PEC 3. Desarrollo del trabajo: Fase 1

Título del trabajo: Generación de un modelo de clasificación de demencia mediante técnicas de aprendizaje automático

Roberto Saborit Roig

1/4/2021

Contents

1. Descripción del avance del proyecto **2**

1.1. Grado de cumplimiento de los objetivos y resultados previstos en el plan de trabajo. 2

1.2. Justificación de los cambios en caso necesario 2

2. Relación de las actividades realizadas **2**

2.1. Actividades previstas en el plan de trabajo 2

2.2 Actividades no previstas y realizadas o programas 4

3. Relación de las desviaciones en la temporización y acciones de mitigación si procede y actualización del cronograma si procede **4**

4. Listado de los resultados parciales obtenidos hasta el momento (entregables que se adjuntan) **5**

Paquetes utilizados 5

Random forest. Variables más influyentes. 7

Varianza 0 9

Normalización 10

Preparación de los datos de entrenamiento y test 11

Aplicación de los modelos 12

Árboles de decisión (C.50) 14

Support Vector Machine 15

Curvas ROC y valor AUC 16

Optimización del modelo 20

5. Comentarios de vuestro director particular si lo consideráis necesario **32**

1. Descripción del avance del proyecto

1.1. Grado de cumplimiento de los objetivos y resultados previstos en el plan de trabajo.

El objetivo general del trabajo es:

- Establecer un modelo de machine learning que sea capaz de detectar pacientes con demencia en fase leve o muy leve, y desarrollar una aplicación web para facilitar el acceso al modelo.

Y los objetivos específicos de esta fase del trabajo son los siguientes:

- Evaluación de diferentes técnicas de arendizaje automático sobre los datos de demencia y seleccionar la que proporcione mejores resultados.
- Obtención de un modelo mediante las técnicas anteriores con un nivel mínimo de precisión del 0.8, de AUC de al menos el 0.7, y de kappa del 0.6, en la clasificación de sujetos con demencia leve/muy leve y sujetos sin demencia.
- Optimización del modelo mediante técnicas de "automatic tuning" y realización de otros cambios que puedan mejorar la precisión, y el resto de parámetros.

Los objetivos específicos de esta fase ya se han alcanzado. Se han aplicado distintos algoritmos de aprendizaje automático sobre un conjunto de datos de demencia, y los modelos se han evaluado, consiguiendo los valores mínimos que nos habíamos fijado en algunos de los modelos obtenidos. Tras esto se optimizó el modelo aplicando otra técnica de normalización y realizando atomatic tuning para obtener los mejores modelos realizando cambios en los parámetros de los algoritmos. Los resultados de los modelos generados se pueden cosultar en el apartado 4 de este documento.

1.2. Justificación de los cambios en caso necesario

La idea inicial del trabajo era realizar una prognosis de la demencia aunque, finalmente se ha optado por simplemente obtener un modelo que diagnostique la demencia en fase muy leve o leve, siendo capaz de clasificar indiviudos en sujetos con demencia leve/muy leve y sujetos sin demencia. Este cambio se ha debido a dos cuestiones, la primera que los datos que disponemos para hacer el pronóstico son los de cohorte longitudinal, y estos no son demasiado extensos, por tanto si queríamos dividirlos en 2 conjuntos para por lo menos tener dos períodos de tiempo y además dividirlos en test y entrenamiento, lo cual era un problema por la poca extensión de las observaciones. Por otro lado, ya que teníamos los datos etiquetados como demencia leve, muy leve, etc, esto nos podría servir para generar un modelo que identifique a los pacientes en esta fase que muchos todavía no tienen síntomas clínicos evidentes, lo cual serviría para realizar un diagnótico temprano que es el objetivo final del modelo. Y de esta forma podríamos utilizar los dos conjuntos de datos.

Por otro lado se han realizado cambios en los parámetros para considerar los modelos válidos. Para ello hemos seguido el criterio utilizado en el libreo de Brett Lantz: Machine learning with R. En este libro, el autor fija que un modelo para ser considerado como buen predictor, será necesario que tenga un valor kappa mínimo de 0.6 y un valor de AUC de 0.7. Además, hemos querido añadir una precisión mínima del 0.8 para nuestro modelo.

2. Relación de las actividades realizadas

2.1. Actividades previstas en el plan de trabajo

2.1.1. Búsqueda bibliográfica de trabajos similares

La primera actividad prevista en el plan de trabajo era identificar estudios que trataran de resolver problemas parecidos al propuesto en este proyecto, y de esta manera inspirarnos y ver que tipo de técnicas usan en este tipo de estudios. Para ello se realizó una búsqueda en pubmed con las palabras clave "dementia", "machine learning" y "neuroimaging". La búsqueda nos dió 491 resultados (Figura 1).

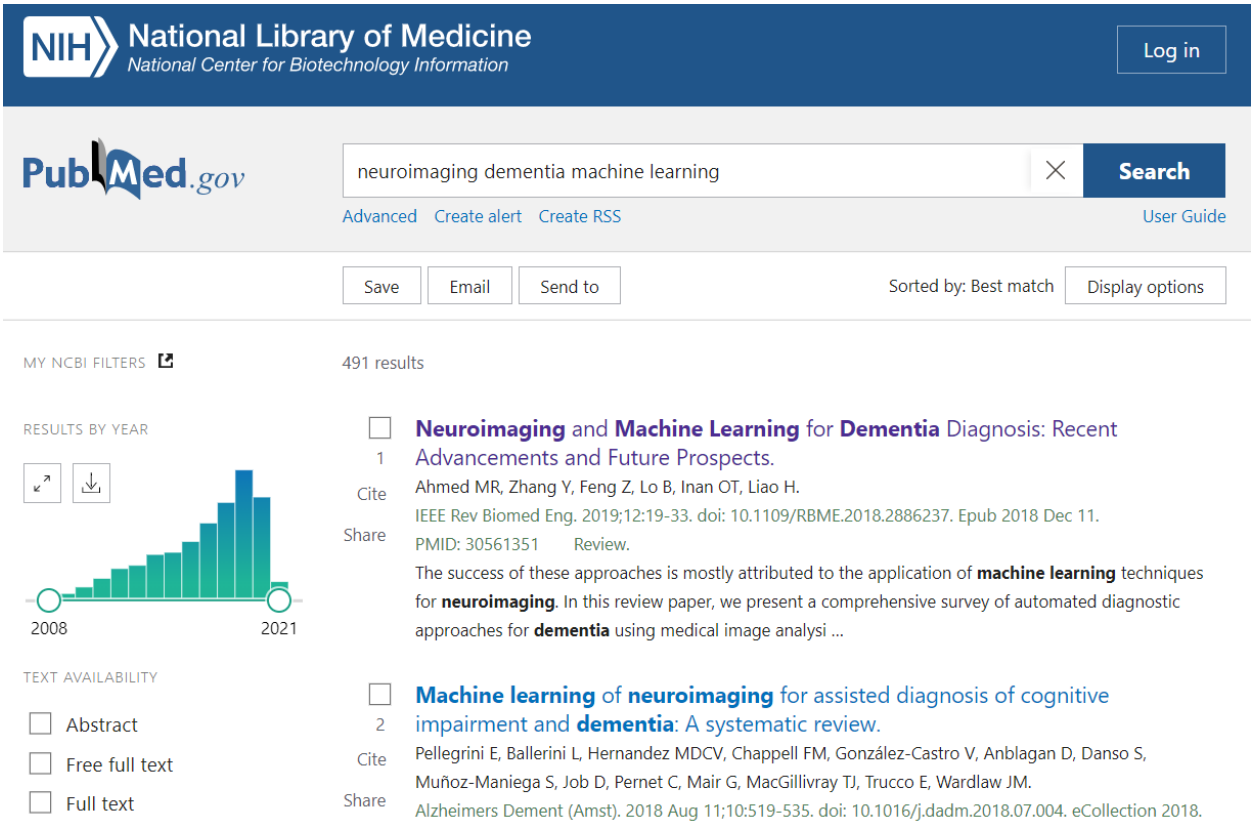


Figure 1: Búsqueda en Pubmed

La mayoría de trabajos consultados mediante esta búsqueda en Pubmed utilizan imágenes obtenidas mediante técnicas de neuroimagen, como MRI, para crear modelos que clasifiquen a los individuos en personas con demencia o sin demencia. En cambio no encontramos ningún trabajo que tratará de identificar y clasificar a los individuos en función de variables demográficas como edad o sexo, en combinación con variables de tests neuropsicológicos y de resonancia magnética como el volumen intracraneal.

En cuanto a las técnicas o algoritmos de apredizaje automático más utilizados son las redes bayesianas, las redes neuronales, los SVM (Support Vector Matrix), y los árboles de decisión. Este tipo de algoritmos aparecen continuamente en la bibliografía consultada. La mayoría de estudios utilizan AUC como parámetro de selección del mejor modelo.

2.1.2. Exploración de los datos y preprocesamiento

La siguiente actividad a realizar era la exploración de los datos y la realización del preprocesado de los mismos. Los datos fueron obtenidos del proyecto OASIS, el cual pone a libre disposición datos de resonancia magnética. De este proyecto hemos podido disponer de 2 conjuntos de datos, unos de cohorte longitudinal (oasis_longitudinal) y otros de cohorte seccional (oasis_cross_sectional), estos conjuntos nos dan datos demográficos, de resonancia magnética y neuropsicológicos de sujetos con demencia y sin ella. Al tener

dos conjuntos de datos seguimos los pasos de la exploración y el preprocesado paralelamente en ambos conjuntos. Aquí resumiremos los pasos que seguimos, pero se puede consultar el proceso completo en el documento preprocesado del github del trabajo:

https://github.com/robertosaroig/Modelo-predictivo

Lo primero fue hacer una análisis exploratorio de los dos conjuntos de datos con los siguientes objetivos:

- Ver si existen valores ausentes en el conjunto de datos y ver su distribución entre las distintas variables.
- Explorar los tipos de variables y ver si necesitamos cambiar la clase de alguna de ellas.
- Ver la distribución de las variables, tanto de la respuesta como de las explicativas.

En esta parte se descartó ya una de las variables puesto que solo tenía un nivel, por lo que no nos iba a aportar ninguna información de cara a construir el modelo final. También se vio que había muchos valores ausentes en el conjunto seccional, lo cual iba a suponer un problema y habría que eliminar muchas observaciones o bien imputarlos. También se vio la distribución y la capacidad predictora de las variables mediante un análisis Random Forest. Por último, se analizó si alguna variable tenía varianza igual o cercana a 0, ya que esto significa que le poder predictivo de esa variable no sería significativo y por tanto la eliminaríamos, pero no fue así, ya que todas las variables nos dieron negativo en este anásis.

En el preprocesado de los datos realizados una normalización, eliminamos las observaciones con valores ausentes, ya que teníamos valores ausentes en la variable respuesta, por tanto no podíamos imputarlos. Y finalmente se prepararon los datos de entrenamiento y los de test solo con las variables predictoras que utilizaríamos para la aplicación del modelos, y por otro lado un vector con las variables respuesta.

2.1.3. Aplicación del modelo, evaluación y optimiación

Para la aplicación del modelo se utilizaron distintos algoritmos clasificatorios de machine learning, entre los cuales se encuentran el algoritmo k-NN, redes bayesianas, árboles de decisión y SVMs. También se pretendía aplicar redes neuronales, pero finalmente se descartó este último por errores en el código que no se supieron solucionar. Se utilizó el conjunto de datos longitudinal para el entrenamiento de este modelo, ya que tenía menos datos tras la aplicación del preprocesado.

Tras aplicar los algoritmos se estudiaron los resultados de los modelos construidos fijándonos en los valores de preicisión o accuracy, en el valor kappa y en el valor que nos dio el AUC de la curva ROC. En base a estos resultados elegimos el mejor modelo.

2.2 Actividades no previstas y realizadas o programas

Hasta este punto las actividades previstas se realizaron todas, y la mayoría del plan previsto se ha seguido. El único punto extra que se ha realizado es la curva ROC y la valoración del modelo mediante el parámetro AUC, ya que en nuestro objetivo nos fijamos utilizar la precisión del modelo como el parámetro para valorar si este era suficientemente bueno prediciendo.

Otro análisis que se realizó y no estaba previsto, es que dentro del preprocesado se realizó un test de varianza para ver si alguna tenía varianza 0, lo que haría que no aportará valor al modelo, y por tanto sería eliminada, aunque en este análisis no obtuvimos ninguna varianza igual a 0.

3. Relación de las desviaciones en la temporización y acciones de mitigación si procede y actualización del cronograma si procede

Dentro de nuestro cronograma se retrasó el proceso de normalización y por tanto, en el plan de trabajo le dedicamos más tiempo del precisado en este. Por tanto tuvimos que reducir el tiempo de la siguiente

actividad y realizarla a la vez que las siguientes. Esto no fue demasiado problemático ya que la siguiente actividad era buscar la información sobre los modelos de machine learning adecuados para nuestro proyecto, lo que pudimos solapar con la propia aplicación de los modelos en nuestros datos y reducir así el tiempo total, para recuperar los días extra perdidos en la normalización.

Las desviaciones del trabajo, por tanto, se recuperaron durante esta misma fase, deonde se han podido cumplir los objetivos y por tanto el coronograma no ha cambiado (Figura 2). A partir de la siguiente fase, comenzaremos con la generación de la aplicación web para que el modelo pueda utilizarse sin problema por cualquier usuario introduciendo los datos necesarios.

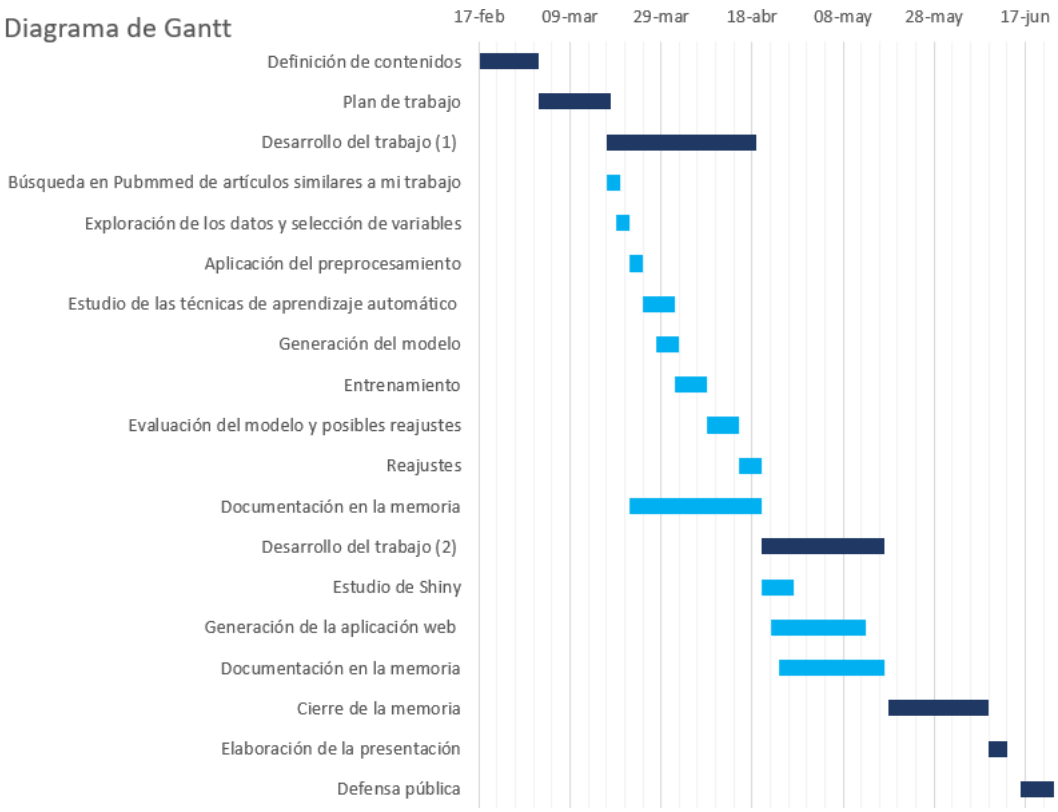


Figure 2: Diagrama de Gantt sobre el plan de trabajo del proyecto

4. Listado de los resultados parciales obtenidos hasta el momento (entregables que se adjuntan)

Paquetes utilizados

A continuación se muestran todos los paquetes utilizados:

```
library(ggpubr) # Realización de gráficos
```

```
## Loading required package: ggplot2
```

```r
library(randomForest) # Para la realización del análisis random forest
```

```
## randomForest 4.6-14

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(tidyverse) # Para la función map_if(), que nos sirve para generar el análisis de random forest
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --

## v tibble  3.0.3     v dplyr   1.0.2
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## v purrr   0.3.4

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::combine()       masks randomForest::combine()
## x dplyr::filter()        masks stats::filter()
## x dplyr::lag()           masks stats::lag()
## x randomForest::margin() masks ggplot2::margin()
```

```r
library(caret) # Paquete en el que están algunos de los algoritmos de machine learning utilizados
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
require(reshape) # Para utilizar la funcion rename, que sirve ara renombrar columnas y si las de un dat
```

```
## Loading required package: reshape

##
## Attaching package: 'reshape'

## The following object is masked from 'package:dplyr':
##
##     rename
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, smiths
```

```
library(class) # Para utilizar la función k-NN, algoritmo de machine learning que uilizamos para genera
```

```
##
## Attaching package: 'class'
```

```
## The following object is masked from 'package:reshape':
##
##     condense
```

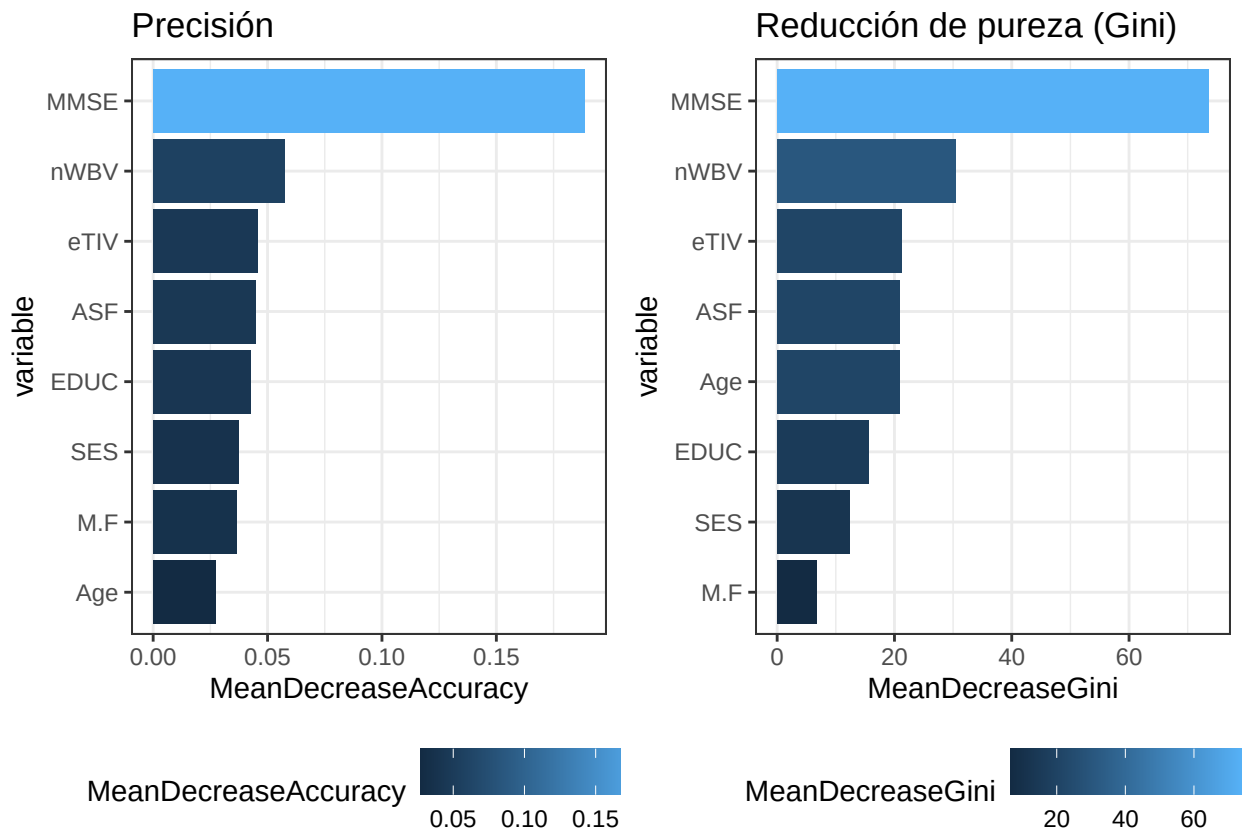
```
library(e1071) # Para el algoritmo de SVM
library(gmodels) # Para utilizar una de las funciones de evaluación de los modelos
library(C50) # Para los árboles de decisión
library(dplyr)# Para la aplicación de funciones para manipulación de los datos como select()
library(ROCR) # Para generar las curvas ROC y obtener los valores AUC.
```

Random forest. Variables más influyentes.

Dentro de la exploración de las variables un resultado interesante es la influencia de cada una de las variables sobre la variable respuesta: `Group`, que divide los datos en demencia y no demencia. Para ello crearemos un gráfico que nos dé la importancia de cada variable.

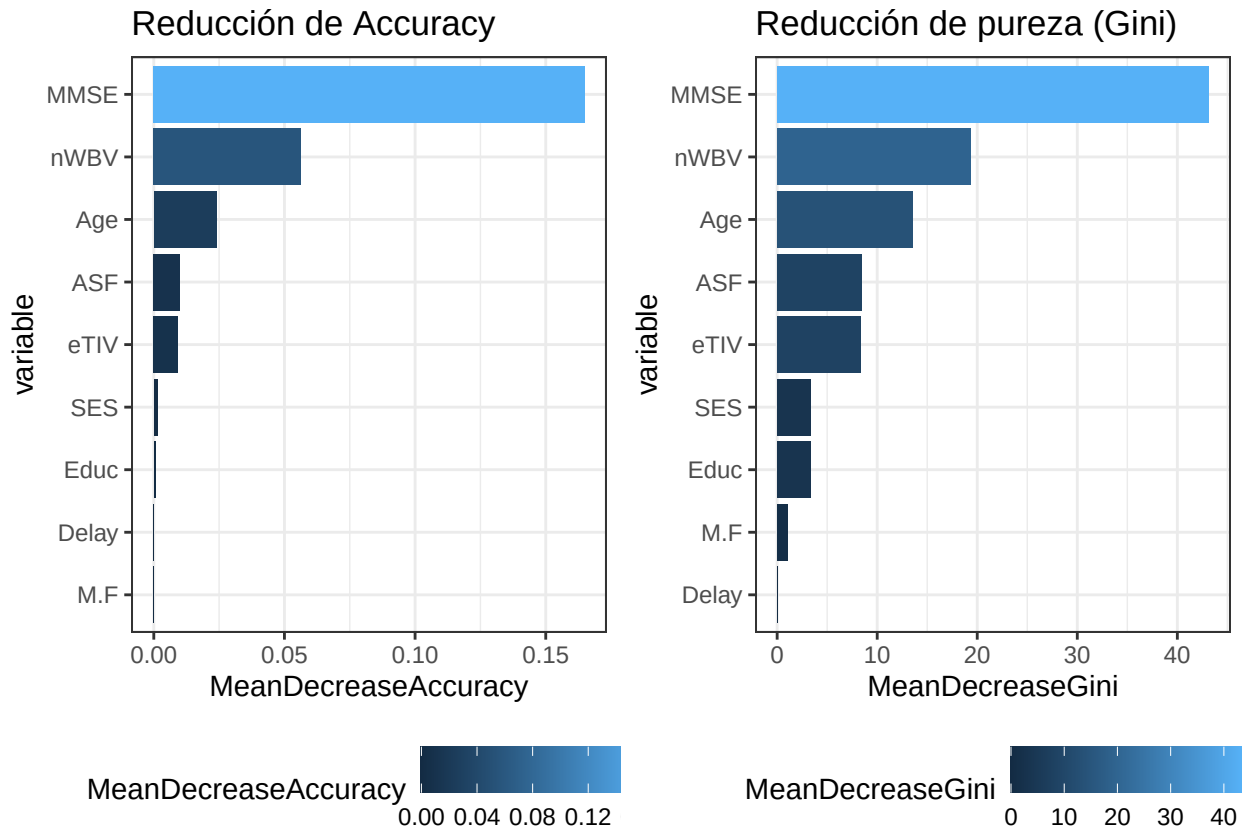
```
datos_rf <- oasis_longitudinal %>%
            select(-`Subject ID`, -`MRI ID`, -`MR Delay`, -Visit, -Hand, -CDR) #Seleccionamos las varia

datos_rf <- map_if(.x = datos_rf, .p = is.character, .f = as.factor) %>%
  as.data.frame()
modelo_randforest <- randomForest(formula = Group ~ . ,
                                  data = na.omit(datos_rf),
                                  mtry = 5,
                                  importance = TRUE,
                                  ntree = 1000) # Generamos el modelo del
#random forest con Group como respuesta u el resto de variables que no hemos
#quitado como predictoras
importancia <- as.data.frame(modelo_randforest$importance)
# Sacamos el valor de importancia del modelo generado
importancia <- rownames_to_column(importancia,var = "variable")

p1 <- ggplot(data = importancia, aes(x = reorder(variable, MeanDecreaseAccuracy),
                                y = MeanDecreaseAccuracy,
                                fill = MeanDecreaseAccuracy)) +
      labs(x = "variable", title = "Precisión") +
      geom_col() +
      coord_flip() +
      theme_bw() +
      theme(legend.position = "bottom") # Generamos un gráfico para
#visualizar cuales obtienen mejor puntuación en cuanto a importancia
p2 <- ggplot(data = importancia, aes(x = reorder(variable, MeanDecreaseGini),
```

```
                                y = MeanDecreaseGini,
                                fill = MeanDecreaseGini)) +
      labs(x = "variable", title = "Reducción de pureza (Gini)") +
      geom_col() +
      coord_flip() +
      theme_bw() +
      theme(legend.position = "bottom")
ggarrange(p1, p2)
```




```
# Y generamos otro gráfico igual con la reducción de la pureza

# Hacemos lo mismo con los datos seccionales:
datos_rf <- oasis_cross_sectional_narm %>% select(-ID, -Hand, -CDR)
datos_rf <- map_if(.x = datos_rf, .p = is.character, .f = as.factor) %>% as.data.frame()
modelo_randforest <- randomForest(formula = Group ~ . ,
                                  data = na.omit(datos_rf),
                                  mtry = 5,
                                  importance = TRUE,
                                  ntree = 1000)
importancia <- as.data.frame(modelo_randforest$importance)
importancia <- rownames_to_column(importancia,var = "variable")

p1 <- ggplot(data = importancia, aes(x = reorder(variable, MeanDecreaseAccuracy),
                                y = MeanDecreaseAccuracy,
                                fill = MeanDecreaseAccuracy)) +
```

```
      labs(x = "variable", title = "Reducción de Accuracy") +
      geom_col() +
      coord_flip() +
      theme_bw() +
      theme(legend.position = "bottom")

p2 <- ggplot(data = importancia, aes(x = reorder(variable, MeanDecreaseGini),
                                     y = MeanDecreaseGini,
                                     fill = MeanDecreaseGini)) +
      labs(x = "variable", title = "Reducción de pureza (Gini)") +
      geom_col() +
      coord_flip() +
      theme_bw() +
      theme(legend.position = "bottom")
ggarrange(p1, p2)
```



Ambos análisis, apuntan a que las variables con mayor influencia sobre la variable `Group` son `MMSE`, que es el test cognitivo y `nWBV` que es el bvolumen intracraneal normalizado.

Varianza 0

El test de varianza cercana a 0, nos servirá para descartar aquellas variables donde la varianza sea 0 o muy cercana, puesto que estas no tendrán ningún poder predictivo. El test nos ofrecerá una columna donde se ve FALSE si no tiene varianza cercana a 0 y TRUE si sí la tiene.

```
oasis_longitudinal %>%
  select(Age, `M/F`, EDUC, SES, MMSE, CDR, eTIV, nWBV, ASF) %>%
  nearZeroVar(saveMetrics = TRUE)
```

```
##      freqRatio percentUnique zeroVar   nzv
## Age   1.181818     10.455764   FALSE FALSE
## M/F   1.331250      0.536193   FALSE FALSE
## EDUC  1.271605      3.217158   FALSE FALSE
## SES   1.170455      1.340483   FALSE FALSE
## MMSE  1.252747      4.825737   FALSE FALSE
## CDR   1.674797      1.072386   FALSE FALSE
## eTIV  1.000000     99.463807   FALSE FALSE
## nWBV  1.000000    100.000000   FALSE FALSE
## ASF   1.000000     99.463807   FALSE FALSE
```

```
# Aplicamos la función nearZeroVar a las variables seleccionadas
```

Como vemos no se muestra ningún dato con zeroVar=TRUE, por lo que descartamos que alguna de las variables tenga varianza cercana a 0.

Normalización

Para la normalización aplicamos una fórmula muy sencilla que resta el mínimo a cada valor y divide el resultado entre la resta del mínimo y el máximo, de esta forma obtenemos los datos normalizados de cada variable a valores entre 0 y 1.

```
normalize <- function(x) {
  return ((x - min(x)) / (max(x) - min(x)))
  }
# Creamos la función

oasis_longitudinal_n=as.data.frame(lapply(oasis_longitudinal_narm_2[8:15], normalize))
# Aplicamos la función sobre oasis_longitudinal al que ya habíamos quitado
# previamente los valores NA, escogemos solo las variables numéricas

oasis_cross_sectional_n=as.data.frame(lapply(oasis_cross_sectional_narm[4:11], normalize))
# Aplicamos la función de normalización sobre oasis_cross_sectional también
# sin valores NA, y escogemos solo las variables numéricas

head(oasis_longitudinal_n)
```

```
##         Age      EDUC  SES      MMSE CDR      eTIV      nWBV       ASF
## 1 0.7500000 0.4705882 0.25 0.7692308 0.0 1.0000000 0.2055591 0.0000000
## 2 0.7777778 0.7058824 0.50 0.8461538 0.0 0.1071569 0.2812347 0.8248536
## 3 0.5555556 0.3529412 0.75 0.8461538 0.0 0.6550395 0.2924680 0.2293647
## 4 0.9166667 0.4705882 0.25 1.0000000 0.0 0.1721966 0.2139869 0.7309537
## 5 0.2222222 0.3529412 0.25 0.7692308 0.5 0.3864757 0.8276781 0.4729129
## 6 0.1666667 0.3529412 0.50 1.0000000 0.5 0.3749682 0.6153903 0.4850955
```

```r
head(oasis_cross_sectional_n)
```

```
##         Age Educ  SES      MMSE CDR       eTIV      nWBV       ASF
## 1 0.6507937 0.25 0.50 0.9333333 0.0 0.25431530 0.5025381 0.6231672
## 2 0.3492063 0.75 0.00 0.9333333 0.0 0.02761795 0.8426396 0.9530792
## 3 0.6349206 0.75 0.50 0.8000000 0.5 0.38089758 0.3248731 0.4780059
## 4 0.6507937 1.00 0.25 1.0000000 0.0 0.59033372 0.2284264 0.2815249
## 5 0.3015873 0.50 0.25 1.0000000 0.0 0.22784810 0.9289340 0.6568915
## 6 0.7619048 1.00 0.25 1.0000000 0.0 0.62255466 0.1776650 0.2551320
```

Ahora todas las variables tienen valores comprendidos entre 0 y 1.

Preparación de los datos de entrenamiento y test

Una vez hecho la normalización y los diversos pasos de preprocesamiento, necesitamos preparar los conjuntos de datos de manera que tengamos exactamente las mismas variables, con los mismos nombres y asegurarnos de que sean todas numéricas, en un conjunto de datos para el entrenamiento y en otro para el test. Además crearemos un vector para cada conjunto que contenga los grupos a los que pertenece cada observación: Demencia y no demencia.

```r
oasis_longitudinal_n = rename(oasis_longitudinal_n, c(EDUC="Educ"))
# Cambiamos el nombre de la variable educ para que esté igual en ambos
# conjuntos de datos

oasis_longitudinal_n2=cbind(oasis_longitudinal_narm_2$`M/F`, oasis_longitudinal_n) # Tras normalizarlo
oasis_cross_sectional_n2=cbind(oasis_cross_sectional_narm$M.F, oasis_cross_sectional_n)

# Cambiamos el nombre de esta variable a Sex, para que sea igual en ambos conjuntos
oasis_longitudinal_n3 = rename(oasis_longitudinal_n2, c("oasis_longitudinal_narm_2$`M/F`"="Sex"))
names(oasis_longitudinal_n3)
```

```
## [1] "Sex"  "Age"  "Educ" "SES"  "MMSE" "CDR"  "eTIV" "nWBV" "ASF"
```

```r
oasis_cross_sectional_n3 = rename(oasis_cross_sectional_n2, c(`oasis_cross_sectional_narm$M.F`="Sex"))
names(oasis_cross_sectional_n3) # ya tenemos todos los nombres iguales lo que nos evitará problemas en
```

```
## [1] "Sex"  "Age"  "Educ" "SES"  "MMSE" "CDR"  "eTIV" "nWBV" "ASF"
```

```r
oasis_longitudinal_n4=select(oasis_longitudinal_n3, -CDR) # Borramos la variable CDR ya que es realment
oasis_cross_sectional_n4=select(oasis_cross_sectional_n3, -CDR)


oasis_longitudinal_n4$Sex=factor(oasis_longitudinal_n4$Sex,levels=c("F", "M"),
                           labels=c(0,1)) # Convertimos la variable Sex a números f=0 y m=1, ya que alg

oasis_cross_sectional_n4$Sex=factor(oasis_cross_sectional_n4$Sex,levels=c("F", "M"),
                           labels=c(0,1))


test=oasis_longitudinal_n4 #Los datos longitudinales serán los del test
```

```r
train=oasis_cross_sectional_n4 #Los seccionales el entrenamiento
test_labels=oasis_longitudinal_narm_2$Group #Guardamos los labels en otro vector
train_labels=oasis_cross_sectional_narm$Group
test_labels=as.factor(test_labels) #Lo convertimos a factor

head(test)
```

```
##   Sex       Age      Educ  SES      MMSE      eTIV      nWBV       ASF
## 1   1 0.7500000 0.4705882 0.25 0.7692308 1.0000000 0.2055591 0.0000000
## 2   0 0.7777778 0.7058824 0.50 0.8461538 0.1071569 0.2812347 0.8248536
## 3   1 0.5555556 0.3529412 0.75 0.8461538 0.6550395 0.2924680 0.2293647
## 4   0 0.9166667 0.4705882 0.25 1.0000000 0.1721966 0.2139869 0.7309537
## 5   1 0.2222222 0.3529412 0.25 0.7692308 0.3864757 0.8276781 0.4729129
## 6   0 0.1666667 0.3529412 0.50 1.0000000 0.3749682 0.6153903 0.4850955
```

```r
head(train)
```

```
##   Sex       Age Educ  SES      MMSE       eTIV      nWBV       ASF
## 1   0 0.6507937 0.25 0.50 0.9333333 0.25431530 0.5025381 0.6231672
## 2   0 0.3492063 0.75 0.00 0.9333333 0.02761795 0.8426396 0.9530792
## 3   0 0.6349206 0.75 0.50 0.8000000 0.38089758 0.3248731 0.4780059
## 4   1 0.6507937 1.00 0.25 1.0000000 0.59033372 0.2284264 0.2815249
## 5   0 0.3015873 0.50 0.25 1.0000000 0.22784810 0.9289340 0.6568915
## 6   0 0.7619048 1.00 0.25 1.0000000 0.62255466 0.1776650 0.2551320
```

```r
head(test_labels)
```

```
## [1] Nondemented Nondemented Nondemented Nondemented Demented    Demented
## Levels: Demented Nondemented
```

```r
head(train_labels)
```

```
## [1] Nondemented Nondemented Demented    Nondemented Nondemented Nondemented
## Levels: Nondemented Demented
```

Como vemos en estos 4 datasets tenemos guardada toda la información para realizar el entrenamiento de los modelos y la evaluación de los mismos.

Aplicación de los modelos

k-NN

El primero algoritmo que probamos es el k-NN. Este es un algoritmo de clasificación, con un parámetro que es la k, que podemos variar para encontrar el mejor modelo. Tras probar k [1:10], vimos que el mejor resultado lo daba k=9, como se muestra a continuación:

```r
knn_model <- knn(train = train, test = test, cl = train_labels, k = 9, prob=TRUE) # Generamos el modelo
prob=attr(knn_model,"prob")
# Obtenemos las probabilidades para luego sacar la curva ROC y el valor AUC
prob_knn <- ifelse(knn_model =="Demented", 1-prob, prob) # Restamos 1 a las
```

```r
#probabilidades que no sean "Demented", ya que realmente las probabilidades
#que nos saca el modelo son las de que el valor obtenido sea el correcto,
#pero para generar la curva ROC, necesitamos las probabilidades de que el
#valor obtenido sea el positivo
confusionMatrix(test_labels, knn_model, positive = "Demented")
```

```
## Warning in confusionMatrix.default(test_labels, knn_model, positive =
## "Demented"): Levels are not in the same order for reference and data.
## Refactoring data to match.
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    Nondemented Demented
##   Nondemented          67        5
##   Demented             17       39
##
##                Accuracy : 0.8281
##                  95% CI : (0.7514, 0.889)
##     No Information Rate : 0.6562
##     P-Value [Acc > NIR] : 1.232e-05
##
##                   Kappa : 0.6423
##
##  Mcnemar's Test P-Value : 0.01902
##
##             Sensitivity : 0.8864
##             Specificity : 0.7976
##          Pos Pred Value : 0.6964
##          Neg Pred Value : 0.9306
##              Prevalence : 0.3438
##          Detection Rate : 0.3047
##    Detection Prevalence : 0.4375
##       Balanced Accuracy : 0.8420
##
##        'Positive' Class : Demented
##
```

```r
# Esta función nos da una evaluación del modelo
```

Bayes

El alogritmo bayes también se utiliza para clasificación de un conjunto de observaciones en grupos:

```r
bayes_model <- naiveBayes(train, train_labels) # Entrenamos el modelo
pred_bayes <- predict(bayes_model, test) # Obtenemos las predicciones que
# hace el modelo con los datos de test

confusionMatrix(test_labels, pred_bayes, positive = "Demented")
```

```
## Warning in confusionMatrix.default(test_labels, pred_bayes, positive =
## "Demented"): Levels are not in the same order for reference and data.
## Refactoring data to match.
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    Nondemented Demented
##   Nondemented          67        5
##   Demented             18       38
##
##                Accuracy : 0.8203
##                  95% CI : (0.7427, 0.8826)
##     No Information Rate : 0.6641
##     P-Value [Acc > NIR] : 6.422e-05
##
##                   Kappa : 0.6253
##
##  Mcnemar's Test P-Value : 0.01234
##
##             Sensitivity : 0.8837
##             Specificity : 0.7882
##          Pos Pred Value : 0.6786
##          Neg Pred Value : 0.9306
##              Prevalence : 0.3359
##          Detection Rate : 0.2969
##    Detection Prevalence : 0.4375
##       Balanced Accuracy : 0.8360
##
##        'Positive' Class : Demented
##
```

```
# Evaluamos el modelo
```

Árboles de decisión (C.50)

```
c50_model <- C5.0(train, train_labels) # Entrenamos el modelo
prc50=predict(c50_model, test) # Obtenemos las predicciones
confusionMatrix(test_labels, prc50, positive = "Demented") #Lo evaluamos
```

```
## Warning in confusionMatrix.default(test_labels, prc50, positive = "Demented"):
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    Nondemented Demented
##   Nondemented          71        1
##   Demented             19       37
##
##                Accuracy : 0.8438
##                  95% CI : (0.7691, 0.9019)
##     No Information Rate : 0.7031
##     P-Value [Acc > NIR] : 0.0001753
```

```
##
##                  Kappa : 0.6708
##
##  Mcnemar's Test P-Value : 0.0001439
##
##             Sensitivity : 0.9737
##             Specificity : 0.7889
##          Pos Pred Value : 0.6607
##          Neg Pred Value : 0.9861
##              Prevalence : 0.2969
##          Detection Rate : 0.2891
##    Detection Prevalence : 0.4375
##       Balanced Accuracy : 0.8813
##
##        'Positive' Class : Demented
##
```

Support Vector Machine

```
datalabels=cbind(train_labels, train) # En este caso tensmo que unir los
#datos con los labels por como se va a entrenar el modelo
tlb=cbind(test_labels, test) # Hacemos lo mismo con los datos de test

svm_model <- svm(train_labels ~ ., data = datalabels, probability=TRUE)
# Generamos el modelo
pred <- predict(svm_model, tlb, probability=TRUE)
# Obtenmos las predicciones
confusionMatrix(test_labels, pred, positive = "Demented")
```

```
## Warning in confusionMatrix.default(test_labels, pred, positive = "Demented"):
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    Nondemented Demented
##   Nondemented          61       11
##   Demented             15       41
##
##                Accuracy : 0.7969
##                  95% CI : (0.7167, 0.8628)
##     No Information Rate : 0.5938
##     P-Value [Acc > NIR] : 8.459e-07
##
##                   Kappa : 0.584
##
##  Mcnemar's Test P-Value : 0.5563
##
##             Sensitivity : 0.7885
##             Specificity : 0.8026
```

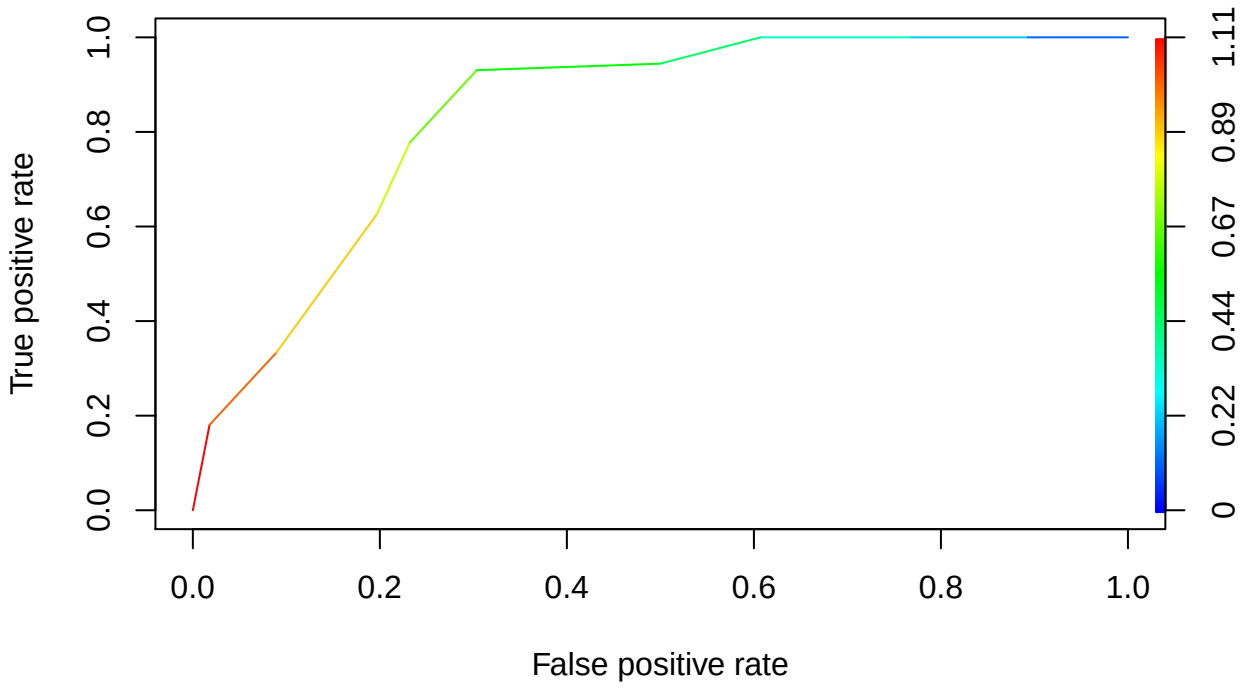
```
##          Pos Pred Value : 0.7321
##          Neg Pred Value : 0.8472
##              Prevalence : 0.4062
##          Detection Rate : 0.3203
##    Detection Prevalence : 0.4375
##       Balanced Accuracy : 0.7955
##
##        'Positive' Class : Demented
##
```

```
# Evaluamos
prob_SVM=attr(pred, "probabilities")
# Obtenemos las probabilidades
```

Curvas ROC y valor AUC

Una vez tenemos los modelos, podemos realizar las curvas ROC y el valor AUC, que será un parámetro importante a la hora de decir cual es el mejor de los modelos generados.

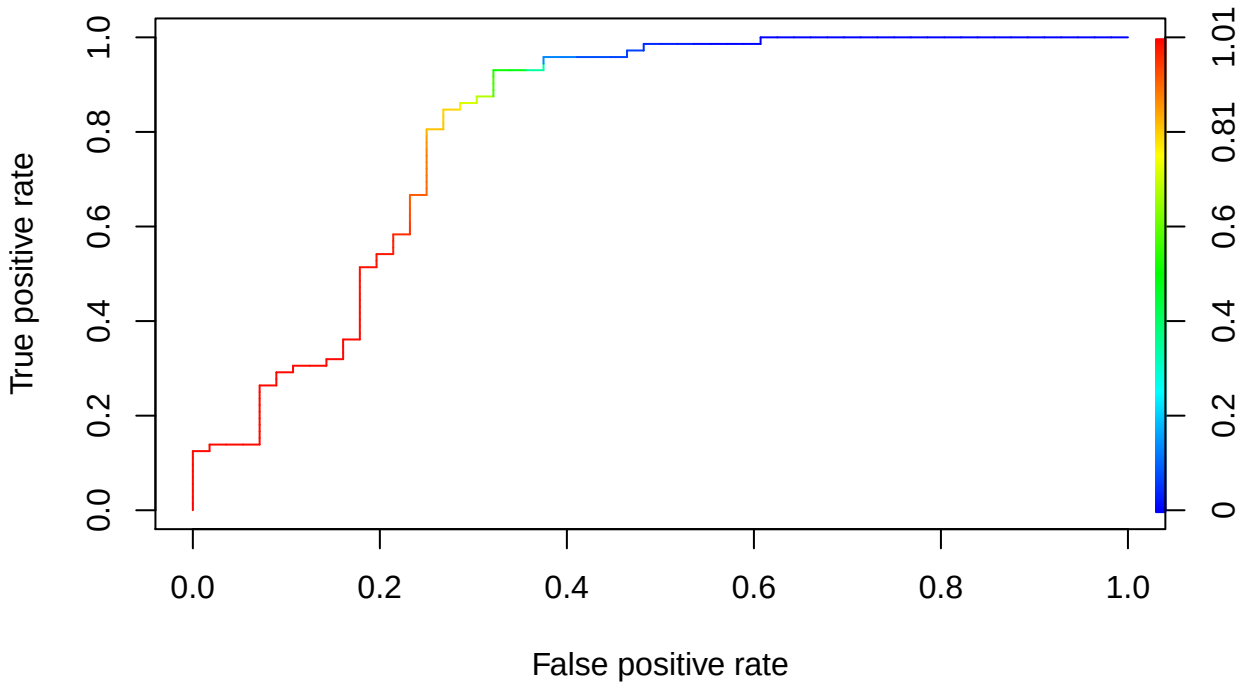
```
#knn
predicciones=prediction(prob_knn, test_labels)
# Obtenemos las prediciones con las probabilidades

perf=performance(predicciones, "tpr", "fpr") # La funcion performance
# nos da el true positive ratio false positive ratio para la curva ROC
plot(perf, colorize=TRUE) # Generamos la gráfica
```




```
perf.auc <- performance(predicciones, measure = "auc") #Obtenemos el AUC
AUC=unlist(perf.auc@y.values)
AUC
```

```
## [1] 0.8385417
```

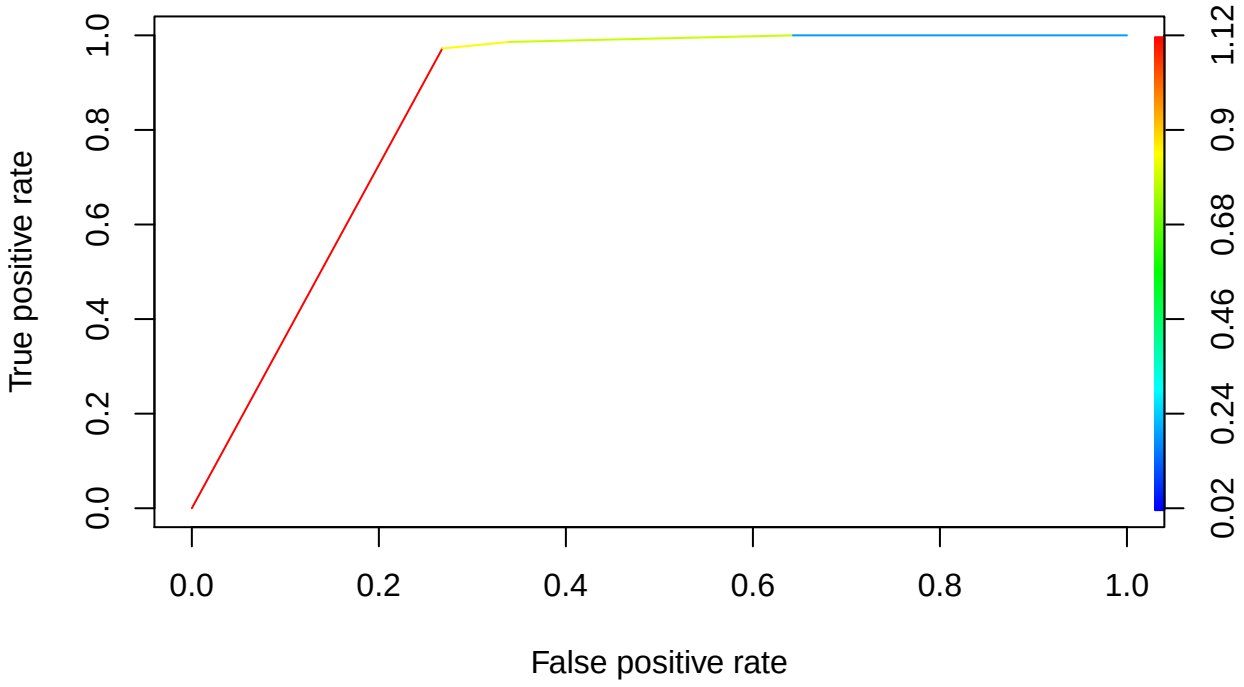
```
#bayes
pred <- predict(bayes_model, test, type="raw")
predicciones=prediction(pred[,1], test_labels)

perf=performance(predicciones, "tpr", "fpr")
plot(perf, colorize=TRUE)
```




```
perf.auc <- performance(predicciones, measure = "auc")
AUC=unlist(perf.auc@y.values)
AUC
```

```
## [1] 0.813244
```

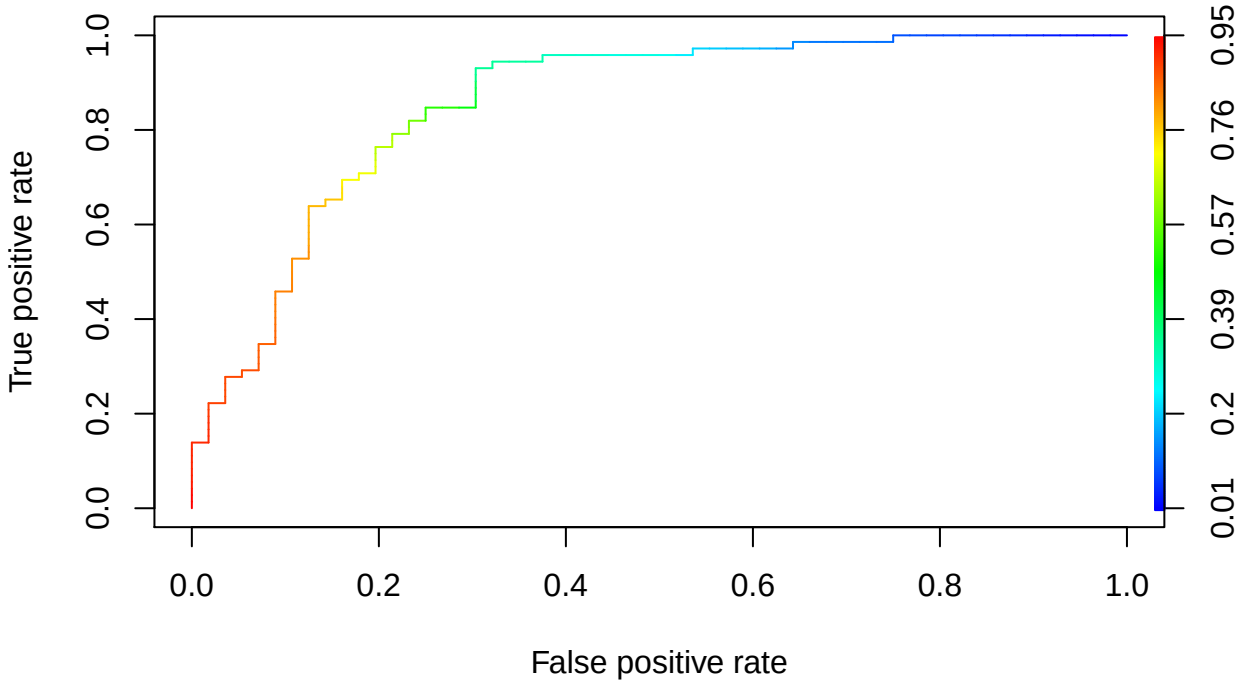
```
#Árboles de decisión
pred <- predict(c50_model, test, type = "prob")
predicciones=prediction(pred[,1], test_labels)

perf=performance(predicciones, "tpr", "fpr")
plot(perf, colorize=TRUE)
```




```
perf.auc <- performance(predicciones, measure = "auc")
AUC=unlist(perf.auc@y.values)
AUC
```

```
## [1] 0.858755
```

```
#SVM
predicciones=prediction(prob_SVM[,1], test_labels)

perf=performance(predicciones, "tpr", "fpr")
plot(perf, colorize=TRUE)
```




```
perf.auc <- performance(predicciones, measure = "auc")
AUC=unlist(perf.auc@y.values)
AUC
```

```
## [1] 0.8573909
```

Como vemos las curvas ROC y los valores AUC obtenidos son bastante satisfactorios en todos los casos, aunque las curvas de los modelos k-NN y árboles de decisión, no son escalonadas como lo son habitualmente las curvas ROC, y nos hace sospechar que el valor AUC estará algo inflado.

Optimización del modelo

Normalización z score

Para tratar de optimizar algo más el modelo probamos una normalización distinta basada en el z score, en vez, de en el máximo y mínimo. Y probamos otra vez los modelos con esta normalización a ver si mejora el resultado.

```
#Normalización z score
oasis_longitudinal_z <- as.data.frame(scale(oasis_longitudinal_narm_2[8:15]))

oasis_seccional_z <- as.data.frame(scale(oasis_cross_sectional_narm[4:11]))
```

```
oasis_longitudinal_z = rename(oasis_longitudinal_z, c(EDUC="Educ"))

#Tras normalizarlo añadimos la variable sexo al conjunto
oasis_longitudinal_z2=cbind(oasis_longitudinal_narm_2$`M/F`, oasis_longitudinal_z)

#Camiamos el nombre de esta variable a Sex
oasis_longitudinal_z3 = rename(oasis_longitudinal_z2, c("oasis_longitudinal_narm_2$`M/F`"="Sex"))



#Tras normalizarlo añadimos la variable sexo al conjunto
oasis_seccional_z2=cbind(oasis_cross_sectional_narm$M.F, oasis_seccional_z)

#Camiamos el nombre de esta variable a Sex
oasis_seccional_z3 = rename(oasis_seccional_z2, c("oasis_cross_sectional_narm$M.F"="Sex"))

names(oasis_longitudinal_z3)
```

```
## [1] "Sex"  "Age"  "Educ" "SES"  "MMSE" "CDR"  "eTIV" "nWBV" "ASF"
```

```
names(oasis_seccional_z3)
```

```
## [1] "Sex"  "Age"  "Educ" "SES"  "MMSE" "CDR"  "eTIV" "nWBV" "ASF"
```

```
#Borramos la variable CDR
oasis_longitudinal_z4=select(oasis_longitudinal_z3, -CDR)
oasis_seccional_z4=select(oasis_seccional_z3, -CDR)


#Convertimos por último la variable Sex a números f=0 y m=1
oasis_longitudinal_z4$Sex=factor(oasis_longitudinal_z4$Sex,levels=c("F", "M"),
                           labels=c(0,1))

oasis_seccional_z4$Sex=factor(oasis_seccional_z4$Sex,levels=c("F", "M"),
                           labels=c(0,1))


test_z=oasis_longitudinal_z4
train_z=oasis_seccional_z4
test_labels=oasis_longitudinal_narm_2$Group
train_labels=oasis_cross_sectional_narm$Group
test_labels=as.factor(test_labels)

#knn
knn_9=knn(train = train_z, test = test_z, train_labels, k=9, prob=TRUE)
confusionMatrix(test_labels, knn_9, positive = "Demented")
```

```
## Warning in confusionMatrix.default(test_labels, knn_9, positive = "Demented"):
## Levels are not in the same order for reference and data. Refactoring data to
## match.

## Confusion Matrix and Statistics
```

```
##
##              Reference
## Prediction    Nondemented Demented
##   Nondemented          67        5
##   Demented             23       33
##
##                Accuracy : 0.7812
##                  95% CI : (0.6996, 0.8495)
##     No Information Rate : 0.7031
##     P-Value [Acc > NIR] : 0.030398
##
##                   Kappa : 0.5391
##
##  Mcnemar's Test P-Value : 0.001315
##
##             Sensitivity : 0.8684
##             Specificity : 0.7444
##          Pos Pred Value : 0.5893
##          Neg Pred Value : 0.9306
##              Prevalence : 0.2969
##          Detection Rate : 0.2578
##    Detection Prevalence : 0.4375
##       Balanced Accuracy : 0.8064
##
##        'Positive' Class : Demented
##
```

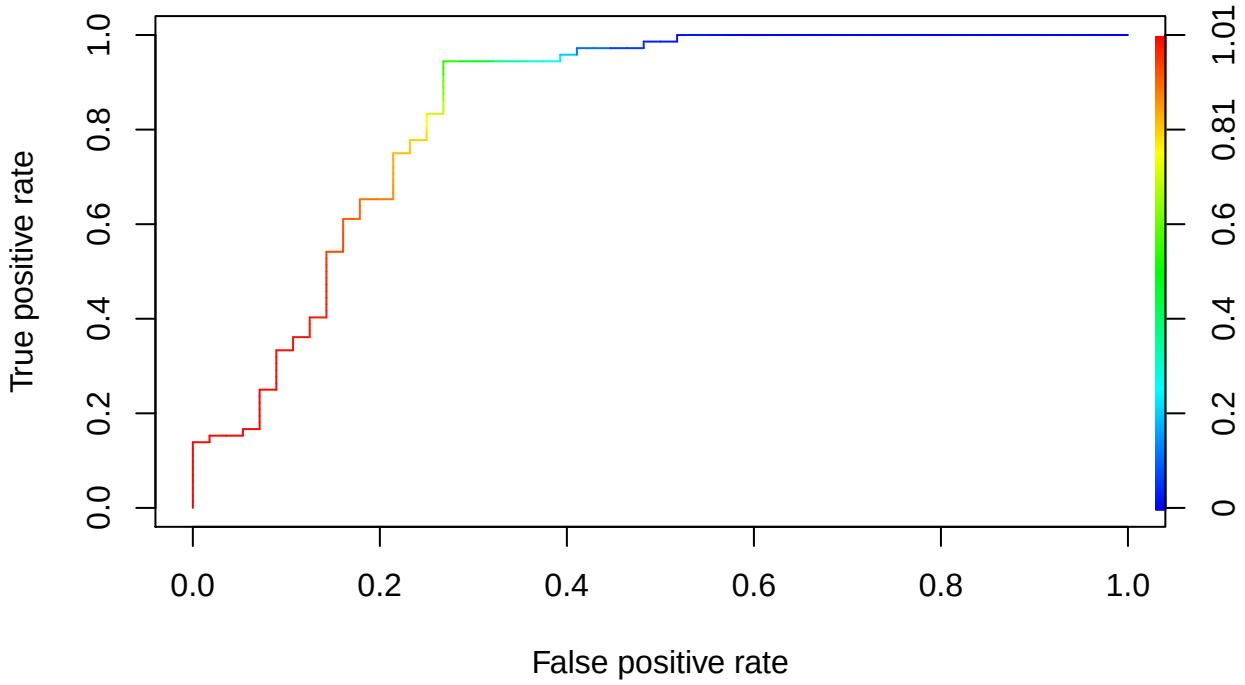
```
#Bayes
bayes_model <- naiveBayes(train_z, train_labels)
pred_bayes <- predict(bayes_model, test_z)
confusionMatrix(test_labels, pred_bayes, positive = "Demented")
```

```
## Warning in confusionMatrix.default(test_labels, pred_bayes, positive =
## "Demented"): Levels are not in the same order for reference and data.
## Refactoring data to match.
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    Nondemented Demented
##   Nondemented          68        4
##   Demented             15       41
##
##                Accuracy : 0.8516
##                  95% CI : (0.7779, 0.9082)
##     No Information Rate : 0.6484
##     P-Value [Acc > NIR] : 2.347e-07
##
##                   Kappa : 0.6917
##
##  Mcnemar's Test P-Value : 0.02178
##
##             Sensitivity : 0.9111
```

```
##             Specificity : 0.8193
##          Pos Pred Value : 0.7321
##          Neg Pred Value : 0.9444
##              Prevalence : 0.3516
##          Detection Rate : 0.3203
##    Detection Prevalence : 0.4375
##       Balanced Accuracy : 0.8652
##
##        'Positive' Class : Demented
##
```

```
pred_bayes <- predict(bayes_model, test_z, type="raw")
predicciones=prediction(pred_bayes[,1], test_labels)

perf=performance(predicciones, "tpr", "fpr")
plot(perf, colorize=TRUE)
```




```
perf.auc <- performance(predicciones, measure = "auc")
AUC=unlist(perf.auc@y.values)
AUC
```

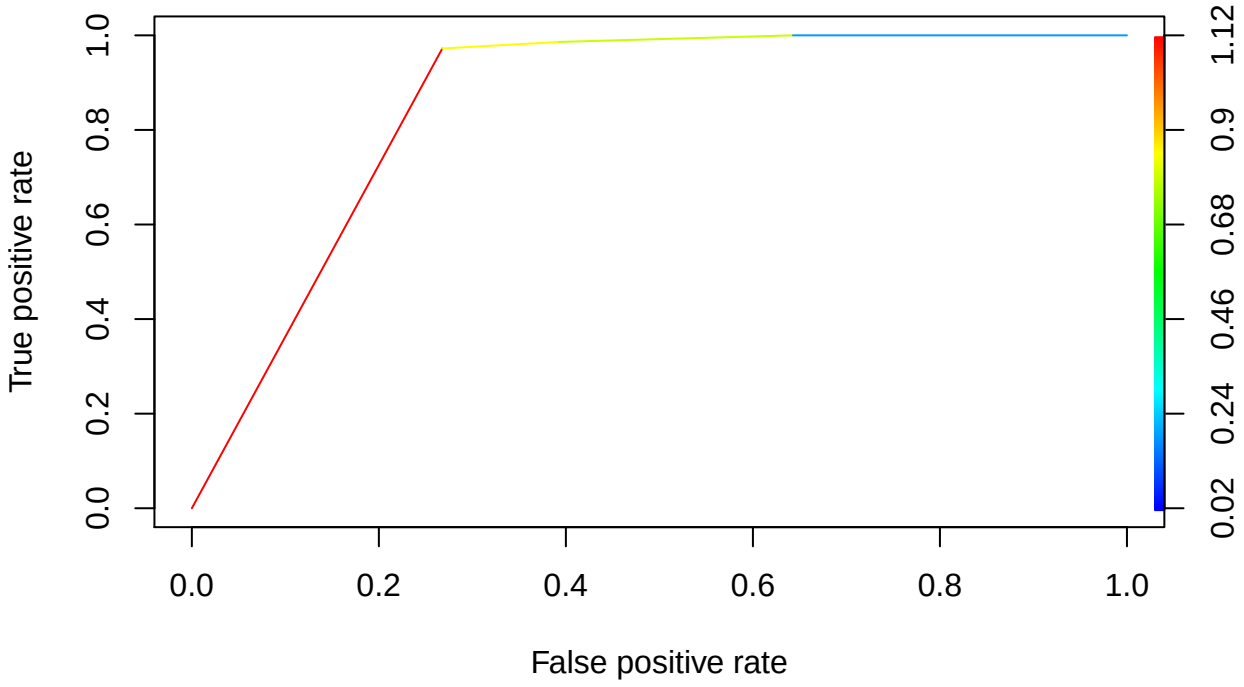
```
## [1] 0.843006
```

```r
#Árboles
c50_model <- C5.0(train_z, train_labels)
confusionMatrix(test_labels, prc50, positive = "Demented")
```

```
## Warning in confusionMatrix.default(test_labels, prc50, positive = "Demented"):
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
## Confusion Matrix and Statistics
##
##              Reference
## Prediction    Nondemented Demented
##   Nondemented          71        1
##   Demented             19       37
##
##                Accuracy : 0.8438
##                  95% CI : (0.7691, 0.9019)
##     No Information Rate : 0.7031
##     P-Value [Acc > NIR] : 0.0001753
##
##                   Kappa : 0.6708
##
##  Mcnemar's Test P-Value : 0.0001439
##
##             Sensitivity : 0.9737
##             Specificity : 0.7889
##          Pos Pred Value : 0.6607
##          Neg Pred Value : 0.9861
##              Prevalence : 0.2969
##          Detection Rate : 0.2891
##    Detection Prevalence : 0.4375
##       Balanced Accuracy : 0.8813
##
##        'Positive' Class : Demented
##
```

```r
pred <- predict(c50_model, test_z, type = "prob")
predicciones=prediction(pred[,1], test_labels)

perf=performance(predicciones, "tpr", "fpr")
plot(perf, colorize=TRUE)
```




```
perf.auc <- performance(predicciones, measure = "auc")
AUC=unlist(perf.auc@y.values)
AUC
```

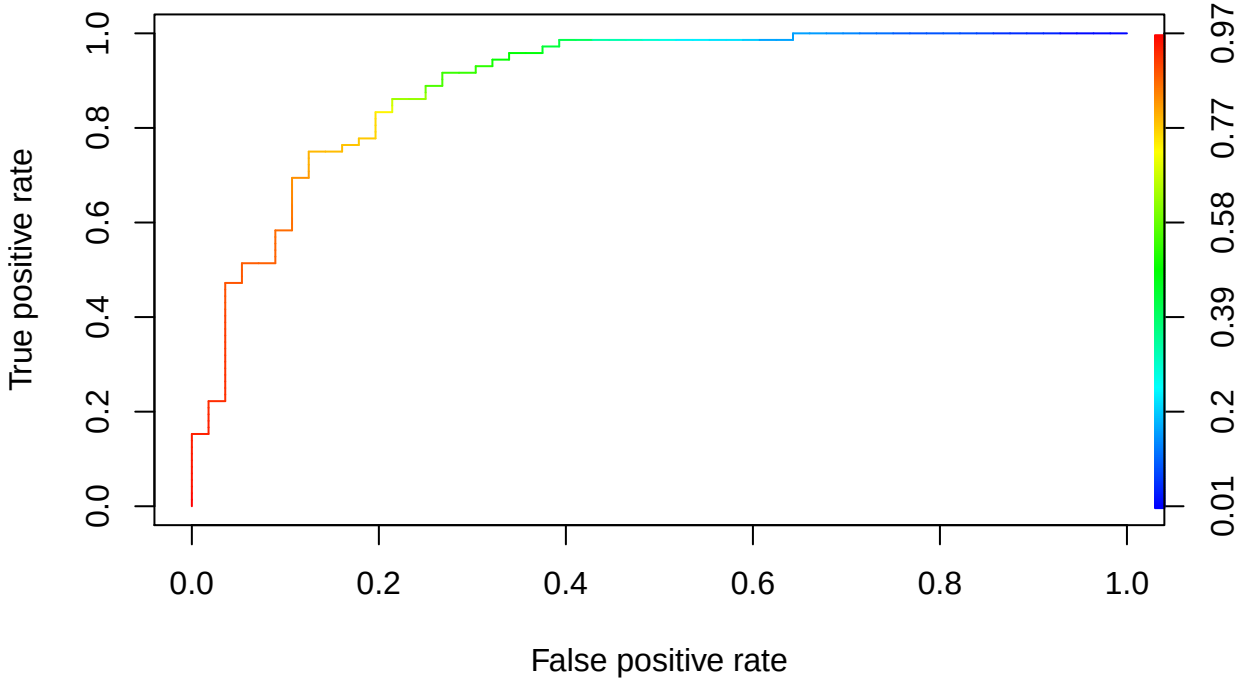
```
## [1] 0.8580109
```

```
#SVM
datalabels_z=cbind(train_labels, train_z)

tlb_z=cbind(test_labels, test_z)
model <- svm(train_labels ~ ., data = datalabels_z, probability=TRUE)

pred <- predict(model, tlb_z, probability=TRUE)

confusionMatrix(test_labels, pred, positive = "Demented")
```

```
## Warning in confusionMatrix.default(test_labels, pred, positive = "Demented"):
## Levels are not in the same order for reference and data. Refactoring data to
## match.

## Confusion Matrix and Statistics
##
##              Reference
## Prediction    Nondemented Demented
```

```
##   Nondemented          67        5
##   Demented             18       38
##
##                Accuracy : 0.8203
##                  95% CI : (0.7427, 0.8826)
##     No Information Rate : 0.6641
##     P-Value [Acc > NIR] : 6.422e-05
##
##                   Kappa : 0.6253
##
##  Mcnemar's Test P-Value : 0.01234
##
##             Sensitivity : 0.8837
##             Specificity : 0.7882
##          Pos Pred Value : 0.6786
##          Neg Pred Value : 0.9306
##              Prevalence : 0.3359
##          Detection Rate : 0.2969
##    Detection Prevalence : 0.4375
##       Balanced Accuracy : 0.8360
##
##        'Positive' Class : Demented
##
```

```
prob_SVM=attr(pred, "probabilities")


predicciones=prediction(prob_SVM[,1], test_labels)

perf=performance(predicciones, "tpr", "fpr")
plot(perf, colorize=TRUE)
```




```
perf.auc <- performance(predicciones, measure = "auc")
AUC=unlist(perf.auc@y.values)
AUC
```

```
## [1] 0.8936012
```

Como vemos los modelos mejoran con excepción del k-NN. De hecho hemos obtenido la mejor precisión (0.85) y valor kappa (0.69) en el modelo bayesiano, y el mejor AUC (0.89) en el modelo SVM, lo cual son resultados muy satisfactorios por los objetivos que nos habíamos propuesto.

Automatic tuning

Por último podemos realizar un entrenamiento de los modelos probando diferentes cambios en los parámetros de los modelos, a ver si obtenemos un resultado mejor con algun cambio de parametro. Aunque en este caso solo utilizamos el conjunto de datos de entrenamiento. Lo haremos con ambas normalizaciones.

```
#Con normalización max/min
set.seed(123)
knn=train(train_labels ~ ., data = datalabels, method = "knn")
knn
```

```
## k-Nearest Neighbors
##
## 214 samples
```

```
##   8 predictor
##   2 classes: 'Nondemented', 'Demented'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 214, 214, 214, 214, 214, 214, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.7406930  0.4357329
##   7  0.7551139  0.4624777
##   9  0.7584501  0.4627087
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 9.
```

```
arbol <- train(train_labels ~ ., data = datalabels, method = "C5.0")
arbol
```

```
## C5.0
##
## 214 samples
##   8 predictor
##   2 classes: 'Nondemented', 'Demented'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 214, 214, 214, 214, 214, 214, ...
## Resampling results across tuning parameters:
##
##   model  winnow  trials  Accuracy   Kappa
##   rules  FALSE    1      0.8025091  0.5749996
##   rules  FALSE   10      0.8287478  0.6257659
##   rules  FALSE   20      0.8314367  0.6320133
##   rules   TRUE    1      0.8109305  0.5882802
##   rules   TRUE   10      0.8201923  0.6054489
##   rules   TRUE   20      0.8248962  0.6169025
##   tree   FALSE    1      0.7951641  0.5588480
##   tree   FALSE   10      0.8292029  0.6269106
##   tree   FALSE   20      0.8217522  0.6096168
##   tree    TRUE    1      0.8029467  0.5750924
##   tree    TRUE   10      0.8180770  0.6018856
##   tree    TRUE   20      0.8191349  0.6023755
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were trials = 20, model = rules and
##  winnow = FALSE.
```

```
bayes <- train(train_labels ~ ., data = datalabels, method = "nb")
bayes
```

```
## Naive Bayes
##
```

```
## 214 samples
##   8 predictor
##   2 classes: 'Nondemented', 'Demented'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 214, 214, 214, 214, 214, 214, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.8223682  0.6231698
##    TRUE      0.8094231  0.5835302
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = FALSE and adjust
##  = 1.
```

```
svmmodel <- train(train_labels ~ ., data = datalabels, method = "svmRadial")
svmmodel
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 214 samples
##   8 predictor
##   2 classes: 'Nondemented', 'Demented'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 214, 214, 214, 214, 214, 214, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7757655  0.4916837
##   0.50  0.7829656  0.5151112
##   1.00  0.7812342  0.5190828
##
## Tuning parameter 'sigma' was held constant at a value of 0.1264807
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.1264807 and C = 0.5.
```

```
#Con normalización z score
set.seed(123)
knn_z=train(train_labels ~ ., data = datalabels_z, method = "knn")
knn_z
```

```
## k-Nearest Neighbors
##
## 214 samples
##   8 predictor
##   2 classes: 'Nondemented', 'Demented'
```

```
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 214, 214, 214, 214, 214, 214, ...
## Resampling results across tuning parameters:
##
##   k  Accuracy   Kappa
##   5  0.7657508  0.4931871
##   7  0.7623757  0.4792319
##   9  0.7652998  0.4826126
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 5.
```

```
arbol_z <- train(train_labels ~ ., data = datalabels_z, method = "C5.0")
arbol_z
```

```
## C5.0
##
## 214 samples
##   8 predictor
##   2 classes: 'Nondemented', 'Demented'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 214, 214, 214, 214, 214, 214, ...
## Resampling results across tuning parameters:
##
##   model  winnow  trials  Accuracy   Kappa
##   rules  FALSE    1      0.8030029  0.5759738
##   rules  FALSE   10      0.8269953  0.6215156
##   rules  FALSE   20      0.8292086  0.6273447
##   rules   TRUE    1      0.8109305  0.5882802
##   rules   TRUE   10      0.8207727  0.6062828
##   rules   TRUE   20      0.8254335  0.6170386
##   tree   FALSE    1      0.7962059  0.5611270
##   tree   FALSE   10      0.8277284  0.6241817
##   tree   FALSE   20      0.8233814  0.6127662
##   tree    TRUE    1      0.8029467  0.5750924
##   tree    TRUE   10      0.8165846  0.5988199
##   tree    TRUE   20      0.8169620  0.5969468
##
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were trials = 20, model = rules and
##  winnow = FALSE.
```

```
bayes_z <- train(train_labels ~ ., data = datalabels_z, method = "nb")
```

```
## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 70

## Warning in FUN(X[[i]], ...): Numerical 0 probability for all classes with
## observation 72
```

```
bayes_z
```

```
## Naive Bayes
##
## 214 samples
##   8 predictor
##   2 classes: 'Nondemented', 'Demented'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 214, 214, 214, 214, 214, 214, ...
## Resampling results across tuning parameters:
##
##   usekernel  Accuracy   Kappa
##   FALSE      0.8223682  0.6231698
##    TRUE      0.8099360  0.5845402
##
## Tuning parameter 'fL' was held constant at a value of 0
## Tuning
##  parameter 'adjust' was held constant at a value of 1
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 0, usekernel = FALSE and adjust
##  = 1.
```

```
svmmodel_z <- train(train_labels ~ ., data = datalabels_z, method = "svmRadial")
svmmodel_z
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 214 samples
##   8 predictor
##   2 classes: 'Nondemented', 'Demented'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 214, 214, 214, 214, 214, 214, ...
## Resampling results across tuning parameters:
##
##   C     Accuracy   Kappa
##   0.25  0.7757655  0.4916837
##   0.50  0.7829656  0.5151112
##   1.00  0.7812342  0.5190828
##
## Tuning parameter 'sigma' was held constant at a value of 0.1264807
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were sigma = 0.1264807 and C = 0.5.
```

En este caso los mejores resultados se obtienen en el modelo de bayes con el parámetro kernel=FALSE, y en los árboles de decisión con los parámetros model=tree, winnow=FALSE y trial=10.

5. Comentarios de vuestro director particular si lo consideráis necesario