

Introducción a la Ciencia de Datos: Trabajo Final

Juan Ramón Gómez Berzosa

12/12/2018

Introducción

Durante el desarrollo del trabajo trataremos tres apartados donde realizaremos las operaciones pertinentes para el preprocesamiento de los dos dataset que nos han sido asignados y aplicaremos las diferentes técnicas de regresión y clasificación que hemos visto para intentar obtener el mejor modelo que se ajuste a nuestros datos. Los dataset que nos han sido asignados son los siguientes:

- *Concrete*: Dataset sobre el que aplicaremos las diferentes técnicas de regresión.
- *Australian*: Dataset sobre el que aplicaremos las diferentes técnicas de clasificación.

En primer lugar importaremos todas las librerías que vamos a necesitar para ejecutar los comandos que necesitaremos a lo largo del análisis de nuestros dataset.

```
library(ISLR)
library(MASS)
require(kknn)

## Loading required package: kknn
require(caret)

## Loading required package: caret
## Loading required package: lattice
## Loading required package: ggplot2
##
## Attaching package: 'caret'
## The following object is masked from 'package:kknn':
## 
##     contr.dummy
library(klaR)
library(ggplot2)
```

Preprocesamiento

En la tarea de preprocesamiento cargaremos los dataset en R y los analizaremos con el fin de tratar datos perdidos, normalizar y preparar los datos para el problema, obteniendo información relevante a priori sobre las variables que componen el dataset para saber cuales nos serán de mayor utilidad para ajustar unos buenos modelos de regresión o clasificación.

Preprocesamiento: Regresión

###Preparación de los datos

En primer lugar empezaremos con el conjunto de regresión. El dataset que nos ha sido asignado es *concrete*, lo cargaremos y pondremos su nombre a cada una de sus variables.

```
datasetR <- read.csv("concrete/concrete.dat", comment.char="@", header = FALSE)
names(datasetR) <- c("Cement", "BlastFurnaceSlag", "FlyAsh",
                      "Water", "Superplasticizer", "CoarseAggregate",
                      "FineAggregate", "Age",
                      "ConcreteCompressiveStrength")
str(datasetR)

## 'data.frame': 1030 obs. of 9 variables:
## $ Cement : num 252 296 252 172 162 ...
## $ BlastFurnaceSlag : num 0 0 0 13.6 214 ...
## $ FlyAsh : num 0 0 98.8 172.4 164 ...
## $ Water : num 185 186 146 157 202 ...
## $ Superplasticizer : num 0 0 14.2 4.1 10 11.6 0 9.1 7.8 0 ...
## $ CoarseAggregate : num 1111 1091 988 1006 820 ...
## $ FineAggregate : num 784 769 889 856 680 ...
## $ Age : num 7 7 3 28 28 56 14 28 56 14 ...
## $ ConcreteCompressiveStrength: num 13.7 14.8 21.8 33.7 30.6 ...

summary(datasetR)

##      Cement      BlastFurnaceSlag      FlyAsh          Water
## Min.   :102.0   Min.   : 0.0   Min.   : 0.00   Min.   :121.8
## 1st Qu.:192.4   1st Qu.: 0.0   1st Qu.: 0.00   1st Qu.:164.9
## Median :272.9   Median :22.0   Median : 0.00   Median :185.0
## Mean   :281.2   Mean   :73.9   Mean   :54.19   Mean   :181.6
## 3rd Qu.:350.0   3rd Qu.:142.9   3rd Qu.:118.30   3rd Qu.:192.0
## Max.   :540.0   Max.   :359.4   Max.   :200.10   Max.   :247.0
##      Superplasticizer      CoarseAggregate      FineAggregate      Age
## Min.   : 0.000   Min.   :801.0   Min.   :594.0   Min.   : 1.00
## 1st Qu.: 0.000   1st Qu.:932.0   1st Qu.:731.0   1st Qu.: 7.00
## Median : 6.400   Median :968.0   Median :779.5   Median :28.00
## Mean   : 6.205   Mean   :972.9   Mean   :773.6   Mean   :45.66
## 3rd Qu.:10.200   3rd Qu.:1029.4   3rd Qu.:824.0   3rd Qu.:56.00
## Max.   :32.200   Max.   :1145.0   Max.   :992.6   Max.   :365.00
##      ConcreteCompressiveStrength
## Min.   : 2.33
## 1st Qu.:23.71
## Median :34.45
## Mean   :35.82
## 3rd Qu.:46.13
## Max.   :82.60
```

Como podemos observar tenemos un dataset con 1030 observaciones y 9 variables, de las cuales 8 variables son de entrada y una es de salida: “ConcreteCompressiveStrength”. A continuación haremos una breve descripción de sus variables:

- *Cement*: Cemento, variable real en el intervalo [102.0, 540.0].
- *BlastFurnaceSlag*: Escoria de alto horno, variable real en el intervalo [0.0, 359.4].
- *FlyAsh*: Cenizas volantes, variable real en el intervalo [0.0, 200.100006].
- *Water*: Agua, variable real en el intervalo [121.8, 247.0].
- *Superplasticizer*: Superplastificante, variable real en el intervalo [0.0, 32.200001].
- *CoarseAggregate*: Agregado grueso, variable real en el intervalo [801.0, 1145.0].
- *FineAggregate*: Agregado fino, variable real en el intervalo [594.0, 992.6]

- *Age*: Edad, variable entera en el intervalo [1, 365].
- *ConcreteCompressiveStrength*: Resistencia a la compresión, real [2.33, 82.6]

Todas las variables menos la edad son medidas de kg por m^3 que hay en una mezcla. La variable de salida medirá la capacidad que tiene una estructura para soportar cargas que tienden a reducir el tamaño.

En primer lugar es importante saber que todas las variables están en las mismas unidades, a continuación procederemos a comprobar si existen valores perdidos. En la página web de keel ya hemos podido comprobar que no los hay pero lo probaremos.

```
datasetR[is.na(datasetR)]
```

```
## numeric(0)
```

Como podemos comprobar no tenemos missing values lo cual nos ahorra ya un gran trabajo del preprocesamiento. Existen muchos campos con valor 0.0, pero en este caso no tienen porque tratarse como missing values ya que perfectamente puede darse que en la mezcla de un material no haya alguno de estos componentes (variables).

Además, como hemos comprobado en el análisis de las variables, no tenemos ninguna variable categórica, con lo cual podemos trabajar perfectamente con los datos para regresión sin tener que hacer la transformación de ninguna variable. A continuación comprobaremos si existen valores repetidos.

```
datasetR.cleaned <- unique(datasetR)
nrow(datasetR)
```

```
## [1] 1030
```

```
nrow(datasetR.cleaned)
```

```
## [1] 1005
```

Como hemos observado hemos encontrado un total de 25 casos repetidos, los cuales hemos eliminado. Ahora nuestro dataset tendrá un total de 1005 observaciones limpias de missing values y valores repetidos, así como todas preparadas para los posteriores algoritmos de regresión que utilizaremos. Posteriormente, procederemos a normalizar los datos una vez hayamos analizado algunos valores estos.

Análisis de los datos

En primer lugar haremos un análisis de cada variable, para ello calcularemos algunas medidas como son la tendencia central de los datos (media y mediana), algunas de las medidas de dispersión (desviación estándar y varianza) ya que las otras las hemos mostrado en el resumen inicial de los datos y otras medidas relativas estandar (cuartiles, iqr).

```
# Cálculo de la media
apply(datasetR.cleaned, 2, mean)
```

```
##              Cement          BlastFurnaceSlag
##      278.631343      72.043482
##              FlyAsh          Water
##      55.536318     182.075323
##          Superplasticizer CoarseAggregate
##      6.033234      974.376815
##          FineAggregate        Age
##      772.688258      45.856716
## ConcreteCompressiveStrength
##      35.250378
```

```
# Cálculo de la mediana
apply(datasetR.cleaned, 2, median)
```

```
##              Cement          BlastFurnaceSlag
```

```

##          265.0
##          FlyAsh
##          0.0
##          Superplasticizer
##          6.1
##          FineAggregate
##          780.0
## ConcreteCompressiveStrength
##          33.8

```

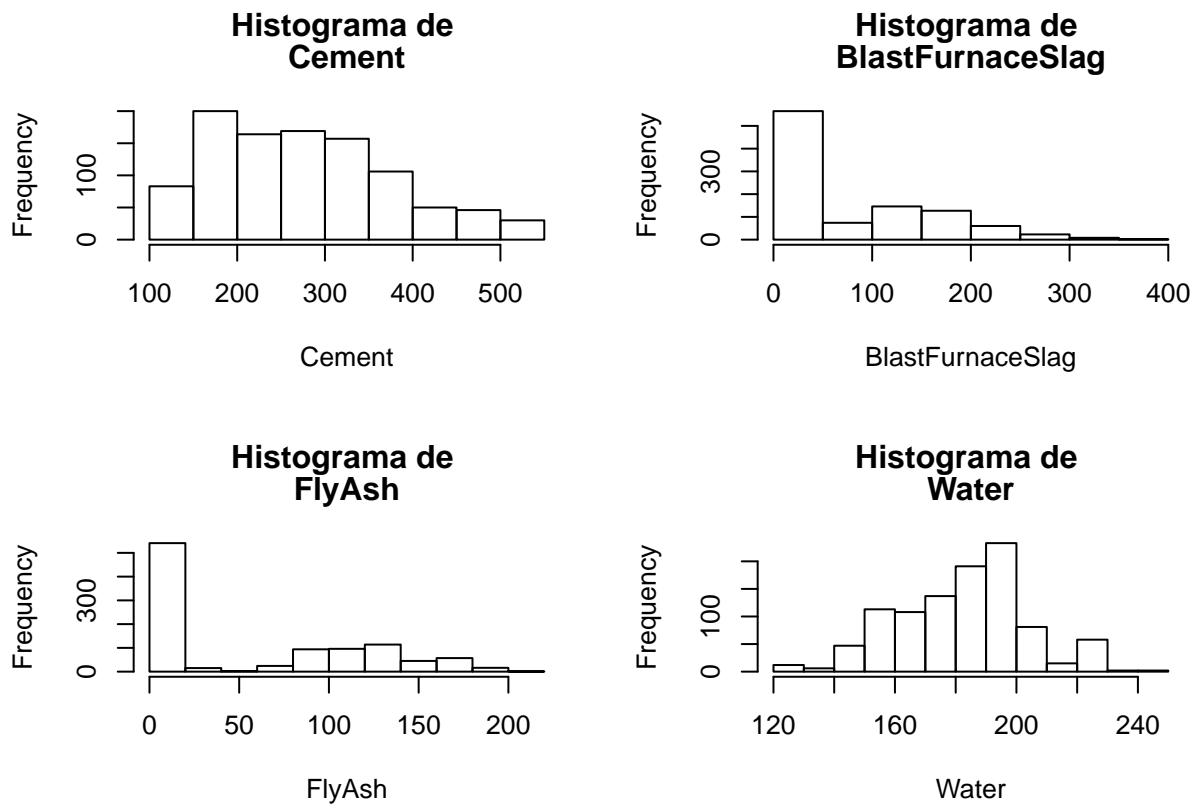
Ahora vamos a representar mediante histogramas las tendencias centrales de los datos.

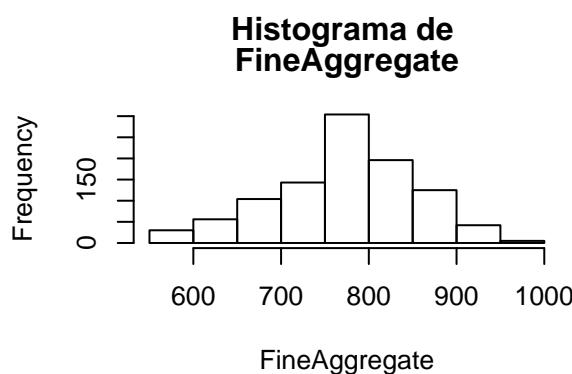
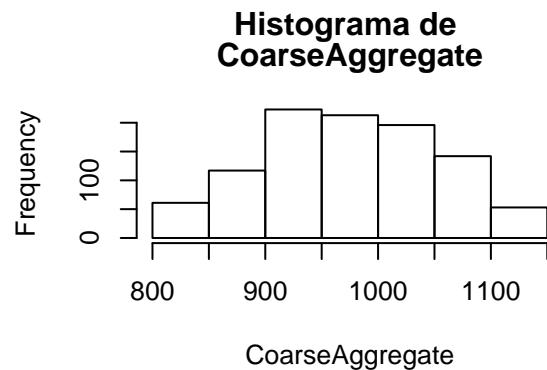
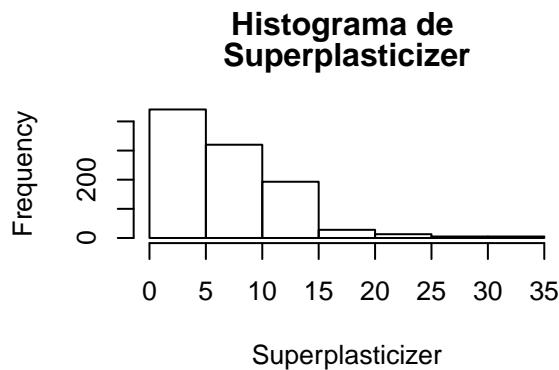
```

i = 1
histograma <- function (x) {
  hist(x, xlab = names(datos)[i], main = c("Histograma de ", names(datos)[i]) )
  assign("i", i + 1, envir = .GlobalEnv)
}

par(mfrow=c(2,2))
datos = datasetR.cleaned[,-9]
x <- sapply(datasetR.cleaned[,-9], histograma)

```





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

Como podemos observar, hay algunas variables donde los valores están más o menos repartidos como es el caso de CoarseAggregate, Cement o FineAggregate, con lo que podemos decir que están siempre presentes en bastante cantidad. Podemos también observar como hay muchos componentes que el valor que más se repite es el 0, como FlyAsh o BlastFurnaceSlag, lo que quiere decir que no se encuentran en la composición del material. Las cantidades de Superplasticizer también predomina el valor 0, aunque en este caso vemos que son decrecientes a partir de 0 hasta 15 kg por m³, siendo este prácticamente su valor más alto posible en una mezcla, ya que cantidades más altas no se dan en casi ninguna estructura. También podemos afirmar que la mayoría de las estructuras tienen entre 0 y 100 años.

```
# Cálculo de la desviación típica
apply(datasetR.cleaned, 2, sd)
```

```
##          Cement      BlastFurnaceSlag
##        104.344260       86.170807
##          FlyAsh         Water
##        64.207969      21.339334
##      Superplasticizer    CoarseAggregate
##        5.919967      77.579668
##      FineAggregate           Age
##        80.340435      63.734692
## ConcreteCompressiveStrength
##        16.284815
```

```
# Cálculo de la varianza
apply(datasetR.cleaned, 2, var)
```

```
##          Cement      BlastFurnaceSlag
##        10887.72469      7425.40792
```

```

##                  FlyAsh          Water
##        4122.66323      455.36717
##      Superplasticizer CoarseAggregate
##        35.04601       6018.60488
##      FineAggregate           Age
##        6454.58548      4062.11092
## ConcreteCompressiveStrength
##        265.19521

```

Podemos observar que la desviación típica es más alta en Cement, FineAggregate y CoarseAggregate, lo cual tiene sentido ya que eran los atributos en los que sus valores estaban más repartidos según la tendencia central de los datos. Otros como Superplasticizer tiene una desviación standar muy baja ya que no existe tanta dispersión de los datos ya que la mayor tendencia de los datos era entre el valor 0 y 15 aunque de forma decreciente y predominando muy por encima los valores cercanos a 0.

Cálculo de los cuartiles

```
apply(datasetR.cleaned, 2, quantile)
```

```

##      Cement BlastFurnaceSlag FlyAsh Water Superplasticizer CoarseAggregate
## 0%    102.0            0.0    0.0 121.8          0.0          801
## 25%   190.7            0.0    0.0 166.6          0.0          932
## 50%   265.0            20.0   0.0 185.7          6.1          968
## 75%   349.0           142.5  118.3 192.9         10.0         1031
## 100%  540.0           359.4  200.1 247.0         32.2         1145
##      FineAggregate Age ConcreteCompressiveStrength
## 0%      594.0     1           2.33
## 25%    724.3     7           23.52
## 50%    780.0    28           33.80
## 75%    822.2    56           44.87
## 100%   992.6   365           82.60

```

Cálculo del IQR

```
apply(datasetR.cleaned, 2, IQR)
```

```

##                  Cement          BlastFurnaceSlag
##        158.30000      142.50000
##                  FlyAsh          Water
##        118.30000      26.29999
##      Superplasticizer CoarseAggregate
##        10.00000      99.00000
##      FineAggregate           Age
##        97.90002      49.00000
## ConcreteCompressiveStrength
##        21.35000

```

A continuación vamos a observar como están dispuestas las variables en relación a una distribución normal.

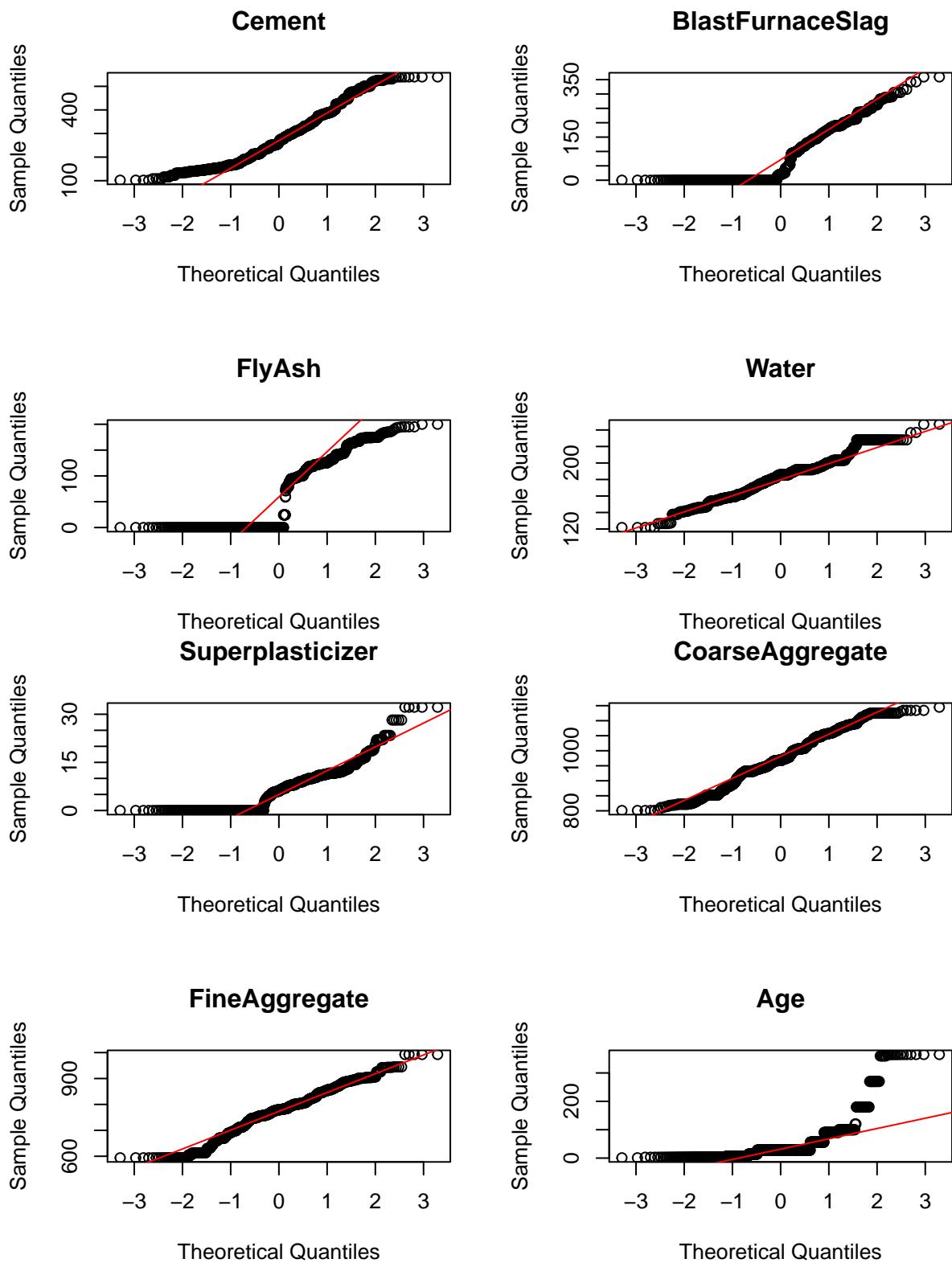
```

i = 1
qqplotAll <- function (x) {
  qqnorm(x, main = names(datos)[i])
  qqline(x, col = "Red")
  assign("i", i + 1, envir = .GlobalEnv)
}

par(mfrow=c(2,2))
datos = datasetR.cleaned[,-9]

```

```
x <- sapply(datos[,-9], qqplotAll)
```

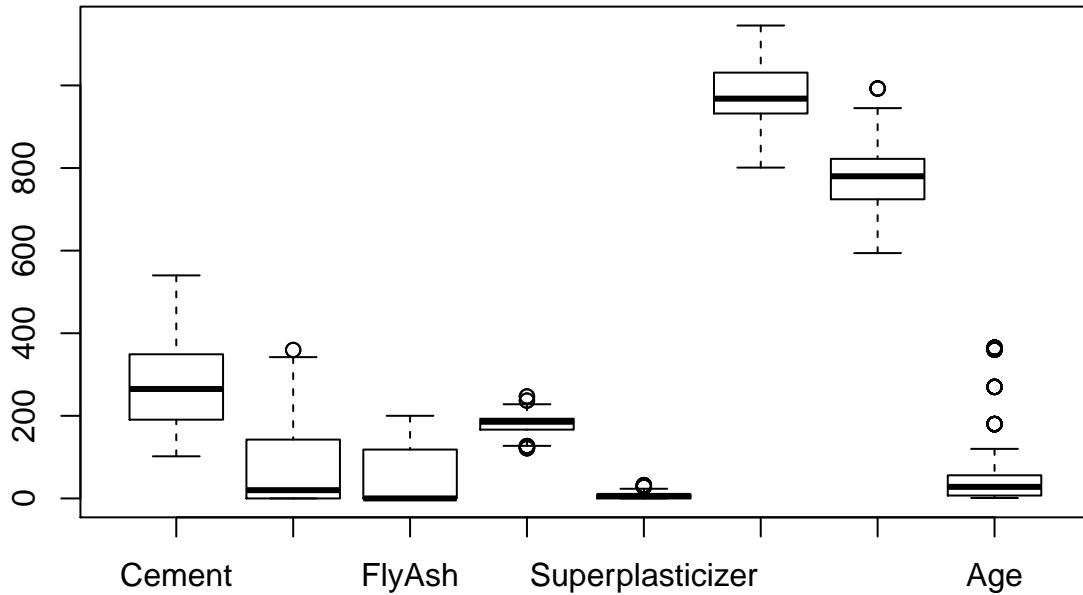


```
par(mfrow=c(1,1))
```

Las variables como Cement, Water, CoarseAggregate o Fine Aggregate siguen más o menos una distribución normal.

Ahora mostraremos graficamente mediante boxplot la dispersión de los datos y los cuartiles de los datos, así como sus outliers, a excepción de la variable de salida.

```
boxplot(datasetR.cleaned[,-9])
```

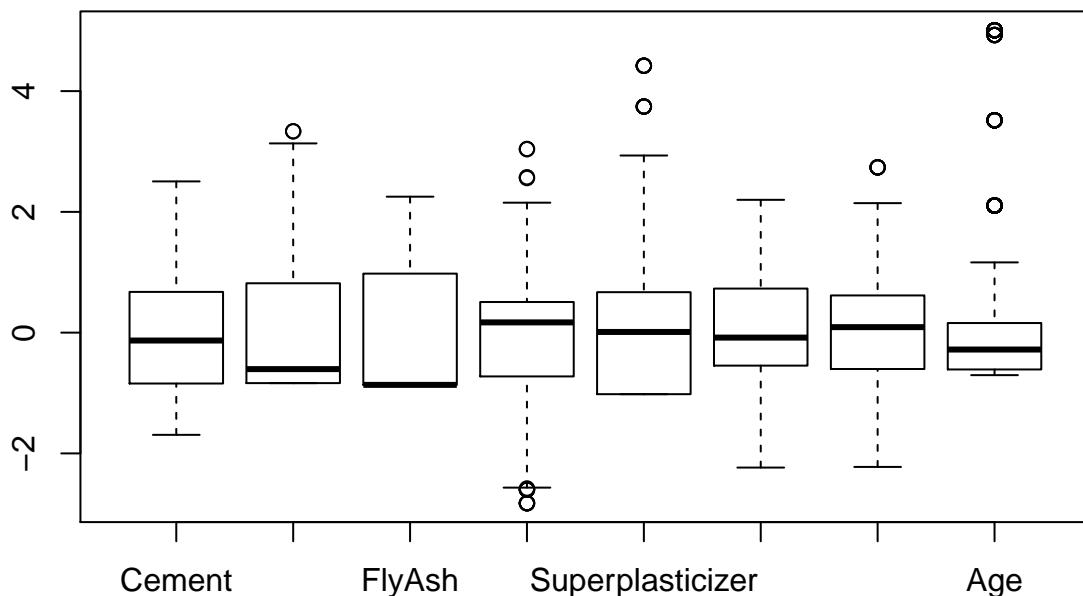


Como podemos observar los valores están muy dispares en los boxplot, así que procederemos a realizar una normalización de los datos. Usaremos el centrado y escalado.

```
datasetR.scaled = data.frame( scale(datasetR.cleaned))
```

Volveremos a graficar para comprobar los cambios.

```
boxplot(datasetR.scaled[,-9])
```



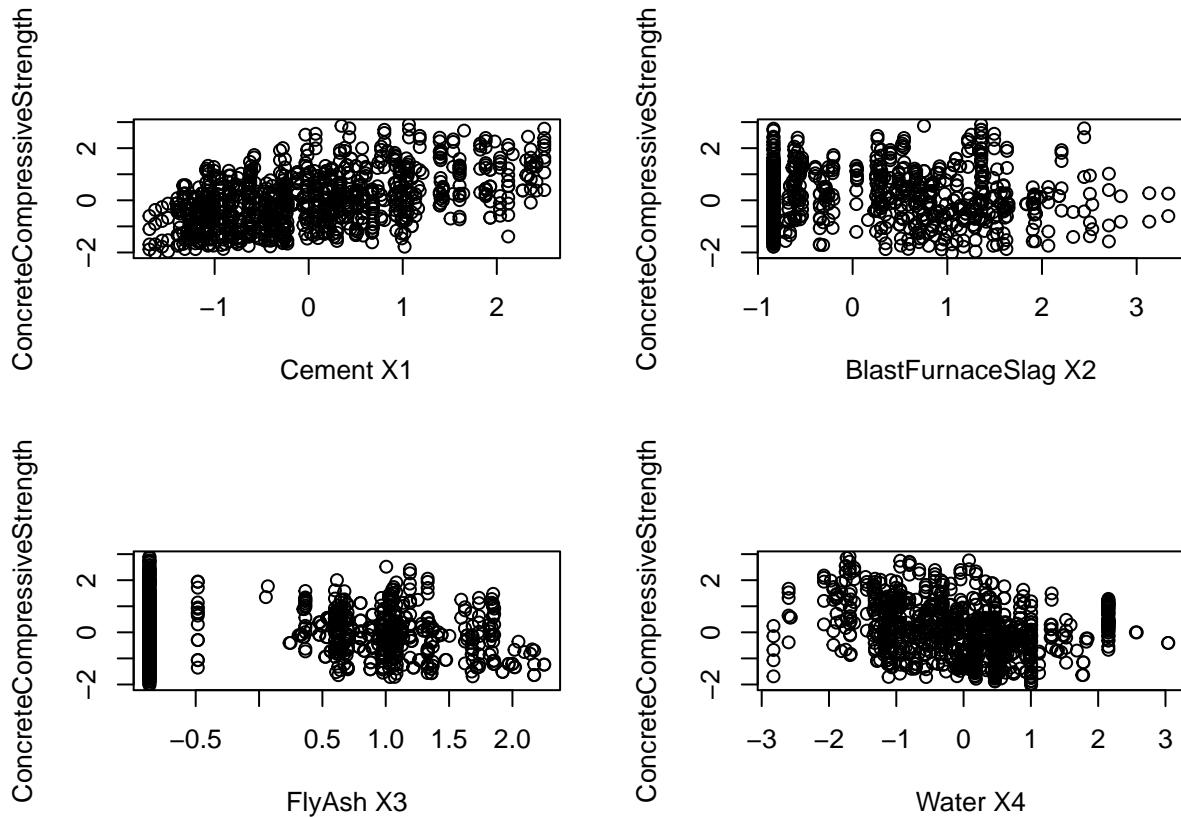
Ahora podemos observar los datos distribuidos de una forma más equiparable. Podemos observar como algunas variables tienen algunos outliers, sobre todo la variable age, lo cual tiene sentido ya que la mayor parte de las observaciones tenían una edad entre 0 y 100 y los demás eran muy poco frecuentes.

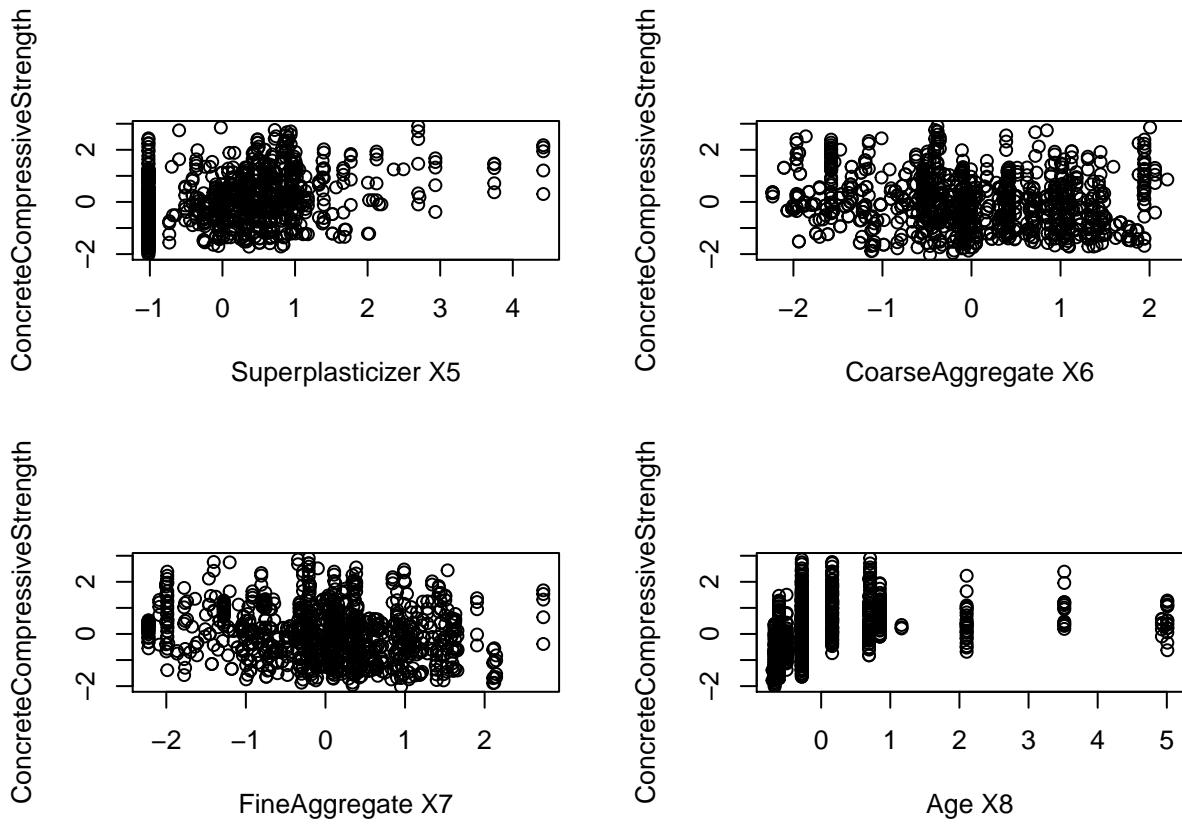
Una vez hecho esto procederemos a graficar utilizando scatterPlot para comparar pares de variables y ver cuales a priori serían buenas para ajustar un modelo de regresión, es decir, empezar a sacar nuestras primeras hipótesis. Para ello compararemos cada variable con la variable de salida, "ConcreteCompressiveStrength".

```
concrete = datasetR.scaled
```

```
plotY <- function (x,y) {
  plot(concrete[,y]~concrete[,x], xlab=paste(names(concrete)[x], " X", x, sep=""), ylab=names(concrete)[y])
}

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





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

A priori podemos afirmar que con la que mayor correlación tiene sería la variable "Cement", ya que aunque hay bastante dispersión de los datos sigue una distribución normal. También podríamos decir que una de las peores variables respecto a correlación sería la variable Water. Vamos a calcular la correlación de todas las variables.

```
cor(concrete)
```

| | Cement | BlastFurnaceSlag | FlyAsh |
|--------------------------------|-------------|------------------|-----------------|
| ## Cement | 1.0000000 | -0.30332393 | -0.38560976 |
| ## BlastFurnaceSlag | -0.30332393 | 1.0000000 | -0.31235235 |
| ## FlyAsh | -0.38560976 | -0.31235235 | 1.0000000 |
| ## Water | -0.05662463 | 0.13026209 | -0.28331363 |
| ## Superplasticizer | 0.06090565 | 0.01980023 | 0.41421275 |
| ## CoarseAggregate | -0.08620530 | -0.27755886 | -0.02646848 |
| ## FineAggregate | -0.24537544 | -0.28968492 | 0.09026179 |
| ## Age | 0.08634782 | -0.04275925 | -0.15894045 |
| ## ConcreteCompressiveStrength | 0.48828334 | 0.10337407 | -0.08064815 |
| | Water | Superplasticizer | CoarseAggregate |
| ## Cement | -0.05662463 | 0.06090565 | -0.086205304 |
| ## BlastFurnaceSlag | 0.13026209 | 0.01980023 | -0.277558860 |
| ## FlyAsh | -0.28331363 | 0.41421275 | -0.026468479 |
| ## Water | 1.0000000 | -0.64694605 | -0.212479710 |
| ## Superplasticizer | -0.64694605 | 1.0000000 | -0.241720523 |
| ## CoarseAggregate | -0.21247971 | -0.24172052 | 1.000000000 |
| ## FineAggregate | -0.44491470 | 0.20799294 | -0.162187008 |
| ## Age | 0.27928352 | -0.19407633 | -0.005263593 |
| ## ConcreteCompressiveStrength | -0.26962401 | 0.34420888 | -0.144717507 |

```

##                                     FineAggregate          Age
## Cement                           -0.24537544  0.086347818
## BlastFurnaceSlag                -0.28968492 -0.042759251
## FlyAsh                            0.09026179 -0.158940454
## Water                             -0.44491470  0.279283518
## Superplasticizer                  0.20799294 -0.194076333
## CoarseAggregate                   -0.16218701 -0.005263593
## FineAggregate                     1.00000000 -0.156572486
## Age                                -0.15657249  1.000000000
## ConcreteCompressiveStrength     -0.18644813  0.337366935
##                                     ConcreteCompressiveStrength
## Cement                           0.48828334
## BlastFurnaceSlag                 0.10337407
## FlyAsh                            -0.08064815
## Water                             -0.26962401
## Superplasticizer                  0.34420888
## CoarseAggregate                   -0.14471751
## FineAggregate                     -0.18644813
## Age                                0.33736693
## ConcreteCompressiveStrength     1.00000000

```

Nos fijaremos en la última columna que son los valores de correlación con respecto a la variable de salida. Como podemos comprobar, la variable “Cement” es la que mayor correlación tiene con la variable de salida, siguiéndola la variable “Superplasticizer” y “Age”. Las que peor correlación tienen con la variable de salida serían “Water”, “FineAggregate” y “CoarseAggregate”. Por tanto, para aprender nuestro regresor en principio probaremos con la variable “Cement” ya que es la que más correlación tiene con la variable de salida y alguna de las que descartaremos a priori será la variable “Water”.

En principio no vamos a generar variables nuevas ya que no tenemos variables categóricas para separar y crear nuevas variables y cada una de las variables que tenemos son bastante representativas por sí mismas, ya que solo tenemos la edad y componentes de la mezcla, siendo todos los componentes distintos.

Regresión Lineal

Regresión Lineal Simple

En primer lugar cabe destacar que nuestro dataset tiene un total de 8 variables de entrada, con lo cual nos quedaremos con las 5 que mejor correlación tienen con la variable de salida para generar un regresor lineal simple y ver cual nos da un mejor ajuste. Cabe destacar que según la página de keel, la variable de salida es una función no lineal entre la edad y la mezcla de los componentes, por lo tanto una regresión lineal simple no nos dará un buen ajuste a priori.

Atendiendo al estudio de correlación que hemos realizado anteriormente, nos quedaremos con:

- *Cement*
- *Superplasticizer*
- *Age*
- *BlastFurnaceSlag*
- *FlyAsh*

Empezaremos a aplicar el algoritmo de regresión lineal sobre cada una:

```
concrete = datasetR.scaled
```

```
# Obtenemos el modelo para cada variable
```

```

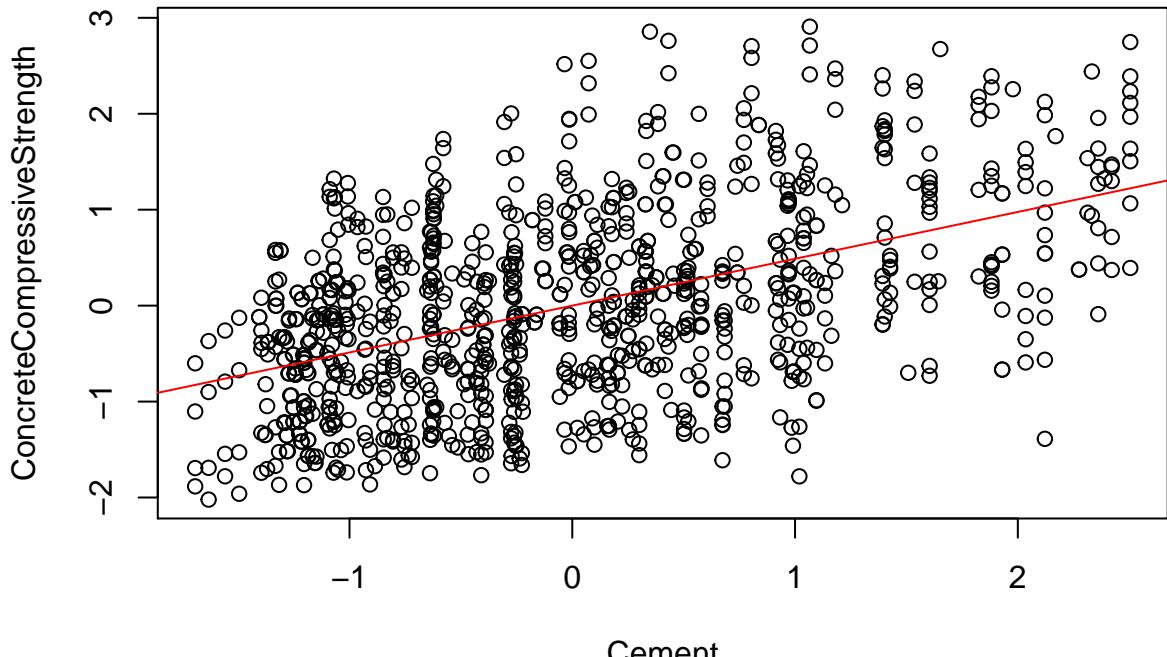
fit1=lm(ConcreteCompressiveStrength~Cement, data=concrete)
fit1

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ Cement, data = concrete)
##
## Coefficients:
## (Intercept)      Cement
## -8.629e-17    4.883e-01
#Visualizamos los resultados
summary(fit1)

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ Cement, data = concrete)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.42434 -0.67499 -0.03113  0.60472  2.68521
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -8.629e-17 2.754e-02    0.00     1
## Cement      4.883e-01 2.756e-02   17.72  <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8731 on 1003 degrees of freedom
## Multiple R-squared:  0.2384, Adjusted R-squared:  0.2377
## F-statistic:  314 on 1 and 1003 DF,  p-value: < 2.2e-16

par(mfrow=c(1,1))
plot(ConcreteCompressiveStrength~Cement,concrete)
abline(fit1,col="red")

```



```

confint(fit1)

##                   2.5 %      97.5 %
## (Intercept) -0.05404598 0.05404598
## Cement       0.43421045 0.54235623
# Obtenemos el modelo para cada variable
fit2=lm(ConcreteCompressiveStrength~Superplasticizer,data=concrete)
fit2

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ Superplasticizer,
##      data = concrete)
##
## Coefficients:
##             (Intercept)  Superplasticizer
##                 -2.182e-17    3.442e-01
#Visualizamos los resultados
summary(fit2)

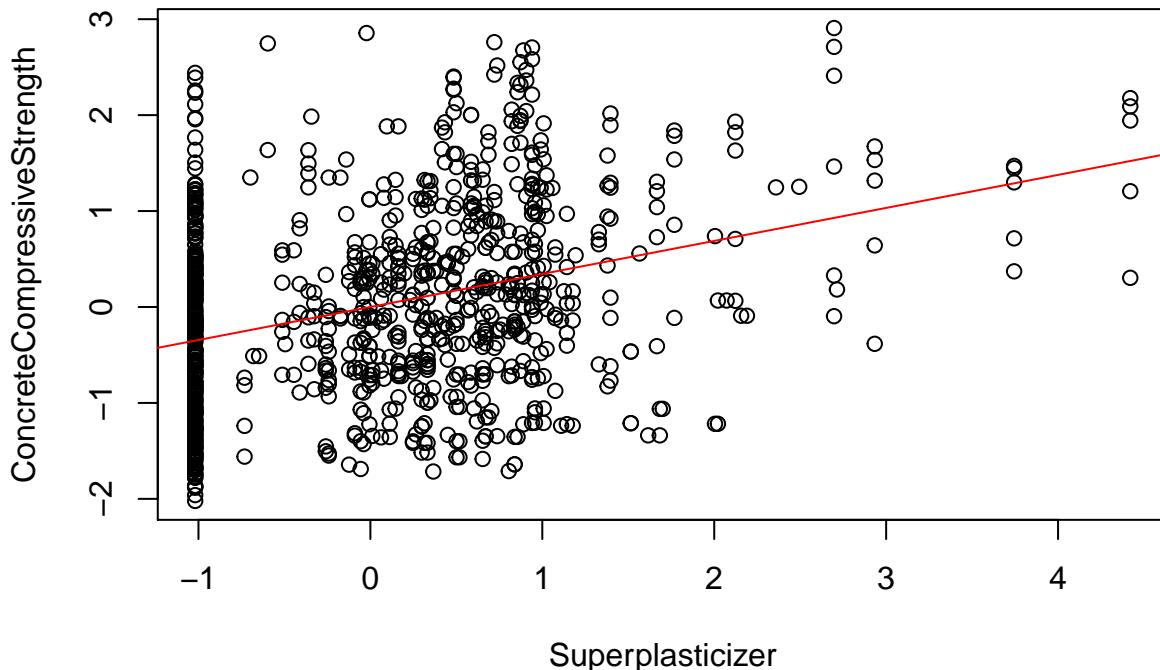
##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ Superplasticizer,
##      data = concrete)
##
## Residuals:
##      Min        1Q        Median         3Q        Max 
## -1.98736 -0.70051 -0.04512  0.61666  2.95276 
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -2.182e-17 2.963e-02   0.00     1    
## Superplasticizer 3.442e-01 2.965e-02  11.61 <2e-16 ***

```

```

## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9394 on 1003 degrees of freedom
## Multiple R-squared: 0.1185, Adjusted R-squared: 0.1176
## F-statistic: 134.8 on 1 and 1003 DF, p-value: < 2.2e-16
par(mfrow=c(1,1))
plot(ConcreteCompressiveStrength~Superplasticizer,concrete)
abline(fit2,col="red")

```



```

confint(fit2)

##                   2.5 %      97.5 %
## (Intercept) -0.05814629  0.05814629
## Superplasticizer  0.28603365  0.40238412
# Obtenemos el modelo para cada variable
fit3=lm(ConcreteCompressiveStrength~Age,data=concrete)
fit3

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ Age, data = concrete)
##
## Coefficients:
## (Intercept)          Age
## -2.756e-16  3.374e-01
# Visualizamos los resultados
summary(fit3)

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ Age, data = concrete)

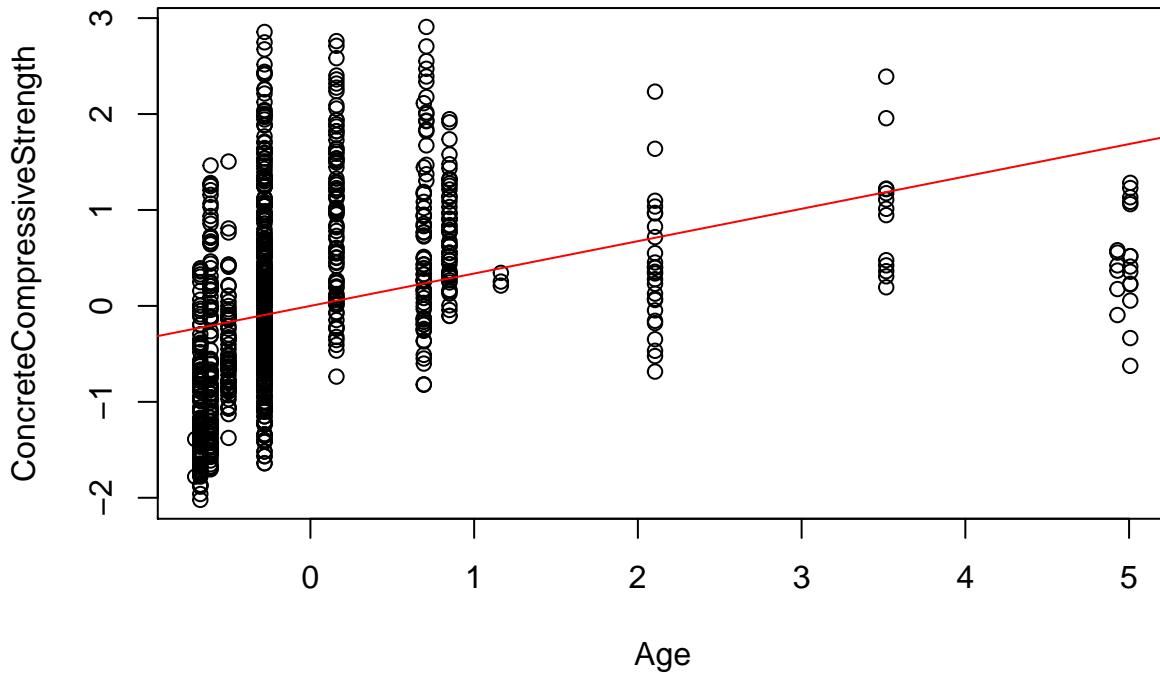
```

```

## 
## Residuals:
##      Min       1Q   Median      3Q      Max 
## -2.31385 -0.67123 -0.08051  0.55588  2.94992 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -2.756e-16 2.971e-02   0.00     1    
## Age         3.374e-01 2.972e-02  11.35  <2e-16 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 0.9418 on 1003 degrees of freedom
## Multiple R-squared:  0.1138, Adjusted R-squared:  0.1129 
## F-statistic: 128.8 on 1 and 1003 DF,  p-value: < 2.2e-16 

par(mfrow=c(1,1))
plot(ConcreteCompressiveStrength~Age,concrete)
abline(fit3,col="red")

```



```

confint(fit3)

##           2.5 %    97.5 %
## (Intercept) -0.05829988  0.05829988
## Age         0.27903803  0.39569584

# Obtenemos el modelo para cada variable
fit4=lm(ConcreteCompressiveStrength~BlastFurnaceSlag,data=concrete)
fit4

```

```

## 
## Call:
## lm(formula = ConcreteCompressiveStrength ~ BlastFurnaceSlag,
##     data = concrete)
## 
```

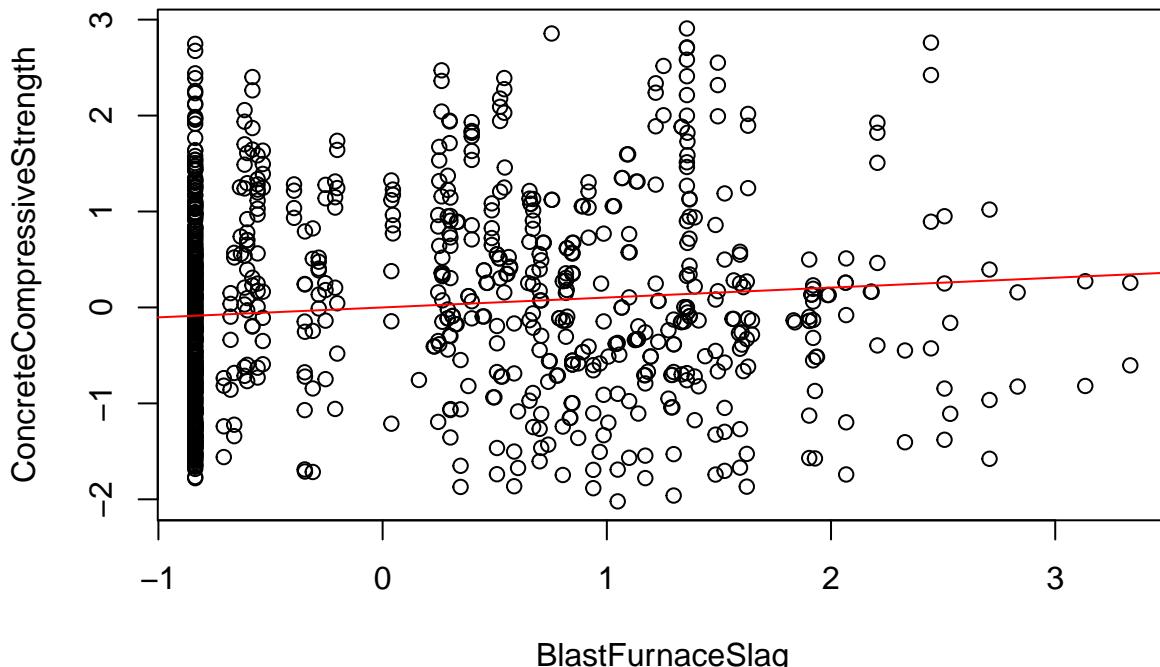
```

## Coefficients:
## (Intercept) BlastFurnaceSlag
## -8.959e-17 1.034e-01
#Visualizamos los resultados
summary(fit4)

## 
## Call:
## lm(formula = ConcreteCompressiveStrength ~ BlastFurnaceSlag,
##      data = concrete)
## 
## Residuals:
##    Min     1Q   Median     3Q    Max 
## -2.12993 -0.73249 -0.06896  0.62071  2.83375 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -8.959e-17 3.139e-02  0.000 1.00000  
## BlastFurnaceSlag 1.034e-01 3.141e-02   3.292  0.00103 ** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 0.9951 on 1003 degrees of freedom 
## Multiple R-squared:  0.01069,   Adjusted R-squared:  0.0097 
## F-statistic: 10.83 on 1 and 1003 DF,  p-value: 0.001031

par(mfrow=c(1,1))
plot(ConcreteCompressiveStrength~BlastFurnaceSlag,concrete)
abline(fit4,col="red")

```



```

confint(fit4)

##              2.5 %    97.5 %

```

```

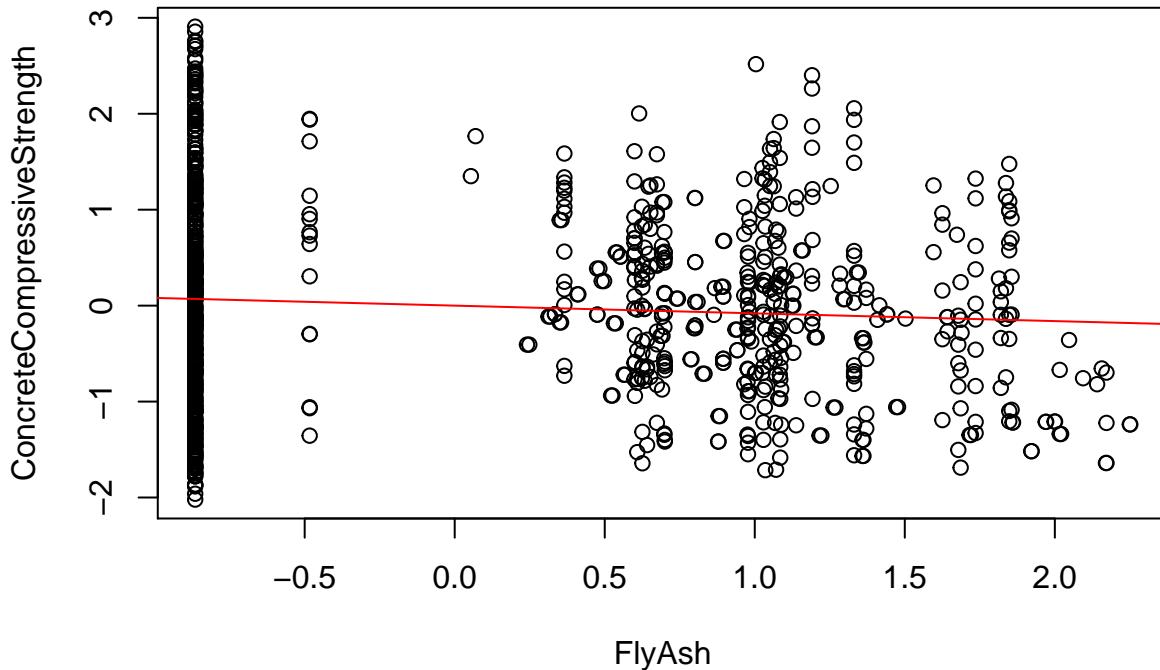
## (Intercept)      -0.06159889  0.06159889
## BlastFurnaceSlag  0.04174452  0.16500363
# Obtenemos el modelo para cada variable
fit5=lm(ConcreteCompressiveStrength~FlyAsh,data=concrete)
fit5

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ FlyAsh, data = concrete)
##
## Coefficients:
## (Intercept)      FlyAsh
## -9.631e-17   -8.065e-02
#Visualizamos los resultados
summary(fit5)

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ FlyAsh, data = concrete)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -2.09129 -0.73189 -0.02474  0.61096  2.83784
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -9.631e-17  3.146e-02   0.000   1.0000
## FlyAsh      -8.065e-02  3.147e-02  -2.562   0.0105 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9972 on 1003 degrees of freedom
## Multiple R-squared:  0.006504, Adjusted R-squared:  0.005514
## F-statistic: 6.566 on 1 and 1003 DF, p-value: 0.01054

par(mfrow=c(1,1))
plot(ConcreteCompressiveStrength~FlyAsh,concrete)
abline(fit5,col="red")

```



```
confint(fit5)
```

```
##              2.5 %      97.5 %
## (Intercept) -0.06172895  0.06172895
## FlyAsh       -0.14240783 -0.01888847
```

Como podemos comprobar mirando el coeficiente de determinación (R-squared) tanto el normal como el ajustado, el mejor modelo lo daría la variable “Cement” y tiene un p-value bastante bajo, lo cual nos afirma que ambos términos son significativos y se pueden utilizar como predictores para nuestro modelo lineal.

Sin embargo, el R-squared nos da un valor muy bajo así que no podemos decir que sea un buen modelo de regresión para nuestro conjunto de datos.

Ahora vamos a predecir los datos utilizando el modelo de regresión lineal que hemos aprendido usando la variable “Cement” y vamos a calcular el RMSE (raíz de la suma del error cuadrático medio) con respecto al conjunto inicial. Usaremos el conjunto inicial a modo de test.

```
yprime=predict(fit1,concrete)
sqrt(sum(abs(concrete$ConcreteCompressiveStrength-yprime)^2)/length(yprime))

## [1] 0.8722509
```

Regresión Lineal Múltiple

Para aplicar la regresión lineal múltiple voy a empezar desde arriba, es decir, en primer lugar generaremos un modelo multivariante con todas las variables e iré quitando poco a poco las que peor p-value tengan para intentar maximizar el valor del R-squared, introduciendo interacciones y no linealidad.

Como ya hemos graficado anteriormente todas las salidas una a una contra todas las variables y hemos aplicado regresión lineal múltiple, ya sabemos cuáles son a priori las mejores variables. Empezaremos viendo el valor de R-squared con todas las variables e iremos quitando algunas a ver si podemos maximizarlo.

```
fit.11=lm(ConcreteCompressiveStrength~.,data=concrete)
summary(fit.11)
```

```

## 
## Call:
## lm(formula = ConcreteCompressiveStrength ~ ., data = concrete)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -1.72801 -0.38713  0.03681  0.40816  2.14888 
## 
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -3.817e-16 1.993e-02 0.000 1.000000  
## Cement        7.512e-01 5.443e-02 13.802 < 2e-16 ***
## BlastFurnaceSlag 5.263e-01 5.375e-02  9.793 < 2e-16 *** 
## FlyAsh        3.376e-01 4.919e-02  6.863 1.19e-11 *** 
## Water         -1.995e-01 5.215e-02 -3.825 0.000139 *** 
## Superplasticizer 1.036e-01 3.377e-02  3.067 0.002218 ** 
## CoarseAggregate 7.469e-02 4.440e-02  1.682 0.092832 .  
## FineAggregate  9.043e-02 5.266e-02  1.717 0.086246 .  
## Age           4.390e-01 2.111e-02 20.801 < 2e-16 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 0.6319 on 996 degrees of freedom
## Multiple R-squared:  0.6039, Adjusted R-squared:  0.6007 
## F-statistic: 189.8 on 8 and 996 DF, p-value: < 2.2e-16

```

Podemos comprobar que hemos mejorado de forma bastante significativa con respecto al modelo de regresión lineal simple, ya que hemos pasado de un valor de R^2 de 0.24 a 0.60. Ahora iremos quitando las variables que menos p-value tienen para intentar mejorar.

```
fit.12=lm(ConcreteCompressiveStrength~.-Water,data=concrete)
summary(fit.12)
```

```

## 
## Call:
## lm(formula = ConcreteCompressiveStrength ~ . - Water, data = concrete)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -1.77657 -0.37344  0.02923  0.40864  2.21708 
## 
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -5.646e-16 2.007e-02 0.000      1  
## Cement        8.974e-01 3.902e-02 22.997 < 2e-16 ***
## BlastFurnaceSlag 6.650e-01 3.994e-02 16.650 < 2e-16 *** 
## FlyAsh        4.538e-01 3.895e-02 11.652 < 2e-16 *** 
## Superplasticizer 1.720e-01 2.885e-02  5.961 3.48e-09 *** 
## CoarseAggregate 2.140e-01 2.556e-02  8.375 < 2e-16 *** 
## FineAggregate  2.524e-01 3.152e-02  8.009 3.21e-15 *** 
## Age           4.345e-01 2.122e-02 20.478 < 2e-16 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 0.6362 on 997 degrees of freedom

```

```

## Multiple R-squared:  0.5981, Adjusted R-squared:  0.5952
## F-statistic: 211.9 on 7 and 997 DF,  p-value: < 2.2e-16
fit.13=lm(ConcreteCompressiveStrength~.-Superplasticizer,data=concrete)
summary(fit.13)

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ . - Superplasticizer,
##      data = concrete)
##
## Residuals:
##       Min     1Q   Median     3Q    Max 
## -1.85444 -0.38946  0.04933  0.41371  2.10711
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -3.380e-16  2.002e-02   0.000  1.000    
## Cement        7.474e-01  5.464e-02  13.678 < 2e-16 ***
## BlastFurnaceSlag 5.226e-01  5.396e-02   9.685 < 2e-16 *** 
## FlyAsh        3.554e-01  4.905e-02   7.245 8.67e-13 ***
## Water         -2.842e-01  4.443e-02  -6.395 2.46e-10 ***
## CoarseAggregate 2.635e-02  4.168e-02   0.632   0.527    
## FineAggregate  6.316e-02  5.212e-02   1.212   0.226    
## Age           4.410e-01  2.118e-02  20.819 < 2e-16 *** 
## --- 
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6346 on 997 degrees of freedom
## Multiple R-squared:  0.6001, Adjusted R-squared:  0.5973 
## F-statistic: 213.8 on 7 and 997 DF,  p-value: < 2.2e-16
fit.14=lm(ConcreteCompressiveStrength~.-CoarseAggregate,data=concrete)
summary(fit.14)

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ . - CoarseAggregate,
##      data = concrete)
##
## Residuals:
##       Min     1Q   Median     3Q    Max 
## -1.75156 -0.39765  0.03525  0.40246  2.17658
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -2.953e-16  1.995e-02   0.000  1.00000  
## Cement        6.802e-01  3.437e-02  19.790 < 2e-16 *** 
## BlastFurnaceSlag 4.546e-01  3.273e-02  13.888 < 2e-16 *** 
## FlyAsh        2.801e-01  3.542e-02   7.909 6.85e-15 ***
## Water         -2.715e-01  2.984e-02  -9.097 < 2e-16 *** 
## Superplasticizer 8.342e-02  3.160e-02   2.640  0.00842 ** 
## FineAggregate  1.718e-02  2.964e-02   0.580  0.56232  
## Age           4.373e-01  2.110e-02  20.725 < 2e-16 *** 
## --- 

```

```

## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6325 on 997 degrees of freedom
## Multiple R-squared:  0.6028, Adjusted R-squared:  0.6
## F-statistic: 216.1 on 7 and 997 DF,  p-value: < 2.2e-16
fit.15=lm(ConcreteCompressiveStrength~.~FineAggregate,data=concrete)
summary(fit.15)

```

```

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ . - FineAggregate,
##      data = concrete)
##
## Residuals:
##       Min        1Q     Median        3Q       Max
## -1.76111 -0.39750  0.03089  0.40825  2.14851
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.853e-16 1.995e-02  0.000 1.00000
## Cement       6.708e-01 2.781e-02 24.124 < 2e-16 ***
## BlastFurnaceSlag 4.470e-01 2.753e-02 16.236 < 2e-16 ***
## FlyAsh        2.715e-01 3.064e-02  8.860 < 2e-16 ***
## Water         -2.715e-01 3.103e-02 -8.750 < 2e-16 ***
## Superplasticizer 9.379e-02 3.332e-02  2.815 0.00497 **
## CoarseAggregate 1.165e-02 2.500e-02  0.466 0.64128
## Age           4.358e-01 2.104e-02 20.710 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```

## Residual standard error: 0.6325 on 997 degrees of freedom
## Multiple R-squared:  0.6027, Adjusted R-squared:  0.5999
## F-statistic: 216.1 on 7 and 997 DF,  p-value: < 2.2e-16
fit.16=lm(ConcreteCompressiveStrength~.~FineAggregate ~CoarseAggregate
          ~ Superplasticizer ~Water,data=concrete)
summary(fit.16)

```

```

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ . - FineAggregate -
##      CoarseAggregate - Superplasticizer - Water, data = concrete)
##
## Residuals:
##       Min        1Q     Median        3Q       Max
## -2.09846 -0.46135  0.01779  0.45400  2.69016
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -2.680e-16 2.209e-02  0.00      1
## Cement       7.672e-01 2.737e-02 28.03 <2e-16 ***
## BlastFurnaceSlag 4.837e-01 2.670e-02 18.11 <2e-16 ***
## FlyAsh        4.234e-01 2.783e-02 15.21 <2e-16 ***
## Age           3.591e-01 2.250e-02 15.96 <2e-16 ***

```

```

## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7002 on 1000 degrees of freedom
## Multiple R-squared: 0.5116, Adjusted R-squared: 0.5097
## F-statistic: 261.9 on 4 and 1000 DF, p-value: < 2.2e-16

```

Como podemos ver si quitamos las variables con el p-value más bajo no nos mejora el modelo, vamos a probar haciendo diferentes interacciones entre variables y usando no linealidad a ver si podemos mejorarlo.

```

fit.17=lm(ConcreteCompressiveStrength~ Cement *
           Age * Superplasticizer + BlastFurnaceSlag +
           FlyAsh + Water ,data=concrete)
summary(fit.17)

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ Cement * Age * Superplasticizer +
##     BlastFurnaceSlag + FlyAsh + Water, data = concrete)
##
## Residuals:
##       Min     1Q   Median     3Q    Max 
## -1.91673 -0.35102  0.01955  0.36364  2.23380
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 0.11198   0.01885  5.941 3.90e-09 ***
## Cement      0.64237   0.02496 25.738 < 2e-16 ***
## Age         0.88713   0.03361 26.393 < 2e-16 ***
## Superplasticizer 0.26400   0.03126  8.446 < 2e-16 ***
## BlastFurnaceSlag 0.45203   0.02330 19.403 < 2e-16 ***
## FlyAsh      0.22625   0.02862  7.906 7.03e-15 ***
## Water        -0.21397   0.02464 -8.685 < 2e-16 ***
## Cement:Age   -0.20030   0.02967 -6.752 2.48e-11 ***
## Cement:Superplasticizer -0.10215   0.01802 -5.670 1.87e-08 ***
## Age:Superplasticizer  0.53004   0.03479 15.236 < 2e-16 ***
## Cement:Age:Superplasticizer -0.13133   0.02840 -4.625 4.25e-06 ***
## --- 
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5586 on 994 degrees of freedom
## Multiple R-squared: 0.691, Adjusted R-squared: 0.6879
## F-statistic: 222.3 on 10 and 994 DF, p-value: < 2.2e-16

fit.18=lm(ConcreteCompressiveStrength~ Cement * Age * Superplasticizer +
           I(Age^2) +
           I(Superplasticizer^2) + BlastFurnaceSlag +
           Water ,data=concrete)
summary(fit.18)

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ Cement * Age * Superplasticizer +
##     I(Age^2) + I(Superplasticizer^2) + BlastFurnaceSlag + Water,
##     data = concrete)
## 
```

```

## Residuals:
##      Min     1Q   Median     3Q    Max
## -1.63603 -0.28025 -0.02727  0.27768  2.02932
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                0.33639   0.02123 15.846 < 2e-16 ***
## Cement                     0.59532   0.01814 32.811 < 2e-16 ***
## Age                        1.14803   0.03411 33.657 < 2e-16 ***
## Superplasticizer           0.38069   0.02500 15.230 < 2e-16 ***
## I(Age^2)                  -0.16265   0.01037 -15.680 < 2e-16 ***
## I(Superplasticizer^2)      -0.12873   0.01254 -10.268 < 2e-16 ***
## BlastFurnaceSlag          0.34755   0.01744 19.929 < 2e-16 ***
## Water                      -0.20251   0.02182 -9.281 < 2e-16 ***
## Cement:Age                 -0.10358   0.02685 -3.858 0.000122 ***
## Cement:Superplasticizer    -0.00937   0.01864 -0.503 0.615397
## Age:Superplasticizer       0.20790   0.03689  5.637 2.26e-08 ***
## Cement:Age:Superplasticizer -0.04149   0.02572 -1.613 0.106971
##
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4942 on 993 degrees of freedom
## Multiple R-squared:  0.7585, Adjusted R-squared:  0.7558
## F-statistic: 283.5 on 11 and 993 DF, p-value: < 2.2e-16
fit.19=lm(ConcreteCompressiveStrength~ Cement * Age * Superplasticizer
          + I(Age^2) + I(Superplasticizer^2) + BlastFurnaceSlag + FlyAsh
          + Water ,data=concrete)
summary(fit.19)

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ Cement * Age * Superplasticizer +
##     I(Age^2) + I(Superplasticizer^2) + BlastFurnaceSlag + FlyAsh +
##     Water, data = concrete)
##
## Residuals:
##      Min     1Q   Median     3Q    Max
## -1.56489 -0.28023 -0.01729  0.27517  2.08151
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                0.309998   0.021399 14.486 < 2e-16 ***
## Cement                     0.666684   0.021779 30.611 < 2e-16 ***
## Age                        1.150805   0.033580 34.271 < 2e-16 ***
## Superplasticizer           0.272314   0.031043  8.772 < 2e-16 ***
## I(Age^2)                  -0.162354   0.010211 -15.900 < 2e-16 ***
## I(Superplasticizer^2)      -0.103411   0.013110 -7.888 8.08e-15 ***
## BlastFurnaceSlag          0.415243   0.020844 19.922 < 2e-16 ***
## FlyAsh                     0.151587   0.026474  5.726 1.36e-08 ***
## Water                      -0.218714   0.021665 -10.095 < 2e-16 ***
## Cement:Age                 -0.100036   0.026438 -3.784 0.000164 ***
## Cement:Superplasticizer    0.001403   0.018449  0.076 0.939414
## Age:Superplasticizer       0.205340   0.036311  5.655 2.04e-08 ***
## Cement:Age:Superplasticizer -0.035499   0.025337 -1.401 0.161507

```

```

## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4865 on 992 degrees of freedom
## Multiple R-squared:  0.7662, Adjusted R-squared:  0.7634
## F-statistic: 270.9 on 12 and 992 DF,  p-value: < 2.2e-16
fit.l10=lm(ConcreteCompressiveStrength~ Cement * Age * Superplasticizer *
           Water * FlyAsh * BlastFurnaceSlag+
           I(Age^2) + I(Superplasticizer^2) +I(Water^2),data=concrete)
summary(fit.l10)

##
## Call:
## lm(formula = ConcreteCompressiveStrength ~ Cement * Age * Superplasticizer *
##      Water * FlyAsh * BlastFurnaceSlag + I(Age^2) + I(Superplasticizer^2) +
##      I(Water^2), data = concrete)
##
## Residuals:
##       Min     1Q   Median     3Q    Max 
## -1.49565 -0.25131 -0.02037  0.25517  2.34900
##
## Coefficients:
##                               Estimate
## (Intercept)                  0.623462
## Cement                      0.894416
## Age                         1.865768
## Superplasticizer              0.324324
## Water                       -0.075734
## FlyAsh                      0.554089
## BlastFurnaceSlag             0.847951
## I(Age^2)                     -0.142774
## I(Superplasticizer^2)        -0.054424
## I(Water^2)                   0.034856
## Cement:Age                   0.588379
## Cement:Superplasticizer      -0.015685
## Age:Superplasticizer          0.486435
## Cement:Water                 -0.091779
## Age:Water                     0.610050
## Superplasticizer:Water       0.145284
## Cement:FlyAsh                0.397481
## Age:FlyAsh                   0.970078
## Superplasticizer:FlyAsh      -0.042546
## Water:FlyAsh                 -0.004482
## Cement:BlastFurnaceSlag      0.418729
## Age:BlastFurnaceSlag         1.134589
## Superplasticizer:BlastFurnaceSlag 0.108905
## Water:BlastFurnaceSlag       0.100191
## FlyAsh:BlastFurnaceSlag      0.461263
## Cement:Age:Superplasticizer  0.096166
## Cement:Age:Water              0.130326
## Cement:Superplasticizer:Water 0.235411
## Age:Superplasticizer:Water   0.181224
## Cement:Age:FlyAsh            0.915112
## Cement:Superplasticizer:FlyAsh 0.187678

```

| | |
|--|------------|
| ## Age:Superplasticizer:FlyAsh | 0.539963 |
| ## Cement:Water:FlyAsh | 0.087617 |
| ## Age:Water:FlyAsh | 0.541907 |
| ## Superplasticizer:Water:FlyAsh | 0.083013 |
| ## Cement:Age:BlastFurnaceSlag | 1.045355 |
| ## Cement:Superplasticizer:BlastFurnaceSlag | 0.190778 |
| ## Age:Superplasticizer:BlastFurnaceSlag | 0.526675 |
| ## Cement:Water:BlastFurnaceSlag | 0.180069 |
| ## Age:Water:BlastFurnaceSlag | 0.712847 |
| ## Superplasticizer:Water:BlastFurnaceSlag | 0.177841 |
| ## Cement:FlyAsh:BlastFurnaceSlag | 0.342323 |
| ## Age:FlyAsh:BlastFurnaceSlag | 0.960421 |
| ## Superplasticizer:FlyAsh:BlastFurnaceSlag | 0.128381 |
| ## Water:FlyAsh:BlastFurnaceSlag | 0.146871 |
| ## Cement:Age:Superplasticizer:Water | 0.230343 |
| ## Cement:Age:Superplasticizer:FlyAsh | 0.617177 |
| ## Cement:Age:Water:FlyAsh | 0.680946 |
| ## Cement:Superplasticizer:Water:FlyAsh | 0.319808 |
| ## Age:Superplasticizer:Water:FlyAsh | 0.060127 |
| ## Cement:Age:Superplasticizer:BlastFurnaceSlag | 0.697783 |
| ## Cement:Age:Water:BlastFurnaceSlag | 0.857225 |
| ## Cement:Superplasticizer:Water:BlastFurnaceSlag | 0.340271 |
| ## Age:Superplasticizer:Water:BlastFurnaceSlag | 0.137841 |
| ## Cement:Age:FlyAsh:BlastFurnaceSlag | 0.802447 |
| ## Cement:Superplasticizer:FlyAsh:BlastFurnaceSlag | 0.061142 |
| ## Age:Superplasticizer:FlyAsh:BlastFurnaceSlag | 0.793109 |
| ## Cement:Water:FlyAsh:BlastFurnaceSlag | 0.063967 |
| ## Age:Water:FlyAsh:BlastFurnaceSlag | 1.224268 |
| ## Superplasticizer:Water:FlyAsh:BlastFurnaceSlag | 0.198040 |
| ## Cement:Age:Superplasticizer:Water:FlyAsh | 0.509070 |
| ## Cement:Age:Superplasticizer:Water:BlastFurnaceSlag | 0.637351 |
| ## Cement:Age:Superplasticizer:FlyAsh:BlastFurnaceSlag | 0.638104 |
| ## Cement:Age:Water:FlyAsh:BlastFurnaceSlag | 0.975423 |
| ## Cement:Superplasticizer:Water:FlyAsh:BlastFurnaceSlag | 0.341967 |
| ## Age:Superplasticizer:Water:FlyAsh:BlastFurnaceSlag | 0.173417 |
| ## Cement:Age:Superplasticizer:Water:FlyAsh:BlastFurnaceSlag | 0.594243 |
| ## | Std. Error |
| ## (Intercept) | 0.103996 |
| ## Cement | 0.101984 |
| ## Age | 0.349608 |
| ## Superplasticizer | 0.276479 |
| ## Water | 0.167012 |
| ## FlyAsh | 0.118697 |
| ## BlastFurnaceSlag | 0.137867 |
| ## I(Age^2) | 0.010388 |
| ## I(Superplasticizer^2) | 0.038341 |
| ## I(Water^2) | 0.016525 |
| ## Cement:Age | 0.334464 |
| ## Cement:Superplasticizer | 0.292997 |
| ## Age:Superplasticizer | 0.958731 |
| ## Cement:Water | 0.158929 |
| ## Age:Water | 0.572538 |
| ## Superplasticizer:Water | 0.171079 |
| ## Cement:FlyAsh | 0.116733 |

| | |
|--|------------------|
| ## Age:FlyAsh | 0.389841 |
| ## Superplasticizer:FlyAsh | 0.324935 |
| ## Water:FlyAsh | 0.189761 |
| ## Cement:BlastFurnaceSlag | 0.165598 |
| ## Age:BlastFurnaceSlag | 0.468882 |
| ## Superplasticizer:BlastFurnaceSlag | 0.396600 |
| ## Water:BlastFurnaceSlag | 0.236552 |
| ## FlyAsh:BlastFurnaceSlag | 0.168213 |
| ## Cement:Age:Superplasticizer | 1.018778 |
| ## Cement:Age:Water | 0.543119 |
| ## Cement:Superplasticizer:Water | 0.206377 |
| ## Age:Superplasticizer:Water | 0.533033 |
| ## Cement:Age:FlyAsh | 0.395246 |
| ## Cement:Superplasticizer:FlyAsh | 0.337009 |
| ## Age:Superplasticizer:FlyAsh | 1.122620 |
| ## Cement:Water:FlyAsh | 0.185383 |
| ## Age:Water:FlyAsh | 0.650853 |
| ## Superplasticizer:Water:FlyAsh | 0.196134 |
| ## Cement:Age:BlastFurnaceSlag | 0.574493 |
| ## Cement:Superplasticizer:BlastFurnaceSlag | 0.426296 |
| ## Age:Superplasticizer:BlastFurnaceSlag | 1.388700 |
| ## Cement:Water:BlastFurnaceSlag | 0.247901 |
| ## Age:Water:BlastFurnaceSlag | 0.822086 |
| ## Superplasticizer:Water:BlastFurnaceSlag | 0.248267 |
| ## Cement:FlyAsh:BlastFurnaceSlag | 0.174308 |
| ## Age:FlyAsh:BlastFurnaceSlag | 0.578677 |
| ## Superplasticizer:FlyAsh:BlastFurnaceSlag | 0.454059 |
| ## Water:FlyAsh:BlastFurnaceSlag | 0.276911 |
| ## Cement:Age:Superplasticizer:Water | 0.686659 |
| ## Cement:Age:Superplasticizer:FlyAsh | 1.183915 |
| ## Cement:Age:Water:FlyAsh | 0.644635 |
| ## Cement:Superplasticizer:Water:FlyAsh | 0.236072 |
| ## Age:Superplasticizer:Water:FlyAsh | 0.620363 |
| ## Cement:Age:Superplasticizer:BlastFurnaceSlag | 1.503098 |
| ## Cement:Age:Water:BlastFurnaceSlag | 0.868225 |
| ## Cement:Superplasticizer:Water:BlastFurnaceSlag | 0.326665 |
| ## Age:Superplasticizer:Water:BlastFurnaceSlag | 0.809922 |
| ## Cement:Age:FlyAsh:BlastFurnaceSlag | 0.604476 |
| ## Cement:Superplasticizer:FlyAsh:BlastFurnaceSlag | 0.492110 |
| ## Age:Superplasticizer:FlyAsh:BlastFurnaceSlag | 1.594779 |
| ## Cement:Water:FlyAsh:BlastFurnaceSlag | 0.281108 |
| ## Age:Water:FlyAsh:BlastFurnaceSlag | 0.964708 |
| ## Superplasticizer:Water:FlyAsh:BlastFurnaceSlag | 0.285620 |
| ## Cement:Age:Superplasticizer:Water:FlyAsh | 0.788341 |
| ## Cement:Age:Superplasticizer:Water:BlastFurnaceSlag | 1.115093 |
| ## Cement:Age:Superplasticizer:FlyAsh:BlastFurnaceSlag | 1.735760 |
| ## Cement:Age:Water:FlyAsh:BlastFurnaceSlag | 0.983737 |
| ## Cement:Superplasticizer:Water:FlyAsh:BlastFurnaceSlag | 0.378990 |
| ## Age:Superplasticizer:Water:FlyAsh:BlastFurnaceSlag | 0.932065 |
| ## Cement:Age:Superplasticizer:Water:FlyAsh:BlastFurnaceSlag | 1.295479 |
| ## | t value Pr(> t) |
| ## (Intercept) | 5.995 2.90e-09 |
| ## Cement | 8.770 < 2e-16 |
| ## Age | 5.337 1.19e-07 |

| | | |
|--|---------|----------|
| ## Superplasticizer | 1.173 | 0.24107 |
| ## Water | -0.453 | 0.65032 |
| ## FlyAsh | 4.668 | 3.48e-06 |
| ## BlastFurnaceSlag | 6.150 | 1.14e-09 |
| ## I(Age^2) | -13.744 | < 2e-16 |
| ## I(Superplasticizer^2) | -1.419 | 0.15610 |
| ## I(Water^2) | 2.109 | 0.03518 |
| ## Cement:Age | 1.759 | 0.07887 |
| ## Cement:Superplasticizer | -0.054 | 0.95732 |
| ## Age:Superplasticizer | 0.507 | 0.61201 |
| ## Cement:Water | -0.577 | 0.56375 |
| ## Age:Water | 1.066 | 0.28692 |
| ## Superplasticizer:Water | 0.849 | 0.39598 |
| ## Cement:FlyAsh | 3.405 | 0.00069 |
| ## Age:FlyAsh | 2.488 | 0.01300 |
| ## Superplasticizer:FlyAsh | -0.131 | 0.89585 |
| ## Water:FlyAsh | -0.024 | 0.98116 |
| ## Cement:BlastFurnaceSlag | 2.529 | 0.01162 |
| ## Age:BlastFurnaceSlag | 2.420 | 0.01572 |
| ## Superplasticizer:BlastFurnaceSlag | 0.275 | 0.78369 |
| ## Water:BlastFurnaceSlag | 0.424 | 0.67199 |
| ## FlyAsh:BlastFurnaceSlag | 2.742 | 0.00622 |
| ## Cement:Age:Superplasticizer | 0.094 | 0.92482 |
| ## Cement:Age:Water | 0.240 | 0.81041 |
| ## Cement:Superplasticizer:Water | 1.141 | 0.25429 |
| ## Age:Superplasticizer:Water | 0.340 | 0.73394 |
| ## Cement:Age:FlyAsh | 2.315 | 0.02081 |
| ## Cement:Superplasticizer:FlyAsh | 0.557 | 0.57773 |
| ## Age:Superplasticizer:FlyAsh | 0.481 | 0.63064 |
| ## Cement:Water:FlyAsh | 0.473 | 0.63659 |
| ## Age:Water:FlyAsh | 0.833 | 0.40528 |
| ## Superplasticizer:Water:FlyAsh | 0.423 | 0.67221 |
| ## Cement:Age:BlastFurnaceSlag | 1.820 | 0.06914 |
| ## Cement:Superplasticizer:BlastFurnaceSlag | 0.448 | 0.65460 |
| ## Age:Superplasticizer:BlastFurnaceSlag | 0.379 | 0.70458 |
| ## Cement:Water:BlastFurnaceSlag | 0.726 | 0.46779 |
| ## Age:Water:BlastFurnaceSlag | 0.867 | 0.38610 |
| ## Superplasticizer:Water:BlastFurnaceSlag | 0.716 | 0.47397 |
| ## Cement:FlyAsh:BlastFurnaceSlag | 1.964 | 0.04984 |
| ## Age:FlyAsh:BlastFurnaceSlag | 1.660 | 0.09731 |
| ## Superplasticizer:FlyAsh:BlastFurnaceSlag | 0.283 | 0.77744 |
| ## Water:FlyAsh:BlastFurnaceSlag | 0.530 | 0.59597 |
| ## Cement:Age:Superplasticizer:Water | 0.335 | 0.73736 |
| ## Cement:Age:Superplasticizer:FlyAsh | 0.521 | 0.60228 |
| ## Cement:Age:Water:FlyAsh | 1.056 | 0.29109 |
| ## Cement:Superplasticizer:Water:FlyAsh | 1.355 | 0.17584 |
| ## Age:Superplasticizer:Water:FlyAsh | 0.097 | 0.92281 |
| ## Cement:Age:Superplasticizer:BlastFurnaceSlag | 0.464 | 0.64259 |
| ## Cement:Age:Water:BlastFurnaceSlag | 0.987 | 0.32373 |
| ## Cement:Superplasticizer:Water:BlastFurnaceSlag | 1.042 | 0.29784 |
| ## Age:Superplasticizer:Water:BlastFurnaceSlag | 0.170 | 0.86490 |
| ## Cement:Age:FlyAsh:BlastFurnaceSlag | 1.328 | 0.18466 |
| ## Cement:Superplasticizer:FlyAsh:BlastFurnaceSlag | 0.124 | 0.90115 |
| ## Age:Superplasticizer:FlyAsh:BlastFurnaceSlag | 0.497 | 0.61908 |

| | | |
|--|-------|---------|
| ## Cement:Water:FlyAsh:BlastFurnaceSlag | 0.228 | 0.82004 |
| ## Age:Water:FlyAsh:BlastFurnaceSlag | 1.269 | 0.20474 |
| ## Superplasticizer:Water:FlyAsh:BlastFurnaceSlag | 0.693 | 0.48825 |
| ## Cement:Age:Superplasticizer:Water:FlyAsh | 0.646 | 0.51860 |
| ## Cement:Age:Superplasticizer:Water:BlastFurnaceSlag | 0.572 | 0.56775 |
| ## Cement:Age:Superplasticizer:FlyAsh:BlastFurnaceSlag | 0.368 | 0.71324 |
| ## Cement:Age:Water:FlyAsh:BlastFurnaceSlag | 0.992 | 0.32167 |
| ## Cement:Superplasticizer:Water:FlyAsh:BlastFurnaceSlag | 0.902 | 0.36712 |
| ## Age:Superplasticizer:Water:FlyAsh:BlastFurnaceSlag | 0.186 | 0.85244 |
| ## Cement:Age:Superplasticizer:Water:FlyAsh:BlastFurnaceSlag | 0.459 | 0.64655 |
| ## | | |
| ## (Intercept) | *** | |
| ## Cement | *** | |
| ## Age | *** | |
| ## Superplasticizer | | |
| ## Water | | |
| ## FlyAsh | *** | |
| ## BlastFurnaceSlag | *** | |
| ## I(Age^2) | *** | |
| ## I(Superplasticizer^2) | | |
| ## I(Water^2) | * | |
| ## Cement:Age | . | |
| ## Cement:Superplasticizer | | |
| ## Age:Superplasticizer | | |
| ## Cement:Water | | |
| ## Age:Water | | |
| ## Superplasticizer:Water | | |
| ## Cement:FlyAsh | *** | |
| ## Age:FlyAsh | * | |
| ## Superplasticizer:FlyAsh | | |
| ## Water:FlyAsh | | |
| ## Cement:BlastFurnaceSlag | * | |
| ## Age:BlastFurnaceSlag | * | |
| ## Superplasticizer:BlastFurnaceSlag | | |
| ## Water:BlastFurnaceSlag | | |
| ## FlyAsh:BlastFurnaceSlag | ** | |
| ## Cement:Age:Superplasticizer | | |
| ## Cement:Age:Water | | |
| ## Cement:Superplasticizer:Water | | |
| ## Age:Superplasticizer:Water | | |
| ## Cement:Age:FlyAsh | * | |
| ## Cement:Superplasticizer:FlyAsh | | |
| ## Age:Superplasticizer:FlyAsh | | |
| ## Cement:Water:FlyAsh | | |
| ## Age:Water:FlyAsh | | |
| ## Superplasticizer:Water:FlyAsh | | |
| ## Cement:Age:BlastFurnaceSlag | . | |
| ## Cement:Superplasticizer:BlastFurnaceSlag | | |
| ## Age:Superplasticizer:BlastFurnaceSlag | | |
| ## Cement:Water:BlastFurnaceSlag | | |
| ## Age:Water:BlastFurnaceSlag | | |
| ## Superplasticizer:Water:BlastFurnaceSlag | | |
| ## Cement:FlyAsh:BlastFurnaceSlag | * | |
| ## Age:FlyAsh:BlastFurnaceSlag | . | |

```

## Superplasticizer:FlyAsh:BlastFurnaceSlag
## Water:FlyAsh:BlastFurnaceSlag
## Cement:Age:Superplasticizer:Water
## Cement:Age:Superplasticizer:FlyAsh
## Cement:Age:Water:FlyAsh
## Cement:Superplasticizer:Water:FlyAsh
## Age:Superplasticizer:Water:FlyAsh
## Cement:Age:Superplasticizer:BlastFurnaceSlag
## Cement:Age:BlastFurnaceSlag
## Cement:Superplasticizer:Water:BlastFurnaceSlag
## Age:Superplasticizer:Water:BlastFurnaceSlag
## Cement:Age:FlyAsh:BlastFurnaceSlag
## Cement:Superplasticizer:FlyAsh:BlastFurnaceSlag
## Age:Superplasticizer:FlyAsh:BlastFurnaceSlag
## Cement:Water:FlyAsh:BlastFurnaceSlag
## Age:Water:FlyAsh:BlastFurnaceSlag
## Superplasticizer:Water:FlyAsh:BlastFurnaceSlag
## Cement:Age:Superplasticizer:Water:FlyAsh
## Cement:Age:Superplasticizer:Water:BlastFurnaceSlag
## Cement:Age:Superplasticizer:FlyAsh:BlastFurnaceSlag
## Cement:Age:Water:FlyAsh:BlastFurnaceSlag
## Cement:Superplasticizer:Water:FlyAsh:BlastFurnaceSlag
## Age:Superplasticizer:Water:FlyAsh:BlastFurnaceSlag
## Cement:Age:Superplasticizer:Water:FlyAsh:BlastFurnaceSlag
## ---  

## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4467 on 938 degrees of freedom
## Multiple R-squared: 0.8136, Adjusted R-squared: 0.8005
## F-statistic: 62.04 on 66 and 938 DF, p-value: < 2.2e-16

```

Haciendo diferentes pruebas, hemos conseguido mejorar el R-squared del modelo de 0.60 a 0.81 y a 0.80 el ajustado, respecto al aprendido con todas las variables, teniendo también un p-value muy bajo. Con lo cual podemos decir que hemos podido ajustar un buen modelo de regresión para nuestros datos.

Ahora vamos a hacer la predicción y calcular el RMSE del mejor modelo de regresión lineal múltiple que hemos obtenido y compararlo con el modelo de regresión lineal simple para comprobar cuanto hemos conseguido minimizar su RMSE, para el cual obtuvimos un valor de 0.87.

```

yprime=predict(fit.110,concrete)
sqrt(sum(abs(concrete$ConcreteCompressiveStrength-yprime)^2)/length(yprime))

```

```
## [1] 0.4315158
```

El valor del RMSE lo hemos conseguido reducir casi a la mitad, por lo tanto atendiendo a estas medidas podemos decir que el modelo generado es mucho mejor con respecto al de regresión lineal simple.

K-NN

Ahora vamos a pasar a utilizar el algoritmo KNN para regresión. En primer lugar vamos a ajustar el modelo para el conjunto de datos de california, usando la variable de salida “ConcreteCompressiveStrength” con el resto.

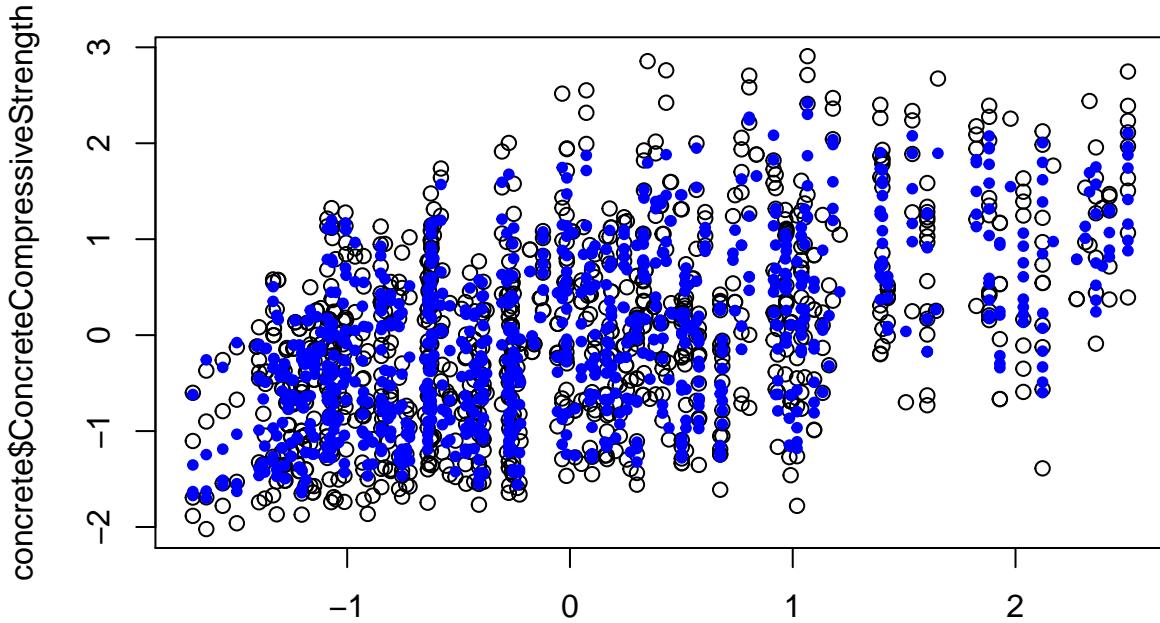
```

# Por defecto k = 7, distance = 2, kernel = "optimal"
# Ponemos scale = false porque los datos ya están normalizados
fitknn1 <- kknn(ConcreteCompressiveStrength ~ ., concrete, concrete, scale = FALSE)

```

A continuación vamos a visualizar los datos obtenidos, como sabemos que la mejor variable era "Cement" vamos a visualizar los datos respecto a esta.

```
plot(concrete$ConcreteCompressiveStrength~concrete$Cement)
points(concrete$Cement, fitknn1$fitted.values, col="blue", pch=20)
```



concrete\$Cement

Ahora

vamos a crear una predicción de los datos del conjunto con el modelo que hemos aprendido y posteriormente vamos a calcular de forma manual la raíz de ECM (RMSE).

```
yprime = fitknn1$fitted.values

sqrt(sum((concrete$ConcreteCompressiveStrength-yprime)^2)/length(yprime))

## [1] 0.3185735
```

El mejor modelo con la regresión lineal múltiple que habíamos nos daba un error de 0.43, un 0.12 menos comparado con el obtenido con knn. Sin embargo, esto también puede significar que haya mucho sobreajuste.

Ahora vamos a intentar minimizar el error utilizando la información que hemos obtenido en el análisis de regresión.

```
fitknn2 <- kknn(ConcreteCompressiveStrength ~ Cement * Age * Superplasticizer
+ I(Age^2) + I(Superplasticizer^2) + BlastFurnaceSlag +
FlyAsh + Water , concrete, concrete)

yprime = fitknn2$fitted.values
sqrt(sum((concrete$ConcreteCompressiveStrength-yprime)^2)/length(yprime)) #RMSE

## [1] 0.27073

fitknn3 <- kknn(ConcreteCompressiveStrength ~ Cement + Age + Superplasticizer
+ BlastFurnaceSlag + FlyAsh +
Water, concrete, concrete)

yprime = fitknn3$fitted.values
sqrt(sum((concrete$ConcreteCompressiveStrength-yprime)^2)/length(yprime)) #RMSE

## [1] 0.2973718
```

```

fitknn4 <- kknn(ConcreteCompressiveStrength ~ Cement * Age * Superplasticizer * BlastFurnaceSlag * Fly
yprime = fitknn4$fitted.values
sqrt(sum((concrete$ConcreteCompressiveStrength-yprime)^2)/length(yprime)) #RMSE
## [1] 0.2537708

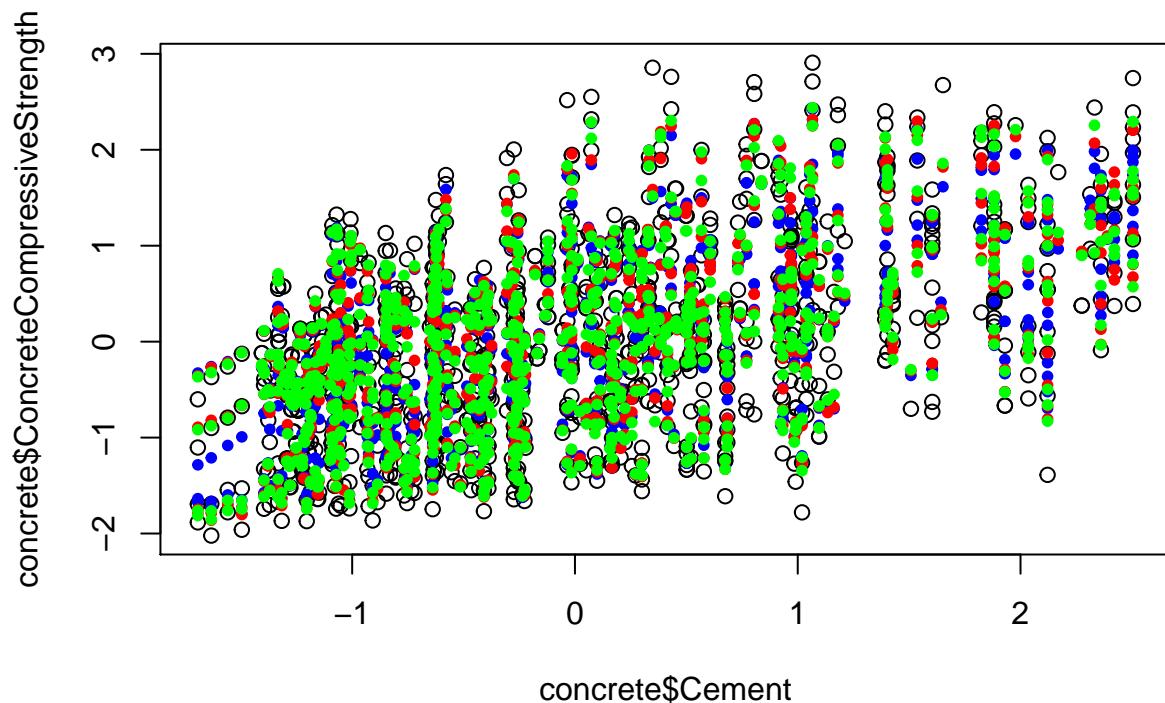
```

Hemos conseguido un RMSE menor que el del primer modelo que hemos aprendido con knn, los mostraremos ahora en un gráfico.

```

plot(concrete$ConcreteCompressiveStrength~concrete$Cement)
points(concrete$Cement,fitknn3$fitted.values,col="blue",pch=20)
points(concrete$Cement,fitknn2$fitted.values,col="red",pch=20)
points(concrete$Cement,fitknn4$fitted.values,col="green",pch=20)

```



Ahora vamos a utilizar k-fcv para ver la capacidad de generalización del modelo. Vamos a probarlo con la regresión lineal múltiple con todas las variables, luego con el mejor modelo que hemos generado y después con knn, primero el de todas las variables y segundo el mejor que hemos generado.

Empezaremos con las regresiones lineales múltiples, en primer lugar la regresión lineal múltiple con todas las variables.

```

nombre <- "concrete"

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

lmMSEtrain
## [1] 107.069
lmMSEtest
## [1] 109.0223

```

Continuaremos con el modelo que habíamos generado y que era el mejor en términos de R-squared.

```

nombre <- "concrete"

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~X1 * X8 * X5 * X4 * X3 * X2+ I(X8^2) + I(X5^2) +I(X4^2),x_tra)
  yprime=predict(fitMulti,test)
  sum(abs(test$Y-yprime)^2)/length(yprime) ##MSE
}

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

lmMSEtrain

```

```

## [1] 49.40483
lmMSEtest

## [1] 61.22548

Ahora seguiremos con el knn con todas las variables.

nombre <- "concrete"

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

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

knnMSEtrain

## [1] 28.69404
knnMSEtest

## [1] 68.34849

```

Como podemos observar, existe un sobreajuste muy alto, ya que el RMSE del train era muy bajo y el del test es más del doble. Ahora lo realizaremos con el mejor modelo que hemos generado para knn.

```

nombre <- "concrete"

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=knn(Y~X1 * X8 * X5 * X2 * X3 + X4,x_tra,test)
yprime=fitMulti$fitted.values
sum(abs(test$Y-yprime)^2)/length(yprime)
}

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

knnMSEtrain

## [1] 15.28562
knnMSEtest

## [1] 41.66119

```

Como podemos observar ahora el sobreajuste es aún mayor, existiendo una diferencia de casi el triple entre el train y el test. Podemos decir que en cuanto al train, como es lógico, el knn es el modelo que mejor se ajusta a los datos pero sin embargo es el modelo que más sobreajuste produce sobre este. El algoritmo de regresión lineal que hemos aprendido con las diferentes interacciones y no linealidad da un mayor error tanto en el train como en el test, sin embargo no sobreajusta tanto ya que los errores entre train y test no difieren tanto como el algoritmo de knn.

Por tanto podemos afirmar que nuestro modelo de regresión lineal múltiple es mejor modelo que el de Knn aunque este tenga un RMSE un poco más bajo.

Comparativa entre algoritmos

Para ello usaremos las tablas comparativas que nos han dado, donde los algoritmos han sido ejecutados en la máxima igualdad de condiciones y aplicaremos los pertinentes algoritmos de comparación. Sustituiremos los valores correspondientes a los datos obtenidos para nuestro dataset en el análisis. En primer lugar analizaremos las diferencias respecto al test y posteriormente respecto al train para comprobar si existe sobreajuste.

```

#leemos la tabla con los errores medios de test
resultados <- read.csv("regr_test_alumnos.csv")
tablatst <- cbind(resultados[,2:dim(resultados)[2]])
colnames(tablatst) <- names(resultados)[2:dim(resultados)[2]]
rownames(tablatst) <- resultados[,1]

#leemos la tabla con los errores medios de entrenamiento
resultados <- read.csv("regr_train_alumnos.csv")
tablatra <- cbind(resultados[,2:dim(resultados)[2]])
colnames(tablatra) <- names(resultados)[2:dim(resultados)[2]]

```

```
rownames(tablatra) <- resultados[,1]
```

Wilcoxon's Test (Datos Test)

En primer lugar aplicaremos el test de Wilcoxon, el cuál ordena por las diferencias para los rankings. Como estamos en regresión tenemos que normalizar el error, para ello usaremos la normalización propuesta en el laboratorio.

```
##lm (other) vs knn (ref)
# + 0.1 porque wilcox R falla para valores == 0 en la tabla
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

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

##  V
## 78
Rmenos

##  V
## 93
pvalue

## [1] 0.7660294
```

Según este test y tomando como algoritmo de referencia knn, no existen diferencias significativas entre ambos. Teniendo solamente un 23.4% de confianza de que sean distintos.

Friedman's Test (Datos Test)

```
test_friedman <- friedman.test(as.matrix(tablatst))
test_friedman

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

Con esto podemos afirmar, aproximadamente al 99% de confianza, que existen diferencias significativas al menos entre un par de algoritmos.

Holm's Test (Datos Test)

A continuación aplicaremos el test de post-hoc 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
```

Según este test podemos afirmar que existen diferencias significativas entre M5 y los otros dos, knn y lm, aproximadamente al 90 % de confianza, mientras que estos dos últimos pueden ser considerados prácticamente equivalentes.

Una vez hecho esto hemos comprobado las diferencias en cuanto a los datos del conjunto de datos de test. A continuación vamos a estudiar el conjunto de datos de train para ver los resultados y poder detectar un posible sobreaprendizaje.

Wilcoxon's Test (Datos Train)

En primer lugar aplicaremos el test de Wilcoxon, el cuál ordena por las diferencias para los rankings. Como estamos en regresión tenemos que normalizar el error, para ello usaremos la normalización propuesta en el laboratorio.

```
##lm (other) vs knn (ref)
# + 0.1 porque wilcox R falla para valores == 0 en la tabla
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
LMvsKNNtst <- wilcox.test(wilc_1_2[,2], wilc_1_2[,1], alternative = "two.sided", paired=TRUE)
```

```
Rmenos <- LMvsKNNtst$statistic
Rmas

##   V
## 10
Rmenos

##   V
## 161
pvalue

## [1] 0.000328064
```

Según este test aplicando en los resultados del training, podemos afirmar con una confianza del 99% que los algoritmos knn y lm son distintos.

Friedman's Test (Datos Train)

```
test_friedman <- friedman.test(as.matrix(tablatra))
test_friedman

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

Con esto podemos afirmar, al 99% de confianza, que existen diferencias significativas al menos entre un par de algoritmos.

Holm's Test (Datos Train)

A continuación aplicaremos el test de post-hoc Holm.

```
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 0.0039 0.0039
##
## P value adjustment method: holm
```

Según los datos de este test estamos viendo que entre todos los algoritmos existen diferencias significativas al 99% de confianza, siendo el algoritmo M5 el mejor, el segundo el knn, siendo el peor el algoritmo de regresión lineal múltiple. Por lo tanto, podemos decir que existe bastante sobreajuste en el algoritmo de knn, ya que en las pruebas realizadas en el conjunto de datos del test habíamos encontrado que no existían

diferencias significativas entre knn y lm, por consiguiente podemos afirmar que existe sobreajuste ya que en esta comparativa nos está diciendo que el algoritmo de Knn sería mejor.

Preprocesamiento: Clasificación

###Preparación de los datos

A continuación seguiremos con el conjunto de clasificación. El dataset que nos ha sido asignado es *australian*, lo cargaremos y pondremos su nombre a cada una de sus variables.

```
datasetC <- read.csv("australian/australian.dat", comment.char="@", header = FALSE)
names(datasetC) <- c("A1", "A2", "A3",
                      "A4", "A5", "A6",
                      "A7", "A8", "A9", "A10", "A11", "A12", "A13", "A14",
                      "Class")
str(datasetC)

## 'data.frame':   690 obs. of  15 variables:
## $ A1    : int  1 0 0 0 1 0 1 0 1 0 ...
## $ A2    : num  2208 2267 2958 2167 2017 ...
## $ A3    : num  1146 7 175 115 817 ...
## $ A4    : int  2 2 1 1 2 2 2 2 1 2 ...
## $ A5    : int  4 8 4 5 6 8 3 11 2 4 ...
## $ A6    : int  4 4 4 3 4 8 4 8 8 8 ...
## $ A7    : num  1585 165 125 0 196 ...
## $ A8    : int  0 0 0 1 1 1 0 1 0 1 ...
## $ A9    : int  0 0 0 1 1 1 0 1 0 1 ...
## $ A10   : int  0 0 0 11 14 2 0 6 0 3 ...
## $ A11   : int  1 0 1 1 0 0 0 0 0 1 ...
## $ A12   : int  2 2 2 2 2 2 2 2 2 2 ...
## $ A13   : int  100 160 280 0 60 100 60 43 176 100 ...
## $ A14   : int  1213 1 1 1 159 1 101 561 538 51 ...
## $ Class: int  0 0 0 1 1 1 0 1 0 0 ...

summary(datasetC)

##      A1          A2          A3          A4
## 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
## 
##      A5          A6          A7          A8
## 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
## 
##      A9          A10         A11         A12
## 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
##          A13          A14          Class
## 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

```

Como podemos observar tenemos un dataset con 690 observaciones y 15 variables, de las cuales 8 variables son de entrada y una es la clase, la salida: "Class". A continuación haremos una breve descripción de sus variables:

- A_1 : Variable entera en el intervalo $[0,1]$.
- A_2 : Variable real en el intervalo $[16.0,8025.0]$.
- A_3 : Variable real en el intervalo $[0.0,26335.0]$.
- A_4 : Variable entera en el intervalo $[1,3]$.
- A_5 : Variable entera en el intervalo $[1,14]$.
- A_6 : Variable entera en el intervalo $[1,9]$.
- A_7 : Variable real en el intervalo $[0.0,14415.0]$.
- A_8 : Variable entera en el intervalo $[0,1]$.
- A_9 : Variable entera en el intervalo $[0,1]$.
- A_{10} : Variable entera en el intervalo $[0,67]$.
- A_{11} : Variable entera en el intervalo $[0,1]$.
- A_{12} : Variable entera en el intervalo $[1,3]$.
- A_{13} : Variable entera en el intervalo $[0,2000]$.
- A_{14} : Variable entera en el intervalo $[1,100001]$.
- *Class*: Atributo de clasificación con valores $\{0,1\}$.

Según la información obtenida de la página de keel, este dataset hace referencia a las solicitudes de tarjetas de crédito. Sin embargo, para garantizar la confidencialidad se han puesto nombres insignificantes a las variables. También cabe destacar que el dataset original consta con 3 variables reales, 5 variables enteras y 6 variables nominales, pero en este dataset ya se han sustituido las variables nominales por valores numéricos representativos.

A continuación procederemos a comprobar si existen valores perdidos. En la página web de keel ya hemos podido comprobar que no los hay pero lo probaremos.

```
datasetC[is.na(datasetC)]
```

```
## numeric(0)
```

Como podemos comprobar no tenemos missing values lo cual nos ahorra ya un gran trabajo del preprocesamiento. Existen muchos valores con valor 0, pero en este caso no tienen porque tratarse como missing values ya que realmente la mayoría de 0s provienen de variables que eran nominales, lo cual el 0 representa a alguno de los tipos.

Además, como hemos comprobado en el análisis de las variables, no tenemos ninguna variable nominal entre las variables de entrada pues ya han sido transformadas, con lo cual podemos trabajar perfectamente con los datos para clasificación sin tener que hacer la transformación de ninguna variable. A continuación comprobaremos si existen valores repetidos.

```
datasetC.cleaned <- unique(datasetC)
nrow(datasetC)
```

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

```
nrow(datasetC.cleaned)
```

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

Como hemos observado no tenemos casos repetidos. Posteriormente, procederemos a normalizar los datos una vez hayamos analizado algunos valores estos.

```
####Análisis de los datos
```

En primer lugar haremos un análisis de cada variable, para ello calcularemos algunas medidas como son la tendencia central de los datos (media y mediana), algunas de las medidas de dispersión (desviación estándar y varianza) ya que las otras las hemos mostrado en el resumen inicial de los datos y otras medidas relativas estandar (quartiles, iqr).

```
# Cálculo de la media
```

```
apply(datasetC.cleaned, 2, mean)
```

```
##          A1           A2           A3           A4           A5  
## 0.6782609 2697.2855072 1187.3159420 1.7666667 7.3724638  
##          A6           A7           A8           A9           A10  
## 4.6927536 453.3666667 0.5231884 0.4275362 2.4000000  
##          A11          A12          A13          A14          Class  
## 0.4579710 1.9289855 184.0144928 1018.3855072 0.4449275
```

```
# Cálculo de la mediana
```

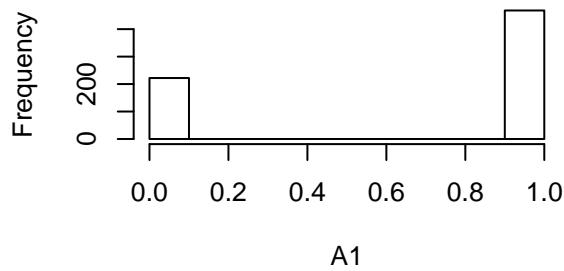
```
apply(datasetC.cleaned, 2, median)
```

```
##    A1     A2     A3     A4     A5     A6     A7     A8     A9     A10    A11    A12  
##    1    2629   125     2     8     4    35     1     0     0     0     0     2  
##    A13    A14 Class  
##   160      6     0
```

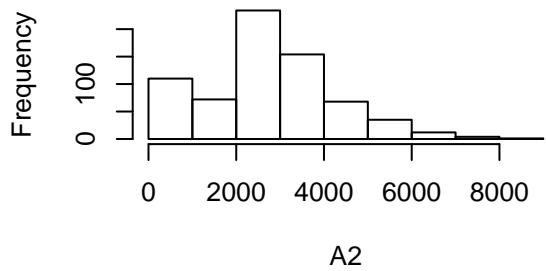
Ahora vamos a representar mediante histogramas las tendencias centrales de los datos.

```
i = 1  
histograma <- function (x) {  
  hist(x, xlab = names(datos)[i], main = c("Histograma de ", names(datos)[i]) )  
  assign("i", i + 1, envir = .GlobalEnv)  
}  
  
par(mfrow=c(2,2))  
datos = datasetC.cleaned[,-15]  
x <- sapply(datos, histograma)
```

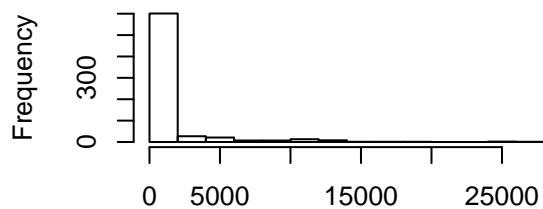
Histograma de A1



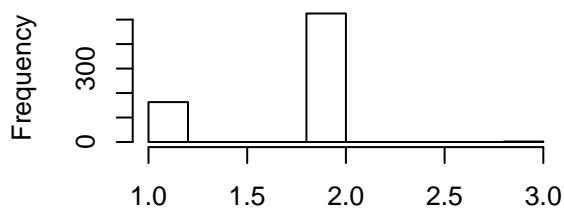
Histograma de A2



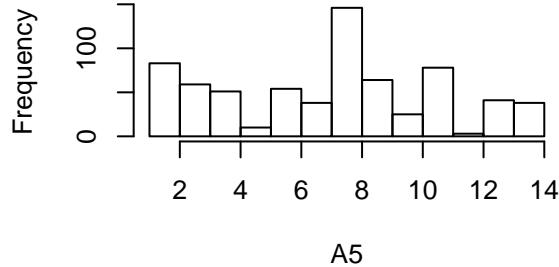
Histograma de A3



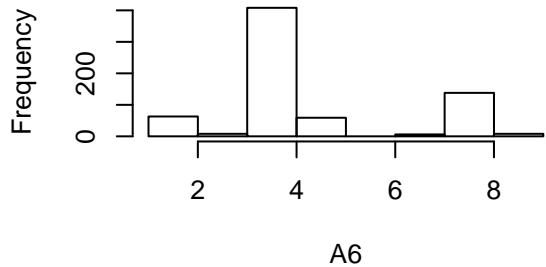
Histograma de A4



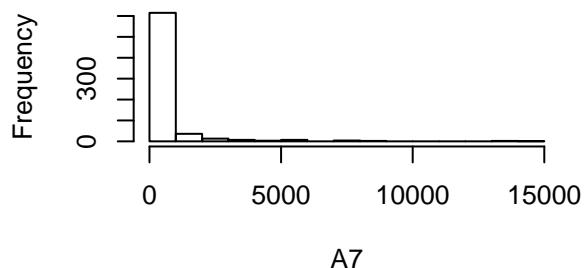
**A3
Histograma de A5**



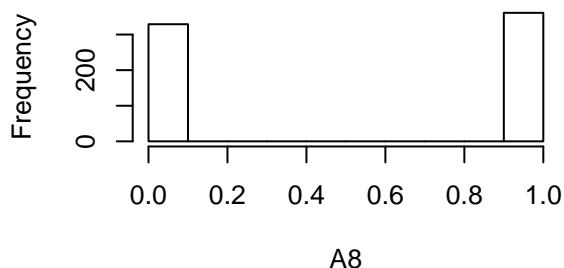
**A4
Histograma de A6**

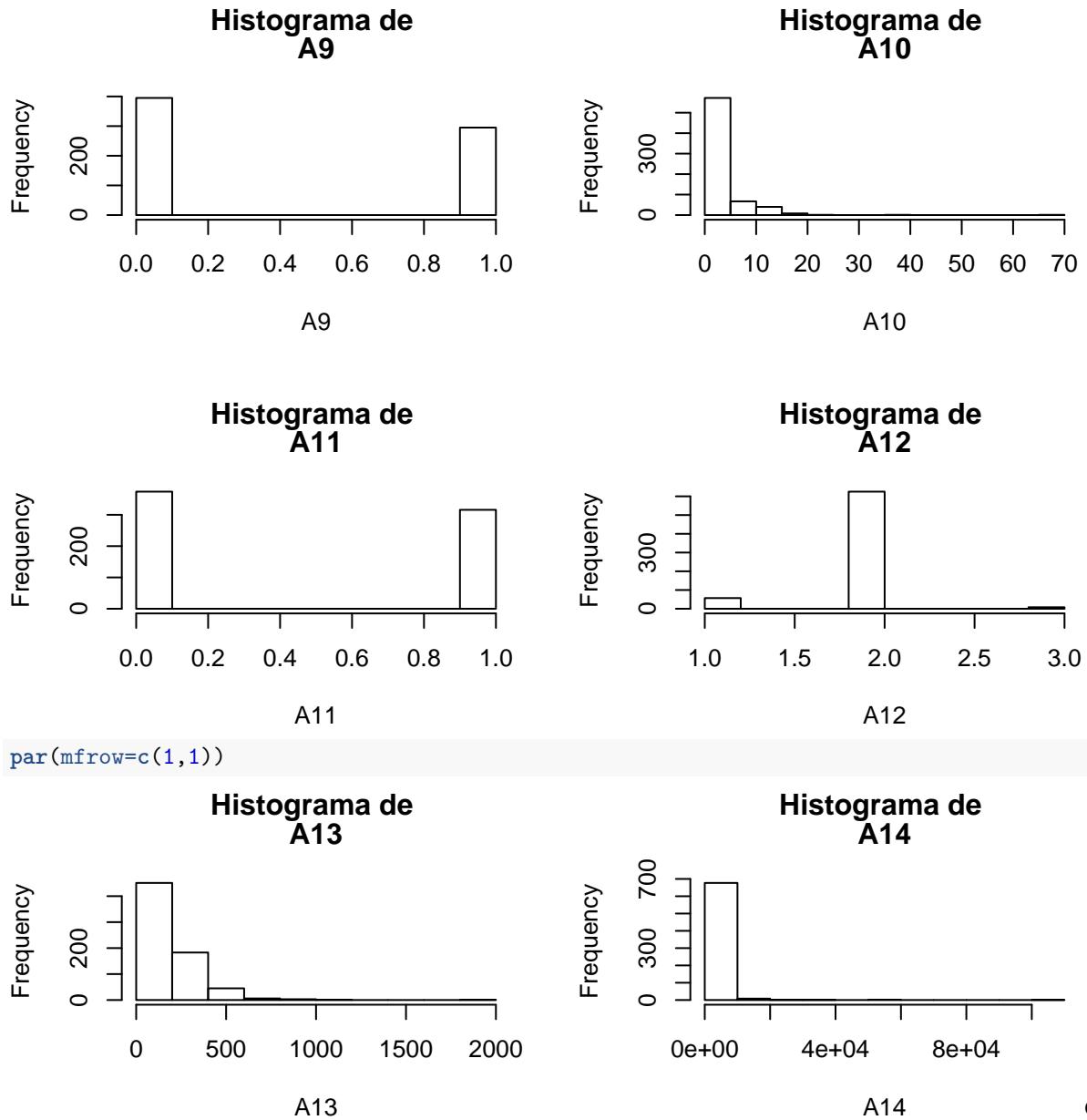


Histograma de A7



Histograma de A8





Como podemos observar, tenemos muchas variables con un solo par de valores como es lógico al provenir de variables nominales que han sido transformadas. Podemos observar por ejemplo la variable A1 y A8, además de observar que la variable A4 aunque existen tres posibles tipos realmente los que más abundan son de valor 1 o 2 y casi desapreciable con valor 3. Otras variables como son la variable A2, A5 y A10 tienen más dispersión. También podemos observar como muchas de estas variables tienen outliers como son la variable A13 o A14.

```
# Cálculo de la desviación típica
apply(datasetC.cleaned[,-15], 2, sd)
```

| | | | | | |
|----|-----------|--------------|--------------|--------------|-----------|
| ## | A1 | A2 | A3 | A4 | A5 |
| ## | 0.4674824 | 1554.5597320 | 3069.1100423 | 0.4300628 | 3.6832648 |
| ## | A6 | A7 | A8 | A9 | A10 |
| ## | 1.9923161 | 1387.9003240 | 0.4998243 | 0.4950800 | 4.8629400 |
| ## | A11 | A12 | A13 | A14 | |
| ## | 0.4985919 | 0.2988131 | 172.1592735 | 5210.1025983 | |

```

# Cálculo de la varianza
apply(datasetC.cleaned[,-15], 2, var)

##          A1           A2           A3           A4           A5
## 2.185398e-01 2.416656e+06 9.419436e+06 1.849540e-01 1.356644e+01
##          A6           A7           A8           A9           A10
## 3.969323e+00 1.926267e+06 2.498244e-01 2.451042e-01 2.364819e+01
##          A11          A12          A13          A14
## 2.485938e-01 8.928925e-02 2.963882e+04 2.714517e+07

```

Podemos observar como las variables A8, A9 y A11 tienen poca dispersión y otras como A2, A3 o A7 tienen una gran dispersión.

En cuanto a las varianzas, después las analizaremos más en profundidad.

```

# Cálculo de la media
apply(datasetC.cleaned[,-15], 2, quantile)

```

```

##          A1         A2         A3         A4         A5         A6
## 0%      0        16        0        1        1        1
## 25%     0       1942       15        2        4        4
## 50%     1       2629      125        2        8        4
## 75%     1       3525      665        2       10        5
## 100%    1       8025     26335       3       14       9
##          A7         A8         A9         A10        A11        A12        A13        A14
## 0.00    0        0        0        0        0        1        0        1.0
## 25.00   5.00     0        0        0        0        2       80        1.0
## 50.00  35.00     1        0        0        0        2      160        6.0
## 75.00 219.75     1        1        3        1        2      272      396.5
## 100.00 14415.00   1        1       67        1        3      2000 100001.0

```

```

# Cálculo de la mediana
apply(datasetC.cleaned[,-15], 2, IQR)

```

```

##          A1         A2         A3         A4         A5         A6         A7         A8         A9
## 1.00 1583.00 650.00 0.00 6.00 1.00 214.75 1.00 1.00
##          A10        A11        A12        A13        A14
## 3.00 1.00 0.00 192.00 395.50

```

Aquí podemos observar los cuartiles de las variables, aunque esto lo observaremos mejor después con la representación mediante boxplots.

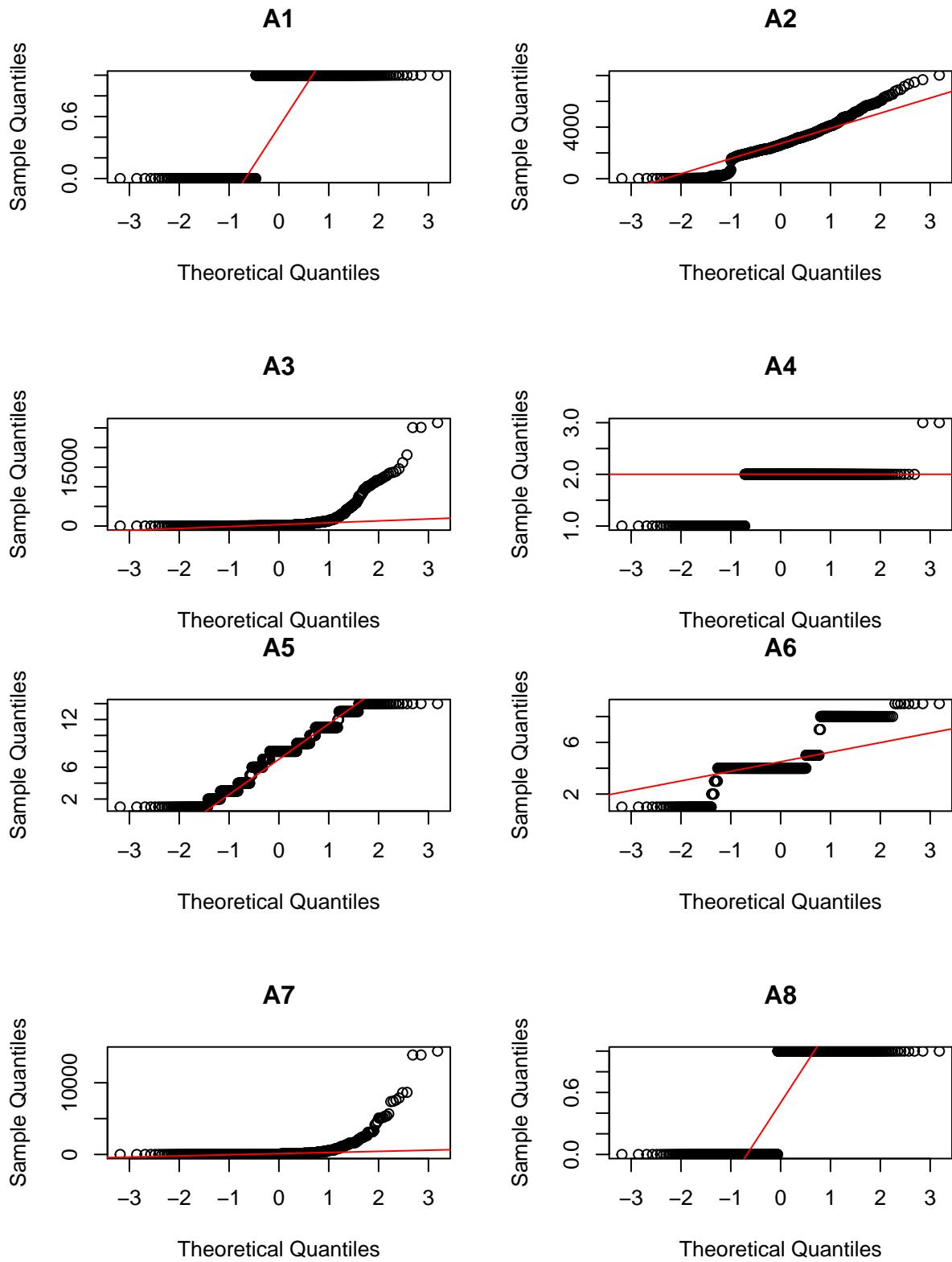
A continuación vamos a observar como están dispuestas las variables en relación a una distribución normal.

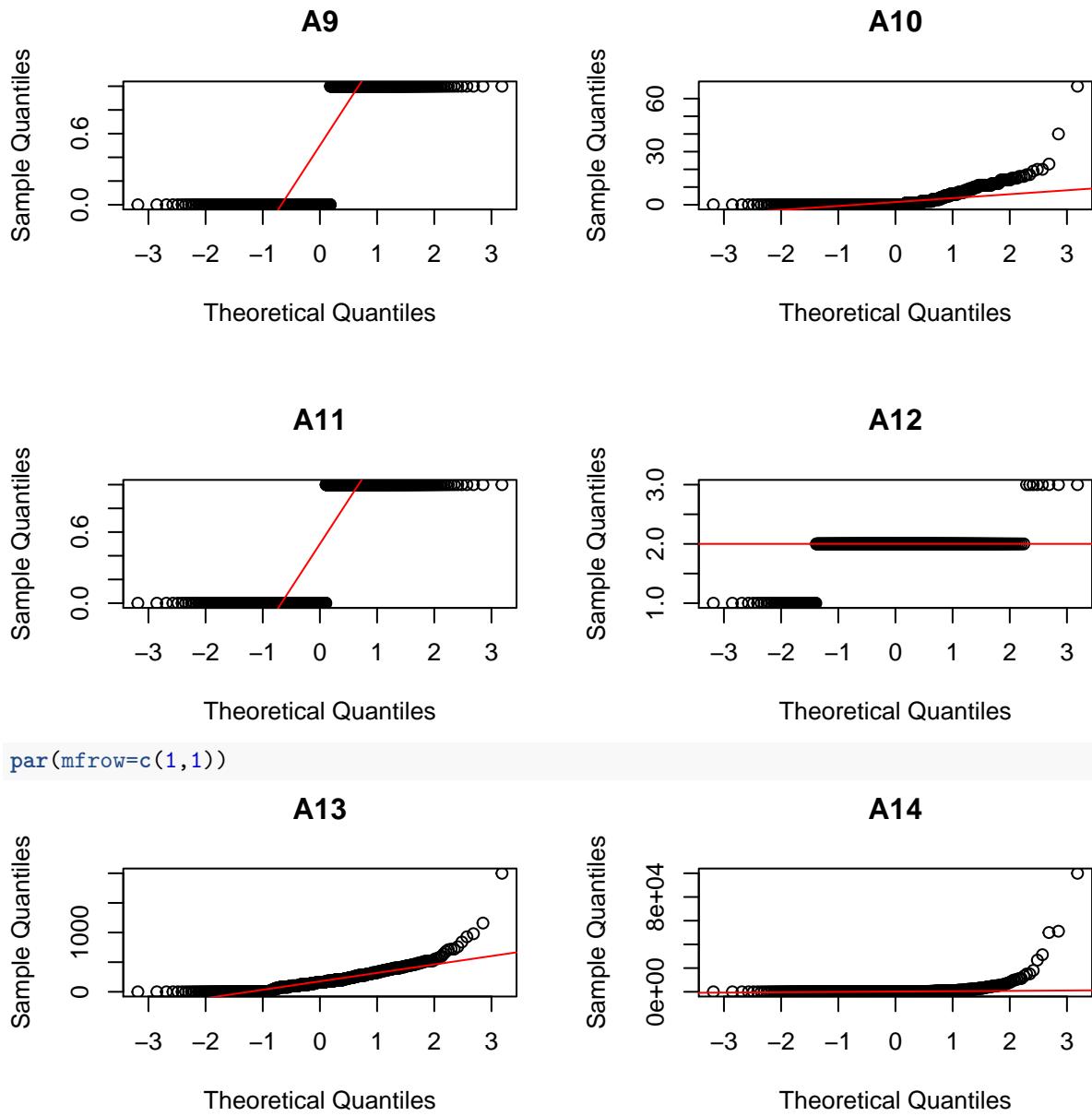
```

i = 1
qqplotAll <- function (x) {
  qqnorm(x, main = names(datos)[i])
  qqline(x, col = "Red")
  assign("i", i + 1, envir = .GlobalEnv)
}

par(mfrow=c(2,2))
datos = datasetC.cleaned[,-15]
x <- sapply(datos, qqplotAll)

```

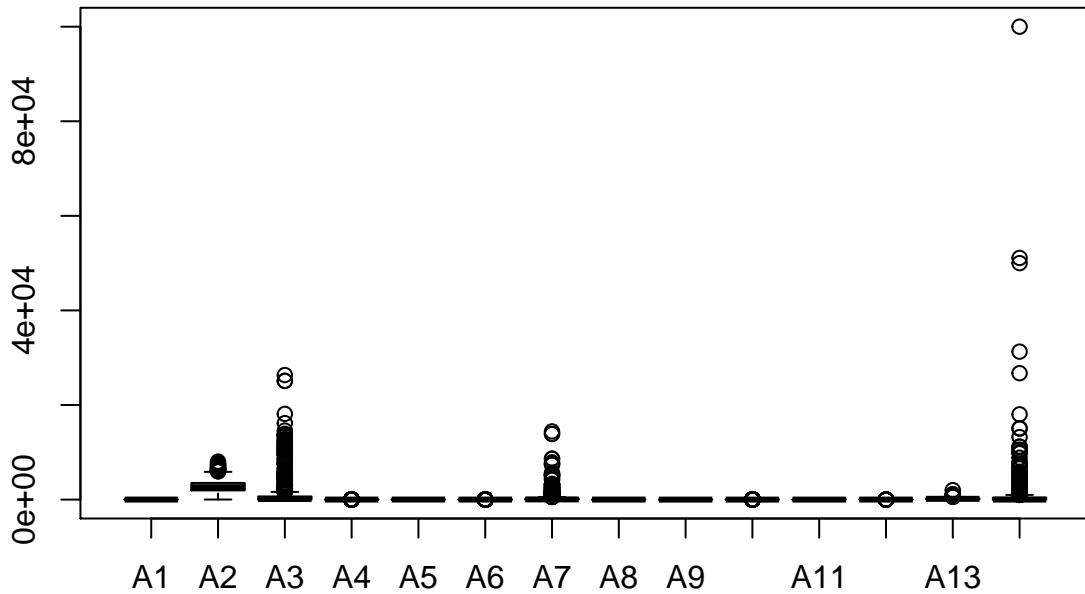




Como podemos observar no todas las variables siguen una distribución normal, las que mejor pinta tienen son A13, A10 y A5.

Ahora mostraremos graficamente mediante boxplot la dispersión de los datos y los cuartiles de los datos, así como sus outliers, a excepción de la variable de salida.

```
boxplot(datasetC.cleaned[,-15])
```

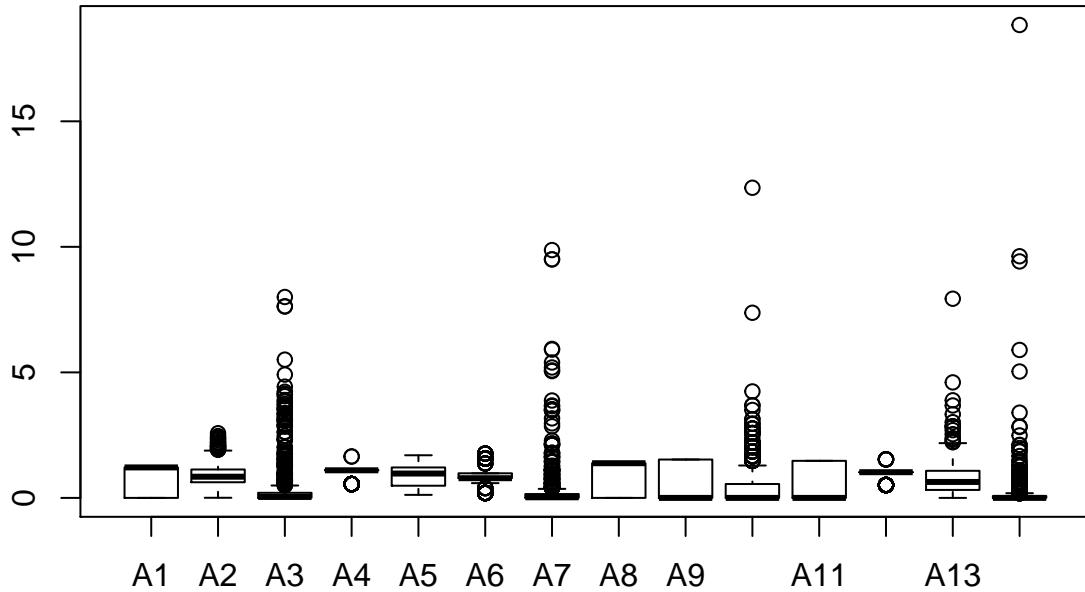


Como podemos observar los valores están muy dispares en los boxplot, así que procederemos a realizar una normalización de los datos. Usaremos sólamente el escalado, para evitar valores negativos.

```
datasetC.scaled = data.frame(scale(datasetC.cleaned[, -15], center = FALSE))
australian_n = datasetC.scaled
australian = datasetC.cleaned
```

Volveremos a graficar para comprobar los cambios.

```
boxplot(australian_n)
```



Ahora podemos observar los datos distribuidos de una forma más equiparable. Podemos observar como algunas variables tienen bastantes outliers, como las variables A3, A7, A10 o A14.

Ahora, antes de continuar, vamos a transformar nuestra variable de clasificación en un factor donde el valor “0” tomará el valor “clase 0” y el valor “1” tomará el valor “clase 1”, esto también nos ayudará para poder graficar por colores. Despues observaremos cuantos elementos hay de cada clase.

```

australian$Class <- factor(australian$Class, levels = c("0", "1"),
                           labels = c("Clase 0", "Clase 1"))
table(australian$Class)

##
## Clase 0 Clase 1
##      383      307
round(prop.table(table(australian$Class)) * 100, digits = 1)

##
## Clase 0 Clase 1
##      55.5      44.5

```

A continuación vamos a realizar un estudio sobre las varianzas de los predictores para saber cuales tienen varianzas similares, lo cual será interesante para aplicar algoritmos como lda o qda.

```
var(australian_n$A1)
```

```
## [1] 0.3217391
```

```
var(australian_n$A2)
```

```
## [1] 0.2490739
```

```
var(australian_n$A3)
```

```
## [1] 0.8696576
```

```
var(australian_n$A4)
```

```
## [1] 0.05586731
```

```
var(australian_n$A5)
```

```
## [1] 0.1995107
```

```
var(australian_n$A6)
```

```
## [1] 0.15253
```

```
var(australian_n$A7)
```

```
## [1] 0.9034572
```

```
var(australian_n$A8)
```

```
## [1] 0.4768116
```

```
var(australian_n$A9)
```

```
## [1] 0.5724638
```

```
var(australian_n$A10)
```

```
## [1] 0.8039076
```

```
var(australian_n$A11)
```

```
## [1] 0.542029
```

```
var(australian_n$A12)
```

```
## [1] 0.02340064
```

```

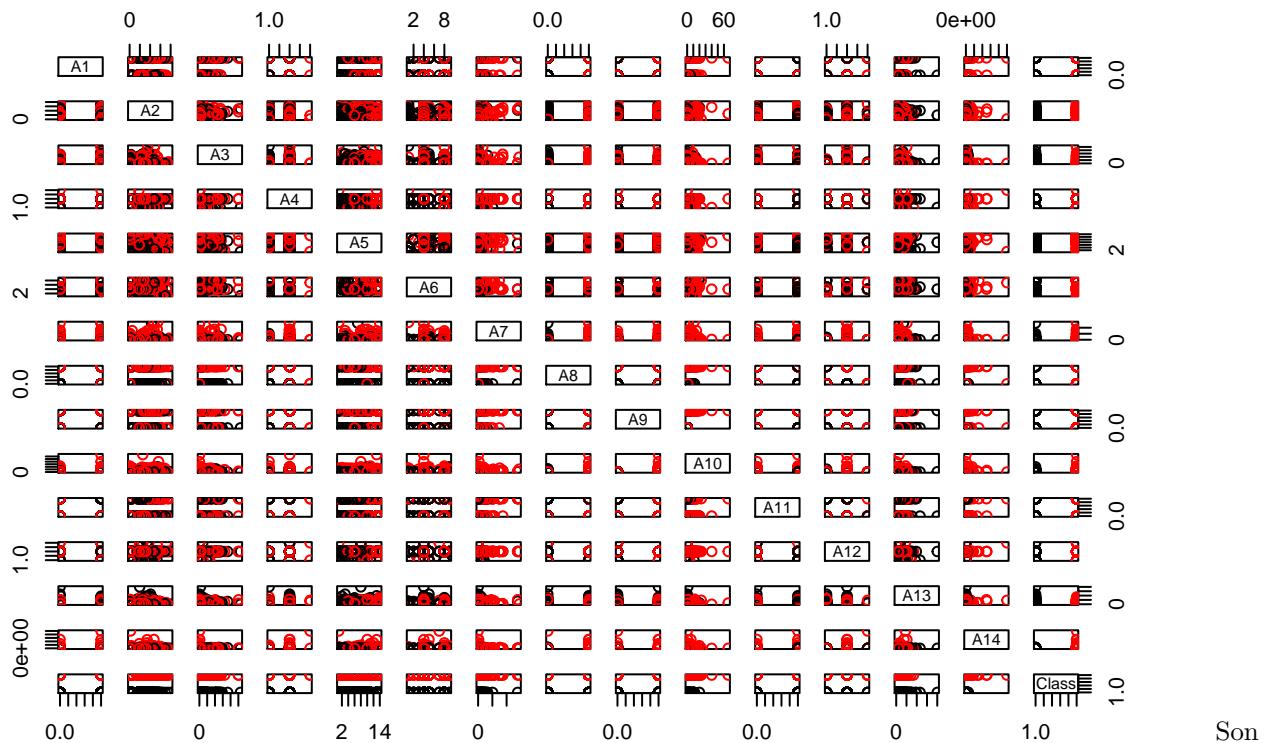
var(australian_n$A13)
## [1] 0.4663909
var(australian_n$A14)
## [1] 0.9631485

```

Las varianzas son muy dispares, aunque tenemos parejas de variables con varianza similar. Las más semejantes son la variable A8 con la variable A13. Los que tienen una varianza más cercana a 0 son la variable 4 y la variable 12.

A continuación vamos a dibujar todas las variables contra todas, menos la variable de clasificación, pintando en función de esta, para ver la correlación que guardan las variables entre sí.

```
plot(australian, col = australian$Class)
```



Son muchísimas posibles combinaciones, por lo que en primer lugar estudiaremos sus valores numéricos de correlación.

En primer lugar, comprobaremos si existe alta correlación entre alguna variable y la variable de clase. Para ello transformaremos en valor numérico la variable de clasificación, para poder aplicar la función de correlación.

```
cor(cbind(australian_n, class = as.numeric(australian$Class)))
```

| | A1 | A2 | A3 | A4 | A5 |
|-------|---------------|--------------|--------------|--------------|--------------|
| ## A1 | 1.0000000000 | 0.044898427 | 0.021323374 | -0.063528161 | -0.005321344 |
| ## A2 | 0.0448984268 | 1.0000000000 | 0.014978547 | 0.102699081 | -0.058621212 |
| ## A3 | 0.0213233743 | 0.014978547 | 1.0000000000 | 0.009327796 | 0.052321072 |
| ## A4 | -0.0635281608 | 0.102699081 | 0.009327796 | 1.0000000000 | 0.088846105 |
| ## A5 | -0.0053213443 | -0.058621212 | 0.052321072 | 0.088846105 | 1.0000000000 |
| ## A6 | 0.0526577428 | -0.001506815 | 0.065151425 | 0.046639021 | 0.402283761 |
| ## A7 | 0.0364498392 | 0.073636240 | 0.149224719 | 0.075503424 | 0.097800977 |
| ## A8 | 0.0009182252 | 0.115333982 | 0.023119586 | 0.129863180 | 0.311832581 |

```

## A9   -0.0632557880  0.063864422  0.020704655  0.162464416  0.222957198
## A10  -0.0084273404  0.117647696 -0.009713481  0.106457267  0.150165862
## A11   0.0539843221  0.013923701 -0.007909129  0.004963695  0.024738143
## A12  -0.0599006635  0.028160650  0.043200753  0.006399956  0.070222095
## A13   0.0585954466 -0.025026165 -0.014903065 -0.001757719  0.088139683
## A14   0.0036156380  0.002460758 -0.035580640  0.120065000  0.030735269
## class -0.0138970694  0.078889513  0.027546567  0.194306066  0.373711613
##           A6          A7          A8          A9          A10
## A1    0.052657743  0.036449839  0.0009182252 -0.06325579 -0.008427340
## A2   -0.001506815  0.073636240  0.1153339819  0.06386442  0.117647696
## A3    0.065151425  0.149224719  0.0231195865  0.02070465 -0.009713481
## A4    0.046639021  0.075503424  0.1298631797  0.16246442  0.106457267
## A5    0.402283761  0.097800977  0.3118325808  0.22295720  0.150165862
## A6    1.0000000000  0.077021503  0.2461930486  0.08039692  0.098840728
## A7    0.077021503  1.0000000000  0.1574065920  0.17103979  0.104332170
## A8    0.246193049  0.157406592  1.0000000000  0.43203236  0.379531965
## A9    0.080396918  0.171039794  0.4320323641  1.000000000  0.571498114
## A10   0.098840728  0.104332170  0.3795319648  0.57149811  1.000000000
## A11   0.093641130  0.126122087  0.0912757632  0.01704281  0.006943759
## A12   0.036434833  0.032238534  0.0450535439  0.17609731  0.101478957
## A13   0.070661763  0.005644437 -0.0673868187 -0.05332075 -0.119808064
## A14   0.064840849 -0.018071625  0.0900119022  0.07765182  0.063692439
## class  0.246567488  0.171810480  0.7204068159  0.45830133  0.406410009
##           A11         A12         A13         A14         class
## A1    0.053984322 -0.059900664  0.058595447  0.003615638 -0.01389707
## A2    0.013923701  0.028160650 -0.025026165  0.002460758  0.07888951
## A3   -0.007909129  0.043200753 -0.014903065 -0.035580640  0.02754657
## A4    0.004963695  0.006399956 -0.001757719  0.120065000  0.19430607
## A5    0.024738143  0.070222095  0.088139683  0.030735269  0.37371161
## A6    0.093641130  0.036434833  0.070661763  0.064840849  0.24656749
## A7    0.126122087  0.032238534  0.005644437 -0.018071625  0.17181048
## A8    0.091275763  0.045053544 -0.067386819  0.090011902  0.72040682
## A9    0.017042806  0.176097311 -0.053320746  0.077651821  0.45830133
## A10   0.006943759  0.101478957 -0.119808064  0.063692439  0.40641001
## A11   1.0000000000 -0.044416634  0.144253549  0.019201414  0.03162481
## A12  -0.044416634  1.0000000000 -0.079286984  0.139674137  0.11526093
## A13   0.144253549 -0.079286984  1.000000000  0.065608872 -0.09997241
## A14   0.019201414  0.139674137  0.065608872  1.000000000  0.17565720
## class  0.031624814  0.115260932 -0.099972408  0.175657201  1.000000000

```

De los datos podemos observar que la variable A8 tiene mucha correlación con la variable de clasificación, lo cual nos puede indicar que la variable de clasificación puede que derive de la variable A8.

Ahora comprobaremos que variables están más correladas entre sí.

```
cor(australian_n)
```

```

##           A1          A2          A3          A4          A5
## A1    1.0000000000  0.044898427  0.021323374 -0.063528161 -0.005321344
## A2    0.0448984268  1.000000000  0.014978547  0.102699081 -0.058621212
## A3    0.0213233743  0.014978547  1.000000000  0.009327796  0.052321072
## A4   -0.0635281608  0.102699081  0.009327796  1.000000000  0.088846105
## A5   -0.0053213443 -0.058621212  0.052321072  0.088846105  1.000000000
## A6    0.0526577428 -0.001506815  0.065151425  0.046639021  0.402283761
## A7    0.0364498392  0.073636240  0.149224719  0.075503424  0.097800977
## A8    0.0009182252  0.115333982  0.023119586  0.129863180  0.311832581

```

```

## A9 -0.0632557880 0.063864422 0.020704655 0.162464416 0.222957198
## A10 -0.0084273404 0.117647696 -0.009713481 0.106457267 0.150165862
## A11 0.0539843221 0.013923701 -0.007909129 0.004963695 0.024738143
## A12 -0.0599006635 0.028160650 0.043200753 0.006399956 0.070222095
## A13 0.0585954466 -0.025026165 -0.014903065 -0.001757719 0.088139683
## A14 0.0036156380 0.002460758 -0.035580640 0.120065000 0.030735269
##          A6      A7      A8      A9      A10
## A1 0.052657743 0.036449839 0.0009182252 -0.06325579 -0.008427340
## A2 -0.001506815 0.073636240 0.1153339819 0.06386442 0.117647696
## A3 0.065151425 0.149224719 0.0231195865 0.02070465 -0.009713481
## A4 0.046639021 0.075503424 0.1298631797 0.16246442 0.106457267
## A5 0.402283761 0.097800977 0.3118325808 0.22295720 0.150165862
## A6 1.000000000 0.077021503 0.2461930486 0.08039692 0.098840728
## A7 0.077021503 1.000000000 0.1574065920 0.17103979 0.104332170
## A8 0.246193049 0.157406592 1.000000000 0.43203236 0.379531965
## A9 0.080396918 0.171039794 0.4320323641 1.00000000 0.571498114
## A10 0.098840728 0.104332170 0.3795319648 0.57149811 1.000000000
## A11 0.093641130 0.126122087 0.0912757632 0.01704281 0.006943759
## A12 0.036434833 0.032238534 0.0450535439 0.17609731 0.101478957
## A13 0.070661763 0.005644437 -0.0673868187 -0.05332075 -0.119808064
## A14 0.064840849 -0.018071625 0.0900119022 0.07765182 0.063692439
##          A11      A12      A13      A14
## A1 0.053984322 -0.059900664 0.058595447 0.003615638
## A2 0.013923701 0.028160650 -0.025026165 0.002460758
## A3 -0.007909129 0.043200753 -0.014903065 -0.035580640
## A4 0.004963695 0.006399956 -0.001757719 0.120065000
## A5 0.024738143 0.070222095 0.088139683 0.030735269
## A6 0.093641130 0.036434833 0.070661763 0.064840849
## A7 0.126122087 0.032238534 0.005644437 -0.018071625
## A8 0.091275763 0.045053544 -0.067386819 0.090011902
## A9 0.017042806 0.176097311 -0.053320746 0.077651821
## A10 0.006943759 0.101478957 -0.119808064 0.063692439
## A11 1.000000000 -0.044416634 0.144253549 0.019201414
## A12 -0.044416634 1.000000000 -0.079286984 0.139674137
## A13 0.144253549 -0.079286984 1.000000000 0.065608872
## A14 0.019201414 0.139674137 0.065608872 1.000000000

```

Las variables que más correlación tienen entre sí son la variable A9 y A10, lo que puede indicar que sean similares. Las variables A5 y A6 también tienen alta correlación.

También cabe destacar que no vamos a crear nuevas variables ya que al tratarse de variables con nombres sin sentido no sabemos que significado tienen y por tanto no sabemos si podemos juntar alguna variable más.

Clasificación

k-NN

En primer lugar utilizaremos el algoritmo k-NN para clasificación, probando con distintos valores de k buscando el mejor modelo que se adapte a nuestros datos. Para los valores de k nos centraremos en aquellos valores raíz de n que sean impares, para evitar sobreajuste al utilizar una gran cantidad de vecinos y evitar casos de empates en el vecindario.

En primer lugar vamos a dividir nuestro dataset en un conjunto de train y test, separándolo de las etiquetas, dónde el 80% de los datos será para entrenar y el 20% para probar el modelo.

```

set.seed(2)
shuffle_ds <- sample(dim(australian_n)[1])
eightypct <- (dim(australian_n)[1] * 80) %/% 100
australian_train <- australian_n[shuffle_ds[1:eightypct], ]
australian_test <- australian_n[shuffle_ds[(eightypct+1):dim(australian_n)[1]], ]

australian_train_labels <- australian[shuffle_ds[1:eightypct], 15]
australian_test_labels <- australian[shuffle_ds[(eightypct+1):dim(australian_n)[1]], 15]

```

Ahora vamos a seleccionar los posibles valores de K con los que probaremos. Como nuestro dataset tiene un total de 690 observaciones nos fijaremos en los valores impares de raiz de 690.

```
vectorK = seq(1,as.integer(sqrt(dim(australian_n)[1])),2)
```

```
vectorK
```

```
## [1] 1 3 5 7 9 11 13 15 17 19 21 23 25
```

Una vez obtenidos los posibles k-valores, pasaremos a aplicar el algoritmo Knn.

```

set.seed(2)
knnModel.k <- train(australian_train, australian_train_labels, method="knn",
                     metric="Accuracy", tuneGrid = data.frame(.k=vectorK))

```

```
knnModel.k
```

```

## k-Nearest Neighbors
##
## 552 samples
## 14 predictor
## 2 classes: 'Clase 0', 'Clase 1'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 552, 552, 552, 552, 552, 552, ...
## Resampling results across tuning parameters:
##
##     k    Accuracy   Kappa
##     1    0.7653141  0.5202038
##     3    0.7788272  0.5489047
##     5    0.7959952  0.5833468
##     7    0.8116324  0.6149026
##     9    0.8182697  0.6278343
##    11    0.8223214  0.6364722
##    13    0.8225542  0.6366134
##    15    0.8297084  0.6505520
##    17    0.8301151  0.6512957
##    19    0.8358468  0.6627918
##    21    0.8342520  0.6593905
##    23    0.8362638  0.6633499
##    25    0.8338327  0.6577839
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 23.

```

La columna de "Accuracy" nos da la precisión del modelo, como es de bueno el ajuste. La columna "Kappa"

nos dice cuánto de mejor es nuestro clasificador respecto a una asignación aleatoria. En este caso el k que nos da una mejor precisión es $k = 23$, a continuación predeciremos las etiquetas para nuestro conjunto de test y comprobaremos la tasa de acierto.

```
set.seed(2)
knnPred.k <- predict(knnModel.k, newdata = australian_test)
postResample(pred = knnPred.k, obs = australian_test_labels)

## Accuracy      Kappa
## 0.8623188 0.7196921
```

Como podemos comprobar los resultados mejoran un poco respecto al train.

LDA

En primer lugar para que el algoritmo LDA funcione mejor tendríamos que partir de aquellos predictores que tengan una varianza similar y que sigan una distribución normal. También es aconsejable que aquellos predictores con varianza cercana a cero no utilizarlos.

Volveremos a comprobar el análisis de las variables en cuanto a la varianza. Nos fijaremos en aquellas que sean similares y desecharemos las que tengan varianza cercana a cero.

```
var(australian_n$A1)

## [1] 0.3217391
var(australian_n$A2)

## [1] 0.2490739
var(australian_n$A3)

## [1] 0.8696576
var(australian_n$A4)

## [1] 0.05586731
var(australian_n$A5)

## [1] 0.1995107
var(australian_n$A6)

## [1] 0.15253
var(australian_n$A7)

## [1] 0.9034572
var(australian_n$A8)

## [1] 0.4768116
var(australian_n$A9)

## [1] 0.5724638
var(australian_n$A10)

## [1] 0.8039076
```

```

var(australian_n$A11)

## [1] 0.542029

var(australian_n$A12)

## [1] 0.02340064

var(australian_n$A13)

## [1] 0.4663909

var(australian_n$A14)

## [1] 0.9631485

```

Las variables A4 y A12 tienen una varianza muy cercana a 0, por tanto no las utilizaremos a priori. Las variables A8, A9, A11 Y A13 son las que tienen una varianza más similar.

Vamos a proceder ahora a pasarles el test de Shapiro para comprobar su normalidad:

```

shapiro.test(australian_n$A8)

##
##  Shapiro-Wilk normality test
##
## data:  australian_n$A8
## W = 0.63579, p-value < 2.2e-16
shapiro.test(australian_n$A9)

##
##  Shapiro-Wilk normality test
##
## data:  australian_n$A9
## W = 0.62889, p-value < 2.2e-16
shapiro.test(australian_n$A11)

##
##  Shapiro-Wilk normality test
##
## data:  australian_n$A11
## W = 0.63399, p-value < 2.2e-16
shapiro.test(australian_n$A13)

##
##  Shapiro-Wilk normality test
##
## data:  australian_n$A13
## W = 0.82069, p-value < 2.2e-16

```

Aplicando el test de shapiro podemos comprobar como A8, A9 Y A11 son muy similares, sin embargo A13 no es tan similar. También cabe destacar que el p-value es similar. Por tanto usaremos estas para aprender nuestro modelo mediante lda.

```

lda.fit <- lda(australian_train_labels~A8+A9+A11,data=australian_train)
lda.fit

## Call:

```

```

## lda(australian_train_labels ~ A8 + A9 + A11, data = australian_train)
##
## Prior probabilities of groups:
##   Clase 0   Clase 1
## 0.5561594 0.4438406
##
## Group means:
##           A8         A9         A11
## Clase 0 0.2745032 0.3484643 0.6493240
## Clase 1 1.2518229 1.0230017 0.6448872
##
## Coefficients of linear discriminants:
##           LD1
## A8 1.8352849
## A9 0.4084482
## A11 -0.0877258

lda.pred <- predict(lda.fit, australian_test)

table(lda.pred$class, australian_test_labels)

##          australian_test_labels
##          Clase 0 Clase 1
## Clase 0      60      0
## Clase 1      16     62

mean(lda.pred$class == australian_test_labels)

## [1] 0.884058

```

Como podemos observar, hemos obtenido un mejor modelo que con knn ya que hemos aprendido un modelo con un 88 % de acierto.

Ahora predeciremos las etiquetas del train para obtener el porcentaje de acierto del modelo sobre el train

```

lda.pred.train <- predict(lda.fit, australian_train)
table(lda.pred.train$class, australian_train_labels)

##          australian_train_labels
##          Clase 0 Clase 1
## Clase 0      246      23
## Clase 1      61     222

mean(lda.pred.train$class == australian_train_labels)

## [1] 0.8478261

```

##QDA

Para QDA lo que tendremos en cuenta no es la varianza entre cada predictor, si no la varianza de cada predictor respecto a cada clase.

En primer lugar analizaremos la varianza de los predictores con respecto a la clase 0.

```

var(australian_n[australian$Class == "Clase 0", ]$A1)

## [1] 0.3190053

var(australian_n[australian$Class == "Clase 0", ]$A2)

## [1] 0.2096936

```

```

var(australian_n[australian$Class == "Clase 0",]$A3)

## [1] 0.7651166

var(australian_n[australian$Class == "Clase 0",]$A4)

## [1] 0.06455943

var(australian_n[australian$Class == "Clase 0",]$A5)

## [1] 0.1751803

var(australian_n[australian$Class == "Clase 0",]$A6)

## [1] 0.1372549

var(australian_n[australian$Class == "Clase 0",]$A7)

## [1] 0.3492665

var(australian_n[australian$Class == "Clase 0",]$A8)

## [1] 0.3073704

var(australian_n[australian$Class == "Clase 0",]$A9)

## [1] 0.4077462

var(australian_n[australian$Class == "Clase 0",]$A10)

## [1] 0.1227264

var(australian_n[australian$Class == "Clase 0",]$A11)

## [1] 0.539633

var(australian_n[australian$Class == "Clase 0",]$A12)

## [1] 0.02814837

var(australian_n[australian$Class == "Clase 0",]$A13)

## [1] 0.5093076

var(australian_n[australian$Class == "Clase 0",]$A14)

## [1] 0.01600418

```

Las varianzas son similares para las variables A1, A7 y A8. Ahora analizaremos respecto de la clase 1.

```

var(australian_n[australian$Class == "Clase 1",]$A1)

## [1] 0.3260635

var(australian_n[australian$Class == "Clase 1",]$A2)

## [1] 0.2955587

var(australian_n[australian$Class == "Clase 1",]$A3)

## [1] 1.001519

var(australian_n[australian$Class == "Clase 1",]$A4)

## [1] 0.04044965

```

```

var(australian_n[australian$Class == "Clase 1",]$A5)

## [1] 0.167797

var(australian_n[australian$Class == "Clase 1",]$A6)

## [1] 0.1512177

var(australian_n[australian$Class == "Clase 1",]$A7)

## [1] 1.538194

var(australian_n[australian$Class == "Clase 1",]$A8)

## [1] 0.1327084

var(australian_n[australian$Class == "Clase 1",]$A9)

## [1] 0.5092251

var(australian_n[australian$Class == "Clase 1",]$A10)

## [1] 1.357925

var(australian_n[australian$Class == "Clase 1",]$A11)

## [1] 0.54555707

var(australian_n[australian$Class == "Clase 1",]$A13)

## [1] 0.4038438

var(australian_n[australian$Class == "Clase 1",]$A14)

## [1] 2.081764

```

Las clases con varianza más similar son la A10 y la A12, sin embargo no coinciden con las variables que era más similares con respecto a la clase 0. Por tanto, vamos a aplicar qda utilizando todas las variables.

```

qda.fit <- qda(australian_train_labels ~ ., data=australian_train)

qda.fit

## Call:
## qda(australian_train_labels ~ ., data = australian_train)
## 
## Prior probabilities of groups:
##   Clase 0   Clase 1
## 0.5561594 0.4438406
## 
## Group means:
##          A1        A2        A3        A4        A5        A6
## Clase 0 0.8102185 0.8365106 0.3009092 0.9237582 0.7398601 0.8128483
## Clase 1 0.8320126 0.9204703 0.3614297 1.0229296 1.0637033 1.0201489
##          A7        A8        A9        A10       A11       A12
## Clase 0 0.1419000 0.2745032 0.3484643 0.1303243 0.6493240 0.9738426
## Clase 1 0.4816928 1.2518229 1.0230017 0.7886773 0.6448872 1.0113316
##          A13       A14
## Clase 0 0.7630804 0.03775891
## Clase 1 0.6525370 0.40009739

```

```

qda.pred <- predict(qda.fit,australian_test)



```

Como podemos observar, la tasa de acierto es más baja respecto a LDA, dándonos un valor del 83%. Por tanto, podemos afirmar que LDA es el mejor algoritmo para realizar una clasificación de nuestros datos.

Ahora repetiremos el mismo proceso que antes con lda para predecir las etiquetas del train.

```

qda.pred.train <- predict(qda.fit,australian_train)



```

Comparativa entre algoritmos

Para ello usaremos las tablas comparativas que nos han dado, donde los algoritmos han sido ejecutados en la máxima igualdad de condiciones y aplicaremos los pertinentes algoritmos de comparación. Sustituiremos los valores correspondientes a los datos obtenidos para nuestro dataset en el análisis. En primer lugar analizaremos las diferencias respecto al test y posteriormente respecto al train para comprobar si existe sobreajuste.

```

#leemos la tabla con los errores medios de test
resultados <- read.csv("clasif_test_alumnos.csv")
tablatst <- cbind(resultados[,2:dim(resultados)[2]])
colnames(tablatst) <- names(resultados)[2:dim(resultados)[2]]
rownames(tablatst) <- resultados[,1]

#leemos la tabla con los errores medios de entrenamiento
resultados <- read.csv("clasif_train_alumnos.csv")
tablatra <- cbind(resultados[,2:dim(resultados)[2]])
colnames(tablatra) <- names(resultados)[2:dim(resultados)[2]]
rownames(tablatra) <- resultados[,1]

```

Wilcoxon's Test (Datos Test)

En primer lugar aplicaremos el test de Wilcoxon, el cuál ordena por las diferencias para los rankings. Como estamos en clasificación no tenemos que normalizar el error.

```

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

##   V
## 94
Rmenos

##   V
## 116
pvalue

## [1] 0.7011814

```

Según este valor tan elevado, no podemos afirmar que existan diferencias significativas entre los dos algoritmos ya que no tenemos una alta confianza.

Friedman's Test (Datos Test)

```

test_friedman <- friedman.test(as.matrix(tablatst))
test_friedman

##
## Friedman rank sum test
##
## data: as.matrix(tablatst)
## Friedman chi-squared = 0.3, df = 2, p-value = 0.8607

```

Con esto no podemos afirmar que existan diferencias significativas, ya que la confianza que nos ha dado es del 14%.

Holm's Test (Datos Test)

A continuación aplicaremos el test de post-hoc 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 1 -
## 3 1 1
##
## P value adjustment method: holm

```

Los valores del p-value en todos los casos son 1, por tanto la confianza de que haya diferencias significativas en cada caso es del 0% por lo que no se puede afirmar esto.

Una vez hecho esto hemos comprobado las diferencias en cuanto a los datos del conjunto de datos de test. A continuación vamos a estudiar el conjunto de datos de train para ver los resultados y poder detectar un posible sobreaprendizaje.

Wilcoxon's Test (Datos Train)

En primer lugar aplicaremos el test de Wilcoxon, el cuál ordena por las diferencias para los rankings. Como estamos en clasificación no tenemos que normalizar el error.

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

##   V
## 125
Rmenos

##   V
## 85
pvalue

## [1] 0.474905
```

Tenemos un 53% de confianza de que entre ambos algoritmos existan diferencias significativas, lo cual es una confianza muy baja y no permite afirmarlo con seguridad. ### Friedman's Test (Datos Train)

```
test_friedman <- friedman.test(as.matrix(tablatra))
test_friedman

##
##  Friedman rank sum test
##
## data: as.matrix(tablatra)
## Friedman chi-squared = 0.9, df = 2, p-value = 0.6376
```

Al tener una confianza tan baja no podemos afirmar con seguridad de que existan diferencias significativas en los algoritmos.

Holm's Test (Datos Train)

A continuación aplicaremos el test de post-hoc Holm.

```
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 0.95 -  
## 3 0.95 0.53  
##  
## P value adjustment method: holm
```

Según esto podemos afirmar que el algoritmo lda y el algoritmo qda son mejores que el algoritmo qda, no pudiendo asegurar que existan diferencias significativas entre lda y qda.