pipeline weighted K-medoid

Zhixiang Lin

3/7/2017

```
library(readr)
library(WeightedCluster)
```

```
## Loading required package: TraMineR
##
## TraMineR stable version 1.8-13 (Built: 2017-03-02)
## Website: http://traminer.unige.ch
## Please type 'citation("TraMineR")' for citation information.
## Loading required package: cluster
## This is WeightedCluster stable version 1.2 (Built: 2017-03-02)
##
## To get the manuals, please run:
##    vignette("WeightedCluster") ## Complete manual in English
##    vignette("WeightedClusterFR") ## Complete manual in French
##    vignette("WeightedClusterPreview") ## Short preview in English
##
## To cite WeightedCluster in publications please use:
## Studer, Matthias (2013). WeightedCluster Library Manual: A practical guide to
##    creating typologies of trajectories in the social sciences with R.
##    LIVES Working Papers, 24. doi: 10.12682/lives.2296-1658.2013.24
```

input

Load the data. $ForeGround$ and $BackGround$ are $region \times numofsamples$.

```
#setwd("")
ForeGround <- read_csv("C:/Users/Zhixiang/Data/CellLines6/SelectedPeaksLargeSub/ForeGround.csv", col_na
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer()
## )
## See spec(...) for full column specifications.
```

```
ForeGround <- as.matrix(ForeGround)

BackGround <- read_csv("C:/Users/Zhixiang/Data/CellLines6/SelectedPeaksLargeSub/BackGround.csv", col_na
```

```
## Parsed with column specification:
## cols(
##   .default = col_integer()
## )
## See spec(...) for full column specifications.
```

```
BackGround <- as.matrix(BackGround)
```

run weighted K-medoid

Calculate 1 - Spearman

```
distS <- 1-cor(ForeGround, method="spearman")
```

Calculate the median of BackGround for each sample

```
BackGroundMedian <- apply(BackGround, 2, median)
```

Use the sigmoid function $\frac{1}{1+\exp^{-\lambda(BackGroundMedian-a)}}$ to calculate the weight. λ and a are two tuning parameters. By default $a = quantile(BackGroundMedian, 0.5)$ and $\lambda = 1$. $clusterW$ is the clustering result for each sample

```
nCluster <- 6
lambda <- 1
a <- quantile(BackGroundMedian, 0.5)
W <- 1/(1+exp(-lambda*(BackGroundMedian-a)))
resultW <- wcKMedoids(distS, k=nCluster, weights=W)
clusterW <- resultW$clustering
clusterW <- as.numeric(factor(clusterW))
```

get landmark

$landmark$ is $numofpeaks \times numoflanmarks$

```
landmarks <- c()
  for (i in 1:nCluster){
    tmp <- which(clusterW==i)
    if (length(tmp)==1){
        landmarks <- cbind(landmarks, ForeGround[,tmp]   )
    } else {
        landmarks <- cbind(landmarks, rowSums(ForeGround[,tmp])   )
    }
  }
```

```
selectTop <- function(x, top){
  thres <- sort(x, decreasing=T)[top]
  x[x<thres] <- 0
  return(x)
}
```

pick the top peaks in the landmarks

```
top <- 2000
landmarksTop <- apply(landmarks, 2, selectTop, top)
```

run KNN

```r
scor <- cor(ForeGround, landmarksTop, method="spearman")
clusterWKNN <- apply(scor, 1, which.max)
```

check the clustering result. optional

samples is a length n vector with the true cell types for each sample. Same order as the data matrix

```r
samples <- read_delim("C:/Users/Zhixiang/Data/CellLines6/SelectedPeaksLargeSub/SampleOrder.txt", "\t",
```

```
## Parsed with column specification:
## cols(
##   X1 = col_character(),
##   X2 = col_character()
## )
```

```r
samples <- samples[,2][[1]]
```

order of the cell types.

```r
cells <- c("K562", "GM12878", "HL-60", "BJ", "TF-1", "H1")
```

getClusterCount gets the cell type count for each cluster. *getCorrectCount* calculates the clustering accuracy.

```r
getClusterCount <- function(cluster, samples, cells){
  ### input
  #cluster: clustering result for each sample
  #samples: vector with the cell types for each sample
  #cells: order of the cell types
  clusterCount <- matrix(0, nrow=length(unique(cluster)), ncol=length(cells))
  for (i in 1:length(unique(cluster))){
    tmp <- samples[which(cluster==i)]
    for (j in 1:length(cells)){
      cell <- cells[j]
      clusterCount[i, j] <- sum(tmp==cell)
    }
  }
  tmp <- apply(clusterCount, 2, which.max)
  if (max(table(tmp))>1){
    seqs <- c()
    for (i in 1:nrow(clusterCount)){
      if (i==1){
        seqs <- c(seqs, which.max(clusterCount[,i])[1])
      }
      if (i>1 & i<nrow(clusterCount)){
        seqs <- c(seqs, c(c(1:nrow(clusterCount))[-seqs])[which.max(clusterCount[-seqs,i])[1]])
      }
      if (i==nrow(clusterCount)){
        seqs <- c(seqs, c(1:nrow(clusterCount))[-seqs])
      }
    }
  } else {
    seqs <- tmp
  }
  clusterCount <- clusterCount[seqs,]
  row.names(clusterCount) <- paste("cluster", 1:length(unique(cluster)))
  colnames(clusterCount) <- cells
  return(clusterCount)
}

getCorrectCount <- function(clusterCount){
  ### input
  #c lusterCount: output from getClusterCount
  ### output, we assign the cell type of each cluster by the majority
  # c(the number of correctly clusterd cells, the percentage of correctly clustered cells)
  return(c( sum(apply(clusterCount, 1, max)),  sum(apply(clusterCount, 1, max))/sum(clusterCount)) )
}
```

examples:

weighted K-medoids

```
ClusterCountW <- getClusterCount(cluster=clusterW, samples=samples, cells=cells)
CorrectCountW <- getCorrectCount(ClusterCountW)
print(ClusterCountW)
```

```
##           K562 GM12878 HL-60 BJ TF-1 H1
## cluster 1  660       1     1  0    3  0
## cluster 2    0     372     3  0    2  0
## cluster 3    2       0    88  0    1  0
## cluster 4    0       0     0 75    0  0
## cluster 5    4       0     0  0   87  0
## cluster 6    0       0     0  0    1 77
```

```
print(CorrectCountW)
```

```
## [1] 1359.0000000    0.9869281
```

weighted K-medoids + KNN

```
ClusterCountWKNN <- getClusterCount(cluster=clusterWKNN, samples=samples, cells=cells)
CorrectCountWKNN <- getCorrectCount(ClusterCountWKNN)
print(ClusterCountWKNN)
```

```
##           K562 GM12878 HL-60 BJ TF-1 H1
## cluster 1  665       0     0  0    1  0
## cluster 2    0     372     0  0    0  0
## cluster 3    0       0    92  0    1  0
## cluster 4    0       0     0 75    0  0
## cluster 5    1       0     0  0   92  0
## cluster 6    0       1     0  0    0 77
```

```
print(CorrectCountWKNN)
```

```
## [1] 1373.0000000    0.9970951
```