RAProject

Mingzhi Ye

2022/10/10

#Global parameter

```
poolSize=2000
SampleSize=5000
```

Get the combined distribution of number of Alpha chains and number of Beta chains in a clonotype

```
IndexnAnBMap=read.csv("IndexnAnBMap.csv",header = F)
IndexnAnBMap=as.matrix(IndexnAnBMap)
```

Get a template dataframe of a clonotype which has one alpha chain and one beta chain

```
sce0 <- readRDS('toyTCRdata.rds')
TemplateClonotype=sce0$contigs[[4]][c(2,1),]
```

Generate a map which map generated random int to the index in sample pool

```
MapRandomIntToIndex=rgeom(poolSize, .25)+1
getRep <- function(x){


  return(rep(x,MapRandomIntToIndex[x]))
}
MapRandomIntToIndex=sapply(1:poolSize,getRep,simplify = TRUE)
MapRandomIntToIndex=unlist(MapRandomIntToIndex)
MapLength=length(MapRandomIntToIndex)
```

Generate sample pool

```r
AmiNoAcids=c('A','R','N','D','C','Q','E','G','H','I','L','K','M','F','P','S','T','W','Y','V')
getCDR3 <- function(x){

  cdr3=sample(AmiNoAcids,15, replace=T)
  return(paste(cdr3,collapse = ''))
}



getClonotypeForPool<-function(clonotype_index, CDRAlpha, CDRBeta){
    result=TemplateClonotype
    result$cdr3[1]=CDRAlpha
    result$cdr3[2]=CDRBeta
    return(result)
}
getSamplePool <- function(poolSize){
    Alpha_pool=sapply(1:poolSize,getCDR3,simplify = TRUE)
    Beta_pool=sapply(1:poolSize,getCDR3,simplify = TRUE)

    CDR_Alpha=sample(Alpha_pool,poolSize, replace=T)
    CDR_Beta=sample(Beta_pool,poolSize, replace=T)




    barcode <- paste('cell', 1:poolSize)


    samplePool <- lapply(1:poolSize, function(clonotype_index){
        getClonotypeForPool(clonotype_index, CDR_Alpha[clonotype_index], CDR_Beta[clonotype_index])
    })
    samplePool <- SplitDataFrameList(samplePool)
    samplePool[,'barcode'] <- barcode
    samplePool[,'sample'] <- 'sim'
    names(samplePool) <- barcode

    # sample1<-as(sample,'SplitDFrameList')
    return(samplePool)

}
samplePool=getSamplePool(poolSize)
```

Generate the dataframe recording the abundance for each clonotype in the sample pool

```r
clonotypes_in_pool=samplePool[,'cdr3']
concatCDR3<-function(x){
  res=paste(x[1],x[2],sep=' ')
    res=paste(x[1],x[2],sep=' ')
    return(res)
}
clonotypes_in_pool=lapply(clonotypes_in_pool,concatCDR3)
clonotypes_in_pool=unlist(clonotypes_in_pool)
abundance=rep(1/poolSize,poolSize)
realDF<- data.frame(clonotypes_in_pool,abundance)
rownames(realDF) <- NULL
realDF=aggregate(realDF$abundance, by=list(clonotype=realDF$clonotypes_in_pool ), FUN=sum)
colnames(realDF) <- c('clonotype','abundance')
```

Functions for generate a sample

```
getClonotype<-function(clonotype_index, numberA, numberB, errorProb = .01){
    result=DataFrame()
    indexInPool=sample.int(MapLength, 3, replace=FALSE)
    indexInPool=MapRandomIntToIndex[indexInPool]
    setAlpha <- hashset()
    setBeta <- hashset()

    if (numberA>=1){
        if(runif(1) < errorProb){
          result=rbind(result,samplePool[[indexInPool[2]]][1,])
          insert(setAlpha,indexInPool[2])
        }else{
          result=rbind(result,samplePool[[indexInPool[1]]][1,])
          insert(setAlpha,indexInPool[1])
        }
        numberA=numberA-1
    }
    if (numberB>=1){
        if(runif(1) < errorProb){
          result=rbind(result,samplePool[[indexInPool[3]]][2,])
          insert(setBeta,indexInPool[3])
        }else{
          result=rbind(result,samplePool[[indexInPool[1]]][2,])
          insert(setBeta,indexInPool[1])
        }

        numberB=numberB-1
    }
    while(numberA>0){
        indexInPool=sample.int(MapLength, 1, replace=TRUE)
        indexInPool=MapRandomIntToIndex[indexInPool]
        while(setAlpha[indexInPool]==TRUE){
          indexInPool=sample.int(MapLength, 1, replace=TRUE)
          indexInPool=MapRandomIntToIndex[indexInPool]
        }
        insert(setAlpha,indexInPool)
        result=rbind(result,samplePool[[indexInPool]][1,])
        numberA=numberA-1


    }
    while(numberB>0){
        indexInPool=sample.int(MapLength, 1, replace=TRUE)
        indexInPool=MapRandomIntToIndex[indexInPool]
        while(setBeta[indexInPool]==TRUE){
          indexInPool=sample.int(MapLength, 1, replace=TRUE)
          indexInPool=MapRandomIntToIndex[indexInPool]
        }
        insert(setBeta,indexInPool)
        result=rbind(result,samplePool[[indexInPool]][2,])
        numberB=numberB-1


    }
    return(result)

}
```

Use the functions to generate a sample with size 50

However, function ‘clonoStats’ is not compatible with SimpleDFrameList, how to transform SimpleDFrameList to CompressedSplitDFrameList

```
getSample <- function(samplesize){
    RandomIntegers <- sample(1:(length(IndexnAnBMap)/2), samplesize, replace=T)

    barcode <- paste('cell', 1:samplesize)


    sample <- lapply(1:samplesize, function(clonotype_index){
        getClonotype(clonotype_index, IndexnAnBMap[RandomIntegers[clonotype_index],1], IndexnAnBMap[Ran
    })
    sample <- SplitDataFrameList(sample)
    sample[,'barcode'] <- barcode
    sample[,'sample'] <- 'sim'
    names(sample) <- barcode

    # sample1<-as(sample,'SplitDFrameList')
    return(sample)

}
# samplelist=getSample(SampleSize)
# EMpredicted <- clonoStats(samplelist, method = 'EM')
# UNpredicted <- clonoStats(samplelist, method = 'unique')
```

Get Variation Distance

```r
getDistance_R<-function(clonotype,abundance){
  stimulated<- data.frame(clonotype,abundance)

  sum_abundance=sum(stimulated['abundance'])
  stimulated['abundance']=stimulated['abundance']/sum_abundance

  merged = merge(x = realDF, y = stimulated, by = "clonotype",all = TRUE)
  merged[is.na(merged)] <- 0
  return (sum(abs(merged['abundance.x']-merged['abundance.y'])))
}
```

Test

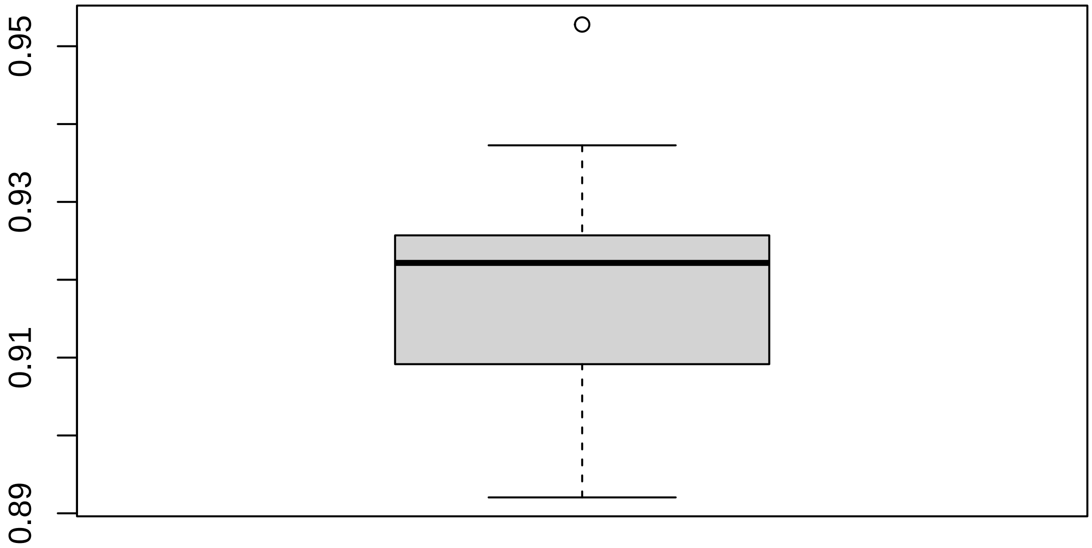
```r
time0=Sys.time()
samplelist=getSample(5000)
time1=Sys.time()
print(time1-time0)
EMpredicted <- clonoStats(samplelist, method = 'EM')
time2=Sys.time()
print(time2-time1)
distance1=getDistance_R(clonoNames(EMpredicted),clonoAbundance(EMpredicted)[,1])
time3=Sys.time()
print(time3-time2)
UNpredicted <- clonoStats(samplelist, method = 'unique')
time4=Sys.time()
print(time4-time3)
distance2=getDistance_R(clonoNames(UNpredicted),clonoAbundance(UNpredicted)[,1])
time5=Sys.time()
print(time5-time4)
```

Calculate distances for samples

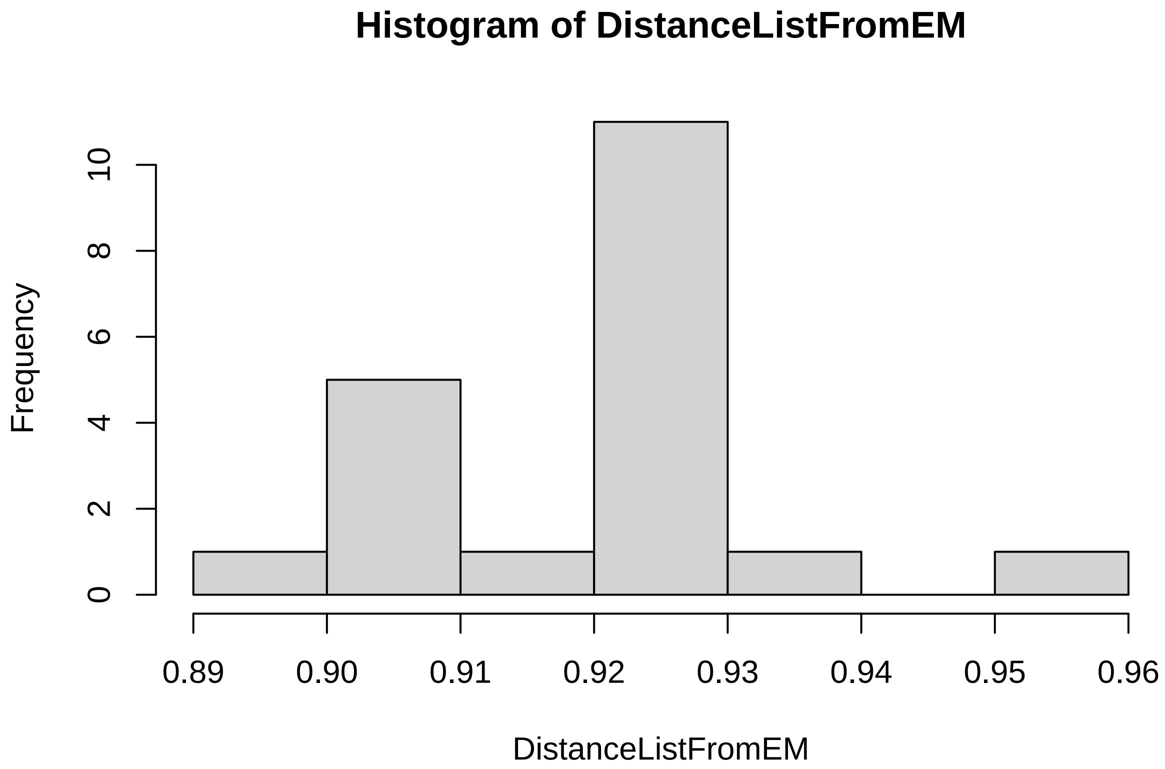
```r
DistanceListFromEM=c()
DistanceListFromUN=c()
for(i in 1:20){
  samplelist=getSample(SampleSize)
  EMpredicted <- clonoStats(samplelist, method = 'EM')
  distance1=getDistance_R(clonoNames(EMpredicted),clonoAbundance(EMpredicted)[,1])
  DistanceListFromEM[i]=distance1
  UNpredicted <- clonoStats(samplelist, method = 'unique')
  distance2=getDistance_R(clonoNames(UNpredicted),clonoAbundance(UNpredicted)[,1])
  DistanceListFromUN[i]=distance2
}
```

Visualize

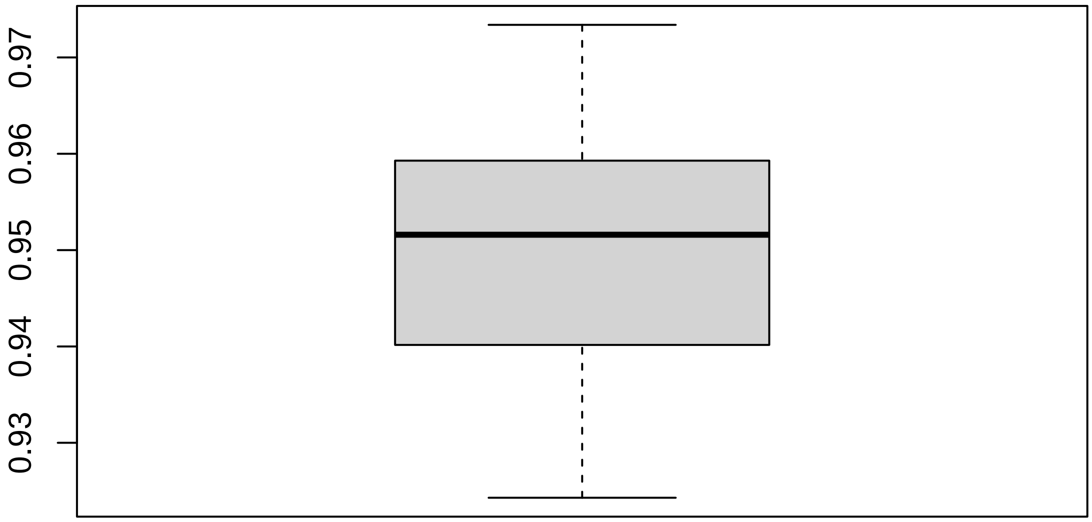
```
boxplot(DistanceListFromEM)
```



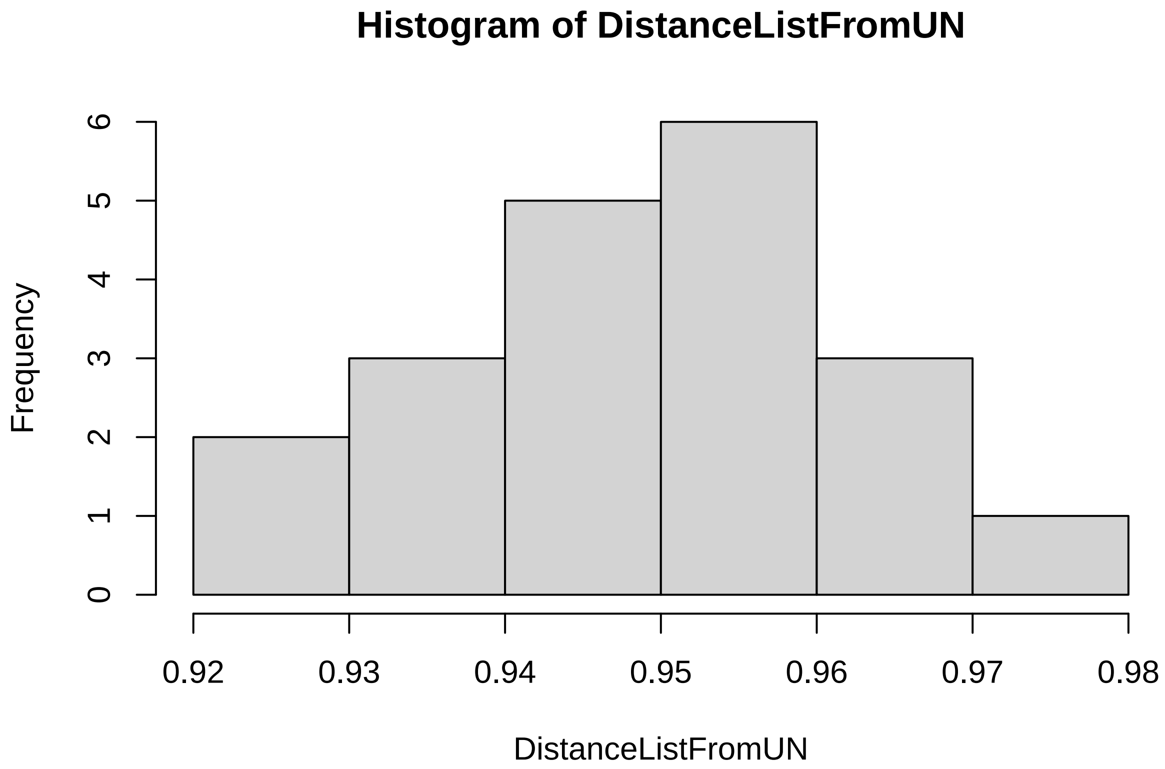
```
hist(DistanceListFromEM)
```



```
boxplot(DistanceListFromUN)
```



```
hist(DistanceListFromUN)
```



```
print(median(DistanceListFromEM))
```

```
## [1] 0.9221661
```

```
print(median(DistanceListFromUN))
```

```
## [1] 0.9515974
```

```
print(mean(DistanceListFromEM))
```

```
## [1] 0.9193623
```

```
print(mean(DistanceListFromUN))
```

```
## [1] 0.9502348
```