ANÁLISIS DE DATOS ÓMICOS

PEC1

T SAN-MIGUEL DIEZ

Abstract

Este informe recoge mi primera experiencia en análisis de datos ómicos. A partir de datos depositados en un repositorio en GitHub y siguiendo las instrucciones guiadas en pyhton que lo acompañaban, he tratado de replicar la experiencia en Rstudio, con las peculiaridades explicadas en la asignatura, siendo la principal, la creación de un objeto SummarizedExperiment. Este informe contiene parcialmente los códigos necesarios-no en el orden adecuado, sino adaptado al formato aquí presentado- para la inspección de los datos, la generación del objeto de estudio, la limpieza de los datos, el análisis para determinar la reproducibilidad técnica de la obtención de los datos, el análisis estadístico y la generación de un modelo de aprendizaje-máquina para, a través del análisis metabolómico de muestras biológicas, diferencias individuos sanos y personas afectadas de cáncer gástrico.

Objetivos

1. Ser capaz de utilizar datos depositados por otros investigadores en un repositorio. En este caso, datos metabolómicos en individuos sanos, con tumores benignos y con cáncer gástrico.
2. Explorar los datos depositados. Ver las características de la información y determinar si hay datos faltantes.
3. Construir un objeto de clase SummarizedExperiemnt
4. Evaluar los datos y analizarlos estadísticamente
5. Generar un modelo de aprendizaje-máquina discriminante.

Métodos

Tras clonar el repositorio https://github.com/nutrimetabolomics/metaboData/tree/main al escritorio, se exploran los ficheros list.files(), se identifica el proyecto de interés y se inspecciona la información que hay. Se extraer la información read_excel() y se inspecciona de manera rápida con glimpse(). El código completo para las primeras etapas, que incluyen desde la exploración del fichero, los cambios necesarios para la creación del SummarizedExperiment, la limpieza de datos, el SummarizedExperiment tras el filtrado por calidad y el análisis de PCA para comprobar la calidad de la técnica se recogen en el anexo 1.

1. Creación del objeto SummarizedExperiment

Para la creación del objeto SummarizedExperiment (SE) las etapas realizadas han sido:

1.1 Preparación de la matriz de expresión de datos. Con as.matrix() y transponiendo la matriz. Se seleccionan las columnas (variables) que incluyen el valor obtenido en cada una de las muestras. Esto servirá como elemento **assay** del SE y las columnas iniciales descriptivas se llevarán al segundo elemento del SE.

1.2 Preparación de la información sobre las muestras incluidas (metadatos). Se prepara el elemento **colData** del SE, con la información de esas primeras columnas de la hoja de datos. Usamos select().

1.3 Preparación de la información sobre los picos. Se prepara el elemento **rowData** a partir de la información de la hoja de picos.

1.4 Creación del objeto SummarizedExperiment.*Respondiendo a la pregunta sobre comparar los objetos SummarizedExperiment y ExpressionSet:* tanto SummarizedExperiments (SE) como ExpressionSet(ES) son clases S4 de Bioconductor para organizar datos ómicos. La estructura de los ES es la utilizada clásicamente para análisis de microarrays y soporta principalmente datos de expresión génica. Los datos se almacenan como assayData o phenoData y featureData, mientras que en el que se solicita para este proyecto, SE, tiene una estructura más moderna, que soporta cualquier tipo de dato ómico y se almacena como **assays, rowData y colData**. La ventaja parece ser la flexibilidad y compatibilidad.

```
se <- SummarizedExperiment(
  assays = list(counts = matriz_datos),
  colData = meta_muestras,
  rowData = rowData
)
```

Al explorar veo que puedo añadir como metadata al SE directamente la fuente de los datos y la fecha.

2. Limpieza de datos

Siguiendo las indicaciones del tutorial de análisis usando Python, que menciona la información del repositorio clonado, como buenas prácticas aplicamos 2 criterios. La proporción de datos faltantes *Perc_missing*; si es mayor al 10% hay mucho dato faltante y es mejor eliminarlo. La variabilidad entre réplicas *QC_RSD*; si es mayor al 20% se considera muy variable. Tras la limpieza de datos se construye un segundo SummarizedExperiment, ahora filtrado.

3. Análisis de componentes principales PCA

Tras leer el tutorial en python, he visto una serie de procedimientos que indago para qué sirven. 1-Transforman los datos con log10: esto reduce el efecto de valores extremos y distribuciones sesgadas; ayuda a que el PCA se concentre en patrones más biológicos. La escalabilidad de los valores mejora al reducir amplitudes de orden de magnitud. 2-En vez de imputar medianas en los valores faltantes, que es mi enfoque clásico, imputan k-NN (k=3); es la llamada imputación con 3 vecinos más cercanos; esto mantiene mejor la estructura de los datos SIEMPRE QUE los QC (controles de calidad) estén bien medidos (esto lo sabré cuando los visualice en el PCE. 3-Hacen el PCA para ver si se agrupan controles *vs* resto de muestras. Además del PCA, se prepara una tabla para leer la carga de cada metabolito en el PCA. El código hasta este punto se encuentra en el anexo 2.

4. Análisis Estadístico

Los códigos utilizados se incluyen en el anexo 3. Siguiendo la idea del tutorial, comparamos los metabolitos que hay en los pacientes con cáncer gástrico (GC) Y en los individuos sanos (HE). Para ellos hacemos un ***filtrado de datos:*** evitamos que estén los QC y lo tumores benignos. A continuación, recurrimos a ***comparaciones no paramétricas: s***i estuviéramos en análisis de 5-10 variables, haría Shapiro-Wilk para comprobar normalidad, y Levene o Bartlett para la homogeneidad de varianzas, pero por lo visto con datos ómicos esto no es recomendable. Al haber muchos test, el error tipo I aumenta. En metabolómica se asume que la mayoría de variables no siguen una distribución perfectamente normal, suele haber outliers y/o asimetrías importantes, las varianzas entre grupos tampoco suelen ser homogéneas, y por tanto está aceptado recurrir **a test no paramétricos como *Wilcoxon***. Por contra, si quisiera usar un t-test sí que tendría que justificar que se cumplen los supuestos para usarlo.

Se crea una función de test para comparar cada metabolito, hacemos **Wilcoxon Rank Sum test**, y además, calculamos el fold change (el fold change indica cuantas veces es superior o inferior la *mediana*, 2 es el doble/0.5 es la mitad). Se genera una tabla con los p valores ajustados por FDR = False Discovery Rate. Cuando se hacen muchos tests estadísticos, para controlar el error tipo I, el ajuste por FDR (por ejemplo, con el método Benjamini-Hochberg) corrige los p-valores para que controles este error. Se genera un **Volcano Plot** para visualizar metabolitos diferenciales entre pacientes con cáncer gástrico (GC) e individuos sanos (HE).

```
ggplot(tabla_con_nombres, aes(x = log2FC, y = negLogP, color = Significativo)) +
  geom_point() +
  geom_vline(xintercept = c(-1, 1), linetype = "dotted") +
  geom_hline(yintercept = -log10(0.05), linetype = "dashed", color = "red") +
  scale_color_manual(values = c("GC ↑" = "red", "HE ↑" = "blue", "No significativo" = "grey")) +
  labs(title = "Volcano plot: GC vs HE",
       x = "log2(Fold Change)",
       y = "-log10(p-valor)") +
  theme_minimal()
```

No obstante, dado que el tutorial en phyton que se sigue como ejemplo, genera una tabla para evaluar los supuestos que deben cumplirse para usar test paramétricos, se construye la misma tabla aquí. Incluye valores medios, IC95%, test de Shapiro-Wilk y test de Levene, además de TRUE o FALSE para visualmente identificarlo.

5. Machine Learning

Para ver si esta colección nos permite obtener un modelo predictivo, con el método PLS-DA, recurrimos a las siguientes etapas que se recogen en el anexo 4:

División de datos entrenamiento/prueba. Se hace una *partición estratificada* para garantizar que no se acumulen las muestras de un tipo (GC o HE) en uno de los dos set.

- Transponer y transformar los datos. Se debe hacer una serie de cambios para el modelo: Transponer, transformar con logaritmo 10 los datos e imputar por le método knn (en realidad lo había hecho antes para el PCA, pero no está guardado en mi SE_filtrado por lo que el PLS-DA falla si no vuelvo a hacerlo).

- Dividir entre train y test. Se distribuyen las muestras entre set de entrenamiento (train) y de prueba (test).

- Hacer un escalado con Z-score. Se realiza un escalado Z-score con scale() antes proceder a generar el modelo, con las medias y desviación estándar del train.

A continuación se crea el modelo PLS-DA y se entrena. Y la siguiente fase es evaluar la **validez del modelo**, mediante validación cruzada con el método perf() de mixomics y extraer métricas de error (error global usando 2 componentes y distancia por centroides; error por clase usando 2 componentes y distancia por centroides y número óptimo de componentes según error global). Procedemos a ***determinar las variables más importantes*** obteniendo los VIP scores, con vip() que permite ver en una tabla la contribución de los metabolitos a cada una de las componentes del modelo. Se acaba ***evaluando el modelo con los datos de prueba,*** es decir, queremos predecir qué tal funciona. Se usa factor() y predict() y se construye la matriz de confusión (anexo 4:).

Resultados

Se selecciona `Datasets/2023-CIMCBTutorial/GastricCancer_NMR.xlsx` y se explora el fichero con excel_sheets() para visualizar cuantas hojas tiene. Es importante utilizar el código adecuado para explorar que el Excel no tiene una única hoja. Se encuentran dos pestañas. La primera tiene la información de qué metabolitos hay en cada una de las muestras e información adicional sobre las muestras; es decir, cada fila es una muestra analizada, las primeras columnas (variables) son identificativas y caracterizadoras de cada muestra, y a continuación, una columna por cada metabolito explorado. La segunda contiene la información sobre los metabolitos analizados y sus indicios de calidad; es decir, cada fila es un metabolito, y las columnas son identificativas de metabolitos y los valores que cada metabolito ha obtenido con respecto a parámetros de calidad. La dataset escogida se encuentra en un repositorio con un enlace a un tutorial en Python para su análisis. Aunque la herramienta de análisis en este trabajo es diferente, la guía que supone poder ver las etapas tiene un valor inestimable como elemento formativo.

1. Creación del objeto SummarizedExperiment

Con los métodos indicados en el apartado anterior se construye el **SE** con assays que incluye la matriz transpuesta de datos, colData que incluye la información en meta_muestras (`Idx, SampleID, SampleType, Class`) y el rowData que incluye los picos.

2. Limpieza de datos

Tras la limpieza de datos, hay 52 metabolitos de calidad en 140 muestras; por suerte puedo comparar con el filtrado del tutorial y el resultado es coincidente. Se obtiene un SummarizedExperiment denominado SE_filtrado con los datos que cumplen estos parámetros de calidad.

3. Análisis de componentes principales PCA

Tras la transformación log10 y la imputación del tercer vecino más cercano, se hace el análisis PCA (figura 1).

```
library(factoextra)
fviz_pca_ind(res.pca,
             geom.ind = "point",
             habillage = colData(se_filtrado)$SampleType,
             addEllipses = TRUE,
             title = "PCA con log10 + imputación k-NN (QC vs Sample)")
```

El PCA muestra que las CQ están agrupadas, es decir, cada vez que se pone el control, la máquina proporciona valores similares. Esto quiere decir que el aparato funciona correctamente y los controles están ok. Esto, además de confirmar el buen funcionamiento técnico, me confirma que el método de imputación ha sido adecuado. Los valores de las muestras están más dispersos-tendré que analizar los diferentes tipos de muestras en la fase siguiente. Como la tabla generada de contribución ocupa mucho espacio, se genera un gráfico (figura 2.)

```
df_cargas <- df_ordenado

library(ggplot2)
ggplot(df_cargas, aes(x = PC1, y = PC2, label = Nombre)) +
  geom_point(color = "green", size = 3) +
  geom_text(vjust = -0.5, hjust = 0.5, color = "black", size = 3) +
  labs(title = "PCA Loadings",
       x = "PC1",
       y = "PC2") +
  theme_minimal()
```

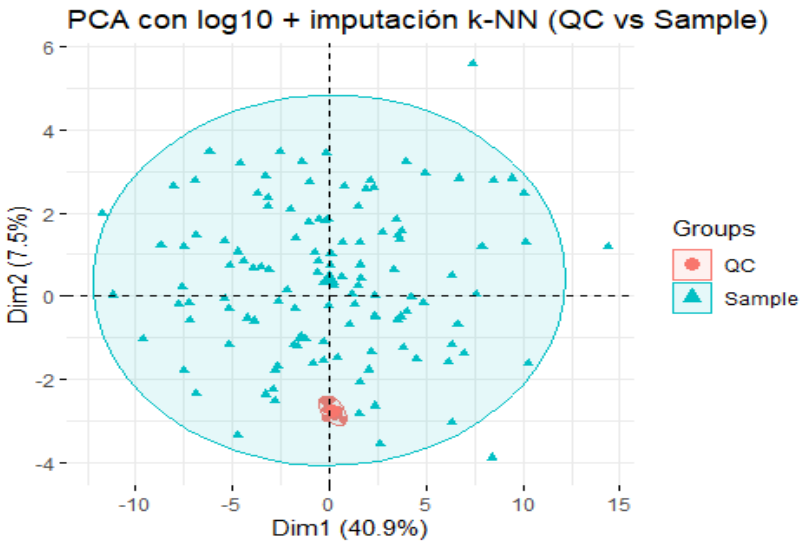

Figura 1. Análisis PCA de 52 metabolitos en 140 muestras. Se resaltan en rojo las muestras *control de calidad (QC)*. Su localización próxima en gráfico indica que las mediciones son fiables y reproducibles. Este test está evaluando el procedimiento de la técnica ómica utilizada.

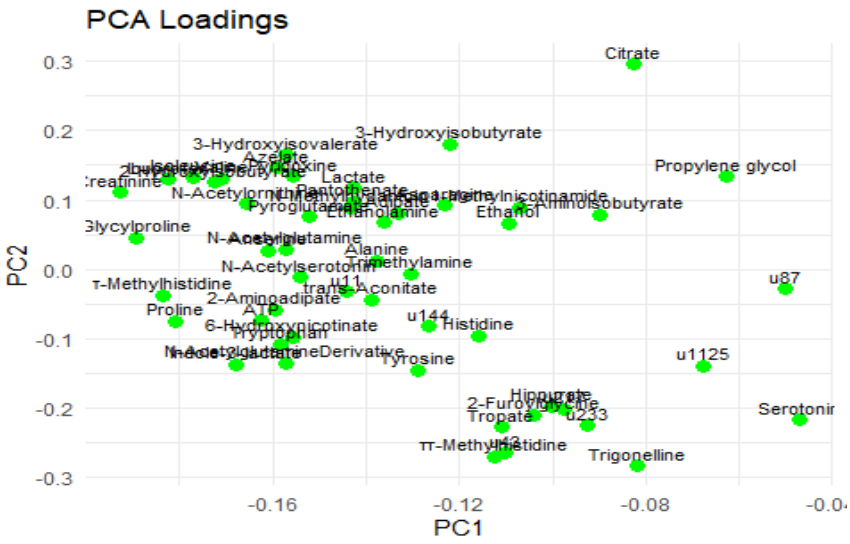

Figura 2. Peso de los diferentes metabolitos en el PCA. Se ha reemplazado el número de metabolito por su nombre

4. Análisis Estadístico

Se comprueba que ningún metabolito cumple los supuestos para hacer pruebas paramétricas. El Wilcoxon Rank Sum test encuentra diferencias significativas que visualizamos con un **Volcano Plot** (figura 3). Es una visualización que combina: *magnitud del cambio* en el eje X: log2(fold change) y *significancia estadística* en el eje Y:-log10(p-valor). Así se ve qué metabolitos cambian más entre GC y HE (fold-change) y cuáles son más significativos (p-valor). Arriba a la derecha: gran aumento en GC con alta significancia (posibles biomarcadores de GC); arriba a la izquierda: gran disminución en GC con alta significancia (posiblemente elevados en HE). Centro: no informativos. Y señalo en rojo los aumentados en cáncer y en azul los aumentados en sanos (en realidad son los disminuidos en cáncer, que es lo mismo que decir los aumentados en sanos). Los más diferencialmente expresados son M138:u233, M89:n-acetilglutamina derivado y M134:u44.

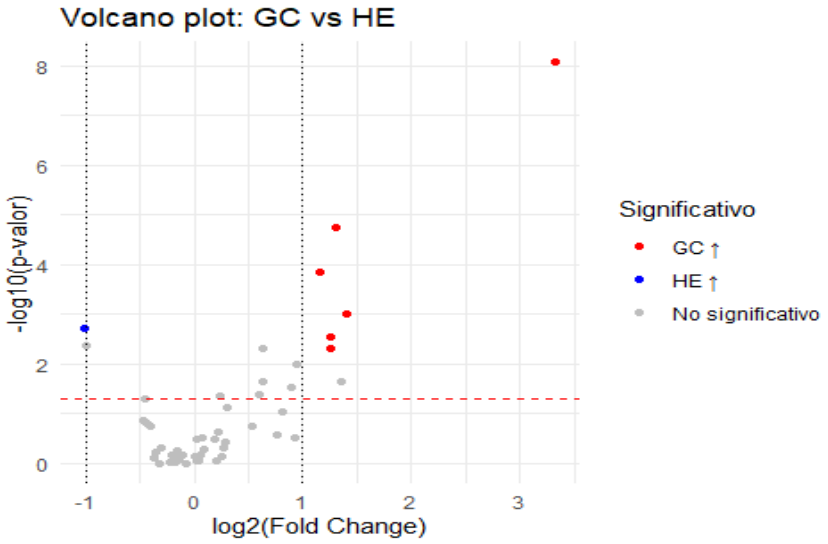

Figura 3. Volcano plot representando los metabolitos diferenciales entre las muestras de cáncer gástrico y de pacientes sanos.

5. Machine Learning

Del PLS-DA realizado obtengo los siguiente. De la validación cruzada:

perf_plsdaerror.rateoverall[“comp2”, “centroids.dist”] Valor: 0.2 Esto significa que el error total de clasificación usando 2 componentes (comp2) y la métrica de distancia entre centroides es del 20%. Es decir, el modelo clasifica correctamente al 80% de las muestras en validación cruzada.

perf_plsdaerror.rate.classcentroids.dist[, “comp2”] valor HE 0.23 y GC 0.17 Para la clase HE, el error es 23%; el 77% de las HE fueron correctamente clasificadas. Para la clase GC, el error es 17.27%; el 82.73% de las GC fueron correctamente clasificadas. El modelo es ligeramente mejor prediciendo GC que HE.

perf_plsda$choice.ncomp[“overall”, “centroids.dist”] Valor: 2 Esto indica que el número óptimo de componentes según la menor tasa de error global usando la distancia entre centroides es 2 componentes.

5.1 Proyección de variables latentes

Proyección de individuos (samples). Muestra cómo se separan las clases GC y HE en el espacio de los componentes latentes (figura 4.A). El gráfico representa la proyección de las muestras en el espacio definido por los dos primeros componentes del PLS. Cada punto representa una muestra, en naranja si es sano y en azul si está enfermo. Las elipses indican la dispersión de cada clase según el intervalo de confianza al 95%. Los ejes se llaman Eje X. X-variate 1: 8% expl.var. El primer componente PLS, explica el 8% de la variabilidad. Eje Y. X-variate 2: 41% expl.var. El segundo componente explica el 41%. Aunque el componente 1 explique poco la variabilidad puede tener un gran poder discriminante

Carga de variables (metabolitos) Esto ayuda a ver qué metabolitos están más asociados a la separación de clases en los componentes. Refleja cuanto contribuye cada variable-aquí, cada metabolito- a los componentes del modelo (figura 4.B). Los metabolitos más alejados del centro tienen más peso.

5.2 Peso de las componentes

VIP scores Se calculan los VIP scores para determinar los metabolitos que más aportan al modelo y se grafica (figura 5).

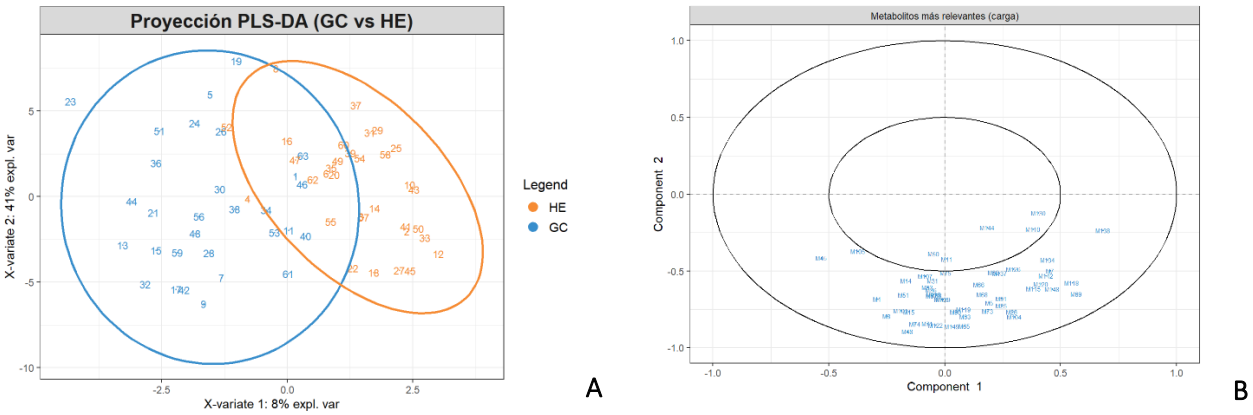

Figura 4. Proyección de variables latentes. A: proyección de samples; B: proyección de metabolitos

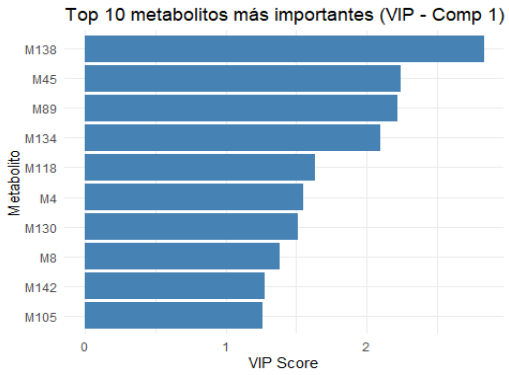

Figura 5. Representación de la contribución del top10 de metabolitos

Podemos ver que los metabolitos que más peso tienen en la componente 1 son:
M138:u233
M45: citrato
M89: n-acetilglutamina derivado
M134:u44

5.2 Evaluación del modelo

La matriz de confusión y el cálculo de sensibilidad y especificidad nos muestran que el modelo es bueno prediciendo (tabla 1). Presenta una sensibilidad: 0.8 y especificidad: 0.9.

Real\Predicho	Gastric Cancer	Sanos
Sanos	2	8
Gástrica Cancer	9	1

Tabla 1. Matriz de confusión.

Discusión

La clonación de proyectos resulta útil para el reanálisis de datos compartidos, e inicialmente sencilla para personas sin experiencia como yo. La construcción de un objeto SummarizedExperiment parece útil para el manejo de datos ómicos. Sin embargo, para un primer contacto la tarea ha sido compleja y no estoy segura de haber operado sobre el objeto siempre que debiera. El proceso ha resultado muy formativo. La disponibilidad de un tutorial en Python que marca los pasos del análisis ha resultado una guía de valor extraordinario para el planteamiento del ejercicio, que debe quedar aquí reflejado pues las ideas no han sido originales.

El filtrado de datos ha reducido de 149 metabolitos a 52 el nº de variables con calidad suficiente. Me sorprende y al mismo tiempo es muy ilustrativo de lo que ocurre en análisis de alto rendimiento. El análisis de PCA inicial es ideal para garantizar la calidad de las muestras control utilizadas. El abordaje estadístico me ha mostrado que las tendencias que se leen en la red, como el uso de test no paramétricos, puede ser demostrado fácilmente y que los datos siguen esa tendencia clásicamente descrita. Además, con el Volcano Plot se ha podido visualizar de manera sencilla que existen unos pocos metabolitos que sí se encuentran de manera diferencial en pacientes con cáncer y personas sanas, e identificar concretamente qué metabolitos están diferencialmente representados en los dos grupos de comparación. La cantidad de muestras incluidas (140) ha permitido generar un modelo PLS-DA sólido, con una buena capacidad predictiva.

Una limitación importante es que creo que no he utilizado la SE_filtrado en el anáisis estadístico, de manera que no he tenido en cuenta los valores faltantes, ni he reducido la dispersión con el log10, y es algo que debería haber repetido pero por falta de tiempo no he podido. En una siguiente fase, sería ideal repetir el procedimiento para ver si es distinguible el conjunto de metabolitos en los pacientes con cáncer gástrico frente a aquellos que tienen tumores benignos.

Conclusiones

El trabajo con control de cambios con github es complicado inicialmente pero cómodo a posteriori. La clonación de repositorios es muy sencilla. Sobre la cuestión biológica:

- Los pacientes con cáncer gástrico pueden distinguirse de las personas sanas mediante un análisis metabolómico.
- Los metabolitos más diferencialmente presentes entre ambos tipos de personas son u233, citrato, N-acetilglutamina derivado y u144.
- La sensibilidad y especificidad del modelo predicitivo aquí señalado que son 0.8 y 0.9 son evidencias de que el modelo es confiable.
- Habría sido imposible para mi realizar este trabajo sin la guía del tutorial en Python enlazado en el repositorio de la dataset.

Referencias

Ese trabajo se ha realizado accediendo al repositorio de GitHub https://github.com/nutrimetabolomics/metaboData
Concretamente https://github.com/nutrimetabolomics/metaboData/tree/79036d1897db72955c0aa0634c1a6aa06d0532fa/Datasets/2023-CIMCBTutorial

Este trabajo se ha colgado con todo el código necesario en
https://github.com/teconsan/SanMiguel_Diez_Teresa_PEC1v1.git

Anexos

PAQUETES UTILIZADOS

```
if (!require("BiocManager", quietly = TRUE)) {
  install.packages("BiocManager")
}
BiocManager::install(version = "3.20", ask = FALSE) #para Bioconductor

if (!require("metabolomicsWorkbenchR")) {
  BiocManager::install("metabolomicsWorkbenchR") #para metabolómica esencial
}
if (!require("MetaboAnalystR")) {
  BiocManager::install("MetaboAnalystR") #para metabolómica avanzada
}

if (!requireNamespace("tidyverse", quietly = TRUE)) {
  install.packages("tidyverse") #para inspección rápida y otras cosas
}
if (!require("VIM")) {
  install.packages("VIM") #para la imputación k-NN
}
if (!requireNamespace("factoextra", quietly = TRUE)) {
  install.packages("factoextra") #para el PCA
}
if (!require("plotly")) {
  install.packages("plotly") #para gráficos interactivos
}
if (!requireNamespace("pheatmap", quietly = TRUE)) {
  install.packages("pheatmap")
}
if (!requireNamespace("caret", quietly = TRUE)) install.packages("caret") #Machinelearning
```

LIBRERÍAS UTILIZADAS

```
library(Biobase)
library(readxl) #para abrir xlsx
library(tidyverse)
library(SummarizedExperiment) #para crear el objeto de tipo SE ideal para omicas
library(VIM) #para la imputación k-NN
library(factoextra) #PCA
library(plotly) #gráficos interactivos
library(pheatmap)#clustering
library(dplyr)
library(ggplot2) #para graficos
library(car)   # Para el test de Levene
library(broom) # Para tidy() si se quiere procesar resultados de test pero no la he usado
#library(caret)  #Para machinelearning-partición estratificada. Cargar cuando toca
#library(mixOmics) #Para crear el modelo de machinelearning. Cargar cuando toca
```

Anexo 1

EXPLORACIÓN DE FICHEROS

```
list.files() #exploro los ficheros del repositorio clonado
list.files(recursive =TRUE) #exploro también los subdirectorios
# Mostrar los ficheros del proyecto que contiene GastricCancer que es el que quiero
files <- list.files(recursive = TRUE, pattern = "GastricCancer.*\\.xlsx$", ignore.case = TRUE)
files <- files[!grepl("~\\$", files)]  # Filtrar archivos temporales de Excel (porque como lo estoy mirando,aparece)
print(files)
excel_sheets("D:/Documentos D/OMICAS/SanMiguel_Diez_Teresa_PEC1v2/GastricCancer_NMR.xlsx") #Inspercciono el excel de interés a ver cuantas hojas tiene
datos <- read_excel("D:/Documentos D/OMICAS/SanMiguel_Diez_Teresa_PEC1v2/GastricCancer_NMR.xlsx", sheet = "Data")
picos <- read_excel("D:/Documentos D/OMICAS/SanMiguel_Diez_Teresa_PEC1v2//GastricCancer_NMR.xlsx", sheet = "Peak")
head(datos) #Tras comprobar que tiene 2, extraigo la información de ambos
head(picos)
glimpse(datos)
glimpse(picos)
```

CREACIÓN DE UN OBJETO SummarizedExperiment

Preparación de la matriz de expresión de datos omicos

```
matriz_datos <- as.matrix(datos %>% dplyr::select(starts_with("M")))
matriz_datos <- t(matriz_datos)  # la transpongo 149 metabolitos × 140 muestras
```

Preparación de la información sobre las muestras incluidas

```
# Datos de las muestras (columnas)
meta_muestras <- datos %>%
```

```
  dplyr::select(Idx, SampleID, SampleType, Class) %>%
  column_to_rownames("SampleID")  # Entre IDx y SampleID, elijo SampleID como rownames
```
Identificación de muestras en la cabecera
```
colnames(matriz_datos) <- rownames(meta_muestras)
```
Asignación de la información de los metabolitos como rowData (identificación e indicios de calidad)
```
rowData <- picos %>%
  column_to_rownames("Name") %>%  # "Name" es M1, M2, etc.
  .[rownames(matriz_datos), ]    # asegurar orden correcto
```
Creación del objeto SummarizedExperiment
```
se <- SummarizedExperiment(
  assays = list(counts = matriz_datos),
  colData = meta_muestras,
  rowData = rowData
)
```
Introducción de metadatos
```
metadata(se)$fuente <- "https://github.com/nutrimetabolomics/metaboData"
metadata(se)$fecha <- Sys.Date()
```
Guardado del objeto SE
```
save(se, file = "SE_GastCancer.Rda")
load("SE_GastCancer.Rda")
se  #
```
LIMPIEZA DE DATOS
```
# Extraer los valores de QC_RSD y Perc_missing desde rowData
rsd <- rowData(se)$QC_RSD
perc_missing <- rowData(se)$Perc_missing

# Crear un vector lógico con los metabolitos que <10% de datos faltantes y <20% de variabilidad entre réplicas
filtro_buena_calidad <- (rsd < 20) & (perc_missing < 10)

# Filtrar el objeto para conservar solo esos metabolitos
se_filtrado <- se[filtro_buena_calidad, ]

# Ver cuántos quedaron
dim(se_filtrado)
save(se, file = "SE_GastCancer_filtrado.Rda")
```
GUARDADO DE LOS TXT QUE CONFORMAN MI SE TRAS EL FILTRADO
Guardado de elementos 1:Guardar la matriz de datos
```
write.table(assay(se), file = "datos_matriz.txt", sep = "\t", quote = FALSE, row.names = TRUE)
```
Guardado de elementos 2: Guardar los metadatos de las muestras (colData)
```
write.table(as.data.frame(colData(se)), file = "metadatos_muestras.txt", sep = "\t", quote = FALSE, row.names = TRUE)
```
Guardado de elementos 3: Guardar metadatos de variables(rowData)
```
write.table(as.data.frame(rowData(se)), file = "metadatos_variables.txt", sep = "\t", quote = FALSE, row.names = TRUE)
```
INSPECCIÓN DE LOS ELEMENTOS QUE CONFORMAN MI SUMMARIZED EXPERIMENT
Para tratar de comprender mejor los “apartados” que se generan al almacenar los datos en este objeto particular,el siguiento fragmento de código recoge las características y qué es cada cosa para que pueda tener claro el nombre con el que se ha almacenado dentro del SE filtrado. *Debo señalar que todavía no he abordado los datos faltantes*.
```
# Cargar el objeto guardado previamente
load("SE_GastCancer_filtrado.Rda")

# === Estructura general ===
cat(" Dimensiones del objeto (metabolitos × muestras):\n")
print(dim(se_filtrado))

cat("\n Metadatos generales:\n")
print(metadata(se_filtrado))

# === Información de muestras ===
cat("\n Primeras muestras (identificadores):\n")
print(head(colnames(se_filtrado)))

cat("\n Información de las muestras (colData):\n")
print(as.data.frame(colData(se_filtrado)) %>% head())

# === Información de metabolitos ===
cat("\n Primeros metabolitos (IDs internos):\n")
print(head(rownames(se_filtrado)))

cat("\n Anotación de metabolitos (Label, %NA, RSD):\n")
rowdata_df <- as.data.frame(rowData(se_filtrado)) %>%
  dplyr::select(Label, Perc_missing, QC_RSD)
```

```
print(head(rowdata_df))

# === Datos cuantitativos ===
cat("\n Matriz de expresión (primeros 5 metabolitos × 5 muestras):\n")
print(assay(se_filtrado)[1:5, 1:5])
```

Anexo 2

Transformación con log10

```
#Filtramos/extraemos las muestras de la matriz del se_filtado, que están en assay
datos_raw <- assay(se_filtrado)  # 52 metabs × 140 muestras
#Transponemos muestras como filas
datos_raw <- t(datos_raw)  # 140 filas (muestras), 52 columnas (metabolitos)
#Hacemos el log10
# Log-transformación (añadiendo +1 para evitar log(0))
datos_log <- log10(datos_raw + 1)
```

Imputación k-NN (k=3)

```
# Imputación por k-NN (k = 3, como en el tutorial)
datos_knn <- kNN(as.data.frame(datos_log), k = 3, imp_var = FALSE)
datos_knn <- as.matrix(datos_knn)  # Para asegurarnos de que sea matriz
```

Análisis PCA

```
res.pca <- prcomp(datos_knn, scale. = TRUE)
```

Visualización PCA

```
library(factoextra)
fviz_pca_ind(res.pca,
             geom.ind = "point",
             habillage = colData(se_filtrado)$SampleType,
             addEllipses = TRUE,
             title = "PCA con log10 + imputación k-NN (QC vs Sample)")
save(datos_knn, res.pca, file = "PCA_log_knn.Rda")
# load("PCA_log_knn.Rda") #para cargarlo cuando siga```
```

Preparo una tabla de datos donde pueda leer la carga en el PCA de cada metabolito.

```
# Paso 1: Extraer las cargas del PCA
loadings <- res.pca$rotation
df_loadings <- as.data.frame(loadings)
df_loadings$M_ID <- rownames(df_loadings)  # Aquí están los M1, M2, ...

# Paso 2: Calcular contribuciones
df_loadings$contrib_PC1 <- df_loadings$PC1^2
df_loadings$contrib_PC2 <- df_loadings$PC2^2
df_loadings$contrib_total <- df_loadings$contrib_PC1 + df_loadings$contrib_PC2

# Paso 3: Crear tabla con nombres de metabolitos
df_nombres <- as.data.frame(rowData(se_filtrado)) %>%
  mutate(M_ID = rownames(.)) %>%
  dplyr::select(M_ID, Nombre = Label)

# Paso 4: Unir ambas tablas por M_ID
df_completo <- left_join(df_loadings, df_nombres, by = "M_ID")

# Paso 5: Ordenar por contribución total
df_ordenado <- df_completo %>%
  arrange(desc(contrib_total)) %>%
  dplyr::select(Nombre, M_ID, PC1, PC2, contrib_PC1, contrib_PC2, contrib_total)

# Mostrar los 10 principales
head(df_ordenado, 10)
```

Y construida la tabla, genero un gráfico estático y un gráfico interactivo similar al del tutorial.

```
df_cargas <- df_ordenado
```

Versión con nombre escrito

```
library(ggplot2)
ggplot(df_cargas, aes(x = PC1, y = PC2, label = Nombre)) +
  geom_point(color = "green", size = 3) +
  geom_text(vjust = -0.5, hjust = 0.5, color = "black", size = 3) +
  labs(title = "PCA Loadings",
       x = "PC1",
       y = "PC2") +
  theme_minimal()
```

Versión con nombre interactivo

Anexo 3

Filtrado de datos Para hacerlo filtramos esos datos que son los que nos interesan. Evitamos que estén los QC y lo tumores benignos.

```r
# Extraer datos del SE filtrado
datos <- assay(se_filtrado) #ojo que con esto sobreescribo "datos" quer era como había llamado inicialmente las cosas
metainfo <- colData(se_filtrado)

# Filtrar solo GC y HE (excluye QC y BN)
idx_gc_he <- metainfo$Class %in% c("GC", "HE")
datos_gc_he <- datos[, idx_gc_he]
metainfo_gc_he <- metainfo[idx_gc_he, ]
```

Comparaciones no paramétricas

```r
# Función para comparar cada metabolito
comparar_metabolito <- function(x, grupo) {
  grupo1 <- x[grupo == "GC"]
  grupo2 <- x[grupo == "HE"]

  # Test de Wilcoxon
  test <- wilcox.test(grupo1, grupo2)

  # Fold change (GC / HE)
  fc <- median(grupo1, na.rm = TRUE) / median(grupo2, na.rm = TRUE)

  data.frame(p_value = test$p.value,
             fold_change = fc)
}

# Aplicar a cada metabolito
grupo <- metainfo_gc_he$Class
resultados <- apply(datos_gc_he, 1, comparar_metabolito, grupo = grupo)

# Convertir a tabla
tabla_resultados <- do.call(rbind, resultados)
tabla_resultados <- as.data.frame(tabla_resultados)
tabla_resultados$metabolito <- rownames(tabla_resultados)

# Ajustar p-valores por FDR
tabla_resultados$adj_p <- p.adjust(tabla_resultados$p_value, method = "fdr")

# Ordenar por significancia
tabla_ordenada <- tabla_resultados %>% arrange(p_value)
head(tabla_ordenada)
# Convertir rowData a data.frame que llamo metanombres
metanombres <- as.data.frame(rowData(se_filtrado))

# Añadir identificador como fila para hacer merge
metanombres$metabolito <- rownames(metanombres)

# Juntar con la tabla ordenada
tabla_con_nombres <- tabla_ordenada %>%
  left_join(metanombres %>% dplyr::select(metabolito, Label), by = "metabolito") %>%
  relocate(Label, .after = metabolito)  # mover el nombre al lado del código
head(tabla_con_nombres)
```

Volcano Plot

```r
# Añadir columnas a la tabla
tabla_con_nombres <- tabla_con_nombres %>%
  mutate(log2FC = log2(fold_change),
         negLogP = -log10(p_value),
         Significativo = case_when(
           adj_p < 0.05 & log2FC > 1 ~ "GC ↑",
           adj_p < 0.05 & log2FC < -1 ~ "HE ↑",
           TRUE ~ "No significativo"
         ))
ggplot(tabla_con_nombres, aes(x = log2FC, y = negLogP, color = Significativo)) +
  geom_point() +
  geom_vline(xintercept = c(-1, 1), linetype = "dotted") +
  geom_hline(yintercept = -log10(0.05), linetype = "dashed", color = "red") +
  scale_color_manual(values = c("GC ↑" = "red", "HE ↑" = "blue", "No significativo" = "grey")) +
  labs(title = "Volcano plot: GC vs HE",
       x = "log2(Fold Change)",
       y = "-log10(p-valor)") +
  theme_minimal()
write.csv(tabla_con_nombres, "Resultados_GC_vs_HE.csv", row.names = FALSE)
```

```r
# Crear función que evalúe por metabolito
evaluar_metabolito <- function(x, grupo) {
  # Separar los valores por grupo
  valores_gc <- x[grupo == "GC"]
  valores_he <- x[grupo == "HE"]

  # Calcular medias
  media_gc <- mean(valores_gc, na.rm = TRUE)
  media_he <- mean(valores_he, na.rm = TRUE)

  # IC 95% (asumiendo t-distribución)
  n_gc <- sum(!is.na(valores_gc))
  n_he <- sum(!is.na(valores_he))
  se_gc <- sd(valores_gc, na.rm = TRUE) / sqrt(n_gc)
  se_he <- sd(valores_he, na.rm = TRUE) / sqrt(n_he)
  ic95_gc <- c(media_gc - 1.96 * se_gc, media_gc + 1.96 * se_gc)
  ic95_he <- c(media_he - 1.96 * se_he, media_he + 1.96 * se_he)

  # Shapiro para normalidad
  shapiro_gc <- if (length(valores_gc) >= 3) shapiro.test(valores_gc)$p.value else NA
  shapiro_he <- if (length(valores_he) >= 3) shapiro.test(valores_he)$p.value else NA

  # Levene test
  df_tmp <- data.frame(valores = c(valores_gc, valores_he),
                       grupo = rep(c("GC", "HE"), times = c(length(valores_gc), length(valores_he))))
  p_levene <- if (nrow(df_tmp) >= 4) leveneTest(valores ~ grupo, data = df_tmp)$`Pr(>F)`[1] else NA

  # Indicadores lógicos
  normalidad_gc <- !is.na(shapiro_gc) && shapiro_gc >= 0.05
  normalidad_he <- !is.na(shapiro_he) && shapiro_he >= 0.05
  homogeneidad <- !is.na(p_levene) && p_levene >= 0.05

  # Resultado
  data.frame(
    media_gc = media_gc,
    ic95_gc_inf = ic95_gc[1],
    ic95_gc_sup = ic95_gc[2],
    media_he = media_he,
    ic95_he_inf = ic95_he[1],
    ic95_he_sup = ic95_he[2],
    p_shapiro_gc = shapiro_gc,
    p_shapiro_he = shapiro_he,
    p_levene = p_levene,
    cumple_shapiro_gc = normalidad_gc,
    cumple_shapiro_he = normalidad_he,
    cumple_levene = homogeneidad
  )
}

# Aplicar a todos los metabolitos
resultados_supuestos <- apply(datos_gc_he, 1, evaluar_metabolito, grupo = grupo)

# Convertir lista en tabla
tabla_supuestos <- bind_rows(resultados_supuestos, .id = "metabolito")

# Añadir nombres de metabolitos
tabla_supuestos <- tabla_supuestos %>%
  left_join(metanombres %>% dplyr::select(metabolito, Label), by = "metabolito") %>%
  relocate(Label, .after = metabolito)

# Ver los primeros resultados
head(tabla_supuestos)
```

Anexo 4

Machine Learning

```r
library(caret)  #Para machinelearning-partición estratificada. Cargar cuando toca
library(mixOmics) #Para crear el modelo de machinelearning. Cargar cuando toca
```

1. División de datos entrenamiento/prueba

```r
if (!requireNamespace("caret", quietly = TRUE)) install.packages("caret")
library(caret)
```

```r
# Extraer los datos de la matriz de expresión y de la metainformación
datos <- assay(se_filtrado)
metainfo <- colData(se_filtrado)

# Filtrar GC y HE
idx_gc_he <- metainfo$Class %in% c("GC", "HE")
datos_gc_he <- datos[, idx_gc_he]
metainfo_gc_he <- metainfo[idx_gc_he, ]
# Transponer: filas = muestras, columnas = metabolitos
datos_gc_he_t <- t(datos_gc_he)

# Log10 (+1 para evitar log(0))
datos_log <- log10(datos_gc_he_t + 1)

# Imputar con kNN
datos_knn <- kNN(as.data.frame(datos_log), k = 3, imp_var = FALSE)
datos_knn <- as.matrix(datos_knn)
set.seed(123)  # reproducibilidad

# Dividir
particion <- createDataPartition(metainfo_gc_he$Class, p = 0.75, list = FALSE)

# Crear subconjuntos
datos_train <- datos_knn[particion, ]
datos_test  <- datos_knn[-particion, ]

metainfo_train <- metainfo_gc_he[particion, ]
metainfo_test  <- metainfo_gc_he[-particion, ]
# Escalar train
datostrain_escalado <- scale(datos_train)

# Guardar medias y SD
media_train <- attr(datostrain_escalado, "scaled:center")
sd_train <- attr(datostrain_escalado, "scaled:scale")

# Escalar test usando media y SD del train
datostest_escalado <- scale(datos_test, center = media_train, scale = sd_train)
library(mixOmics)

# Convertir clases a factor
Y_train <- factor(metainfo_train$Class, levels = c("HE", "GC"))

# Entrenar modelo PLS-DA
modelo_plsda_final <- plsda(datostrain_escalado, Y_train, ncomp = 2)
```

3. Evaluar la validez del modelo

vALIDACIÓN CRUZADA: El método más estándar es perf() de mixOmics, que calcula error de clasificación, pero también R2 y Q2 Validación cruzada (para R2/Q2 y tasa de error).

```r
set.seed(123)
perf_plsda <- perf(modelo_plsda_final, validation = "Mfold", folds = 5, nrepeat = 10, progressBar = TRUE)
plot(perf_plsda)  # ver tasa de error por número de componentes
```

Extraer métricas

```r
# 1. Error global usando 2 componentes y distancia por centroides
perf_plsda$error.rate$overall["comp2", "centroids.dist"]

# 2. Error por clase usando 2 componentes y distancia por centroides
perf_plsda$error.rate.class$centroids.dist[, "comp2"]

# 3. Número óptimo de componentes según error global (distancia centroides)
perf_plsda$choice.ncomp["overall", "centroids.dist"]
```

4. Proyectar las variables latentes

Proyección de individuos (samples)

Esto muestra cómo se separan las clases GC y HE en el espacio de los componentes latentes.

```r
plotIndiv(modelo_plsda_final,
          comp = c(1, 2),
          group = Y_train,
          legend = TRUE,
          ellipse = TRUE,
          title = "Proyección PLS-DA (GC vs HE)")
```

Carga de variables (metabolitos)

```
plotVar(modelo_plsda_final,
        comp = c(1, 2),
        cex = 2,
        title = "Metabolitos más relevantes (carga)")
```

Los metabolitos más alejados del centro tienen más peso.

5. Determinar las variables más importantes

Obtener los VIP scores

```
vip_scores <- vip(modelo_plsda_final)
head(vip_scores)

# Convertimos los VIPs del componente 1 a data frame
vip_df <- data.frame(Metabolito = rownames(vip_scores),
                     VIP = vip_scores[, 1])

# Ordenamos por importancia
vip_df <- vip_df[order(-vip_df$VIP), ]

# Gráfico de barras de los VIPs
ggplot(vip_df[1:10, ], aes(x = reorder(Metabolito, VIP), y = VIP)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  coord_flip() +
  labs(title = "Top 10 metabolitos más importantes (VIP- Comp 1)",
       x = "Metabolito", y = "VIP Score") +
  theme_minimal()
```

6. Evaluar el modelo con los datos de prueba

Predecir en set de test qué tal funciona

```
# Vector de clases reales
Y_test <- factor(metainfo_test$Class, levels = c("HE", "GC"))

# Predecir
predicciones <- predict(modelo_plsda_final, newdata = datostest_escalado)
clases_predichas <- predicciones$class$max.dist[, 2]

# Matriz de confusión
confusion <- table(Real = Y_test, Predicho = clases_predichas)
print(confusion)
```

El modelo acierta al clasificar en 8/10 en los sanos y 9/10 en los pacientes con cáncer gástrico.

```
# HE = clase positiva
sensibilidad <- confusion["HE", "HE"] / sum(confusion["HE", ])
especificidad <- confusion["GC", "GC"] / sum(confusion["GC", ])
sensibilidad
especificidad
```