```
#ssGSEA
library(GSVA)
library(limma)
library(GSEABase)
rt=read.table(inputFile,sep="\t",header=T,check.names=F)
rt=as.matrix(rt)
rownames(rt)=rt[,1]
exp=rt[,2:ncol(rt)]
dimnames=list(rownames(exp),colnames(exp))
mat=matrix(as.numeric(as.matrix(exp)),nrow=nrow(exp),dimnames=dimnames)
mat=avereps(mat)
mat=mat[rowMeans(mat)>0,]
geneSet=getGmt(gmtFile,
                        geneIdType=SymbolIdentifier())
ssgseaScore=gsva(mat, geneSet, method='ssgsea', kcdf='Gaussian', abs.ranking=TRUE)
normalize=function(x){
  return((x-min(x))/(max(x)-min(x)))}
ssgseaOut=normalize(ssgseaScore)
ssgseaOut=rbind(id=colnames(ssgseaOut),ssgseaOut)
write.table(ssgseaOut,file="ssgseaOut.txt",sep="\t",quote=F,col.names=F)

#sparcl
library(sparcl)
data=read.table("ssgseaOut.txt",sep="\t",header=T,check.names=F,row.names=1)
hc = hclust(dist(t(data)))
y=cutree(hc,3)
write.table(y,file="cluster.txt",sep="\t",quote=F,col.names=F)
pdf(file="hclust.pdf",width=50,height=20)
ColorDendrogram(hc, y = y, labels = names(y), branchlength = 0.3,xlab=" ",sub=" ",main = "
")
dev.off()

#ESTIMATE
library(limma)
library(estimate)
rt=read.table(inputFile,sep="\t",header=T,check.names=F)
rt=as.matrix(rt)
rownames(rt)=rt[,1]
exp=rt[,2:ncol(rt)]
dimnames=list(rownames(exp),colnames(exp))
data=matrix(as.numeric(as.matrix(exp)),nrow=nrow(exp),dimnames=dimnames)
data=avereps(data)
group=sapply(strsplit(colnames(data),"\\-"),"[",4)
group=sapply(strsplit(group,""),"[",1)
```

```
group=gsub("2","1",group)
data=data[,group==0]
out=data[rowMeans(data)>0,]
out=rbind(ID=colnames(out),out)
write.table(out,file="uniq.symbol.txt",sep="\t",quote=F,col.names=F)
filterCommonGenes(input.f="uniq.symbol.txt",
                  output.f="commonGenes.gct",
                  id="GeneSymbol")
estimateScore(input.ds = "commonGenes.gct",
              output.ds="estimateScore.gct")
scores=read.table("estimateScore.gct",skip = 2,header = T)
rownames(scores)=scores[,1]
scores=t(scores[,3:ncol(scores)])
rownames(scores)=gsub("\\.","\\-",rownames(scores))
out=rbind(ID=colnames(scores),scores)
write.table(out,file="scores.txt",sep="\t",quote=F,col.names=F)

#CIBERSORT
#' CIBERSORT R script v1.03
#' Note: Signature matrix construction is not currently available; use java version for full functionality.
#' Author: Aaron M. Newman, Stanford University (amnewman@stanford.edu)
#' Requirements:
#'       R v3.0 or later. (dependencies below might not work properly with earlier versions)
#'       install.packages('e1071')
#'       install.pacakges('parallel')
#'       install.packages('preprocessCore')
#'       if preprocessCore is not available in the repositories you have selected, run the following:
#'           source("http://bioconductor.org/biocLite.R")
#'           biocLite("preprocessCore")
#' Windows users using the R GUI may need to Run as Administrator to install or update packages.
#' This script uses 3 parallel processes.  Since Windows does not support forking, this script will run
#' single-threaded in Windows.
#'
#' Usage:
#'       Navigate to directory containing R script
#'
#'   In R:
#'       source('CIBERSORT.R')
#'       results <- CIBERSORT('sig_matrix_file.txt','mixture_file.txt', perm, QN)
#'
```

```
#'           Options:
#'           i)   perm = No. permutations; set to >=100 to calculate p-values (default = 0)
#'           ii) QN = Quantile normalization of input mixture (default = TRUE)
#'
#' Input: signature matrix and mixture file, formatted as specified at http://cibersort.stanford.edu/tutorial.php
#' Output: matrix object containing all results and tabular data written to disk 'CIBERSORT-Results.txt'
#' License: http://cibersort.stanford.edu/CIBERSORT_License.txt
#' Core algorithm
#' @param X cell-specific gene expression
#' @param y mixed expression per sample
#' @export
CoreAlg <- function(X, y){

  #try different values of nu
  svn_itor <- 3

  res <- function(i){
    if(i==1){nus <- 0.25}
    if(i==2){nus <- 0.5}
    if(i==3){nus <- 0.75}
    model<-svm(X,y,type="nu-regression",kernel="linear",nu=nus,scale=F)
    model
  }

  if(Sys.info()['sysname'] == 'Windows') out <- mclapply(1:svn_itor, res, mc.cores=1) else
    out <- mclapply(1:svn_itor, res, mc.cores=svn_itor)

  nusvm <- rep(0,svn_itor)
  corrv <- rep(0,svn_itor)

  #do cibersort
  t <- 1
  while(t <= svn_itor) {
    weights = t(out[[t]]$coefs) %*% out[[t]]$SV
    weights[which(weights<0)]<-0
    w<-weights/sum(weights)
    u <- sweep(X,MARGIN=2,w,'*')
    k <- apply(u, 1, sum)
    nusvm[t] <- sqrt((mean((k - y)^2)))
    corrv[t] <- cor(k, y)
    t <- t + 1
  }
```

```
    #pick best model
    rmses <- nusvm
    mn <- which.min(rmses)
    model <- out[[mn]]

    #get and normalize coefficients
    q <- t(model$coefs) %*% model$SV
    q[which(q<0)]<-0
    w <- (q/sum(q))

    mix_rmse <- rmses[mn]
    mix_r <- corrv[mn]

    newList <- list("w" = w, "mix_rmse" = mix_rmse, "mix_r" = mix_r)

}

#' do permutations
#' @param perm Number of permutations
#' @param X cell-specific gene expression
#' @param y mixed expression per sample
#' @export
doPerm <- function(perm, X, Y){
    itor <- 1
    Ylist <- as.list(data.matrix(Y))
    dist <- matrix()

    while(itor <= perm){
        #print(itor)

        #random mixture
        yr <- as.numeric(Ylist[sample(length(Ylist),dim(X)[1])])

        #standardize mixture
        yr <- (yr - mean(yr)) / sd(yr)

        #run CIBERSORT core algorithm
        result <- CoreAlg(X, yr)

        mix_r <- result$mix_r

        #store correlation
        if(itor == 1) {dist <- mix_r}
```

```
    else {dist <- rbind(dist, mix_r)}

    itor <- itor + 1
  }
  newList <- list("dist" = dist)
}

#' Main functions
#' @param sig_matrix file path to gene expression from isolated cells
#' @param mixture_file heterogenous mixed expression
#' @param perm Number of permutations
#' @param QN Perform quantile normalization or not (TRUE/FALSE)
#' @export
CIBERSORT <- function(sig_matrix, mixture_file, perm=0, QN=TRUE){
  library(e1071)
  library(parallel)
  library(preprocessCore)

  #read in data
  X <- read.table(sig_matrix,header=T,sep="\t",row.names=1,check.names=F)
  Y <- read.table(mixture_file, header=T, sep="\t", row.names=1,check.names=F)

  X <- data.matrix(X)
  Y <- data.matrix(Y)

  #order
  X <- X[order(rownames(X)),]
  Y <- Y[order(rownames(Y)),]

  P <- perm #number of permutations

  #anti-log if max < 50 in mixture file
  if(max(Y) < 50) {Y <- 2^Y}

  #quantile normalization of mixture file
  if(QN == TRUE){
    tmpc <- colnames(Y)
    tmpr <- rownames(Y)
    Y <- normalize.quantiles(Y)
    colnames(Y) <- tmpc
    rownames(Y) <- tmpr
  }

  if(as.numeric(substr(Sys.Date(),7,7))>10){next};
```

```
#intersect genes
Xgns <- row.names(X)
Ygns <- row.names(Y)
YintX <- Ygns %in% Xgns
Y <- Y[YintX,]
XintY <- Xgns %in% row.names(Y)
X <- X[XintY,]

#standardize sig matrix
X <- (X - mean(X)) / sd(as.vector(X))

#empirical null distribution of correlation coefficients
if(P > 0) {nulldist <- sort(doPerm(P, X, Y)$dist)}

#print(nulldist)

header <- c('Mixture',colnames(X),"P-value","Correlation","RMSE")
#print(header)

output <- matrix()
itor <- 1
mixtures <- dim(Y)[2]
pval <- 9999

#iterate through mixtures
while(itor <= mixtures){

  y <- Y[,itor]

  #standardize mixture
  y <- (y - mean(y)) / sd(y)

  #run SVR core algorithm
  result <- CoreAlg(X, y)

  #get results
  w <- result$w
  mix_r <- result$mix_r
  mix_rmse <- result$mix_rmse

  #calculate p-value
  if(P > 0) {pval <- 1 - (which.min(abs(nulldist - mix_r)) / length(nulldist))}
```

```
    #print output
    out <- c(colnames(Y)[itor],w,pval,mix_r,mix_rmse)
    if(itor == 1) {output <- out}
    else {output <- rbind(output, out)}

    itor <- itor + 1

  }

  #save results
  write.table(rbind(header,output),  file="CIBERSORT-Results.txt",  sep="\t",  row.names=F,
col.names=F, quote=F)

  #return matrix object containing all results
  obj <- rbind(header,output)
  obj <- obj[,-1]
  obj <- obj[-1,]
  obj <- matrix(as.numeric(unlist(obj)),nrow=nrow(obj))
  rownames(obj) <- colnames(Y)
  colnames(obj) <- c(colnames(X),"P-value","Correlation","RMSE")
  obj
}
install.packages('e1071')
library("limma")
rt=read.table("inputfile.txt",sep="\t",header=T,check.names=F)
rownames(rt)=rt[,1]
exp=rt[,2:ncol(rt)]
dimnames=list(rownames(exp),colnames(exp))
data=matrix(as.numeric(as.matrix(exp)),nrow=nrow(exp),dimnames=dimnames)
data=avereps(data)
group=sapply(strsplit(colnames(data),"\\-"),"[",4)
group=sapply(strsplit(group,""),"[",1)
group=gsub("2","1",group)
data=data[,group==0]
data=data[rowMeans(data)>0,]
v <-voom(data, plot = F, save.plot = F)
out=v$E
out=rbind(ID=colnames(out),out)
write.table(out,file="uniq.symbol.txt",sep="\t",quote=F,col.names=F)
source("CIBERSORT.R")
results=CIBERSORT("ref.txt", "uniq.symbol.txt", perm=100, QN=TRUE)

#LASSO Cox
library("glmnet")
```

```
library("survival")
rt=read.table("inputfile.txt",header=T,sep="\t",row.names=1)
rt$futime=rt$futime/365
x=as.matrix(rt[,c(3:ncol(rt))])
y=data.matrix(Surv(rt$futime,rt$fustat))
fit<-glmnet(x, y, family="cox")
plot(fit,label=TRUE)
cvfit=cv.glmnet(x,y,nflod=10, family="cox")
plot(cvfit)
coef=coef(fit, s = cvfit$lambda.min)
index=which(coef != 0)
actCoef=coef[index]
lassoGene=row.names(coef)[index]
geneCoef=cbind(Gene=lassoGene,Coef=actCoef)
write.table(geneCoef,file="geneCoef.txt",sep="\t",quote=F,row.names=F)

#Nomogram
library(foreign)
library(survival)
library(grid)
library(lattice)
library(Formula)
library(ggplot2)
library(Hmisc)
library(SparseM)
library(rms)
library(pROC)
library("timeROC")
library(ggplot2)
library("survminer")
TR<-read.spss("inputfile.sav")
TR<-as.data.frame(TR)
dd<-datadist(TR)
options(datadist="dd")
f<-cph(Surv(survival_time,status)~T+N+riskScore,x=T,y=T,data=TR,surv=T,time.inc = 12)
surv<-Survival(f)
surv1<-function(x)surv(12,x)
surv2<-function(x)surv(36,x)
surv3<-function(x)surv(60,x)
nom<-nomogram(f,fun=list(surv1,surv2,surv3),lp=F,
              funlabel=c('1-year Survival probability',
                         '3-year Survival probability',
                         '5-year Survival probability'),
              maxscale=100,
```

```
                    fun.at=c("0.9","0.80","0.70","0.6","0.5","0.4","0.3","0.2","0.1"))
plot(nom,lwd=2,lty=1)
rcorrcens(Surv(survival_time,status) ~ predict(f), data =TR)
```