ClusterID-based consensus clustering

Maxime Tarabichi

November 28, 2016

Description

Consensus cluster assignment

Let us use m methods $M_{1,\ldots,m}$ to cluster v single nucleotide variants $V_{1,\ldots,v}$.
Each method M_i assigns to each variant V_j one cluster ID

$$A(M_i, V_j)$$

For each variant V_j, we write the vector of assignment

$$\vec{VA}(V_j) = [A(M_1, V_j), A(M_2, V_j), \ldots, A(M_m, V_j)]$$

Two mutations V_a and V_b sharing vector of assignments, i.e $\vec{VA}(V_a) = \vec{VA}(V_b)$, have been assigned to the same cluster within all methods.
Across all mutations, there will be u unique vectors of assignment $U\vec{VA}_{1\ldots u}$, shared by $N_{1\ldots u}$ number of mutations, respectively.
For each unique vector of assignment $U\vec{VA}_k$, we compute a distance d_{kl} to all $U\vec{VA}_l$. That distance is defined as a the sum of two distances

$$d_{kl} = d1(k,l) + d2(k,l)$$

with:

$$d1(k,l) = Cmax(N_{1\ldots u})sum(U\vec{VA}_k != U\vec{VA}_l)$$
$$d2(k,l) = -max(N_k, N_l)$$

C is set to 10 and ensures that d1=method assignment prevails over d2=number of mutations in a cluster.
Next we perform hierarchical clustering on the distance matrix $D_{uxu} = d_{kl}$ using Ward's criterion and cut the tree to get a desired number of clusters of $U\vec{VA}$. This method thus takes the final number of consensus clusters as input.
Finally, given $U\vec{VA}s_c$ in each consensus cluster c, mutations V_i for which $\vec{VA}(V_i) \in U\vec{VA}s_c$ are assigned to c.

Optimal K

To obtain the optimal number of clusters K, we derive consensus matrices for $K \in [2, max(K_{methods})]$. If a majority of methods have K=1, we set K=1. Otherwise for each values of K, we repeat the consensus clustering procedure 100 times, after independent sampling with replacement from the methods. We then obtain a consensus matrix for each K value, corresponding to the proportion of times the mutations were clustered together. We take optimal K as the K yielding the matrix with minimal PAC [1].

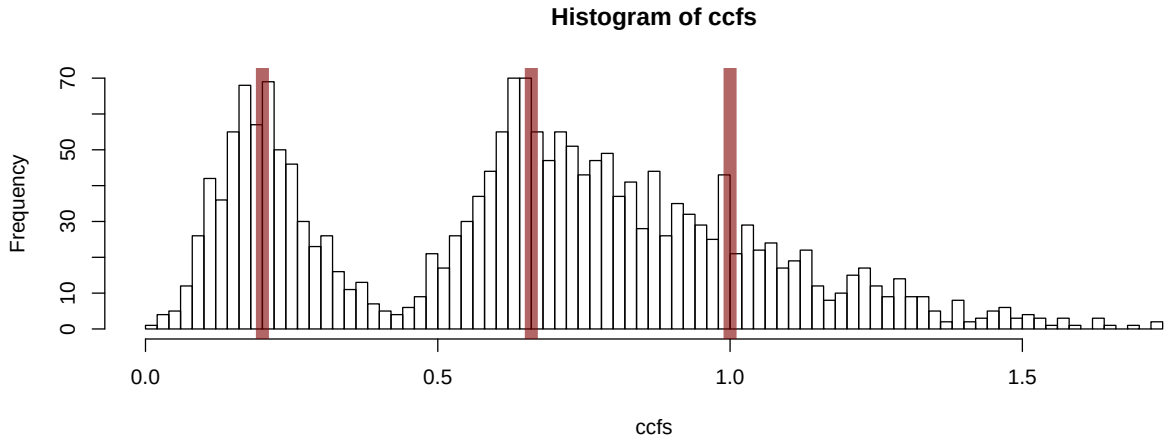
Consensus CCF

The CCFs of each consensus cluster can then be calculated by summarising the CCFs of individual mutations assigned to that cluster. We take the weighted average of these CCFs with the weights of each mutation proportional to its number of aligned reads.
This last step requires to compute a consensus CCF for each mutation. We rescale CCF of each method using the consensus purity ($CCF_{rescaled} = \frac{purity_{method}}{purity_{consensus}} CCF_{method}$) and take the median across methods.

Example

Let's generate simple dummy data for illustration purposes.

```
## ####################################
## generate data
## list with cluster assignments of each method
allA <- list(method1=c(rep(1,1000),rep(2,1000)),
             method2=c(rep(1,800),rep(2,900), rep(3,300)),
             method3=c(rep(1,800),rep(2,600), rep(3,600)),
             method4=c(rep(1,1200),rep(2,600), rep(3,200)),
             method5=c(rep(4,200),rep(3,500), rep(2,700),rep(1,600)),
             method6=c(rep(4,600),rep(3,800), rep(2,400),rep(1,200))
             )
allA <- lapply(allA,function(x) {names(x) <- paste("mutation",1:2000,sep="");return(x);})
sapply(allA,head)

##           method1 method2 method3 method4 method5 method6
## mutation1       1       1       1       1       4       4
## mutation2       1       1       1       1       4       4
## mutation3       1       1       1       1       4       4
## mutation4       1       1       1       1       4       4
## mutation5       1       1       1       1       4       4
## mutation6       1       1       1       1       4       4

## simulate a dummy underlying ccf
cov <- rpois(700,70)
cov2 <- rpois(600,70)
ccfs <- c(sort(rbinom(700, cov, .5)*2/sample(cov),decreasing=T),
          sort(rbinom(700, cov, .33)*2/cov,decreasing=T),
          sort(rbinom(600, cov, .1)*2/cov2,decreasing=T))
## ####################################
## plot ccfs
hist(ccfs,100)
abline(v=c(1,.66,.2),lwd=10,col=rgb(.5,0,0,.6))
## ####################################
```



This is a dummy -not necessarily realistic- example to illustrate how the methods would treat the typical input, i.e. a list of vectors of cluster assignments (=cluster IDs). We start with the cluster assignments from 6 methods for the same 2,000 mutations. In this example, the truth has three clusters of 700, 700, and 600 mutations, resp. For illustration purposes, their cancer cell fractions (CCFs) are modelled as binomials around 100%, 66% and 20% CCF, respectively. However, only the cluster assignments are actually used for consensus clustering.

In this example the methods find 2, 3, 3, 3, 4 and 4 clusters resp., and each differ slightly in cluster assignments. Unlike the four first methods, the two last methods have assigned highest cluster ID to the lowest CCF cluster. Because the cluster-ID consensus clustering is not based on the CCF, this will have no impact on the final consensus clustering.

The two first mutations are assigned to cluster1, cluster1, cluster1, cluster1, cluster4 and cluster4, resp. by each of the methods and therefore their vector of assignments is defined as: 1-1-1-1-4-4. The last mutation is assigned to cluster2, cluster3, cluster3, cluster3, cluster1, and cluster1, resp. by each of the methods and its vector of assignments is defined as: 2-3-3-3-1-1.

The consensus clustering distance between the two first mutations is minimal (d=0) since they have been assigned to the same cluster by all methods. The consensus clustering distance between these the two first mutations and the last mutation is maximal (d=6), since all methods have assigned these two mutations to different clusters.

This distance between mutations is used to cluster all mutations into an optimal number K of consensus clusters. K is chosen to minimise the Proportion of Ambiguous Clusters (PAC) [1] of the consensus matrices corresponding to K.

Let's first cluster mutations based on their distances for a given number of clusters K=4.

```
## similarity between two vectors of assignments
votedist <- function(g1,g2)
{
    suppressWarnings(if(T){
    g1 <- as.numeric(strsplit(g1,split="-")[[1]])
    g2 <- as.numeric(strsplit(g2,split="-")[[1]])
    return(sum(g1==g2,na.rm=T))})
}

## distance matrix between pairs of unique vectors of assignments
votedist.matrix <- function(allgs,nMethods)
{
    m <- matrix(0,length(allgs),length(allgs))
    for(i in 1:(length(allgs)-1))
        for(j in (i+1):length(allgs))
        {
            m[i,j] <- votedist(allgs[i],allgs[j])
            m[j,i] <- votedist(allgs[i],allgs[j])
        }
    rownames(m) <- allgs
    colnames(m) <- allgs
    nMethods-m
}

## fast cc using hclust, with method="ward linkage"
fastConsensusClustering <- function(allA,
                                    nbClusters,
                                    keepnames=NULL)
{
    nMut <- length(allA[[1]])
    nMethods <- length(allA)
    matA <- t(sapply(allA,function(x) x))
    clsA <- NULL
    for(i in 1:nrow(matA))
        clsA <- paste(clsA,matA[i,],sep=if(i==1) "" else "-")
    head(clsA)
    vuA <- sort(table(clsA),decreasing=T)
    head(vuA)
    distF <- as.dist(votedist.matrix(names(vuA),nMethods)*max(vuA)*10)
    hc <- hclust(distF,method="ward.D")
    lClusts <- lapply(nbClusters,function(nC) cutree(hc,k=nC))
    return(lClusts)
}
## run and print results
lResultClusters <- fastConsensusClustering(allA,nbClusters=4)
print(lResultClusters)

## [[1]]
## 1-1-1-1-3-4 2-2-3-2-1-2 1-1-1-1-4-4 1-2-2-1-2-3 2-2-2-1-2-3 2-2-2-2-2-3 2-3-3-3-1-1
##           1           2           1           3           3           3           4
## 1-1-1-1-2-3 1-1-1-1-3-3 2-3-3-2-1-2
##           1           1           2
```

We can see after printing the results that each mutation has been grouped together with all other mutations with the same vector of assignments. We are actually clustering unique vectors of assignments rather

than individual mutations. This will prove very useful when clustering high number of mutations, as the time complexity of the method does not depend on the number of mutations but the number of vectors of assignments.

The clusters assigned are not necessarily ordered with regards to the underlying CCFs (see example below).

```
## compute consensus ccfs
clsA <- NULL
for(i in 1:length(allA))
    clsA <- paste(clsA,allA[[i]],sep=if(i==1) "" else "-")
ccClusters <- unique(lResultClusters[[1]])
ccClusters

## [1] 1 2 3 4

meanCCFs <- sapply(ccClusters,
                   function(x)
                       mean(ccfs[lResultClusters[[1]][clsA]==x]))
## print ccfs
print(meanCCFs)

## [1] 0.9905733 0.2453534 0.6330913 0.1211358
```

Let's try with K=3 clusters.

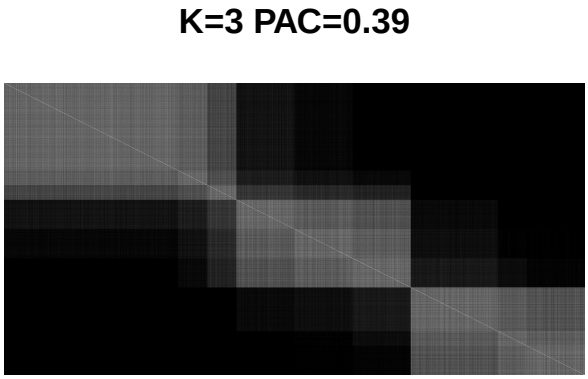
```
lResultClusters <- fastConsensusClustering(allA,nbClusters=3)
## final cc with K=2
print(lResultClusters)

## [[1]]
## 1-1-1-1-3-4 2-2-3-2-1-2 1-1-1-1-4-4 1-2-2-1-2-3 2-2-2-1-2-3 2-2-2-2-2-3 2-3-3-3-1-1
##           1           2           1           3           3           3           2
## 1-1-1-1-2-3 1-1-1-1-3-3 2-3-3-2-1-2
##           1           1           2

clsA <- NULL
for(i in 1:length(allA))
    clsA <- paste(clsA,allA[[i]],sep=if(i==1) "" else "-")
ccClusters <- unique(lResultClusters[[1]])
ccClusters

## [1] 1 2 3

meanCCFs <- sapply(ccClusters,
                   function(x)
                       mean(ccfs[lResultClusters[[1]][clsA]==x]))
## final ccfs with K=2
print(meanCCFs)

## [1] 0.9905733 0.2039475 0.6330913
```

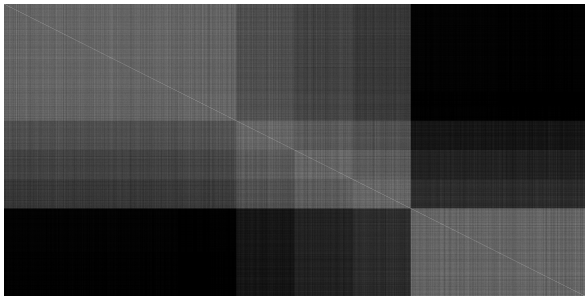
To obtain the optimal number of clusters $optK$, we derive *consensusmatrices* for $optK \in [2, max(K_{methods})]$. If a majority of methods have $K = 1$, we set $optK = 1$. Otherwise for each value of $optK$, we repeat the consensus clustering procedure 100 times, after independent sampling with replacement from the methods. We then obtain one *consensusmatrix* for each K value, corresponding to proportion of times the mutations were co-clustered. We take $optK = \arg\min_K PAC(consensusmatrix(K))$ [1].

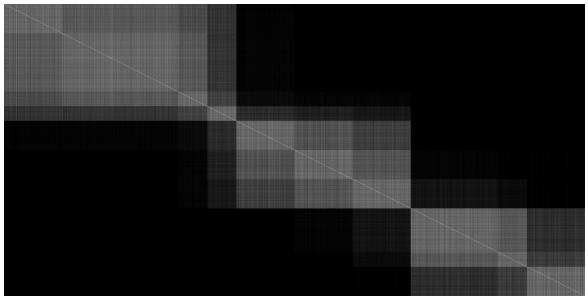
```
dyn.load("scoringlite.so")
makeHardAss <- function(v)
{
    matrix(.C("hardass",
              as.integer(length(v)),
              as.integer(v),
              as.integer(rep(0,length(v)*length(v))))[[3]],
           length(v),length(v))
}

CCM <- function(ccm, allCC,ii,jj,repeats,i,rn)
{
```

```
    for(j in 1:repeats)
    {
        v <- allCC[[ii]][[jj]][[j]][[i]][rn]
        v[is.na(v)] <- (max(v,na.rm=T)+1):(max(v,na.rm=T)+sum(is.na(v)))
        ccm <- ccm+makeHardAss(v)
        gc()
    }
    ccm
}

chooseOptimalK <- function(ccm,maxK)
{
    ## from Dr. Yasin Åženbabaoğlu:
    ## shenbaba.weebly.com/blog/how-to-use-the-pac-measure-in-consensus-clustering
    Kvec = 2:maxK
    x1 = 0.1; x2 = 0.9 ## threshold defining the intermediate sub-interval
    PAC = rep(NA,length(Kvec))
    names(PAC) = paste("K=",Kvec,sep="") # from 2 to maxK
    for(i in Kvec){
        M = ccm[[i]]
        Fn = ecdf(M[lower.tri(M)])
        PAC[i-1] = Fn(x2) - Fn(x1)
    }
    ## The optimal K
    optK = Kvec[which.min(PAC)]
    list(optK=optK,PAC=PAC)
}

votedist.matrix.nbmut <- function(allgs,tA)
{
    m <- matrix(NA,length(allgs),length(allgs))
    for(i in 1:(length(allgs)-1))
        for(j in (i+1):length(allgs))
            m[j,i] <- m[i,j] <- -max(tA[allgs[i]],tA[allgs[j]])
    rownames(m) <- allgs
    colnames(m) <- allgs
    m
}

fastConsensusClustering <- function(allA,
                                    nbClusters,
                                    keepnames)
{
    nMut <- length(allA[[1]])
    nMethods <- length(allA)
    matA <- t(sapply(allA,function(x) x))
    clsA <- NULL
    for(i in 1:nrow(matA))
        clsA <- paste(clsA,matA[i,],sep=if(i==1) "" else "-")
    vuA <- sort(table(clsA),decreasing=T)
    dist1 <- votedist.matrix(names(vuA),nMethods)*max(vuA)*10
    dist2 <- votedist.matrix.nbmut(names(vuA),vuA)
    distF <- as.dist(dist1+dist2)
    hc <- hclust(distF,method="ward.D")
    lClusts <- lapply(nbClusters,function(nC) cutree(hc,k=nC))
    return(lapply(lClusts,function(x){
        clusts <- x[clsA]
        names(clusts) <- keepnames
        clusts
    }))
}

consensusMatrix <- function(allA,
                            pMethods=c(0.5,0.75,0.8,1,1.1,1.2),
                            pFeatures=c(0.5,0.75,0.8,1,1.1,1.2),
                            repeats=2)
{
    nbClustersMethods <-sapply(allA,function(x) length(unique(x[!is.na(x)])))
    if(median(nbClustersMethods)==1) return(1)
    nbClusters <- max(nbClustersMethods)
    nbMethods <- length(allA)
    nbFeatures <- length(allA[[1]])
    timeCC <- system.time(allCC <- lapply(pMethods,function(x)
        lapply(pFeatures,function(y)
            lapply(1:repeats,function(smp)
            {
                keepMethod <- sample(1:nbMethods,round(nbMethods*x),rep=T)
                keepFeatures <- sample(1:nbFeatures,round(nbFeatures*x),rep=T)
                lAa <- lapply(keepMethod,function(a)
                {
```

```r
                    allA[[a]][keepFeatures]
                })
                fastConsensusClustering(lAa,
                                        1:nbClusters,
                                        keepnames=paste("snv",
                                                        keepFeatures,
                                                        sep=":"))
            }))))
    clsA <- paste("snv",1:length(allA[[1]]),sep=":")
    l <- length(clsA)
    ccm <- list()
    for(i in 2:nbClusters)
    {
        ccm[[i]] <- matrix(0,l,l)
        rownames(ccm[[i]]) <- colnames(ccm[[i]]) <- clsA
        for(ii in 1:length(pMethods))
        {
            for(jj in 1:length(pFeatures))
            {
                ccm[[i]] <- CCM(ccm[[i]],allCC,ii,jj,repeats,i,clsA)
            }
        }
    }
    K <- chooseOptimalK(lapply(ccm,function(x)
    {
        x/(length(pMethods)*length(pFeatures)*repeats)
    }),
    nbClusters)
    list(K=K,ccm=ccm,timeCC=timeCC)
}

ccm <- consensusMatrix(allA,pMethods=1,pFeatures=1,repeats=100)
str(ccm)

## List of 3
##  $ K     :List of 2
##   ..$ optK: int 4
##   ..$ PAC : Named num [1:3] 0.701 0.389 0.362
##   .. ..- attr(*, "names")= chr [1:3] "K=2" "K=3" "K=4"
##  $ ccm   :List of 4
##   ..$ : NULL
##   ..$ : num [1:2000, 1:2000] 100 44 38 35 46 44 38 34 41 40 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2000] "snv:1" "snv:2" "snv:3" "snv:4" ...
##   .. .. ..$ : chr [1:2000] "snv:1" "snv:2" "snv:3" "snv:4" ...
##   ..$ : num [1:2000, 1:2000] 100 44 38 35 46 44 38 34 41 40 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2000] "snv:1" "snv:2" "snv:3" "snv:4" ...
##   .. .. ..$ : chr [1:2000] "snv:1" "snv:2" "snv:3" "snv:4" ...
##   ..$ : num [1:2000, 1:2000] 100 44 38 35 46 44 38 34 41 40 ...
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2000] "snv:1" "snv:2" "snv:3" "snv:4" ...
##   .. .. ..$ : chr [1:2000] "snv:1" "snv:2" "snv:3" "snv:4" ...
##  $ timeCC:Class 'proc_time'  Named num [1:5] 1.577 0.027 1.606 0 0
##   .. ..- attr(*, "names")= chr [1:5] "user.self" "sys.self" "elapsed" "user.child" ...

plotCCM <- function(ccm,...)
{
    par(mar=c(0,0,3,0))
    plot(0,0,xaxt="n",yaxt="n",frame=F,xlab="",ylab="",col=rgb(0,0,0,0),
         xlim=c(0,1),ylim=c(0,1),...)
    rasterImage(ccm/100,0,0,1,1)
}

par(mfcol=c(2,2))
par(mar=c(0,0,3,0))
plot.new()
tmp_null <- lapply(2:4,function(x) plotCCM(ccm$ccm[[x]],
                                           main=if(x==ccm$K$optK)
                                                    paste0("optimal K=",x,
                                                           " PAC=",signif(ccm$K$PAC[x-1],2))
                                                else paste0("K=",x,
                                                            " PAC=",signif(ccm$K$PAC[x-1],2))))
```







For this dummy example we find that the PAC is minimal for K=4, which reflects in the consensus matrix (see heatmaps).

Once the optimal K is defined, we can either run the fastConsensusClustering with K=optimalK or we can use the consensus matrix to cluster mutations together. We go for the latter.

```
findClusts <- function(ccm,nbClust)
{
    hc <- hclust(as.dist(100-ccm),method="ward.D")
    clusts <- cutree(hc,k=nbClust)
}
finalClusts <- findClusts(ccm$ccm[[ccm$K$optK]],ccm$K$optK)
table(finalClusts)

## finalClusts
##   1   2   3   4
## 800 600 400 200
```

Conclusion

This method takes the cluster assignments (=cluster IDs) and CCF of all mutations for each method and returns the consensus cluster assignments, the corresponding optimal number of clusters and their consensus CCFs.

```
sessionInfo()

## R version 3.1.3 (2015-03-09)
## Platform: x86_64-apple-darwin10.8.0 (64-bit)
## Running under: OS X 10.10.5 (Yosemite)
##
## locale:
## [1] C
##
## attached base packages:
```

```
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] knitr_1.10.5
##
## loaded via a namespace (and not attached):
## [1] compiler_3.1.3 evaluate_0.7   formatR_1.2    highr_0.5      magrittr_1.5
## [6] stringi_0.4-1  stringr_1.0.0  tools_3.1.3
```

References

[1] Yasin Shenbabaoğlu, George Michailidis, and Jun Z. Li. Critical limitations of consensus clustering in class discovery. *Scientific Reports*, 4, August 2014.