "UGCluster", u133a, "UniGene", "A")
partBprobesUG <- createTable2(sorliem, "UGCluster", u133b, "UniGene", "B")

allparts <- rbind(partAprobesLLID, partBprobesLLID, partAprobesGB, partBprobesGB,
    partAprobesUG, partBprobesUG)
dv <- duplicated(allparts[, c("ORIGINALID", "ProbeSet", "Match")])
allparts <- allparts[!dv, ]
u133ab <- rbind(u133a, u133b)

MappableByLLID <- u133ab[allparts[, "ProbeSet"], "GeneID"] == allparts[, "LLID"]
MappableByGenBank <- u133ab[allparts[, "ProbeSet"], "GenBank"] == allparts[,
    "Acc"]
MappableByUniGene <- u133ab[allparts[, "ProbeSet"], "UniGene"] == allparts[,
    "UGCluster"]

sorlieAffy <- cbind(allparts, MappableByLLID, MappableByGenBank, MappableByUniGene)
sorlieAffy <- data.frame(sorlieAffy, stringsAsFactors = F)  # Get expected warning message
sorlieAffy <- sorlieAffy[order(sorlieAffy$ORIGINALID), ]

rownames(sorlieAffy) <- 1:dim(sorlieAffy)[1]
save(sorlieAffy, file = "Affy.RData")


# UppsalaA
load("your.file.path/uppA.RData")

# UppsalaB
load("your.file.path/uppB.RData")

# Annotation
annot = sorlieAffy
uppASorlie = rownames(uppA) %in% annot[(annot[, "Match"] == "A"), "ProbeSet"]
uppBSorlie = rownames(uppB) %in% annot[(annot[, "Match"] == "B"), "ProbeSet"]

a = uppA[uppASorlie, ]
b = uppB[uppBSorlie, ]

dim(a)
# [1] 667 253

dim(b)
# [1] 267 253

rmA = rowMeans(a)
rmB = rowMeans(b)
mcA = a - rmA
mcB = b - rmB

colnames(mcB) <- paste("X", colnames(mcB))
colnames(mcB) <- sub(" ", "", colnames(mcB))
mcB <- mcB[, colnames(mcA)]

# Are the column names identical in A and B?
identical(colnames(mcA), colnames(mcB))
# [1] TRUE

# everything matches, so row bind to combine the probes
probes = rbind(mcA, mcB)
con = merge(annot, probes, by.x = "ProbeSet", by.y = "row.names", all.x = T)
```

```
dim(con)
# [1] 934 269

centroids = aggregate(con, list(con$ORIGINALID), mean)  # Get expected warning
values = centroids[, -c(2:17)]
values[values == ""] <- NA
dim(values)
# [1] 421 254

rownames(values) <- values[, "Group.1"]
values = values[, -1]
values_df = data.frame(values)
rownames(values_df) <- sub(" ", "", rownames(values_df))
rownames(values_df) <- sub(" ", "", rownames(values_df))

tfannot = sorlie[, "ORIGINALID"] %in% rownames(values_df)
sorlie_matched = sorlie[tfannot, ]
dim(sorlie_matched)
# [1] 421 11

sorlie_matched = sorlie_matched[with(sorlie_matched, order(ORIGINALID)), ]
values_df = values_df[with(values_df, rownames(values_df)), ]

rownames(sorlie_matched) <- sorlie_matched[, "ORIGINALID"]
identical(rownames(sorlie_matched), rownames(values_df))
# [1] TRUE

sorlie_centroids = sorlie_matched[, c(7:11)]

pearson_correl = cor(sorlie_centroids, values_df, use = "everything", method = c("pearson"))
trans_pearson_correl = t(pearson_correl)
call = max.col(trans_pearson_correl)
call_names = colnames(trans_pearson_correl)[call]
sorlie_subtypes = cbind(rownames(trans_pearson_correl), call_names)
dim(sorlie_subtypes)
# [1] 253 2

no_subtype = apply(trans_pearson_correl, 1, function(x) {
    any(x > 0.1)
})
final_sorlie_calls = cbind(sorlie_subtypes, no_subtype)
final_sorlie_calls = final_sorlie_calls[, -1]
colnames(final_sorlie_calls) = c("pearson", "No_Subtype")
save(final_sorlie_calls, file = "final_sorlie_calls.RData")
```

6. Hu subtypes

```
# Here, we don't reproduce the code to create the Hu Affy annotation object
# as its identical to that used for Sorlie The file used to generate the
# affy object is 'HuAnnotation.RData'

# UppsalaA
load("your.file.path/uppA.RData")
```

```r
# UppsalaB
load("your.file.path/uppB.RData")

# Annotation
annot = huAffy
uppAHu = rownames(uppA) %in% annot[(annot[, "Match"] == "A"), "ProbeSet"]
uppBHu = rownames(uppB) %in% annot[(annot[, "Match"] == "B"), "ProbeSet"]

a = uppA[uppAHu, ]
b = uppB[uppBHu, ]

dim(a)
# [1] 543 253

dim(b)
# [1] 166 253

rmA = rowMeans(a)
rmB = rowMeans(b)
mcA = a - rmA
mcB = b - rmB

colnames(mcB) <- paste("X", colnames(mcB))
colnames(mcB) <- sub(" ", "", colnames(mcB))
mcB <- mcB[, colnames(mcA)]

# Are the column names identical in A and B?
identical(colnames(mcA), colnames(mcB))
# [1] TRUE

# everything matches, so row bind to combine the probes
probes = rbind(mcA, mcB)
con = merge(annot, probes, by.x = "ProbeSet", by.y = "row.names", all.x = T)
dim(con)
# [1] 709 268

centroids = aggregate(con, list(con$ORIGINALID), mean)  # Expected warning
values = centroids[, -c(2:16)]
values[values == ""] <- NA
dim(values)
# [1] 280 254

rownames(values) <- values[, "Group.1"]
values = values[, -1]
values_df = data.frame(values)

tfannot = hu[, "ORIGINALID"] %in% rownames(values_df)
hu_matched = hu[tfannot, ]
dim(hu_matched)
# [1] 280 10

hu_matched = hu_matched[with(hu_matched, order(ORIGINALID)), ]
values_df = values_df[with(values_df, rownames(values_df)), ]
```

```r
rownames(hu_matched) <- hu_matched[, "ORIGINALID"]
identical(rownames(hu_matched), rownames(values_df))
# [1] TRUE

hu_centroids = hu_matched[, c(6:10)]

spearman_correl = cor(hu_centroids, values_df, use = "everything", method = c("spearman"))
trans_spearman_correl = t(spearman_correl)
call = max.col(trans_spearman_correl)
call_names = colnames(trans_spearman_correl)[call]
hu_subtypes = cbind(rownames(trans_spearman_correl), call_names)
dim(hu_subtypes)
# [1] 253 2

rownames(hu_subtypes) = hu_subtypes[, 1]
final_hu_calls = data.frame(hu_subtypes[, -1])
colnames(final_hu_calls) = c("spearman")
save(final_hu_calls, file = "final_hu_calls.RData")
```

7. Parker subtypes

```r
# Here, we don't reproduce the code to create the PAM Affy annotation object
# as its identical to that used for Sorlie The file used to generate the
# affy object is 'PAMAnnotation.RData'

# UppsalaA
load("your.file.path/uppA.RData")

# UppsalaB
load("your.file.path/uppB.RData")

# Annotation
annot = PAMAffy
uppAPAM = rownames(uppA) %in% annot[(annot[, "Match"] == "A"), "ProbeSet"]
uppBPAM = rownames(uppB) %in% annot[(annot[, "Match"] == "B"), "ProbeSet"]

a = uppA[uppAPAM, ]
b = uppB[uppBPAM, ]

dim(a)
# [1] 90 253

dim(b)
# [1] 24 253

rmA = rowMeans(a)
rmB = rowMeans(b)
mcA = a - rmA
mcB = b - rmB

colnames(mcB) <- paste("X", colnames(mcB))
colnames(mcB) <- sub(" ", "", colnames(mcB))
```

```
mcB <- mcB[, colnames(mcA)]

# Are the column names identical in A and B?
identical(colnames(mcA), colnames(mcB))
# [1] TRUE

# everything matches, so row bind to combine the probes
probes = rbind(mcA, mcB)
con = merge(annot, probes, by.x = "ProbeSet", by.y = "row.names", all.x = T)
dim(con)
# [1] 114 268

centroids = aggregate(con, list(con$ORIGINALID), mean)  # Get expected warning
values = centroids[, -c(2:16)]
values[values == ""] <- NA
dim(values)
# [1] 50 254

rownames(values) <- values[, "Group.1"]
values = values[, -1]
values_df = data.frame(values)

tfannot = PAM[, "ORIGINALID"] %in% rownames(values_df)
PAM_matched = PAM[tfannot, ]
dim(PAM_matched)
# [1] 50 10

PAM_matched = PAM_matched[with(PAM_matched, order(ORIGINALID)), ]
values_df = values_df[with(values_df, rownames(values_df)), ]

rownames(PAM_matched) <- PAM_matched[, "ORIGINALID"]
identical(rownames(PAM_matched), rownames(values_df))
# [1] TRUE

PAM_centroids = PAM_matched[, c(6:10)]

spearman_correl = cor(PAM_centroids, values_df, use = "everything", method = c("spearman"))
trans_spearman_correl = t(spearman_correl)
call = max.col(trans_spearman_correl)
call_names = colnames(trans_spearman_correl)[call]
PAM_subtypes = cbind(rownames(trans_spearman_correl), call_names)
dim(PAM_subtypes)
# [1] 253 2

rownames(PAM_subtypes) = PAM_subtypes[, 1]
final_PAM_calls = data.frame(PAM_subtypes[, -1])
colnames(final_PAM_calls) = c("spearman")
save(final_PAM_calls, file = "final_PAM_calls.RData")
```