VignettePTRViewer

Premiliminary work

This vignette shows how to use chemosensR package to read .txt files, calculate AUC and return statistical analyses. This command updates the chemosensR package

```r
library(PTRMSR)
library(devtools)
library(MSnbase)
library(reshape2)
library(pheatmap)
library(ggplot2)

library(ellipse)
install_github("https://github.com/ChemoSens/ChemoSensPrivate/CSUtils")
install_github("https://github.com/ChemoSens/PTRMSR")
```

Then, chemosensR should be loaded.

```r
library(PTRMSR)
```

Meta-data

All the data files (.txt from ptrViewer) and the metadata file (.csv) should be in a single repository (wd)

```r
wd="C:/INRA/Data/Donnees Cantin/Cantin-DTS-PTRviewer"
setwd(wd)
listFiles=list.files(pattern="*.txt")[-1]
metaData2=read.table("metaData2.csv",sep=";",header=T)
head(metaData2[,-c(2:3)])
```

Analysis of PTR-Viewer files

Selection of relevant ions only

This function returns the names of ions whose maximal intensity is 3 times higher than the maximal intensity during the noise period. It returns a list containing (i) a list containing for each file the ratio maximal intensity during the tasting/maximal intensity during the noise period, (ii) a vector (intersection) containing the ions which are significant in all files (iii) a vector (union) containing the ions which are significant in at least one file

```r
setwd(wd)
referenceBreath="m69.06906..69.06906...Conc."
sigIons=ptrvListSignificantSNRIons(listFiles=listFiles[1:4],
               metaData=metaData2,noisePeriod=c(0,25))
ionSigUnique=sigIons$union
```

To obtain further help for this (or any) function, enter:

```r
?ptrvListSignificantSNRIons
```

AUC calculations

This function allows statistics to be calculated for each file after breathing correction

The results can be saved into csv files

```
res_auc$listRes
write.table(file="auc.csv",sep=";",res_auc$listRes,row.names=F)
```

The stat option can be used to select the statistic.

```
res_tmax=ptrvListIntensityByTime(listFiles=listFiles,metaData=metaData2,ions=ionSigUnique,stat="tmax")
```

Statistical analysis

In this part, the evaluations can be normalized (a total abundance of 1 for each evaluation) or logged. ### PCA of observations

```
respca=ptrvListPCA(ptrvList$totalIntensity,dataType="productMeans",log=FALSE)
# rajouter projetions individuelles plutot qu'ellipses
plotPCAgg(respca,type="ind")
plotPCAgg(respca,type="var",text=FALSE)
p=plotPCAgg(respca,type="var",text=FALSE)
p
library(plotly)
ggplotly(p)

p=plotPCAgg(respca,type="cor1",n=80)
plotPCAgg(respca,type="cor2")

respca=ptrvListPCA(ptrvList$totalIntensity,dataType="raw")
p=plotPCAgg(respca,type="ind",text=FALSE)
ggplotly(p)
plotPCAgg(respca,type="var",text=FALSE)
```

Anova results

```
resanova=ptrvListAnova(ptrvList$totalIntensity[ptrvList$totalIntensity[,"ion"]=="m101.09",],normalizeBy
```

Heatmaps

```
ptrvListHeatmap(df=ptrvList$totalIntensity,formula=as.formula(product+subject+rep~ion),
                fun.aggregate="mean",clusterRows=T,clusterCols=T)
ptrvListHeatmap(df=ptrvList$totalIntensity,formula=as.formula(product+subject+rep~ion),
                fun.aggregate="mean",clusterRows=T,clusterCols=T)
ptrvListHeatmap(df=ptrvList$totalIntensity,formula=as.formula(product+subject+rep~ion),
                fun.aggregate="max",clusterRows=T,clusterCols=T)
ptrvListHeatmap(df=ptrvList$totalIntensity,formula=as.formula(product+subject~ion),
                fun.aggregate="max",clusterRows=T,clusterCols=T,showRownames=T)
ptrvListHeatmap(df=ptrvList$totalIntensity,formula=as.formula(product+subject~ion),
                fun.aggregate="mean",normalization="row",clusterRows=T,clusterCols=T,showRownames=T)
ptrvListHeatmap(df=ptrvList$totalIntensity,formula=as.formula(product+subject~ion),
                fun.aggregate="mean",normalization="row",clusterRows=T,clusterCols=T,showRownames=T,ann
```

Outputs for one given file

It could be interesting to see the raw data of one file. In this purpose, the function ptrReport returns several plots which could help to interpretation.

```
setwd(wd)
file=listFiles[1]
dataset=read.table(file=file,header=TRUE,sep="\t")
report=ptrvReport(dataset,selecIons="evolving",
                  listIons=ionSigUnique[1:3],
                  referenceBreath=referenceBreath,
                  methodDetectStart="startPeakProportion",
                  noisePeriodIBT=c(0,30),noisePeriodSig=c(0,30),
                  noisePeriodDS=c(0,30),
                  proportionOfMax=0.3,halfWindowSize=12,maxPeaks=30)
names(report$gg)
report$gg$p_breath
```

Regarding the breathing cycle detection, these two outputs gives the results of cycle limits and the smooth data for breathing (useful to adjust the parameters halfWindowSize and maxPeaks)

```
plot(report$gg$p_breath$p_cyclelimits)
plot(report$gg$p_breath$p_smoothbreath)
```

Regarding the ions distributions

```
plot(report$gg$p_curves$p_raw)
plot(report$gg$p_curves$p_cycle)
```

Regarding the breathing ion distribution

```
plot(report$gg$p_curves$p_breath_raw)
plot(report$gg$p_curves$p_breath_cycle)
```

Do not hesitate to use plotly for interactive graphs

```
library(plotly)
ggplotly(report$gg$p_curves$p_breath_raw)
```

Appendix

Help for filling meta-data

```
ptrvCreateMetaFile(wd,subject=c(9,12),product=NULL,replicate=NULL,sep=";")
metaData=read.csv("metaData.csv",sep=";",header=TRUE,dec=",")
metaData[,"resp"]=referenceBreath
```

###[OPTIONAL] Completing meta-data file with the tasting time

```
# courbes + lissage
 t0=tn=suj=ordre_produit=product=repet=finTS=rep(NA,nrow(metaData))
  names(t0)=listFiles
  for(i in 1:nrow(metaData))
  {
    print("dataset")
    print(i)
    dataset=read.table(listFiles[i],sep="\t",dec=dec_vec[i],header=T)
    dataset[,"RelTime"]=as.numeric(dataset[,"RelTime"])
```

```r
   # dataset[,ions]=apply(dataset[,ions],2,as.numeric)
    res_intensity=ptrvIntensityByTime(dataset,ions=ionSigAll
                                      ,referenceBreath=referenceBreath,
                                      correction = "cycle",
                                      removeNoise=TRUE,breathRatio =FALSE,timeBlank=c(0,25),
                                    halfWindowSize=12,maxPeaks=30,method="SuperSmoother",total=FALSE)
    t0[i]=ptrvDetectStart(res=res_intensity,starts=ionSigAll, method="startPeakProportion",proportionOf
                          noisePeriod=c(0,25),startPeriod=c(25,100))$tx
    tn[i]=max(res_intensity$res[,"time"])
    suj[i]=substr(listFiles[i],9,12)
    ordre_produit[i]=substr(listFiles[i],14,14)
    tds_raw_i=tds_raw[tds_raw[,"SubjectCode"]==suj[i]&tds_raw[,"Rang.produit"]==ordre_produit[i],]
    product[i]=unique(tds_raw_i[,"ProductCode"])
    repet[i]=unique(tds_raw_i[,"Replicate"])
    finTS[i]=max(tds_raw_i[,"Time"])
  }
metaData[,"miseEnBouche_s"]=t0;metaData[,"finPTR_s"]=tn;
metaData[,"finTimeSens_s"]=finTS
metaData[,"product"]=product;metaData[,"subject"]=suj;metaData[,"rep"]=repet
write.table(metaData,file="metaData2.csv",sep=";",row.names=F,dec=".")
```

##Installing chemosensR (to be done only one time) As a first step, chemosensR package should be installed. As chemosensR is on a private GitHub repository, this step required the use of a token which is send to the user.

```r
install.packages("devtools")
library(devtools)
install_github("MahieuB/MultiResponseR")
if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install("MSnbase")
BiocManager::install("rhdf5")
install.packages("multcompView")
#BiocManager::install("mzR")
install_github("https://github.com/ChemoSens/ChemoSensPrivate/ChemosensR",             auth_token="d92a

# FOR SHINY
setwd(paste0(.libPaths()[1],"/chemosensR/extdata"))
then open the .R file and run
```

Then, the package has to be loaded with library function. Contrarily to the previous installation, this step of loading is required at any use of chemosensR package.