Biología Computacional con Big Data-Omics e Ingeniería Biomédica

DATCOM
Máster Universitario en Ciencia de Datos e Ingeniería de Computadores
Especialidad en Ciencia de Datos

UNIVERSIDAD DE GRANADA

DATCOM

TRES TIPOS DE CÁNCER DE CEREBRO: astrocitoma, glioma mixto y oligodendroglioma.

Elena Maroto Rica

Pablo Heredero García

7-02-2025

Contenido

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 cáncer 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 cáncer son las mutaciones en el ADN transmitidos de generación en generación y la exposición a radiaciones ionizantes.

Los astrocitomas 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 de crecimiento 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 Andaluz 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:

ASTROCITOMA	GLIOMA MXTO	OLIGODENDROGLIOMA
197	135	201

- miRNA:

ASTROCITOMA	GLIOMA MXTO	OLIGODENDROGLIOMA

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:

ENTRENBAMIENTO	PRUEBA
XTrn 26.079 observaciones (genes), 413 columnas. Por tanto, YTrn es un vector de 413 etiquetas.	XTest 26.079 observaciones (genes), 104 columnas (*samples*). Por tanto, YTest es un vector de 104 etiquetas.

- miRNA:

ENTRENBAMIENTO	PRUEBA

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 *knn_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, especialmente en presencia de datos de alta dimensión y correlacionados. Esto último es la razón por la que este algoritmo de clasificación es apropiado para clasificar los tres tipos de cáncer de cerebro

  Por otro lado, la parte principal del código de la función *logistic_test_multiclase* es el *for*, en el cual se irán seleccionando genes, con el mejor parámetro *decay* seleccionado en la fase de entrenamiento, de la siguiente forma: en la primera iteración se escoge 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:

```
down_folder = "brain/"

sample_sheet = "gdc_sample_sheet.2025-01-10.tsv"

samples <- convert_to_counts(down_folder, sample_sheet, outpath = "brain/counts", col
umn_name = 'unstranded', filter_lines = 4)

load(file ='samples')
```

Para adaptar los counts a nuestro problema, 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] <- 'oligo
dendroglioma'
```

Una vez etiquetado correctamente el data frame *sample* con los ficheros counts, creamos las variables necesarias para el data frame 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 muestras por clase, donde se pueden ver que están 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 aúnan 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 las 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 expresión 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 = myAnnotati
on)
# 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 análisis de calidad y se seleccionan las muestras que pasen por el filtro (no outliers):

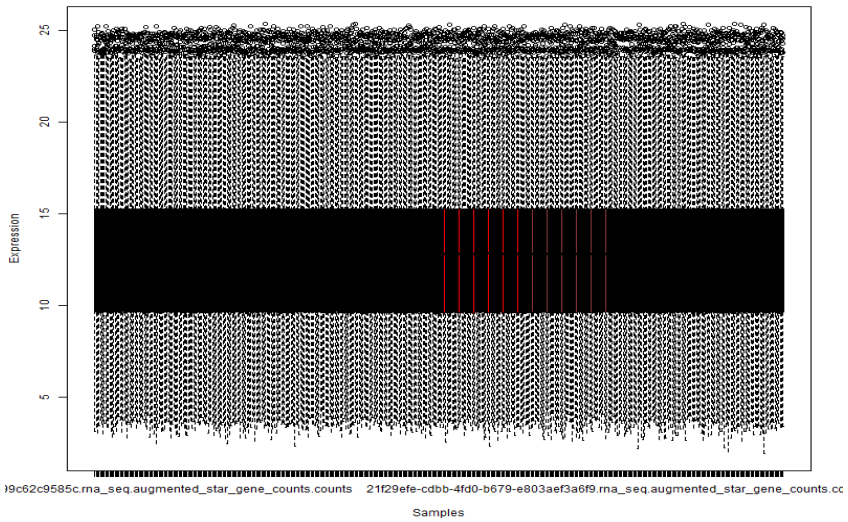
```
QAResults <- RNAseqQA(geneExprMatrix, toRemoval = TRUE,
                      toPNG=FALSE, toPDF=FALSE)
save('QAResults', file = 'QAResults')

# 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 subrogadas para tratar el efecto batch:

```
# Creamos el modelo SVA de variables subrogadas 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 ejecucione
s
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:

```
dataPlot(MATRIZ, qualityLabels, mode = "orderedBoxplot")
```



2.2. 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]
```

Los resultados de esta división se han detallado en la introducción del trabajo.
Del conjunto de entrenamiento se propone extraer genes diferencialmente expresados (DE) jugando con diferentes valores de COV y LFC:

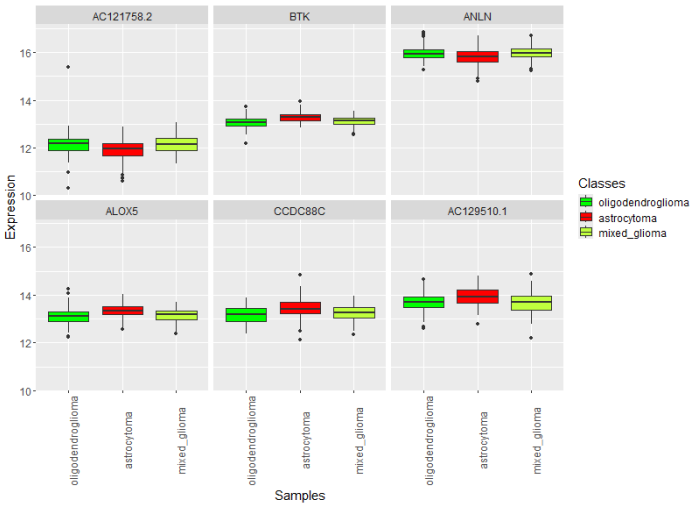
```
LFC_values <- c(0.1, 0.15, 0.25, 0.75, 1, 1.5, 2, 2.25, 2.75, 3, 3.5)
COV_values <- c(1, 2, 3)

results <- data.frame(LFC = numeric(0), COV = numeric(0), NumGenes = numeric(0))

# Función para extraer los genes diferencialmente expresados con LFC y COV específico
s
for (LFC in LFC_values) {
  for (COV in COV_values) {
    tryCatch({
      # DEGsExtraction para esta combinación de LFC y COV
      DEGsInfo <- DEGsExtraction(XTrn, YTrn, lfc = LFC, cov = COV, pvalue=0.001)

      # Número de genes DEGs (filas de DEGs_Matrix)
      num_genes <- nrow(DEGsInfo$DEG_Results$DEGs_Matrix)

      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))
      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
```

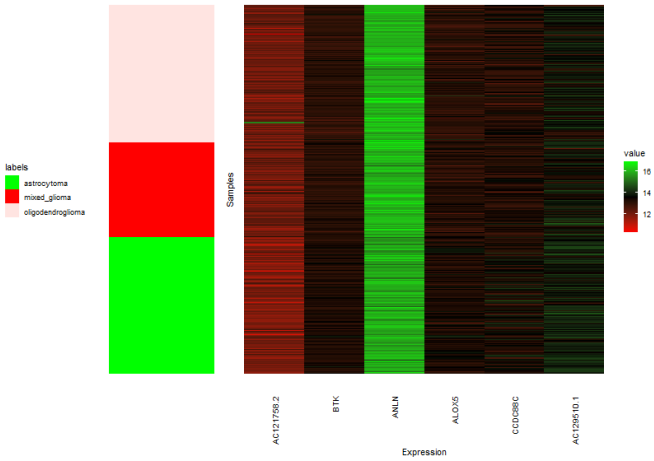
Como lo conveniente es que salga un número de genes entre 50-300 aproximadamente, filtramos los resultados obtenidos anteriormente:

```
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 expresión de 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)

DEGsMatrix <- DEGsInfo$DEG_Results$DEGs_Matrix
```





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 cuatro algoritmos de selección de características y los cuatro 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 = coln
ames(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 = coln
ames(MLMatrix))
load('FSRankingDA')
```

El cuarto se muestra a partir de la 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 cad
a 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)
```

```r
      # Verificar si hay suficientes columnas en Mj para realizar el cálculo de infor
mación 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 pérdi
da
      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')
```

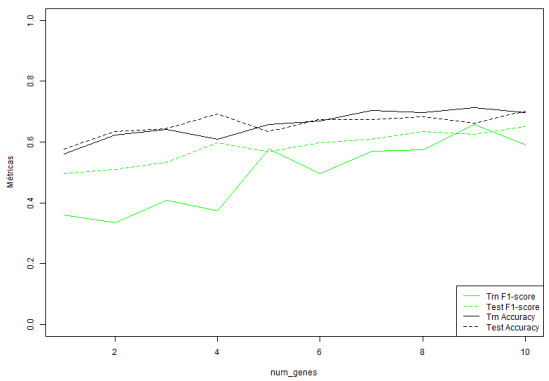
3.1. Clasificador k-NN

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

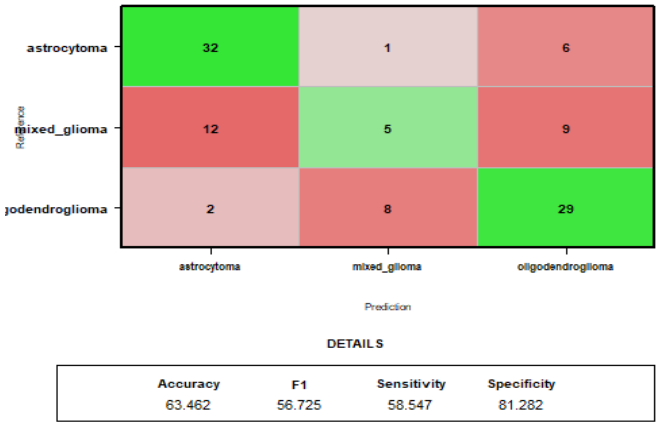
3.1.1. Selección de características con FSRankingMRMR

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(FS
RankingMRMR[1:10]), bestK=knn_trn_mrmr$bestK)
knn_results_mrmr_test <- rbind(knn_trn_mrmr$F1Info$meanF1[1:10], knn_test_mrmr$f1Vect
or, knn_trn_mrmr$accuracyInfo$meanAccuracy, knn_test_mrmr$accVector)
```



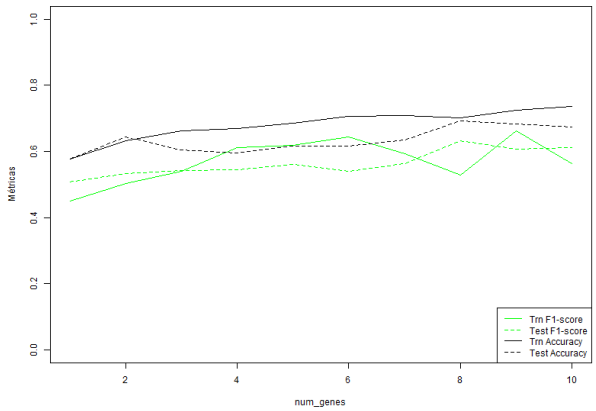
Cuando num_genes = 5, se estabiliza las curvas alcanzando el 60% de acierto (accuracy) y de F1 para la evaluación del conjunto train y test. Comprobamos esta información con la matriz de confusión para el conjunto test:



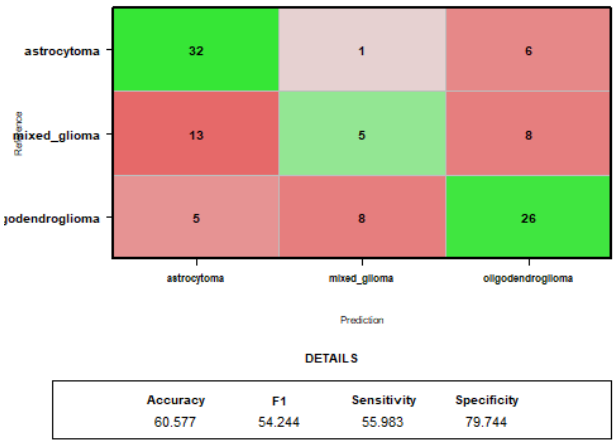
3.1.2. Selección de características con FSRankingRF

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

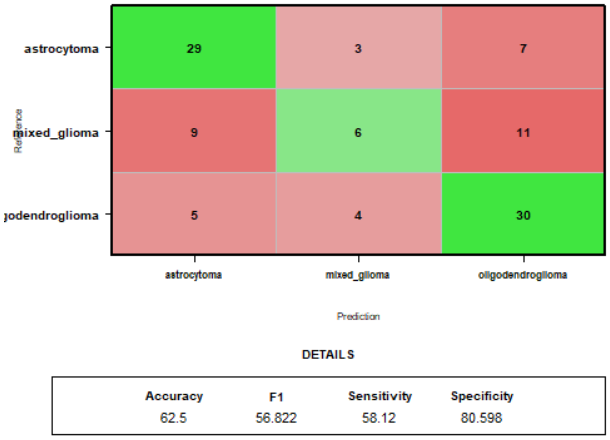
```
# Evaluar biomarcadores en TRN. Es multiclase, mejor ver accuracy y F1-Score
knn_trn_rf <- knn_trn(MLMatrix, MLLabels, vars_selected = FSRankingRF[1:10])
knn_results_rf <- rbind(knn_trn_rf$F1Info$meanF1[1:10],knn_trn_rf$accuracyInfo$meanAccuracy)

# 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 del test 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:



En segundo lugar, para una huella de seis genes:

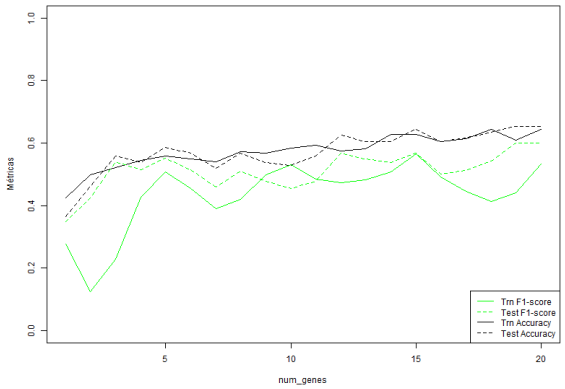


El aumento del porcentaje de acierto es muy poco tras elegir el doble de genes. Por tanto, la huella escogida en este caso podría ser 3.

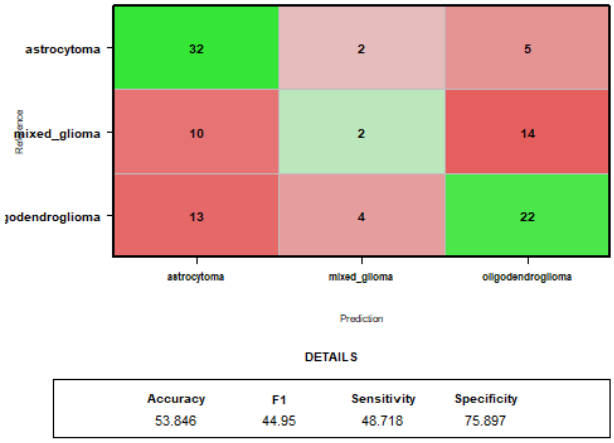
3.1.3. Selección de variables con FSRankingDA

Repetimos el mismo procedimiento para el ranking de DA. En este caso, se han considerado los 20 primeros genes seleccionado con el objetivo de visualizar mejor los datos y el valor de la huella.

```
# 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$accuracyInfo$meanAc
curacy)

# 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(FSR
ankingDA[1:20]), bestK=knn_trn_da$bestK)
knn_results_da_test <- rbind(knn_trn_da$F1Info$meanF1[1:20], knn_test_da$f1Vector, kn
n_trn_da$accuracyInfo$meanAccuracy, knn_test_da$accVector)
```



Parece que, a partir de 10 genes, deja 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 buena clasificación a menos que escojamos un número muy elevado de genes. Se contrasta esta información con la matriz de confusión para el conjunto test:



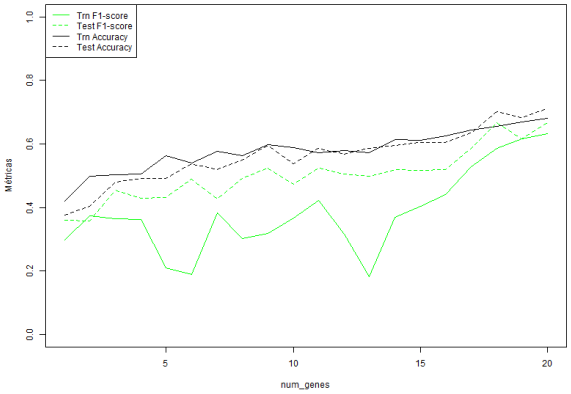
DETAILS

Accuracy	F1	Sensitivity	Specificity
53.846	44.95	48.718	75.897

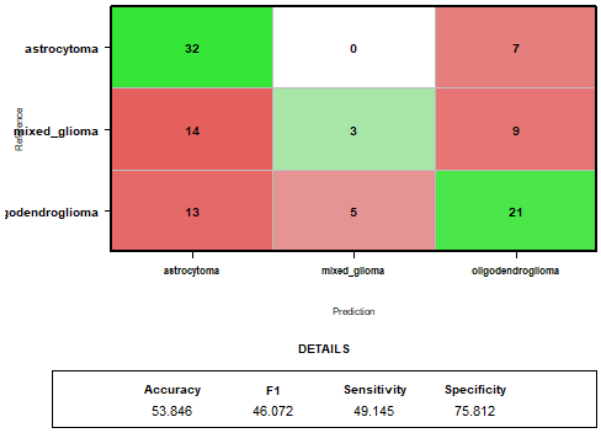
3.1.4. Selección de variables con FSRanking_MB

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$accuracyInfo$meanAc
curacy)
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, kn
n_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 la matriz 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 destacable.

3.1.5. Conclusiones

Para el clasificador k-NN, el algoritmo de selección de selección de características mRMR es el que mejor porcentaje de Accuracy proporciona: 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 conclusión, 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 Random Forest

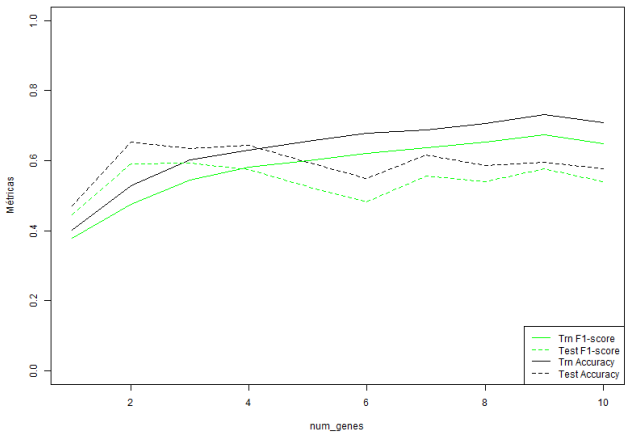
En esta sección se repetirá el procedimiento aplicado con el algoritmo k-NN (k-Vecinos más cercanos) utilizando los cuatro rankings de características

disponibles. Se evaluará el rendimiento del modelo en términos de métricas de clasificación como precisión 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.

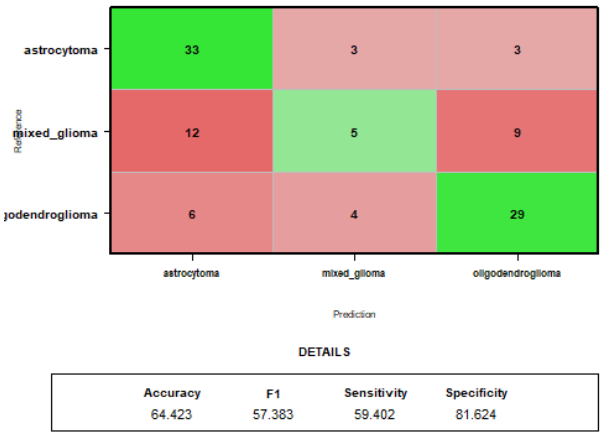
3.2.1. Selección de variables con FSRankingMRMR

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]),n
umFold = 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$accuracyInfo$mea
nAccuracy)

# 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(FSR
ankingMRMR[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)
```



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 métricas con esta huella, se presenta la matriz de confusión para el conjunto test:

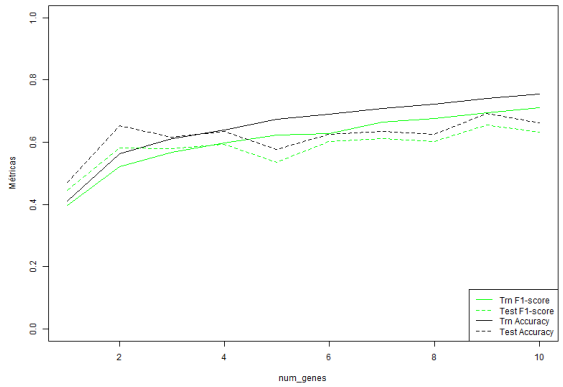


Este es el modelo con mayor porcentaje de acierto y menor huella hasta el momento.

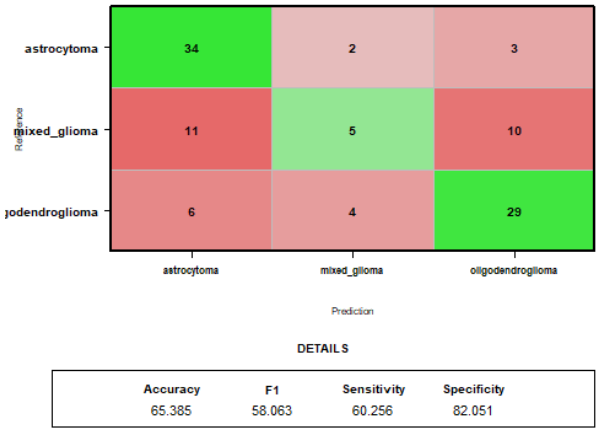
3.2.2. Selección de variables con FSRankingRF

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$accuracyInfo$meanAccur
acy)

# 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 = (FSRanking
RF[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$f1Vector, rf_tr
n_rf$accuracyInfo$meanAccuracy, rf_test_rf$accVector)
```



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 poder hacer conclusiones más fiables:

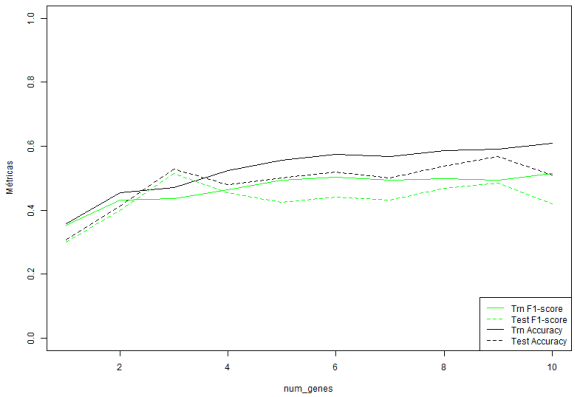


Destaca que el accuracy en este caso y en el anterior para una huella de 2 genes, es el mismo. Por tanto, para distinguirlos podríamos fijarnos en que el F1-score para el ranking del método mRMR (59.1%) es algo mayor que en este caso.

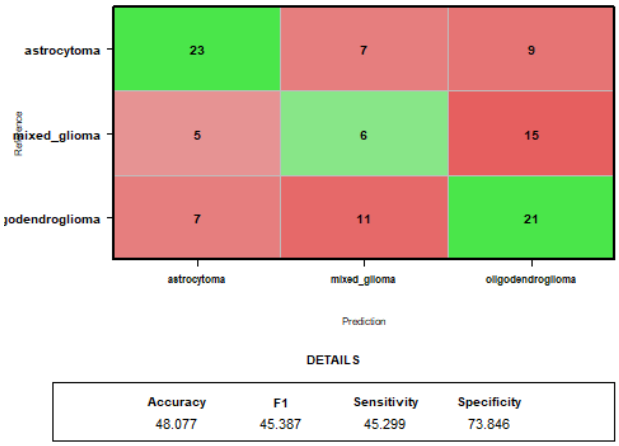
3.2.3. Selección de variables con FSRankingDA

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 precisión 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)

# 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)
```



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 apenas alcanza el 60%. Veamos lo explicado en la siguiente matriz de confusión para el conjunto test:

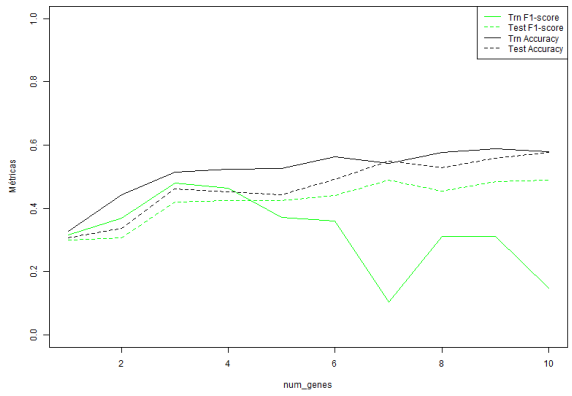


Con la mejor huella para el conjunto train, el test a penas acierta en el 50% de las ocasiones.

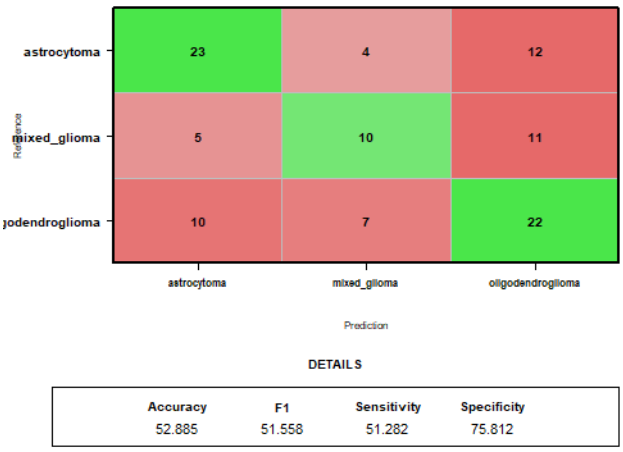
3.2.4. Selección de variables con FSRanking_MB

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$accuracyInfo$meanAccur
acy)
rf_test_mb <- rf_test(MLMatrix, MLLabels, t(XTest), YTest, vars_selected=FSRanking_MB
[1:10], bestParameters = rf_trn_mb$bestParameters)

rf_results_mb_test <- rbind(rf_trn_mb$F1Info$meanF1[1:10], rf_test_mb$f1Vector, rf_tr
n_mb$accuracyInfo$meanAccuracy, rf_test_mb$accVector)
```



Destaca el descenso del F1-score para el conjunto train a partir de 3 genes.
Elegimos esta huella aun 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 utilizarán 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 como comentarios para el buen funcionamiento en nuestro 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. Pl
ease, 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. Plea
se, 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 sintácti
camente válidos
    tr_ctr <- trainControl(method="none")
    colnames(train)<-make.names(colnames(train)) # se alteran los nombres para que coincidan con los de las co
lumnas
    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 correctamen
te.
```

```
      # 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 prec
isión sea NA y el recall sea 0. Esto sucede cuando una clase no es predicha ni correctamente ni incorrectament
e, 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 (`# 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)
```

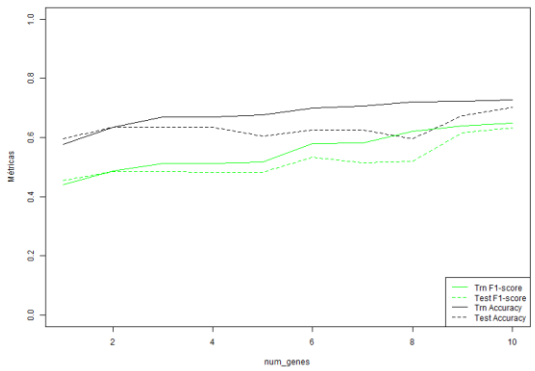
```r
  colnames(dataForTunning) <- make.names(colnames(dataForTunning))
  Rsvm_sb <- train(labels ~ ., data = dataForTunning, type = "C-svc", method = "svmRadial", preProc = c("cente
r", "scale"), trControl = fitControl, tuneGrid = grid_radial)

  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)

}
```

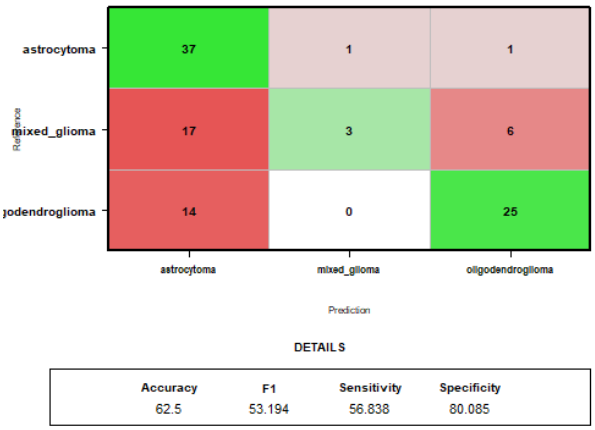
3.3.1. Selección de variables con FSRankingMRMR

Comencemos 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)

svm_test_mrmr <- svm_test2(MLMatrix, MLLabels, t(XTest), YTest, vars_selected=names(F
SRankingMRMR[1:10]), bestParameters= svm_trn_mrmr$bestParameters)
svm_results_mrmr_test <- rbind(svm_trn_mrmr$F1Info$meanF1[1:10], svm_test_mrmr$f1Vect
or, svm_trn_mrmr$accuracyInfo$meanAccuracy, svm_test_mrmr$accVector)
```



En este caso, para num_genes = 6, destaca el pico de F1 para el conjunto 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:



DETAILS

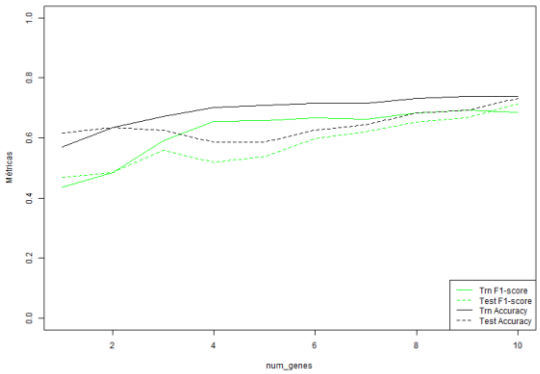
Accuracy	F1	Sensitivity	Specificity
62.5	53.194	56.838	80.085

Cabe destacar que ningún astrocitoma 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

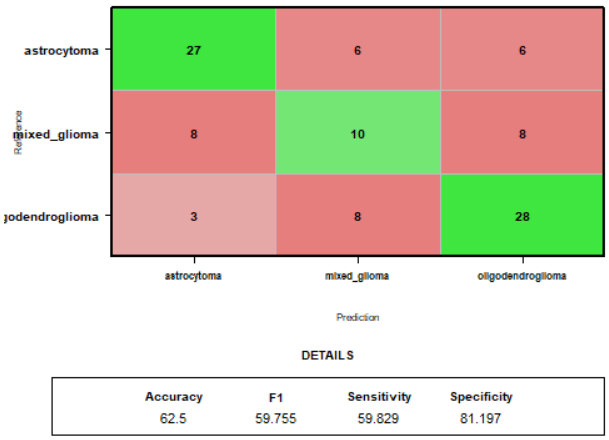
En este caso, se entrenan los modelos *svm_train_rf* y *svm_test_rf*. Calculando las métricas correspondientes para la valoración de la clasificación en el conjunto test.

```
# 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$accuracyInfo$meanAc
curacy)

# 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 = (FSRank
ingRF[1:10]), bestParameters= svm_trn_rf$bestParameters)
svm_results_rf_test <- rbind(svm_trn_rf$F1Info$meanF1[1:10], svm_test_rf$f1Vector, sv
m_trn_rf$accuracyInfo$meanAccuracy, svm_test_rf$accVector)
```



En este caso, elegiría una huella de 4 genes ya que, a partir de ahí, el crecimiento 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.

Veamos así, la matriz de confusión para el conjunto test cuando la huella es 4:

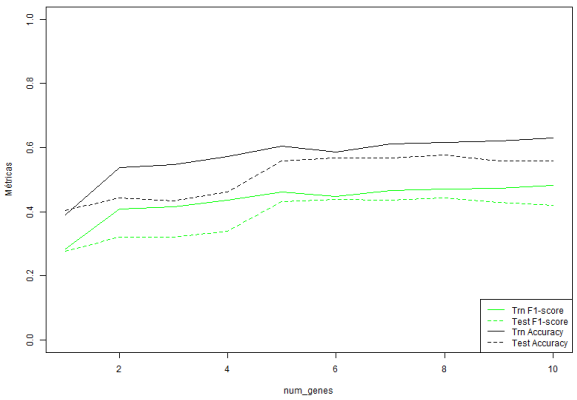


Hay una clasificación decente y podemos quedarnos con esta huella para esta combinación de seleccionador de variables y clasificador.

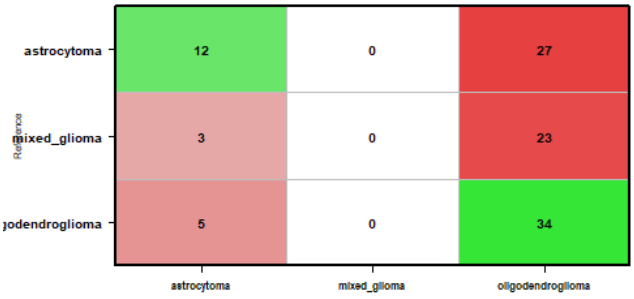
3.3.3. Selección de variables con FSRankingDA

Repetimos el proceso 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$accuracyInfo$meanAc
curacy)

# 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(F
SRankingDA[1:10]), bestParameters= svm_trn_da$bestParameters)
svm_results_da_test <- rbind(svm_trn_da$F1Info$meanF1[1:10], svm_test_da$f1Vector, sv
m_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:



DETAILS

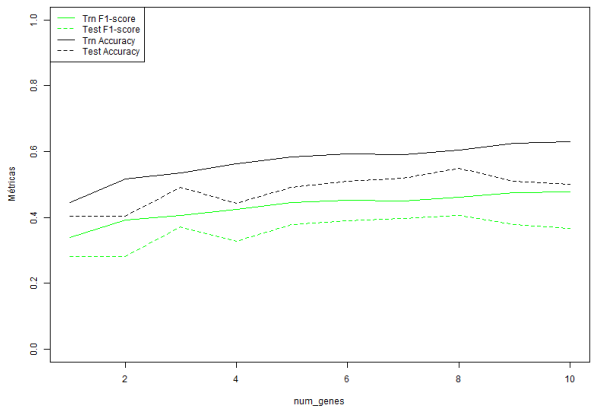
Accuracy	F1	Sensitivity	Specificity
44.231	31.988	39.316	70.256

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

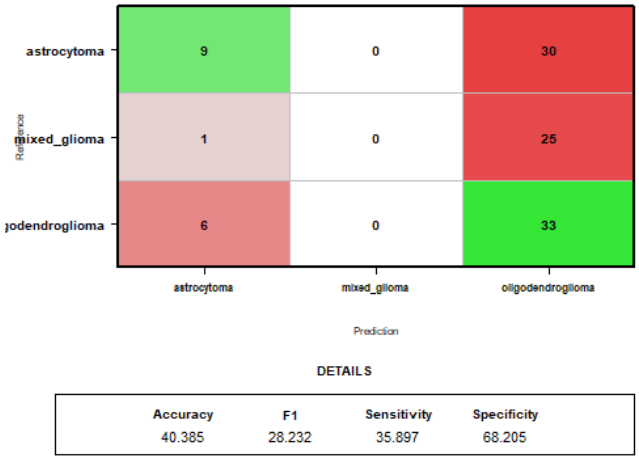
3.3.4. Selección de variables con FSRanking_MB:

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$accuracyInfo$meanAc
curacy)
svm_test_mb <- svm_test2(MLMatrix, MLLabels, t(XTest), YTest, vars_selected=FSRanking
_MB[1:10], bestParameters = svm_trn_mb$bestParameters)

svm_results_mb_test <- rbind(svm_trn_mb$F1Info$meanF1[1:10], svm_test_mb$f1Vector, sv
m_trn_mb$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:



DETAILS

Accuracy	F1	Sensitivity	Specificity
40.385	28.232	35.897	68.205

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 función *logistic_train2* ayudándonos del código fuente de la función de KnowSeq *knn_trn*:

```
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 ens
ure 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")

  # regresión logística triclase
  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]


      logistic_mod <- multinom(labelsTrain ~ ., data = trainingDataset)
      predicts <- predict(logistic_mod, testDataset)

      cfMatList[[i]] <- confusionMatrix(predicts, labelsTest)
      acc_cv[i, 1] <- 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]]$byClass[,2])
          f1 <- mean(cfMatList[[i]]$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, logist
ic_model)
    names(results_cv) <- c("cfMats", "accuracyInfo", "sensitivityInfo", "specificityInfo", "F1In
fo", "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", "F
1Info")
    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
```

```r
, 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 etiqu
etasTren.")
  }

  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 eti
quetasPrueba.")
  }

  # 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", "predicc
iones")

  invisible(resultados)
}
```

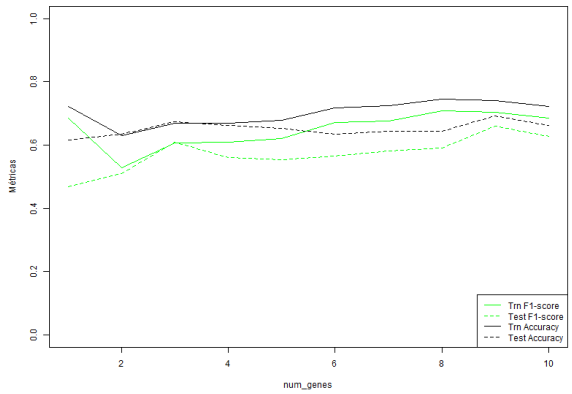
En estas funciones también se ha considerado que, si no es posible calcular el F1 para alguna clase e iteración debido a que no se predice ni correctamente ni erróneamente esa clase, el valor de dicho F1 será cero, lo que permite calcular adecuadamente la media de los tres F1 y tener en cuenta los F1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

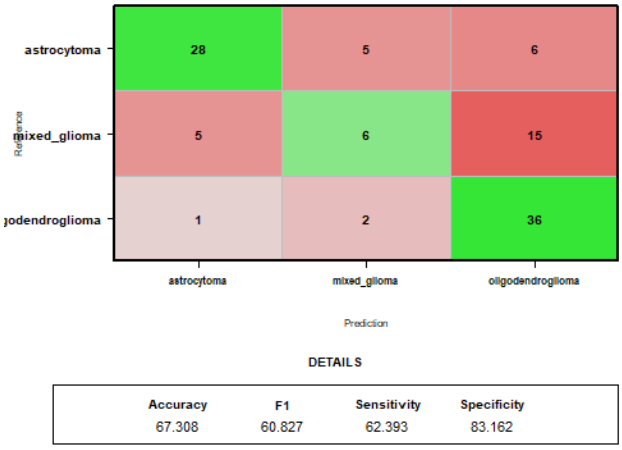
3.4.1. Selección de variables con FSRankingMRMR

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(FSRankin
gMRMR[1:10]), numFold = 5)
logistic_results_mrmr <- rbind(logistic_trn_mrmr$F1Info$meanF1[1:10],logistic_trn_mrm
r$accuracyInfo$meanAccuracy)

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score
logistic_test_mrmr <- logistic_test_multiclase(MLMatrix, MLLabels, t(XTest), YTest, n
ames(FSRankingMRMR[1:10]), logistic_trn_mrmr)

logistic_results_mrmr_test <- rbind(logistic_trn_mrmr$F1Info$meanF1[1:10], logistic_t
est_mrmr$f1Vector, logistic_trn_mrmr$accuracyInfo$meanAccuracy, logistic_test_mrmr$ac
cVector)
```



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 para los dos conjuntos. Se comprueba con la matriz de confusión para el conjunto test y una huella de 3 genes:



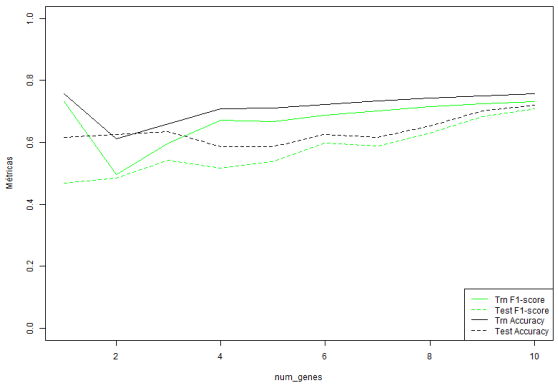
Esta es la mejor predicción hasta el momento. Es interesante destacar que la clase que más se confunde es el glioma mixto con el oligodendroglioma. Esto puede deberse a que el glioma mixto presenta características del oligodendroglioma, lo que hace más difícil diferenciarlas en el modelo. Dado que ambas 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.

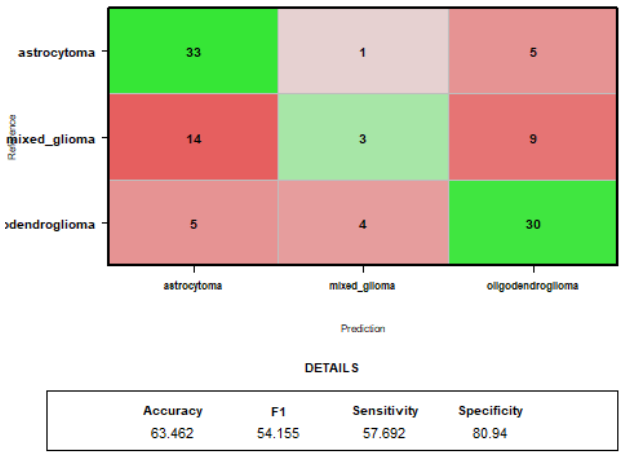
3.4.2. Selección de variables con FSRankingRF

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[1:1
0]), numFold = 5)
logistic_results_rf <- rbind(logistic_trn_rf$F1Info$meanF1[1:10],logistic_trn_rf$accu
racyInfo$meanAccuracy)

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score
logistic_test_rf <- logistic_test_multiclase(MLMatrix, MLLabels, t(XTest), YTest, (FS
RankingRF[1:10]), logistic_trn_rf)

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 desciende de forma leve. Veamos así, la matriz de confusión para el conjunto test y una huella de 3 genes:



DETAILS

Accuracy	F1	Sensitivity	Specificity
63.462	54.155	57.692	80.94

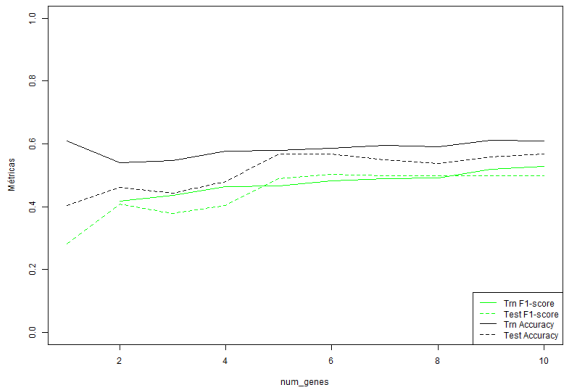
En contraste con el caso anterior, donde la mayor confusión se da entre el glioma mixto y el oligodendroglioma, en este caso los que más se confunden son el glioma mixto y el astrocitoma. Esto puede ocurrir debido a que ambas clases presentan

características similares, lo que dificulta que el modelo las distinga con precisión de nuevo.

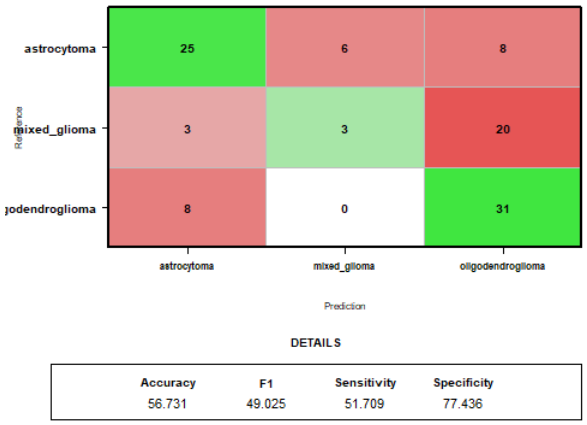
3.4.3. Selección de variables con FSRankingDA

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$accu
racyInfo$meanAccuracy)

# Evaluar huella genética en TEST. Es multiclase, mejor ver accuracy y F1-Score
logistic_test_da <- logistic_test_multiclase(MLMatrix, MLLabels, t(XTest), YTest, var
s_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:



DETAILS

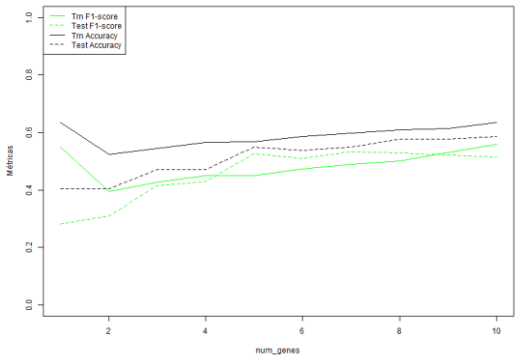
Accuracy	F1	Sensitivity	Specificity
56.731	49.025	51.709	77.436

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

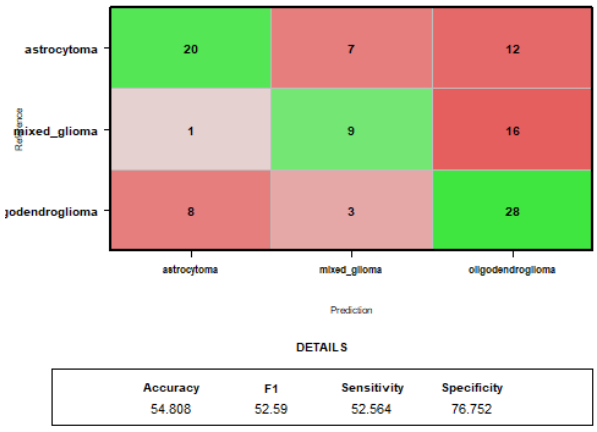
3.4.3. Selección de variables con FSRanking_MB

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[1:1
0])
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$accu
racyInfo$meanAccuracy)
logistic_test_mb <- logistic_test_multiclase(MLMatrix, MLLabels, t(XTest), YTest, var
s_selected = (FSRanking_MB[1:10]), logistic_trn_mb)

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 se ha obtenido una gráfica muy similar a la anterior, pero no igual. Por ello, calculemos la matriz de confusión para el conjunto test y esta huella para visualizar los resultados con más detalle:



DETAILS

Accuracy	F1	Sensitivity	Specificity
54.808	52.59	52.564	76.752

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

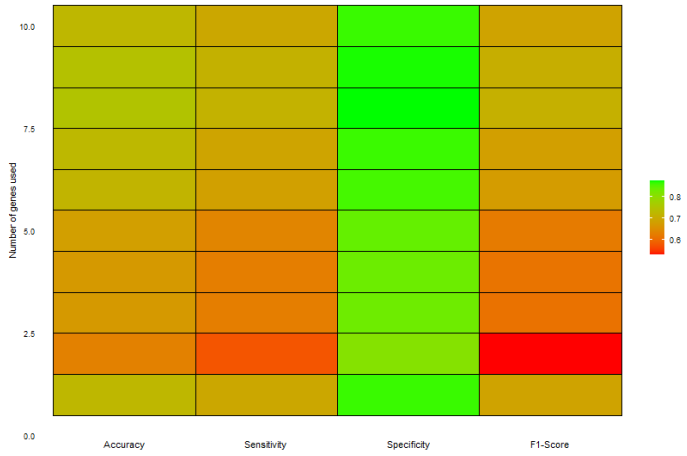
3.4.5. Conclusiones

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 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 relevancia.

En primer lugar, el algoritmo de selección de variables elegido es el **mRMR** y el clasificador será **Regresión Logística múltiple con penalización**; ya que es con el que se ha obtenido mejores valores de Accuracy, F1, sensibilidad y especificidad. Se muestra, a continuación, el heatmap de esta combinación para el conjunto train:



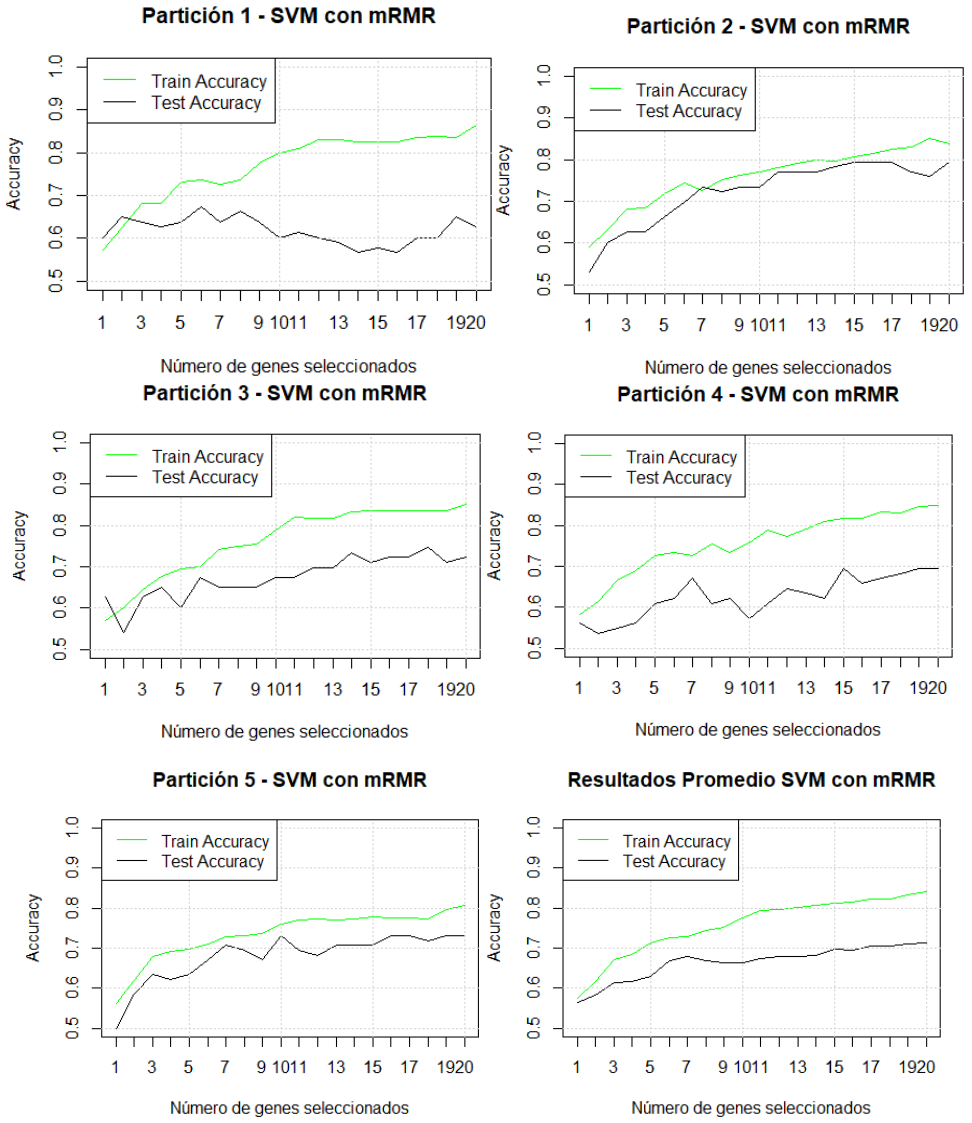
En esta imagen podemos observar que la especificidad ronda el 85%, luego el modelo es capaz de identificar correctamente a las personas **sin** la enfermedad. Así, también se observa que a partir una huella de 3 o 5 genes va aumentando el accuracy y el F1-Score, los cuales, junto con la sensibilidad están entorno al 70%.

En segundo lugar, se presenta el siguiente código con el objetivo de mostrar resultado, para el modelo de regresión logística con una penalización de $decay = 10^{-4}$ (la obtenida 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 a
umentó hasta 50
PREDICCIONES_SVM <- matrix(0, nVarsMAX, length(MLLabels)) # Matriz para almacenar pre
dicciones y facilitar su acceso

# Matrices para almacenar métricas por partición
ACCTrnSVM <- matrix(0, nfolds, nVarsMAX)
ACCTestSVM <- matrix(0, nfolds, nVarsMAX)

# Matriz de confusión acumulada para el test
accTestSum <- matrix(0, nrow = length(unique(YTest)), ncol = length(unique(YTest)))

# 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)) {
  # Particiones de entrenamiento y prueba
  datosParaTRN <- which(foldIDx != particion)  # Índices de entrenamiento
  segmentoTest <- which(foldIDx == particion)  # Índices de prueba

  XTrn <- as.data.frame( MLMatrix[datosParaTRN, ] ) # Datos de entrenamiento
  XTest <- as.data.frame(MLMatrix[segmentoTest, ])  # Datos de prueba

  YTrn <- as.factor(MLLabels[datosParaTRN])  # Etiquetas de entrenamiento
  YTest <- as.factor(MLLabels[segmentoTest])  # Etiquetas de prueba

  # Ranking con mRMR
  rankingMRMR <-featureSelection(XTrn, YTrn, mode = "mrmr", vars_selected = colnames(
XTrn))

  # 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)[rankingMRMR[1:numGenes]]

    # Entrenar modelo Regresión Logística multinomial
    modelo <- multinom(YTrn ~ ., data = subset(XTrn, select=topGenes), decay = best_d
ecay)

    # Predicciones en Train
    predTrain <- predict(modelo, subset(XTrn, select=topGenes))
    confMatTrain <- confusionMatrix(as.factor(predTrain), as.factor(YTrn))
    ACCTrnSVM[particion, numGenes] <- confMatTrain$overall["Accuracy"]

    # Predicciones en Test
    predTest <- predict(modelo, subset(XTest, select=topGenes))
    confMatTest <- confusionMatrix(as.factor(predTest), as.factor(YTest))
    ACCTestSVM[particion, numGenes] <- confMatTest$overall["Accuracy"] # se van añadi
```

```
endo 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)
    # Guardar predicciones de Test
    PREDICCIONES_SVM[particion, segmentoTest] <- as.numeric(predTest)

    # Sumar la matriz de confusión al total
    accTestSum <- accTestSum + confMatTest$table
  }

  # Graficar los resultados para la partición actual
  num_genes <- 1:nVarsMAX
  Trn_Acc <- ACCTrnSVM[particion, 1:nVarsMAX]
  Test_Acc <- ACCTestSVM[particion, 1:nVarsMAX]

  plot(num_genes, Trn_Acc, type = "l", col = "green", ylim = c(0.5, 1), ylab = "Accur
acy",        xlab = "Número de genes seleccionados", main = paste("Partición", partic
ion, "- SVM 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","blac
k"),lty = c(1, 1))
}

# Medias de Accuracy en las 5 particiones
ACCTrnSVM_Mean <- colMeans(ACCTrnSVM)
ACCTestSVM_Mean <- colMeans(ACCTestSVM)

# Resultados promedio
plot(1:nVarsMAX, ACCTrnSVM_Mean, type = "l", col = "green", ylim = c(0.5, 1), xlim=c(
1,20), xlab = "Número de genes seleccionados", ylab = "Accuracy", main = "Resultados
Promedio SVM con mRMR")
lines(1:nVarsMAX, ACCTestSVM_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", "bl
ack"),
       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 en 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 *ACCTrnSVM_Mean* y *ACCTestSVM_Mean* que almacenan la media de los accuracy de cada iteración para el conjunto train y test, respectivamente:

```
ACCTrnSVM_Mean

##  [1] 0.5756697 0.6180408 0.6713046 0.6846178 0.7130770 0.7251854 0.7300284
##  [8] 0.7445537 0.7530477 0.7754426 0.7942067 0.7972443 0.8020837 0.8075254
## [15] 0.8129708 0.8141847 0.8214483 0.8220580 0.8335494 0.8420269

ACCTestSVM_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
```

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 10 primeros genes seleccionados en las 5 particiones del bucle utilizado para buscar la huella:

```
# 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"
```

Debemos estudiar esta matriz por filas, pues corresponden a cada iteración y las columnas son los primero 10 genes del ranking.

Tal y como se había discutido, la huella elegida es 5, por tanto, la idea es fijarnos en los 5 primeros genes seleccionados de las 5 iteraciones en los 5 rankings. Entre ellos, los genes más frecuentes son: el 170 o **IGHV3-73**, el 81 o **IGHV6-1** y el 51 o **UBQLN4P1**; estos son los primeros genes seleccionados en todos los rankings. Para seleccionar los dos genes restantes nos fijamos en los 10 primeros genes de los rankings, seleccionado: el 41 o **RAPH1** y el 10 o **PI4K2B**; presentes en 3 de las 5 iteraciones.

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 los genes escogidos y *numGenes <- 5*:

```
set.seed(19)

nfolds <- 5  # Número de particiones en la validación cruzada
foldIDx <- cvGenStratified(MLLabels, nfolds)  # Particiones estratificadas

numGenes <- 5  # Número fijo de genes seleccionados
PREDICCIONES_SVM <- numeric(length(MLLabels))  # Vector para almacenar predicciones

# Vectores para almacenar métricas por partición
ACCTrnSVM <- numeric(nfolds)
ACCTestSVM <- numeric(nfolds)

# Matriz de confusión acumulada para el test
accTestSum <- matrix(0, nrow = length(unique(MLLabels)), ncol = length(unique(MLLabel
s)))
```

```r
# 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 <- as.data.frame(MLMatrix[datosParaTRN, ])  # Datos de entrenamiento
  XTest <- as.data.frame(MLMatrix[segmentoTest, ])  # Datos de prueba

  YTrn <- as.factor(MLLabels[datosParaTRN])  # Etiquetas de entrenamiento
  YTest <- as.factor(MLLabels[segmentoTest])  # Etiquetas de prueba


  # Entrenar modelo Regresión Logística multinomial
  modelo <- multinom(YTrn ~ ., data = subset(XTrn, select=huella), decay = best_decay
)

  # Predicciones en Train
  predTrain <- predict(modelo, XTrn[, huella])
  confMatTrain <- confusionMatrix(as.factor(predTrain), YTrn)
  ACCTrnSVM[particion] <- confMatTrain$overall["Accuracy"]

  # Predicciones en Test
  predTest <- predict(modelo, XTest[, huella])
  confMatTest <- confusionMatrix(as.factor(predTest), YTest)
  #print(confMatTest$table)
  ACCTestSVM[particion] <- confMatTest$overall["Accuracy"]
  #print(ACCTestSVM[particion])

  # Guardar predicciones de Test
  PREDICCIONES_SVM[segmentoTest] <- as.numeric(predTest)

  # 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               132           26                 6
##   mixed_glioma               17           30                24
##   oligodendroglioma           4           50               124
```

La matriz de confusión muestra que el modelo tiene un buen desempeño clasificando astrocitoma con 132 predicciones correctas y oligodendroglioma con 124 correctas, concluyendo que la huella elegida es adecuada. Sin embargo, presenta confusión con mixed glioma, que se clasifica erróneamente como astrocitoma en 17 ocasiones y como oligodendroglioma en 24. Esto puede deberse a que el glioma mixto comparte características similares con las otras dos clases, lo que dificulta identificar sus diferencias de manera clara. Además, el conjunto de datos no contiene tantas muestras de este tipo de cáncer como del resto, lo que podría hacer que el modelo tenga dificultades para aprender los patrones distintivos de esta clase.

Finalmente, con estos cinco genes seleccionados, se harán dos particiones train (80%) y test (20%) con el objetivo de entrenar un modelo Regresión Logística múltiple y buscar el mejor hiperparámetro *decay* para este problema con un GridSearch. Para concluir esta sección se predecirán las clases de este nuevo conjunto 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 proba
r
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 p
liegues

# Modelo de Regresión Logística Multinomial
logistic_model <- train(x = X_train, y = Y_train, method = "multinom", trControl = tr
ain_control, tuneGrid = grid)

# 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_par
ams$decay)

# 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 cual coincide con el mejor hiperparámetro seleccionado en el modelo *logistic_trn_mrmr.* Esto ha permitido obtener una precisión del 72.84% en el conjunto de prueba, mostrando una buena clasificación. En cuanto a los estadísticos por clase, la sensibilidad es particularmente alta en la clase astrocitoma 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 para el astrocitoma (91.76%) y oligodendroglioma (76.86%). Así, es importante destacar que la clase del glioma mixto presenta un desafío en su clasificación con una sensibilidad baja (42.86%) 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 cinco 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 diferenciar con mayor precisión las muestras de astrocitoma, glioma mixto y oligodendroglioma, y también identificar potenciales biomarcadores que puedan contribuir a futuros estudios sobre el diagnóstico y tratamiento del cáncer de cerebro.

Primero, se utiliza la función *getGenesAnnotation* para 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_ge
ne_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
```

En el data frame *GOs* se almacenan los términos GO relacionados con el gen **PI4K2B** y el 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 **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**: la red del Golgi-trans (la red de vesículas que clasifica y envía proteínas y lípidos a sus destinos, subcompartimento del aparato de Golgi), el endosoma temprano y la 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 asociados al gen **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 estén 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** (aunque en el segundo caso aparezca NA, tras varias comprobaciones he descubierto que se refiere al gen **RAPH1**), junto con las **enfermedades** asociadas a ellos y los respectivos **puntajes de asociación**.

En primer lugar, el gen **PI4K2B** está asociado con diversas enfermedades, 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 para el lector.

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.

Por otro lado, 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.

Para concluir, podemos fijarnos en que los valores de *Affected Pathways* son más altos en ciertas enfermedades, 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 de las **enfermedades neurodegenerativas**.