Minería de datos: PRA2 - Modelado de un juego de datos

Autor: Reynel López Lantigua

Diciembre 2020

Contents

Introducción

Presentación

Esta práctica cubre de forma transversal la asignatura.

Las Prácticas 1 y 2 de la asignatura se plantean de una forma conjunta de modo que la Práctica 2 será continuación de la 1.

El objetivo global de las dos prácticas consiste en seleccionar uno o varios juegos de datos, realizar las tareas de preparación y análisis exploratorio con el objetivo de disponer de datos listos para aplicar algoritmos de clustering, asociación y clasificación.

Competencias

Las competencias que se trabajan en esta prueba son:

- Uso y aplicación de las TIC en el ámbito académico y profesional.
- Capacidad para innovar y generar nuevas ideas.
- Capacidad para evaluar soluciones tecnológicas y elaborar propuestas de proyectos teniendo en cuenta los recursos, las alternativas disponibles y las condiciones de mercado.
- Conocer las tecnologías de comunicaciones actuales y emergentes así como saberlas aplicar convenientemente para diseñar y desarrollar soluciones basadas en sistemas y tecnologías de la información.
- Aplicación de las técnicas específicas de ingeniería del software en las diferentes etapas del ciclo de vida de un proyecto.
- Capacidad para aplicar las técnicas específicas de tratamiento, almacenamiento y administración de datos.
- Capacidad para proponer y evaluar diferentes alternativas tecnológicas para resolver un problema concreto.

Objetivos

La correcta asimilación de todos los aspectos trabajados durante el semestre.
En esta práctica abordamos un caso real de minería de datos donde tenemos que poner en juego todos los conceptos trabajados. Hay que trabajar todo el ciclo de vida del proyecto. Desde el objetivo del proyecto hasta la implementación del conocimiento encontrado pasando por la preparación, limpieza de los datos, conocimiento de los datos, generación del modelo, interpretación y evaluación.

Descripción de la PEC a realizar

Recursos Básicos

Material docente proporcionado por la UOC.

Criterios de valoración

Ejercicios prácticos

Para todas las PEC es necesario documentar en cada apartado del ejercicio práctico que se ha hecho y como se ha hecho.

Formato y fecha de entrega

El formato de entrega es: usernameestudiante-PECn.html/doc/docx/odt/pdf/rmd
Fecha de entrega: 15/01/2020
Se debe entregar la PEC en el buzón de entregas del aula

Nota: Propiedad intelectual

> A menudo es inevitable, al producir una obra multimedia, hacer uso de recursos creados por terceras personas. Es por lo tanto comprensible hacerlo en el marco de una práctica de los estudios de Informática, Multimedia y Telecomunicación de la UOC, siempre y cuando esto se documente claramente y no suponga plagio en la práctica.

Por lo tanto, al presentar una práctica que haga uso de recursos ajenos, se debe presentar junto con ella un documento en que se detallen todos ellos, especificando el nombre de cada recurso, su autor, el lugar donde se obtuvo y su estatus legal: si la obra esta protegida por el copyright o se acoge a alguna otra licencia de uso (Creative Commons, licencia GNU, GPL ...). El estudiante deberá asegurarse de que la licencia no impide específicamente su uso en el marco de la práctica. En caso de no encontrar la información correspondiente tendrá que asumir que la obra esta protegida por copyright.

Deberéis, además, adjuntar los ficheros originales cuando las obras utilizadas sean digitales, y su código fuente si corresponde.

Enunciado

Como continuación del estudio iniciado en la práctica 1, procedemos en esta práctica 2 a aplicar modelos analíticos sobe el juego de datos seleccionado y preparado en la práctica anterior.

De este modo se pide al estudiante que complete los siguientes pasos:

1. Tratar la práctica como un proyecto real de minería de datos, con todos los puntos de su **ciclo de vida**.
2. Aplicar un modelo de generación de reglas a partir de **árboles de decisión**.
3. Aplicar un modelo **no supervisado** y basado en el concepto de **distancia**, sobre el juego de datos.
4. Aplica de nuevo el modelo anterior, pero usando una **métrica distinta** y compara los resultados.
5. Aplicar un **modelo supervisado** sobre el juego de datos **sin** haber aplicado previamente **PCA/SVD**.
6. Aplicar un **modelo supervisado** sobre el juego de datos habiendo aplicado previamente **PCA/SVD**.
7. ¿Ha habido mejora en capacidad predictiva, tras aplicar PCA/SVD? ¿A qué crees que es debido?.

Rúbrica

- 10%. Trata todos los puntos de la práctica como un proyecto real de minería de datos, siguiendo todos los puntos del ciclo de vida, desde la descripción funcional hasta la integración de los resultados, comentando los puntos en los que sería necesario volver atrás cuando sea necesario.
- 15%. Se generan reglas y se comentan e interpretan las más significativas. Adicionalmente se genera matriz de confusión para medir la capacidad predictiva del algoritmo.
- 15%. Se genera modelo no supervisado, se muestran y comentan medidas de calidad del modelo generado y se comentan las conclusiones.
- 15%. Se genera modelo no supervisado con métrica de distancia distinta al anterior. Se muestran y comentan medidas de calidad del modelo generado y se comentan las conclusiones. Adicionalmente se comparan los dos modelos no supervisados con métricas de distancia distinta.
- 15%. Se genera un modelo supervisado sin PCA/SVD previo, se muestran y comentan medidas de calidad del modelo generado y se comenta extensamente el conocimiento extraído del modelo.

- 15%. Se genera un modelo supervisado con PCA/SVD previo, se muestran y comentan medidas de calidad del modelo generado y se comenta extensamente el conocimiento extraído del modelo.

- 15%. Se compara la capacidad predictiva de los dos modelos supervisados y se comenta la diferencia de rendimiento en base al efecto PCA/SVD.

Recursos de programación

- Incluimos en este apartado una lista de recursos de programación para minería de datos donde podréis encontrar ejemplos, ideas e inspiración:
  - Material adicional del libro: Minería de datos Modelos y Algoritmos
  - Espacio de recursos UOC para ciencia de datos
  - Buscador de código R

  - Colección de cheatsheets en R

Solución

Carga de los datos

Para comenzar con esta práctica es necesario realizar la carga de los datos. Los dos archivos con los que se trabaja son los resultantes de la ejecución de las tareas de pre-procesamiento y limpieza realizadas en la práctica 1, con lo cual estos archivos están almacenados previamente en el directorio data. De manera análoga a los procedimientos realizados durante todo el curso, se realiza un pequeño análisis para observar los datos. En este punto no se realizará un análisis muy detallado porque estos datos ya son conocidos de la práctica pasada.

```
# Loading the data file.
diabetes <- read.csv("../data/diabetes.csv", header = TRUE, row.names = 'X')
str(diabetes)
```

```
## 'data.frame':    650 obs. of  9 variables:
##  $ Pregnancies             : num  0.678 -0.863 1.294 -0.863 0.369 ...
##  $ Glucose                 : num  0.992 -1.167 2.191 -1.03 -0.105 ...
##  $ BloodPressure           : num  -0.00802 -0.53837 -0.71516 -0.53837 0.16876 ...
##  $ SkinThickness           : num  0.7516 0.0358 0.0358 -0.6801 0.0358 ...
##  $ Insulin                 : num  -0.134 -0.134 -0.134 -0.558 -0.134 ...
##  $ BMI                     : num  0.253 -0.841 -1.357 -0.607 -0.997 ...
##  $ DiabetesPedigreeFunction: num  0.828 -0.305 1.012 -1.059 -0.92 ...
##  $ Age                     : num  1.4438 -0.1793 -0.0938 -1.0335 -0.2647 ...
##  $ Outcome                 : int  1 0 1 0 0 1 0 1 1 1 ...
```

```
head(diabetes)
```

```
##   Pregnancies    Glucose BloodPressure SkinThickness    Insulin        BMI
## 1   0.6775211  0.9916292  -0.008023249    0.75163444 -0.1337277  0.2533708
## 2  -0.8633772 -1.1671808  -0.538373623    0.03579212 -0.1337277 -0.8411296
## 3   1.2938805  2.1909681  -0.715157080    0.03579212 -0.1337277 -1.3571084
```

```
## 4  -0.8633772 -1.0301135  -0.538373623   -0.68005020 -0.5578867 -0.6065938
## 6   0.3693415 -0.1049092   0.168760209    0.03579212 -0.1337277 -0.9974868
## 7  -0.2470179 -1.4070486  -1.952641285    0.39371328 -0.6762566 -0.1531579
##   DiabetesPedigreeFunction         Age Outcome
## 1                0.8278066  1.44377913       1
## 2               -0.3045754 -0.17925676       0
## 3                1.0124341 -0.09383382       1
## 4               -1.0594968 -1.03348617       0
## 6               -0.9200004 -0.26467970       0
## 7               -0.7271673 -0.60637146       1
```

```
diabetes$Outcome <- as.factor(diabetes$Outcome)
diabetes_svd <- read.csv("../data/diabetes.svd.csv", header=TRUE, row.names = 'X')
head(diabetes_svd)
```

```
##           V1        V2         V3         V4          V5       V6         V7
## 1 -1.1445400 2.0662193 1.60275413 -0.1746373 -0.13555959 1.857054  1.0783147
## 2 -0.7082929 1.8127617 1.29446236  0.8305354 -0.03456945 2.173382  0.8579285
## 3 -0.8690656 1.6092878 0.56640658  1.1865084 -0.15423840 2.328864 -0.1567305
## 4 -0.7946770 0.9886867 1.98675596  1.5391466 -0.13838597 2.611378 -0.2543914
## 5  0.4309283 1.4343615 0.52713560  0.4215423 -0.15880006 2.030806 -0.6526686
## 6  0.1381721 1.8148796 0.04820556  0.1161313 -0.13894945 1.708432  0.2446478
##           V8
## 1 -0.9068783
## 2 -0.5265352
## 3 -1.1016780
## 4 -0.6334398
## 5 -0.6992985
## 6 -0.7324700
```

```
diabetes_svd$Outcome <- as.factor(diabetes$Outcome)
head(diabetes_svd)
```

```
##           V1        V2         V3         V4          V5       V6         V7
## 1 -1.1445400 2.0662193 1.60275413 -0.1746373 -0.13555959 1.857054  1.0783147
## 2 -0.7082929 1.8127617 1.29446236  0.8305354 -0.03456945 2.173382  0.8579285
## 3 -0.8690656 1.6092878 0.56640658  1.1865084 -0.15423840 2.328864 -0.1567305
## 4 -0.7946770 0.9886867 1.98675596  1.5391466 -0.13838597 2.611378 -0.2543914
## 5  0.4309283 1.4343615 0.52713560  0.4215423 -0.15880006 2.030806 -0.6526686
## 6  0.1381721 1.8148796 0.04820556  0.1161313 -0.13894945 1.708432  0.2446478
##           V8 Outcome
## 1 -0.9068783       1
## 2 -0.5265352       0
## 3 -1.1016780       1
## 4 -0.6334398       0
## 5 -0.6992985       0
## 6 -0.7324700       1
```

```
str(diabetes)
```

```
## 'data.frame':    650 obs. of  9 variables:
##  $ Pregnancies             : num  0.678 -0.863 1.294 -0.863 0.369 ...
##  $ Glucose                 : num  0.992 -1.167 2.191 -1.03 -0.105 ...
##  $ BloodPressure           : num  -0.00802 -0.53837 -0.71516 -0.53837 0.16876 ...
##  $ SkinThickness           : num  0.7516 0.0358 0.0358 -0.6801 0.0358 ...
##  $ Insulin                 : num  -0.134 -0.134 -0.134 -0.558 -0.134 ...
```

```
##  $ BMI                      : num  0.253 -0.841 -1.357 -0.607 -0.997 ...
##  $ DiabetesPedigreeFunction: num  0.828 -0.305 1.012 -1.059 -0.92 ...
##  $ Age                      : num  1.4438 -0.1793 -0.0938 -1.0335 -0.2647 ...
##  $ Outcome                  : Factor w/ 2 levels "0","1": 2 1 2 1 1 2 1 2 2 2 ...
```

Preparación de los datos

En este apartado se realizan las tareas necesarias para preparar tanto los datos originales como los obtenidos después de aplicar SVD, estas tareas ofrecen como resultados particiones en los datos para que luego sean consumidas por los diferentes algoritmos. Se intentará que estas particiones presenten la mejor calidad posibles utilizando las particiones estratificadas y posteriormente comprobando la calidad de las mismas.

Partición del Dataset original

En este apartado se realizan algunas tareas de transformación a los datos con el objetivo de prepararlos para alimentar los diferentes algoritmos que se probarán durante la práctica. Principalmente los algoritmos de clasificación que necesitan que los datos estén divididos en *training* y *testing* para su correcta evaluación. Otros de los objetivos de esta partición es permitir probar los modelos implementados con datos desconocidos para el algoritmo, de esta manera se evitan entre otras cosas el *overfitting* y se obtiene un valor mucho más realista de la capacidad clasificadora del modelo.

Para realizar las particiones se utiliza el mismo método que en la práctica 1, la función createDataPartition() que crea particiones estratificadas de los datos teniendo como valor de referencia la clase objetivo.

```
library(caret)


# Make the data replicable
set.seed(2021)

# Create the inedx list
diab_indexes <- createDataPartition(diabetes$Outcome, p = .7, list = FALSE)


# Get the train segment
diab_train <- diabetes[diab_indexes,]

# Get the test segment
diab_test <- diabetes[-diab_indexes, ]
```

También se separan las variables predictoras de las objetivos, esto mayormente se utiliza para obtener las métricas de rendimiento de los modelos, al comparar las salidas del modelo con las clases de la variable objetivo.

```
# Split the target and predictors variables
diab_train.x <- diab_train[1:8]
diab_train.y <- diab_train[ , 9 ]

diab_test.x <- diab_test[1:8]
diab_test.y <- diab_test[, 9]

diab_predictors <- diabetes[ ,-9]
diab_target <- diabetes$Outcome
```

Análisis estadístico de los datos divididos.

Para asegurar que los datos han sido divididos adecuadamente se comprueban las distribuciones de la variable *target*. El objetivo de esta comprobación es demostrar que en ambos conjuntos de datos las distribuciones de ambas clases son similares, y no perjudicarán la creación del modelo sesgando los resultados por la presencia superior de una clase sobre la otra.

```
# Get the distribution of each variable in each splited set

# Training set
train.prop <- table(diab_train.y)
train.prop
```

```
## diab_train.y
##   0   1
## 313 143
```

```
print(sprintf("La proporción de clases en el conjunto de entrenamiento es: %.4f", train.prop[2]/train.p
```

```
## [1] "La proporción de clases en el conjunto de entrenamiento es: 0.4569"
```

```
# Test set
test.prop <- table(diab_test.y)
test.prop
```

```
## diab_test.y
##   0   1
## 134  60
```

```
print(sprintf("La proporción de clases en el conjunto de prueba es: %.4f", test.prop[2]/test.prop[1]))
```

```
## [1] "La proporción de clases en el conjunto de prueba es: 0.4478"
```

Como demuestra el estudio anterior ambos conjuntos presentan una distribución parecida de las clases de la variable objetivo, lo que demuestra que los datos han sido divididos de manera correcta.

Dataset modificado

En el caso de los datos obtenidos después de aplicar el algoritmo SVD, se seguirá el mismo procedimiento que para los datos originales. Se realizarán las correspondientes particiones estratificadas para luego separar las variables predictoras de la objetivo.

```
library(caret)


# Make the data replicable
set.seed(2021)

# Create the inedx list
svd_indexes <- createDataPartition(diabetes_svd$Outcome, p = .7, list = FALSE)


# Get the train segment
svd_train <- diabetes_svd[svd_indexes,]

# Get the test segment
svd_test <- diabetes_svd[-svd_indexes, ]

# Split the target and predictors variables
```

```r
svd_train.x <- svd_train[1:8]
svd_train.y <- svd_train[ , 9 ]

svd_test.x <- svd_test[1:8]
svd_test.y <- svd_test[, 9]
```

Análisis estadístico de los datos divididos.

Para los datos modificados también se comprueba los resutlados del proceso de partición.

```r
# Get the distribution of each variable in each splited set

# Training set
train.prop <- table(svd_train.y)
train.prop
```

```
## svd_train.y
##   0   1
## 313 143
```

```r
print(sprintf("La proporción de clases en el conjunto de entrenamiento es: %.4f", train.prop[2]/train.pr
```

```
## [1] "La proporción de clases en el conjunto de entrenamiento es: 0.4569"
```

```r
# Test set
test.prop <- table(svd_test.y)
test.prop
```

```
## svd_test.y
##   0   1
## 134  60
```

```r
print(sprintf("La proporción de clases en el conjunto de prueba es: %.4f", test.prop[2]/test.prop[1]))
```

```
## [1] "La proporción de clases en el conjunto de prueba es: 0.4478"
```

Como demuestra el estudio anterior ambos conjuntos presentan una distribución parecida de las clases de la variable objetivo, lo que demuestra que los datos han sido divididos de manera correcta.

Obtención de reglas de asociación mediante Árboles de decisión

Como su nombre lo indica en este apartado se aplicarán las técnicas necesarias para obtener un conjuntos de reglas de asociación a partir de un algoritmo basado en los árboles de decisión. Se utilizará el conocido algoritmo C5.0 con el parámetro "*rules* = TRUE para que devuelva una lista con dichas reglas.

Creación del modelo y obtención de las Reglas de asociación

Para crear el modelo se utilizan los datos originales divididos adecuadamente para este algoritmo, una vez creado el modelo se ofrece un resumen de los resultados que aporta el mismo.

En este resumen se ofrecen datos muy interesantes del modelo. Ofrece un resumen de la cantidad de observaciones (*cases*) que utilizó para la construcción de árbol además de la cantidad de variables (*attributes*). Indica cuál fue la clase usada como *target* y ofrece un listado con las **reglas de asociación** obtenida por el modelo, para cada regla se muestra el valor del *lift* y las diferentes condiciones que las componen. Este resumen también contiene un análisis del rendimiento del modelo comparando los casos de entrenamiento con los que se construyó el modelo. Al final del resumen se muestra el porcentaje de los atributos utilizados en el proceso de construcción del árbol.

```r
# Built the model
model <- C50::C5.0(diab_train.x, diab_train.y,rules=TRUE )
summary(model)
```

```
##
## Call:
## C5.0.default(x = diab_train.x, y = diab_train.y, rules = TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]      Wed Jan 13 16:43:22 2021
## -------------------------------
##
## Class specified by attribute `outcome'
##
## Read 456 cases (9 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (84/1, lift 1.4)
##  Glucose <= 0.8202951
##  BMI <= -0.8880368
##  ->  class 0  [0.977]
##
## Rule 2: (50/1, lift 1.4)
##  Glucose <= 0.8202951
##  BMI <= -0.2782437
##  DiabetesPedigreeFunction <= -0.7230644
##  ->  class 0  [0.962]
##
## Rule 3: (20, lift 1.4)
##  Glucose > -0.6874452
##  Glucose <= 0.8202951
##  BMI > 0.3784566
##  DiabetesPedigreeFunction <= -0.7230644
##  ->  class 0  [0.955]
##
## Rule 4: (126/6, lift 1.4)
##  Glucose <= 0.8202951
##  DiabetesPedigreeFunction <= 0.3149524
##  Age <= -0.4355256
##  ->  class 0  [0.945]
##
## Rule 5: (116/7, lift 1.4)
##  Glucose <= -0.6874452
##  ->  class 0  [0.932]
##
## Rule 6: (167/13, lift 1.3)
##  Glucose <= 0.8202951
##  Insulin <= 0.8428243
##  Age <= -0.4355256
##  ->  class 0  [0.917]
##
## Rule 7: (80/6, lift 1.3)
##  Glucose <= 0.8202951
```

```
##  BloodPressure > -0.3615902
##  Age <= -0.4355256
##  ->  class 0  [0.915]
##
## Rule 8: (5, lift 2.7)
##  Glucose > -0.6874452
##  Glucose <= -0.139176
##  BloodPressure <= -0.3615902
##  BMI > -0.8880368
##  DiabetesPedigreeFunction > 0.3149524
##  Age <= -0.4355256
##  ->  class 1  [0.857]
##
## Rule 9: (90/23, lift 2.4)
##  Glucose > 0.8202951
##  ->  class 1  [0.739]
##
## Rule 10: (56/18, lift 2.1)
##  Glucose > -0.6874452
##  BMI > -0.2782437
##  BMI <= 0.3784566
##  Age > -0.4355256
##  ->  class 1  [0.672]
##
## Rule 11: (130/44, lift 2.1)
##  Glucose > -0.6874452
##  BMI > -0.8880368
##  DiabetesPedigreeFunction > -0.7230644
##  Age > -0.4355256
##  ->  class 1  [0.659]
##
## Rule 12: (95/38, lift 1.9)
##  Glucose > -0.6874452
##  BMI > -0.8880368
##  DiabetesPedigreeFunction > 0.3149524
##  ->  class 1  [0.598]
##
## Default class: 0
##
##
## Evaluation on training data (456 cases):
##
##          Rules
##    ----------------
##      No      Errors
##
##      12   76(16.7%)   <<
##
##
##     (a)   (b)    <-classified as
##    ----  ----
##     252    61    (a): class 0
##      15   128    (b): class 1
##
```

```
##
##  Attribute usage:
##
##  100.00% Glucose
##   72.15% Age
##   70.18% DiabetesPedigreeFunction
##   67.32% BMI
##   36.62% Insulin
##   18.64% BloodPressure
##
##
## Time: 0.0 secs
```

Las reglas más importantes generadas por el modelo son:

1. Si:

- Glucose <= 0.8202951
  - BMI <= -0.8880368
  - **class** -> 0 con una validez del 97%

2. Si:
   - Glucose <= 0.8202951
   - BMI <= -0.2782437
   - DiabetesPedigreeFunction <= -0.7230644
   - **class** -> 0 con una validez del 96.2%
3. Si:
   - Glucose > -0.6874452
   - Glucose <= 0.8202951
   - BMI > 0.3784566
   - DiabetesPedigreeFunction <= -0.7230644
   - **class** -> 0 con una validez del 95.5%
4. Si:
   - Glucose <= 0.8202951
   - DiabetesPedigreeFunction <= 0.3149524
   - Age <= -0.4355256
   - **class** -> 0 con una validez del 94.5

Como es de suponer la mayoría de las reglas se basan en los niveles de glucosa, parámetros claramente decisivo a la hora de evaluar un enfermedad como la diabetes. En las demás reglas seleccionadas para ser mostradas se aprecia como hay otros factores importantes como el índice de masa corporal o la edad. Estos resultados corroboran en gran medida los resultados obtenidos en el análisis exploratorio de los datos realizados en la práctica 1.

Validación y calidad del modelo

Una vez conocidos los principales parámetros y reglas que componen el árbol se procede a evaluar su rendimiento en los datos de prueba, ests datos hasta ahora no eran conocidos por el modelo, lo que ofrecerá una medida mucho más realista de la capacidad clasificadora del mismo. utilizando la función CrossTable() del paquete *gmodels* se crea una matriz de confusión y se ofrece una medida del rendimiento del modelo (*accuracy*).

```
library(gmodels)

# Test the model with unknown data
predicted_model <- predict( model, diab_test.x, type="class" )
```

```
CrossTable(diab_test.y, predicted_model,prop.chisq  = FALSE, prop.c = FALSE, prop.r =FALSE,dnn = c('Rea
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  194
##
##
##              | Prediction
##      Reality |         0 |         1 | Row Total |
## -------------|-----------|-----------|-----------|
##            0 |       106 |        28 |       134 |
##              |     0.546 |     0.144 |           |
## -------------|-----------|-----------|-----------|
##            1 |        17 |        43 |        60 |
##              |     0.088 |     0.222 |           |
## -------------|-----------|-----------|-----------|
## Column Total |       123 |        71 |       194 |
## -------------|-----------|-----------|-----------|
##
##
```

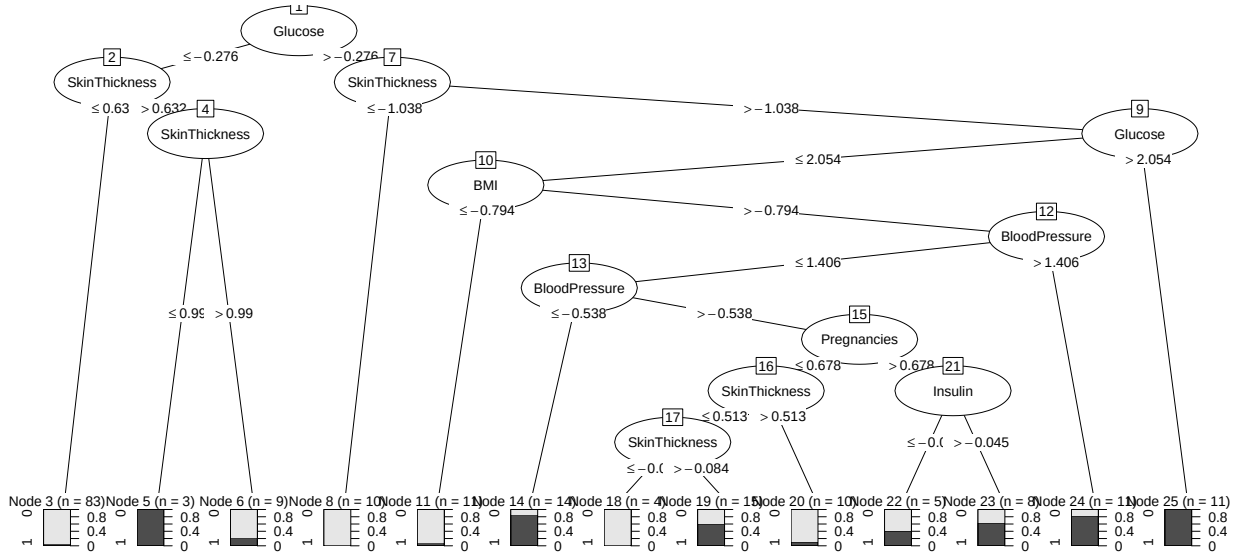
```
c50_ave <- 100*sum(predicted_model == diab_test.y) / length(predicted_model)
print(sprintf("La precisión del árbol es: %.4f %%",c50_ave))
```

```
## [1] "La precisión del árbol es: 76.8041 %"
```

Visualización del modelo

para finalizar con este apartado se propone un visualización del árbol resultante de aplicar el algoritmo C5.0 a los datos de *testing*. Gracias a este gráfico se pueden corroborar las diferentes reglas y colocarlas dentro del árbol, tarea que facilita el entendimiento de este tipo de modelos.

```
# Built the model
model <- C50::C5.0(diab_test.x, diab_test.y)

plot(model)
```



Modelo "No supervisado" basado en distancias (*euclidean*)

En este apartado se evaluará el conjunto de datos originales sobre el algoritmo de agrupamiento *K-medoids*. Este algoritmo basado en distancias permite realizar particiones de los datos en k *clusters*, de manera similar a como lo hace el algoritmo *k-means*, solo que en *k-medoids* el centro está representado por una observación presente el *cluster* (medoid) y en el anterior el centro está representado por un punto correspondiente a al valor del promedio de todas las observaciones del *cluster* sin especificar ninguna.

El hecho de que el algoritmo use los *medoids* en lugar de los centroides del *k_means* hace que este algoritmo sea más robusto, viéndose menos afectado por el ruido o por la presencia de *outliers*. El algoritmo más utilizado para aplicar este método de *k-medoids* es le conocido como PAM *Partitioning Around Medoids*.

Para este algoritmo hay varias suposiciones que pueden ayuar con su rendimiento a al hora de realizar las particiones, por ejemplo el tema de la elección del método de calcular las distancias, si se sospecha la presencia de *outliers* se recomienda usar la distancia de *manhattan* en caso contrario se recomienda la *euclidean*. En este primer ejemplo se evalúa el modelo utilizando la distancia euclidea, ya que en la práctica anterior se realizó un procedimiento para el tratamiento de valores extremos.

Medidas de calidad del modelo

Las medidas de calidad de este modelo coinciden cun las de cualquier otro algoritmo de agrupamiento, son aquellas que persiguen que los grupos estén muy cohesionados entre si y a la vez muy distintos de los demás grupos. Este hecho sugiere que todos los miembros de un mismo *clusters* sean muy similares ente si y a la vez diferentes a los miembros de otros *clusters*. Estas medidas también pueden utilizarse para elegir el número de *clústers* cuando este valor no es conocido a priori. En este apartado se comentarán estas medidas y se realizarán algunas pruebas para simular su utilización.

- Similitud intragrupo: Esta medida establece cuan semejantes son los objetos que el proceso ha incluido en el mismo grupo, una medida típica es la varianza de estos atributos.
- Similitud intergrupos: Esta medida determina la distancia entre los centros de los diferentes grupos. Una vez aplicados los procesos de agregación, proporciona mejor resultado el que presente una distancia mayor entre los centros de los *clsuters*.
- Variantes: Esta medida consiste en utilizar la medida de distancia que se ha empleado para construir los grupos y comparar la distancia media dentro de los *clusters* con respecto a la distancia media entre grupos.

La Silueta proporciona información de cuan similar es un objeto a los objetos de su *cluster* y de cuan diferente es a los de *clusters* vecinos, al final de la iteración, el valor más alto de la silueta es el que propone un número de k óptimo.

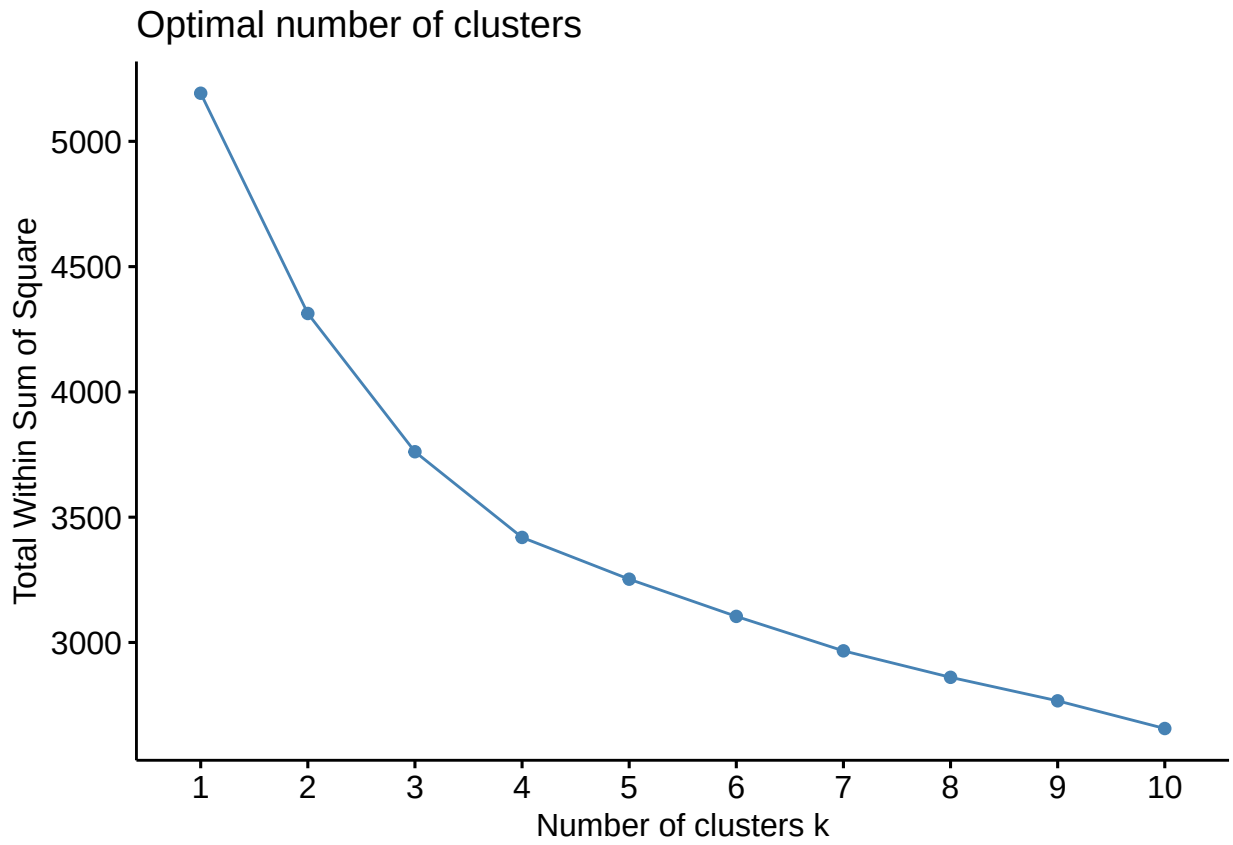
Utilización de las medidas de calidad para determinar k

En ocasiones el número de agrupaciones no es conocido, es entonces cuando las medidas de calidad entran en juego para apoyar la decisión del valor de k. Existen varias pruebas para determinar este valor basado en las medidas de calidad, entre las más comunes se encuentran la prueba del codo y la prueba de la silueta.

Por ejemplo, la prueba del codo (*elbow method*) utiliza la medida de la similitud intragrupo, evaluando la reducción de la varianza de los elementos de un mismo *cluster* para un rango de k. El objetivo de esta prueba es identificar el valor el número óptimo de *clusters* basándose en este parámetro. Gracias a las librerías "cluster" y "factoextra" se puede implementar esta prueba de manera muy sencilla, en la gráfica el valor de k se identifica en el punto que simula la forma de un codo humano.

Remarcar que se establecen algunos parámetros a la hora de realizar la prueba, como por ejemplo el método del cálculo de las distancias o el número máximo de *cluster* que se desea evaluar.

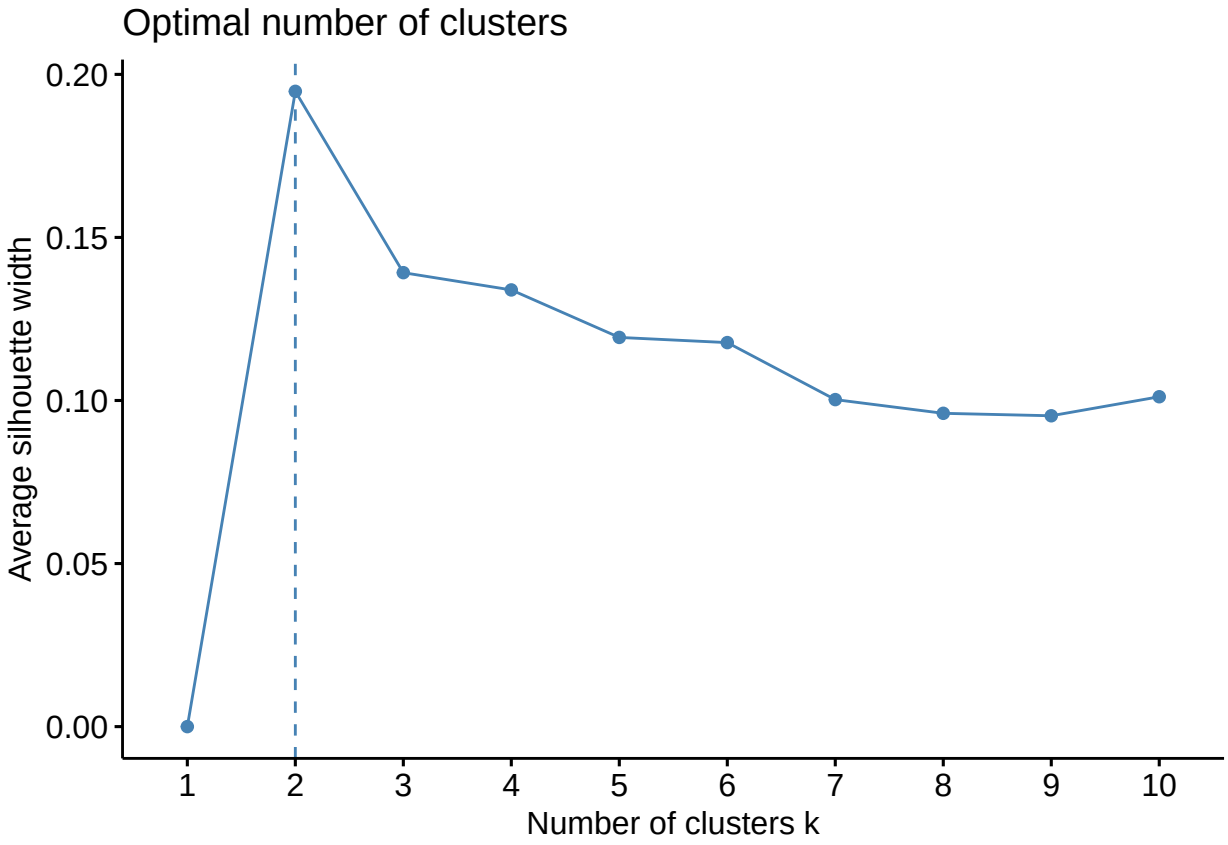
```
library(cluster)
library(factoextra)

fviz_nbclust(x = diab_predictors, FUNcluster = pam, method = "wss", k.max = 10,
             diss = dist(diab_predictors, method = "euclidean"))
```



Como se puede apreciar en los resultados el test arroja resultados que contrastan con la realidad de los datos, el punto de la gráfica similar a un codo está para k = 3 (aproximadamente) y es conocido que las clases presentes en los datos son dos.

Se puede realizar entonces la prueba de la silueta y comprobar sus resultados, la silueta proporciona información de cuan similar es un objeto a los objetos de su *cluster* y de cuan diferente es a los de *clusters* vecinos, al final de la iteración, el valor más alto de la silueta es el que propone un número de k óptimo.

```
library(cluster)
library(factoextra)

fviz_nbclust(x = diab_predictors, FUNcluster = pam, method = "silhouette", k.max = 10,
             diss = dist(diab_predictors, method = "euclidean"))
```

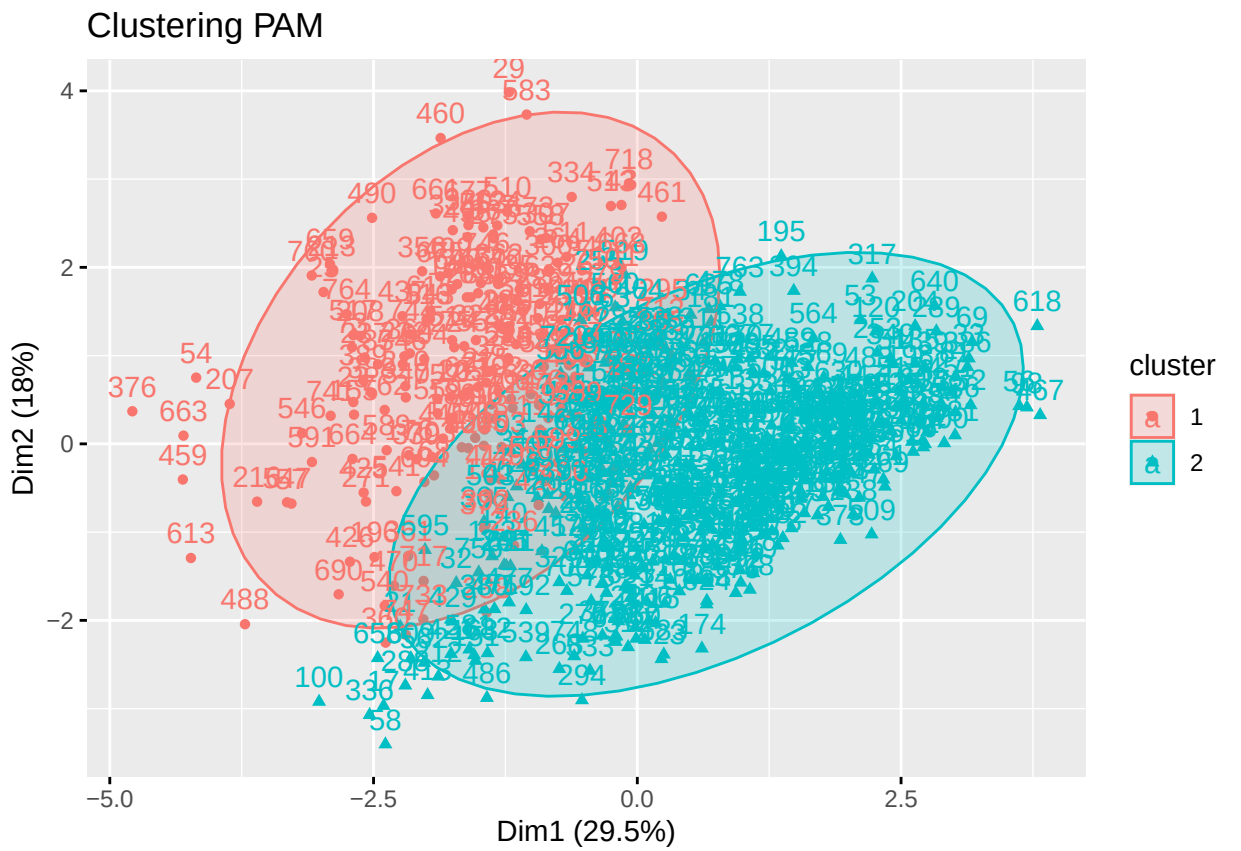


En este caso la prueba sí arroja resultados coherentes con las clases que tenemos en los datos, el valor de la silueta para k = 2, es el que muestra mejores resultados, indicando que es la configuración que mejor relación inter/intra grupos presenta.

Creación del modelo para k =2

En este punto se puede proceder con la creación del modelo, para ello se utiliza la función pam(), y el valor de k = 2, destacando la utilización de la distancia euclídea. Para facilitar el análisis de los resultados se ofrece una gráfica en la que se muestran los dos *clusters* obtenidos.

```
# Create the model
k_medoids <- pam(diab_predictors,k=2, metric = "euclidean")

# Plot the clusters
fviz_cluster(object = k_medoids, data = diab_predictors, ellipse.type = "norm") +
  labs(title = "Clustering PAM")
```



Como se puede apreciar hay algún solapamiento entre ambos grupos, esto indica que puede haber errores en el proceso de agrupamiento y que algunos valores pueden estar en un grupo equivocado.

Evaluación de la clasificación

Es ceirto que en los algoritmos no supervisados no se conoce la clase a la que pertenece cada observación, pero como en este caso se han modificado los datos con el objetivo de poder aplicar estos métodos, se puede obtener una medida cuantitativa de la calidad del proceso de agrupamiento. Para ello se crea una matriz de confusión entre los valores de los *clusters* y los valores de la variable objetivo conocidos.

```
# Get the accuray of the model
cm <- table(k_medoids$clustering, diab_target)
cm
```

```
##    diab_target
##       0   1
##   1  81 109
##   2 366  94
```

```
k_medoids_euc_ave <- (cm[1,2] + cm[2,1]) / (cm[1,1] + cm[2,2] +cm[1,2] + cm[2,1]) * 100
print(sprintf("La precisión del modelo es: %.2f %%",k_medoids_euc_ave))
```

```
## [1] "La precisión del modelo es: 73.08 %"
```

Como se puede apreciar el rendimiento del modelo presenta valores similares al obtenido con los árboles de decisión.

Modelo "No supervisado" basado en distancias (*manhattan*)

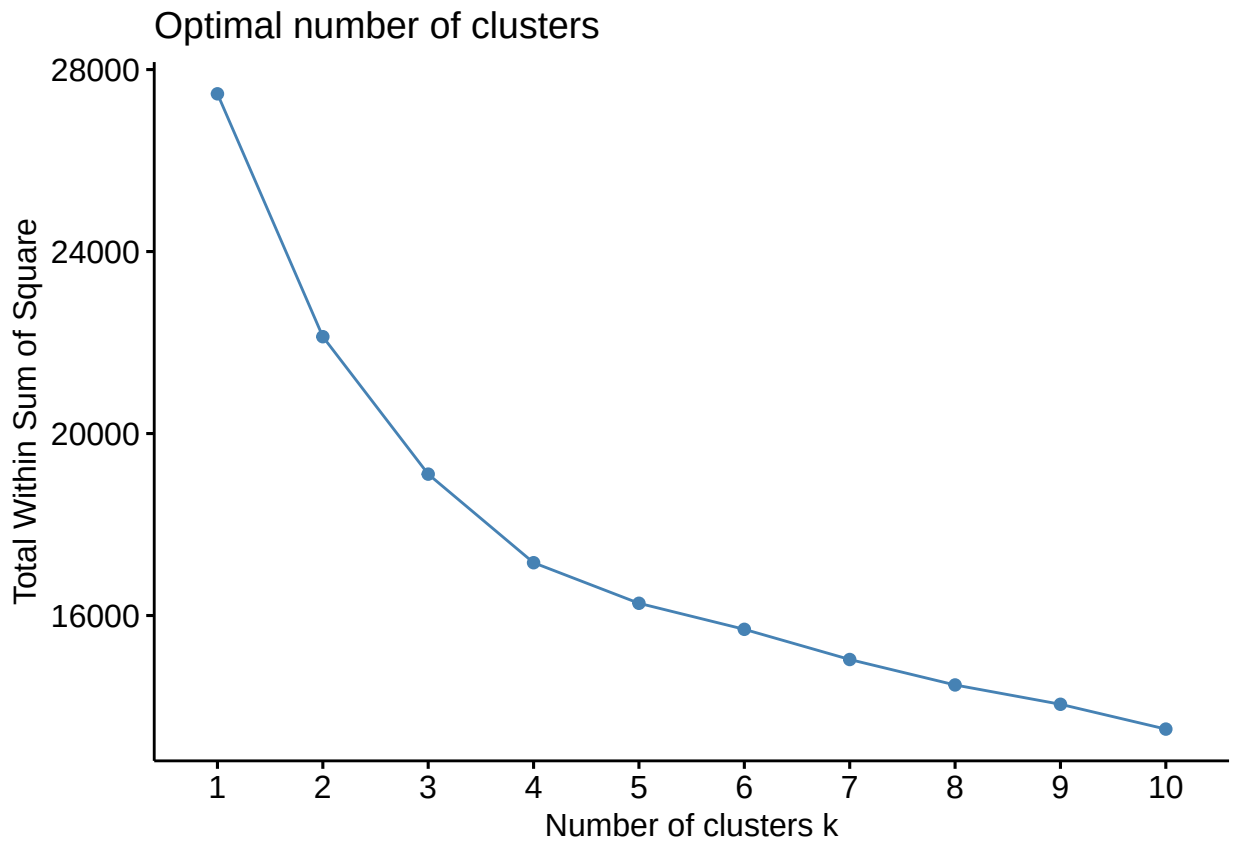
En este apartado se evaluará el mismo conjunto de datos sobre el mismo algoritmo pero basado en un método de distancias diferente, en este caso se usa la distancia de *manhattan* que en ocasiones puede presentar mejores rendimientos frente a datos con peor calidad. Tal y como se comentó en el apartado anterior, este método es el más indicado para evaluar datos con ruido o con *outliers*, por lo que será interesante comparar el rendimiento del proceso de agrupamiento con este método, ya que estos datos están tratados contra *outiers*.

Para obtener una comparativa de ambos métodos se repetirá exactamente el mismo proceso que para la distancia euclídea.

Utilización de las medidas de calidad para determinar k

Primeramente se repite la prueba del codo, en este caso se mantienen todos los parámetros de la prueba anterior solo se cambia el parámetro *method* que se establece a "*manhattan*"

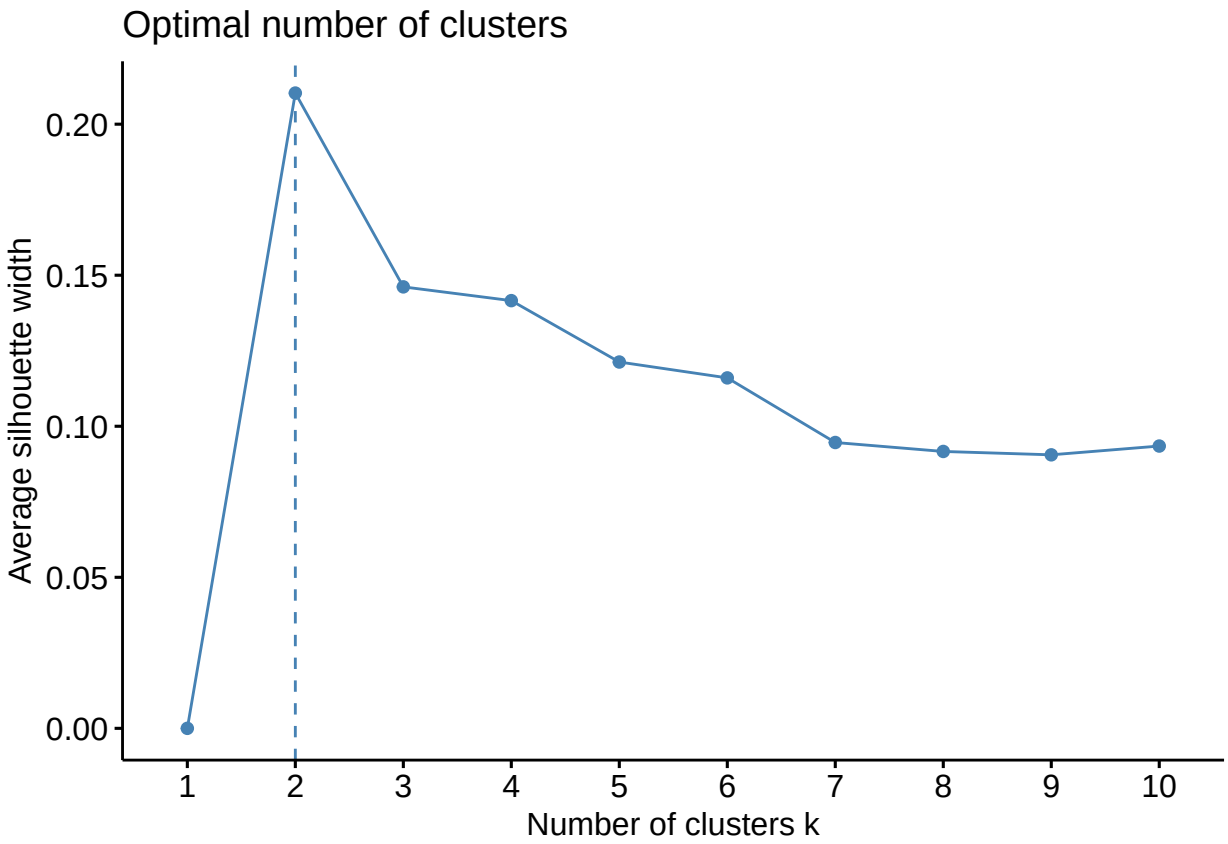
```
library(cluster)
library(factoextra)

fviz_nbclust(x = diab_predictors, FUNcluster = pam, method = "wss", k.max = 10,
             diss = dist(diab_predictors, method = "manhattan"))
```



Como se puede apreciar en los resultados el test arroja resultados similares a los obtenidos con la prueba aplicada con la distancia euclídea. Se aplica entonces la prueba de la silueta a ver si arroja resultados similares o hay alguna variación en el cálculo de la k óptima.

```
library(cluster)
library(factoextra)
```

```r
fviz_nbclust(x = diab_predictors, FUNcluster = pam, method = "silhouette", k.max = 10,
             diss = dist(diab_predictors, method = "manhattan"))
```

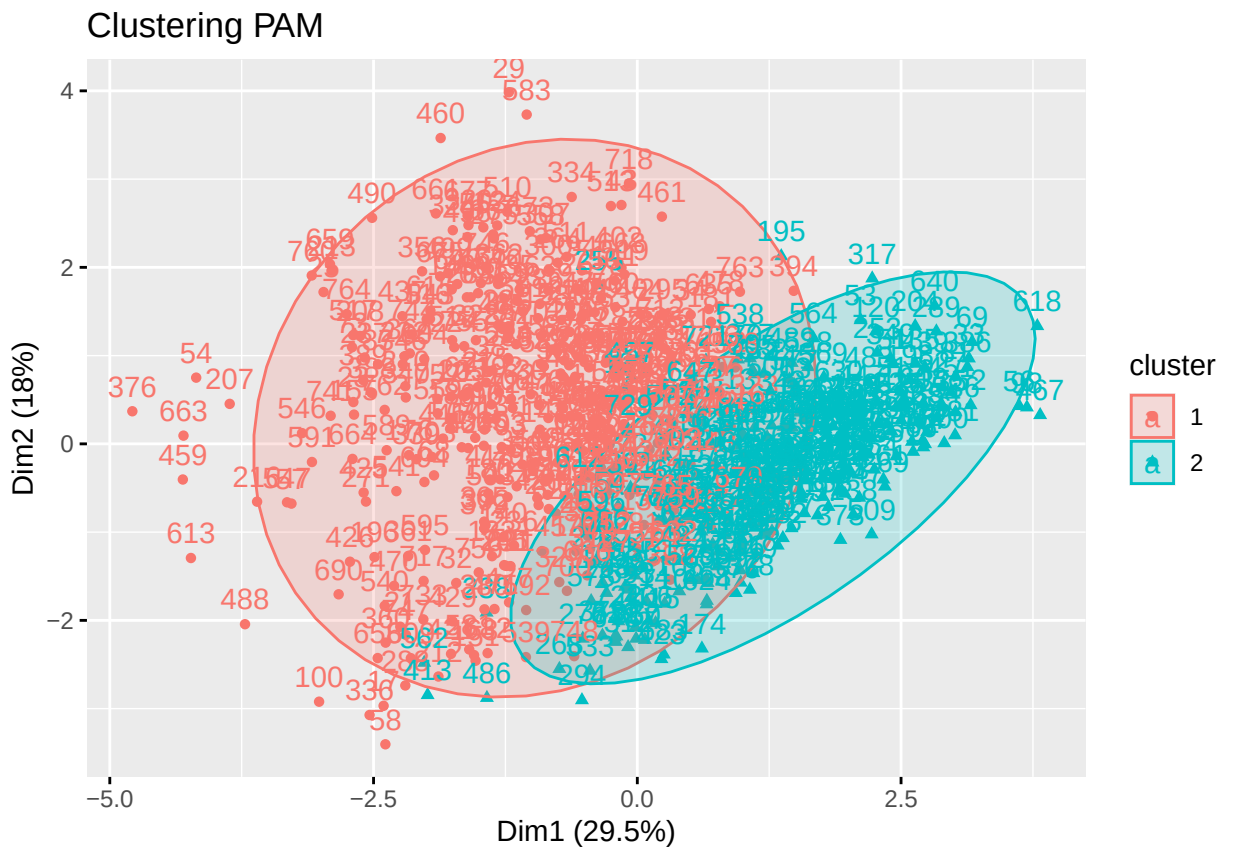


De manera similar se obtiene que el número de k óptimo es 2. Con lo cual hasta ahora no se han notado diferencias significativas entre ambos métodos. Se procede entonces a la creación del modelo y a evaluar su calidad.

Creación del modelo para k =2

Tal y como se implementó el modelo basado en la métrica de la distancia euclídea, se procede a evaluar la función *pam()* pero con la distancia de *manhatan* e igualmente con el valor de k = 2.

```r
# Create the model
k_medoids <- pam(diab_predictors,k=2, metric = "manhattan")

# Plot the clusters
fviz_cluster(object = k_medoids, data = diab_predictors, ellipse.type = "norm") +
  labs(title = "Clustering PAM")
```



Ahora si se notan cierta diferencia en los resultados, se puede apreciar un mayor solapamiento entre ambos *clusters*, lo que puede indicar una pérdida en la capacidad clasificadora del modelos, para corroborar esta suposición se calcula el valor de la exactitud de la clasificación.

Evaluación de la clasificación

Se implementa de nuevo la matriz de confusión y se obtiene la medida cuantitativa de la calidad del proceso de agrupamietno.

```
# Get the accuray of the model
cm <- table(k_medoids$clustering, diab_target)
cm
```

```
##    diab_target
##       0   1
##   1 202 168
##   2 245  35
```

```
k_medoids_man_ave <- (cm[1,2] + cm[2,1]) / (cm[1,1] + cm[2,2] +cm[1,2] + cm[2,1]) * 100
print(sprintf("La precisión del modelo es: %.2f %%",k_medoids_man_ave))
```

```
## [1] "La precisión del modelo es: 63.54 %"
```

Corroborando los resultados obtenidos de la visualización, se aprecia un detrimento de la calidad clasificadora del modelo aproximadamente de un 10%, con lo cual se puede afirmar que para este *dataset* la elección del método de cálculo basado en la distancia euclídea es el más apropiado.

Modelo supervisado sobre los datos originales.

En este apartado se aplica al conjunto de datos original un algoritmo supervisado. Se selecciona el algoritmo K Nearest Neighbours estudiado en clase para clasificar los registros de los datos en positivos o negativos para diabetes.

Este algoritmo se basa en la métrica de distancia, inicialmente calcula las distancias entre las observaciones y pregunta a los k vecinos más cercanos acerca de cuál debería ser su clase, entonces cada vecino decide y la clase que tenga más votos a favor es la seleccionada para clasificar cada observación. A priori parece un algoritmo muy sencillo, pero es capaz de mostrar muy buena capacidad predictiva en algunos casos. Este modelo tiene la condición que las variables predictoras deben ser numéricas y que la variable objetivo sea categórica, condiciones ambas que cumplen el juego de datos utilizado en la práctica.

Como es de esperar para las tareas de clasificación se utilizan los datos divididos en los grupos de *training* y *testing*.

Medidas de calidad del modelo KNN

En este caso al tratarse de un algoritmo de agrupamiento, las medidas de calidad están muy relacionadas a las conocidas de los demás algoritmos. Básicamente se sustentan sobre las diferencias y similitudes entre los diferentes grupos, en este caso las medidas son:

- Comparar centroides: Esta medida compara cuán distantes están los centros de cada uno de los grupos, siendo los centros (centroides) el punto equivalente a la media de los valores de todos los objetos para todos los atributos.
- Comparación de enlace sencillo: Esta comparación mide cuán diferentes son dos grupos basándose en la distancia entre los dos elementos más cercanos, o sea los extremos de cada grupo en la dirección del grupo mas cercano con el que se desea comparar. Será mejor el modelo que proporcione distancias de enlace sencillo más grandes.
- Comparación de enlace completo: Similar al enlace sencillo pero cambian los objetos por los más alejados ente cada grupo.Será mejor el modelo que proporcione distancias de enlace complejo más grandes.

Eleción de k.

La elección de k es una de las tareas más importantes al usar este algoritmo, es evidente que los resultados pueden cambiar en dependencia de a cuantos vecinos se le pregunte, por este motivo es necesario realizar un pequeño análisis para seleccionar el valor de k óptimo.

Este valor puede ser determinado por varios métodos, como por ejemplo por el método del codo o el método de valor máximo de *accurancy*, también en ocasiones se suele utilizar una relación matemática para evaluar este algoritmo, se en la cantidad de registros N, y se evalúa como la mitad de la raíz cuadrada de N (k = sqrt(N)/2). En este caso se evaluarán el método del máximo valor del porcentaje de *accuracy*.

El siguiente procedimiento evalúa la capacidad predictiva del modelo para todos los valores de k entre 1 y 50, y se actualiza el valor de k siempre que se detecte un valor de *accuracy* mayor que el mejor encontrado anteriormente. También se muestra una gráfica con estos datos para facilitar la comprensión de los resultados.

```
library(class)
set.seed(2021)

# Set the inicial values
accuracies = 1
best_k = 1
best_accuracy = 0

for (i in 1:50){
    # Fit the model
```

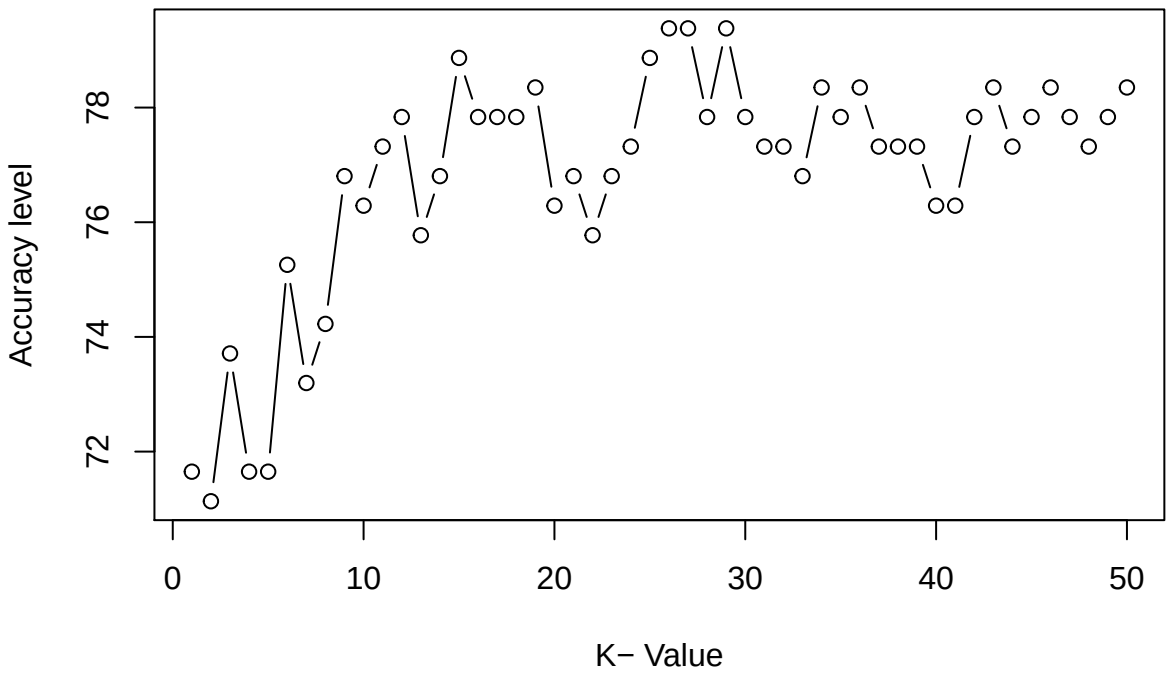
```
    knn_model <-  knn(train = diab_train.x, diab_test.x, diab_train.y , k= i )

    # Get the accuracy value of the ieration and store in a accuracy list
    accuracy <- 100 * sum(diab_test.y == knn_model)/NROW(diab_test)
    accuracies[i] <- accuracy

    # Compare and update the best values
    if(accuracy > best_accuracy){
      best_accuracy <- accuracy
      best_k <- i
      # Print the updates
      k=i
      cat(k,'=',accuracy,'\n')
    }
}
```

```
## 1 = 71.64948
## 3 = 73.71134
## 6 = 75.25773
## 9 = 76.80412
## 11 = 77.31959
## 12 = 77.83505
## 15 = 78.86598
## 26 = 79.38144
```

```
print(sprintf("El mejor valor de accuracy es: %.2f %% , con k = %i",best_accuracy, best_k))
```

```
## [1] "El mejor valor de accuracy es: 79.38 % , con k = 26"
```

```
plot(accuracies, type="b", xlab="K- Value",ylab="Accuracy level")
```



Creación y evaluación del rendimiento del modelo KNN

Una vez calculado el valor óptimo de k se procede a evaluar el modelo con este valor y obtener el valor de su rendimiento.

```
library(class)
set.seed(2021)
knn_model <- knn(train = diab_train.x, diab_test.x, diab_train.y , k= best_k )
cm <- table(diab_test.y, knn_model, dnn = c("Actual", "Predicted"))

cm
```

```
##       Predicted
## Actual   0   1
##      0 127   7
##      1  35  25
```

```
knn_ave <- (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] +cm[1,2] + cm[2,1]) * 100
print(sprintf("La precisión del modelo es: %.2f %%",knn_ave))
```

```
## [1] "La precisión del modelo es: 78.35 %"
```

Se puede aprecair que los resultados obtenidos son similares a los obtenidos con los datos de entrenamiento y a los obtenidos hasta el momento por los demás algoritmos.

Modelo supervisado sobre datos modificados (SDV).

Para este apartado se seguirá el mismo procedimiento que para el apartado anterior, solo que se utilizarán los datos modificados en la práctica anterior con el procedimiento "Singular Value Decomposition SDV". El objetivo de utilizar estos datos es comparar la capacidad predictiva del modelo para ambos conjuntos de datos, teniendo en cuenta que los modificados son una representación de los originales y con suficiente información como para ser estadísticamente similares a los originales.

Elección de k

De manera similar es necesario encontrar el valor óptimo de k.

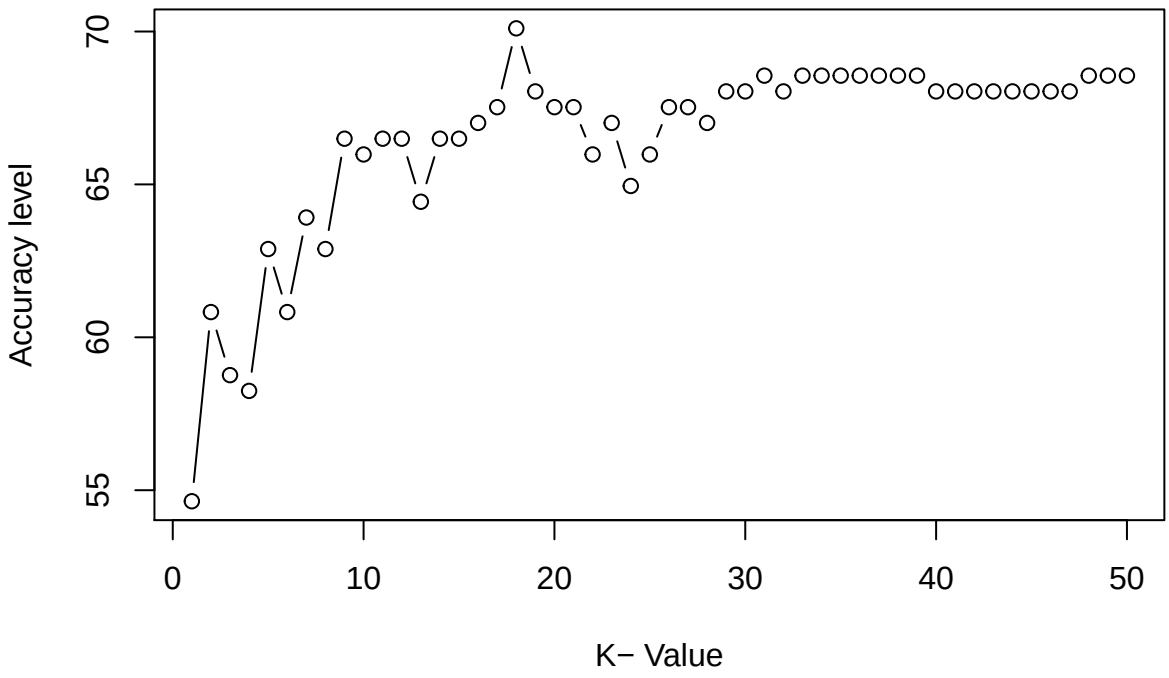
```
set.seed(2021)

# Set the inicial values
accuracies = 1
best_k = 1
best_accuracy = 0

for (i in 1:50){
    # Fit the model
    knn_model <-  knn(train = svd_train.x, svd_test.x, svd_train.y , k= i )

    # Get the accuracy value of the ieration and store in a accuracy list
    accuracy <- 100 * sum(svd_test.y == knn_model)/NROW(svd_test)
    accuracies[i] <- accuracy

    # Compare and update the best values
    if(accuracy > best_accuracy){
      best_accuracy <- accuracy
      best_k <- i
      # Print the updates
      k=i
      cat(k,'=',accuracy,'\n')
    }
}
```

```
## 1 = 54.63918
## 2 = 60.82474
## 5 = 62.8866
## 7 = 63.91753
## 9 = 66.49485
## 16 = 67.01031
## 17 = 67.52577
## 18 = 70.10309
```

```
print(sprintf("El mejor valor de accuracy es: %.2f %% , con k = %i",best_accuracy, best_k))
```

```
## [1] "El mejor valor de accuracy es: 70.10 % , con k = 18"
```

```
plot(accuracies, type="b", xlab="K- Value",ylab="Accuracy level")
```



Creación y evaluación del rendimiento del modelo KNN

Una vez calculado el valor óptimo de k se procede a evaluar el modelo con este valor y obtener el valor de su rendimiento.

```
library(class)
set.seed(2021)
knn_model <- knn(train = svd_train.x, svd_test.x, svd_train.y , k= best_k )
cm <- table(svd_test.y, knn_model, dnn = c("Actual", "Predicted"))

cm
```

```
##       Predicted
## Actual   0   1
##      0 125   9
##      1  53   7
```

```
svd_ave <- (cm[1,1] + cm[2,2]) / (cm[1,1] + cm[2,2] +cm[1,2] + cm[2,1]) * 100
print(sprintf("La precisión del modelo es: %.2f %%",svd_ave))
```

```
## [1] "La precisión del modelo es: 68.04 %"
```

Comparativa entre KNN para los datos originales y modificados

Como se puede apreciar en ambos apartados, la capacidad de clasificación del algoritmo para ambos conjuntos de datos es similar. Esto sucede porque realmente ambos conjuntos tienen la misma información útil, y esta información es la que consumen los algoritmos para clasificar cada una de las observaciones. En este caso se había aplicado SVD, pero debido a las características de los datos originales no fue muy beneficiosa la reducción,

los datos estaban poco relacionados entre si y además eran algo bajo el número de atributos/variables, con lo cual la única diferencia significativa entre estos conjuntos de datos era su tamaño y su formato.

Es importante destacar que para las evaluaciones se han utilizado todas las variables devueltas por el algoritmo, ya que en su momento la diferencia de información entre este número de atributos y el inmediato inferior era demasiado significativa. Cabe la posibilidad de que al aplicar este mismo método sobre datos con características diferentes se pudiesen obtener mejoras considerables (tiempo de ejecución, almacenamiento, *accuracy*), pero este conjunto de datos no fue el caso, los valores del rendimiento se este modelo oscila entre los valores mostrados por los demás modelos de la práctica.

Análisis de los resultados

Luego de haber aplicado todos estos modelos a los dos *datasets* se obtienen los siguientes resultados, es preciso destacar que ninguno de los modelos presenta un rendimiento significativamente superior al resto ofreciendo todos los algoritmos valores de rendimiento similares. Estos resultados pueden estar condicionados por muchos motivos, entre los que se encuentran, la preparación de los datos, el tratamiento de los valores ausentes y extremos, incluso la posible selección de la cantidad de variables luego de la reducción de la dimensionalidad o la calidad de los datos en sí. Comparando estos resultados con otros análisis del mismo estilo realizados en la plataforma kaggle he podido apreciar que mis resultados están dentro de los rangos normales de los ejemplos consultados. En la gráfica que se muestra a continuación se aprecian los valores de *accuracy* para cada uno de los modelos implementados.

```
library(plotly)

aves <- c(c50_ave, k_medoids_euc_ave ,k_medoids_man_ave, knn_ave, svd_ave)
algo <- c("C5.0s", "KNN (Euclidean)", "KNN (Manhattan)", "K Nearest N", "K Nearest N (SVD)")

fig <- plot_ly(
  x = algo,
  y = aves,
  name = "Average",
  type = "bar"
)

fig
```

Bibliografía

- Material docente de la asignatura Minería de Datos
- Contenido de la plataforma kaggle
- Documentación oficial de R