

Trabajo Final Introducción a la Ciencia de Datos

Juanjo Sierra

27 de diciembre de 2018

Planteamiento

El trabajo final de la asignatura *Introducción a la Ciencia de Datos* se divide en dos secciones. Consiste en realizar un estudio sobre un conjunto de datos de regresión y otro sobre un conjunto de datos de clasificación. Se aplicarán distintas técnicas aprendidas durante la asignatura para conseguir los resultados adecuados.

Librerías y paquetes a cargar

```
library(scales)
library(dummies)

## dummies-1.5.6 provided by Decision Patterns

library(ggplot2)
library(dplyr)

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##   filter, lag

## The following objects are masked from 'package:base':
##
##   intersect, setdiff, setequal, union

library(tidyr)
library(caret)

## Loading required package: lattice

library(kknn)

##
## Attaching package: 'kknn'

## The following object is masked from 'package:caret':
##
##   contr.dummy

library(MASS)

##
## Attaching package: 'MASS'

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

Análisis de los datos

En este primer apartado se estudiarán la distribución de los datos, su rango de valores, y toda aquella información que se considere necesaria para la correcta aplicación del trabajo correspondiente a cada parte.

Análisis para regresión

En primer lugar se realizará el estudio de la base de datos de regresión. En este caso el conjunto de datos a analizar es **Friedman**, que se ha descargado desde el repositorio de datasets de la asignatura. Se puede leer utilizando la siguiente orden:

```
friedman = read.csv("Datos/friedman/friedman.dat", header = FALSE, comment.char = "@")
head(friedman)
```

```
##           V1           V2           V3           V4           V5           V6
## 1 0.6964817 0.3584375 0.4258343 0.33031373 0.22249090 11.09496
## 2 0.5903899 0.4306749 0.8690418 0.07091161 0.63430253 13.22921
## 3 0.8276557 0.6178330 0.9494409 0.67013843 0.64080838 25.33973
## 4 0.8107169 0.2621162 0.4541944 0.85470608 0.27976951 15.18159
## 5 0.4068430 0.8161745 0.8611055 0.12890196 0.15747881 14.43310
## 6 0.6299940 0.3821170 0.9819543 0.98471273 0.07506318 20.97857
```

Dado que los nombres asignados a las variables no aportan ninguna información, y en el resumen del dataset en formato KEEL podemos comprobar que sus nombres tampoco son representativos, se procede a asignarles una notación genérica.

```
n = length(names(friedman))-1
names(friedman)[1:n] = paste ("X", 1:n, sep="")
names(friedman)[n+1] = "Y"
head(friedman)
```

```
##           X1           X2           X3           X4           X5           Y
## 1 0.6964817 0.3584375 0.4258343 0.33031373 0.22249090 11.09496
## 2 0.5903899 0.4306749 0.8690418 0.07091161 0.63430253 13.22921
## 3 0.8276557 0.6178330 0.9494409 0.67013843 0.64080838 25.33973
## 4 0.8107169 0.2621162 0.4541944 0.85470608 0.27976951 15.18159
## 5 0.4068430 0.8161745 0.8611055 0.12890196 0.15747881 14.43310
## 6 0.6299940 0.3821170 0.9819543 0.98471273 0.07506318 20.97857
```

Ahora podemos comprobar de forma más directa que existen 5 variables de entrada (X1-5) que determinan una única variable de salida (Y). Es interesante comprobar las dimensiones del dataset para poder asegurar que se está asumiendo lo correcto.

```
dim(friedman)
```

```
## [1] 1200    6
```

Con esto se puede confirmar que existen un total de 1200 ejemplos en el conjunto de datos, cada uno con 6 variables (5 de entrada y 1 de salida).

Antes de nada se debe comprobar si existen valores perdidos en el dataset. Para esto se puede utilizar la función `anyNA`:

```
anyNA(friedman)
```

```
## [1] FALSE
```

Este resultado indica que no hay valores perdidos y que por lo tanto no es necesario imputar ni tomar ninguna decisión para restablecer dichos valores.

Utilizando la función `summary` se puede obtener una visión más completa del dataset, arrojando nuevos valores interesantes para su estudio como los rangos de las variables, sus cuartiles o su media y mediana.

```
summary(friedman)
```

```
##           X1           X2           X3
## Min.      :0.001212   Min.      :0.0001603   Min.      :0.0006546
## 1st Qu.:0.249184     1st Qu.:0.2423287   1st Qu.:0.2485096
## Median :0.519293     Median :0.4932687   Median :0.4993111
## Mean      :0.506193     Mean      :0.4999592   Mean      :0.4995141
## 3rd Qu.:0.751131     3rd Qu.:0.7655960   3rd Qu.:0.7441912
## Max.      :0.999719     Max.      :0.9996775   Max.      :0.9990619
##           X4           X5           Y
## Min.      :0.0002123   Min.      :0.0004299   Min.      : 0.664
## 1st Qu.:0.2703118     1st Qu.:0.2578755   1st Qu.:10.859
## Median :0.5328840     Median :0.4753492   Median :14.654
## Mean      :0.5122272     Mean      :0.4928214   Mean      :14.567
## 3rd Qu.:0.7566648     3rd Qu.:0.7385440   3rd Qu.:18.494
## Max.      :0.9994802     Max.      :0.9995394   Max.      :28.590
```

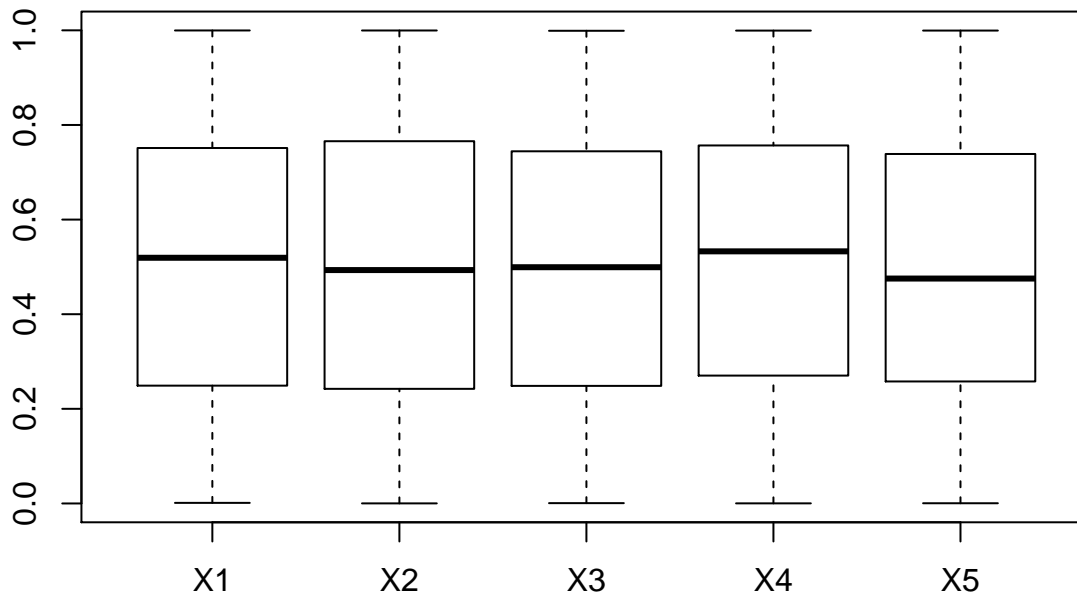
Se puede calcular la desviación típica utilizando la función `sd`:

```
sapply(friedman, sd)
```

```
##           X1           X2           X3           X4           X5           Y
## 0.2910749 0.2938897 0.2900604 0.2842922 0.2795909 5.1844877
```

Mediante una gráfica box-plot se puede comprobar si los datos están escalados o centrados, y valorar cómo se comparan sus varianzas. Utilizando la función `var` se calculan dichas varianzas.

```
boxplot(friedman[1:5])
```



```
sapply(friedman[1:5], var)
```

```
##           X1           X2           X3           X4           X5
## 0.08472459 0.08637114 0.08413502 0.08082205 0.07817105
```

En función de la gráfica obtenida, se puede valorar que los datos están más o menos centrados y escalados entre 0 y 1, y que sus varianzas son muy similares. Esto se puede comprobar con los resultados reflejados por

la función anteriormente citada.

A continuación se puede comprobar si existen ejemplos duplicados, y eliminarlos del dataset. Para ello se utiliza la función `duplicated` acompañado de la función `any`:

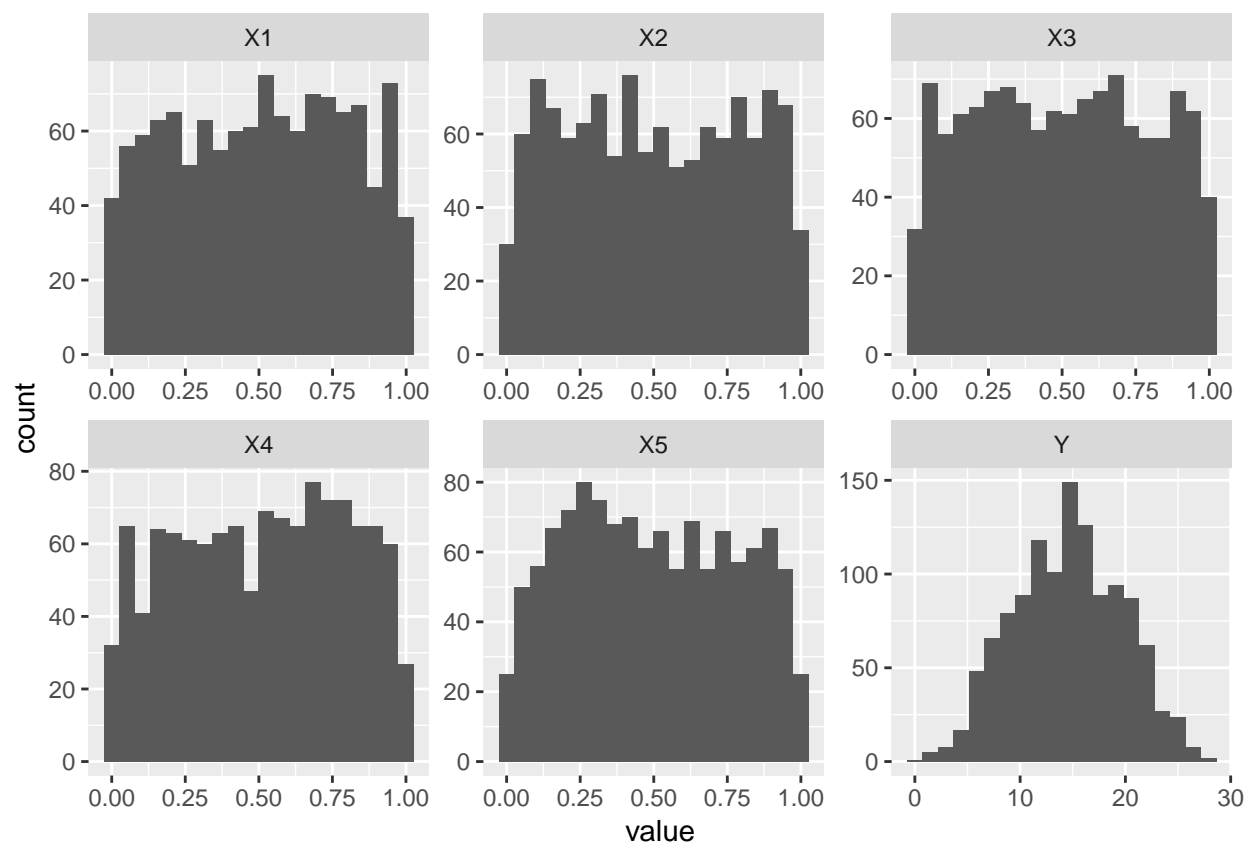
```
any(duplicated(friedman))
```

```
## [1] FALSE
```

Como no hay duplicados se puede continuar con el estudio sin realizar ninguna alteración en el conjunto de datos.

Se puede comprobar la distribución de los datos utilizando histogramas. En este caso se utilizan simultáneamente las librerías `ggplot2` y `tidyr`.

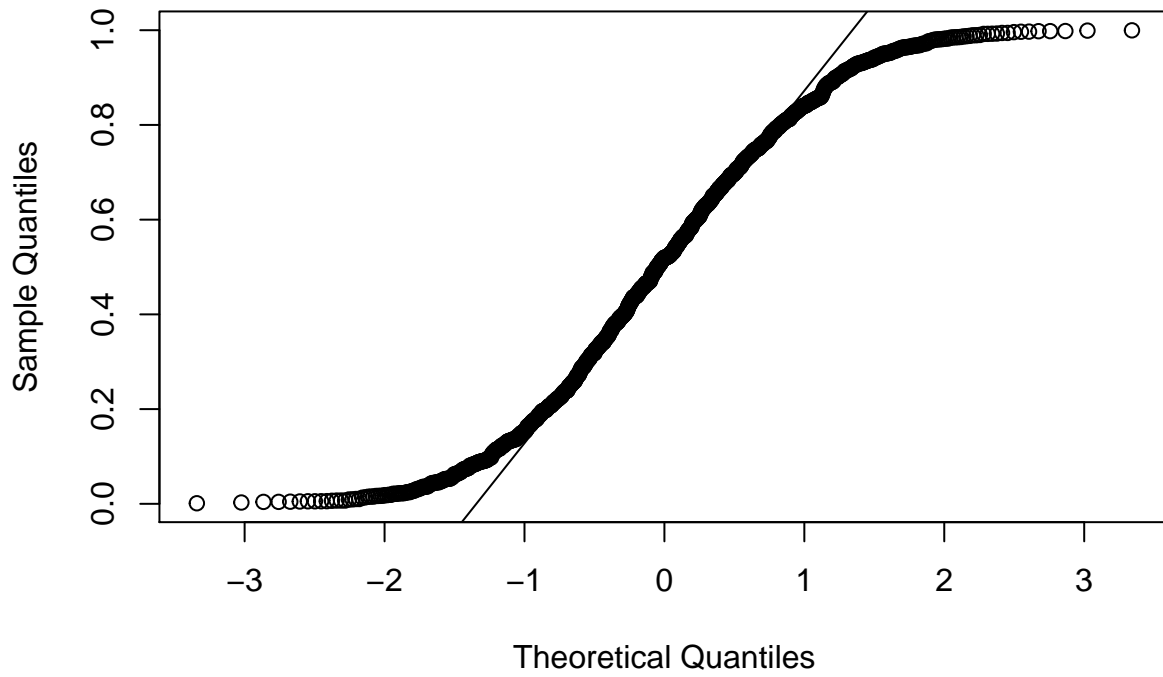
```
friedman %>% gather() %>% ggplot(aes(value)) +  
facet_wrap(~ key, scales = "free") + geom_histogram(bins = 20)
```



No parece que las variables sigan una distribución normal, pero pese a todo no está de más comprobarlo estadísticamente. Se puede estudiar la normalidad de los datos utilizando `qqnorm` y `qqline` para dar una primera visión de cómo se aproximan a una normal. Igualmente, se debe realizar el test de Shapiro-Wilk para contrastar la normalidad o no de los datos.

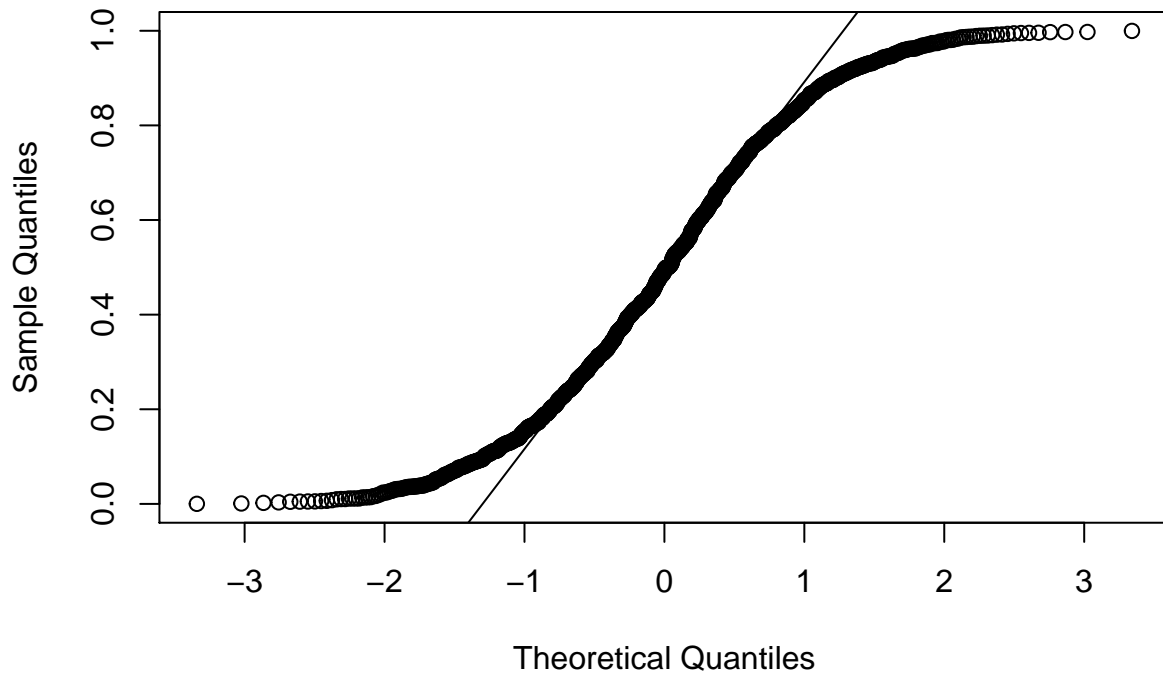
```
qqnorm(friedman$X1)  
qqline(friedman$X1)
```

Normal Q-Q Plot



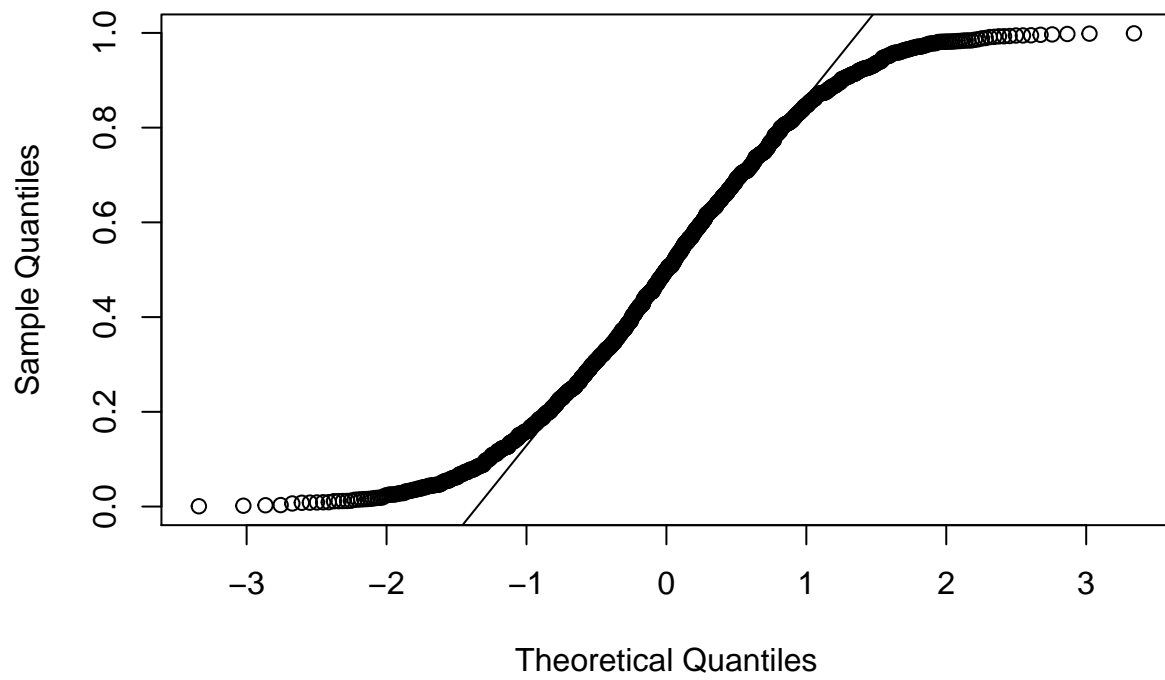
```
qqnorm(friedman$X2)  
qqline(friedman$X2)
```

Normal Q-Q Plot



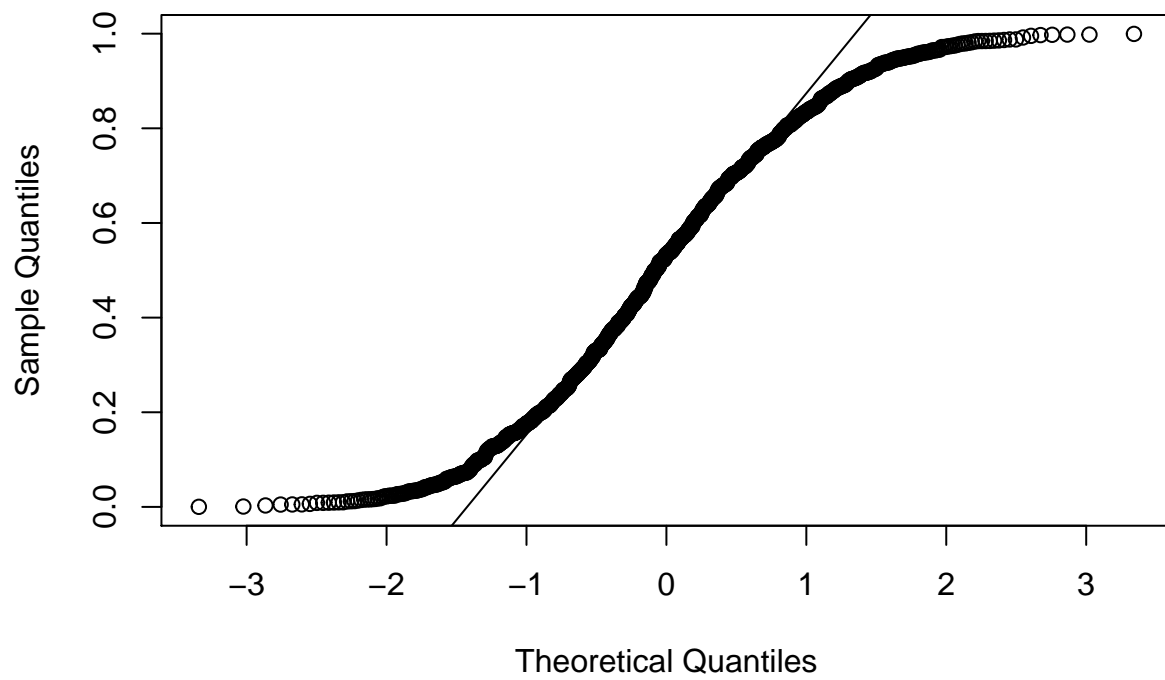
```
qqnorm(friedman$X3)  
qqline(friedman$X3)
```

Normal Q-Q Plot



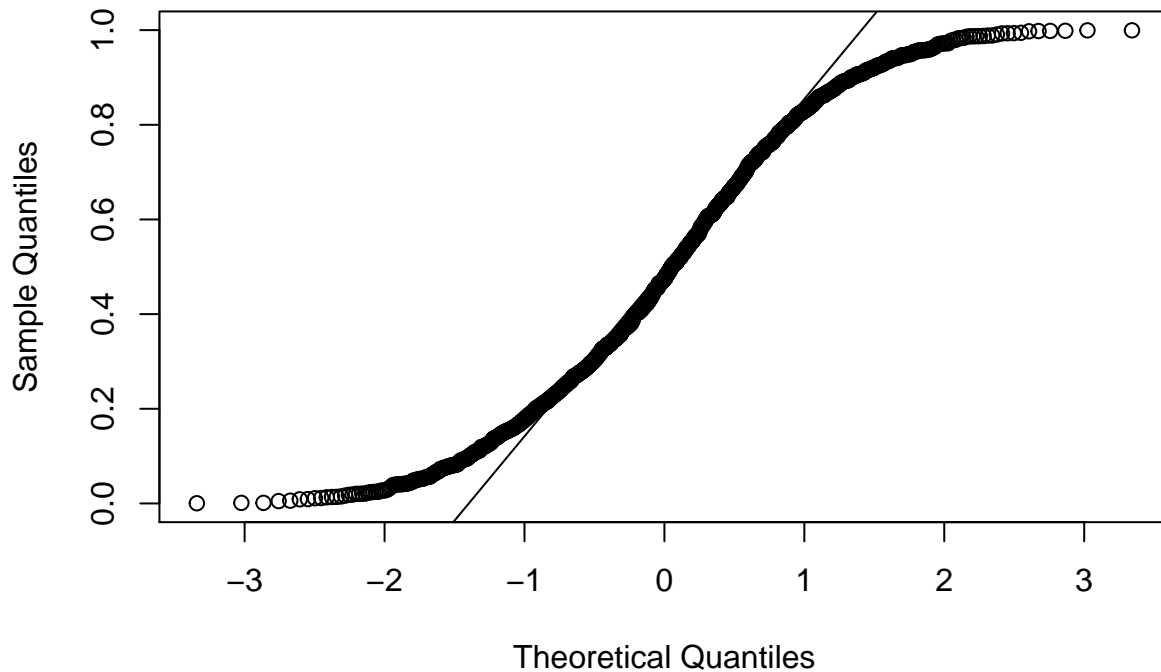
```
qqnorm(friedman$X4)  
qqline(friedman$X4)
```

Normal Q-Q Plot



```
qqnorm(friedman$X5)  
qqline(friedman$X5)
```

Normal Q-Q Plot



En base a las gráficas no se puede concluir de forma clara si las variables se ajustan a una distribución normal por lo que es necesario validar con el test de Shapiro-Wilk.

```
sapply(friedman[1:5], shapiro.test)
```

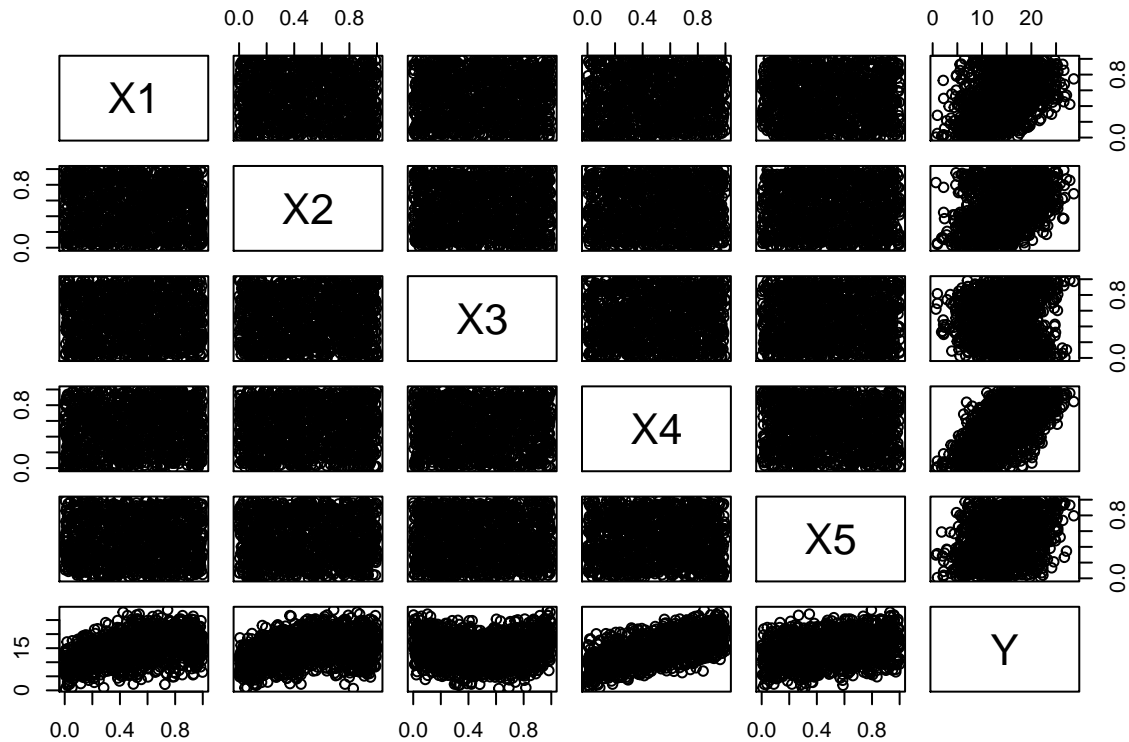
```
##           X1           X2
## statistic 0.9551311      0.948869
## p.value   1.032993e-18    5.451349e-20
## method    "Shapiro-Wilk normality test" "Shapiro-Wilk normality test"
## data.name "X[[i]]"        "X[[i]]"
##           X3           X4
## statistic 0.9548493      0.9564912
## p.value   8.993121e-19    2.034613e-18
## method    "Shapiro-Wilk normality test" "Shapiro-Wilk normality test"
## data.name "X[[i]]"        "X[[i]]"
##           X5
## statistic 0.9583177
## p.value   5.180939e-18
## method    "Shapiro-Wilk normality test"
## data.name "X[[i]]"
```

Los p-valores tan bajos que se obtienen de cada una de las variables con el test de Shapiro-Wilk provocan que se rechace la hipótesis de que dichas variables siguen una distribución normal.

Además, como el rango de las variables está entre 0 y 1 (como se pudo comprobar anteriormente con el `summary` y el box-plot), no es necesario realizar un escalado ni una transformación en los valores. En este punto se puede afirmar que los datos están listos para poder trabajar con ellos.

Para continuar con el análisis del dataset se puede mostrar una gráfica comparativa de cada variable con respecto del resto, así se puede visualizar cuáles tienen una correlación alta.

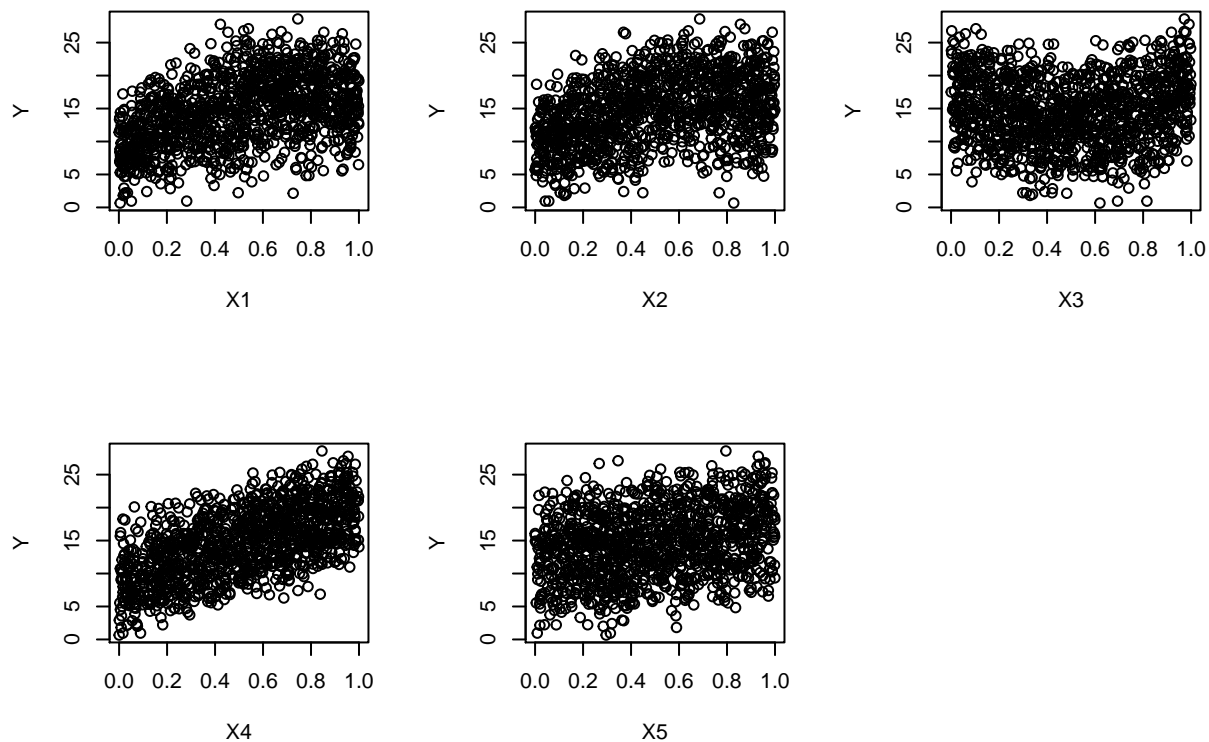
```
plot(friedman)
```



Visualizando la gráfica anterior se aprecia que entre las variables de entrada no existe ninguna correlación clara, de hecho los puntos se distribuyen casi por todo el espacio de la gráfica. Las que sí tienen más interés son aquellas que comparan las distintas variables de entrada con la variable de salida Y. Ver directamente estas gráficas permitirá averiguar de un vistazo cuál tiene más potencial de determinar qué valor de salida obtendrá dicho ejemplo.

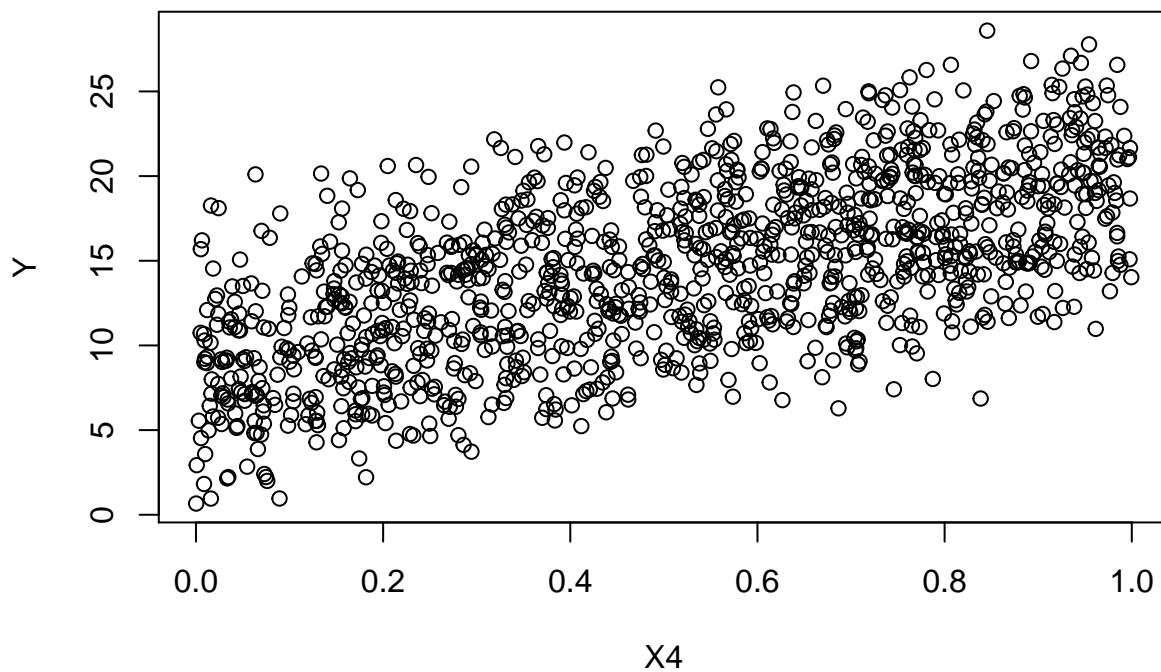
```
plotY = function (x,y) {
  plot(friedman[,y]~friedman[,x], xlab=names(friedman)[x], ylab=names(friedman)[y])
}

par(mfrow=c(2,3))
x = sapply(1:(dim(friedman)[2]-1), plotY, dim(friedman)[2])
par(mfrow=c(1,1))
```

Basándose en un modelo de regresión lineal, se puede especular que la variable X4 parece tener una correlación más alta con la variable de salida Y, y por tanto podría resultar en un mejor modelo. A continuación se muestra la gráfica de X4 frente a Y más grande para poder apreciar mejor la posible correlación.

```
plotY(4,dim(friedman)[2])
```



La correlación de las variables entre sí y con la variable de salida puede obtenerse de forma directa gracias a la función `cor`:

```
cor(friedman)
```

```
##           X1           X2           X3           X4           X5           Y
## X1  1.000000000 -0.03227730  0.009162253  0.09172183  0.01124122  0.4334883
## X2 -0.032277302  1.00000000  0.010233226  0.03639529  0.02452585  0.3713814
## X3  0.009162253  0.01023323  1.000000000  0.03883400  0.02110537  0.0356199
## X4  0.091721829  0.03639529  0.038834002  1.00000000 -0.02445055  0.6157779
## X5  0.011241216  0.02452585  0.021105366 -0.02445055  1.00000000  0.2757470
## Y   0.433488308  0.37138140  0.035619901  0.61577794  0.27574703  1.0000000
```

Como se había supuesto anteriormente en función de las gráficas obtenidas, es la variable X4 la que más correlación tiene con la variable de salida Y (~0.616).

Análisis para clasificación

El conjunto de datos que se va a utilizar para el estudio de clasificación es **Australian**, también descargado desde el repositorio de datasets correspondiente a la asignatura. Se puede leer el dataset completo mediante la siguiente orden:

```
australian = read.csv("Datos/australian/australian.dat", header = FALSE, comment.char = "@")
head(australian)
```

```
##   V1   V2   V3 V4 V5 V6   V7 V8 V9 V10 V11 V12 V13  V14 V15
## 1  1 2208 1146  2  4  4 1585  0  0  0  1  2 100 1213  0
## 2  0 2267   7  2  8  4  165  0  0  0  0  2 160   1  0
## 3  0 2958 175  1  4  4  125  0  0  0  1  2 280   1  0
## 4  0 2167 115  1  5  3   0  1  1 11  1  2  0   1  1
## 5  1 2017 817  2  6  4  196  1  1 14  0  2  60 159  1
## 6  0 1583 585  2  8  8   15  1  1  2  0  2 100   1  1
```

En este caso se comprueba que existen muchas variables cuyo tipo atómico es entero, pero tan sólo algunas resultan ser variables categóricas. Más adelante se verá cuáles son.

De nuevo los nombres asignados a las variables no aportan ninguna información, y ya que en el resumen en formato KEEL correspondiente al dataset podemos comprobar que sus nombres no aportan ninguna información, se procede a asignarles una notación genérica.

```
n = length(names(australian))-1
names(australian)[1:n] = paste ("X", 1:n, sep="")
names(australian)[n+1] = "Y"
head(australian)
```

```
##   X1   X2   X3 X4 X5 X6   X7 X8 X9 X10 X11 X12 X13  X14 Y
## 1  1 2208 1146  2  4  4 1585  0  0  0  1  2 100 1213  0
## 2  0 2267   7  2  8  4  165  0  0  0  0  2 160   1  0
## 3  0 2958 175  1  4  4  125  0  0  0  1  2 280   1  0
## 4  0 2167 115  1  5  3   0  1  1 11  1  2  0   1  1
## 5  1 2017 817  2  6  4  196  1  1 14  0  2  60 159  1
## 6  0 1583 585  2  8  8   15  1  1  2  0  2 100   1  1
```

Para comprobar que se ha leído de forma correcta el dataset, se pueden estudiar sus dimensiones y asegurarse de que hay 690 ejemplos con 14 variables de entrada y una de salida.

```
dim(australian)
```

```
## [1] 690 15
```

Antes de continuar con el resto del estudio del dataset debe asegurarse que no contiene valores perdidos. En caso de haberlos, se deberá tomar una decisión adecuada para solventar la situación.

```
anyNA(australian)
```

```
## [1] FALSE
```

Como no existen valores perdidos en el dataset no hay que hacer ningún cálculo para imputarlos, ni es necesario tomar una decisión al respecto. Ahora sí se puede utilizar `summary` para obtener una visión general del dataset, para averiguar sus cuartiles, su media y mediana y el rango de cada variable.

```
summary(australian)
```

```
##           X1           X2           X3           X4
## Min.      :0.0000   Min.      : 16   Min.      :    0   Min.      :1.000
## 1st Qu.:0.0000   1st Qu.:1942   1st Qu.:   15   1st Qu.:2.000
## Median :1.0000   Median :2629   Median :   125   Median :2.000
## Mean     :0.6783   Mean     :2697   Mean      :1187   Mean     :1.767
## 3rd Qu.:1.0000   3rd Qu.:3525   3rd Qu.:   665   3rd Qu.:2.000
## Max.      :1.0000   Max.      :8025   Max.      :26335   Max.      :3.000
##           X5           X6           X7           X8
## Min.      : 1.000   Min.      :1.000   Min.      :    0.0   Min.      :0.0000
## 1st Qu.: 4.000   1st Qu.:4.000   1st Qu.:    5.0   1st Qu.:0.0000
## Median : 8.000   Median :4.000   Median :   35.0   Median :1.0000
## Mean     : 7.372   Mean     :4.693   Mean      :453.4   Mean     :0.5232
## 3rd Qu.:10.000   3rd Qu.:5.000   3rd Qu.:  219.8   3rd Qu.:1.0000
## Max.     :14.000   Max.      :9.000   Max.     :14415.0   Max.      :1.0000
##           X9           X10          X11          X12
## Min.      :0.0000   Min.      : 0.0   Min.      :0.000   Min.      :1.000
## 1st Qu.:0.0000   1st Qu.: 0.0   1st Qu.:0.000   1st Qu.:2.000
## Median :0.0000   Median : 0.0   Median :0.000   Median :2.000
## Mean     :0.4275   Mean      : 2.4   Mean     :0.458   Mean     :1.929
## 3rd Qu.:1.0000   3rd Qu.: 3.0   3rd Qu.:1.000   3rd Qu.:2.000
## Max.      :1.0000   Max.      :67.0   Max.      :1.000   Max.      :3.000
##           X13          X14           Y
## Min.      :    0   Min.      :    1.0   Min.      :0.0000
## 1st Qu.:   80   1st Qu.:    1.0   1st Qu.:0.0000
## Median :  160   Median :    6.0   Median :0.0000
## Mean     :  184   Mean     :1018.4   Mean     :0.4449
## 3rd Qu.:  272   3rd Qu.:   396.5   3rd Qu.:1.0000
## Max.     :2000   Max.      :100001.0   Max.      :1.0000
```

El resumen del dataset muestra los valores anteriormente citados. Aquí puede observarse qué variables corresponden a datos reales y cuáles a enteros, y entre ellas, se puede apreciar mejor cuáles corresponden a un valor booleano (rango 0-1) y cuáles son variables categóricas con más valores (por ejemplo X12, que toma los valores 1, 2 y 3).

Es interesante saber cuántos valores de clase puede tomar la variable de salida Y, para saber si el dataset está balanceado o no.

```
table(australian$Y)
```

```
##
##    0    1
## 383 307
```

Dados estos datos se puede afirmar que está bastante equilibrada, dado que sólo hay dos clases posibles y toman en torno al 50% de los datos cada una.

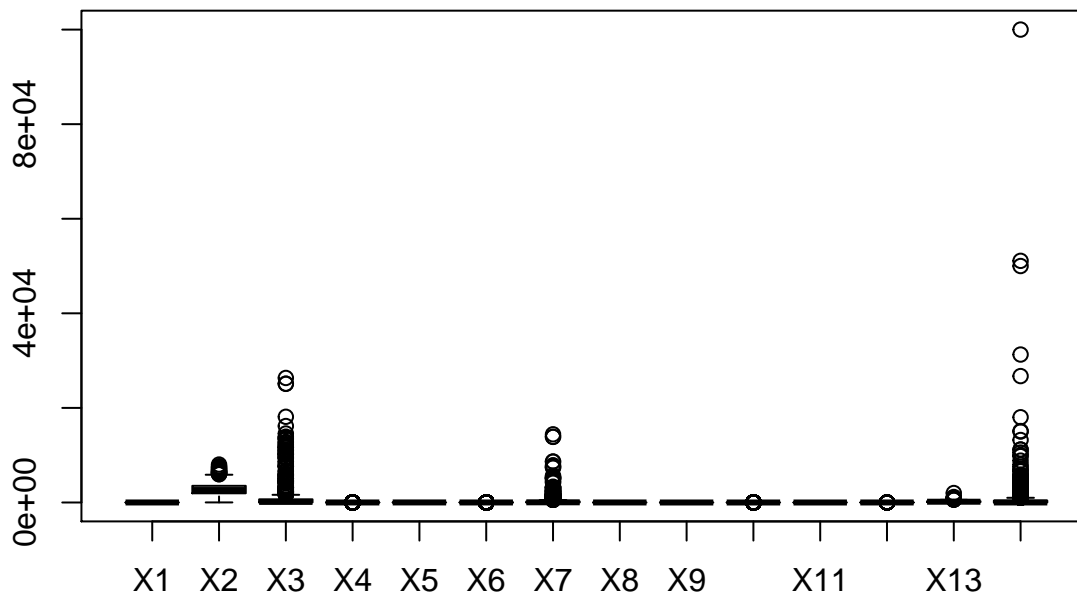
Un dato que no aparece en el resumen y que se puede calcular es la desviación típica. Para ello se utiliza la función `sd`:

```
sapply(australian, sd)
```

```
##           X1           X2           X3           X4           X5
## 0.4674824 1554.5597320 3069.1100423 0.4300628 3.6832648
##           X6           X7           X8           X9          X10
## 1.9923161 1387.9003240 0.4998243 0.4950800 4.8629400
##           X11          X12          X13          X14           Y
## 0.4985919 0.2988131 172.1592735 5210.1025983 0.4973183
```

Con una gráfica box-plot pueden verse de forma gráfica los cuartiles de las variables, además de sus varianzas y lo centrados o escalados que están los datos. Sin embargo, como los valores para este dataset no están normalizados es de esperar que la gráfica no aporte mucha información. Sin embargo, para calcular directamente las varianzas se puede utilizar la función `var`:

```
boxplot(australian[1:14])
```

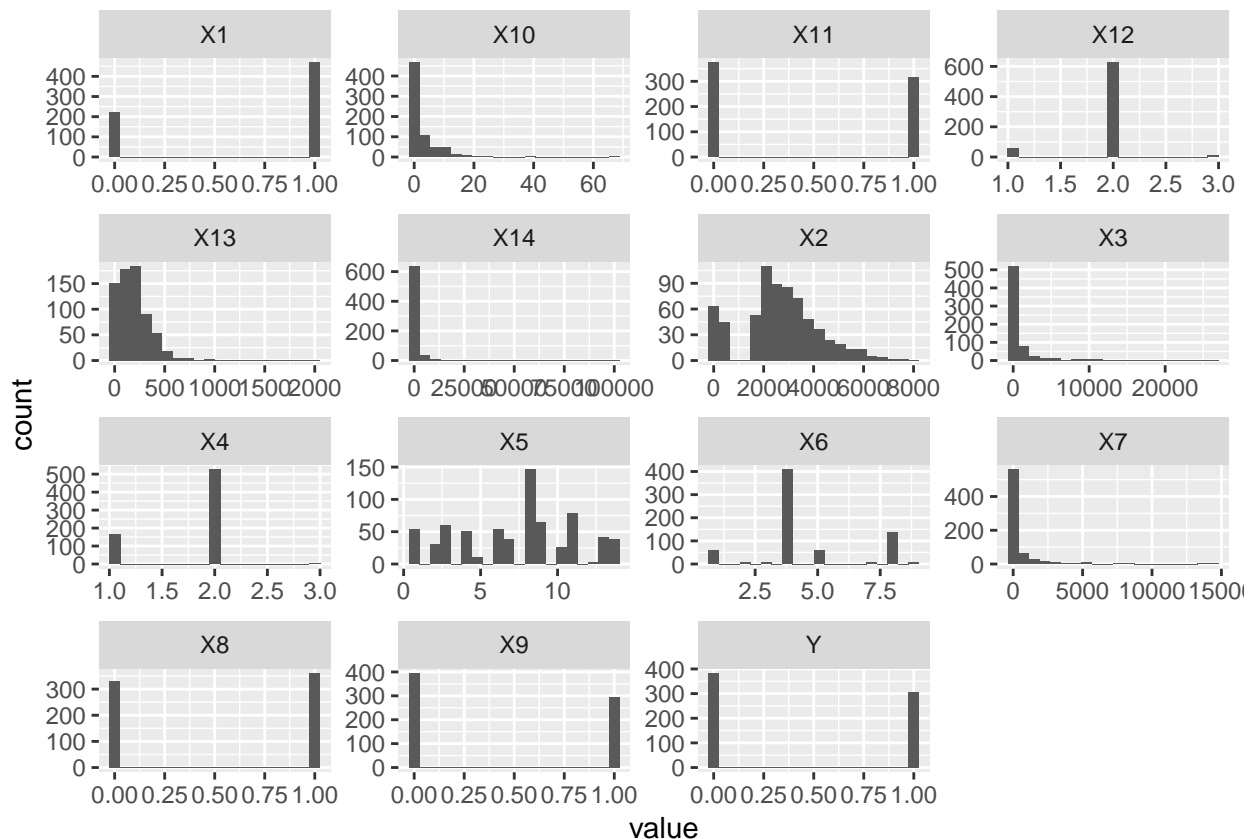


```
sapply(australian[1:14], var)
```

```
##           X1           X2           X3           X4           X5
## 2.185398e-01 2.416656e+06 9.419436e+06 1.849540e-01 1.356644e+01
##           X6           X7           X8           X9          X10
## 3.969323e+00 1.926267e+06 2.498244e-01 2.451042e-01 2.364819e+01
##           X11          X12          X13          X14
## 2.485938e-01 8.928925e-02 2.963882e+04 2.714517e+07
```

Utilizando `tidyr` y `ggplot` se pueden ver estos histogramas que aporten una idea general de la distribución de cada una de las variables del dataset.

```
australian %>% gather() %>% ggplot(aes(value)) +
  facet_wrap(~ key, scales = "free") + geom_histogram(bins = 20)
```



Para obtener de forma más precisa una conclusión con respecto a la normalidad de los datos, se debe realizar el test de Shapiro-Wilk sobre las variables a estudiar. En este caso no tiene sentido sobre las variables categóricas, así que se hará sobre las que no lo son.

```
apply(australian[c(2,3,5,7,10,13,14)], shapiro.test)
```

```
##           X2                               X3
## statistic 0.9629707                        0.4262456
## p.value   3.451734e-12                     4.450405e-42
## method    "Shapiro-Wilk normality test"    "Shapiro-Wilk normality test"
## data.name "X[[i]]"                        "X[[i]]"
##           X5                               X7
## statistic 0.9530628                        0.3413943
## p.value   5.121202e-14                     4.32144e-44
## method    "Shapiro-Wilk normality test"    "Shapiro-Wilk normality test"
## data.name "X[[i]]"                        "X[[i]]"
##           X10                              X13
## statistic 0.5330628                        0.8206891
## p.value   3.662318e-39                     4.787516e-27
## method    "Shapiro-Wilk normality test"    "Shapiro-Wilk normality test"
## data.name "X[[i]]"                        "X[[i]]"
##           X14
## statistic 0.1698544
## p.value   1.405343e-47
## method    "Shapiro-Wilk normality test"
## data.name "X[[i]]"
```

Como el p-valor es muy bajo en todas, se puede afirmar con una confianza muy alta (superior al 99%) que

dichas variables no siguen una distribución normal.

Con respecto a las variables dummy, para este dataset sólo se van a aplicar a aquellas variables que deberían ser consideradas factor a pesar de haber sido leídas como entero. Se trata de las variables X4 y X12, que solamente toman valores 1, 2 y 3. Para no darle más peso a las diferencias entre distintos pares de valores, se toman valores dummy.

```
australian$X4 = as.factor(australian$X4)
australian$X12 = as.factor(australian$X12)

australian = dummy.data.frame(australian, sep=".")
head(australian)
```

```
##   X1   X2   X3 X4.1 X4.2 X4.3 X5 X6   X7 X8 X9 X10 X11 X12.1 X12.2 X12.3
## 1  1 2208 1146    0    1    0  4  4 1585  0  0  0    1    0    1    0
## 2  0 2267    7    0    1    0  8  4  165  0  0  0    0    0    1    0
## 3  0 2958  175    1    0    0  4  4  125  0  0  0    1    0    1    0
## 4  0 2167  115    1    0    0  5  3    0  1  1  11    1    0    1    0
## 5  1 2017  817    0    1    0  6  4  196  1  1  14    0    0    1    0
## 6  0 1583  585    0    1    0  8  8   15  1  1  2    0    0    1    0
##   X13  X14 Y
## 1 100 1213 0
## 2 160    1 0
## 3 280    1 0
## 4   0    1 1
## 5  60  159 1
## 6 100    1 1
```

Como para aplicar el algoritmo k-NN va a ser necesario normalizar los datos, se proceden a escalar a continuación.

```
maxs = apply(australian, 2, max)
mins = apply(australian, 2, min)
australian = as.data.frame(scale(australian, center = mins, scale = maxs - mins))
head(australian)
```

```
##   X1         X2         X3 X4.1 X4.2 X4.3         X5         X6         X7 X8
## 1  1 0.2736921 0.043516233    0    1    0 0.2307692 0.375 0.109954908  0
## 2  0 0.2810588 0.000265806    0    1    0 0.5384615 0.375 0.011446410  0
## 3  0 0.3673367 0.006645149    1    0    0 0.2307692 0.375 0.008671523  0
## 4  0 0.2685729 0.004366812    1    0    0 0.3076923 0.250 0.000000000  1
## 5  1 0.2498439 0.031023353    0    1    0 0.3846154 0.375 0.013596948  1
## 6  0 0.1956549 0.022213784    0    1    0 0.5384615 0.875 0.001040583  1
##   X9         X10 X11 X12.1 X12.2 X12.3 X13         X14 Y
## 1  0 0.00000000    1    0    1    0 0.05 0.01212 0
## 2  0 0.00000000    0    0    1    0 0.08 0.00000 0
## 3  0 0.00000000    1    0    1    0 0.14 0.00000 0
## 4  1 0.16417910    1    0    1    0 0.00 0.00000 1
## 5  1 0.20895522    0    0    1    0 0.03 0.00158 1
## 6  1 0.02985075    0    0    1    0 0.05 0.00000 1
```

Como existen tantas variables es complicado realizar una gráfica comparativa de cada variable con respecto al resto y que esta aporte información útil, por lo que se muestran directamente los valores de correlación utilizando la función `cor`. Como la matriz es enorme se obtiene un subconjunto de aquellos valores superiores a 0.5 y que no sean el 1.0 correspondiente a la diagonal.

```
correlation = as.data.frame(as.table(cor(australian)))
subset(correlation, abs(Freq) > 0.5 & abs(Freq) < 1.0)
```

```
##      Var1 Var2      Freq
## 62   X4.2 X4.1 -0.9920331
## 80   X4.1 X4.2 -0.9920331
## 113  X14  X4.3  0.5072609
## 190    Y   X8  0.7204068
## 202  X10   X9  0.5714981
## 220   X9  X10  0.5714981
## 262 X12.2 X12.1 -0.9305054
## 280 X12.1 X12.2 -0.9305054
## 329  X4.3  X14  0.5072609
## 352   X8    Y  0.7204068
```

Cabe destacar la alta correlación negativa entre X4.2 y X4.1, obviamente esto se debe a que es una variable categórica que no puede tomar ambos valores a la vez, y el tercer valor apenas está presente. Por tanto es un indicativo de que una de las variables se puede eliminar por “repetir información” y estar muy ligadas. Lo mismo ocurre con X12.2 y X12.1.

```
australian = australian[, -which(names(australian) %in% c("X4.1", "X12.1"))]
```

Trabajo Final Regresión

Modelos de regresión lineal simple

El primer objetivo del trabajo final de regresión es generar un modelo lineal con cada una de las variables de entrada del dataset. De esta forma se puede obtener de una manera sencilla la información sobre qué variable es mejor para un modelo lineal, es decir, qué variable es más representativa de la de salida.

Para realizar los modelos de regresión lineal se va a utilizar la función `lm` que ya viene entre las funciones base de R. Es necesario indicar cuál es la variable de salida y cuál (o cuáles) son las que se van a utilizar para construir el modelo.

Se van a analizar todas las variables X con respecto a la variable de salida Y. Se construye un modelo con cada una de estas variables, y con la función `summary` se obtiene una información más detallada del modelo resultante.

```
lmsimple1 = lm(Y~X1, data=friedman)
lmsimple2 = lm(Y~X2, data=friedman)
lmsimple3 = lm(Y~X3, data=friedman)
lmsimple4 = lm(Y~X4, data=friedman)
lmsimple5 = lm(Y~X5, data=friedman)
```

```
summary(lmsimple1)
```

```
##
## Call:
## lm(formula = Y ~ X1, data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.1161  -3.3974   0.0156   3.3261  13.8565
##
```

```
## Coefficients:
##           Estimate Std. Error t value Pr(>|t|)
## (Intercept)  10.6586    0.2708   39.37  <2e-16 ***
## X1           7.7211    0.4637   16.65  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.674 on 1198 degrees of freedom
## Multiple R-squared:  0.1879, Adjusted R-squared:  0.1872
## F-statistic: 277.2 on 1 and 1198 DF,  p-value: < 2.2e-16
```

```
summary(lmsimple2)
```

```
##
## Call:
## lm(formula = Y ~ X2, data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -16.0532  -3.3110  -0.1019   3.5042  12.8661
##
## Coefficients:
##           Estimate Std. Error t value Pr(>|t|)
## (Intercept)  11.2915    0.2744   41.15  <2e-16 ***
## X2           6.5515    0.4732   13.84  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.816 on 1198 degrees of freedom
## Multiple R-squared:  0.1379, Adjusted R-squared:  0.1372
## F-statistic: 191.7 on 1 and 1198 DF,  p-value: < 2.2e-16
```

```
summary(lmsimple3)
```

```
##
## Call:
## lm(formula = Y ~ X3, data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.9798  -3.6750   0.1493   3.8077  13.7227
##
## Coefficients:
##           Estimate Std. Error t value Pr(>|t|)
## (Intercept)  14.2490    0.2981  47.805  <2e-16 ***
## X3           0.6367    0.5161   1.234   0.218
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 5.183 on 1198 degrees of freedom
## Multiple R-squared:  0.001269, Adjusted R-squared:  0.0004351
## F-statistic: 1.522 on 1 and 1198 DF,  p-value: 0.2176
```

```
summary(lmsimple4)
```

```
##
```



```
## Call:
## lm(formula = Y ~ X4, data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.3664  -3.1954  -0.0698   3.0166  10.5726
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   8.8149     0.2432   36.25  <2e-16 ***
## X4            11.2296     0.4151   27.05  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.087 on 1198 degrees of freedom
## Multiple R-squared:  0.3792, Adjusted R-squared:  0.3787
## F-statistic: 731.7 on 1 and 1198 DF,  p-value: < 2.2e-16
```

```
summary(lmsimple5)
```

```
##
## Call:
## lm(formula = Y ~ X5, data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.2433  -3.8083   0.1361   3.6498  13.2920
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  12.0471     0.2918  41.292  <2e-16 ***
## X5           5.1132     0.5150   9.929  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.986 on 1198 degrees of freedom
## Multiple R-squared:  0.07604, Adjusted R-squared:  0.07527
## F-statistic: 98.59 on 1 and 1198 DF,  p-value: < 2.2e-16
```

De los resultados anteriores se pueden extraer varias afirmaciones. En primer lugar, el p-value de los modelos de X1, X2, X4 y X5 es muy pequeño ($< 2.2e-16$) por lo que se puede afirmar con una confianza casi cercana al 100% que existe algún tipo de dependencia lineal entre dichas variables y la variable de salida. En el caso de la variable X3, sin embargo, el p-value es muy alto (> 0.2) por lo que no se puede afirmar lo anterior con suficiente confianza, es decir, es una variable que no se utilizará generalmente para construir un modelo de regresión lineal.

De entre los modelos aceptables, como era de esperar el mejor es el que utiliza X4, la variable que más correlación mantiene con la salida, a pesar de ser “tan sólo” un valor de R-cuadrado de 0.379. El R-squared o R-cuadrado indica cómo de bueno es el modelo de regresión lineal. Cuanto más próximo a 1 más acertado es, y cuanto más próximo a 0 al contrario. Es por esto que a pesar de que el valor de X4 no es muy óptimo, es el que más cerca se encuentra del 1, y por tanto el mejor de las variables estudiadas.

Modelos de regresión lineal múltiple

A continuación se va a intentar llegar a un modelo de regresión lineal múltiple que obtenga mejores resultados que el modelo anterior. Para ello se construirá un modelo con todas las variables de entrada posibles y se eliminarán las menos prometedoras para encontrar el mejor balance entre complejidad y acierto.

Lo primero es construir el modelo con todas las variables, como se ha indicado anteriormente.

```
lmmultiple1 = lm(Y~., data=friedman)
summary(lmmultiple1)

##
## Call:
## lm(formula = Y ~ ., data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.4519  -1.5973   0.0415   1.7213   6.8138
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.024019   0.303396   0.079   0.937
## X1           6.932948   0.268826  25.790 <2e-16 ***
## X2           6.284951   0.265372  23.684 <2e-16 ***
## X3           0.005065   0.268706   0.019   0.985
## X4          10.465240   0.275546  37.980 <2e-16 ***
## X5           5.130121   0.278745  18.404 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.696 on 1194 degrees of freedom
## Multiple R-squared:  0.7307, Adjusted R-squared:  0.7296
## F-statistic: 648 on 5 and 1194 DF, p-value: < 2.2e-16
```

Se puede observar que este modelo, siendo más complejo, es también sustancialmente mejor que cualquier modelo de regresión lineal simple. Este modelo alcanza un valor de R-cuadrado ajustado de casi 0.73, mientras que el mejor de los anteriores tan sólo llegaba a 0.379. Sin embargo, comparando los p-values de los diferentes atributos se puede observar que el de X3 es muy elevado, al igual que evaluándolos individualmente. Es por ello que el siguiente paso es eliminarlo del modelo.

```
lmmultiple2 = lm(Y~.-X3, data=friedman)
summary(lmmultiple2)

##
## Call:
## lm(formula = Y ~ . - X3, data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -12.4519  -1.5974   0.0413   1.7224   6.8164
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.02636    0.27682   0.095   0.924
## X1           6.93298    0.26871  25.801 <2e-16 ***
## X2           6.28499    0.26525  23.694 <2e-16 ***
```

```
## X4          10.46544    0.27523  38.025   <2e-16 ***
## X5           5.13024    0.27856  18.417   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.695 on 1195 degrees of freedom
## Multiple R-squared:  0.7307, Adjusted R-squared:  0.7298
## F-statistic: 810.7 on 4 and 1195 DF,  p-value: < 2.2e-16
```

Este modelo incluso da un R-cuadrado ajustado mayor que el que contenía todas las variables, por lo que es intrínsecamente mejor tanto en complejidad como en acierto. Ahora todas las variables tienen un p-value ínfimo por lo que no se puede elegir una clara que eliminar para seguir probando a hacer un modelo más simple. Por ello, se va a eliminar la que menor R-cuadrado tuviese en los modelos lineales simples. En este caso, X5 (0.07).

```
lmmultiple3 = lm(Y~.-X3-X5, data=friedman)
summary(lmmultiple3)
```

```
##
## Call:
## lm(formula = Y ~ . - X3 - X5, data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.1609  -1.9004  -0.0158   2.0840   7.7125
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.5241     0.2733   9.235   <2e-16 ***
## X1             7.0046     0.3043  23.018   <2e-16 ***
## X2             6.4117     0.3003  21.350   <2e-16 ***
## X4            10.3306     0.3116  33.152   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.052 on 1196 degrees of freedom
## Multiple R-squared:  0.6543, Adjusted R-squared:  0.6534
## F-statistic: 754.5 on 3 and 1196 DF,  p-value: < 2.2e-16
```

El modelo resultante desciende su valor de R-cuadrado ajustado hasta 0.653, casi un 0.1 con respecto al modelo anterior. Teniendo esto en cuenta, se puede afirmar que la reducción de complejidad eliminando variables en este caso no compensa ya que se pierde demasiado acierto con respecto al modelo inmediatamente superior en complejidad. El mejor modelo lineal múltiple hasta el momento es `lmmultiple2`.

Una vez se ha llegado a esta conclusión, se pueden realizar transformaciones sobre las variables o interacciones entre ellas para tratar de obtener un modelo más preciso.

Interacciones

En primer lugar se realizarán algunos modelos basados en interacciones entre las variables más prometedoras del dataset, con el objetivo de buscar posibles combinaciones que arrojen un modelo más adecuado al problema.

Para empezar se pueden probar las dos variables que, individualmente, han resultado más adecuadas para realizar un modelo lineal.

```
lminteraccion1 = lm(Y~X4*X1, data=friedman)
summary(lminteraccion1)
```

```
##
## Call:
## lm(formula = Y ~ X4 * X1, data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.4758  -2.4193   0.0361   2.4213  10.3090
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    5.4721     0.4032  13.572  <2e-16 ***
## X4              11.0946     0.7245  15.313  <2e-16 ***
## X1               7.2666     0.7133  10.187  <2e-16 ***
## X4:X1          -0.9980     1.2459  -0.801    0.423
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.586 on 1196 degrees of freedom
## Multiple R-squared:  0.5228, Adjusted R-squared:  0.5216
## F-statistic: 436.7 on 3 and 1196 DF,  p-value: < 2.2e-16
```

El modelo tan sólo llega a un R-cuadrado de 0.52, y el p-value del producto $X4 \cdot X1$ no garantiza una alta confianza de que dicho valor resulte significativo para la predicción de la variable de salida Y. El siguiente modelo será, por tanto, el que combine lo anterior con una nueva variable, la X2, que es la siguiente en la lista de prometedoras.

```
lminteraccion2 = lm(Y~X4*X1*X2, data=friedman)
summary(lminteraccion2)
```

```
##
## Call:
## lm(formula = Y ~ X4 * X1 * X2, data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.4444  -1.8606  -0.0631   2.1395   7.8986
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    3.5243     0.6886   5.118 3.60e-07 ***
## X4              9.4596     1.2620   7.496 1.28e-13 ***
## X1              4.3620     1.2182   3.581 0.000356 ***
## X2              3.8700     1.1550   3.351 0.000832 ***
## X4:X1           2.8933     2.1588   1.340 0.180409
## X4:X2           2.9348     2.1101   1.391 0.164535
## X1:X2           6.5580     2.0850   3.145 0.001700 **
## X4:X1:X2       -8.3626     3.6596  -2.285 0.022481 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.041 on 1192 degrees of freedom
## Multiple R-squared:  0.658, Adjusted R-squared:  0.656
```

```
## F-statistic: 327.6 on 7 and 1192 DF, p-value: < 2.2e-16
```

Este modelo ya es sustancialmente mejor, llegando hasta un R-cuadrado ajustado de 0.656. La confianza de la combinación de X1, X2 y X4 es superior a un 97% por lo que no se puede rechazar la hipótesis de que tenga una correlación lineal con la variable de salida. A pesar de que haya valores más altos de p-value para interacciones que son un subconjunto de la interacción principal, es el p-value de la última el que indica si es confiable.

Dado que el modelo de regresión lineal múltiple que mejor resultado ha dado ha sido el que combinaba todas las variables menos X3, se puede añadir X5, la variable que falta, y comprobar cómo se integra en el modelo.

```
lminteraccion3 = lm(Y~X4*X1*X2*X5, data=friedman)
summary(lminteraccion3)
```

```
##
## Call:
## lm(formula = Y ~ X4 * X1 * X2 * X5, data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.8663  -1.6229   0.0125   1.7488   6.8136
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.2115     1.2777  -0.166  0.86853
## X4             10.6976     2.2618   4.730 2.52e-06 ***
## X1              6.9770     2.2622   3.084  0.00209 **
## X2              5.9794     2.1664   2.760  0.00587 **
## X5              7.6391     2.3618   3.234  0.00125 **
## X4:X1          1.0171     3.9296   0.259  0.79582
## X4:X2          0.7184     3.8699   0.186  0.85276
## X1:X2          2.5026     3.8327   0.653  0.51391
## X4:X5         -1.9897     4.3047  -0.462  0.64402
## X1:X5         -5.5109     4.0148  -1.373  0.17013
## X2:X5         -4.8001     3.9352  -1.220  0.22279
## X4:X1:X2      -6.3673     6.6718  -0.954  0.34010
## X4:X1:X5       3.5243     7.2060   0.489  0.62487
## X4:X2:X5       4.5470     7.2148   0.630  0.52866
## X1:X2:X5       8.9523     6.8265   1.311  0.18998
## X4:X1:X2:X5  -4.6139    12.2215  -0.378  0.70585
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.676 on 1184 degrees of freedom
## Multiple R-squared:  0.7369, Adjusted R-squared:  0.7336
## F-statistic: 221.1 on 15 and 1184 DF, p-value: < 2.2e-16
```

Este modelo supone un caso particular, y es que su acierto es superior a la de los modelos anteriores (R-cuadrado superior a 0.73), pero a su vez no refleja una confianza que garantice que la interacción estudiada se ajuste de forma lineal a la variable de salida. Por dicha razón, mientras se esté buscando un modelo de regresión lineal no se va a utilizar este. De nuevo, la falta de confianza viene dada por un p-valor muy alto (~0.706).

No linealidad

El siguiente paso es tratar de encontrar si alguna de las combinaciones que se han probado tienen un componente no lineal que pueda ayudar a ajustar mejor. En primer lugar se probará con X4, la mejor variable encontrada mediante modelos de regresión lineal simples, dado que ninguna variable aparenta seguir una distribución cuadrática con respecto a la variable de salida.

```
lmnolinealidad1 = lm(Y~X4+I(X4^2), data=friedman)
summary(lmnolinealidad1)
```

```
##
## Call:
## lm(formula = Y ~ X4 + I(X4^2), data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11.2754  -3.0983  -0.0216   2.9983  10.8607
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    8.3691     0.3642  22.979  <2e-16 ***
## X4             13.8767     1.6635   8.342  <2e-16 ***
## I(X4^2)        -2.6524     1.6143  -1.643   0.101
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.084 on 1197 degrees of freedom
## Multiple R-squared:  0.3806, Adjusted R-squared:  0.3795
## F-statistic: 367.7 on 2 and 1197 DF,  p-value: < 2.2e-16
```

El modelo es ligeramente mejor al lineal simple, pero como se puede observar en el resumen, el p-valor de la variable al cuadrado es sustancialmente mayor al de la variable simple. Por tanto no merece la pena investigar esa vertiente. Se puede probar con otra variable distinta, como X1.

```
lmnolinealidad2 = lm(Y~X1+I(X1^2), data=friedman)
summary(lmnolinealidad2)
```

```
##
## Call:
## lm(formula = Y ~ X1 + I(X1^2), data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.5809  -3.0448   0.0221   3.3661  12.7482
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    8.352     0.386   21.64  < 2e-16 ***
## X1             21.734     1.772   12.27  < 2e-16 ***
## I(X1^2)       -14.044     1.717   -8.18  7.19e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.55 on 1197 degrees of freedom
## Multiple R-squared:  0.2309, Adjusted R-squared:  0.2296
## F-statistic: 179.7 on 2 and 1197 DF,  p-value: < 2.2e-16
```

Dado que el p-valor de la variable cuadrática es pequeño, puede afirmarse con una alta confianza que $X1^2$ tiene dependencia lineal con Y, y por tanto puede ser útil en modelos más complejos. Se puede probar a añadirla al mejor modelo hasta ahora, el que utilizaba todas las variables menos X3.

```
lmnolinealidad3 = lm(Y~.-X3+I(X1^2), data=friedman)
summary(lmnolinealidad3)

##
## Call:
## lm(formula = Y ~ . - X3 + I(X1^2), data = friedman)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -10.6302  -1.6178  -0.1023   1.5998   8.2333
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.9294     0.3007  -6.417   2e-10 ***
## X1             19.2436     0.9885  19.468  <2e-16 ***
## X2              6.4223     0.2489   25.805  <2e-16 ***
## X4             10.2530     0.2585   39.659  <2e-16 ***
## X5              5.0524     0.2612   19.343  <2e-16 ***
## I(X1^2)       -12.3129     0.9560  -12.879  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.526 on 1194 degrees of freedom
## Multiple R-squared:  0.7636, Adjusted R-squared:  0.7626
## F-statistic: 771.2 on 5 and 1194 DF,  p-value: < 2.2e-16
```

Se ha encontrado un modelo en el que una transformación no lineal ha aportado una solución considerablemente mejor a la anterior. El R-cuadrado ajustado ha pasado de 0.72 a 0.76, y el p-valor de todas las variables que componen el modelo asegura con confianza su correlación lineal con la variable Y. Por tanto este modelo supera a `lmmultiple2` como mejor modelo encontrado.

Algoritmo k-NN para regresión no paramétrica

Para el siguiente apartado del trabajo final de regresión se va a realizar un estudio del dataset con el algoritmo k-NN. Se probarán distintas configuraciones para encontrar la mejor combinación de variables en el caso de dicho algoritmo.

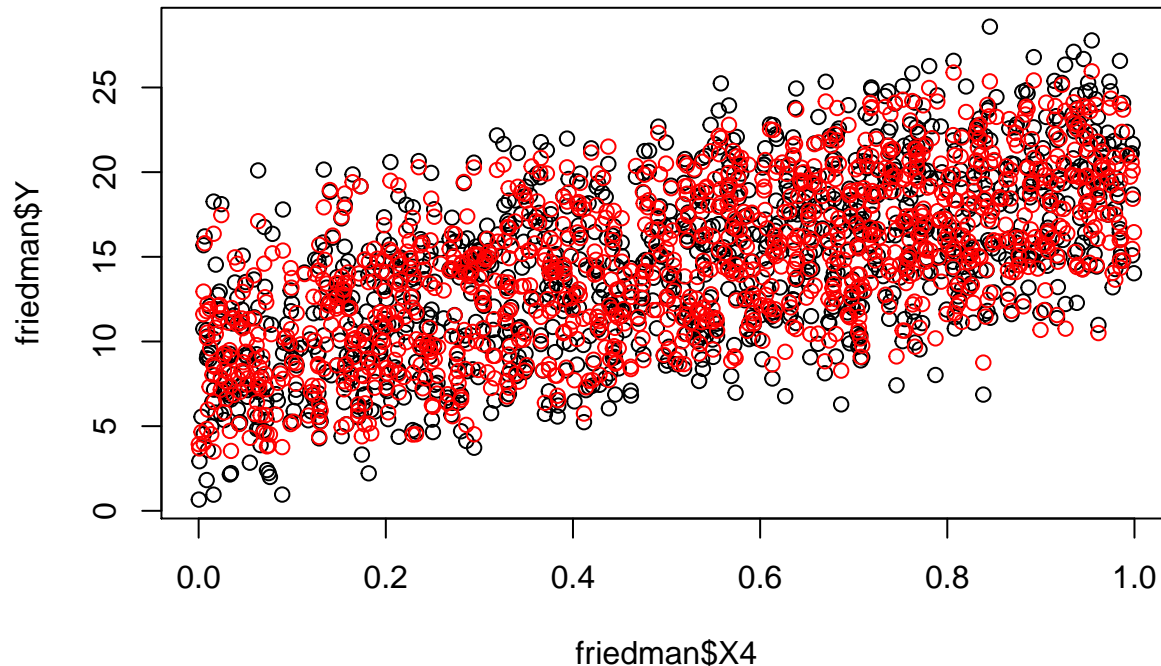
Cabe destacar que los datos utilizados por k-NN deben estar normalizados, para que las variables con valores de mayor orden no tengan más influencia en el cálculo de la distancia que aquellas con valores de menor orden. En el caso de este dataset, las variables de entrada tienen los valores entre 0 y 1 por lo que no es necesario realizar el escalado. Aunque, por defecto, la normalización se hace gracias al parámetro `scale = TRUE` de la función `kknn`. Para utilizar dicha función es necesario cargar el paquete homónimo, como ya se hizo al comienzo del trabajo.

En primer lugar se va a partir del modelo que contiene todas las variables. Los valores por defecto que se utilizan son `k=7`, y el kernel óptimo. Se utiliza el mismo conjunto de datos (el dataset entero) tanto para train como para test en este caso, para obtener un único valor que ejemplifique cómo se comporta el modelo para train.

```
fitknn1 = kknn(Y~., friedman, friedman)
```

A continuación se muestran los valores reales de Y frente a los que k-NN ha predicho, para comprobar de forma visual cuánto se parece la predicción a la realidad.

```
plot(friedman$Y~friedman$X4)
points(friedman$X4, fitknn1$fitted.values, col="red")
```



De esta gráfica sólo se puede intuir que el modelo construido con k-NN calcula unos valores de Y que siguen una distribución similar a la de la variable real. Sin embargo, se necesita una medida objetiva que ayude a valorar estos modelos de forma más precisa. Para ello se utiliza la raíz del error cuadrático medio (Root-Mean-Square Error, o **RMSE**). Se puede calcular siguiendo la siguiente fórmula:

```
RMSE = function(fit, labels) {
  yprime = fit$fitted.values
  sqrt(sum((labels-yprime)^2)/length(yprime)) # RMSE
}

RMSE(fitknn1, friedman$Y)
```

```
## [1] 1.166675
```

Ahora que se ha implementado la función del cálculo del RMSE, se pueden generar nuevos modelos con el algoritmo k-NN y compararlos entre sí. Por ejemplo, el modelo que contiene todas las variables menos X3, que anteriormente se ha comprobado que no ha aportado nada.

```
fitknn2 = kknn(Y~.-X3, friedman, friedman)
RMSE(fitknn2, friedman$Y)
```

```
## [1] 1.507862
```

En este caso se puede comprobar que los mejores modelos para regresión lineal no tienen por qué ser los mejores para el algoritmo k-NN. Aun así, se puede probar la selección de características del mejor modelo de regresión lineal múltiple para k-NN y de esta forma valorar cómo de bien aproxima la variable de salida con respecto a los otros modelos de este algoritmo.

```
fitknn3 = kknn(Y~.-X3+I(X1^2), friedman, friedman)
RMSE(fitknn3, friedman$Y)
```



```
## [1] 1.494644
```

La adición de la variable X_1 cuadrática ha aportado al modelo, aunque sin embargo este sigue sin ser mejor que el que tiene todas las variables. Si la añadimos a dicho modelo se podrá comprobar si realmente es una buena característica a tener en cuenta.

```
fitknn4 = kknn(Y~.+I(X1^2), friedman, friedman)
RMSE(fitknn4, friedman$Y)
```

```
## [1] 1.162829
```

El modelo es mejor que el mejor obtenido previamente pero no es una mejora sustancial a cambio de aumentar la complejidad con una nueva variable. Se puede comprobar si eliminando alguna otra variable el RMSE disminuye (X_1 no puede ser eliminada porque es necesaria para el cálculo de X_1^2).

```
fitknn5 = kknn(Y~.-X2+I(X1^2), friedman, friedman)
fitknn6 = kknn(Y~.-X4+I(X1^2), friedman, friedman)
fitknn7 = kknn(Y~.-X5+I(X1^2), friedman, friedman)
RMSE(fitknn5, friedman$Y)
```

```
## [1] 2.111819
```

```
RMSE(fitknn6, friedman$Y)
```

```
## [1] 2.380083
```

```
RMSE(fitknn7, friedman$Y)
```

```
## [1] 1.447878
```

Los resultados reflejan que no merece la pena eliminar ninguna de las variables, y que el mejor modelo según el criterio seguido en el estudio sería el que contiene todas las variables, ya que añadir la variable X_1 cuadrática no aporta una suficiente mejora, y sin embargo sí añade complejidad al modelo.

Comparativa de algoritmos

En este último apartado se van a comparar los resultados de los dos modelos de regresión múltiple estudiados (regresión lineal múltiple y k-NN) para comprobar si existen diferencias significativas entre ambos, mediante tests de Wilcoxon, Friedman y Holm. Para los tests que soportan más de dos algoritmos como entrada, se añadirán también los resultados del algoritmo M5 como parte de la comparativa.

En este caso se van a comparar únicamente los modelos que contienen todas las variables, cómo punto de partida igualitario para ambos regresores. Llegados a este punto sí tiene sentido contar con un error de train y test, por lo que se van a utilizar las particiones ubicadas dentro de la carpeta del dataset para realizar validación cruzada. La medida de error será el error cuadrático medio o MSE, que viene determinado por la distancia que existe entre las variables de salida predichas y las verdaderas.

En primer lugar se realizará la validación cruzada y el estudio de la regresión lineal múltiple.

```
path = "./Datos/friedman/friedman"

run_lm_fold = function(i, x, tt = "test") {
  file = paste(x, "-5-", i, "tra.dat", sep="")
  x_tra = read.csv(file, comment.char="@")
  file = paste(x, "-5-", i, "tst.dat", sep="")
  x_tst = read.csv(file, comment.char="@")
  In = length(names(x_tra)) - 1
  names(x_tra)[1:In] = paste ("X", 1:In, sep="")
  names(x_tra)[In+1] = "Y"
```

```

names(x_tst)[1:In] = paste ("X", 1:In, sep="")
names(x_tst)[In+1] = "Y"

if (tt == "train") {
  test = x_tra
}
else {
  test = x_tst
}

fitMulti = lm(Y~., x_tra)
yprime = predict(fitMulti, test)
sum(abs(test$Y-yprime)^2)/length(yprime) # MSE
}

lmMSEtrain = mean(sapply(1:5, run_lm_fold, path, "train"))
lmMSEtest = mean(sapply(1:5, run_lm_fold, path, "test"))

lmMSEtrain

## [1] 7.229588
lmMSEtest

## [1] 7.298681

```

Como se puede observar en base a los resultados, el modelo de regresión lineal múltiple que contiene a todas las variables obtiene un error cuadrático medio rondando los 7.2, con una ligera mejora en train con respecto a test, como es lógico dado que son los datos en los que se ha basado para construir el modelo.

A continuación se prueba el mismo experimento con el algoritmo k-NN, para ver cómo se comparan sus errores cuadráticos medios con los de regresión lineal.

```

run_knn_fold = function(i, x, tt = "test") {
  file = paste(x, "-5-", i, "tra.dat", sep="")
  x_tra = read.csv(file, comment.char="@")
  file = paste(x, "-5-", i, "tst.dat", sep="")
  x_tst = read.csv(file, comment.char="@")
  In = length(names(x_tra)) - 1
  names(x_tra)[1:In] = paste ("X", 1:In, sep="")
  names(x_tra)[In+1] = "Y"
  names(x_tst)[1:In] = paste ("X", 1:In, sep="")
  names(x_tst)[In+1] = "Y"

  if (tt == "train") {
    test = x_tra
  }
  else {
    test = x_tst
  }

  fitMulti = kknm(Y~., x_tra, test)
  yprime = fitMulti$fitted.values
  sum(abs(test$Y-yprime)^2)/length(yprime) # MSE
}

```

```
knnMSEtrain = mean(sapply(1:5, run_knn_fold, path, "train"))
knnMSEtest = mean(sapply(1:5, run_knn_fold, path, "test"))
```

```
knnMSEtrain
```

```
## [1] 1.42313
```

```
knnMSEtest
```

```
## [1] 3.196096
```

Para el k-NN la diferencia entre los errores de train y test se hace más notable, aumentando desde un 1.423 hasta un 3.196. A pesar de la diferencia entre ambas pruebas, el MSE en test, que es el que realmente es significativo, sigue estando bastante por debajo con respecto a su análogo de regresión lineal. El modelo que contiene todas las variables se ajusta por tanto bastante mejor con el algoritmo k-NN que con regresión lineal.

Antes de comenzar con las comparativas entre los algoritmos, se han de leer las tablas con los errores medios tanto de train como de test para tener los datos disponibles para aplicar los tests estadísticos. Estas tablas deberán contener los nuevos valores obtenidos por regresión lineal y k-NN, sustituyendo a los que aparecían por defecto.

```
resultados = read.csv("Datos/regr_train_alumnos.csv")
tablatra = cbind(resultados[,2:dim(resultados)[2]])
colnames(tablatra) = names(resultados)[2:dim(resultados)[2]]
rownames(tablatra) = resultados[,1]
```

```
resultados = read.csv("Datos/regr_test_alumnos.csv")
tablatst = cbind(resultados[,2:dim(resultados)[2]])
colnames(tablatst) = names(resultados)[2:dim(resultados)[2]]
rownames(tablatst) = resultados[,1]
```

Wilcoxon

A continuación se va a ejecutar el test de Wilcoxon para comprobar si existen diferencias significativas entre ambos modelos. Dado que Wilcoxon trabaja con diferencias de error, en regresión hay que normalizarlo para que se obtengan valores útiles. Se toma como referencia el algoritmo que parece ser mejor (k-NN en este caso) y se compara con el otro (regresión lineal múltiple). Se suma 0.1 porque Wilcoxon no trabaja bien con valores iguales a 0.

```
difs = (tablatst[,1] - tablatst[,2]) / tablatst[,1]
wilc_1_2 = cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1),
                  ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) = c(colnames(tablatst)[1], colnames(tablatst)[2])
head(wilc_1_2)
```

```
##      out_test_lm out_test_kknn
## [1,]    0.1909091    0.1000000
## [2,]    0.1000000    1.0294118
## [3,]    0.1000000    0.4339071
## [4,]    0.1000000    0.3885965
## [5,]    0.1548506    0.1000000
## [6,]    0.1000000    0.3061057
```

Con esta tabla se puede entrar a evaluar ya el test de Wilcoxon como tal.

```
LMvsKNNtst = wilcox.test(wilc_1_2[,1], wilc_1_2[,2],
                        alternative = "two.sided", paired=TRUE)
```

```

Rmas = LMvsKNNtst$statistic
pvalue = LMvsKNNtst$p.value
KNNvsLMtst = wilcox.test(wilc_1_2[,2], wilc_1_2[,1],
                        alternative = "two.sided", paired=TRUE)
Rmenos = KNNvsLMtst$statistic
Rmas

```

```

## V
## 78

```

```
Rmenos
```

```

## V
## 93

```

```
pvalue
```

```
## [1] 0.7660294
```

Se puede considerar que no existen diferencias significativas entre ambos algoritmos. El grado de confianza que se tiene de que los algoritmos sean distintos es de apenas un 23.4%, por lo que no se puede validar dicha hipótesis.

También se puede evaluar la diferencia existente entre estos algoritmos para los resultados de train, con los valores recopilados anteriormente.

```

difs = (tablatra[,1] - tablatra[,2]) / tablatra[,1]
wilc_1_2 = cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1),
                 ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) = c(colnames(tablatra)[1], colnames(tablatra)[2])
head(wilc_1_2)

```

```

##      out_train_lm out_train_kknn
## [1,]          0.1      0.6394191
## [2,]          0.1      1.0629412
## [3,]          0.1      0.7873339
## [4,]          0.1      0.7709917
## [5,]          0.1      0.6490708
## [6,]          0.1      0.7765836

```

```

LMvsKNNtst = wilcox.test(wilc_1_2[,1], wilc_1_2[,2],
                        alternative = "two.sided", paired=TRUE)
Rmas = LMvsKNNtst$statistic
pvalue = LMvsKNNtst$p.value
KNNvsLMtst = wilcox.test(wilc_1_2[,2], wilc_1_2[,1],
                        alternative = "two.sided", paired=TRUE)
Rmenos = KNNvsLMtst$statistic
Rmas

```

```

## V
## 10

```

```
Rmenos
```

```

## V
## 161

```

```
pvalue
```

```
## [1] 0.000328064
```

Según el test de Wilcoxon, para el conjunto de datos de train sí que hay casi un 100% de confianza de que existan diferencias significativas entre ambos algoritmos.

Friedman

Para el test de Friedman se valora la posición (el ranking) de cada algoritmo en cada conjunto de datos, con lo que no es necesario normalizar nada. Como la tabla `tablatst` ya contiene los valores de los algoritmos anteriormente mencionados además del algoritmo M5, se puede utilizar directamente para evaluar el test. Eso sí, la función `friedman.test` recibe una matriz, por lo que sí que es necesario realizar el casting correspondiente de forma previa.

```
test_friedman_test = friedman.test(as.matrix(tablatst))
test_friedman_test
```

```
##
## Friedman rank sum test
##
## data:  as.matrix(tablatst)
## Friedman chi-squared = 8.4444, df = 2, p-value = 0.01467
```

Dado el p-valor tan bajo que se obtiene como resultado del test de Friedman, se puede afirmar con casi un 99% de confianza que al menos un par de algoritmos son diferenciables entre sí significativamente.

A pesar de que en train existía un valor muy alto de confianza de que regresión lineal y k-NN fueran significativamente distintos, se puede probar el test de Friedman para ver si corrobora este hecho.

```
test_friedman_train = friedman.test(as.matrix(tablatra))
test_friedman_train
```

```
##
## Friedman rank sum test
##
## data:  as.matrix(tablatra)
## Friedman chi-squared = 20.333, df = 2, p-value = 3.843e-05
```

De nuevo el valor es muy pequeño, reforzando la hipótesis obtenida a partir de Wilcoxon de que hay diferencias significativas entre algún par de algoritmos (según Wilcoxon, k-NN y regresión lineal).

Holm

Ahora que se tiene certeza de que existe algún par de algoritmos con diferencias significativas, para averiguar cuál o cuáles de estas combinaciones son las que cumplen esta hipótesis se realiza el test de Holm.

```
tam = dim(tablatst)
groups = rep(1:tam[2], each=tam[1])
pairwise.wilcox.test(as.matrix(tablatst),
                     groups, p.adjust = "holm", paired = TRUE)
```

```
##
## Pairwise comparisons using Wilcoxon signed rank test
##
## data:  as.matrix(tablatst) and groups
##
##      1      2
## 2 0.580 -
## 3 0.081 0.108
```

```
##
## P value adjustment method: holm
```

En base a los resultados obtenidos, se puede deducir que existen dos pares de algoritmos diferentes significativamente (p-valor rondando el 0.1 o menor). Sabiendo que, según el orden en el que aparecen en la tabla, el 1 representa regresión lineal, el 2 k-NN y el 3 M5, se obtiene que el algoritmo M5 tiene diferencias significativas con respecto tanto a regresión lineal (~92% de confianza) como a k-NN (~90% de confianza). Los otros dos algoritmos, según la tabla, no pueden ser considerados diferentes, como se había comprobado antes en el test de Wilcoxon. Por tanto, se puede afirmar que todas las diferencias significativas entre estos algoritmos son a favor de M5.

Para el conjunto de train, el test de Holm también revelará qué algoritmos son diferentes entre sí, y con qué confianza.

```
tam = dim(tablatra)
groups = rep(1:tam[2], each=tam[1])
pairwise.wilcox.test(as.matrix(tablatra),
                     groups, p.adjust = "holm", paired = TRUE)
```

```
##
## Pairwise comparisons using Wilcoxon signed rank test
##
## data: as.matrix(tablatra) and groups
##
##      1      2
## 2 0.0031 -
## 3 0.0032 0.0032
##
## P value adjustment method: holm
```

Los valores tan bajos en todos los pares indican que los tres algoritmos son mutuamente distintos entre sí cuando se trata del conjunto de train. Esto era de esperar dado que se ha visto que M5 es distinto al resto en test y que k-NN y regresión lineal tenían una altísima confianza de serlo.

Trabajo Final Clasificación

Algoritmo K-NN para clasificación

El primer objetivo del trabajo de la parte de clasificación es ejecutar el algoritmo k-NN con este conjunto de datos. Además, se pide que se estudie cuál es la mejor k , para con ello obtener el mejor modelo posible.

Creación de particiones

Lo primero que hay que hacer es separar el dataset en conjuntos de train y de test, y guardar sus etiquetas en una variable distinta. Estos subconjuntos se utilizarán para llevar a cabo las ejecuciones de los algoritmos. Se fija una semilla para que las ejecuciones posteriores siempre reflejen el mismo resultado.

```
set.seed(1010)
shuffle_ds = sample(dim(australian)[1])
eightypct = (dim(australian)[1] * 80) %/% 100
australian[,dim(australian)[2]] = as.factor(australian[,dim(australian)[2]])

head(australian)
```

```
##      X1      X2      X3 X4.2 X4.3      X5      X6      X7 X8 X9
## 1  1 0.2736921 0.043516233 1 0 0.2307692 0.375 0.109954908 0 0
## 2  0 0.2810588 0.000265806 1 0 0.5384615 0.375 0.011446410 0 0
## 3  0 0.3673367 0.006645149 0 0 0.2307692 0.375 0.008671523 0 0
## 4  0 0.2685729 0.004366812 0 0 0.3076923 0.250 0.000000000 1 1
## 5  1 0.2498439 0.031023353 1 0 0.3846154 0.375 0.013596948 1 1
## 6  0 0.1956549 0.022213784 1 0 0.5384615 0.875 0.001040583 1 1
##      X10 X11 X12.2 X12.3 X13      X14 Y
## 1 0.00000000 1 1 0 0.05 0.01212 0
## 2 0.00000000 0 1 0 0.08 0.00000 0
## 3 0.00000000 1 1 0 0.14 0.00000 0
## 4 0.16417910 1 1 0 0.00 0.00000 1
## 5 0.20895522 0 1 0 0.03 0.00158 1
## 6 0.02985075 0 1 0 0.05 0.00000 1

aust_train = australian[shuffle_ds[1:eightypct], 1:dim(australian)[2]-1]
aust_test = australian[shuffle_ds[(eightypct+1):dim(australian)[1]], 1:dim(australian)[2]-1]

aust_train_labels = australian[shuffle_ds[1:eightypct], dim(australian)[2]]
aust_test_labels = australian[shuffle_ds[(eightypct+1):dim(australian)[1]], dim(australian)[2]]
```

Mediante el paquete `caret` y sus funciones `train` y `predict` se pueden construir modelos y calcular sus valores predichos respectivamente. Se va a utilizar la función `train` con el método `knn` para calcular de forma más sencilla y directa la mejor k . Se utilizan únicamente valores impares para forzar el desempate entre clases.

```
knnModel = train(aust_train, aust_train_labels, method="knn",
  metric="Accuracy", tuneGrid = data.frame(.k=seq(1,15,2)))
knnModel
```

```
## k-Nearest Neighbors
##
## 552 samples
## 16 predictor
## 2 classes: '0', '1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 552, 552, 552, 552, 552, ...
## Resampling results across tuning parameters:
##
## k Accuracy Kappa
## 1 0.8008787 0.5927196
## 3 0.8141089 0.6198630
## 5 0.8280850 0.6496759
## 7 0.8426123 0.6799020
## 9 0.8523967 0.7000663
## 11 0.8541014 0.7038798
## 13 0.8527183 0.7013975
## 15 0.8515019 0.6987002
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 11.
```

Los resultados obtenidos reflejan que el modelo con $k=11$ es el que mejor acierto tiene. Sin embargo, en caso de que este produzca sobreaprendizaje, se van a estudiar los 3 mejores k para asegurar un resultado óptimo.

En primer lugar se construyen los modelos.

```
mejoresK = knnModel$results$k[order(knnModel$results$Accuracy, decreasing = TRUE)[1:3]]
mejoresK
```

```
## [1] 11 13 9
```

```
knnModel.1 = train(aust_train, aust_train_labels, method="knn",
  metric="Accuracy", tuneGrid = data.frame(.k=mejoresK[1]))

knnModel.2 = train(aust_train, aust_train_labels, method="knn",
  metric="Accuracy", tuneGrid = data.frame(.k=mejoresK[2]))

knnModel.3 = train(aust_train, aust_train_labels, method="knn",
  metric="Accuracy", tuneGrid = data.frame(.k=mejoresK[3]))
```

A continuación, se calculan las etiquetas predichas por estos modelos utilizando los datos de test.

```
knnPred.1 = predict(knnModel.1, aust_test)
knnPred.2 = predict(knnModel.2, aust_test)
knnPred.3 = predict(knnModel.3, aust_test)
```

Y una vez obtenidas las etiquetas, se comprueba el acierto logrado por cada uno de los modelos en el conjunto de test.

```
postResample(knnPred.1, aust_test_labels)
```

```
## Accuracy      Kappa
## 0.8478261 0.6953963
```

```
postResample(knnPred.2, aust_test_labels)
```

```
## Accuracy      Kappa
## 0.8478261 0.6956522
```

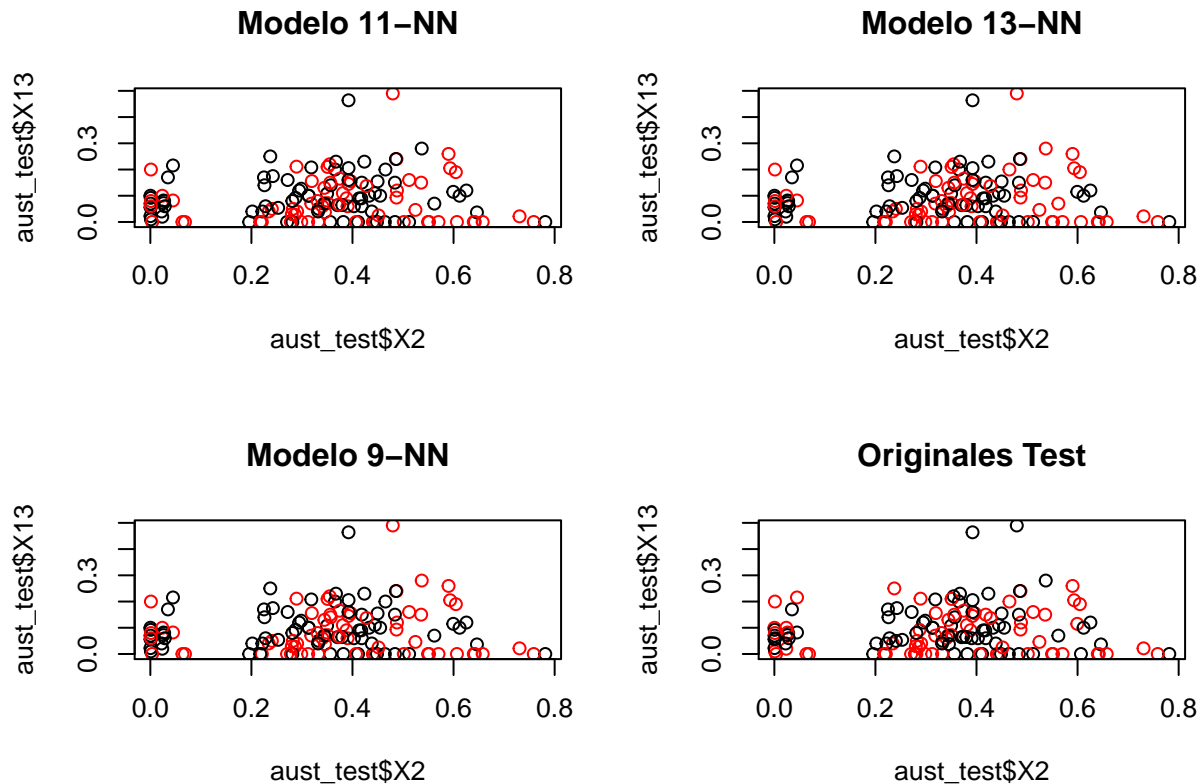
```
postResample(knnPred.3, aust_test_labels)
```

```
## Accuracy      Kappa
## 0.8333333 0.6665266
```

Como se puede observar, el mejor modelo, k=11, no sobreaprende, y obtiene un acierto similar al segundo, de un 84.78%. Además el descenso de acierto en test no es muy grande por lo que se trata de un modelo que se puede considerar aceptable.

Se pueden comparar los resultados de los modelos de un vistazo mostrándolos junto a las etiquetas reales, para valorar gráficamente su acierto.

```
par(mfrow=c(2,2))
plot(aust_test$X13~aust_test$X2,col=knnPred.1, main="Modelo 11-NN")
plot(aust_test$X13~aust_test$X2,col=knnPred.2, main="Modelo 13-NN")
plot(aust_test$X13~aust_test$X2,col=knnPred.3, main="Modelo 9-NN")
plot(aust_test$X13~aust_test$X2,col=aust_test_labels, main="Originales Test")
```

```
par(mfrow=c(1,1))
```

También se puede comprobar cómo se ajustan estos modelos a los datos de train para un estudio posterior.

```
knnPredTra.1 = predict(knnModel.1, aust_train)
knnPredTra.2 = predict(knnModel.2, aust_train)
knnPredTra.3 = predict(knnModel.3, aust_train)
```

```
postResample(knnPredTra.1, aust_train_labels)
```

```
## Accuracy      Kappa
## 0.8840580 0.7654962
```

```
postResample(knnPredTra.2, aust_train_labels)
```

```
## Accuracy      Kappa
## 0.8768116 0.7518150
```

```
postResample(knnPredTra.3, aust_train_labels)
```

```
## Accuracy      Kappa
## 0.8822464 0.7619490
```

Algoritmo LDA

El siguiente objetivo de la práctica es aplicar el algoritmo LDA al conjunto de datos de clasificación. LDA es un algoritmo que funciona mejor si los datos son normales y las varianzas similares. Se ha comprobado anteriormente en el apartado de análisis de datos de clasificación que, sin embargo, las variables correspondientes a este problema no siguen una distribución normal.

Antes se calcularon las varianzas de las variables sin normalizar. Las normalizadas son las siguientes:

```
sapply(australian[1:dim(australian)[2]-1], var)
```

```
##           X1           X2           X3           X4.2           X4.3           X5
## 0.218539787 0.037675432 0.013581830 0.182211144 0.002894344 0.080274790
##           X6           X7           X8           X9           X10           X11
## 0.062020677 0.009270161 0.249824362 0.245104226 0.005268030 0.248593845
##           X12.2          X12.3          X13           X14
## 0.085452557 0.011476410 0.007409704 0.002714517
```

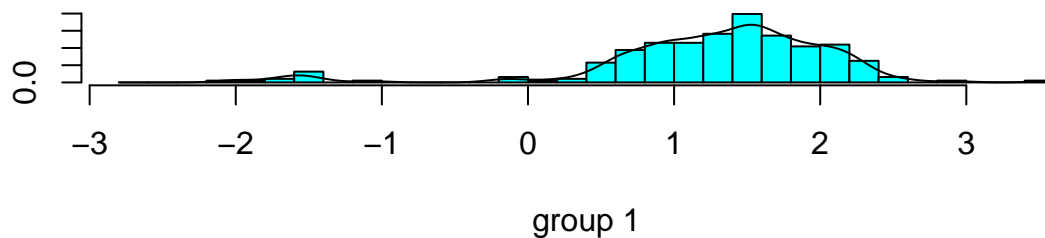
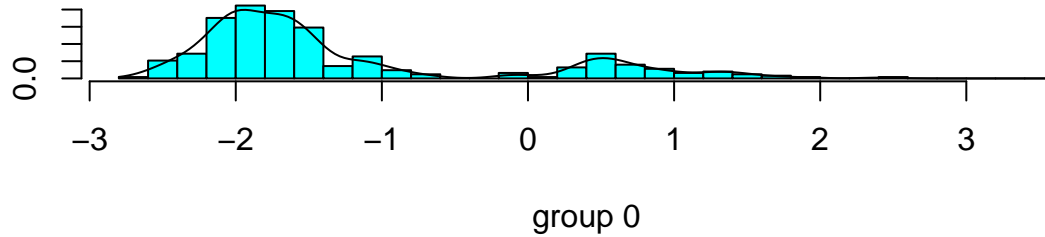
Se puede apreciar que las varianzas no son demasiado similares, por lo que las condiciones para aplicar el algoritmo LDA no son idóneas. Igualmente, como es el objetivo de la práctica, se procede a aplicarlo. Hay que utilizar la función `lda` del paquete MASS.

```
Y = aust_train_labels
lda.fit = lda(Y~., data=cbind(aust_train,Y))
lda.fit
```

```
## Call:
## lda(Y ~ ., data = cbind(aust_train, Y))
##
## Prior probabilities of groups:
##      0      1
## 0.567029 0.432971
##
## Group means:
##           X1           X2           X3           X4.2           X4.3           X5           X6
## 0 0.6741214 0.3183719 0.03841933 0.6932907 0.000000000 0.3949373 0.4073482
## 1 0.6861925 0.3515700 0.05683275 0.8619247 0.008368201 0.5925330 0.5329498
##           X7           X8           X9           X10           X11           X12.2
## 0 0.01678785 0.1948882 0.2172524 0.008726336 0.4345048 0.8849840
## 1 0.05290253 0.9330544 0.6694561 0.065072129 0.4895397 0.9414226
##           X12.3          X13           X14
## 0 0.009584665 0.09922843 0.00205377
## 1 0.016736402 0.08356904 0.02227962
##
## Coefficients of linear discriminants:
##           LD1
## X1      0.02760916
## X2      0.05102267
## X3      0.56462025
## X4.2    0.27040607
## X4.3    2.71309775
## X5      0.89798766
## X6      0.19540010
## X7      0.79131159
## X8      2.45626226
## X9      0.38156531
## X10     3.11213588
## X11     -0.05942872
## X12.2   0.12660185
## X12.3   1.64152301
## X13     -1.17703201
## X14     1.90501386
```

Se pueden visualizar los datos del modelo para ambas clases.

```
plot(lda.fit, type="both")
```



Ahora se puede realizar la predicción del test con el modelo que se ha obtenido.

```
lda.pred = predict(lda.fit, aust_test)
```

```
table(lda.pred$class, aust_test_labels)
```

```
##    aust_test_labels
##      0  1
##  0 55  7
##  1 15 61
```

```
ldaPred = mean(lda.pred$class==aust_test_labels)
ldaPred
```

```
## [1] 0.8405797
```

Esta matriz de confusión muestra los ejemplos que el modelo ha clasificado bien y mal respectivamente para cada clase. Como en el subconjunto de test hay más ejemplos de la clase 1, es normal que se haya equivocado en más que en lo que respecta a la clase 0. El modelo obtiene en test un acierto del 84.06%, algo menor que el que presentaba k-NN pero sin embargo nada desdeñable, sobre todo considerando el hecho de que las condiciones no son óptimas para este algoritmo.

Se puede estudiar el acierto con el propio conjunto de train para un apartado posterior.

```
lda.predTra = predict(lda.fit, aust_train)
```

```
table(lda.predTra$class, aust_train_labels)
```

```
##    aust_train_labels
##      0  1
##  0 253 14
##  1  60 225
```

```
ldaPredTra = mean(lda.predTra$class==aust_train_labels)
ldaPredTra
```

```
## [1] 0.865942
```

Algoritmo QDA

El tercer algoritmo que es necesario evaluar para la práctica es QDA. Este algoritmo funciona mejor si las varianzas entre las variables de unas clases con respecto a las otras son altas. Para comprobarlo se realizan dos subconjuntos del dataset.

```
aust_0 = australian[australian$Y == 0,][1:dim(australian)[2]-1]
aust_1 = australian[australian$Y == 1,][1:dim(australian)[2]-1]
```

```
sapply(aust_0, var)
```

```
##           X1           X2           X3           X4.2           X4.3
## 2.166828e-01 3.171867e-02 1.194917e-02 2.137301e-01 0.000000e+00
##           X5           X6           X7           X8           X9
## 7.048524e-02 5.580961e-02 3.583741e-03 1.610460e-01 1.745793e-01
##           X10          X11          X12.2          X12.3          X13
## 8.042296e-04 2.474950e-01 1.039602e-01 7.791888e-03 8.091535e-03
##           X14
## 4.510584e-05
```

```
sapply(aust_1, var)
```

```
##           X1           X2           X3           X4.2           X4.3           X5
## 0.221477082 0.044706816 0.015641170 0.130080262 0.006493368 0.067514527
##           X6           X7           X8           X9           X10          X11
## 0.061487074 0.015783041 0.069532265 0.218028145 0.008898520 0.250218220
##           X12.2          X12.3          X13          X14
## 0.061101531 0.016073748 0.006415997 0.005867198
```

Las varianzas de la mayoría de las variables son sustancialmente distintas, ya que aunque se trata de valores cercanos a 0, son de órdenes muy distintos entre sí. Por ese motivo se puede esperar que QDA se comporte de forma positiva.

Se ejecuta el algoritmo QDA utilizando la función `qda` del paquete `MASS`.

```
qda.fit = qda(Y~.-X4.3, data=cbind(aust_train,Y))
qda.fit
```

```
## Call:
## qda(Y ~ . - X4.3, data = cbind(aust_train, Y))
##
## Prior probabilities of groups:
##           0           1
## 0.567029 0.432971
##
## Group means:
##           X1           X2           X3           X4.2           X5           X6           X7
## 0 0.6741214 0.3183719 0.03841933 0.6932907 0.3949373 0.4073482 0.01678785
## 1 0.6861925 0.3515700 0.05683275 0.8619247 0.5925330 0.5329498 0.05290253
##           X8           X9           X10          X11          X12.2          X12.3
## 0 0.1948882 0.2172524 0.008726336 0.4345048 0.8849840 0.009584665
```

```
## 1 0.9330544 0.6694561 0.065072129 0.4895397 0.9414226 0.016736402
##          X13          X14
## 0 0.09922843 0.00205377
## 1 0.08356904 0.02227962
```

```
qda.pred = predict(qda.fit, aust_test)
```

```
table(qda.pred$class, aust_test_labels)
```

```
##      aust_test_labels
##      0  1
## 0 67 23
## 1  3 45
```

```
qdaPred = mean(qda.pred$class==aust_test_labels)
qdaPred
```

```
## [1] 0.8115942
```

El resultado de QDA es 81.16%, que pese a lo analizado anteriormente es el peor de los 3 resultados obtenidos. El mejor del estudio, por tanto, parece ser el modelo de 11-NN que se ha estudiado en primer lugar.

Para terminar QDA se puede valorar el acierto en el propio conjunto de train.

```
qda.predTra = predict(qda.fit, aust_train)
```

```
table(qda.predTra$class, aust_train_labels)
```

```
##      aust_train_labels
##      0  1
## 0 296 86
## 1  17 153
```

```
qdaPredTra = mean(qda.predTra$class==aust_train_labels)
qdaPredTra
```

```
## [1] 0.8134058
```

Comparativa de algoritmos

Una vez concluidos los apartados de cada algoritmo se han obtenido los resultados del estudio, el valor de acierto (en %) de cada algoritmo para el dataset *Australian*. Ahora se pueden estudiar las diferencias entre algoritmos mediante los tests de Wilcoxon, Friedman y Holm.

Wilcoxon

En primer lugar hay que leer las tablas de clasificación, para poder realizar los cálculos adecuados para este caso. Estas tablas tienen los valores recogidos por los algoritmos de este estudio.

```
resultados = read.csv("Datos/clasif_train_alumnos.csv")
tablatra = cbind(resultados[,2:dim(resultados)[2]])
colnames(tablatra) = names(resultados)[2:dim(resultados)[2]]
rownames(tablatra) = resultados[,1]
```

```
resultados = read.csv("Datos/clasif_test_alumnos.csv")
tablatst = cbind(resultados[,2:dim(resultados)[2]])
```

```
colnames(tablatst) = names(resultados)[2:dim(resultados)[2]]
rownames(tablatst) = resultados[,1]
```

Una vez leídos los datos, se puede realizar el test de Wilcoxon para cada dos pares de algoritmos. Los primeros en ser probados son k-NN y LDA para train.

```
difs = (tablatra[,1] - tablatra[,2]) / tablatra[,1]
wilc_1_2 = cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1),
                  ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) = c(colnames(tablatra)[1], colnames(tablatra)[2])
head(wilc_1_2)
```

```
##      out_train_knn out_train_lda
## [1,]          0.1    0.1021667
## [2,]          0.1    0.1204919
## [3,]          0.1    0.1309740
## [4,]          0.1    0.1514882
## [5,]          0.1    0.2511537
## [6,]          0.1    0.1353018
```

```
KNNvsLDAtst = wilcox.test(wilc_1_2[,1], wilc_1_2[,2],
                          alternative = "two.sided", paired=TRUE)
Rmas = KNNvsLDAtst$statistic
pvalue = KNNvsLDAtst$p.value
LDAvsKNNtst = wilcox.test(wilc_1_2[,2], wilc_1_2[,1],
                          alternative = "two.sided", paired=TRUE)
Rmenos = LDAvsKNNtst$statistic
Rmas
```

```
## V
## 80
```

```
Rmenos
```

```
## V
## 130
```

```
pvalue
```

```
## [1] 0.3682766
```

Para train el p-valor es muy alto, con lo que la confianza resulta ser muy pequeña y no se puede asumir que sean significativamente distintos.

A continuación, los mismos algoritmos son probados para test.

```
difs = (tablatst[,1] - tablatst[,2]) / tablatst[,1]
wilc_1_2 = cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1),
                  ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) = c(colnames(tablatst)[1], colnames(tablatst)[2])
head(wilc_1_2)
```

```
##      out_test_knn out_test_lda
## [1,]    0.1000000    0.1307536
## [2,]    0.1000000    0.1085470
## [3,]    0.1000000    0.1443729
## [4,]    0.1000000    0.1040566
## [5,]    0.1000000    0.1655377
## [6,]    0.1026335    0.1000000
```

```
KNNvsLDAstst = wilcox.test(wilc_1_2[,1], wilc_1_2[,2],
                           alternative = "two.sided", paired=TRUE)
Rmas = KNNvsLDAstst$statistic
pvalue = KNNvsLDAstst$p.value
LDAvsKNNtst = wilcox.test(wilc_1_2[,2], wilc_1_2[,1],
                          alternative = "two.sided", paired=TRUE)
Rmenos = LDAvsKNNtst$statistic
Rmas
```

```
## V
## 111
```

```
Rmenos
```

```
## V
## 99
```

```
pvalue
```

```
## [1] 0.8408222
```

El valor tan elevado que se obtiene del test de Wilcoxon no permite afirmar con apenas confianza que se trate de dos algoritmos significativamente diferentes. Ahora se realiza el test con los algoritmos k-NN y QDA, primero para train.

```
difs = (tablatra[,1] - tablatra[,3]) / tablatra[,1]
wilc_1_2 = cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1),
                 ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) = c(colnames(tablatra)[1], colnames(tablatra)[3])
head(wilc_1_2)
```

```
##      out_train_knn out_train_qda
## [1,]      0.1000000      0.1163404
## [2,]      0.1000000      0.1799181
## [3,]      0.1105683      0.1000000
## [4,]      0.1000000      0.2293485
## [5,]      0.1000000      0.2385592
## [6,]      0.1000000      0.1292493
```

```
KNNvsQDAstst = wilcox.test(wilc_1_2[,1], wilc_1_2[,2],
                           alternative = "two.sided", paired=TRUE)
Rmas = KNNvsQDAstst$statistic
pvalue = KNNvsQDAstst$p.value
QDAvsKNNtst = wilcox.test(wilc_1_2[,2], wilc_1_2[,1],
                          alternative = "two.sided", paired=TRUE)
Rmenos = QDAvsKNNtst$statistic
Rmas
```

```
## V
## 123
```

```
Rmenos
```

```
## V
## 87
```

```
pvalue
```

```
## [1] 0.5216732
```

Para este par de algoritmos el p-valor se reduce pero sigue sin alcanzar valores interesantes de confianza. No se puede afirmar que ambos algoritmos tengan diferencias significativas.

Ahora se prueban de nuevo k-NN y QDA, esta vez para test.

```
difs = (tablatst[,1] - tablatst[,3]) / tablatst[,1]
wilc_1_2 = cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1),
                  ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) = c(colnames(tablatst)[1], colnames(tablatst)[3])
head(wilc_1_2)
```

```
##      out_test_knn out_test_qda
## [1,]  0.1000000  0.1956404
## [2,]  0.1000000  0.1427351
## [3,]  0.1158854  0.1000000
## [4,]  0.1000000  0.2273004
## [5,]  0.1000000  0.1505725
## [6,]  0.1068124  0.1000000
```

```
KNNvsQDATst = wilcox.test(wilc_1_2[,1], wilc_1_2[,2],
                           alternative = "two.sided", paired=TRUE)
Rmas = KNNvsQDATst$statistic
pvalue = KNNvsQDATst$p.value
QDAvsKNNtst = wilcox.test(wilc_1_2[,2], wilc_1_2[,1],
                           alternative = "two.sided", paired=TRUE)
Rmenos = QDAvsKNNtst$statistic
Rmas
```

```
## V
## 128
```

```
Rmenos
```

```
## V
## 82
```

```
pvalue
```

```
## [1] 0.4090977
```

Este valor es menor que el que tenían k-NN y LDA en test, pero la confianza de que los algoritmos sean significativamente diferentes sigue siendo baja, de un 60%, con lo que tampoco se puede afirmar en este caso. Por último, se realiza el test con LDA y QDA, primero para train.

```
difs = (tablatra[,2] - tablatra[,3]) / tablatra[,2]
wilc_1_2 = cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1),
                  ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) = c(colnames(tablatra)[2], colnames(tablatra)[3])
head(wilc_1_2)
```

```
##      out_train_lda out_train_qda
## [1,]  0.1000000  0.1142045
## [2,]  0.1000000  0.1606694
## [3,]  0.1428702  0.1000000
## [4,]  0.1000000  0.1820867
## [5,]  0.1148372  0.1000000
## [6,]  0.1062741  0.1000000
```

```
LDAvsQDATst = wilcox.test(wilc_1_2[,1], wilc_1_2[,2],
                           alternative = "two.sided", paired=TRUE)
```



```
Rmas = LDAvsQDATst$statistic
pvalue = LDAvsQDATst$p.value
QDAvsLDAtst = wilcox.test(wilc_1_2[,2], wilc_1_2[,1],
                           alternative = "two.sided", paired=TRUE)
Rmenos = QDAvsLDAtst$statistic
Rmas
```

```
## V
## 144
```

```
Rmenos
```

```
## V
## 66
```

```
pvalue
```

```
## [1] 0.1536465
```

Este valor es el más bajo que se ha encontrado hasta ahora, con un 85% de confianza podría estudiarse el caso de que ambos algoritmos fueran significativamente distintos en el conjunto de train. A pesar de todo, sigue siendo un p-valor demasiado alto. Por último, toca probar con test.

```
difs = (tablatst[,2] - tablatst[,3]) / tablatst[,2]
wilc_1_2 = cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1),
                  ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) = c(colnames(tablatst)[2], colnames(tablatst)[3])
head(wilc_1_2)
```

```
##      out_test_lda out_test_qda
## [1,] 0.1000000    0.1669456
## [2,] 0.1000000    0.1344828
## [3,] 0.1630563    0.1000000
## [4,] 0.1000000    0.2237458
## [5,] 0.1160149    0.1000000
## [6,] 0.1041679    0.1000000
```

```
LDAvsQDATst = wilcox.test(wilc_1_2[,1], wilc_1_2[,2],
                           alternative = "two.sided", paired=TRUE)
Rmas = LDAvsQDATst$statistic
pvalue = LDAvsQDATst$p.value
QDAvsLDAtst = wilcox.test(wilc_1_2[,2], wilc_1_2[,1],
                           alternative = "two.sided", paired=TRUE)
Rmenos = QDAvsLDAtst$statistic
Rmas
```

```
## V
## 116
```

```
Rmenos
```

```
## V
## 94
```

```
pvalue
```

```
## [1] 0.7011814
```

Como en algunos casos anteriores, el p-valor es muy elevado por lo que la confianza de que se trate de dos algoritmos con diferencias significativas entre sí es muy baja. Ninguno de los tres algoritmos ha resultado ser

diferente con respecto a otro según este Wilcoxon.

Friedman

Lo siguiente a probar en el estudio comparativo es el test de Friedman. Dicho test reflejará si existe al menos una pareja de algoritmos con diferencias significativas entre sí. A pesar de lo que ha dicho el test de Wilcoxon, hay que probarlo como objetivo de la práctica.

En primer lugar se estudiará el conjunto de train.

```
test_friedman_train = friedman.test(as.matrix(tablatra))
test_friedman_train
```

```
##
## Friedman rank sum test
##
## data:  as.matrix(tablatra)
## Friedman chi-squared = 1.3, df = 2, p-value = 0.522
```

El p-valor obtenido es bastante alto, dejando una confianza de que existan diferencias significativas en al menos un par de algoritmos en tan sólo un 48%, tal y como se veía en el test de Wilcoxon. A continuación se realiza la misma prueba en test.

```
test_friedman_test = friedman.test(as.matrix(tablatst))
test_friedman_test
```

```
##
## Friedman rank sum test
##
## data:  as.matrix(tablatst)
## Friedman chi-squared = 0.1, df = 2, p-value = 0.9512
```

Reafirmando lo que ocurría en el test de Wilcoxon, el p-valor tan alto que devuelve el test de Friedman indica que solamente habría un par de algoritmos significativamente distintos con apenas un 5% de confianza, por lo que prácticamente se descarta esta idea.

Holm

Por último, hay que evaluar la tabla anterior con el test de Holm. Este test dará una información más detallada para confirmar si existen o no parejas de algoritmos con diferencias significativas. A juzgar por los dos tests anteriores, todo apunta a que no las habrá.

Primero se comprueba el conjunto de train.

```
tam = dim(tablatra)
groups = rep(1:tam[2], each=tam[1])
pairwise.wilcox.test(as.matrix(tablatra),
                     groups, p.adjust = "holm", paired = TRUE)
```

```
##
## Pairwise comparisons using Wilcoxon signed rank test
##
## data:  as.matrix(tablatra) and groups
##
##      1      2
## 2 0.66 -
## 3 0.66 0.53
```

```
##  
## P value adjustment method: holm
```

Los p-valores son demasiado elevados para considerar que pueda haber diferencias significativas, la confianza no aumenta por encima del 47% para ningún par de algoritmos. Por ello, se puede descartar que haya un par de algoritmos distintos. Veamos qué ocurre en test.

```
tam = dim(tablatst)  
groups = rep(1:tam[2], each=tam[1])  
pairwise.wilcox.test(as.matrix(tablatst),  
                      groups, p.adjust = "holm", paired = TRUE)
```

```
##  
## Pairwise comparisons using Wilcoxon signed rank test  
##  
## data: as.matrix(tablatst) and groups  
##  
## 1 2  
## 2 1 -  
## 3 1 1  
##  
## P value adjustment method: holm
```

Como los p-value dan valores tan altos, la confianza de que haya diferencias significativas en cada caso es 0%, se puede negar con rotundidad este caso según el test de Holm.