brain\_trabajo

Elena Maroto Rica

2025-01-10

Table of Contents

[1. Introducción 2](#_Toc189420097)

[1.1. Datos bajados de GDC 4](#_Toc189420098)

[1.2. Algoritmos escogidos 4](#_Toc189420099)

[2. Preparación de los datos 5](#_Toc189420100)

[2.1. Cargamos los datos completos 5](#_Toc189420101)

[2.1. Tabla con número de genes para distintas combinaciones de LFC y COV. 8](#_Toc189420102)

[3. Selección de características y clasificación con KnowSeq 11](#_Toc189420103)

[3.1. Clasificador k-NN 14](#_Toc189420104)

[3.1.1. Selección de características con FSRankingMRMR: 14](#_Toc189420105)

[3.1.2. Selección de características con FSRankingRF: 16](#_Toc189420106)

[3.1.3. Selección de variables con FSRankingDA: 19](#_Toc189420107)

[3.1.4. Selección de variables con FSRanking\_MB: 20](#_Toc189420108)

[3.1.5. Conclusiones 22](#_Toc189420109)

[3.2. Clasificador RandomForest 22](#_Toc189420110)

[3.2.1. Selección de variables con FSRankingMRMR 22](#_Toc189420111)

[3.2.2. Selección de variables con FSRankingRF 24](#_Toc189420112)

[3.2.3. Selección de variables con FSRankingDA 26](#_Toc189420113)

[3.2.4. Selección de variables con FSRankingMB 28](#_Toc189420114)

[3.2.5. Conclusiones 30](#_Toc189420115)

[3.3. Clasificador SVM 30](#_Toc189420116)

[3.3.1. Selección de variables con FSRankingMRMR 37](#_Toc189420117)

[3.3.2. Selección de variables con FSRankingRF: 39](#_Toc189420118)

[3.3.3. Selección de variables con FSRankingDA: 41](#_Toc189420119)

[3.3.4. Selección de variables con FSRanking\_MB: 43](#_Toc189420120)

[3.3.5. Conclusiones 45](#_Toc189420121)

[3.4. Clasificador Regresión Logística 45](#_Toc189420122)

[3.4.1. Selección de variables con FSRankingMRMR 53](#_Toc189420123)

[3.4.2. Selección de variables con FSRankingRF 55](#_Toc189420124)

[3.4.3. Selección de variables con FSRankingDA 57](#_Toc189420125)

[3.4.3. Selección de variables con FSRanking\_MB 59](#_Toc189420126)

[3.4.5. Conclusiones 62](#_Toc189420127)

[4. Resultados bajo el algoritmo de selección y clasificador escogidos en 5CV 62](#_Toc189420128)

[4.1. Resultado de la huella final escogida y clasificador en 5 CV, matriz de confusión final del dataset. 71](#_Toc189420129)

[5. Enriquecimiento de los genes de la huella escogida 77](#_Toc189420130)

# 1. Introducción

El cáncer de cerebro es una enfermedad que presenta una incidencia variable en la población a nivel mundial. Así, comenzaremos este proyecto, analizando tres tipos de cancer de cerebro en diferentes países y grupos de población.

El cerebro está compuesto por una gran variedad de células, entre las que se encuentran las neuronas, encargadas de transmitir señales eléctricas esenciales para las funciones cerebrales, y los **astrocitos**, que brindan soporte y estructura para su adecuado funcionamiento. Así, los **astrocitomas** son tumores que se desarrollan a partir de los astrocitos y representan el tipo más frecuente de tumor cerebral en adultos. En Estados Unidos, se detectan aproximadamente 15.000 nuevos casos de astrocitoma cada año y según la Asociación Americana de Cirujanos Neurológicos (AANS, <https://www.aans.org/patients/conditions-treatments/astrocytoma-tumors/>), estos tumores afectan a los hombres con una ligera mayor frecuencia que a las mujeres, con una proporción de 1,3 a 1. Algunos factores de riesgo para la aparición de este tipo de cancer son las mutaciones en el ADN transmitidos de generación en generación y la exposición a radiaciones ionizantes.

Los astocitomas más benignos (Grado 1) destacan entre la población más joven, como por ejemplo el astrocitoma pilocítico, frecuente en el cerebelo o el astrocitoma subependimario de células gigantes (SEGA) que crece característicamente dentro de los ventrículos. Sin embargo, el resto de astrocitomas afectan a pacientes mayores de 40 años, y la peligrosidad se acentúa con la edad. De hecho, el astrocitoma más peligro, rápido y común es el glioblastoma el cual, en la mayoría de las ocasiones se origina directamente como un tumor de Grado 4.

En segundo lugar, conozcamos el **oligodendroglioma**, un tipo de tumor primario del sistema nervioso central (SNC), es decir, que se origina o en el cerebro o en la médula espinal. Este tumor se llama oligodendroglioma porque las células son similares a los oligodendrocitos, cuya función principal es la de producir y mantener la *mielina*, una sustancia grasa que recubre y aísla las fibras nerviosas (*axones*), permitiendo una transmisión rápida y eficiente de los impulsos eléctricos entre las células nerviosas.

Con los datos recogidos del portal GDC, se han agrupado dos tipos de oligodendrogliomas para formar esta clase: el anaplasmático, es crecieminto rápido, maligno y por tanto, de Grado 3, y el oligodendroglioma NOS (*Not Other Specified*, término que se utiliza en el diagnóstico cuando no se dispone de suficiente información molecular para clasificar el tumor con precisión o si el estudio molecular directamente no se ha realizado). Según el Instituto nacional del Cáncer (<https://www.cancer.gov/rare-brain-spine-tumor/espanol/tumores/oligodendroglioma>), el oligodendroglioma suele diagnosticarse con mayor frecuencia en personas de entre 35 y 44 años, aunque puede manifestarse en cualquier etapa de la vida. Tiende a afectar ligeramente más a los hombres que a las mujeres y es raro en la población infantil. Este tipo de tumor es más habitual en individuos de ascendencia caucásica y en aquellos que no tienen origen hispano.

Por otro lado, según el Servicio Anadluz de Salud (<https://www.sspa.juntadeandalucia.es/servicioandaluzdesalud/hrs3/fileadmin/user_upload/area_medica/comite_tumores/guia_gliomas_alto_grado.pdf>) los oligodendrogliomas anaplásicos tienen una tasa de incidencia anual de 0,11 casos por cada 100.000 personas. Representan el 0,5% de todos los tumores cerebrales primarios y el 33% de los tumores oligodendrogliales. Este tipo de tumor se presenta principalmente en adultos, con una edad media de diagnóstico de 49 años, y afecta ligeramente más a los hombres. Además, su pronóstico es relativamente más favorable en comparación con los tumores de origen astrocítico.

Por último, tratemos los **gliomas mixtos**. Este tumor se origina a partir de más de un tipo de célula, por lo general, astrocitos y oligodendrocitos; y por tanto, tienden a adoptar las características del tipo de glioma que es más abundante en el tumor. Según la Universidad de Columbia, específicamente de su departamento de neurocirugía en Nueva York (<https://www.neurosurgery.columbia.edu/patient-care/conditions/mixed-gliomas>) los gliomas mixtos representan aproximadamente el 1% de los tumores cerebrales y entre el 5 y el 10% de los gliomas. Pueden ser de grado 2 (de bajo crecimiento) o grado 3 (más agresivos) y se diagnostican en adultos comprendidos de entre 35 y 50 años, siendo poco frecuentes en niños.

Finalmente, según la Sociedad Española de Neurología (SEN, <https://www.sen.es/saladeprensa/pdf/Link406.pdf>) se estima que actualmente en España hay cerca de 20.000 personas con algún tipo de tumor cerebral, ya sea primario, cuando se desarrolla directamente en el cerebro, o metastásico, cuando se origina en otra parte del cuerpo y se extiende hasta el cerebro. En los adultos, los tumores cerebrales representan aproximadamente el 2% de todos los casos de cáncer, mientras que en los niños suponen el 15%, siendo el segundo tipo de cáncer más común en la infancia, después de la leucemia. A pesar de no ser un cáncer muy frecuente, algunos hospitales, como el Hospital Universitario Virgen de las Nievas de Granada, están instalando equipos de última tecnología para tratar tumores y otras patología generales con el objetivo de *“obtener un resultado óptimo en el control tumoral y la minimización de efectos secundarios, al disminuir la dosis en los tejidos sanos cercanos a la lesión”* (<https://www.huvn.es/noticias/el-hospital-universitario-virgen-de-las-nieves-de-granada-instala-un-equipo-de-ultima-tecnologia-para-tratar-tumores-y-otras-patologias-cerebrales>).

## 1.1. Datos bajados de GDC

Para cada problema tratado, se presentará una tabla con los datos obtenidos de la plataforma Genomic Data Commons (GDC), describiendo el número de clases y muestras disponibles por cada clase. A partir de estos datos, se seleccionará un conjunto aleatorio de entrenamiento y otro de prueba, los cuales serán descritos en detalle mediante tablas que reflejen la distribución de las muestras.

RNA-Seq: Class astrocytoma mixed\_glioma oligodendroglioma 197 135 201

miRNA:

Por otro lado, partiendo de la matriz de expresión final y las respectivas etiquetas, se utiliza la función cvGenStratified para dividir los datos en un conjunto de entrenamiento y un conjunto de prueba, asegurando que cada partición tenga la misma proporción de ejemplos de cada clase que el conjunto de datos completo. A continuación, la función cvGenStratified genera 5 particiones (nfolds = 5), se elige la primera de ellas, por ejemplo, y se utiliza esta información para crear índices que separan los datos en un conjunto de prueba (indexTest) y uno de entrenamiento (indexTrn). Los datos de entrenamiento se asignan a la variable XTrn y las etiquetas correspondientes a YTrn, mientras que los datos de prueba se asignan a XTest y sus etiquetas a YTest.

RNA-Seq: XTest 26.079 observaciones (genes), 104 columnas. Por tanto, YTest es un vector de 104 etiquetas. XTrn 6.079 observaciones (genes), 413 columnas. Por tanto, YTrn es un vector de 413 etiquetas.

miRNA:

## 1.2. Algoritmos escogidos

En esta sección, se explicarán las características del algoritmo de clasificación y del algoritmo de selección de características elegidos. Se detallará su funcionamiento, las estrategias de optimización aplicadas y se discutirá cómo estos algoritmos se adaptan a los problemas planteados en el estudio.

* El algoritmo de clasificación elegido ha sido **Regresión Logística multinomial con penalización**, es decir, el clasificador de R multinom. Basándonos en el código fuente de las funciones svm\_trn y svm\_test de KnowSeq, se ha implementado dicho algoritmo. Por un lado, la función logistic\_trn2 es capaz de seleccionar el mejor modelo para la clasificación, haciendo uso del conjunto de entrenamiento y una validación cruzada de 5 particiones de este. Concretamente, se elegirá el mejor valor del hiperparámetro de regularización decay que controla la penalización aplicada a los coeficientes del modelo ayudando a evitar el sobreajuste al reducir los coeficientes menos importantes hacia cero. Por otro lado, la parte principal del código de la función logistic\_test\_multiclase es el for, en cual se irán eleccionando genes, con el mejor parámetro *decay* seleccionado en la fase de entrenamiento, de la siguiente forma: en la primera iteración se ecoge el primer gen del ranking, en la siguiente los dos primeros… Así hasta llegar a 10 genes seleccionados. Recordemos que, al igual que en los demás algoritmos, se guardan las métricas de cada iteración del accuracy y F1 con el objetivo de evaluar la clasificación.
* El algoritmo de selección de características elegido ha sido **Markov Blanket**

# 2. Preparación de los datos

En esta sección, tras preparar los datos, se tomarán distintos valores de LFC (Log Fold Change) y COV para determinar el número óptimo de genes diferencialmente expresados (DE), manteniendo un rango aproximado entre 50 y 300 genes. Se presentará una tabla comparativa con el número de genes obtenidos en función de diferentes combinaciones de estos parámetros.

## 2.1. Cargamos los datos completos

Tras cargar la librería KnowSeq y las funciones proporcionadas, se identifica el directorio con los ficheros de los pacientes y convertimos los star counts a ficheros de counts simples:

load(file='samples')

Para adaptar los counts a nuestro porblema, en primer lugar, se repite el proceso de la creación de counts simples para la clase **astrocitoma** y se etiquetan como tal en el data frame *samples* original:

load(file='samples\_astrocytoma')  
samples$Sample.Type[samples$File.ID %in% samples\_astrocytoma$File.ID] <- 'astrocytoma'

En segundo lugar, se crean los *samples* de la clase **glioma mixto** y se reemplazan las etiquetas:

load(file='samples\_mixed')  
samples$Sample.Type[samples$File.ID %in% samples\_mixed$File.ID] <- 'mixed\_glioma'

Finalmente, se repite el proceso para la clase **oligodendroglioma**:

load(file='samples\_oligodendroglioma')  
samples$Sample.Type[samples$File.ID %in% samples\_oligodendroglioma$File.ID] <- 'oligodendroglioma'

Una vez etiquetado correctamente el dataframe *sample* con los ficheros counts - Creamos las variables necesarias para el dataframe con los datos de los ficheros counts y se identifica la clase de cada muestra:

Run <- samples$File.Name  
Path <- rep("brain/counts",nrow(samples))  
Class <- samples$Sample.Type

* Se imprimen el número de muetras por clase, donde se pueden ver que estan bien balanceadas, y se guardan en un fichero csv:

table(Class)

## Class  
## astrocytoma mixed\_glioma oligodendroglioma   
## 197 135 201

data.info <- data.frame(Run = Run, Path = Path, Class = Class)  
write.csv(file = "data\_info.csv", x = data.info)

* Se cargan y se aunan los ficheros counts:

countsInfo <- countsToMatrix("data\_info.csv", extension = "")  
#Guardar fichero de counts por agilizar futuras ejecuciones  
save(countsInfo, file='CountsInfo')

load(file = 'countsInfo')

* Exportamos tanto la matriz de datos como los labels a nuevas variables:

countsMatrix <- countsInfo$countsMatrix  
labels <- countsInfo$labels

* Consultamos los Gene Symbols y el GC content de cada gen y los valores de expresion usando la matriz de count y la anotación previamente adquirida:

myAnnotation <- getGenesAnnotation(rownames(countsMatrix))

## Getting annotation of the Homo Sapiens...  
## Using reference genome 38.

geneExprMatrix <- calculateGeneExpressionValues(countsMatrix, annotation = myAnnotation)  
# Eliminamos filas sin anotacion de gen  
geneExprMatrix <- geneExprMatrix[!is.na(rownames(geneExprMatrix)),]  
# Guardar fichero de Expresión de Gen por agilizar futuras ejecuciones  
save(geneExprMatrix, file='geneExprMatrix')

load(file = 'geneExprMatrix')

* Realizamos el analisis de calidad y se seleccionan las muestras que pasen por el filtro (no outliers):

QAResults <- RNAseqQA(geneExprMatrix, toRemoval = TRUE,   
 toPNG=FALSE, toPDF=FALSE)  
save('QAResults', file = 'QAResults')

# Se cargan para su uso:  
load('QAResults')  
# Seleccionar muestras que pasen el filtro (no outliers)  
qualityMatrix <- QAResults$matrix  
qualityLabels <- labels[-which(colnames(geneExprMatrix) %in% QAResults$outliers)]

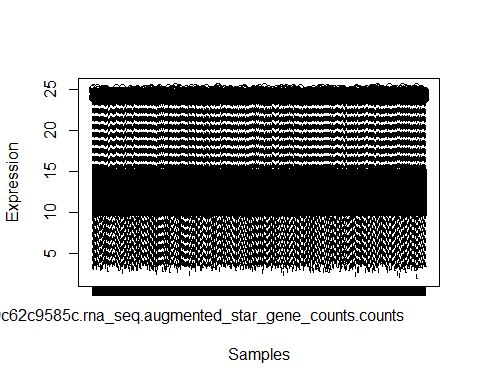
* Creamos el modelo SVA de variables surrogadas para tratar el efecto batch

# Creamos el modelo SVA de variables surrogadas para tratar el efecto batch  
batchMatrix <- batchEffectRemoval(qualityMatrix, qualityLabels, method = "sva")  
  
MATRIZ <- batchMatrix  
LABELS <- qualityLabels  
# Guardar datos de matriz de Expresión Final y Labels por agilizar futuras ejecuciones  
save(MATRIZ, file = 'MATRIZ')  
save(LABELS, file = 'LABELS')

load(file = 'MATRIZ')  
load(file = 'LABELS')  
  
#rownames(MATRIZ)<- make.names(rownames(MATRIZ))

Así, en el siguiente gráfico se muestran un boxplot ordenado que representa cómo los datos (en este caso, los valores en batchMatrix) se distribuyen para cada grupo definido por las etiquetas qualityLabels. Esto es útil para ver si las muestras se agrupan bien por condición biológica y si el efecto batch se ha corregido adecuadamente:

#png('b.png', width = 800, height = 700)  
dataPlot(MATRIZ, qualityLabels, mode = "orderedBoxplot")



# 2.1. Tabla con número de genes para distintas combinaciones de LFC y COV.

* Primero, se hace una subdivisión inicial Trn-Test y preparar CV para pruebas definitivas y escoger huella final:

require(CORElearn)  
set.seed(19)  
nfolds<- 5  
folds<-cvGenStratified(LABELS, nfolds)  
  
nData<-dim(MATRIZ)[1]  
indexTest<-which(folds ==1) # para ejecuciones no CV  
indexTrn<-which(folds != 1) # para ejecuciones no CV  
XTrn<-MATRIZ[, indexTrn]  
XTest<-MATRIZ[,indexTest]  
YTrn<-LABELS[indexTrn]  
YTest<-LABELS[indexTest]

* Del TRAIN se propone extraer genes diferencialmente expresados (DE): vamos a jugar con diferentes valores de COV y LFC:

# Definir los valores de LFC y COV a probar  
LFC\_values <- c(0.1, 0.15, 0.25, 0.75, 1, 1.5, 2, 2.25, 2.75, 3, 3.5)   
COV\_values <- c(1, 2, 3)  
  
  
# Crear una lista vacía para almacenar los resultados  
results <- data.frame(LFC = numeric(0), COV = numeric(0), NumGenes = numeric(0))  
  
# Función para extraer los genes diferencialmente expresados con LFC y COV específicos  
for (LFC in LFC\_values) {  
 for (COV in COV\_values) {  
 tryCatch({  
 # Ejecutar la función DEGsExtraction para esta combinación de LFC y COV  
 DEGsInfo <- DEGsExtraction(XTrn, YTrn, lfc = LFC, cov = COV, pvalue=0.001)  
   
 # Contar el número de genes DEGs (filas de DEGs\_Matrix)  
 num\_genes <- nrow(DEGsInfo$DEG\_Results$DEGs\_Matrix)  
   
 # Agregar los resultados a la tabla  
 results <- rbind(results, data.frame(LFC = LFC, COV = COV, NumGenes = num\_genes))  
 }, error = function(e) {  
 # Si ocurre un error (por ejemplo, si no hay genes DEGs), asignar 0 genes  
 results <- rbind(results, data.frame(LFC = LFC, COV = COV, NumGenes = 0))  
   
 # Mostrar un mensaje informativo sobre el error  
 message(paste("Error con lfc =", LFC, "y cov =", COV, ": ", e$message))  
 })  
 }  
}  
  
results  
save(results, file='results')

load('results')  
results

## LFC COV NumGenes  
## 1 0.10 1 338  
## 2 0.10 2 2  
## 3 0.15 1 197  
## 4 0.25 1 67

* Como lo conveniente es que salga un número de genes entre 50-300 aproximadamente, filtramos los resultados obtenido anteriormente:

resultado\_filtrados <- results[results$NumGenes >= 50 & results$NumGenes <= 300, ]  
resultado\_filtrados

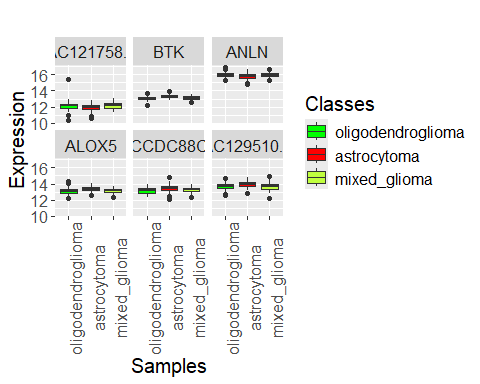
## LFC COV NumGenes  
## 3 0.15 1 197  
## 4 0.25 1 67

* Boxplot y Heatmap de expresion de los 6 DEGs con número de genes entre 50-300: Observando lo anterior elegimos, por ejemplo, lfc = 0.15 y cov = 1 para obtener la DEGsMatrix.

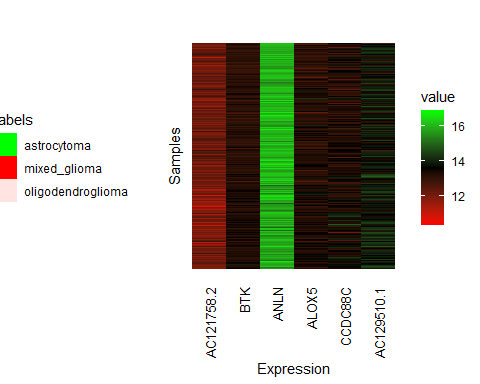
DEGsInfo <- DEGsExtraction(XTrn, YTrn, lfc = 0.15, cov = 1, pvalue=0.001)

## More than two classes detected, applying limma multiclass  
## Contrasts: astrocytoma-mixed\_glioma  
## Contrasts: astrocytoma-oligodendroglioma  
## Contrasts: mixed\_glioma-oligodendroglioma  
## 153 106 154

DEGsMatrix <- DEGsInfo$DEG\_Results$DEGs\_Matrix  
#png('2b.png', width = 800, height = 600)  
dataPlot(DEGsMatrix[1:6,], YTrn, mode = "genesBoxplot")



#png('3b.png', width = 800, height = 600)  
dataPlot(DEGsMatrix[1:6,], YTrn, mode = "heatmap")



Las clases se diferencian correctamente a simple vista, es decir, no coinciden las medias o las medianas, por ejemplo.

# 3. Selección de características y clasificación con KnowSeq

A continuación, se analizarán y compararán los resultados de los tres algoritmos de selección de características y los tres clasificadores disponibles en la herramienta KnowSeq y el seleccionado para este trabajo: k-Nearest Neighbors (k-NN), Random Forest, Support Vector Machines (SVM) y Regresión Logística multiclase. Para cada clasificador, se evaluará el rendimiento con los cuatro algoritmos de selección (mRMR, RF, DA y Markov Blanket), generando gráficos y tablas de confusión que muestran el desempeño de estas combinaciones.

Para ello, se preparan, en primer lugar, tanto la matriz como los labels:

MLMatrix <- t(DEGsMatrix)  
MLLabels <- YTrn

Los cuatro rankings para la selección de características son:

FSRankingDA <- featureSelection(MLMatrix, MLLabels, mode = "da", vars\_selected = colnames(MLMatrix), disease='BrainCancer')  
save(FSRankingDA, file='FSRankingDA')

FSRankingMRMR <- featureSelection(MLMatrix, MLLabels, mode = "mrmr", vars\_selected = colnames(MLMatrix))  
FSRankingRF <- featureSelection(MLMatrix, MLLabels, mode = "rf", vars\_selected = colnames(MLMatrix))  
load('FSRankingDA')

Implementación del algoritmo de selección de características Markov Blanket:

library(infotheo) # Para calcular información mutua  
  
# Función para calcular la información mutua entre dos variables  
calculate\_mutual\_info <- function(X, Y) {  
 # Discretizar las variables y calcular la información mutua  
 return(mutinformation(discretize(X), discretize(Y)))  
}  
  
# Algoritmo hacia atrás Markov Blanket  
featureSelection\_MB <- function(data, target) {  
 # Asegurarse de que 'data' sea una matriz y 'target' sea un vector  
 if (!is.matrix(data)) {  
 stop("La variable 'data' debe ser una matriz.")  
 }  
 if (!is.vector(target)) {  
 stop("La variable 'target' debe ser un vector.")  
 }  
   
 # Inicializar el conjunto de características (columnas de la matriz)  
 XG <- colnames(data) # Conjunto de todas las características  
   
 # Asegurar que el vector 'target' tenga la misma longitud que las filas de 'data'  
 if (length(target) != nrow(data)) {  
 stop("El tamaño de 'target' no coincide con el número de filas de 'data'.")  
 }  
  
 # Lista para almacenar las pérdidas de las características  
 feature\_loss <- numeric(length(XG)) # Para almacenar las pérdidas por eliminar cada característica  
   
 # Iterar hasta que el conjunto de características sea pequeño  
 while (length(XG) > 0) {  
 # Inicializar el vector de pérdidas  
 loss\_values <- numeric(length(XG))   
   
 # Calcular las pérdidas para cada característica  
 for (i in 1:length(XG)) {  
 Xj <- XG[i]  
   
 # Obtener el marco de Markov Mj (las características más informativas respecto a Xj)  
 Mj <- setdiff(XG, Xj) # Inicializamos Mj sin Xj (esto es un proxy de MB)  
   
 # Verificar si hay suficientes columnas en Mj para realizar el cálculo de información mutua  
 if (length(Mj) == 0) {  
 next  
 }  
   
 # Calcular la información mutua I({Mj ∪ Xj}, Y)  
 IMjXj\_Y <- tryCatch({  
 calculate\_mutual\_info(data[, c(Mj, Xj)], target)  
 }, error = function(e) {  
 NA # Si ocurre un error, devolver NA  
 })  
   
 # Calcular la información mutua I(Mj, Y)  
 IMj\_Y <- tryCatch({  
 calculate\_mutual\_info(data[, Mj], target)  
 }, error = function(e) {  
 NA # Si ocurre un error, devolver NA  
 })  
   
 # Si alguna de las informaciones mutuas no es válida (NA), no calcular la pérdida  
 if (is.na(IMjXj\_Y) | is.na(IMj\_Y)) {  
 loss\_values[i] <- NA  
 } else {  
 # Calcular la pérdida para Xj  
 loss\_values[i] <- IMjXj\_Y - IMj\_Y  
 }  
 }  
   
 # Si todas las pérdidas son NA, detener el proceso  
 if (all(is.na(loss\_values))) {  
 cat("Todas las pérdidas son NA. Deteniendo el algoritmo.\n")  
 break  
 }  
   
 # Filtrar las pérdidas no-NA y seleccionar la característica con la menor pérdida  
 valid\_loss\_values <- loss\_values[is.finite(loss\_values)]  
   
 # Si no hay valores válidos para calcular la pérdida, detener  
 if (length(valid\_loss\_values) == 0) {  
 cat("No hay valores válidos para calcular las pérdidas. Deteniendo el algoritmo.\n")  
 break  
 }  
   
 # Seleccionar la característica con la menor pérdida  
 best\_feature\_to\_remove <- XG[which.min(valid\_loss\_values)]  
   
 # Almacenar la pérdida y la característica eliminada  
 feature\_loss[which(XG == best\_feature\_to\_remove)] <- loss\_values[which(XG == best\_feature\_to\_remove)]  
   
 # Eliminar la característica seleccionada del conjunto  
 XG <- setdiff(XG, best\_feature\_to\_remove)  
   
 cat("Eliminada característica:", best\_feature\_to\_remove, "\n")  
 }  
   
 # Crear un data frame con el nombre de la característica y su pérdida  
 feature\_ranking <- data.frame(  
 Feature = colnames(data),  
 Loss = feature\_loss  
 )  
   
 # Ordenar el ranking de características por la pérdida (menor a mayor)  
 feature\_ranking <- feature\_ranking[order(feature\_ranking$Loss), ]  
   
 # Devolver el ranking de las características ordenado  
   
 return(feature\_ranking)  
}

#Feature selection con Markov Blanket:  
FSRanking\_MB <- featureSelection\_MB(MLMatrix, MLLabels)$Feature  
save(FSRanking\_MB, file='FSRanking\_MB')

load('FSRanking\_MB')

## 3.1. Clasificador k-NN

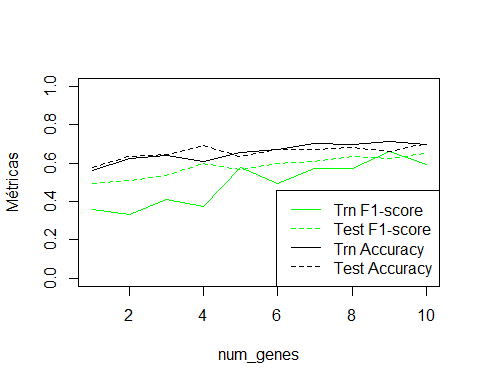
Calculados los rankings de carcaterísticas, comencemos con el clasificador k-NN.

### 3.1.1. Selección de características con FSRankingMRMR:

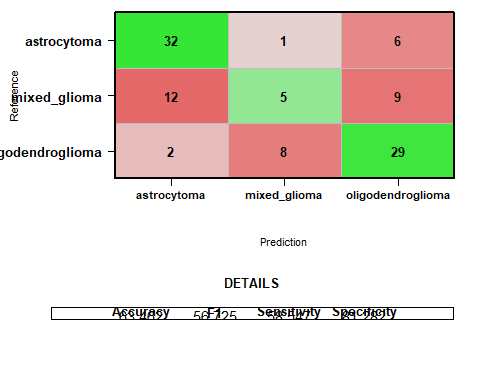
Esta sección se centra en la evaluación del rendimiento del modelo de clasificación basado en los k-Vecinos más cercanos (KNN) utilizando una selección de características mediante el método mRMR (Minimum Redundancy Maximum Relevance). Primero, se entrena el modelo, knn\_trn\_mrmr, utilizando la función knn\_trn con un subconjunto de 10 biomarcadores seleccionados, FSRankingMRMR[1:10], evaluando su desempeño en términos de puntuación F1 y precisión. Luego, se prueba el modelo knn\_test\_mrmr con la función knn\_test, con datos de la partición test para comparar los resultados de entrenamiento y prueba. Finalmente, se visualizan las métricas mediante gráficos comparativos para analizar la efectividad del modelo.

# Evaluar biomarcadores en TRN. Es multiclase, mejor ver accuracy y F1-Score  
knn\_trn\_mrmr<- knn\_trn(MLMatrix, MLLabels, vars\_selected = names(FSRankingMRMR[1:10]))   
knn\_results\_mrmr <- rbind(knn\_trn\_mrmr$F1Info$meanF1[1:10],knn\_trn\_mrmr$accuracyInfo$meanAccuracy)

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score  
knn\_test\_mrmr<-knn\_test(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected = names(FSRankingMRMR[1:10]), bestK=knn\_trn\_mrmr$bestK)  
knn\_results\_mrmr\_test <- rbind(knn\_trn\_mrmr$F1Info$meanF1[1:10], knn\_test\_mrmr$f1Vector, knn\_trn\_mrmr$accuracyInfo$meanAccuracy, knn\_test\_mrmr$accVector)



Cuando num\_genes = 5, se estabiliza las curvas alcanzando el 60% de acierto (accuracy) y de F1 para la evaluación del conjunto train y test. Comprobamos esta información con la matriz de confusión para el conjunto test:

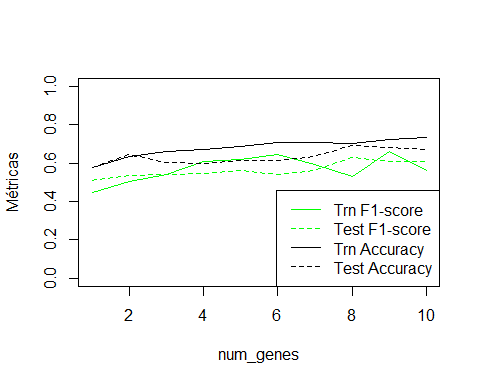


### 3.1.2. Selección de características con FSRankingRF:

Utilicemos ahora las 10 primeras características seleccionadas con Random Forest para entrenar el modelo

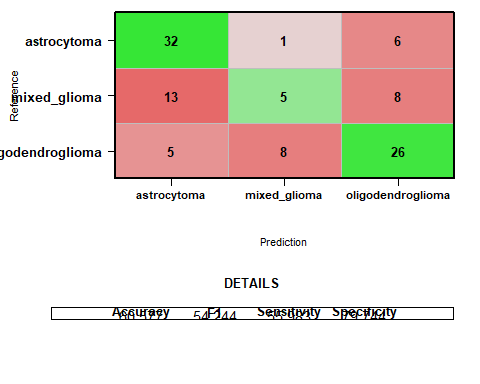
# Evaluar biomarcadores en TRN. Es multiclase, mejor ver accuracy y F1-Score  
knn\_trn\_rf <- knn\_trn(MLMatrix, MLLabels, vars\_selected = FSRankingRF[1:10])  
knn\_results\_rf <- rbind(knn\_trn\_rf$F1Info$meanF1[1:10],knn\_trn\_rf$accuracyInfo$meanAccuracy)

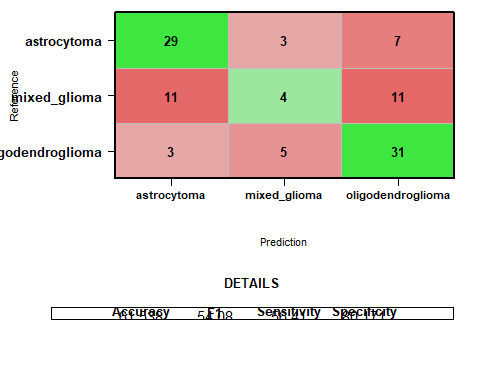
# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score  
knn\_test\_rf <- knn\_test(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected= FSRankingRF[1:10], bestK=knn\_trn\_rf$bestK)  
knn\_results\_rf\_test <- rbind(knn\_trn\_rf$F1Info$meanF1[1:10], knn\_test\_rf$f1Vector, knn\_trn\_rf$accuracyInfo$meanAccuracy, knn\_test\_rf$accVector)



Mientras que el accuracy se mantiene muy constante para el conjunto train, cuando num\_genes está entre 3 y 6, el accuracy comienza a superar el 60%. Comprobemos esta información con la matriz de confusión para el conjunto test. En primer lugar, para una huella de tres genes:

#png('m2.png', width = 500, height = 400)  
dataPlot(knn\_test\_rf$cfMats[[3]]$table, MLLabels, mode = "confusionMatrix")

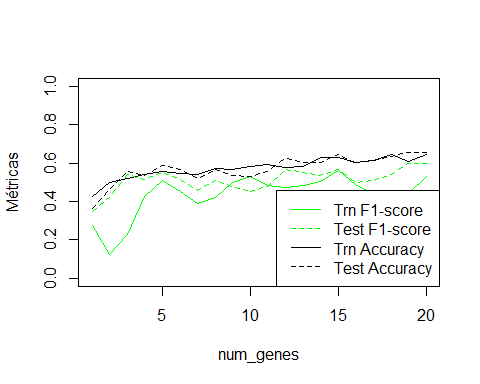
 En segundo lugar, para una huella de seis genes:

 El umento del porcentaje de acierto es muy poco tras elegir el docle de genes. Por tanto, la huella escogida en este caso podría ser 3.

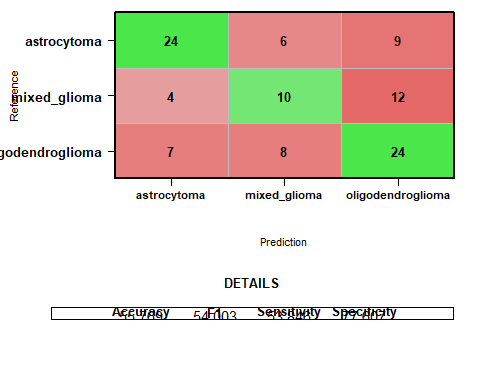
### 3.1.3. Selección de variables con FSRankingDA:

# Evaluar biomarcadores en TRN. Es multiclase, mejor ver accuracy y F1-Score  
knn\_trn\_da <- knn\_trn(MLMatrix, MLLabels, vars\_selected = names(FSRankingDA[1:20]))  
knn\_results\_da <- rbind(knn\_trn\_da$F1Info$meanF1[1:20],knn\_trn\_da$accuracyInfo$meanAccuracy)

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score  
knn\_test\_da <- knn\_test(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected= names(FSRankingDA[1:20]), bestK=knn\_trn\_da$bestK)  
knn\_results\_da\_test <- rbind(knn\_trn\_da$F1Info$meanF1[1:20], knn\_test\_da$f1Vector, knn\_trn\_da$accuracyInfo$meanAccuracy, knn\_test\_da$accVector)



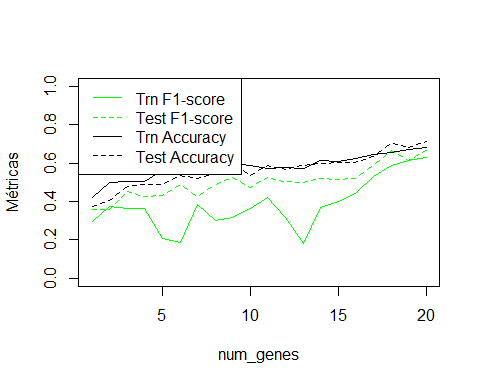
Parece que a partir de 10 genes, dejan de haber picos muy pronunciados para el train. Sin embargo, a partir de ese valor el F1 y el accuracy del conjunto test están entorno al 50%, luego, no lograremos una bune clasificación a menos que escojamos un número muy elevado de genes. Se contrasta esta información con la matriz de confusión para el conjunto test:

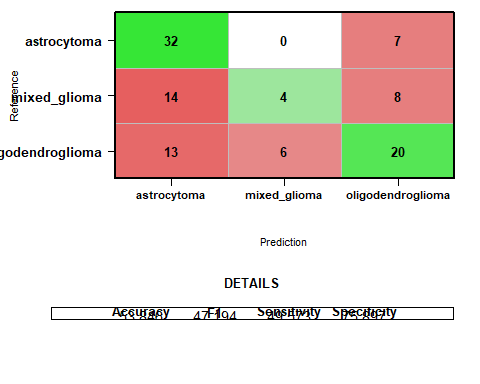


### 3.1.4. Selección de variables con FSRanking\_MB:

Por último, se entrena el modelo knn\_trn\_mb con el ranking creado con el método Markov Blanket y se ajusta el modelo test knn\_test\_mb con el mejor k obtenido en el entrenamiento:

knn\_trn\_mb <- knn\_trn(MLMatrix, MLLabels, vars\_selected = FSRanking\_MB[1:20])  
knn\_results\_mb <- rbind(knn\_trn\_mb$F1Info$meanF1[1:20],knn\_trn\_mb$accuracyInfo$meanAccuracy)  
knn\_test\_mb <- knn\_test(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected=FSRanking\_MB[1:20], bestK=knn\_trn\_mb$bestK)  
knn\_results\_mb\_test <- rbind(knn\_trn\_mb$F1Info$meanF1[1:20], knn\_test\_mb$f1Vector, knn\_trn\_mb$accuracyInfo$meanAccuracy, knn\_test\_mb$accVector)

 Parece que sería apropiado escoger 10 genes o 14 aunque es a partir de 18 genes cuando se empieza a obtener un accuracy más alto, tanto para el train como para el test. Veamos las matrices de confusión para 10 genes, ya que una huella de 14 o 18 ya sería demasiado:



No se obtiene una buena clasificación para la cantidad de genes elegidos. Por tanto, la combinación clasificador-seleccionador para el ejemplo del cáncer de cerebro no es apropiada.

### 3.1.5. Conclusiones

Para el clasificador k-NN, el algoritmo de selección de selección de características mRMR es el que mejor porcetajes de Accuracy propociona: 63.46% con cinco genes seleccionados. En condiciones similares, para Random Forest, el accuracy y el F1 están entorno al 60-62% para una huella de entre 3 y 6 genes. Sin embargo, para el DA y el MB, no se llega al 55% de acierto, ni si siquiera con una huella de 10 genes. En conlusión y por simplicidad, para obtener mejores resultados con el clasificador k-NN, se podría escoger el método de selección de características mRMR y una huella de cinco genes.

## 3.2. Clasificador RandomForest

En esta sección se repetirá el procedimiento aplicado con el algoritmo k-NN (k-Vecinos más cercanos) utilizando los cuatro rankings de características disponibles. Se evaluará el rendimiento del modelo en términos de métricas de clasificación como precision y F1-score , tanto en los datos de entrenamiento como de prueba. Posteriormente, se compararán los resultados obtenidos para cada ranking con el fin de determinar qué características permiten una mejor clasificación de los tres tipos de cáncer de cerebro analizados.

### 3.2.1. Selección de variables con FSRankingMRMR

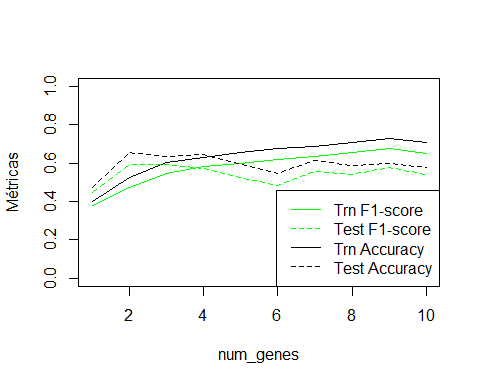
Definamos los modelos de entrenamiento y test, rf\_trn\_mrmr y rf\_test\_mrmr, respectivamente, haciendo uso de las funciones rf\_trn y rf\_test.

# Evaluar biomarcadores en TRN. Es multiclase, mejor ver accuracy y F1-Score  
rf\_trn\_mrmr<- rf\_trn(MLMatrix, MLLabels, vars\_selected = names(FSRankingMRMR[1:10]),numFold = 5)  
save(rf\_trn\_mrmr, file='rf\_trn\_mrmr')

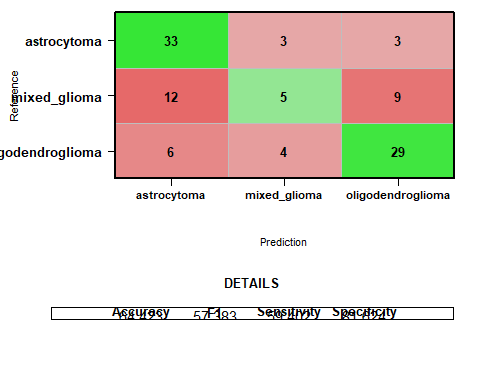
load('rf\_trn\_mrmr')  
rf\_results\_mrmr <- rbind(rf\_trn\_mrmr$F1Info$meanF1[1:10],rf\_trn\_mrmr$accuracyInfo$meanAccuracy)  
#dataPlot(rf\_results\_mrmr, MLLabels, legend = c("F1","Accuracy"), mode = "classResults", xlab="Genes", ylab="Prediction Score")

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score  
rf\_test\_mrmr <- rf\_test(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected= names(FSRankingMRMR[1:10]),bestParameters=rf\_trn\_mrmr$bestParameters)  
  
save(rf\_test\_mrmr, file='rf\_test\_mrmr')

load('rf\_test\_mrmr')  
rf\_results\_mrmr\_test <- rbind(rf\_trn\_mrmr$F1Info$meanF1[1:10], rf\_test\_mrmr$f1Vector, rf\_trn\_mrmr$accuracyInfo$meanAccuracy, rf\_test\_mrmr$accVector)  
#dataPlot(rf\_results\_mrmr\_test, MLLabels, legend = c("Trn F1-score","Test F1-score", "Trn Accuracy","Test Accuracy"), mode = "classResults", xlab="Genes", ylab="Prediction Score")



De la gráfica anterior destaca la armonía entre el accuracy y el F1 para el conjunto test y train. En este caso, se comienzan a obtener mejores valores de accuracy y F1 para train con una huella de 4 genes, aunque después haya un descenso del porcentaje de acierto paulatino para el conjunto test. Para comprobar numéricamente las metricas se presenta la matriz de confusión para el conjunto test:



Este es el modelo con mayor porcentaje de acierto y menor huella hasta el momento.

### 3.2.2. Selección de variables con FSRankingRF

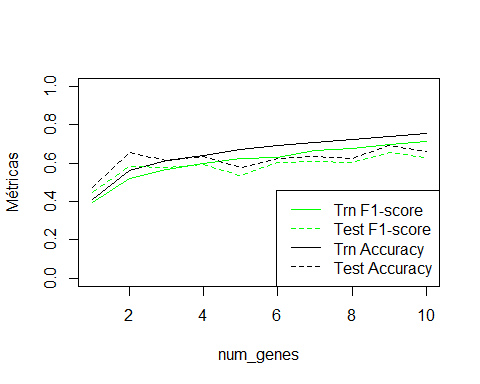
Se definen de la misma manera los modelos rf\_trn\_rf y rf\_test\_rf, haciendo uno del Ranking creado a partir de Random Forest.

# Evaluar biomarcadores en TRN. Es multiclase, mejor ver accuracy y F1-Score  
rf\_trn\_rf<- rf\_trn(MLMatrix, MLLabels, vars\_selected = (FSRankingRF[1:10]),numFold = 5)  
save(rf\_trn\_rf, file='rf\_trn\_rf')

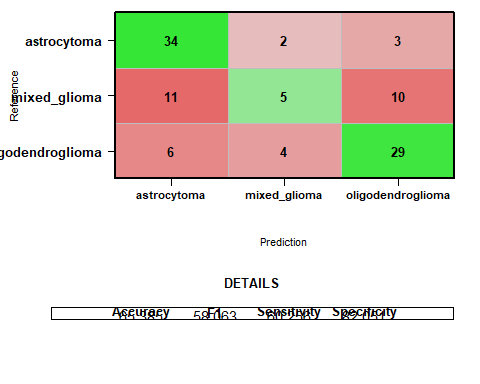
load('rf\_trn\_rf')  
rf\_results\_rf <- rbind(rf\_trn\_rf$F1Info$meanF1[1:10],rf\_trn\_rf$accuracyInfo$meanAccuracy)  
#dataPlot(rf\_results\_rf, MLLabels, legend = c("F1","Accuracy"), mode = "classResults", xlab="Genes", ylab="Prediction Score")

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score  
rf\_test\_rf <- rf\_test(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected = (FSRankingRF[1:10]), bestParameters=rf\_trn\_rf$bestParameters)  
save(rf\_test\_rf, file='rf\_test\_rf')

load('rf\_test\_rf')  
rf\_results\_rf\_test <- rbind(rf\_trn\_rf$F1Info$meanF1[1:10], rf\_test\_rf$f1Vector, rf\_trn\_rf$accuracyInfo$meanAccuracy, rf\_test\_rf$accVector)  
#dataPlot(rf\_results\_rf\_test, MLLabels, legend = c("Trn F1-score","Test F1-score", "Trn Accuracy","Test Accuracy"), mode = "classResults", xlab="Genes", ylab="Prediction Score")



Como en el caso anterior, la mejor huella es cuando num\_genes = 2. A continuación, se muestra la matriz de confusión para el conjunto test, con el objetivo de porder hacer conclusiones más fiables:



Destaca que el accuracy en este caso y en el anterior para una huella de 2 genes, es el mismo. Por tanto, para distinguirlos podríamos fijarnos en que el F1-score para el ranking del método mRMR (59.1%) es algo mayor que en este caso.

### 3.2.3. Selección de variables con FSRankingDA

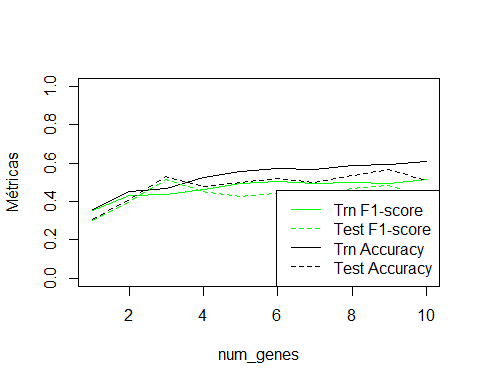
De la misma forma, se ajustan los modelos rf\_trn\_da y rf\_test\_da para el ranking de DA. Se llevará a cabo también la evaluación del rendimiento del modelo mediante métricas como la precision y el F1-score , tanto en los datos de entrenamiento como en los de prueba.

# Evaluar biomarcadores en TRN. Es multiclase, mejor ver accuracy y F1-Score  
rf\_trn\_da<- rf\_trn(MLMatrix, MLLabels, vars\_selected = names(FSRankingDA[1:10]), numFold = 5)   
save(rf\_trn\_da, file='rf\_trn\_da')

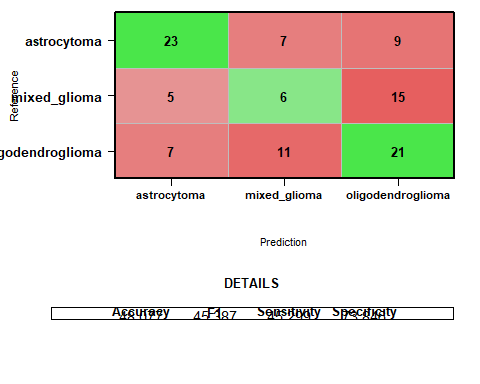
load('rf\_trn\_da')  
rf\_results\_da <- rbind(rf\_trn\_da$F1Info$meanF1[1:10],rf\_trn\_da$accuracyInfo$meanAccuracy)  
#dataPlot(rf\_results\_da, MLLabels, legend = c("F1","Accuracy"), mode = "classResults", xlab="Genes", ylab="Prediction Score")

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score  
rf\_test\_da <- rf\_test(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected = names(FSRankingDA[1:10]), bestParameters=rf\_trn\_da$bestParameters)  
save(rf\_test\_da, file='rf\_test\_da')

load('rf\_test\_da')  
rf\_results\_da\_test <- rbind(rf\_trn\_da$F1Info$meanF1[1:10], rf\_test\_da$f1Vector, rf\_trn\_da$accuracyInfo$meanAccuracy, rf\_test\_da$accVector)  
#dataPlot(rf\_results\_da\_test, MLLabels, legend = c("Trn F1-score","Test F1-score", "Trn Accuracy","Test Accuracy"), mode = "classResults", xlab="Genes", ylab="Prediction Score")



Se observa que, en este caso, el F1-score y el accuracy son más bajos. Además, el accuracy y el F1 del train, a partir de num\_genes = 4, parece que aumenta pero a penas alcanza el 60%. Veamos lo explicado en la siguiente matriz de confusión para el conjunto test:



Con la mejor huella para el conjunto train, el test a penas acierta en el 50% de las ocasiones.

### 3.2.4. Selección de variables con FSRankingMB

Finalmente, busquemos la huella para el Ranking de MB, entrenando los modelos rf\_trn\_mb y rf\_test\_mb.

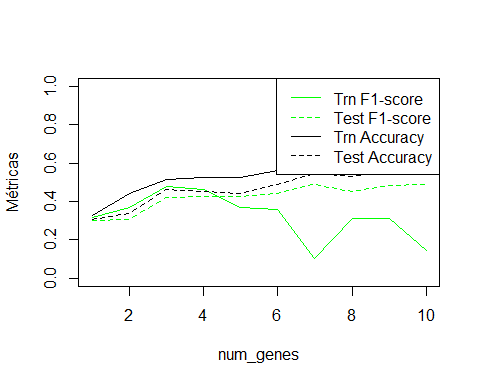
rf\_trn\_mb <- rf\_trn(MLMatrix, MLLabels, vars\_selected = FSRanking\_MB[1:10])  
save(rf\_trn\_mb, file='rf\_trn\_mb')

load('rf\_trn\_mb')

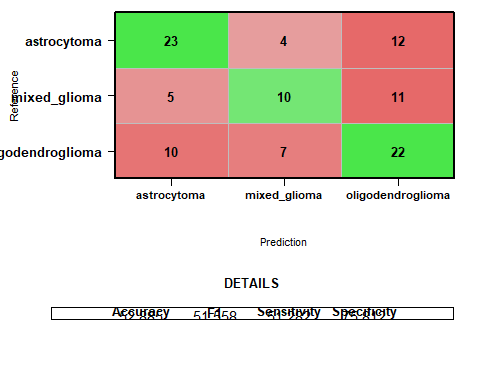
rf\_results\_mb <- rbind(rf\_trn\_mb$F1Info$meanF1[1:10],rf\_trn\_mb$accuracyInfo$meanAccuracy)  
rf\_test\_mb <- rf\_test(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected=FSRanking\_MB[1:10], bestParameters = rf\_trn\_mb$bestParameters)

## Testing with 1 variables...  
## Testing with 2 variables...  
## Testing with 3 variables...  
## Testing with 4 variables...  
## Testing with 5 variables...  
## Testing with 6 variables...  
## Testing with 7 variables...  
## Testing with 8 variables...  
## Testing with 9 variables...  
## Testing with 10 variables...  
## Classification done successfully!

rf\_results\_mb\_test <- rbind(rf\_trn\_mb$F1Info$meanF1[1:10], rf\_test\_mb$f1Vector, rf\_trn\_mb$accuracyInfo$meanAccuracy, rf\_test\_mb$accVector)



Destaca el descenso del F1-score para el conjunto train a partir de 3 genes. Elegimos esta huella aún observando que el accuracy del train está entorno al 50%. Veamos lo obtenido para el conjunto test en la siguiente matriz de confusión:



### 3.2.5. Conclusiones

Si nos fijamos en el porcentaje de acierto del clasificador Random Forest, si seleccionamos características con mRMR, el accuracy y el F1 del conjunto test están entorno al 64.4% y 57.4%, respectivamente, con una huella de 4 genes. Para una huella de 2 genes, con RF se obtienen unos porcentajes de acierto y F1 de 65.4% y 58.1%, respectivamente. En este último caso, además el F1 es algo superior para el conjunto test. Por otro lado, si observamos la matriz de confusión del seleccionador de características DA o MB se concluye que los porcentajes de acierto son menores que los anteriores, estando entrono al 50%, y no siendo, por tanto, buenas opciones.

En conclusión, para el clasificador Random Forest se podría elegir RF como seleccionador de características con una huella de 2 genes.

## 3.3. Clasificador SVM

En esta sección se abordará el entrenamiento de los modelos train y test del clasificador SVM (Support Vector Machine) para la clasificación de los tres tipos de cáncer cerebral. A lo largo de esta sección se utilizaran las funciones svm\_trn y svm\_test2. Esta última función es la misma que la función del KnowSeq svm\_test pero con los arreglos que se indican para el buen funcionamiento en estro problema:

svm\_test2 <-function(train,labelsTrain,test,labelsTest,vars\_selected,bestParameters){  
  
 if(!is.data.frame(train) && !is.matrix(train)){  
   
 stop("The train argument must be a dataframe or a matrix.")  
   
 }  
   
 if(dim(train)[1] != length(labelsTrain)){  
   
 stop("The length of the rows of the argument train must be the same than the length of the lablesTrain. Please, ensures that the rows are the samples and the columns are the variables.")  
   
 }  
   
 if(!is.character(labelsTrain) && !is.factor(labelsTrain)){stop("The class of the labelsTrain parameter must be character vector or factor.")}  
 if(is.character(labelsTrain)){ labelsTrain <- as.factor(labelsTrain) }  
   
 if(!is.character(labelsTest) && !is.factor(labelsTest)){stop("The class of the labelsTest parameter must be character vector or factor.")}  
 if(is.character(labelsTest)){ labelsTest <- as.factor(labelsTest) }  
   
 if(!is.data.frame(test) && !is.matrix(test)){  
   
 stop("The test argument must be a dataframe or a matrix.")  
   
 }  
   
 if(dim(test)[1] != length(labelsTest)){  
   
 stop("The length of the rows of the argument test must be the same than the length of the lablesTest. Please, ensures that the rows are the samples and the columns are the variables.")  
   
 }  
   
 train <- as.data.frame(apply(train,2,as.double))  
 train <- train[,vars\_selected]  
 test <- as.data.frame(apply(test,2,as.double))  
 test <- test[,vars\_selected]  
   
 train = vapply(train, function(x){   
 max <- max(x)  
 min <- min(x)  
 if(max > min){  
 x <- ((x - min) / (max - min)) \* 2 - 1  
 }  
 else{  
 x  
 }}, double(nrow(train)))  
   
 train <- as.data.frame(train)  
   
 test = vapply(test, function(x){   
 max <- max(x)  
 min <- min(x)  
 if(max > min){  
 x <- ((x - min) / (max - min)) \* 2 - 1  
 }  
 else{  
 x  
 }}, double(nrow(test)))  
   
 test <- as.data.frame(test)  
  
 accVector <- double()  
 sensVector <- double()  
 specVector <- double()  
 f1Vector <- double()  
 cfMatList <- list()  
 colNames <- colnames(train)  
 for(i in seq\_len(dim(test)[2])){  
 cat(paste("Testing with ", i," variables...\n",sep=""))  
 columns <- make.names(c(colNames[seq(i)])) # make.names()convierte los nombres en identificadores sintácticamente válidos  
 tr\_ctr <- trainControl(method="none")  
 colnames(train)<-make.names(colnames(train)) # se alteran los nombres para que coincidan con los de las columnas  
 dataForTrt <- data.frame(cbind(subset(train, select=columns),labelsTrain))  
 colnames(train)[seq(i)] <- make.names(columns)  
 svm\_model <- train(labelsTrain ~ ., data = dataForTrt, type = "C-svc",   
 method = "svmRadial", preProc = c("center", "scale"),  
 trControl = tr\_ctr,   
 tuneGrid=data.frame(sigma=getElement(bestParameters, "gamma"),   
 C = getElement(bestParameters, "C")))  
 colnames(test)<-make.names(colnames(test)) # alteramos los nombres en el conjunto test para poder extraer del dataframe test los genes con el mismo nombre que en la variable columns  
 testX = subset(test, select=columns)  
 unkX <- testX  
 colnames(unkX) <- make.names(colnames(testX))  
 colnames(testX) <- make.names(colnames(testX))  
 predicts <- extractPrediction(list(my\_svm=svm\_model), testX = testX, unkX = unkX,  
 unkOnly = !is.null(unkX) & !is.null(testX))  
   
 predicts <- predicts$pred  
   
 cfMat<-confusionMatrix(predicts,labelsTest)  
   
 if (length(levels(labelsTrain))==2){  
 sens <- cfMat$byClass[[1]]  
 spec <- cfMat$byClass[[2]]  
 f1 <- cfMat$byClass[[7]]  
 } else{  
 sens <- mean(cfMat$byClass[,1])  
 spec <- mean(cfMat$byClass[,2])  
 cfMat$byClass[,7][is.na(cfMat$byClass[,7])] <- 0   
 # CAMBIO:  
 # Es mejor reemplazar los valores NA en esta etapa para permitir que el F1-score se grafique correctamente.   
 # Si el reemplazo se realiza después, el promedio de los tres F1-scores inicialmente resultará en NA y luego en cero, lo que excluye el F1 de las demás clases.   
 # Reemplazar el NA aquí garantiza que el cálculo de la media considere un F1 de 0 en caso de que la precisión sea NA y el recall sea 0. Esto sucede cuando una clase no es predicha ni correctamente ni incorrectamente,   
 # es decir, cuando el número de falsos positivos y falsos negativos es 0.  
 f1 <- mean(cfMat$byClass[,7])  
 }  
   
 cfMatList[[i]] <- cfMat  
 accVector[i] <- cfMat$overall[[1]]  
 sensVector[i] <- sens  
 specVector[i] <- spec  
 f1Vector[i] <- f1  
   
 #if(is.na(f1Vector[i])) f1Vector[i] <- 0   
 }  
  
 cat("Classification done successfully!\n")  
 names(accVector) <- vars\_selected  
 names(sensVector) <- vars\_selected  
 names(specVector) <- vars\_selected  
 names(f1Vector) <- vars\_selected  
  
 results <- list(cfMatList,accVector,sensVector,specVector,f1Vector)  
 names(results) <- c("cfMats","accVector","sensVector","specVector","f1Vector")  
 invisible(results)  
  
}

Realizamos el mismo cambio relativo al cálculo del valor F1 para el código fuente de la función svm\_train:

svm\_trn <- function(data, labels, vars\_selected, numFold = 10) {  
 if (!is.data.frame(data) && !is.matrix(data)) {  
 stop("The data argument must be a dataframe or a matrix.")  
 }  
 if (dim(data)[1] != length(labels)) {  
 stop("The length of the rows of the argument data must be the same than the length of the lables. Please, ensures that the rows are the samples and the columns are the variables.")  
 }  
   
 if (!is.character(labels) && !is.factor(labels)) {  
 stop("The class of the labels parameter must be character vector or factor.")  
 }  
 if (is.character(labels)) {  
 labels <- as.factor(labels)  
 }  
   
 if (numFold %% 1 != 0 || numFold == 0) {  
 stop("The numFold argument must be integer and greater than 0.")  
 }  
   
 data <- as.data.frame(apply(data, 2, as.double))  
 data <- data[, vars\_selected]  
   
 data <- vapply(data, function(x) {  
 max <- max(x)  
 min <- min(x)  
 if(max > min){  
 x <- ((x - min) / (max - min)) \* 2 - 1  
 }  
 else{  
 x  
 }  
 }, double(nrow(data)))  
   
 data <- as.data.frame(data)  
   
 fitControl <- trainControl(method = "cv", number = 10)  
 cat("Tuning the optimal C and G...\n")  
   
 grid\_radial <- expand.grid(  
 sigma = c(  
 0, 0.01, 0.02, 0.025, 0.03, 0.04,  
 0.05, 0.06, 0.07, 0.08, 0.09, 0.1, 0.25, 0.5, 0.75, 0.9  
 ),  
 C = c(  
 0.01, 0.05, 0.1, 0.25, 0.5, 0.75,  
 1, 1.5, 2, 5  
 )  
 )  
   
 dataForTunning <- cbind(data, labels)  
 colnames(dataForTunning) <- make.names(colnames(dataForTunning))  
 Rsvm\_sb <- train(labels ~ ., data = dataForTunning, type = "C-svc", method = "svmRadial", preProc = c("center", "scale"), trControl = fitControl, tuneGrid = grid\_radial)  
   
 bestParameters <- c(C = Rsvm\_sb$bestTune$C, gamma = Rsvm\_sb$bestTune$sigma)  
 cat(paste("Optimal cost:", bestParameters[1], "\n"))  
 cat(paste("Optimal gamma:", bestParameters[2], "\n"))  
   
 acc\_cv <- matrix(0L, nrow = numFold, ncol = dim(data)[2])  
 sens\_cv <- matrix(0L, nrow = numFold, ncol = dim(data)[2])  
 spec\_cv <- matrix(0L, nrow = numFold, ncol = dim(data)[2])  
 f1\_cv <- matrix(0L, nrow = numFold, ncol = dim(data)[2])  
 cfMatList <- list()  
 # compute size of val fold  
 lengthValFold <- dim(data)[1]/numFold  
   
 # reorder the data matrix in order to have more  
 # balanced folds  
 positions <- rep(seq\_len(dim(data)[1]))  
 randomPositions <- sample(positions)  
 data <- data[randomPositions,]  
 labels <- labels[randomPositions]  
  
 for (i in seq\_len(numFold)) {  
 cat(paste("Training fold ", i, "...\n", sep = ""))  
   
 # obtain validation and training folds  
 valFold <- seq(round((i-1)\*lengthValFold + 1 ), round(i\*lengthValFold))  
 trainDataCV <- setdiff(seq\_len(dim(data)[1]), valFold)  
 testDataset<- data[valFold,]  
 trainingDataset <- data[trainDataCV,]  
 labelsTrain <- labels[trainDataCV]  
 labelsTest <- labels[valFold]  
 colNames <- colnames(trainingDataset)  
   
 for (j in seq\_len(length(vars\_selected))) {  
 columns <- c(colNames[seq(j)])  
 tr\_ctr <- trainControl(method="none")  
 dataForTrt <- data.frame(cbind(subset(trainingDataset, select=columns),labelsTrain))  
 colnames(dataForTrt)[seq(j)] <- make.names(columns)  
 svm\_model <- train(labelsTrain ~ ., data = dataForTrt, type = "C-svc",   
 method = "svmRadial", preProc = c("center", "scale"),  
 trControl = tr\_ctr,   
 tuneGrid=data.frame(sigma = bestParameters[2], C = bestParameters[1]))  
   
 testX = subset(testDataset, select=columns)  
 unkX <- testX  
 colnames(unkX) <- make.names(colnames(testX))  
 colnames(testX) <- make.names(colnames(testX))  
 predicts <- extractPrediction(list(my\_svm=svm\_model), testX = testX, unkX = unkX,  
 unkOnly = !is.null(unkX) & !is.null(testX))  
   
 predicts <- predicts$pred  
   
 cfMatList[[i]] <- confusionMatrix(predicts, labelsTest)  
 acc\_cv[i, j] <- cfMatList[[i]]$overall[[1]]  
   
 if (length(levels(labelsTrain))==2){  
 sens <- cfMatList[[i]]$byClass[[1]]  
 spec <- cfMatList[[i]]$byClass[[2]]  
 f1 <- cfMatList[[i]]$byClass[[7]]  
 } else{  
 sens <- mean(cfMatList[[i]]$byClass[,1])  
 spec <- mean(cfMatList[[i]]$byClass[,2])  
 cfMatList[[i]]$byClass[,7][is.na(cfMatList[[i]]$byClass[,7])] <- 0 # CAMBIO  
 f1 <- mean(cfMatList[[i]]$byClass[,7])  
 }  
   
 sens\_cv[i, j] <- sens  
 spec\_cv[i, j] <- spec  
 f1\_cv[i, j] <- f1  
   
 if(is.na(sens\_cv[i,j])) sens\_cv[i,j] <- 0  
 if(is.na(spec\_cv[i,j])) spec\_cv[i,j] <- 0  
 if(is.na(f1\_cv[i,j])) f1\_cv[i,j] <- 0  
 }  
 }  
   
 meanAcc <- colMeans(acc\_cv)  
 names(meanAcc) <- colnames(acc\_cv)  
 sdAcc <- apply(acc\_cv, 2, sd)  
 accuracyInfo <- list(meanAcc, sdAcc)  
 names(accuracyInfo) <- c("meanAccuracy","standardDeviation")  
   
   
 meanSens <- colMeans(sens\_cv)  
 names(meanSens) <- colnames(sens\_cv)  
 sdSens <- apply(sens\_cv, 2, sd)  
 sensitivityInfo <- list(meanSens, sdSens)  
 names(sensitivityInfo) <- c("meanSensitivity","standardDeviation")  
   
   
 meanSpec <- colMeans(spec\_cv)  
 names(meanSpec) <- colnames(spec\_cv)  
 sdSpec <- apply(spec\_cv, 2, sd)  
 specificityInfo <- list(meanSpec, sdSpec)  
 names(specificityInfo) <- c("meanSpecificity","standardDeviation")  
   
   
 meanF1 <- colMeans(f1\_cv)  
 names(meanF1) <- colnames(f1\_cv)  
 sdF1 <- apply(f1\_cv, 2, sd)  
 F1Info <- list(meanF1, sdF1)  
 names(F1Info) <- c("meanF1","standardDeviation")  
   
 cat("Classification done successfully!\n")  
 results\_cv <- list(cfMatList,accuracyInfo,sensitivityInfo,specificityInfo,F1Info,bestParameters)  
 names(results\_cv) <- c("cfMats","accuracyInfo","sensitivityInfo","specificityInfo","F1Info","bestParameters")  
 invisible(results\_cv)  
   
}

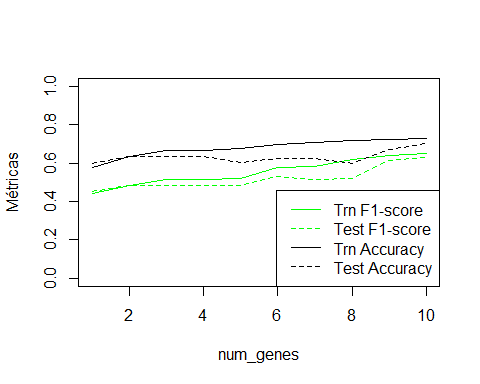
### 3.3.1. Selección de variables con FSRankingMRMR

Comenzamos con las variables seleccionadas con mRMR, entrenando los modelos svm\_trn\_mrmr y svm\_test\_mrmr.

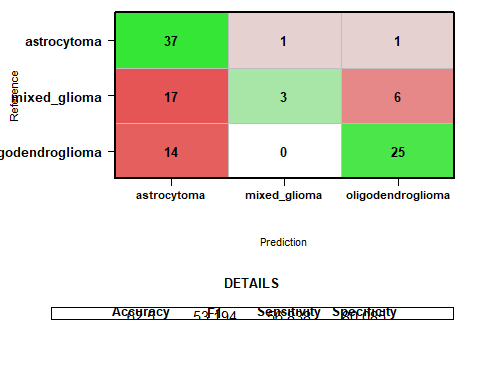
# Evaluar biomarcadores en TRN. Es multiclase, mejor ver accuracy y F1-Score  
svm\_trn\_mrmr<- svm\_trn(MLMatrix, MLLabels, vars\_selected = names(FSRankingMRMR[1:10]),numFold = 5)  
  
save(svm\_trn\_mrmr, file='svm\_trn\_mrmr')

load('svm\_trn\_mrmr')  
svm\_results\_mrmr <- rbind(svm\_trn\_mrmr$F1Info$meanF1[1:10],svm\_trn\_mrmr$accuracyInfo$meanAccuracy)  
#dataPlot(svm\_results\_mrmr, MLLabels, legend = c("F1","Accuracy"), mode = "classResults", xlab="Genes", ylab="Prediction Score")

svm\_test\_mrmr <- svm\_test2(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected=names(FSRankingMRMR[1:10]), bestParameters= svm\_trn\_mrmr$bestParameters)  
svm\_results\_mrmr\_test <- rbind(svm\_trn\_mrmr$F1Info$meanF1[1:10], svm\_test\_mrmr$f1Vector, svm\_trn\_mrmr$accuracyInfo$meanAccuracy, svm\_test\_mrmr$accVector)



En este caso, para num\_genes = 6, destaca el pico de F1 para el conjunt train, por tanto, elegimos esta huella a partir de la cual no hay cambios tan bruscos. Se comprueba con la matriz de confusión para el conjunto test:

 Cabe destacar que ningún astocitoma u oligodendroglioma se ha predicho como glioma mixto. Sin embargo, al contrario sí y en numerosas ocasiones, 17 y 6, respectivamente.

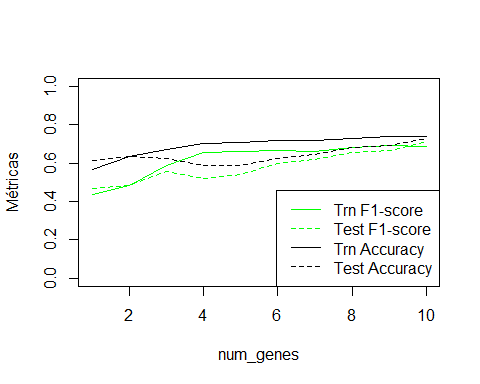
### 3.3.2. Selección de variables con FSRankingRF:

En este caso, se entrenan los modelos svm\_train\_rf y svm\_test\_rf. Calculando las m-etricas correspondientes para la valoreación de la clasificación en el conjunto test. Los mejores parámetros obtenidos en este caso son C = 0.75 y gamma = 0.03.

# Evaluar biomarcadores en TRN. Es multiclase, mejor ver accuracy y F1-Score  
svm\_trn\_rf<- svm\_trn(MLMatrix, MLLabels, vars\_selected = (FSRankingRF[1:10]),numFold = 5)  
save(svm\_trn\_rf, file='svm\_trn\_rf')

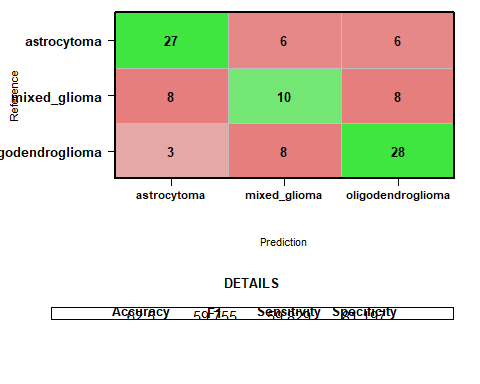
load('svm\_trn\_rf')  
svm\_results\_rf <- rbind(svm\_trn\_rf$F1Info$meanF1[1:10],svm\_trn\_rf$accuracyInfo$meanAccuracy)  
#dataPlot(svm\_results\_rf, MLLabels, legend = c("F1","Accuracy"), mode = "classResults", xlab="Genes", ylab="Prediction Score")

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score  
svm\_test\_rf <- svm\_test2(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected = (FSRankingRF[1:10]), bestParameters= svm\_trn\_rf$bestParameters)  
svm\_results\_rf\_test <- rbind(svm\_trn\_rf$F1Info$meanF1[1:10], svm\_test\_rf$f1Vector, svm\_trn\_rf$accuracyInfo$meanAccuracy, svm\_test\_rf$accVector)



En este caso, elegiría una huella de 4 genes ya que, a partir de ahí, el crecimeinto del F1 y accuracy en train es gradual. Sin embargo, si observamos las métricas del conjunto test, en realidad comienzan a ascender a partir de 6 genes. Vemaos así, la matriz de confusión para el conjunto test cuando la huella es 4:

#png('m10.png', width=500, height = 400)  
dataPlot(svm\_test\_rf$cfMats[[6]]$table, MLLabels, mode = "confusionMatrix")



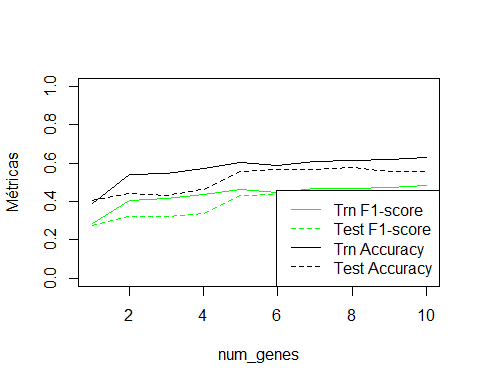
### 3.3.3. Selección de variables con FSRankingDA:

Repetimos el porceso para el ranking de DA.

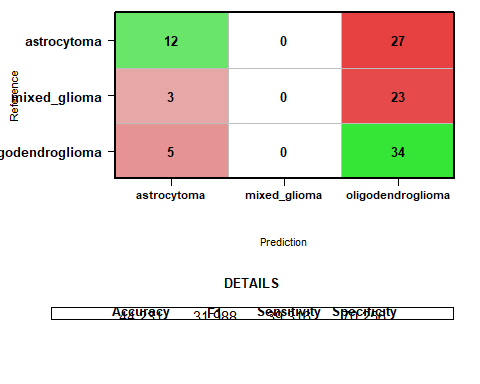
# Evaluar biomarcadores en TRN. Es multiclase, mejor ver accuracy y F1-Score  
svm\_trn\_da<- svm\_trn(MLMatrix, MLLabels, vars\_selected = names(FSRankingDA[1:10]))   
save(svm\_trn\_da, file='svm\_trn\_da')

load('svm\_trn\_da')  
svm\_results\_da <- rbind(svm\_trn\_da$F1Info$meanF1[1:10],svm\_trn\_da$accuracyInfo$meanAccuracy)  
#dataPlot(svm\_results\_da, MLLabels, legend = c("F1","Accuracy"), mode = "classResults", xlab="Genes", ylab="Prediction Score")

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score  
svm\_test\_da <- svm\_test2(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected = names(FSRankingDA[1:10]), bestParameters= svm\_trn\_da$bestParameters)  
svm\_results\_da\_test <- rbind(svm\_trn\_da$F1Info$meanF1[1:10], svm\_test\_da$f1Vector, svm\_trn\_da$accuracyInfo$meanAccuracy, svm\_test\_da$accVector)



A partir de 2 genes, aparece que el accuracy el F1 para el conjunto de entrenamiento mejora y se mantiene bastante constante. Aunque, para las métricas del conjunto test, es a partir de 5 genes cuando mejoran. Veamos la matriz de confusión para el conjunto test y una huella de dos genes:

 La clase glioma mixto no es predecida por el modelo. Además, tiene un accracy muy bajo, el cuál para una huella de 5 genes mejora al 55%. Este aumento no mejora notablemente la clasificación del modelo.

### 3.3.4. Selección de variables con FSRanking\_MB:

Por último, veamos los resultados para el FSRanking\_MB.

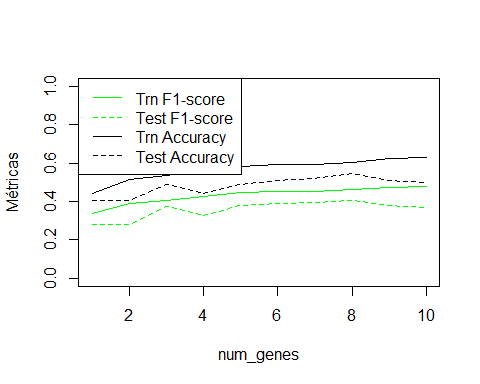
svm\_trn\_mb <- svm\_trn(MLMatrix, MLLabels, vars\_selected = FSRanking\_MB[1:10])  
save(svm\_trn\_mb, file='svm\_trn\_mb')

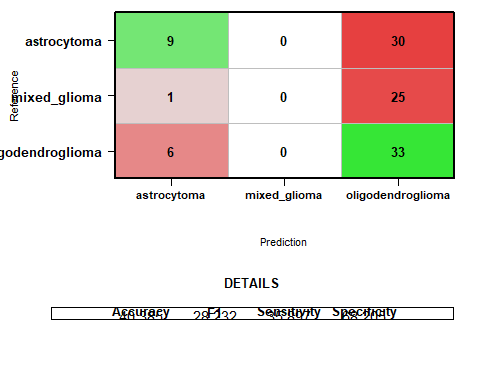
load('svm\_trn\_mb')

svm\_results\_mb <- rbind(svm\_trn\_mb$F1Info$meanF1[1:10],svm\_trn\_mb$accuracyInfo$meanAccuracy)  
svm\_test\_mb <- svm\_test2(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected=FSRanking\_MB[1:10], bestParameters = svm\_trn\_mb$bestParameters)

## Testing with 1 variables...  
## Testing with 2 variables...  
## Testing with 3 variables...  
## Testing with 4 variables...  
## Testing with 5 variables...  
## Testing with 6 variables...  
## Testing with 7 variables...  
## Testing with 8 variables...  
## Testing with 9 variables...  
## Testing with 10 variables...  
## Classification done successfully!

svm\_results\_mb\_test <- rbind(svm\_trn\_mb$F1Info$meanF1[1:10], svm\_test\_mb$f1Vector, svm\_trn\_mb$accuracyInfo$meanAccuracy, svm\_test\_mb$accVector)

 Si nos fijamos en el accuracy y el F1 del train, podríamos escoger una huella de 2 genes. Veamos la matriz de confusión para esta huella en el conjunto test:

 Como vemos es una de las peores combinaciones clasificador-seleccionador hasta el momento.

### 3.3.5. Conclusiones

En el caso del clasificador SVM, si seleccionamos características DA o con MB, se tiene un accuracy muy poco superior al 40% con una huella de 2 genes; con una opción muy baja a mejora si aumentamos esta huella. Sin embargo, para el algoritmo de selección mRMR y una huella de seis genes, el accuracy es del 62.5%, igual al del Random Forest con la misma huella. Además, para Random Forest es con el que también se obtiene mayor porcentaje de F1, 60%. Por ello, nos quedaremos con esta última combinación.

## 3.4. Clasificador Regresión Logística

Se diseña en esta sección la sunción logistic\_train2 ayudandonos del código base de la función de KnowSeq knn\_trn:

data<-MLMatrix  
labels<-MLLabels  
vars\_selected = names(FSRankingMRMR[1:10])  
numFold = 5  
logistic\_trn2 <- function(data, labels, vars\_selected, numFold = 10, LOOCV = FALSE){  
   
 if (!is.data.frame(data) && !is.matrix(data)) {  
 stop("The data argument must be a dataframe or a matrix.")  
 }  
 if (dim(data)[1] != length(labels)) {  
 stop("The length of the rows of the argument data must be the same as the length of the labels. Please ensure that the rows are the samples and the columns are the variables.")  
 }  
   
 if (!is.character(labels) && !is.factor(labels)) {  
 stop("The class of the labels parameter must be a character vector or factor.")  
 }  
   
 if (is.character(labels)) {  
 labels <- as.factor(labels)  
 }  
   
 if (numFold %% 1 != 0 || numFold == 0) {  
 stop("The numFold argument must be an integer and greater than 0.")  
 }  
   
 data <- as.data.frame(apply(data, 2, as.double))  
 data <- data[, vars\_selected]  
   
 # Scaling the data between -1 and 1  
 data = vapply(data, function(x) {   
 max\_val <- max(x)  
 min\_val <- min(x)  
 if (max\_val > min\_val) {  
 x <- ((x - min\_val) / (max\_val - min\_val)) \* 2 - 1  
 }  
 return(x)  
 }, double(nrow(data)))  
   
 data <- as.data.frame(data)  
   
 fitControl <- trainControl(method = "repeatedcv", number = numFold, repeats = 3)  
 cat("Tuning the optimal model...\n")  
   
 # Use logistic regression with multinomial family for 3 classes  
 logistic\_model <- train(data, labels, method = "multinom", trControl = fitControl, preProcess = c("center", "scale"))  
   
 cat(paste("Optimal model tuned.\n"))  
   
 if (LOOCV == FALSE) {   
 accuracyInfo <- list()  
 sensitivityInfo <- list()  
 specificityInfo <- list()  
 f1Info <- list()  
   
 acc\_cv <- matrix(0L, nrow = numFold, ncol = length(vars\_selected))  
 sens\_cv <- matrix(0L, nrow = numFold, ncol = length(vars\_selected))  
 spec\_cv <- matrix(0L, nrow = numFold, ncol = length(vars\_selected))  
 f1\_cv <- matrix(0L, nrow = numFold, ncol = length(vars\_selected))  
   
 cfMatList <- list()  
 lengthValFold <- dim(data)[1] / numFold  
   
 positions <- rep(seq\_len(dim(data)[1]))  
 randomPositions <- sample(positions)  
 data <- data[randomPositions, ]  
 labels <- labels[randomPositions]  
   
 for (i in seq\_len(numFold)) {  
 cat("Running K-Fold Cross-Validation...\n")  
 cat(paste("Training fold ", i, "...\n", sep = ""))  
   
 valFold <- seq(round((i - 1) \* lengthValFold + 1), round(i \* lengthValFold))  
 trainDataCV <- setdiff(seq\_len(dim(data)[1]), valFold)  
 testDataset <- data[valFold, ]  
 trainingDataset <- data[trainDataCV, ]  
 labelsTrain <- labels[trainDataCV]  
 labelsTest <- labels[valFold]  
   
 # Train logistic regression model for 3 classes  
 logistic\_mod <- multinom(labelsTrain ~ ., data = trainingDataset)  
 predicts <- predict(logistic\_mod, testDataset)  
   
 cfMatList[[i]] <- confusionMatrix(predicts, labelsTest)  
 acc\_cv[i, 1] <- cfMatList[[i]]$overall[[1]]  
   
 # Compute sensitivity, specificity, and F1 score  
 if (length(levels(labelsTrain))==2){  
 sens <- cfMatList[[i]]$byClass[[1]]  
 spec <- cfMatList[[i]]$byClass[[2]]  
 f1 <- cfMatList[[i]]$byClass[[7]]  
 } else{  
 sens <- mean(cfMatList[[i]]$byClass[,1])  
 spec <- mean(cfMatList[[i]]$byClass[,2])  
 f1 <- mean(cfMatList[[i]]$byClass[,7])  
 }  
 sens\_cv[i, 1] <- sens  
 spec\_cv[i, 1] <- spec  
 f1\_cv[i, 1] <- f1  
   
 if (is.na(sens\_cv[i, 1])) sens\_cv[i, 1] <- 0  
 if (is.na(spec\_cv[i, 1])) spec\_cv[i, 1] <- 0  
 #if (is.na(f1\_cv[i, 1])) f1\_cv[i, 1] <- 0  
   
 for (j in 2:length(vars\_selected)) {  
 logistic\_mod <- multinom(labelsTrain ~ ., data = trainingDataset[, 1:j])  
 predicts <- predict(logistic\_mod, testDataset[, 1:j])  
   
 cfMatList[[i]] <- confusionMatrix(predicts, labelsTest)  
 acc\_cv[i, j] <- cfMatList[[i]]$overall[[1]]  
   
 cfMatList[[i]]$byClass[, 1][is.na(cfMatList[[i]]$byClass[, 1])] <- 0  
 sens <- mean(cfMatList[[i]]$byClass[, 1])  
 cfMatList[[i]]$byClass[, 2][is.na(cfMatList[[i]]$byClass[, 2])] <- 0  
 spec <- mean(cfMatList[[i]]$byClass[, 2])  
 cfMatList[[i]]$byClass[, 7][is.na(cfMatList[[i]]$byClass[, 7])] <- 0  
 f1 <- mean(cfMatList[[i]]$byClass[, 7])  
   
 sens\_cv[i, j] <- sens  
 spec\_cv[i, j] <- spec  
 f1\_cv[i, j] <- f1  
   
 if (is.na(sens\_cv[i, j])) sens\_cv[i, j] <- 0  
 if (is.na(spec\_cv[i, j])) spec\_cv[i, j] <- 0  
 #if (is.na(f1\_cv[i, j])) f1\_cv[i, j] <- 0  
 }  
 }  
   
 # Calculate mean and standard deviation  
 meanAcc <- colMeans(acc\_cv)  
 sdAcc <- apply(acc\_cv, 2, sd)  
 accuracyInfo <- list(meanAcc, sdAcc)  
 names(accuracyInfo) <- c("meanAccuracy", "standardDeviation")  
   
 meanSens <- colMeans(sens\_cv)  
 sdSens <- apply(sens\_cv, 2, sd)  
 sensitivityInfo <- list(meanSens, sdSens)  
 names(sensitivityInfo) <- c("meanSensitivity", "standardDeviation")  
   
 meanSpec <- colMeans(spec\_cv)  
 sdSpec <- apply(spec\_cv, 2, sd)  
 specificityInfo <- list(meanSpec, sdSpec)  
 names(specificityInfo) <- c("meanSpecificity", "standardDeviation")  
   
 meanF1 <- colMeans(f1\_cv)  
 sdF1 <- apply(f1\_cv, 2, sd)  
 F1Info <- list(meanF1, sdF1)  
 names(F1Info) <- c("meanF1", "standardDeviation")  
   
 cat("Classification done successfully!\n")  
 results\_cv <- list(cfMatList, accuracyInfo, sensitivityInfo, specificityInfo, F1Info, logistic\_model)  
 names(results\_cv) <- c("cfMats", "accuracyInfo", "sensitivityInfo", "specificityInfo", "F1Info", "logisticModel")  
 invisible(results\_cv)  
   
 } else {  
 # LOOCV section (Leave-One-Out Cross-Validation)  
 cat("Running Leave-One-Out Cross-Validation...\n")  
 accuracyInfo <- numeric()  
 sensitivityInfo <- numeric()  
 specificityInfo <- numeric()  
 F1Info <- numeric()  
 predictions <- list()  
 cfMatList <- list()  
   
 for (i in 1:dim(data)[1]) {  
 # Perform LOOCV  
 trainDataCV <- setdiff(1:dim(data)[1], i)  
 testData <- data[i, , drop = FALSE]  
 trainLabels <- labels[trainDataCV]  
 testLabel <- labels[i]  
   
 logistic\_mod <- multinom(labelsTrain ~ ., data = data[trainDataCV, , drop = FALSE])  
 predictLabel <- predict(logistic\_mod, testData)  
   
 cfMatList[[i]] <- confusionMatrix(predictLabel, testLabel)  
 accuracyInfo[i] <- cfMatList[[i]]$overall[[1]]  
 sensitivityInfo[i] <- cfMatList[[i]]$byClass[[1]]  
 specificityInfo[i] <- cfMatList[[i]]$byClass[[2]]  
 F1Info[i] <- cfMatList[[i]]$byClass[[7]]  
 }  
   
 results\_loocv <- list(cfMatList, accuracyInfo, sensitivityInfo, specificityInfo, F1Info)  
 names(results\_loocv) <- c("cfMats", "accuracyInfo", "sensitivityInfo", "specificityInfo", "F1Info")  
 invisible(results\_loocv)  
 }  
}

Se muestra a continuación la implementación de la función logistic\_test\_multiclase. Para crearla nos hemos ayudado de la función svm\_test:

library(nnet)  
logistic\_test\_multiclase <- function(tren, etiquetasTren, prueba, etiquetasPrueba, vars\_selected, logistic\_trn\_mrmr) {  
   
 # Validar tipos de datos de los argumentos  
 if (!is.data.frame(tren) && !is.matrix(tren)) {  
 stop("El argumento tren debe ser un marco de datos o una matriz.")  
 }  
   
 if (dim(tren)[1] != length(etiquetasTren)) {  
 stop("La longitud de las filas del argumento tren debe ser la misma que la longitud de etiquetasTren.")  
 }  
   
 if (!is.character(etiquetasTren) && !is.factor(etiquetasTren)) {  
 stop("La clase del parámetro etiquetasTren debe ser un vector de caracteres o un factor.")  
 }  
 if (is.character(etiquetasTren)) { etiquetasTren <- as.factor(etiquetasTren) }  
   
 if (!is.character(etiquetasPrueba) && !is.factor(etiquetasPrueba)) {  
 stop("La clase del parámetro etiquetasPrueba debe ser un vector de caracteres o un factor.")  
 }  
 if (is.character(etiquetasPrueba)) { etiquetasPrueba <- as.factor(etiquetasPrueba) }  
   
 if (!is.data.frame(prueba) && !is.matrix(prueba)) {  
 stop("El argumento prueba debe ser un marco de datos o una matriz.")  
 }  
   
 if (dim(prueba)[1] != length(etiquetasPrueba)) {  
 stop("La longitud de las filas del argumento prueba debe ser la misma que la longitud de etiquetasPrueba.")  
 }  
   
 # Seleccionar las variables necesarias de los datos  
 tren <- as.data.frame(apply(tren, 2, as.double))  
 tren <- tren[, vars\_selected, drop = FALSE]  
   
 prueba <- as.data.frame(apply(prueba, 2, as.double))  
 prueba <- prueba[, vars\_selected, drop = FALSE]  
   
 # Normalizar las características entre -1 y 1  
 tren <- vapply(tren, function(x) {  
 max\_val <- max(x)  
 min\_val <- min(x)  
 if (max\_val > min\_val) {  
 x <- ((x - min\_val) / (max\_val - min\_val)) \* 2 - 1  
 }  
 return(x)  
 }, double(nrow(tren)))  
 tren <- as.data.frame(tren)  
   
 prueba <- vapply(prueba, function(x) {  
 max\_val <- max(x)  
 min\_val <- min(x)  
 if (max\_val > min\_val) {  
 x <- ((x - min\_val) / (max\_val - min\_val)) \* 2 - 1  
 }  
 return(x)  
 }, double(nrow(prueba)))  
 prueba <- as.data.frame(prueba)  
   
 # Inicializar vectores para almacenar los resultados  
 accVector <- double()  
 sensVector <- double()  
 specVector <- double()  
 f1Vector <- double()  
 cfMatList <- list()  
 predicVector <- list()  
   
 # Usar el modelo entrenado logistic\_trn\_mrmr$logisticModel$finalModel  
 best\_decay <- logistic\_trn\_mrmr$logisticModel$finalModel$decay  
   
 # Realizar predicciones en los datos de prueba usando el modelo entrenado  
 logistic\_model <- multinom(etiquetasTren~., data=subset(tren, select = c(colnames(tren)[1])), decay = best\_decay)  
 predicciones <- predict(logistic\_model, subset(prueba, select = c(colnames(prueba)[1])), type = "class")  
 predictScores <- predict(logistic\_model,subset(prueba, select = c(colnames(prueba)[1])), type = "prob")  
   
 # Calcular la matriz de confusión y las métricas  
 cfMat <- confusionMatrix(predicciones, etiquetasPrueba)  
 sens <- mean(cfMat$byClass[, "Sensitivity"])  
 spec <- mean(cfMat$byClass[, "Specificity"])  
 cfMat$byClass[, "F1"][is.na(cfMat$byClass[, "F1"])] <- 0  
 f1 <- mean(cfMat$byClass[, "F1"])  
   
 # Guardar los resultados para este conjunto de características  
 cfMatList[[1]] <- cfMat  
 accVector[1] <- cfMat$overall["Accuracy"]  
 sensVector[1] <- sens  
 specVector[1] <- spec  
 f1Vector[1] <- f1  
 predicVector[[1]] <- predictScores  
   
 # Si hay más variables, repetimos el proceso con más características  
 if (dim(prueba)[2] > 1) {  
 for (i in 2:dim(prueba)[2]) {  
 cat(paste("Probando con", i, "variables...\n", sep = " "))  
   
 logistic\_model2 <- multinom(etiquetasTren~., data=tren[,1:i], decay = best\_decay)  
 predicciones <- predict(logistic\_model2, prueba[, 1:i], type = "class")  
 predictScores <- predict(logistic\_model2, prueba[, 1:i], type = "prob")  
   
 cfMat <- confusionMatrix(predicciones, etiquetasPrueba)  
 sens <- mean(cfMat$byClass[, "Sensitivity"])  
 spec <- mean(cfMat$byClass[, "Specificity"])  
 cfMat$byClass[, "F1"][is.na(cfMat$byClass[, "F1"])] <- 0  
 f1 <- mean(cfMat$byClass[, "F1"])  
   
 cfMatList[[i]] <- cfMat  
 accVector[i] <- cfMat$overall["Accuracy"]  
 sensVector[i] <- sens  
 specVector[i] <- spec  
 f1Vector[i] <- f1  
 predicVector[[i]] <- predictScores  
 }  
 }  
   
 cat("¡Clasificación realizada con éxito!\n")  
   
 # Asignar nombres a los resultados  
 names(accVector) <- vars\_selected  
 names(sensVector) <- vars\_selected  
 names(specVector) <- vars\_selected  
 names(f1Vector) <- vars\_selected  
   
 # Devolver los resultados como una lista  
 resultados <- list(cfMatList, accVector, sensVector, specVector, f1Vector, predicVector)  
 names(resultados) <- c("cfMats", "accVector", "sensVector", "specVector", "f1Vector", "predicciones")  
   
 invisible(resultados)  
}

En estas funciones también se ha considerado que, si no es posible calcular el F1 para alguna clase e iteración debido a que no se predice ni correctamente ni erróneamente esa clase, el valor de dicho F1 será cero, lo que permite calcular adecuadamente la media de los tres F1 y tener en cuenta los F1 de las demás clases.

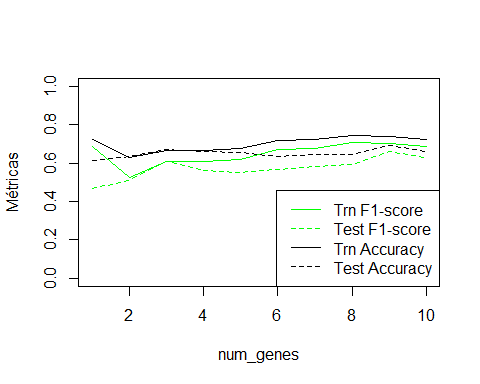
En esta sección, usaremos estas funciones específicas para entrenar y evaluar los modelos de regresión logística en este problema de clasificación multiclase.

### 3.4.1. Selección de variables con FSRankingMRMR

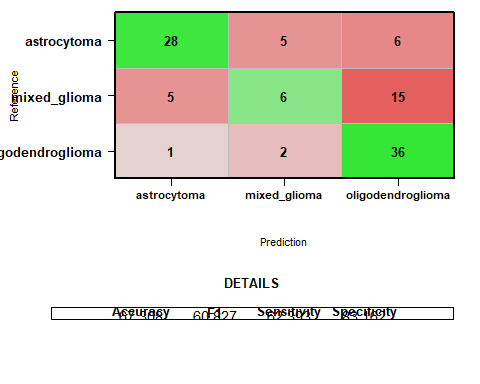
Utilizamos las funciones implementadas para entrenar los modelos logistic\_trn\_mrmr y logistic\_test\_mrmr, y calcular las métricas.

# Aplicar la función con regresión logística  
logistic\_trn\_mrmr <- logistic\_trn2(MLMatrix, MLLabels, vars\_selected = names(FSRankingMRMR[1:10]), numFold = 5)  
logistic\_results\_mrmr <- rbind(logistic\_trn\_mrmr$F1Info$meanF1[1:10],logistic\_trn\_mrmr$accuracyInfo$meanAccuracy)  
#dataPlot(svm\_results\_mrmr, MLLabels, legend = c("F1","Accuracy"), mode = "classResults", xlab="Genes", ylab="Prediction Score")

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score  
logistic\_test\_mrmr <- logistic\_test\_multiclase(MLMatrix, MLLabels, t(XTest), YTest, names(FSRankingMRMR[1:10]), logistic\_trn\_mrmr)  
  
#################################################################################3  
logistic\_results\_mrmr\_test <- rbind(logistic\_trn\_mrmr$F1Info$meanF1[1:10], logistic\_test\_mrmr$f1Vector, logistic\_trn\_mrmr$accuracyInfo$meanAccuracy, logistic\_test\_mrmr$accVector)



En este caso, para num\_genes = 3 las métricas comienzan a mejorar, tanto en train como en test, aunque sin un porcentaje muy alto y siendo similares en train y test. Se comprueba con la matriz de confusión para el conjunto test y una huella de 3 genes:

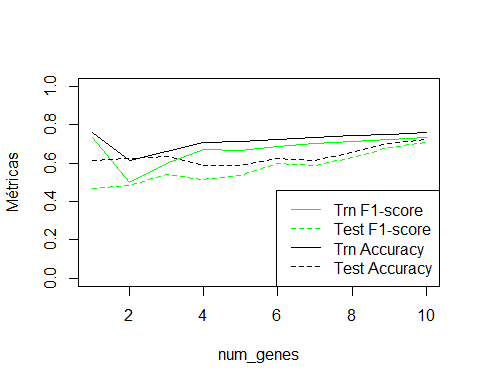
 Esta es la mejor predicción hasta el momento. Es interesante destacar que la clase que más se confunde es el glioma mixto con el oligodendroglioma. Esto puede deberse a que el glioma mixto presenta características del oligodendroglioma, lo que hace más difícil diferenciarlas en el modelo. Dado que ambas clases comparten ciertas similitudes en su perfil genético o características de imagen, la confusión es comprensible, lo que resalta la importancia de seguir perfeccionando los métodos de clasificación para mejorar la precisión.

### 3.4.2. Selección de variables con FSRankingRF

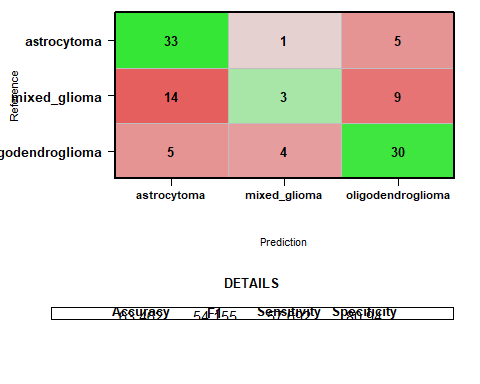
Ajustamos ahora los modelos de test y train para las variables seleccionadas por Random Forest:

# Aplicar la función con regresión logística  
logistic\_trn\_rf <- logistic\_trn2(MLMatrix, MLLabels, vars\_selected = (FSRankingRF[1:10]), numFold = 5)  
logistic\_results\_rf <- rbind(logistic\_trn\_rf$F1Info$meanF1[1:10],logistic\_trn\_rf$accuracyInfo$meanAccuracy)  
#dataPlot(svm\_results\_rf, MLLabels, legend = c("F1","Accuracy"), mode = "classResults", xlab="Genes", ylab="Prediction Score")

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score  
logistic\_test\_rf <- logistic\_test\_multiclase(MLMatrix, MLLabels, t(XTest), YTest, (FSRankingRF[1:10]), logistic\_trn\_rf)  
  
#################################################################################3  
logistic\_results\_rf\_test <- rbind(logistic\_trn\_rf$F1Info$meanF1[1:10], logistic\_test\_rf$f1Vector, logistic\_trn\_rf$accuracyInfo$meanAccuracy, logistic\_test\_rf$accVector)



Por un lado, a partir de 3 genes el F1 score y el accuracy se mantienen bastante constantes. Sin embargo, el porcentaje de acierto a partir de ese valor de genes no llega al 65% y, además, el F1 posteriormente deciende de forma leve. veamos así, la matriz de confusión para el conjunto test y una huella de 3 genes:



En contraste con el caso anterior, donde la mayor confusión se da entre el glioma mixto y el oligodendroglioma, en este caso, los que más se confunden son el glioma mixto y el astrocitoma. Esto puede ocurrir debido a que ambas clases presentan características similares, lo que dificulta que el modelo las distinga con precisión de nuevo.

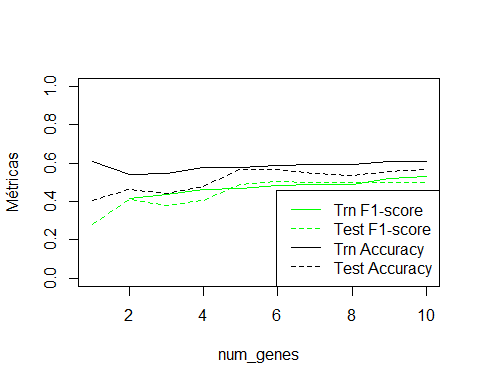
### 3.4.3. Selección de variables con FSRankingDA

Se estudia en esta sección la clasificación para el ranking creado con DA.

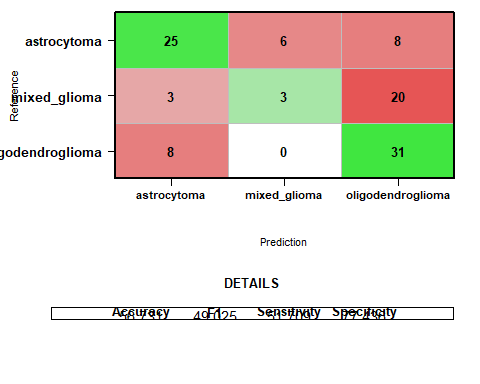
# Evaluar biomarcadores en TRN. Es multiclase, mejor ver accuracy y F1-Score  
logistic\_trn\_da<- logistic\_trn2(MLMatrix, MLLabels, vars\_selected = names(FSRankingDA[1:10]))

logistic\_results\_da <- rbind(logistic\_trn\_da$F1Info$meanF1[1:10],logistic\_trn\_da$accuracyInfo$meanAccuracy)  
#dataPlot(svm\_results\_da, MLLabels, legend = c("F1","Accuracy"), mode = "classResults", xlab="Genes", ylab="Prediction Score")

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score  
logistic\_test\_da <- logistic\_test\_multiclase(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected = names(FSRankingDA[1:10]), logistic\_trn\_da)  
logistic\_results\_da\_test <- rbind(logistic\_trn\_da$F1Info$meanF1[1:10], logistic\_test\_da$f1Vector, logistic\_trn\_da$accuracyInfo$meanAccuracy, logistic\_test\_da$accVector)



A partir de 5 genes, parece que las métricas de train y test dejan de tener altibajos. Destaca que estas métricas son más bajas que los resultados obtenidos con los rankings mRMR y RF. Veamos la matriz de confusión para el conjunto test para facilitar la obtención de conclusiones:



La bajada de precisión y de F1 score es debido a la gran confusión del glioma mixto con el oligodendroglioma.

### 3.4.3. Selección de variables con FSRanking\_MB

Por último, repitamos el estudio para el ranking de Markov Blanket, entrenando los modelos logistic\_trn\_mb y logistic\_test\_mb.

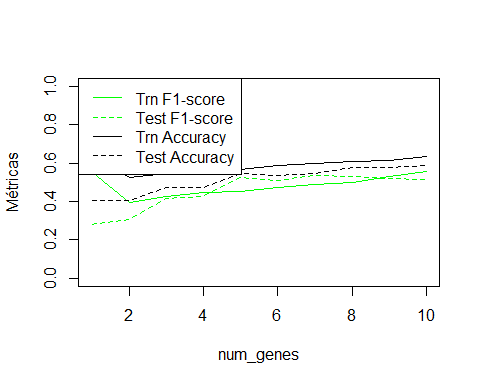
logistic\_trn\_mb <- logistic\_trn2(MLMatrix, MLLabels, vars\_selected = FSRanking\_MB[1:10])  
save(logistic\_trn\_mb, file='logistic\_trn\_mb')

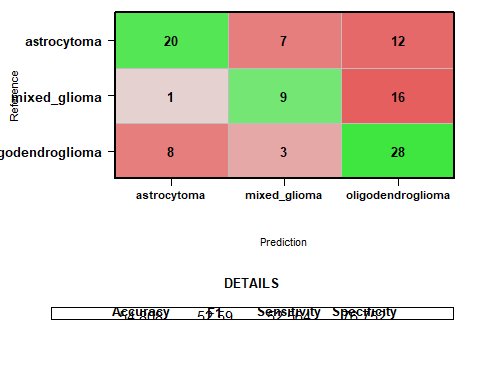
load('logistic\_trn\_mb')

logistic\_results\_mb <- rbind(logistic\_trn\_mb$F1Info$meanF1[1:10],logistic\_trn\_mb$accuracyInfo$meanAccuracy)  
logistic\_test\_mb <- logistic\_test\_multiclase(MLMatrix, MLLabels, t(XTest), YTest, vars\_selected = (FSRanking\_MB[1:10]), logistic\_trn\_mb)

## # weights: 9 (4 variable)  
## initial value 453.726875   
## final value 434.042469   
## converged  
## Probando con 2 variables...  
## # weights: 12 (6 variable)  
## initial value 453.726875   
## iter 10 value 408.412132  
## final value 408.412126   
## converged  
## Probando con 3 variables...  
## # weights: 15 (8 variable)  
## initial value 453.726875   
## iter 10 value 390.420303  
## final value 390.418100   
## converged  
## Probando con 4 variables...  
## # weights: 18 (10 variable)  
## initial value 453.726875   
## iter 10 value 382.647921  
## final value 382.397572   
## converged  
## Probando con 5 variables...  
## # weights: 21 (12 variable)  
## initial value 453.726875   
## iter 10 value 356.698841  
## final value 356.543732   
## converged  
## Probando con 6 variables...  
## # weights: 24 (14 variable)  
## initial value 453.726875   
## iter 10 value 352.990080  
## final value 352.682908   
## converged  
## Probando con 7 variables...  
## # weights: 27 (16 variable)  
## initial value 453.726875   
## iter 10 value 344.741155  
## iter 20 value 344.292651  
## final value 344.292635   
## converged  
## Probando con 8 variables...  
## # weights: 30 (18 variable)  
## initial value 453.726875   
## iter 10 value 338.274453  
## iter 20 value 337.530145  
## final value 337.530078   
## converged  
## Probando con 9 variables...  
## # weights: 33 (20 variable)  
## initial value 453.726875   
## iter 10 value 325.522723  
## iter 20 value 324.617187  
## final value 324.616956   
## converged  
## Probando con 10 variables...  
## # weights: 36 (22 variable)  
## initial value 453.726875   
## iter 10 value 318.929954  
## iter 20 value 317.560275  
## final value 317.558447   
## converged  
## ¡Clasificación realizada con éxito!

logistic\_results\_mb\_test <- rbind(logistic\_trn\_mb$F1Info$meanF1[1:10], logistic\_test\_mb$f1Vector, logistic\_trn\_mb$accuracyInfo$meanAccuracy, logistic\_test\_mb$accVector)

 De nuevo, parece adecuado escoger una huella de 5 genes ya que seha btenido una gráfica muy silimar a la anterior, pero no igual. Por ello, calculemos la matriz de confusión para el conjunto test y esta huella para visulizar los resultados con más detalle:



Es cuarioso ya que es una de la mejores clasificaciones de la clase glioma mixto. Sin emebargo, se confunde en bastantes ocasiones el glioma mixto y el astrocitoma con el oligodendroglioma.

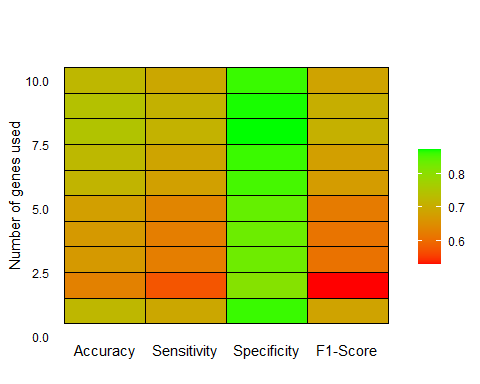
### 3.4.5. Conclusiones

En el caso del clasificador SVM, si seleccionamos características DA o con MB, se tiene un accuracy muy poco superior al 50%. Sin embargo, para el algoritmo de selección mRMR, el accuracy es del 67.3% con muy pocos genes, tres concretamente, y además, este es el porcentaje de acierto el más alto obtenido. También, con el seleccionador Random Forest se obtiene un accuracy para el conjunto test del 63.5%, devolviendo por tanto, una clasificación adecuada pero más pobre que para el mRMR. Por tanto, elegiremos en este caso como mejor seleccionador y huella el mRMR con 3 genes,

# 4. Resultados bajo el algoritmo de selección y clasificador escogidos en 5CV

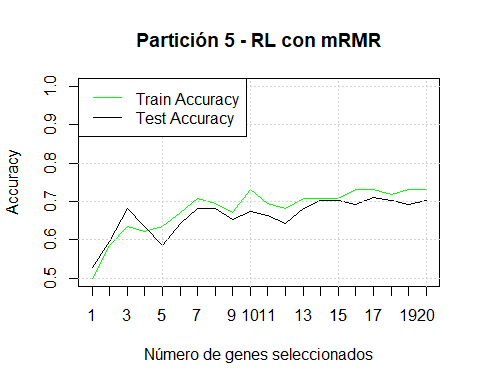
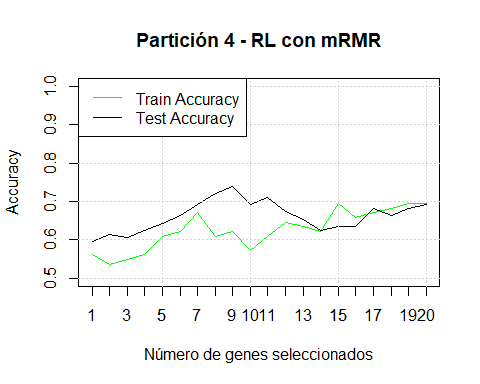
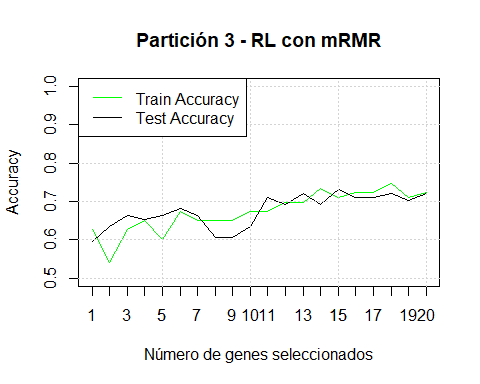
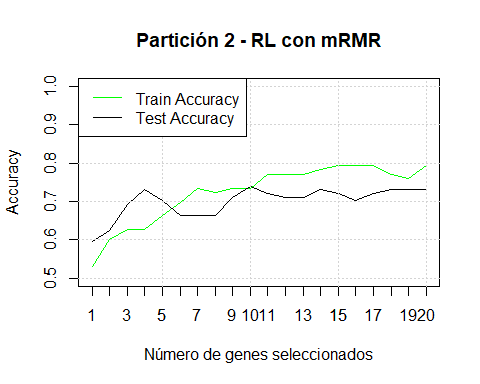
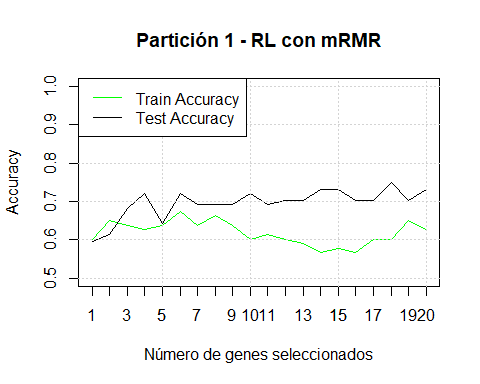
En la siguiente sección, se seleccionará la combinación óptima de algoritmo de selección y clasificador, con vista a la sección anterior, y se validará su rendimiento utilizando una validación cruzada de 5 particiones (5CV). Se evaluará el desempeño del modelo en los conjuntos de entrenamiento (Train) y prueba (Test) mediante gráficos que muestran la evolución del número de genes seleccionados y los resultados medios en las cinco particiones. Además, se proporcionarán los cinco rankings de genes más relevantes y se escogerá la huella génica final basada en su consistencia y relevancia.

En primer lugar, el algoritmo de selección de variables elegido es el **mRMR** y el clasificador será **Regresión Logística múltime con penalización**; ya que es con el que se ha obtenido mejores valores de Accuracy, F1, sensibilidad y especificidad.

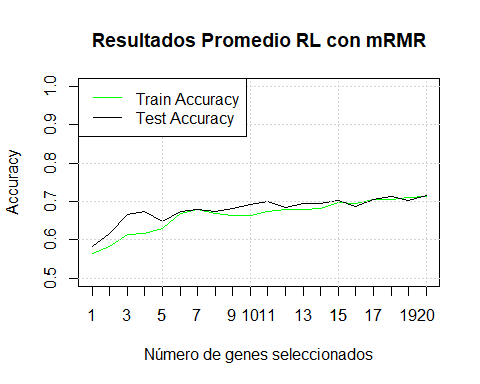


En segundo lugar, se presenta el siguiente código con el objetivo de mostar resultado, para el modelo de regresión logística con una penalización de (la objetenida con el modelo logistic\_trn\_mrmr en la sección anterior) y utilizando el seleccionador de características mRMR, en 5CV Trn-Test para todo el dataset:

set.seed(19)  
  
nfolds <- 5 # Número de particiones en la validación cruzada  
foldIDx <- cvGenStratified(MLLabels, nfolds) # Particiones estratificadas  
  
nVarsMAX <- 20 # Número máximo de genes a seleccionar, en un ejercicio de clase se aumentó hasta 50  
  
# Matrices para almacenar métricas por partición  
ACCTrnRL <- matrix(0, nfolds, nVarsMAX)  
ACCTestRL <- matrix(0, nfolds, nVarsMAX)  
  
  
# Para almacenar los ranking de características de cada partición  
rankingSMRMR <- matrix(0, nfolds, nVarsMAX)   
  
# El Mejor hiperparámetro decay obtenido en la sección enterior  
best\_decay<-logistic\_trn\_mrmr$logisticModel$finalModel$decay   
print(best\_decay)  
  
# Bucle para 5 particiones  
for (particion in seq(1, nfolds)) {  
 # Crear particiones de entrenamiento y prueba  
 datosParaTRN <- which(foldIDx != particion) # Índices de entrenamiento  
 segmentoTest <- which(foldIDx == particion) # Índices de prueba  
   
 XTrn\_P <- as.data.frame( MLMatrix[datosParaTRN, ] ) # Datos de entrenamiento  
 XTest\_P <- as.data.frame(MLMatrix[segmentoTest, ]) # Datos de prueba  
   
 YTrn\_P <- as.factor(MLLabels[datosParaTRN]) # Etiquetas de entrenamiento  
 YTest\_P <- as.factor(MLLabels[segmentoTest]) # Etiquetas de prueba  
   
 # Ranking con mRMR  
 rankingMRMR <-featureSelection(XTrn\_P, YTrn\_P, mode = "mrmr", vars\_selected = colnames(XTrn\_P))  
   
 # Guardar POR FILAS el ranking de características para esta partición   
 rankingSMRMR[particion, ] <- rankingMRMR[1:nVarsMAX]  
  
   
 # Evaluación por número de genes seleccionados  
 for (numGenes in seq\_len(nVarsMAX)) {  
 # Seleccionar las mejores variables  
 topGenes <- colnames(XTrn\_P)[rankingMRMR[1:numGenes]]  
   
 # Entrenar modelo Regresión Logística multinomial  
 modelo <- multinom(YTrn\_P ~ ., data = subset(XTrn\_P, select=topGenes), decay = best\_decay)  
   
 # Predicciones en Train  
 predTrain <- predict(modelo, subset(XTest\_P, select=topGenes))  
 confMatTrain <- confusionMatrix(as.factor(predTrain), as.factor(YTest\_P))  
 ACCTrnRL[particion, numGenes] <- confMatTrain$overall["Accuracy"]  
   
 # Predicciones en Test  
 predTest <- predict(modelo, subset(t(XTest), select=topGenes))  
 confMatTest <- confusionMatrix(as.factor(predTest), as.factor(YTest))   
 ACCTestRL[particion, numGenes] <- confMatTest$overall["Accuracy"] # se van añadiendo los valores por filas. Por tanto, para calcular la media de los accuracy para un número determinado de genes que hacerlo por columnas. #print(confMatTest$overall[1])  
 #print(confMatTest$table)  
 }  
   
 # Graficar los resultados para la partición actual  
 num\_genes <- 1:nVarsMAX  
 Trn\_Acc <- ACCTrnRL[particion, 1:nVarsMAX]  
 Test\_Acc <- ACCTestRL[particion, 1:nVarsMAX]  
   
 plot(num\_genes, Trn\_Acc, type = "l", col = "green", ylim = c(0.5, 1), ylab = "Accuracy", xlab = "Número de genes seleccionados", main = paste("Partición", particion, "- RL con mRMR"))  
 lines(num\_genes, Test\_Acc, col = "black")  
 axis(1, at = 1:20, labels = 1:20) # Configura los ticks y etiquetas del eje X  
 grid()  
 legend("topleft", legend = c("Train Accuracy", "Test Accuracy"),col=c("green","black"),lty = c(1, 1))  
}



# Medias de Accuracy en las 5 particiones  
ACCTrnRL\_Mean <- colMeans(ACCTrnRL)  
ACCTestRL\_Mean <- colMeans(ACCTestRL)  
  
# Resultados promedio  
plot(1:nVarsMAX, ACCTrnRL\_Mean, type = "l", col = "green", ylim = c(0.5, 1), xlim=c(1,20), xlab = "Número de genes seleccionados", ylab = "Accuracy", main = "Resultados Promedio RL con mRMR")  
lines(1:nVarsMAX, ACCTestRL\_Mean, col = "black")  
axis(1, at = 1:20, labels = 1:20) # Configura los ticks y etiquetas del eje X  
grid()  
legend("topleft", legend = c("Train Accuracy", "Test Accuracy"), col = c("green", "black"),  
 lty = c(1, 1))



De todas las gráficas destaca que los porcentajes de acierto oscilan entre el 60% y el 80% cuando se eligen huellas desde 1 a 20 genes. En general, para el conjunto train el accuracy es más alto y observando la última gráfica observamos que, en media, con una huella de 5 genes ya se supera el 70% de acierto. Observamos, para más precisión, las variables ACCTrnRL\_Mean y ACCTestRL\_Mean que almacenan la media de los accuracy de cada iteración para el conjunto train y test, respectivamente:

ACCTrnRL\_Mean

## [1] 0.5640024 0.5834264 0.6148986 0.6173083 0.6295034 0.6681751 0.6804290  
## [8] 0.6682045 0.6633559 0.6633852 0.6730238 0.6803115 0.6803409 0.6827211  
## [15] 0.6973553 0.6949163 0.7045842 0.7045842 0.7094622 0.7142815

ACCTestRL\_Mean

## [1] 0.5826923 0.6173077 0.6653846 0.6730769 0.6480769 0.6750000 0.6788462  
## [8] 0.6730769 0.6807692 0.6923077 0.7000000 0.6846154 0.6942308 0.6961538  
## [15] 0.7038462 0.6884615 0.7057692 0.7134615 0.7019231 0.7153846

Podemos ver que, a partir de la selección de 5 genes, la mayoría de los valores de accuracy superan el 70%, lo que indica una buena clasificación en el conjunto de entrenamiento, aunque un poco más baja en el conjunto de test. Por eso, según la última gráfica, una buena opción sería elegir una huella de 3 o 5 genes. Si miramos los valores de accuracy en el vector ACCTrnSVM\_Mean y en la gráfica, notamos que después de los 5 genes, los aumentos en accuracy son más pequeños y progresivos. En cambio, la diferencia es más notoria cuando se comparan 5 genes o menos. Así que, como 5 genes ofrecen un buen balance entre precisión y simplicidad, se eligen para la huella.

A continuación, podemos mostrar los 5 ranking y los 5 primeros genes seleccionados en las 5 particiones del bucle utilizado para buscar la huella, en la sección anterior:

# Se observan la variable rankingSMRMR  
rankingSMRMR[,1:5]

## [,1] [,2] [,3] [,4] [,5]  
## [1,] 170 81 51 138 41  
## [2,] 170 51 81 10 69  
## [3,] 170 10 51 105 28  
## [4,] 170 62 81 16 21  
## [5,] 81 51 170 34 7

# rankingMRMR de la última iteración pero solo lo utilizamos para saber el nombre de los genes  
colnames(MLMatrix)[c(170, 81, 51, 41, 10)]

## [1] "IGHV3-73" "IGHV6-1" "UBQLN4P1" "RAPH1" "PI4K2B"

Debemos estudiar esta matriz por filas, pues corresponden a cada iteración y las columnas son los primero 5 genes del ranking.

Los 5 genes candidatos a ser la huella son los más frecuentes en las 5 iteraciones, a saber: el 170 o **IGHV3-73**, el 81 o **IGHV6-1**, el 51 o **UBQLN4P1**; estos son los primeros genes seleccionados en todos los rankings y el 10 o **PI4K2B**; seleccioandos en 3 de las 5 iteraciones.

# Se observan la variable rankingSMRMR  
rankingSMRMR[,1:10]

## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
## [1,] 170 81 51 138 41 22 194 158 95 28  
## [2,] 170 51 81 10 69 124 41 105 16 106  
## [3,] 170 10 51 105 28 106 41 181 81 16  
## [4,] 170 62 81 16 21 10 51 127 139 55  
## [5,] 81 51 170 34 7 22 10 95 106 41

# rankingMRMR de la última iteración pero solo lo utilizamos para saber el nombre de los genes  
colnames(MLMatrix)[c(170, 81, 51, 41, 10)]

## [1] "IGHV3-73" "IGHV6-1" "UBQLN4P1" "RAPH1" "PI4K2B"

El 41 o **RAPH1**

## 4.1. Resultado de la huella final escogida y clasificador en 5 CV, matriz de confusión final del dataset.

Podemos comenzar esta sección calculando la matriz final del modelo conocida la huella volviendo a probar en 5CV con el número de genes escogidos, es decir, *numGenes <- 5*:

set.seed(19)  
  
nfolds <- 5 # Número de particiones en la validación cruzada  
foldIDx <- cvGenStratified(LABELS, nfolds) # Particiones estratificadas  
  
# Vectores para almacenar métricas por partición  
ACCTrnRL <- numeric(nfolds)  
ACCTestRL <- numeric(nfolds)  
  
# Matriz de confusión acumulada para el test  
accTestSum <- matrix(0, nrow = 3, ncol = 3)  
  
# Ranking con mRMR  
huella <- c("IGHV3-73", "IGHV6-1", "UBQLN4P1", "PI4K2B", "RAPH1")  
  
# Bucle para 5 particiones  
for (particion in seq\_len(nfolds)) {  
 # Crear particiones de entrenamiento y prueba  
 datosParaTRN <- which(foldIDx != particion) # Índices de entrenamiento  
 segmentoTest <- which(foldIDx == particion) # Índices de prueba  
   
 XTrn\_P <- as.data.frame(t(MATRIZ[,datosParaTRN])) # Datos de entrenamiento  
 XTest\_P <- as.data.frame(t(MATRIZ[,segmentoTest])) # Datos de prueba  
   
 YTrn\_P <- as.factor(LABELS[datosParaTRN]) # Etiquetas de entrenamiento  
 YTest\_P <- as.factor(LABELS[segmentoTest]) # Etiquetas de prueba  
   
   
 # Entrenar modelo Regresión Logística multinomial  
 modelo <- multinom(YTrn\_P ~ ., data = subset(XTrn\_P, select=huella), decay = best\_decay)  
   
 # Predicciones en Test  
 predTest <- predict(modelo, XTest\_P[, huella])  
 confMatTest <- confusionMatrix(as.factor(predTest), YTest\_P)  
 #print(confMatTest$table)  
 ACCTestRL[particion] <- confMatTest$overall["Accuracy"]  
 #print(ACCTestSVM[particion])  
   
 # Sumar la matriz de confusión al total  
 accTestSum <- accTestSum + confMatTest$table  
}

print("Matriz de confusión acumulada en el test:")

## [1] "Matriz de confusión acumulada en el test:"

accTestSum

## Reference  
## Prediction astrocytoma mixed\_glioma oligodendroglioma  
## astrocytoma 162 34 8  
## mixed\_glioma 17 38 32  
## oligodendroglioma 13 60 153

La matriz de confusión muestra que el modelo tiene un buen desempeño clasificando astrocytoma con 125 predicciones correctas y oligodendroglioma con 114 correctas, concluyendo que la huella elegida es adecuada. Sin embargo, presenta confusión con mixed glioma, que se clasifica erróneamente como astrocytoma en 17 ocasiones y como oligodendroglioma en 24. Creo que esto puede deberse a que mixed glioma comparte características similares con las otras dos clases, lo que dificulta al modelo identificar sus diferencias de manera clara. Además, el conjunto de datos no contiene tantas muestras de este tipo de cancer como del resto, lo que podría hacer que el modelo tenga dificultades para aprender los patrones distintivos de esta clase.

table(LABELS)

## LABELS  
## astrocytoma mixed\_glioma oligodendroglioma   
## 192 132 193

Finalmente, con estos cinco genes seleccionados, se harán dos particiones sobre MLMatrix y MLLabels, train (80%) y test (20%) con el objetivo de entrenar un modelo regresión logística y buscar los mejores parámetros para este problema con un GridSearch. Para concluir esta sección se predecirán las clases de conjuto test y evaluaremos el desempeño del modelo con una matriz de confusión:

# Filtrar los datos para incluir solo los genes seleccionados  
X\_selected <- as.data.frame( MLMatrix[, c("IGHV3-73", "IGHV6-1", "UBQLN4P1", "PI4K2B", "RAPH1")])# Seleccionar columnas correspondientes  
Y <- as.factor(MLLabels) # Etiquetas de clase  
  
# Crear particiones: 80% Train, 20% Test  
set.seed(19)   
  
trainIndex <- createDataPartition(Y, p = 0.8, list = FALSE)  
X\_train <- X\_selected[trainIndex, ]  
Y\_train <- Y[trainIndex]  
X\_test <- X\_selected[-trainIndex, ]  
Y\_test <- Y[-trainIndex]  
  
# GridSearch para Regresión Logística Multinomial: Definir el parámetro decay a probar  
grid <- expand.grid(.decay = c(0.0001, 0.01, 0.1, 1))  
  
# Entrenar el modelo con validación cruzada  
train\_control <- trainControl(method = "cv", number = 5) # Validación cruzada de 5 pliegues  
  
# Modelo de Regresión Logística Multinomial  
logistic\_model <- train(x = X\_train, y = Y\_train, method = "multinom", trControl = train\_control, tuneGrid = grid)

## # weights: 21 (12 variable)  
## initial value 292.230869   
## iter 10 value 178.995699  
## iter 20 value 176.396907  
## iter 30 value 176.265189  
## iter 40 value 175.561278  
## iter 50 value 175.260441  
## final value 175.260420   
## converged  
## # weights: 21 (12 variable)  
## initial value 292.230869   
## iter 10 value 180.611227  
## iter 20 value 180.393705  
## final value 180.393339   
## converged  
## # weights: 21 (12 variable)  
## initial value 292.230869   
## iter 10 value 191.388376  
## final value 191.355087   
## converged  
## # weights: 21 (12 variable)  
## initial value 292.230869   
## iter 10 value 228.259320  
## final value 228.219255   
## converged  
## # weights: 21 (12 variable)  
## initial value 291.132256   
## iter 10 value 173.841212  
## iter 20 value 172.617912  
## iter 30 value 172.004136  
## iter 40 value 171.477849  
## iter 50 value 171.219749  
## final value 171.219519   
## converged  
## # weights: 21 (12 variable)  
## initial value 291.132256   
## iter 10 value 175.805093  
## iter 20 value 175.611127  
## final value 175.610652   
## converged  
## # weights: 21 (12 variable)  
## initial value 291.132256   
## iter 10 value 188.214657  
## final value 188.137851   
## converged  
## # weights: 21 (12 variable)  
## initial value 291.132256   
## iter 10 value 227.692026  
## final value 227.659592   
## converged  
## # weights: 21 (12 variable)  
## initial value 292.230869   
## iter 10 value 182.071348  
## iter 20 value 178.851172  
## iter 30 value 178.384083  
## iter 40 value 177.511647  
## iter 50 value 177.142907  
## final value 177.142150   
## converged  
## # weights: 21 (12 variable)  
## initial value 292.230869   
## iter 10 value 183.485600  
## iter 20 value 183.133148  
## final value 183.132860   
## converged  
## # weights: 21 (12 variable)  
## initial value 292.230869   
## iter 10 value 193.239804  
## final value 193.129556   
## converged  
## # weights: 21 (12 variable)  
## initial value 292.230869   
## iter 10 value 228.226948  
## final value 228.170599   
## converged  
## # weights: 21 (12 variable)  
## initial value 292.230869   
## iter 10 value 176.903566  
## iter 20 value 173.902012  
## iter 30 value 173.391968  
## iter 40 value 172.690648  
## iter 50 value 172.500712  
## iter 50 value 172.500710  
## iter 50 value 172.500710  
## final value 172.500710   
## converged  
## # weights: 21 (12 variable)  
## initial value 292.230869   
## iter 10 value 178.564965  
## iter 20 value 178.376981  
## iter 30 value 178.376835  
## iter 30 value 178.376835  
## iter 30 value 178.376835  
## final value 178.376835   
## converged  
## # weights: 21 (12 variable)  
## initial value 292.230869   
## iter 10 value 189.531105  
## final value 189.512853   
## converged  
## # weights: 21 (12 variable)  
## initial value 292.230869   
## iter 10 value 226.946145  
## final value 226.894913   
## converged  
## # weights: 21 (12 variable)  
## initial value 291.132256   
## iter 10 value 181.247908  
## iter 20 value 179.080627  
## iter 30 value 178.967748  
## iter 40 value 178.331334  
## iter 50 value 177.998045  
## final value 177.997998   
## converged  
## # weights: 21 (12 variable)  
## initial value 291.132256   
## iter 10 value 182.883031  
## iter 20 value 182.697180  
## final value 182.696903   
## converged  
## # weights: 21 (12 variable)  
## initial value 291.132256   
## iter 10 value 193.677157  
## final value 193.665677   
## converged  
## # weights: 21 (12 variable)  
## initial value 291.132256   
## iter 10 value 230.021027  
## final value 229.987756   
## converged  
## # weights: 21 (12 variable)  
## initial value 364.739280   
## iter 10 value 224.443248  
## iter 20 value 221.721788  
## iter 30 value 220.889334  
## iter 40 value 219.906000  
## iter 50 value 219.401622  
## iter 50 value 219.401622  
## iter 50 value 219.401622  
## final value 219.401622   
## converged

# Mejores hiperparámetros encontrados  
best\_params <- logistic\_model$bestTune  
best\_params

## decay  
## 1 1e-04

# Entrenar el modelo con el mejor parámetro decay  
final\_model <- multinom(Y\_train ~ ., data = cbind(X\_train, Y\_train), decay = best\_params$decay)

## # weights: 21 (12 variable)  
## initial value 364.739280   
## iter 10 value 224.443248  
## iter 20 value 221.721788  
## iter 30 value 220.889334  
## iter 40 value 219.906000  
## iter 50 value 219.401622  
## iter 50 value 219.401622  
## iter 50 value 219.401622  
## final value 219.401622   
## converged

# Predicciones en el conjunto de prueba  
pred\_test <- predict(final\_model, X\_test)  
conf\_matrix <- confusionMatrix(pred\_test, Y\_test)  
conf\_matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction astrocytoma mixed\_glioma oligodendroglioma  
## astrocytoma 28 2 3  
## mixed\_glioma 2 9 5  
## oligodendroglioma 0 10 22  
##   
## Overall Statistics  
##   
## Accuracy : 0.7284   
## 95% CI : (0.6181, 0.8213)  
## No Information Rate : 0.3704   
## P-Value [Acc > NIR] : 6.36e-11   
##   
## Kappa : 0.5832   
##   
## Mcnemar's Test P-Value : 0.1979   
##   
## Statistics by Class:  
##   
## Class: astrocytoma Class: mixed\_glioma  
## Sensitivity 0.9333 0.4286  
## Specificity 0.9020 0.8833  
## Pos Pred Value 0.8485 0.5625  
## Neg Pred Value 0.9583 0.8154  
## Prevalence 0.3704 0.2593  
## Detection Rate 0.3457 0.1111  
## Detection Prevalence 0.4074 0.1975  
## Balanced Accuracy 0.9176 0.6560  
## Class: oligodendroglioma  
## Sensitivity 0.7333  
## Specificity 0.8039  
## Pos Pred Value 0.6875  
## Neg Pred Value 0.8367  
## Prevalence 0.3704  
## Detection Rate 0.2716  
## Detection Prevalence 0.3951  
## Balanced Accuracy 0.7686

En conclusión, el mejor valor de deacy es 0.0001, lo que ha permitido obtener una precisión del 65.43% en el conjunto de prueba. Este rendimiento, aunque no es extremadamente alto, aunque refleja un desempeño razonable al clasificar las tres clases. En cuanto a los estadísticos por clase, la sensibilidad es particularmente alta en astrocytoma con un 93.33%, lo que indica que el modelo es eficaz para identificar correctamente los positivos en esta clase, al igual que para la clase oligodendroglioma con un 80% de acierto. Por ello, el Balanced Accuracy, que considera tanto la sensibilidad como la especificidad, muestra buenos valores, especialmente en astrocytoma (83.92%) y oligodendroglioma (77.25%). Sin embargo, el valor predictivo positivo es algo bajo en mixed\_glioma (33.33%), pero aún se mantiene a un buen nivel en astrocytoma (68.29%) y oligodendroglioma (64.86%). Así, es importante destacar que mixed\_glioma presenta un desafío, con una sensibilidad baja (4.76%) y un rendimiento general más débil, lo que se refleja en los errores de clasificación en la matriz de confusión.

# 5. Enriquecimiento de los genes de la huella escogida

Por último, en esta sección, se realizará un análisis de enriquecimiento funcional de los tres genes seleccionados en la huella final (**IGHV3-73**, **IGHV6-1**, **UBQLN4P1**, **PI4K2B** y **RAPH1**), identificando su participación en procesos biológicos (Gene Ontology - GO), rutas metabólicas y celulares (Pathways) y su asociación con enfermedades (Disease Association). Los resultados obtenidos permitirán no solo diferenciar con mayor precisión las muestras de LUAD, LUSC y tejido sano, sino también identificar potenciales biomarcadores que puedan contribuir a futuros estudios sobre el diagnóstico y tratamiento del cáncer de pulmón.

Primero, se utiliza la función “*getGenesAnnotationpara*” mapear los nombres de los genes (**IGHV3-73**, **IGHV6-1**, **UBQLN4P1**, **PI4K2B** y **RAPH1**) a identificadores Entrez Gene. Estos identificadores son necesarios para muchos análisis funcionales, ya que son un estándar en bases de datos biológicos como GO, pathways y diseases que se verán a continuación.

#ENRIQUECIMIENTO FUNCIONAL:  
entrezAnnotation <- getGenesAnnotation(c("IGHV3-73", "IGHV6-1", "UBQLN4P1", "PI4K2B", "RAPH1"), attributes = c("external\_gene\_name","entrezgene\_id"), filter = "external\_gene\_name")  
entrezGeneIds<- entrezAnnotation$entrezgene\_id[!is.na(entrezAnnotation$entrezgene\_id)]

entrezAnnotation

## external\_gene\_name entrezgene\_id  
## 3003 IGHV6-1 NA  
## 3246 IGHV3-73 NA  
## 23023 PI4K2B 55300  
## 51715 IGHV6-1 NA  
## 51954 IGHV3-73 NA  
## 59494 UBQLN4P1 NA  
## 63767 RAPH1 65059

Así, el gen **PI4K2B** y el **RAPH1** serán los únicos que no se ignore por tener identificador Entrez y ser los únicos que se mapean. Con estos genes pasemos al análisis de enriquecimiento funcional utilizando Gene Ontology:

# Se descarga informacion sobre los Gene Ontology  
GOs <- geneOntologyEnrichment\_updated(as.character(entrezGeneIds), geneType = "ENTREZ\_GENE\_ID")  
save(GOs, file='GOs')

load('GOs')  
GOs

## ONTOLOGY ID  
## GO:0046854 BP GO:0046854  
## GO:0007032 BP GO:0007032  
## GO:0048675 BP GO:0048675  
## GO:0006661 BP GO:0006661  
## GO:0046488 BP GO:0046488  
## GO:0007030 BP GO:0007030  
## GO:1990138 BP GO:1990138  
## GO:0046474 BP GO:0046474  
## GO:0048588 BP GO:0048588  
## GO:0060560 BP GO:0060560  
## GO:0045017 BP GO:0045017  
## GO:0008654 BP GO:0008654  
## GO:0006650 BP GO:0006650  
## GO:0016050 BP GO:0016050  
## GO:0006644 BP GO:0006644  
## GO:0046486 BP GO:0046486  
## GO:0007409 BP GO:0007409  
## GO:0016049 BP GO:0016049  
## GO:0030175 CC GO:0030175  
## GO:0031901 CC GO:0031901  
## GO:0030027 CC GO:0030027  
## GO:0098858 CC GO:0098858  
## GO:0005802 CC GO:0005802  
## GO:0098791 CC GO:0098791  
## GO:0031252 CC GO:0031252  
## GO:0005769 CC GO:0005769  
## GO:0052742 MF GO:0052742  
## GO:0001727 MF GO:0001727  
## Description GeneRatio  
## GO:0046854 phosphatidylinositol phosphate biosynthetic process 1/2  
## GO:0007032 endosome organization 1/2  
## GO:0048675 axon extension 1/2  
## GO:0006661 phosphatidylinositol biosynthetic process 1/2  
## GO:0046488 phosphatidylinositol metabolic process 1/2  
## GO:0007030 Golgi organization 1/2  
## GO:1990138 neuron projection extension 1/2  
## GO:0046474 glycerophospholipid biosynthetic process 1/2  
## GO:0048588 developmental cell growth 1/2  
## GO:0060560 developmental growth involved in morphogenesis 1/2  
## GO:0045017 glycerolipid biosynthetic process 1/2  
## GO:0008654 phospholipid biosynthetic process 1/2  
## GO:0006650 glycerophospholipid metabolic process 1/2  
## GO:0016050 vesicle organization 1/2  
## GO:0006644 phospholipid metabolic process 1/2  
## GO:0046486 glycerolipid metabolic process 1/2  
## GO:0007409 axonogenesis 1/2  
## GO:0016049 cell growth 1/2  
## GO:0030175 filopodium 1/2  
## GO:0031901 early endosome membrane 1/2  
## GO:0030027 lamellipodium 1/2  
## GO:0098858 actin-based cell projection 1/2  
## GO:0005802 trans-Golgi network 1/2  
## GO:0098791 Golgi apparatus subcompartment 1/2  
## GO:0031252 cell leading edge 1/2  
## GO:0005769 early endosome 1/2  
## GO:0052742 phosphatidylinositol kinase activity 1/2  
## GO:0001727 lipid kinase activity 1/2  
## BgRatio pvalue p.adjust qvalue geneID Count  
## GO:0046854 66/18870 0.006983182 0.040110835 0.002345663 55300 1  
## GO:0007032 96/18870 0.010149267 0.040110835 0.002345663 55300 1  
## GO:0048675 121/18870 0.012783809 0.040110835 0.002345663 65059 1  
## GO:0006661 131/18870 0.013836643 0.040110835 0.002345663 55300 1  
## GO:0046488 155/18870 0.016361153 0.040110835 0.002345663 55300 1  
## GO:0007030 157/18870 0.016571383 0.040110835 0.002345663 55300 1  
## GO:1990138 179/18870 0.018882428 0.040110835 0.002345663 65059 1  
## GO:0046474 210/18870 0.022134285 0.040110835 0.002345663 55300 1  
## GO:0048588 239/18870 0.025171459 0.040110835 0.002345663 65059 1  
## GO:0060560 244/18870 0.025694632 0.040110835 0.002345663 65059 1  
## GO:0045017 250/18870 0.026322254 0.040110835 0.002345663 55300 1  
## GO:0008654 254/18870 0.026740557 0.040110835 0.002345663 55300 1  
## GO:0006650 298/18870 0.031335954 0.043388244 0.002537324 55300 1  
## GO:0016050 366/18870 0.038416541 0.046023042 0.002691406 55300 1  
## GO:0006644 375/18870 0.039351732 0.046023042 0.002691406 55300 1  
## GO:0046486 390/18870 0.040909371 0.046023042 0.002691406 55300 1  
## GO:0007409 448/18870 0.046920352 0.049680372 0.002905285 65059 1  
## GO:0016049 497/18870 0.051983869 0.051983869 0.003039992 65059 1  
## GO:0030175 108/19886 0.010832689 0.041407187 NA 65059 1  
## GO:0031901 195/19886 0.019516120 0.041407187 NA 55300 1  
## GO:0030027 201/19886 0.020113566 0.041407187 NA 65059 1  
## GO:0098858 227/19886 0.022700396 0.041407187 NA 65059 1  
## GO:0005802 259/19886 0.025879492 0.041407187 NA 55300 1  
## GO:0098791 382/19886 0.038050931 0.042484778 NA 55300 1  
## GO:0031252 423/19886 0.042091073 0.042484778 NA 65059 1  
## GO:0005769 427/19886 0.042484778 0.042484778 NA 55300 1  
## GO:0052742 26/18496 0.002809519 0.004428597 NA 55300 1  
## GO:0001727 41/18496 0.004428597 0.004428597 NA 55300 1

En la tabla anterior se han mostrado los términos GO asosiados al gen **PI4K2B** y al gen **RAPH1**, los cuales están asociado principalmente con procesos biológico (BP):

* Asociados al **PI4K2B**: el proceso biosintético del fosfatidilinositol y del fosfatidilinositol fosfato, la organización de vesículas y la organización del Golgi.
* Asociados al **RAPH1**: axogénesis (el proceso mediante el cual las neuronas se desarrollan y extienden sus axones ), crecimiento del desarrollo involucrado en la morfogénesis y extensión de las proyecciones neuronales.

El gen gen **PI4K2B** también se asocia con funciones moleculares (MF), como por ejemplo:

* La actividad de la quinasa de fosfatidilinositol.
* La actividad de la quinasa lipídica.

Y, tanto **PI4K2B** como **RAPH1**, se asocian con componentes celulares (CC):

* Asociados con **PI4K2B**: Red del Golgi-trans: la red de vesículas que clasifica y envía proteínas y lípidos a sus destinos, subcompartimento del aparato de Golgi. endosoma temprano, actividad de quinasa de fosfatidilinositol: actividad enzimática responsable de la fosforilación del fosfatidilinositol, un lípido involucrado en la transducción de señales.
* Asociados con **RAPH1**: borde principal celular, lamelipodio y filopodio.

Por otro lado, veamos los genes más enriquecidos observando la variable de GOs: **GeneRatio**. Sin embargo, esta variable vale en todos los casos 1/2 luego no podemos llegar a ninguna conclusión, a priori.

Otra posibilidad es crear un vector con los p-values ajustados ordenados de menor a mayor. Cuanto menor sea el **p.ajust**, más significativo será el término GO. Se muestran, entonces, los cinco términos más enriquecidos:

sort(GOs$p.adjust, decreasing = F)[1:5]

## [1] 0.004428597 0.004428597 0.040110835 0.040110835 0.040110835

posiciones<-order(GOs$p.adjust, decreasing = F)[1:5]  
posiciones

## [1] 27 28 1 2 3

rownames(GOs)[posiciones][1:5]

## [1] "GO:0052742" "GO:0001727" "GO:0046854" "GO:0007032" "GO:0048675"

GOs$Description[posiciones]

## [1] "phosphatidylinositol kinase activity"   
## [2] "lipid kinase activity"   
## [3] "phosphatidylinositol phosphate biosynthetic process"  
## [4] "endosome organization"   
## [5] "axon extension"

GOs$geneID[posiciones]

## [1] "55300" "55300" "55300" "55300" "65059"

La mayoría de los genes enriquecidos están asocidos al **PI4K2B**.

En segundo lugar, veamos la función de enriquecimiento funcional: DEGsToPathways, la cual descarga información sobre los Pathways, es decir, rutas metabólicas y celulares:

pathways <- DEGsToPathways(entrezAnnotation$external\_gene\_name)  
pathways\_df <- data.frame(  
 Kegg\_path = unlist(pathways$KEGG\_Path),  
 Name = unlist(pathways$Name),  
 Description = unlist(pathways$Description),  
 Class = unlist(pathways$Class),  
 Genes = unlist(pathways$Genes)  
)

pathways\_df

## Kegg\_path Name Description  
## 1 map00562 nothing nothing  
## 2 map01100 nothing nothing  
## 3 map04070 nothing nothing  
## Class Genes  
## 1 Metabolism; Carbohydrate metabolism PI4K2B  
## 2 nothing PI4K2B  
## 3 Environmental Information Processing; Signal transduction PI4K2B

En este caso, el gen **PI4K2B** está presente en las tres rutas metabólicas, es decir, en los tres pathways. Veamos a que clase pertece cada ruta metabólica:

* map00562 pertenece al metabolismo de los carbohidrados.
* map01100 está listado, pero no tiene una descripción específica asociada, por lo que no podemos determinar su categoría.
* map04070 pertenece a la categoría de Procesamiento de Información Ambiental; Transducción de Señales, que abarca cómo las células perciben y responden a señales provenientes del entorno, como estímulos químicos o físicos.

La tercera y última forma de enriquecimiento funcional que se tarta en este proyecto, viene dada por la función DEGsToDiseases. Con ella se descarga información sobre las enfermedades relacionadas y busca las evidencias de que esos genes esten asociados a esas enfermedades.

diseases <- DEGsToDiseases(entrezAnnotation$external\_gene\_name, getEvidences = TRUE)

* *entrezAnnotation$external\_gene\_name*: se pasan los nombres de los genes de interés, en nuestro caso la huella escogida.
* *getEvidences = TRUE*: al activar este parámetro, la función recupera las enfermedades asociadas con esos genes y obtiene las evidencias científicas que respaldan dichas asociaciones, como estudios previos, artículos de investigación y bases de datos.

diseases

## $PI4K2B  
## $PI4K2B$summary  
## Disease Overall Score   
## [1,] "breast adenocarcinoma" "0.245789882786219"   
## [2,] "protein measurement" "0.23772479160914"   
## [3,] "head and neck squamous cell carcinoma" "0.185653041923894"   
## [4,] "lung adenocarcinoma" "0.185159507197036"   
## [5,] "cutaneous melanoma" "0.184789927342351"   
## [6,] "squamous cell lung carcinoma" "0.123050272184642"   
## [7,] "autism" "0.118265553499105"   
## [8,] "blood protein measurement" "0.0802600104157921"  
## [9,] "gut microbiome measurement" "0.0733505881556975"  
## [10,] "small cell lung carcinoma" "0.0569152976214441"  
## Literature RNA Expr. Genetic Assoc. Somatic Mut.  
## [1,] "0" "0" "0" "0"   
## [2,] "0" "0" "0.391039231016243" "0"   
## [3,] "0" "0" "0" "0"   
## [4,] "0.0121586159522324" "0" "0" "0"   
## [5,] "0" "0" "0" "0"   
## [6,] "0" "0" "0" "0"   
## [7,] "0" "0" "0.194537855235719" "0"   
## [8,] "0" "0" "0.13202162274244" "0"   
## [9,] "0" "0" "0.120656147778448" "0"   
## [10,] "0.468106714160948" "0" "0" "0"   
## Known Drug Animal Model Affected Pathways   
## [1,] "0" "0" "0.8086113871903"   
## [2,] "0" "0" "0"   
## [3,] "0" "0" "0.610770313506962"  
## [4,] "0" "0" "0.607930797611621"  
## [5,] "0" "0" "0.607930797611621"  
## [6,] "0" "0" "0.404816708309796"  
## [7,] "0" "0" "0"   
## [8,] "0" "0" "0"   
## [9,] "0" "0" "0"   
## [10,] "0" "0" "0"   
##   
##   
## $<NA>  
## $<NA>$summary  
## Disease Overall Score Literature  
## [1,] "neurodegenerative disease" "0.523939935524986" "0"   
## [2,] "white matter microstructure measurement" "0.380561614948459" "0"   
## [3,] "apolipoprotein B measurement" "0.193076784808251" "0"   
## [4,] "platelet crit" "0.173817972519522" "0"   
## [5,] "butyrylcholinesterase measurement" "0.170720377218997" "0"   
## [6,] "urinary albumin to creatinine ratio" "0.13278286683883" "0"   
## [7,] "osteoarthritis" "0.122583322787939" "0"   
## [8,] "lymphocyte count" "0.118130420080811" "0"   
## [9,] "prostate carcinoma" "0.110873956405411" "0"   
## [10,] "Familial prostate cancer" "0.110873956405411" "0"   
## RNA Expr. Genetic Assoc. Somatic Mut. Known Drug Animal Model  
## [1,] "0" "0" "0" "0" "0"   
## [2,] "0" "0.625994959366383" "0" "0" "0"   
## [3,] "0" "0.317596650090425" "0" "0" "0"   
## [4,] "0" "0.285917366256818" "0" "0" "0"   
## [5,] "0" "0.280822057197474" "0" "0" "0"   
## [6,] "0" "0.218417733334937" "0" "0" "0"   
## [7,] "0" "0.201640257854236" "0" "0" "0"   
## [8,] "0" "0.194315571023725" "0" "0" "0"   
## [9,] "0" "0" "0.182379239283486" "0" "0"   
## [10,] "0" "0" "0.182379239283486" "0" "0"   
## Affected Pathways   
## [1,] "0.861841409554163"  
## [2,] "0"   
## [3,] "0"   
## [4,] "0"   
## [5,] "0"   
## [6,] "0"   
## [7,] "0"   
## [8,] "0"   
## [9,] "0"   
## [10,] "0"

La información proporcionada se refiere a nuestros genes con identificadores Entrez: **PI4K2B** y **RAPH1**, junto con las **enfermedades** asociadas a ellos y los respectivos **puntajes de asociación**. Aunque en el segundo caso aparezca NA, tras varias comprobaciones he descubierto que se refiere al gen **RAPH1**.

En primer lugar, el gen **PI4K2B** está asociado con diversas condiciones, aunque los puntajes de asociación son relativamente bajos. Las enfermedades con mayores *Overall Score*, es decir, más relacionadas con el gen, son: **adenocarcinoma de mama**, **medición de proteínas** y **carcinoma escamoso de cabeza y cuello**. En cuanto a la **medición de proteínas**, la asociación genética muestra un puntaje de 0.391, lo que indica cierta relación genética entre **PI4K2B** y esta condición. Sin embargo, para muchas otras enfermedades, no se observan asociaciones significativas, ya que los puntajes de Genetic Assoc. son todos “0”. Destaca también, el caso del **autismo**, que presenta un puntaje de 0.1945 en asociación genética, lo que sugiere una relación con el gen **PI4K2B**, aunque es débil. En relación con la evidencia en literatura, el único valor relevante se encuentra en el **carcinoma pulmonar de células pequeñas**, con un puntaje de 0.468 y el **adenocarcinoma de pulmón** con un puntuaje de 0.012. Esto indica que **PI4K2B** tiene una expresión moderada en este tipo de cáncer y evidencias disponibles en la literatura.

Por otro lado, en la mayoría de las enfermedades y condiciones mencionadas (como **autismo**, **medición de proteínas sanguíneas**, o **microbioma intestinal**), tanto la expresión de ARN como las mutaciones somáticas, los fármacos conocidos y los modelos animales no muestran evidencia fuerte, ya que todos los valores están en “0”.

En segundo lugar, el gen **RAPH1** parece que tiene más asociación con las enfermedades neurodegenerativas (*Ovarall Score* = 0.53), así como con la **medición de la microestructura de la sustancia blanca**, es decir, con el análisis detallado de las fibras nerviosas; y la **medición de la apolipoproteína B** lo que implica evaluar los niveles de una proteína vinculada al transporte de lípidos, cuya alteración puede estar asociada con enfermedades cardiovasculares y neurodegenerativas. Además, estas dos últimas enfermedades son las que presentan valores más altos en *asociación genética*. Sin embargo, en relación con la literatura, expresión de ARN, fármacos conocidos o modelos animales no se proporciona ninguna información.

En cuanto a mutaciones somáticas para el gen **RAPH1**, se observa que la mayoría de las enfermedades listadas no tienen mutaciones somáticas registradas, ya que los valores en esta columna son “0” para muchas de ellas. Sin embargo, hay dos excepciones: el **carcinoma de próstata** y el **cáncer de próstata familiar**, que tienen un valor de 0.1824, indicando una presencia mínima de mutaciones somáticas asociadas con este gen. Esto sugiere que, aunque la mutación somática no es un factor dominante en la mayoría de los casos, podría tener alguna relevancia en ciertos tipos de cáncer, como los relacionados con la próstata.

Finalmente, los valores de *Affected Pathways* son más altos en ciertas condiciones, como en el **adenocarcinoma de mama**, en el caso del **PI4K2B**, donde la vía afectada muestra un valor de 0.8086, indicando este gen podría estar involucrado en la regulación de ciertas vías metabólicas o celulares relevantes para este tipo de cáncer. Además, para el **RAPH1** solo hay un valor para el *Affected Pathways* y es de 0.862 lo que indica que este gen podría estar involucrado en la regulación de vías metabólicas o celulares específicas que desempeñan un papel importante en los procesos asociados a las **enfermedades neurodegenerativas**.