Proyecto Final Introducción a la Ciencia de Datos

Carmen Biedma Rodriguez

26 de noviembre de 2018

```
library(ggplot2)
library(reshape2)
library(car)
```

```
## Loading required package: carData
```

```
library(corrplot)
```

```
## corrplot 0.84 loaded
```

```
require(ISLR)
```

```
## Loading required package: ISLR
```

```
require(MASS)
```

```
## Loading required package: MASS
```

```
require(kknn)
```

```
## Loading required package: kknn
```

```
library(class)
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 3.5.2
```

```
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

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

1.Regresión: Forest Fires

A. ANÁLISIS DE DATOS

En primer lugar vamos a crear un data frame con los datos. Primero con el comando read.csv leeré los datos y a continuación le asignaré un nombre a cada una de las variables.

```
fires <- read.csv("./forestFires/ForestFires.dat", comment.char="@")
names(fires) <- c("X","Y","Month","Day","FFMC","DMC","DC","ISI","Temp","RH","Wind","Rain","Area")
```

```
str(fires)
```

```
## 'data.frame':    516 obs. of  13 variables:
##  $ X    : int  7 4 8 6 4 4 7 6 2 6 ...
```

```
##  $ Y    : int  4 5 6 3 3 4 4 5 4 5 ...
##  $ Month: int  3 9 3 9 8 8 10 4 9 9 ...
##  $ Day  : int  1 6 7 4 7 6 5 4 6 1 ...
##  $ FFMC : num  90.1 92.5 89.3 92.8 81.6 90.2 90 81.5 93.4 90.9 ...
##  $ DMC  : num  39.7 88 51.3 119 56.7 ...
##  $ DC   : num  86.6 698.6 102.2 783.5 665.6 ...
##  $ ISI  : num  6.2 7.1 9.6 7.5 1.9 8.9 8.7 2.7 8.1 7 ...
##  $ Temp : num  16.1 20.3 11.5 18.9 27.8 18.4 11.3 5.8 28.6 21.3 ...
##  $ RH   : int  29 45 39 34 32 42 60 54 27 42 ...
##  $ Wind : num  3.1 3.1 5.8 7.2 2.7 6.7 5.4 5.8 2.2 2.2 ...
##  $ Rain : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Area : num  1.75 0 7.19 34.36 6.44 ...
```

Como vemos con el comando str, tenemos un dataframe con 516 observaciones y 13 variables, algunas enteras y otras numéricas. Todas estas variables son características determinadas de ciertas condiciones que se miden cuando hay un incendio. El objetivo es, a partir de dichas variables, estimar el área del fuego. Por tanto la variable que queremos estimar es el Áres.

Vamos a ver qué significa cada una de las variables que posee nuestro dataset:

- **X**: Coordenada X en el parque de Montesino
- **Y**: Coordenada Y en el parque se Monterino
- **Month**: Mes en el que ocurrió el incendio
- **Day**: Dia de la semana en el que ocurrió el incendio
- **FFMC (Fine Fuel Moistrure Code)**: Representa la profundidad de la capa superior de la tierra, que contiene basura y residuos de fuel.
- **DMC (Duff Moistrure Code)**: Proporciona la profundidad de la capa de materia orgánica en descomposición, justo la que hay debajo de la medida con el FFMC.
- **DC (Drought Code)**: Profundidad de la capa de materia orgánica ya descompuesta. Se sitúa debajo de la no descompuesta.
- **ISI (Initial Spread Index)**: Es la combinación de las variables FFMC y la velocidad del viento.
- **Temp**: Temperatura en grados Celsius.
- **RH (Relative Humidity)**: Humedad relativa
- **Wind**: Velocidad del viento medida en km/h.
- **Rain**: Lluvia medida en mm/m^2
- **Y**: Variable de salida. Es el área del incendio.

Una de las primeras tareas que hay que realizar cuando vamos a hacer un análisis de los datos es ver si tenemos valores perdidos y tratarlos como sea conveniente. Con el siguiente comando podemos comprobar si tenemos valores perdidos.

```
which(complete.cases(fires) == FALSE)
```

```
## integer(0)
```

Como podemos observar, el comando complete.cases(fires) nos devuelve un vector de booleanos poniendo a TRUE las filas que no tienen datos perdidos, es decir, que están completas. En el caso en el que haya alguna variable sin especificar en una fila, pondrá valor FALSE en dicha posición. Si hacemos una selección de los elementos de dicho vector que figuran como FALSE (filas no completas), vemos que nos devuelve un entero con valor 0. Ésto quiere decir que en nuestro dataset no hay valores perdidos por lo que no tendremos que tratarlos.

A continuación comprobaremos si hay valores repetidos, es decir, dos o más observaciones dentro del dataset que proporcionan la misma información.

```r
which(duplicated(fires) == TRUE)
```

```
## [1] 173 454 493 506
```

En este caso vemos que sí nos devuelve 4 filas que están duplicadas. Para eliminarlas simplemente usaremos el comando unique.

```r
dim(fires)
```

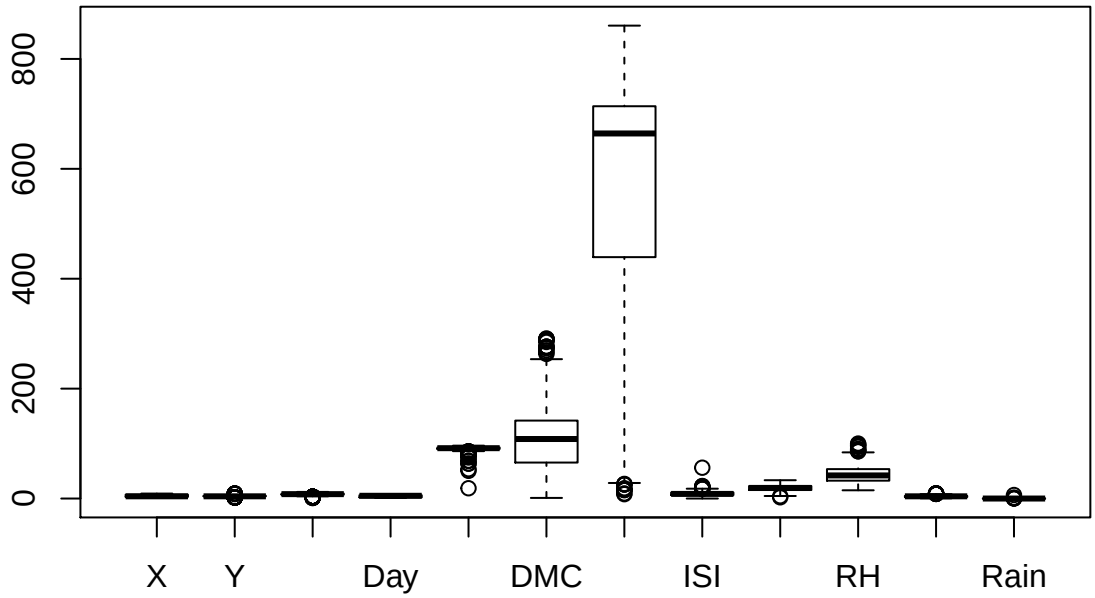
```
## [1] 516  13
```

```r
fires = unique(fires)
dim(fires)
```

```
## [1] 512  13
```

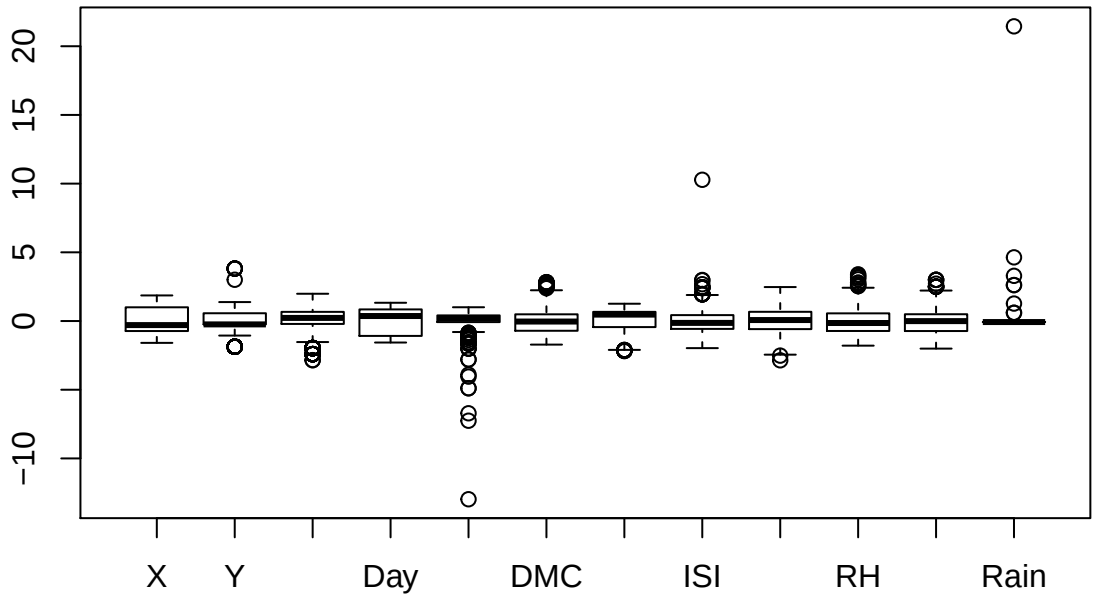
A continuación, realizaremos un boxplot con los datos para ver gráficamente el resumen de 5 valores de nuestros predictores.

```r
boxplot(fires[,-dim(fires)[2]])
```



Si observamos bien el gráfico, tenemos una variable que hace que no podamos observar bien los datos (DC) ya que la escala de sus valores es mucho mas grande que la de los demás. Es por eso que se quedan todos concentrados abajo y no podemos apreciar bien la información. Para solucionar éste problema escalaremos y centraremos los datos.

```r
boxplot(scale(fires[,-dim(fires)[2]]),scale=TRUE,center=TRUE)
```



Ahora podemos ver los datos algo mejor, pero debido a los outlayers, las cajas que contienen los demás datos se han quedado concentradas en el centro. Estos valores habría que tratarlos para determinar si son anomalías o no aunque en éste proyecto dicho trabajo no se realice ya que puede ayudar bastante en el estudio de los datos. De igual forma, es muy difícil sacar información de este boxplot, por lo que para hablar de medias y medianas usaremos el resumen de los cinco valores que genera el comando summary.

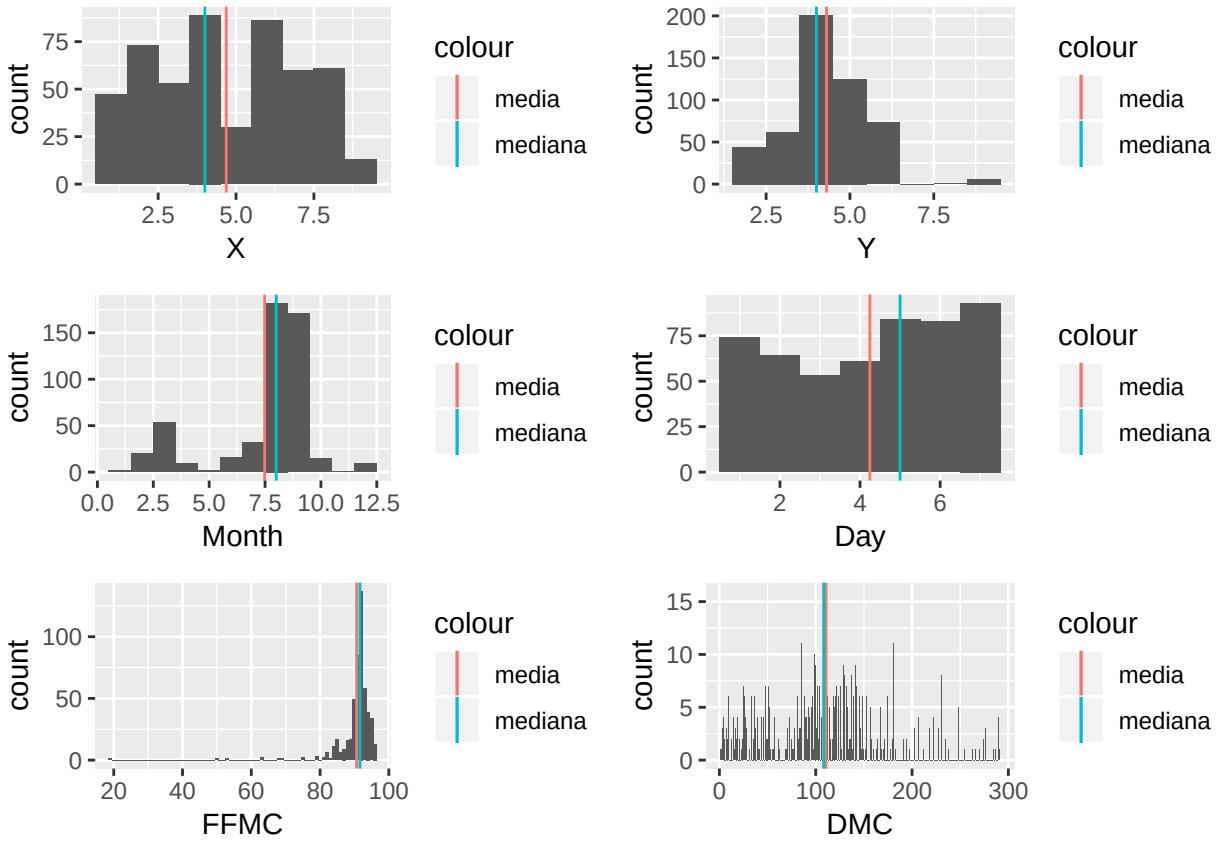
```
summary(fires)
```

```
##        X               Y              Month             Day       
##  Min.   :1.000   Min.   :2.000   Min.   : 1.000   Min.   :1.000  
##  1st Qu.:3.000   1st Qu.:4.000   1st Qu.: 7.000   1st Qu.:2.000  
##  Median :4.000   Median :4.000   Median : 8.000   Median :5.000  
##  Mean   :4.686   Mean   :4.303   Mean   : 7.482   Mean   :4.246  
##  3rd Qu.:7.000   3rd Qu.:5.000   3rd Qu.: 9.000   3rd Qu.:6.000  
##  Max.   :9.000   Max.   :9.000   Max.   :12.000   Max.   :7.000  
##       FFMC             DMC              DC              ISI        
##  Min.   :18.70   Min.   :  1.10   Min.   :  7.9   Min.   : 0.000  
##  1st Qu.:90.20   1st Qu.: 67.03   1st Qu.:440.1   1st Qu.: 6.400  
##  Median :91.60   Median :108.30   Median :664.4   Median : 8.400  
##  Mean   :90.64   Mean   :110.67   Mean   :548.6   Mean   : 9.027  
##  3rd Qu.:92.90   3rd Qu.:141.57   3rd Qu.:713.9   3rd Qu.:11.000  
##  Max.   :96.20   Max.   :291.30   Max.   :860.6   Max.   :56.100  
##       Temp             RH              Wind            Rain          
##  Min.   : 2.20   Min.   : 15.00   Min.   :0.400   Min.   :0.00000  
##  1st Qu.:15.47   1st Qu.: 32.75   1st Qu.:2.700   1st Qu.:0.00000  
##  Median :19.30   Median : 42.00   Median :4.000   Median :0.00000  
##  Mean   :18.88   Mean   : 44.35   Mean   :4.011   Mean   :0.02187  
```

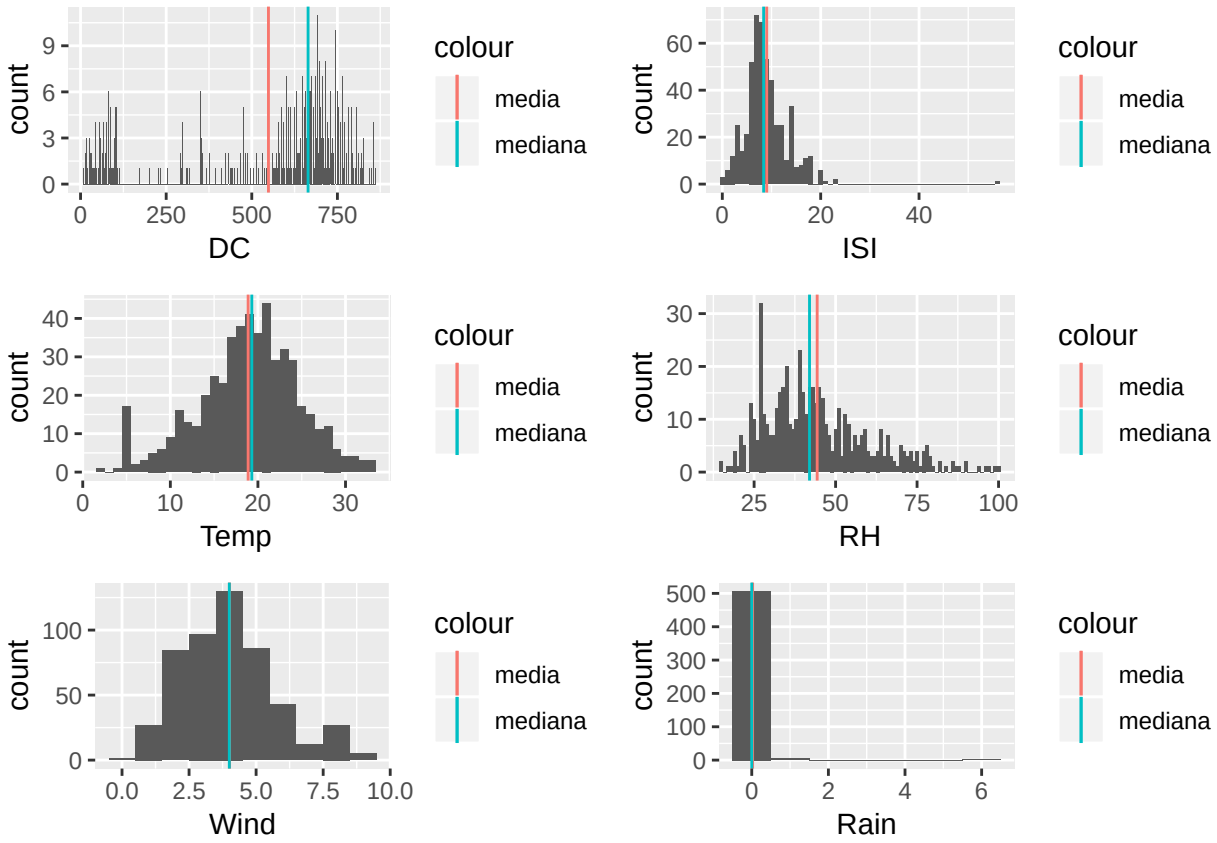
```
##  3rd Qu.:22.80   3rd Qu.: 53.25   3rd Qu.:4.900   3rd Qu.:0.00000
##  Max.   :33.30   Max.   :100.00   Max.   :9.400   Max.   :6.40000
##       Area
##  Min.   :   0.000
##  1st Qu.:   0.000
##  Median :   0.530
##  Mean   :  12.779
##  3rd Qu.:   6.548
##  Max.   :1090.840
```

Si observamos la media y la mediana coinciden mas o menos en el mismo valor para cada una de las variables. Ésto puede decirnos que los datos están distribuidos según una distribución normal en todas las variables, exceptuando el Y que tiene una media muy diferente a su mediana, pero tendríamos que comprobarlo viendo los histogramas de cada una de las variables.

```
par(mfrow=c(2,3))
p1<- ggplot(fires, aes(X)) + geom_histogram(binwidth = 1) + geom_vline(aes(xintercept = mean(fires[,"X"]

p2<- ggplot(fires, aes(Y)) + geom_histogram(binwidth = 1) + geom_vline(aes(xintercept = mean(fires[,"Y"]


p3<- ggplot(fires, aes(Month)) + geom_histogram(binwidth = 1) + geom_vline(aes(xintercept = mean(fires[


p4<- ggplot(fires, aes(Day)) + geom_histogram(binwidth = 1) + geom_vline(aes(xintercept = mean(fires[,"


p5<- ggplot(fires, aes(FFMC)) + geom_histogram(binwidth = 1) + geom_vline(aes(xintercept = mean(fires[,


p6<- ggplot(fires, aes(DMC)) + geom_histogram(binwidth = 1) + geom_vline(aes(xintercept = mean(fires[,"


p7<- ggplot(fires, aes(DC)) + geom_histogram(binwidth = 1) + geom_vline(aes(xintercept = mean(fires[,"DC


p8<- ggplot(fires, aes(ISI)) + geom_histogram(binwidth = 1) + geom_vline(aes(xintercept = mean(fires[,"


p9<- ggplot(fires, aes(Temp)) + geom_histogram(binwidth = 1) + geom_vline(aes(xintercept = mean(fires[,


p10<- ggplot(fires, aes(RH)) + geom_histogram(binwidth = 1) + geom_vline(aes(xintercept = mean(fires[,"


p11<- ggplot(fires, aes(Wind)) + geom_histogram(binwidth = 1) + geom_vline(aes(xintercept = mean(fires[


p12<- ggplot(fires, aes(Rain)) + geom_histogram(binwidth = 1) + geom_vline(aes(xintercept = mean(fires[

grid.arrange(p1,p2,p3,p4,p5,p6)
```



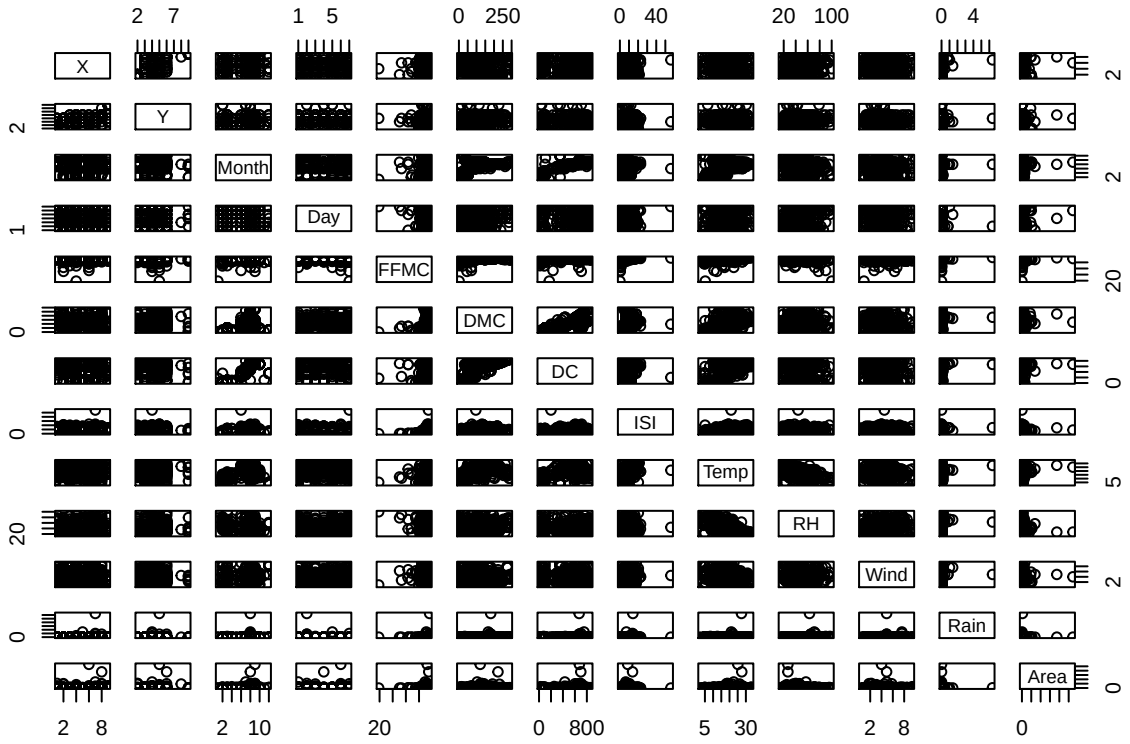

```
grid.arrange(p7,p8,p9,p10,p11,p12)
```



Si observamos los histogramas de las variables, podemos decir que la mayoria no tienen una distribución normal, exceptuando una variable Wind que parece serlo ya que la mediana y la media de los datos se encuentran en el mismo valor practicamente, por lo que a su derecha y a su izquierda tenemos aproximadamente la misma cantidad de datos. Es por ello que ésta variable puede ser buena a la hora de predecir nuestro modelo.

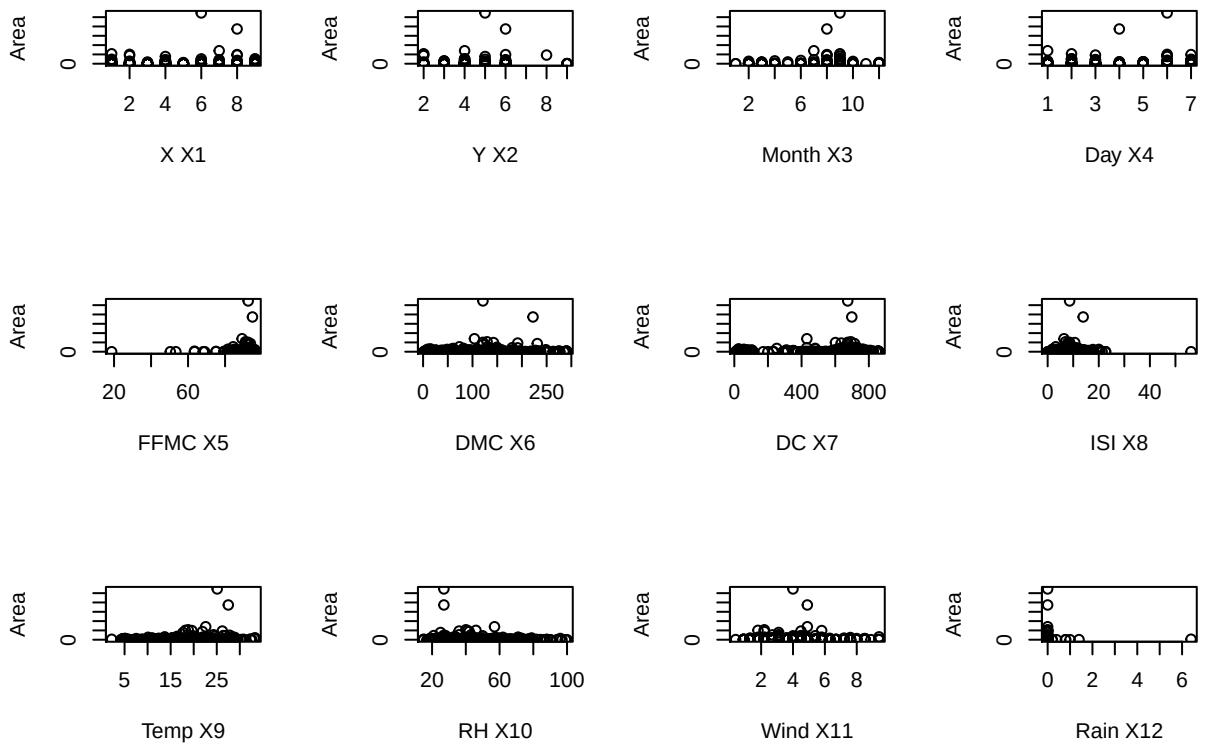
Para ver cuales son las variables que tienen más relación unas con otras, vamos a dibujar un gráfico que compare todas con todas. Para ello usaremos el comando pairs y veremos cuáles parecen seguir una distribución concreta.

```
pairs(fires)
```



Como vemos, hay tantas variables que a simple vista no podemos observar ningún patrón entre ninguna de ellas. Vamos a representar a parte solamente las relaciones entre las variables de entrada con la de salida que son las que nos interesan en un principio para la regresión lineal simple. En el siguiente trozo de codigo pintaremos un scatterplot para cada una de las variables en función de la variable de salida que queremos estimar.

```
X9 <- fires
plotY<-function (x,y) {
plot(X9[,y]~X9[,x], xlab=paste(names(X9)[x]," X",x,sep=""), ylab=names(X9)[y])
}
par(mfrow=c(3,4))
x <-sapply(1:(dim(X9)[2]-1), plotY, dim(X9)[2])
```




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

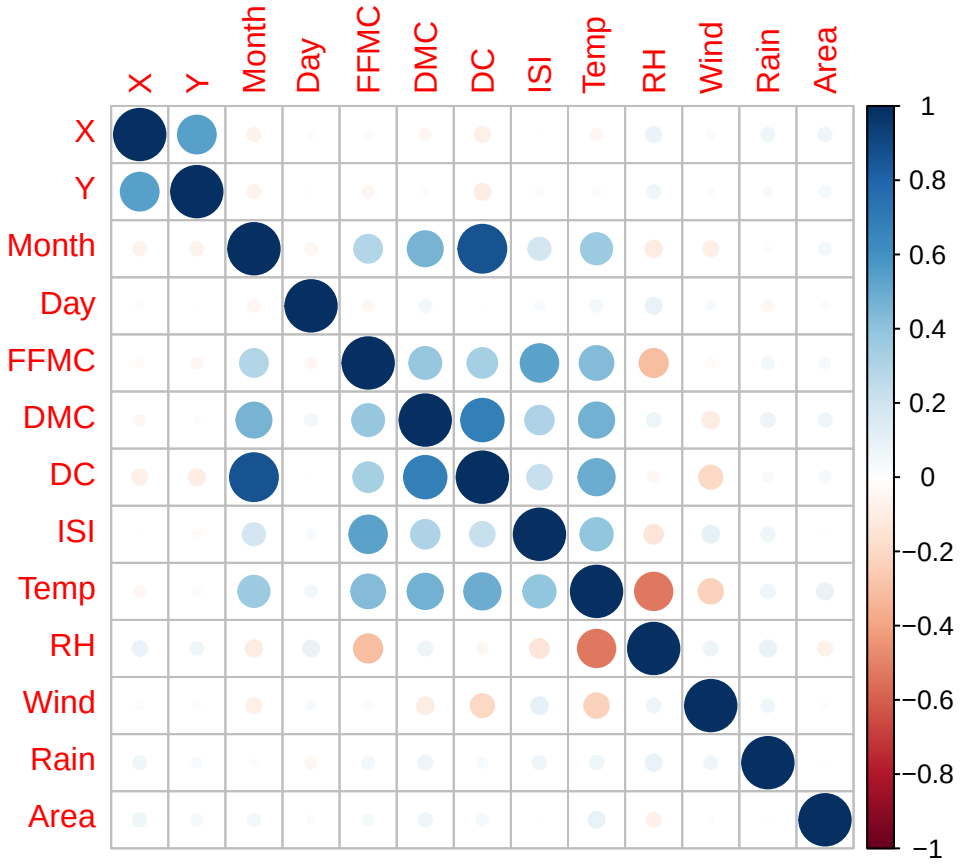
En estos últimos scatterplot observamos que ya sí tenemos algunas variables que son mejores que otras, aunque ninguna de ellas parece ser buena de cara a realizar un modelo lineal. En primer lugar, las que menos se podrían ajustar a un modelo lineal son las 4 primeras (de la X1 a la X4), por tanto estas serán las primeras que eliminaremos de nuestro estudio. No obstante, con el comando cor podemos ver las correlaciones que hay entre las variables y aquellas que más correlación tengan con la variable de salida serán las que mas nos interesen para nuestros modelos. Además, podemos mostrar un gráfico que nos ilustra de forma visual la correlación entre las variables.

```
cor(fires)
```

```
##                 X           Y       Month         Day        FFMC
## X       1.000000000  0.541350921 -0.06506496 -0.018487428 -0.02034651
## Y       0.541350921  1.000000000 -0.06410448 -0.003944934 -0.04589455
## Month -0.065064956 -0.064104479  1.00000000 -0.049664118  0.29347109
## Day   -0.018487428 -0.003944934 -0.04966412  1.000000000 -0.04184813
## FFMC  -0.020346514 -0.045894548  0.29347109 -0.041848135  1.00000000
## DMC   -0.041240000  0.013637205  0.46422827  0.057798413  0.38582459
## DC    -0.085659302 -0.096416245  0.86784421  0.001170117  0.33319931
## ISI    0.004741404 -0.024040966  0.18635708  0.034162453  0.53223068
## Temp  -0.049601050 -0.022833095  0.36871661  0.051509821  0.43165731
## RH     0.083707623  0.062485372 -0.10011144  0.095602840 -0.30081929
## Wind   0.020902015 -0.019570123 -0.08518920  0.030265956 -0.02920771
## Rain   0.065109098  0.033197918  0.01328836 -0.048098586  0.05681634
## Area   0.065955121  0.047129772  0.05651263  0.021020324  0.04014855
##                DMC           DC          ISI        Temp          RH
```

```
## X     -0.04124000 -0.085659302  0.004741404 -0.04960105  0.08370762
## Y      0.01363721 -0.096416245 -0.024040966 -0.02283309  0.06248537
## Month  0.46422827  0.867844206  0.186357084  0.36871661 -0.10011144
## Day    0.05779841  0.001170117  0.034162453  0.05150982  0.09560284
## FFMC   0.38582459  0.333199310  0.532230675  0.43165731 -0.30081929
## DMC    1.00000000  0.681385636  0.308766806  0.47016789  0.07259980
## DC     0.68138564  1.000000000  0.229403157  0.49719131 -0.04394070
## ISI    0.30876681  0.229403157  1.000000000  0.39481360 -0.13345474
## Temp   0.47016789  0.497191313  0.394813596  1.00000000 -0.52872845
## RH     0.07259980 -0.043940701 -0.133454736 -0.52872845  1.00000000
## Wind  -0.10576975 -0.203243734  0.106838273 -0.22767174  0.07143245
## Rain   0.07564316  0.035893896  0.067615541  0.06961921  0.09960520
## Area   0.06972834  0.048436772  0.009165271  0.09740918 -0.07537439
##              Wind         Rain         Area
## X      0.02090201  0.065109098  0.065955121
## Y     -0.01957012  0.033197918  0.047129772
## Month -0.08518920  0.013288363  0.056512628
## Day    0.03026596 -0.048098586  0.021020324
## FFMC  -0.02920771  0.056816337  0.040148547
## DMC   -0.10576975  0.075643159  0.069728337
## DC    -0.20324373  0.035893896  0.048436772
## ISI    0.10683827  0.067615541  0.009165271
## Temp  -0.22767174  0.069619211  0.097409176
## RH     0.07143245  0.099605202 -0.075374387
## Wind   1.00000000  0.061478466  0.012719802
## Rain   0.06147847  1.000000000 -0.007293906
## Area   0.01271980 -0.007293906  1.000000000
```

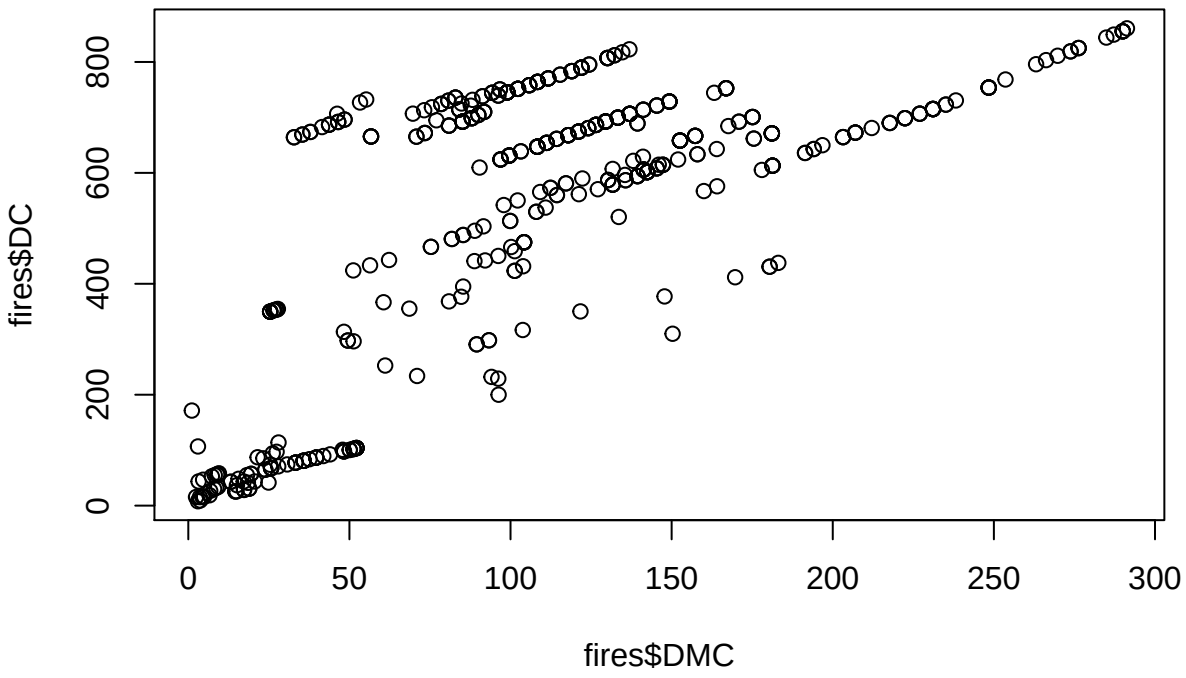
```
y = cor(fires[,1:13])
corrplot(y)
```



En la última de las tablas de correlaciones generadas podemos ver la correlación de la variable de salida (Area) con todas las demás variables. Como ya sabemos, una correlación cercana a 1 significa que las variables están muy correladas, de manera directa si el valor es positivo y de manera inversa si es negativo. En nuestro caso, ninguna de las variables nos da una correlación alta con la variable de salida, por lo que lo más probable sea que un modelo simple no nos dé buenos resultados. Las correlaciones entre variables podriamos comprobarlas con un scatteplot de ambas variables.

Analizando un poco el resultado obtenido las variables que más correlación tienen son DC y Month con un 0.87. Esta correlación debe ser una casualidad de los datos que tenemos porque no tiene sentido que al aumentar en los meses del año, aumente la profundidad de la capa de materia descompuesta en un punto concreto. El segundo par de variables con más correlación es el DC y DMC . En este caso la correlación tiene sentido ya que uno es la cantidad de materia orgánica descompuesta y otro la cantidad de materia orgánica en descomposición. Tiene sentido que a mayor cantidad de materia orgánica en descomposición en un punto, tengamos más descompuesta debajo. Veamos en un gráfico dicha relación.

```
plot(fires$DC~fires$DMC)
```



B. REGRESION

R-1.Regresión lineal simple

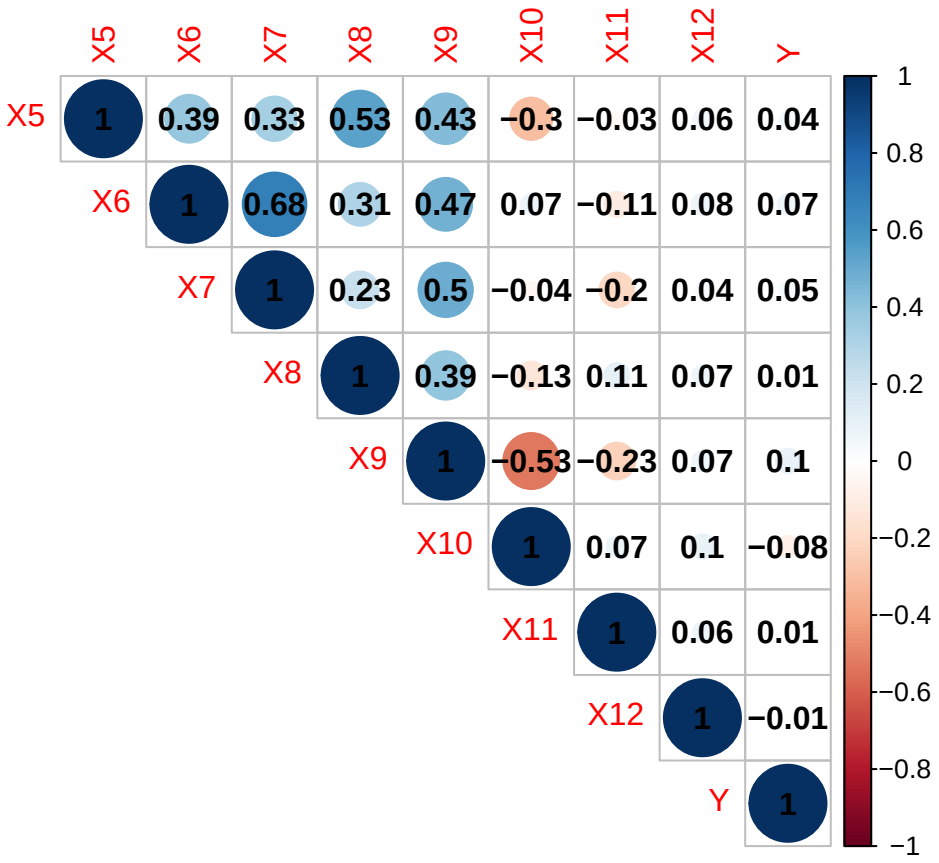
Antes de comenzar con el proceso de regresión, voy a cambiarle los nombres a las variables para que sean más fáciles de comprender. Las variables descriptoras se llamarán X1..Xn y la variable de salida Y.

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

```
##   X1 X2 X3 X4   X5    X6    X7  X8   X9 X10 X11 X12     Y
## 1  7  4  3  1 90.1  39.7  86.6 6.2 16.1  29 3.1   0  1.75
## 2  4  5  9  6 92.5  88.0 698.6 7.1 20.3  45 3.1   0  0.00
## 3  8  6  3  7 89.3  51.3 102.2 9.6 11.5  39 5.8   0  7.19
## 4  6  3  9  4 92.8 119.0 783.5 7.5 18.9  34 7.2   0 34.36
## 5  4  3  8  7 81.6  56.7 665.6 1.9 27.8  32 2.7   0  6.44
## 6  4  4  8  6 90.2  96.9 624.2 8.9 18.4  42 6.7   0  0.00
```

Voy a añadir aquí la tabla de correlaciones para tenerla más a mano. Ésta vez, además de hacer un gráfico solo con los circulos, pondré el valor de la correlacón en cada una de las variables.

```
y = cor(fires[,5:dim(fires)[2]])
corrplot(y,type="upper",addCoef.col = "Black")
```



En primer lugar, vamos a seleccionar las 5 variables que creemos que pueden ser mejores para realizar una regresión lineal simple. Para ello cogeré aquellas 5 con mayor correlación con la variable de salida, ya que deberían ser las que mejor funcionen en éste tipo de modelo. Por lo tanto, las que se utilizarán serán **X9, X10,X6,X7 y X5.**

Para cada uno de los modelos que realicemos, tendremos que ver si son mejores o peores que otros. Para ello, utilizaremos en primer lugar el R-squared (para regresiones lineales simples) y el Adjusted R-squared (para modelos múltiples), que nos determina la bondad del ajuste, cuanto mayor sea dicho valor mejor será el modelo. Asumiremos que un modelo es bueno cuando su Rsquared sea mayor que 0.8 aproximadamente.

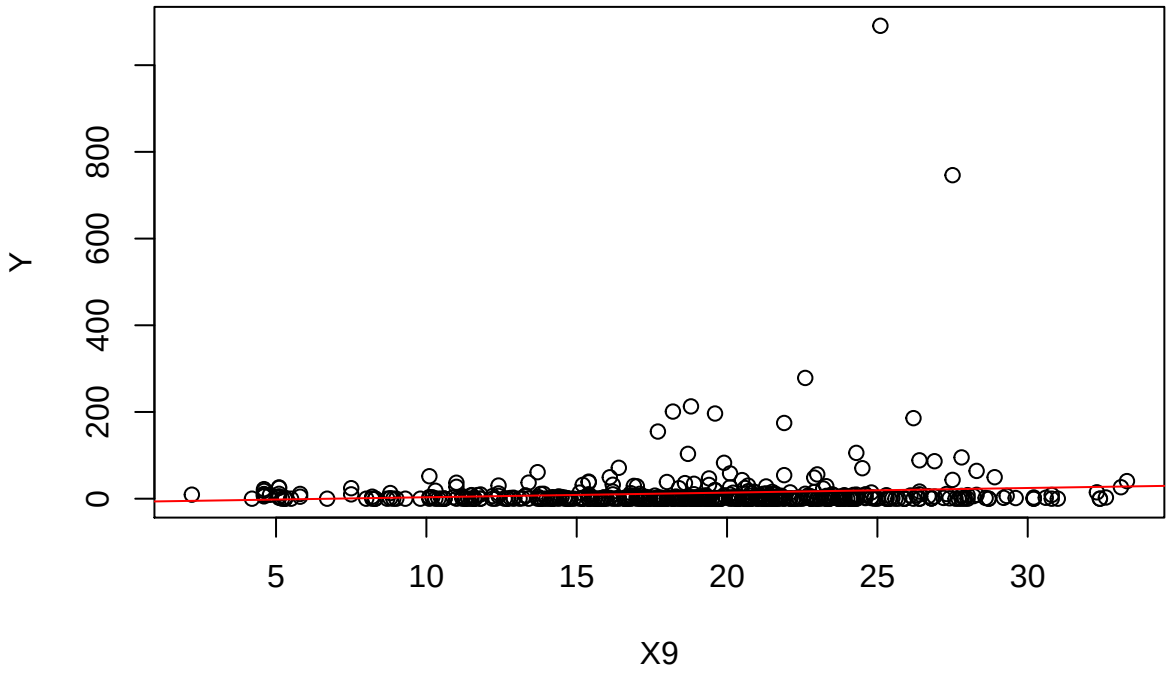
El prodecimiento para crear los modelos será el siguiente: primero generaremos el modelo con el comando lm, al que le pasaremos como argumento la fórmula que queremos que utilice y el conjunto de datos, a continuación veremos en el resumen el r cuadrado que obtiene para ver cuál es mejor y por ultimo dibujaremos la linea del modelo con los datos para ver como se ajusta gráficamente.

```
lineal.simple1 <-lm(Y~X9, data= fires)
summary(lineal.simple1)
```

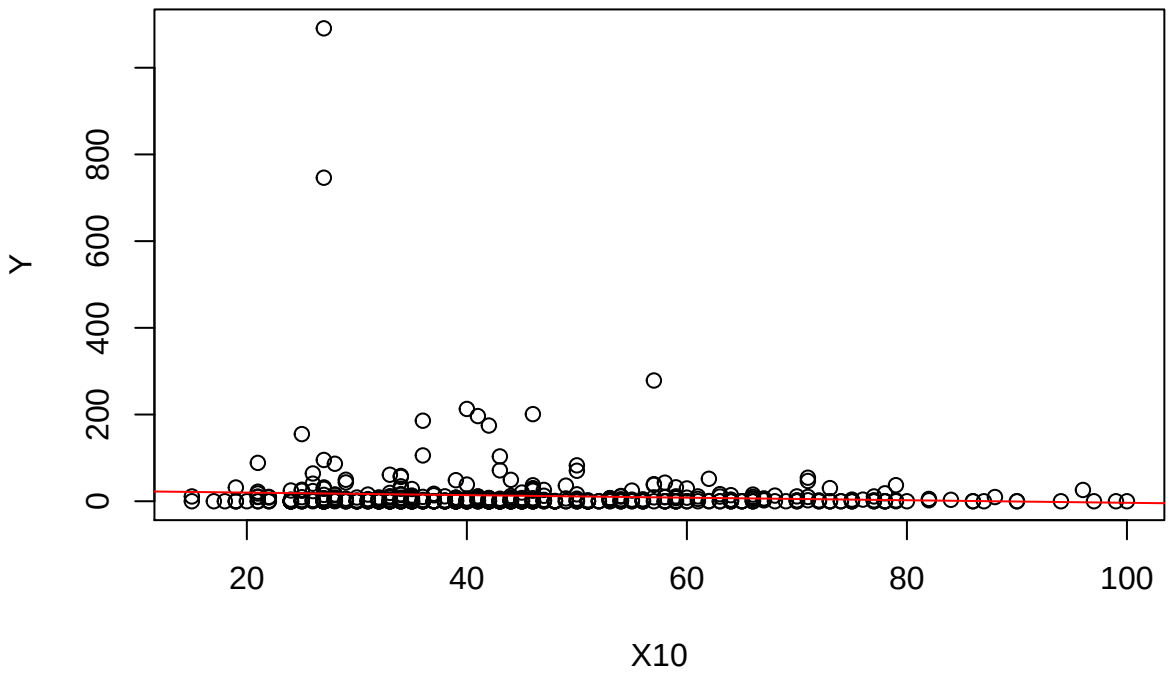
```
##
## Call:
## lm(formula = Y ~ X9, data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -27.21  -14.61  -10.26   -3.57 1071.42
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)  -7.3738     9.5416  -0.773   0.4400
## X9            1.0673     0.4829   2.210   0.0275 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63.66 on 510 degrees of freedom
## Multiple R-squared:  0.009489,   Adjusted R-squared:  0.007546
## F-statistic: 4.886 on 1 and 510 DF,  p-value: 0.02753
```

```
plot(Y~X9,fires)
abline(lineal.simple1,col="red")
```



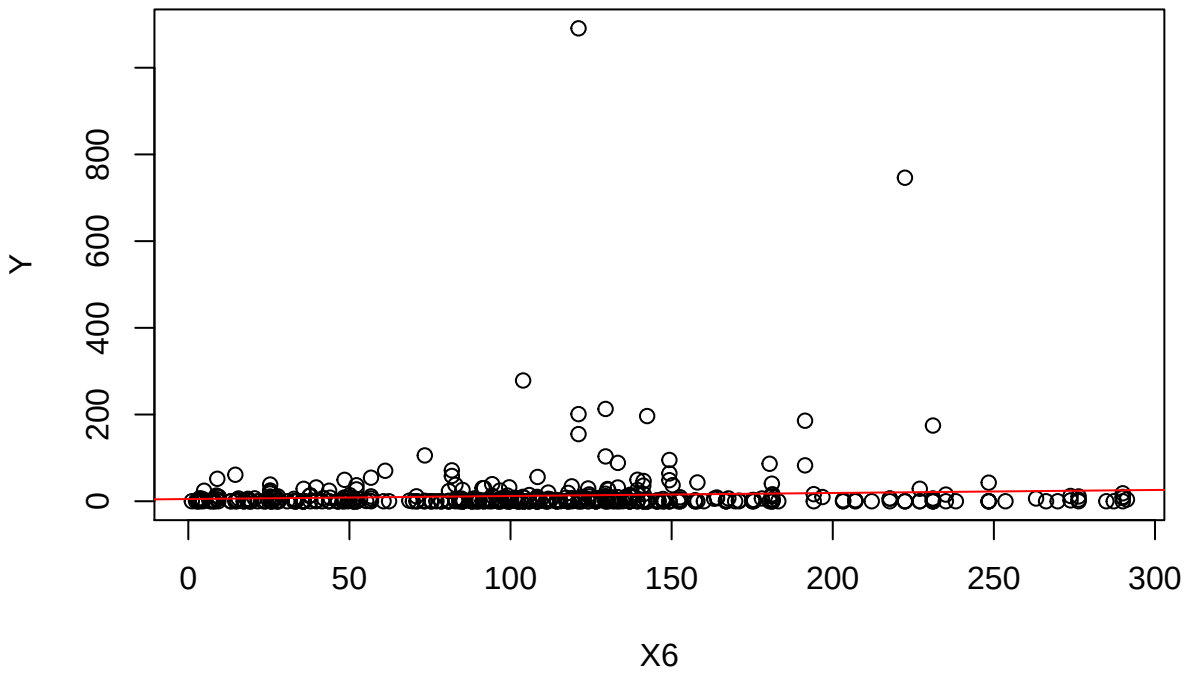

```
lineal.simple2 <-lm(Y~X10, data= fires)
plot(Y~X10,fires)
abline(lineal.simple2,col="red")
```




```r
summary(lineal.simple2)
```

```
##
## Call:
## lm(formula = Y ~ X10, data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -21.41  -14.35  -10.53   -3.59 1072.96
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  25.8248     8.1459   3.170  0.00161 **
## X10          -0.2942     0.1723  -1.707  0.08842 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63.78 on 510 degrees of freedom
## Multiple R-squared:  0.005681,   Adjusted R-squared:  0.003732
## F-statistic: 2.914 on 1 and 510 DF,  p-value: 0.08842
```

```r
lineal.simple3 <-lm(Y~X6, data= fires)
plot(Y~X6,fires)
abline(lineal.simple3,col="red")
```



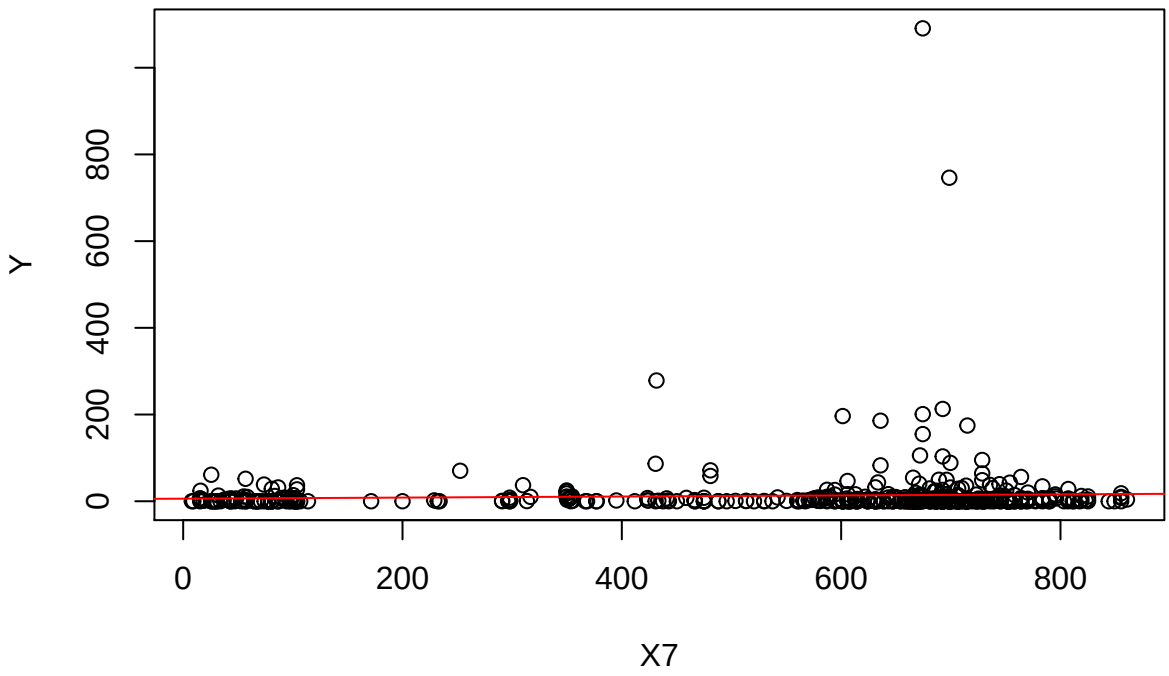

```
summary(lineal.simple3)
```

```
##
## Call:
## lm(formula = Y ~ X6, data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -25.30  -13.35  -10.06   -5.26 1077.33
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.05374    5.64809   0.895    0.371
## X6           0.06980    0.04422   1.579    0.115
##
## Residual standard error: 63.81 on 510 degrees of freedom
## Multiple R-squared:  0.004862,   Adjusted R-squared:  0.002911
## F-statistic: 2.492 on 1 and 510 DF,  p-value: 0.1151
```

```
sqrt(sum(lineal.simple3$residuals^2)/length(lineal.simple3$residuals))
```

```
## [1] 63.68589
```

```
lineal.simple4 <-lm(Y~X7, data= fires)
plot(Y~X7,fires)
abline(lineal.simple4,col="red")
```



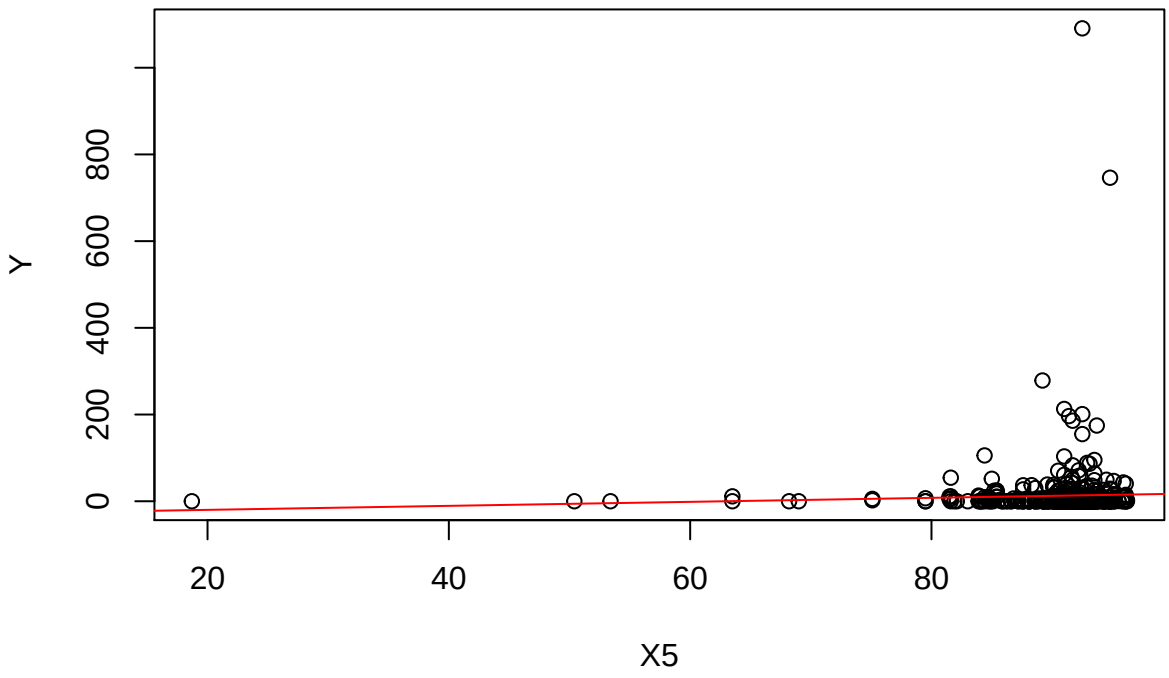

```
summary(lineal.simple4)
```

```
##
## Call:
## lm(formula = Y ~ X7, data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -16.61  -14.22  -10.95   -5.29 1076.49
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  5.92205    6.86813   0.862    0.389
## X7           0.01250    0.01141   1.095    0.274
##
## Residual standard error: 63.89 on 510 degrees of freedom
## Multiple R-squared:  0.002346,   Adjusted R-squared:  0.0003899
## F-statistic: 1.199 on 1 and 510 DF,  p-value: 0.274
```

```
sqrt(sum(lineal.simple4$residuals^2)/length(lineal.simple4$residuals))
```

```
## [1] 63.76635
```

```
lineal.simple5 <-lm(Y~X5, data= fires)
plot(Y~X5,fires)
abline(lineal.simple5,col="red")
```




```
summary(lineal.simple5)
```

```
##
## Call:
## lm(formula = Y ~ X5, data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -15.35  -13.25  -11.77   -5.82 1077.20
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -29.1486    46.2914  -0.630    0.529
## X5            0.4626     0.5098   0.907    0.365
##
## Residual standard error: 63.91 on 510 degrees of freedom
## Multiple R-squared:  0.001612,   Adjusted R-squared:  -0.0003457
## F-statistic: 0.8234 on 1 and 510 DF,  p-value: 0.3646
```

```
sqrt(sum(lineal.simple5$residuals^2)/length(lineal.simple5$residuals))
```

```
## [1] 63.78981
```

En general, podemos decir que todos los modelos simples dan resultados muy malos. Ésto lo sabemos porque los R cuadrado tienen valores extremadamente pequeños, lo que significaría que la bondad del modelo sería nula prácticamente. Si tuviesemos que elegir uno, escogeríamos el que mayor Rcuadrado tuviese, que en este caso es el modelo lineal simple formado con la variable X9. El orden que se ha seguido para realizar los modelos depende de la correlación de cada variable con la de salida. Es por ello que el primer modelo es el

que mejor resultados proporciona y por tanto el que nos quedaremos en esta fase del estudio.

Otra cosa que podemos sacar de los modelos establecidos es que los p-valor de las variables que hemos utilizado para crearlos son muy grandes. Esto quiere decir que ninguna de ellas influye mucho en el modelo establecido, por lo que deberíamos de considerar la posibilidad de agregar más variables o realizar interacciones entre ellas.

Vamos a comprobar que resultados nos proporciona la validación cruzada para el mejor modelo obtenido en regresión lineal simple. Con este test podemos ver si el modelo sobreajusta ya que tenemos el RMSE para train y para test. Si el MSE de train es mucho mejor que el de test podemos decir que nuestro modelo sobreajusta.

Además, usaremos la validación cruzada para ver cuál es el mejor modelo que obtenemos. Dicho modelo será el que menos sobreajuste y menor MSE tenga.

```
nombre<-"./forestFires/forestFires"
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~X9,x_tra)
yprime=predict(fitMulti,test)
sum(abs(test$Y-yprime)^2)/length(yprime) ##MSE
}

lmSimpleMSEtrain<-mean(sapply(1:5,run_lm_fold,nombre,"train"))
lmSimpleMSEtest<-mean(sapply(1:5,run_lm_fold,nombre,"test"))
print(paste("MSE Train:",lmSimpleMSEtrain))
```

```
## [1] "MSE Train: 4009.79173859998"
```

```
print(paste("MSE Test:",lmSimpleMSEtest))
```

```
## [1] "MSE Test: 4042.86864279946"
```

R-2. Regresión lineal múltiple y no linealidad.

Para comenzar con la regresión lineal múltiple vamos a realizar un modelo que tenga en cuenta todas las variables. A partir de él, veremos cuáles son las que más nos interesan observando los p-value de cada una e iremos quitando las que menos aporten a nuestro modelo, es decir, las que mayor pvalor tengan.

```
lineal.multiple1 <-lm(Y~., data= fires)
summary(lineal.multiple1)
```

```
##
```

```
## Call:
## lm(formula = Y ~ ., data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -36.13  -16.02   -8.31    0.01 1063.23
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -16.52390   63.89260  -0.259    0.796
## X1            1.97366    1.46255   1.349    0.178
## X2            0.38005    2.77857   0.137    0.891
## X3            2.76585    2.80489   0.986    0.325
## X4            0.70336    1.39499   0.504    0.614
## X5           -0.05636    0.66704  -0.084    0.933
## X6            0.09179    0.07159   1.282    0.200
## X7           -0.02951    0.03232  -0.913    0.362
## X8           -0.71289    0.77635  -0.918    0.359
## X9            0.88235    0.80749   1.093    0.275
## X10          -0.20510    0.24364  -0.842    0.400
## X11           1.20052    1.71580   0.700    0.484
## X12          -2.99849    9.76198  -0.307    0.759
##
## Residual standard error: 63.9 on 499 degrees of freedom
## Multiple R-squared:  0.02364,    Adjusted R-squared:  0.0001585
## F-statistic: 1.007 on 12 and 499 DF,  p-value: 0.4412
```

Como podemos observar, el R cuadrado ajustado es muy pequeño, por lo que el modelo con todas las variables, al igual que los simples, es muy malo. Vamos a ver si podemos mejorarlo eliminando variables.

El criterio que seguiremos para eliminar variables será quitar la que más p-valor tenga, ya que es el que menos influye dentro del modelo. El p-valor nos dice cuanto influye una variable en el modelo que hemos realizado. Cuánto más grande es dicho valor, menos importancia tiene la variable asociada y cuanto más pequeño más aporta a nuestro modelo.

La primera que eliminaremos será el X5. Si observamos el R cuadrado ajustado vemos que ha mejorado algo pero no mucho, por lo que nos quedaremos con este modelo para seguir el estudio.

```
lineal.multiple2 <-lm(Y~.-X5, data= fires)
summary(lineal.multiple2)
```

```
##
## Call:
## lm(formula = Y ~ . - X5, data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -36.15  -15.93   -8.31    0.17 1063.21
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -21.46831   25.62112  -0.838    0.402
## X1            1.96926    1.46017   1.349    0.178
## X2            0.39053    2.77304   0.141    0.888
## X3            2.74994    2.79579   0.984    0.326
## X4            0.70880    1.39213   0.509    0.611
```

```
## X6            0.09047    0.06980   1.296    0.196
## X7           -0.02944    0.03228  -0.912    0.362
## X8           -0.74122    0.69952  -1.060    0.290
## X9            0.88352    0.80657   1.095    0.274
## X10          -0.20004    0.23593  -0.848    0.397
## X11           1.20666    1.71255   0.705    0.481
## X12          -3.03682    9.74174  -0.312    0.755
##
## Residual standard error: 63.84 on 500 degrees of freedom
## Multiple R-squared:  0.02362,    Adjusted R-squared:  0.002144
## F-statistic:   1.1 on 11 and 500 DF,  p-value: 0.3591
```

La siguiente que eliminaremos según el criterio escogido será la variable con mayor pvalor del último modelo, X12.

```
lineal.multiple3 <-lm(Y~.-X5-X12, data= fires)
summary(lineal.multiple3)
```

```
##
## Call:
## lm(formula = Y ~ . - X5 - X12, data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -35.83  -15.98   -8.28   -0.01 1063.22
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -20.30369   25.32442  -0.802    0.423
## X1            1.94797    1.45726   1.337    0.182
## X2            0.39582    2.77049   0.143    0.886
## X3            2.75490    2.79322   0.986    0.324
## X4            0.74519    1.38597   0.538    0.591
## X6            0.09058    0.06973   1.299    0.195
## X7           -0.02931    0.03224  -0.909    0.364
## X8           -0.74309    0.69886  -1.063    0.288
## X9            0.84702    0.79731   1.062    0.289
## X10          -0.21221    0.23247  -0.913    0.362
## X11           1.16118    1.70479   0.681    0.496
##
## Residual standard error: 63.78 on 501 degrees of freedom
## Multiple R-squared:  0.02343,    Adjusted R-squared:  0.003942
## F-statistic: 1.202 on 10 and 501 DF,  p-value: 0.2867
```

Como seguimos mejorando el modelo, nos quedaremos con este último. La siguiente variable con mayor pvalor es X2.

```
lineal.multiple4 <-lm(Y~.-X5-X12-X2, data= fires)
summary(lineal.multiple4)
```

```
##
## Call:
## lm(formula = Y ~ . - X5 - X12 - X2, data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
##  -35.54  -15.98   -8.41    0.11 1063.35
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -19.21967   24.13746  -0.796   0.4263
## X1            2.05967    1.22860   1.676   0.0943 .
## X3            2.79533    2.77614   1.007   0.3145
## X4            0.74582    1.38461   0.539   0.5904
## X6            0.09188    0.06907   1.330   0.1840
## X7           -0.03001    0.03183  -0.943   0.3462
## X8           -0.74773    0.69742  -1.072   0.2842
## X9            0.85163    0.79588   1.070   0.2851
## X10          -0.21115    0.23212  -0.910   0.3635
## X11           1.14635    1.69996   0.674   0.5004
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63.72 on 502 degrees of freedom
## Multiple R-squared:  0.02339,    Adjusted R-squared:  0.005886
## F-statistic: 1.336 on 9 and 502 DF,  p-value: 0.2152
```

Como sigue mejorando el modelo, seguimos eliminando variables siguiendo el mismo criterio. la siguiente eliminada será X4. A continuación realizaré este proceso hasta que el R cuadrado ajustado sea menor que el anterior y el modelo empeore.

```
lineal.multiple5 <-lm(Y~.-X5-X12-X2-X4, data= fires)
summary(lineal.multiple5)
```

```
##
## Call:
## lm(formula = Y ~ . - X5 - X12 - X2 - X4, data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -33.96  -15.65   -8.89   -0.75 1064.77
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -17.34586   23.86861  -0.727   0.4677
## X1            2.03918    1.22714   1.662   0.0972 .
## X3            2.70329    2.76892   0.976   0.3294
## X6            0.09165    0.06902   1.328   0.1848
## X7           -0.02972    0.03181  -0.934   0.3506
## X8           -0.75015    0.69692  -1.076   0.2823
## X9            0.90346    0.78949   1.144   0.2530
## X10          -0.19363    0.22967  -0.843   0.3996
## X11           1.19796    1.69606   0.706   0.4803
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63.67 on 503 degrees of freedom
## Multiple R-squared:  0.02283,    Adjusted R-squared:  0.007289
## F-statistic: 1.469 on 8 and 503 DF,  p-value: 0.1658
```

```r
lineal.multiple6 <-lm(Y~.-X5-X12-X2-X4-X11, data= fires)
summary(lineal.multiple6)
```

```
##
## Call:
## lm(formula = Y ~ . - X5 - X12 - X2 - X4 - X11, data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -34.39  -15.84   -8.36   -0.49 1065.35
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -11.91491   22.58497  -0.528    0.598
## X1            2.03288    1.22650   1.657    0.098 .
## X3            3.06883    2.71877   1.129    0.260
## X6            0.09728    0.06852   1.420    0.156
## X7           -0.03466    0.03101  -1.117    0.264
## X8           -0.64838    0.68152  -0.951    0.342
## X9            0.80201    0.77592   1.034    0.302
## X10          -0.19925    0.22942  -0.868    0.386
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63.64 on 504 degrees of freedom
## Multiple R-squared:  0.02186,    Adjusted R-squared:  0.008276
## F-statistic: 1.609 on 7 and 504 DF,  p-value: 0.1303
```

```r
lineal.multiple7 <-lm(Y~.-X5-X12-X2-X4-X11-X10, data= fires)
summary(lineal.multiple7)
```

```
##
## Call:
## lm(formula = Y ~ . - X5 - X12 - X2 - X4 - X11 - X10, data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -33.74  -15.51   -8.01   -0.12 1066.43
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -26.76985   14.74525  -1.815   0.0700 .
## X1            1.93184    1.22067   1.583   0.1141
## X3            3.49068    2.67438   1.305   0.1924
## X6            0.08223    0.06628   1.241   0.2153
## X7           -0.03976    0.03044  -1.306   0.1922
## X8           -0.67916    0.68043  -0.998   0.3187
## X9            1.22980    0.59939   2.052   0.0407 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63.62 on 505 degrees of freedom
## Multiple R-squared:  0.0204, Adjusted R-squared:  0.008758
## F-statistic: 1.753 on 6 and 505 DF,  p-value: 0.107
```

En este punto del estudio, podemos ver que en algunas variables del resumen nos aparece un icono al lado. Esto quiere decir que son aquellas que más aportan al modelo y por tanto las que más nos interesan. La leyenda se puede observar debajo de las variables.

```
lineal.multiple8 <-lm(Y~.-X5-X12-X2-X4-X11-X10-X8, data= fires)
summary(lineal.multiple8)
```

```
##
## Call:
## lm(formula = Y ~ . - X5 - X12 - X2 - X4 - X11 - X10 - X8, data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -33.43  -15.05   -8.68   -1.10 1067.97
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -28.04912   14.68939  -1.909   0.0568 .
## X1            1.90303    1.22033   1.559   0.1195
## X3            3.27596    2.66570   1.229   0.2197
## X6            0.07070    0.06526   1.083   0.2792
## X7           -0.03677    0.03030  -1.214   0.2254
## X9            1.04585    0.57035   1.834   0.0673 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63.62 on 506 degrees of freedom
## Multiple R-squared:  0.01846,    Adjusted R-squared:  0.008766
## F-statistic: 1.904 on 5 and 506 DF,  p-value: 0.09212
```

```
lineal.multiple9 <-lm(Y~.-X5-X12-X2-X4-X11-X10-X8-X6, data= fires)
summary(lineal.multiple9)
```

```
##
## Call:
## lm(formula = Y ~ . - X5 - X12 - X2 - X4 - X11 - X10 - X8 - X6,
##     data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -33.68  -15.13   -8.88   -1.35 1067.08
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -25.59543   14.51621  -1.763   0.0785 .
## X1            1.94726    1.21985   1.596   0.1110
## X3            2.32121    2.51623   0.922   0.3567
## X7           -0.01792    0.02481  -0.722   0.4704
## X9            1.15000    0.56229   2.045   0.0413 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63.63 on 507 degrees of freedom
## Multiple R-squared:  0.01619,    Adjusted R-squared:  0.008426
## F-statistic: 2.086 on 4 and 507 DF,  p-value: 0.08149
```

En este punto, el modelo empeora conforme vamos eliminando variables, por lo que pararemos aquí. El modelo múltiple que nos quedaremos por tanto será el 8 (Y~.-X5-X12-X2-X4-X11-X10-X8), es decir el formado por las variables X1,X3,X6,X7y X9.

Veamos a continuación lo que obtenemos con cross validation.

```
nombre<-"./forestFires/forestFires"
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~.-X5-X12-X2-X4-X11-X10-X8,x_tra)
yprime=predict(fitMulti,test)
sum(abs(test$Y-yprime)^2)/length(yprime) ##MSE
}

lmMultipleMSEtrain<-mean(sapply(1:5,run_lm_fold,nombre,"train"))
lmMultipleMSEtest<-mean(sapply(1:5,run_lm_fold,nombre,"test"))
print(paste("Train:",lmMultipleMSEtrain))
```

```
## [1] "Train: 3971.03462324552"
```

```
print(paste("Test:",lmMultipleMSEtest))
```

```
## [1] "Test: 4031.58868701383"
```

Veamos a continuación si con modelos no lineales podemos conseguir mejores resultados. En primer lugar voy a construir un modelo no lineal usando una sola variable, la que más correlacion tiene.

```
nolineal1 <-lm(Y~X9 + I(X9^2), data= fires)
summary(nolineal1)
```

```
##
## Call:
## lm(formula = Y ~ X9 + I(X9^2), data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -36.88  -13.35   -8.71   -4.70 1070.51
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.53429   19.35969   0.441    0.660
## X9          -0.92240    2.16146  -0.427    0.670
## I(X9^2)      0.05547    0.05874   0.944    0.345
```

```
##
## Residual standard error: 63.67 on 509 degrees of freedom
## Multiple R-squared:  0.01122,    Adjusted R-squared:  0.007336
## F-statistic: 2.888 on 2 and 509 DF,  p-value: 0.05659
```

```
nolineal2 <-lm(Y~X9 + I(X9^2) + I(X9^3), data= fires)
summary(nolineal2)
```

```
##
## Call:
## lm(formula = Y ~ X9 + I(X9^2) + I(X9^3), data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -26.48  -15.14   -8.36   -2.00 1068.24
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 37.674568  34.619152   1.088    0.277
## X9          -7.399000   6.735057  -1.099    0.272
## I(X9^2)      0.458503   0.401267   1.143    0.254
## I(X9^3)     -0.007476   0.007363  -1.015    0.310
##
## Residual standard error: 63.67 on 508 degrees of freedom
## Multiple R-squared:  0.01322,    Adjusted R-squared:  0.007396
## F-statistic: 2.269 on 3 and 508 DF,  p-value: 0.07963
```

Conforme aumentamos el grado del polinomio, lo conseguimos mejorar mucho el modelo, por lo que parece que éste camino no nos va a dar buenos resultados.

Vamos a probar a realizar transformaciones logarítmicas a la misma variable.

```
nolineal3 <-lm(Y~I(log(X9)) + I(X9^2), data= fires)
summary(nolineal3)
```

```
##
## Call:
## lm(formula = Y ~ I(log(X9)) + I(X9^2), data = fires)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
##  -36.83  -13.35   -8.62   -4.60 1070.35
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 18.48772   33.42393   0.553    0.580
## I(log(X9))  -7.92671   14.63072  -0.542    0.588
## I(X9^2)      0.04373    0.02685   1.629    0.104
##
## Residual standard error: 63.66 on 509 degrees of freedom
## Multiple R-squared:  0.01144,    Adjusted R-squared:  0.007553
## F-statistic: 2.945 on 2 and 509 DF,  p-value: 0.05352
```

Con estos dos modelos no lineales vemos que no se mejora el modelo mucho. Hay muchisimos más modelos no lineales pero sería imposible sacarlos todos a mano. Para comparar por tanto nos quedaremos con el mejor modelo de regresión lineal multimple.

R-3. k-NN

Para aplicar knn lo primero que haré será normalizar los datos.

```
fires.scaled <- as.data.frame(lapply(fires[,1:13], scale, center = TRUE, scale = TRUE))
```

El primer modelo que realizaré con knn será el que tiene todas las variables. Una vez realizado el modelo calcularemos el RMSE.

```
knn1 <-kknn(Y~., fires.scaled, fires.scaled)
yprime <- knn1$fitted.values
sqrt(sum((fires.scaled$Y-yprime)^2)/length(yprime))
```

```
## [1] 0.7410208
```

Veamos si usando la fórmula del mejor modelo no lineal obtenemos mejores resultados.

```
knn2 <-kknn(Y~.-X5-X12-X2-X4-X11-X10-X8, fires.scaled, fires.scaled)
yprime <- knn2$fitted.values
sqrt(sum((fires.scaled$Y-yprime)^2)/length(yprime))
```

```
## [1] 0.7652488
```

El segundo modelo es un poco peor que el primero pero es mucho más interpretable ya que tiene muchas menos variables. Por tanto, nos quedaremos con éste para realizar la validación cruzada.

```
nombre<-"./forestFires/forestFires"
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~.-X5-X12-X2-X4-X11-X10-X8,x_tra,test)
yprime=fitMulti$fitted.values
sum(abs(test$Y-yprime)^2)/length(yprime) ##MSE
}
knnMSEtrain<-mean(sapply(1:5,run_knn_fold,nombre,"train"))
knnMSEtest<-mean(sapply(1:5,run_knn_fold,nombre,"test"))

print(paste("Train:", knnMSEtrain))
```

```
## [1] "Train: 2363.02927134905"
```

```
print(paste("Test:",knnMSEtest))
```

```
## [1] "Test: 5334.94129109769"
```

Como vemos en los resultados, es un algoritmo que sobreajusta muchísimo al conjunto de datos. La diferencia

entre lo obtenido en train y test es bastante grande, por lo que es un algoritmo que no se comportará bien cuando lo utilicemos con nuevos datos.

R-4. Conclusiones

En este apartado vamos a analizar las conclusiones sacadas con respecto a nuestros modelos obtenidos. En primer lugar, voy a mostrar en una tabla los valores de RMSE para train y test de cada uno de los modelos:

```
data.frame(metodo = c("Regresión lineal simple","Regresión lineal múltiple","Knn"),
           RMSE_Train = c(lmSimpleMSEtrain,lmMultipleMSEtrain,knnMSEtrain) ,
            RMSE_Test=c(lmSimpleMSEtest,lmMultipleMSEtest,knnMSEtest)
           )
```

```
##                       metodo RMSE_Train RMSE_Test
## 1   Regresión lineal simple   4009.792  4042.869
## 2 Regresión lineal múltiple   3971.035  4031.589
## 3                       Knn   2363.029  5334.941
```

Como podemos observar, el peor modelo obtenido es el que proporciona el método knn. Además de sobreajustar mucho, es el que más RMSE en test tiene, por lo que es el primero que descartaremos.

Los otros dos modelos (lineal simple y lineal múltiple) tienen un RMSE en test parecido, por lo que desde este punto de vista son muy parecidos hablando en términos de resultados (aunque el múltple sea un poco mejor). Sin embargo, podemos ver que el modelo de regresión lineal múltiple sobreajusta más que el de regresión lineal simple, por lo que nos quedaríamos finalmente con el simple.

Además de estos dos últimos criterios, hay que tener en cuenta que cuanto más simple sea un modelo mejor. Por lo que, en el caso de tener dos modelos muy parecidos en resultados, nos quedariamos con el más simple ya que será mucho mas interpretable y facil de explicar.

R-5. Comparativas entre algoritmos

```
#leemosla tablacon los erroresmediosde 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]
#leemosla tablacon los erroresmediosde 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]
```

Vamos a sustituir los valores respectivos a forestfires con los que he obtenido a lo largo del estudio.

```
tablatra["forestFires",]$out_train_lm = lmSimpleMSEtrain
tablatra["forestFires",]$out_train_kknn = knnMSEtrain

tablatst["forestFires",]$out_test_lm = lmSimpleMSEtest
tablatst["forestFires",]$out_test_kknn = knnMSEtest
```

Comparativa dos a dos

En primer lugar, vamos a comparar los mejores modelos obtenidos con regresión lineal y knn con el test de Wilcoxon y veremos si tienen diferencias significativas. Para ello pondremos como modelo de referencia el obtenido con knn. Este test nos dará un grado de confianza de que dos modelos sean diferentes (1-pvalor).

```
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
```

Aplicamos el test:

```
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
## 76
```

```
Rmenos
```

```
##  V
## 95
```

```
pvalue
```

```
## [1] 0.7018814
```

No podemos decir que haya diferencias significativas entre ambos métodos porque hay un (1-0.7019) * 100 = 29.81% de confianza de que sean diferentes.

Comparativa todos con todos

Por último, aplicaremos el test de Friedman para comparar todos los algoritmos entre sí (lm,knn y m5). Dicho test nos dice con una confianza 1-pvalor si existen diferencias significativas entre al menos dos de los modelos que estemos estudiando.

```
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 un (1-0.01467)*100 = 98,533% de confianza, podemos decir que si hay dos algortimos que poseen diferencias significativas.

Con el test de post-hoc Holm podemos ver qué modelos son los que poseen diferencias. Para ello nos dará un pvalor para cada pareja de modelos y 1-pvalor será la confianza de que ambos tengan diferencias significativas.

```
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
```

Como podemos ver en la tabla obtenida, los algoritmos que poseen diferencias significativas con más de un 90% de confianza en ambos casos son M5 con lm y M5 con Knn.

Observaciones en el conjunto de training

Vamos a ver que pasa con estos algoritmos cuando observamos el conjunto de training para intentar ver a que se deben dichas diferencias y comprobar si en el conjunto de training también obtenemos los mismos resultados en los test.

```
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
```

Si vemos el pvalor que devuelve el test de Wilcoxon en training aplicandolo a lm y knn poniendo knn como referencia, vemos que hay un pvalor muy pequeño. Esto quiere decir, que a un 99% de confianza no podemos rechazar la hipótesis nula, por lo que puede que haya diferencias significativas entre ambos algoritmos. Si observamos las tabla de train y de test, podemos ver que en la gran mayoria de los casos, knn tiene un sobreaprendizaje bastante grande, por lo que en train obtiene muy buenos resultados pero luego en el conjunto de test no consigue mejorar tanto al modelo lineal.

Vamos a ver ahora con el test de Friedman si es cierto que hay diferencias al menos en dos de ellos.

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

```
##
##  Friedman rank sum test
##
## data:  as.matrix(tablatra)
## Friedman chi-squared = 22.111, df = 2, p-value = 1.58e-05
```

Otra vez, con un pvalor muy pequeño podemos decir que sí que hay diferencias significativas entre al menos dos de ellos con un 99% de confianza.

Veamos con post Holm qué modelos son los que obtienen tantas diferencias.

```
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 0.00209 -
## 3 0.00011 0.00281
##
## P value adjustment method: holm
```

En este caso, todos los métodos tienen diferencias signifivativas entre si, al 99% de confianza. Esto nos afirma lo de antes ya que M55 sigue teniendo diferencias con lm y knn pero ahora además también las tienen lm y knn, por lo que afirmamos que knn tiene mucho sobreaprendizaje por la observación de los datos.

2.Clasificación: Balance

A. ANÁLISIS DE DATOS

En primer lugar vamos a crear un data frame con los datos. Primero con el comando read.csv leeré los datos y a continuación le asignaré un nombre a cada una de las variables.

```
Balance <- read.csv("./balance/balance.dat", comment.char="@")
names(Balance) <- c("Left.weight","Left.distance","Right.weight","Right.distance","Balance.scale")
head(Balance)
```

```
##   Left.weight Left.distance Right.weight Right.distance Balance.scale
## 1           1             1            1              2             R
## 2           1             1            1              3             R
```

```
## 3           1            1            1            4            R
## 4           1            1            1            5            R
## 5           1            1            2            1            R
## 6           1            1            2            2            R
```

```
dim(Balance)
```

```
## [1] 624   5
```

Balance es un dataset de clasificación que posee 624 observaciones, 4 variables descriptivas y una de salida. Vamos a ver a continuación una lista con el significado de cada una de las variables, su rango y sus tipos. Cada una de las observaciones representa el estado de una balanza descrito por 4 variables que se expicarán a continuación. Cada una de ellas tendrá un clasificador que determina su estado: balanceada (B), desnivelada hacia la izquierda (L) y desnivelada hacia la derecha (R).

```
str(Balance)
```

```
## 'data.frame':    624 obs. of  5 variables:
##  $ Left.weight   : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Left.distance : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ Right.weight  : num  1 1 1 1 2 2 2 2 2 3 ...
##  $ Right.distance: num  2 3 4 5 1 2 3 4 5 1 ...
##  $ Balance.scale : Factor w/ 3 levels " B"," L"," R": 3 3 3 3 3 3 3 3 3 3 ...
```

- **Left-weigh** : Peso en la parte izquierda de la balanza. Variable numérica
- **Left-distance**: Distancia desde el centro de la balanza hasta el extremo izquierdo de la misma. Variable numérica
- **Right-weight** : Peso en la parte derecha de la balanza. Variable numérica.
- **Right-distance**: Distancia desde el centro de la balanza hasta el extremo derecho de la misma
- **Balance_scale**: Variable de salida, nos dice si está balanceada (B), desnivelada hacia la izquierda (L) o desnivelada hacia la derecha (R).

Como podemos ver a simple vista las variables que tiene el problema son todas importantes ya que, el desnivel o no de una balanza viene determinado por la distancia y el peso que haya en ambos lados. Si no sabemos el peso o la distancia de alguno de los lados no podemos saber el estado de la misma. Por tanto, la única transformación que podríamos realizar de los datos sería multiplicar el peso por la distancia izquierda y de la misma manera multiplicar por la derecha. De esta manera tendríamos resumida en una variable la información de ambas partes de la balanza en una. A continuación veremos si esta transformación de los datos aporta infromación a los modelos o no.

Vamos a ver si hay valores perdidos en nuestro dataset:

```
which(complete.cases(Balance) == FALSE)
```

```
## integer(0)
```

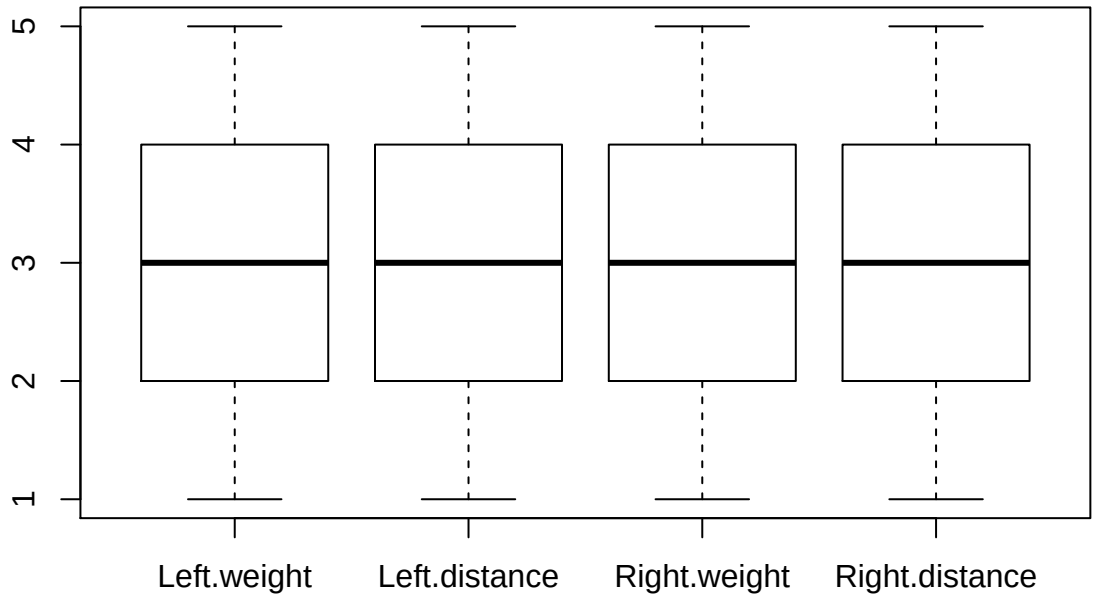
No hay valores perdidos. Ahora vamos a ver si hay observaciones repetidas:

```
which(duplicated(Balance) == TRUE)
```

```
## integer(0)
```

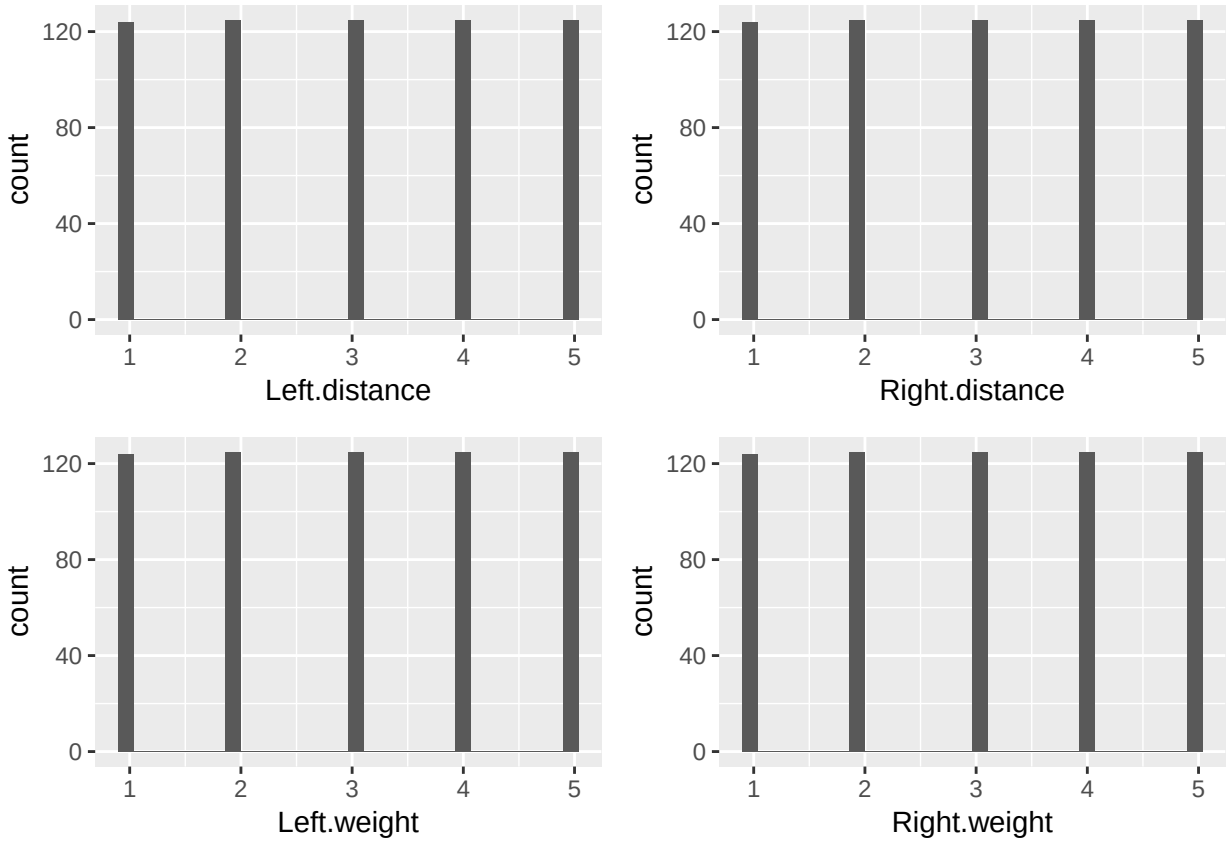
Tampoco hay duplicaciones en los datos.

```
boxplot(Balance[,1:4])
```




```
p1<-ggplot(Balance, aes(x=Left.distance)) + geom_histogram()
p2<-ggplot(Balance, aes(x=Right.distance)) + geom_histogram()
p3<-ggplot(Balance, aes(x=Left.weight)) + geom_histogram()
p4<-ggplot(Balance, aes(x=Right.weight)) + geom_histogram()


grid.arrange(p1,p2,p3,p4)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Si observamos los boxplot vemos que todas las variables se mueven en el mismo rango de valores, y además, están distribuidas de la misma manera ya que las medias y medianas de todas estámn en el mismo valor. Esto lo podemos comprobar en el siguiente resumen de los 5 valores. Además, los histogramas de todas son iguales, por lo que con respecto a los descriptores podemos decir que están balanceados.

```
summary(Balance)
```

```
##   Left.weight    Left.distance    Right.weight   Right.distance
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:2.000   1st Qu.:2.000   1st Qu.:2.000
##  Median :3.000   Median :3.000   Median :3.000   Median :3.000
##  Mean   :3.003   Mean   :3.003   Mean   :3.003   Mean   :3.003
##  3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :5.000   Max.   :5.000   Max.   :5.000   Max.   :5.000
##  Balance.scale
##   B: 48
##   L:288
##   R:288
##
##
##
```

Si observamos los resultados de summary, tenemos 48 ejemplos solamente de observaciones que están balanceadas y 288 de cada uno de los otros dos tipos.

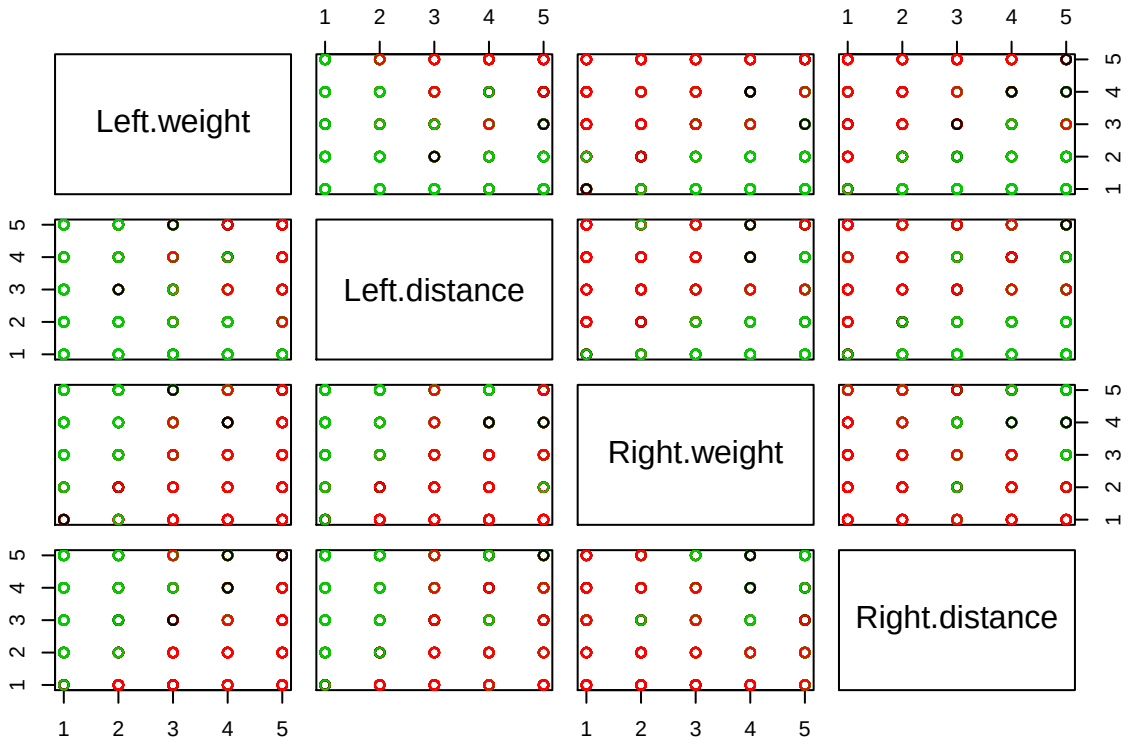
En este caso, como es un problema de clasificación la forma que deben tener los datos para ser buenos a la hora de estudiarlos, a diferencia del apartado de regresión, es que estén poco correlados y con las clases muy bien definidas. Veamos la correlación que tienen las variables a continuación.

```
cor(Balance[,1:4])
```

```
##                 Left.weight Left.distance Right.weight Right.distance
## Left.weight     1.000000000  -0.003215434 -0.003215434   -0.003215434
## Left.distance  -0.003215434   1.000000000 -0.003215434   -0.003215434
## Right.weight   -0.003215434  -0.003215434  1.000000000   -0.003215434
## Right.distance -0.003215434  -0.003215434 -0.003215434    1.000000000
```

Como podemos ver en la tabla de correlaciones, las variables no están correladas entre sí. Ésto tenemos que comprobarlo porque las variables que tienen mucha correlación normalmente no son buenas para modelos de clasificación y deberían ser eliminadas o tratadas previamente. Por tanto, todas nuestras variables van a ser útiles para la realización de nuestros modelos.

```
plot(Balance[,-5],col=Balance$Balance.scale)
```



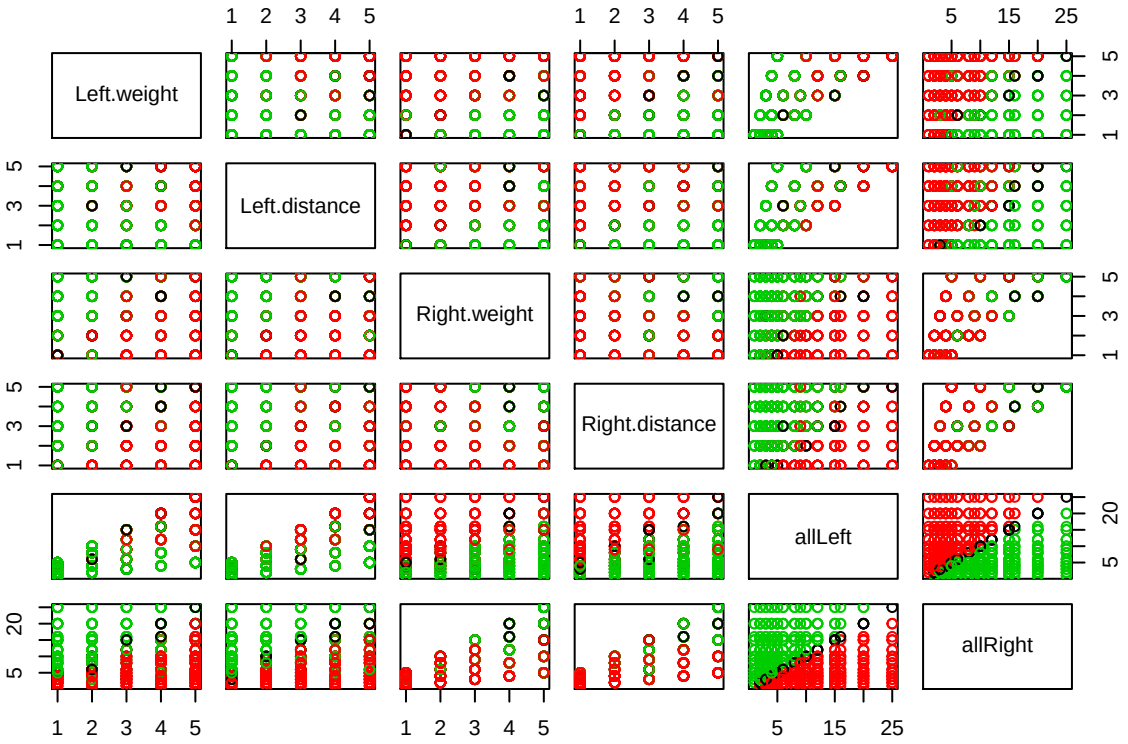
En este último gráfico, podemos ver que las variables que mejor clasificarían el modelo si tuviesemos que usar solamente dos serían las de la izquierda y la derecha a pares. Es decir, peso izquierdo con peso derecho y distancia izquierda con distancia derecha. Comparar distancias y pesos sería mucho peor y tendría mucho menos sentido. Vamos a ver a continuación qué pasa si añadimos una nueva variable.

```
Balance.new <- Balance
Balance.new$allLeft <- Balance$Left.distance*Balance$Left.weight
Balance.new$allRight <- Balance$Right.distance*Balance$Right.weight
head(Balance.new)
```

```
##   Left.weight Left.distance Right.weight Right.distance Balance.scale
## 1           1             1            1              2             R
## 2           1             1            1              3             R
```

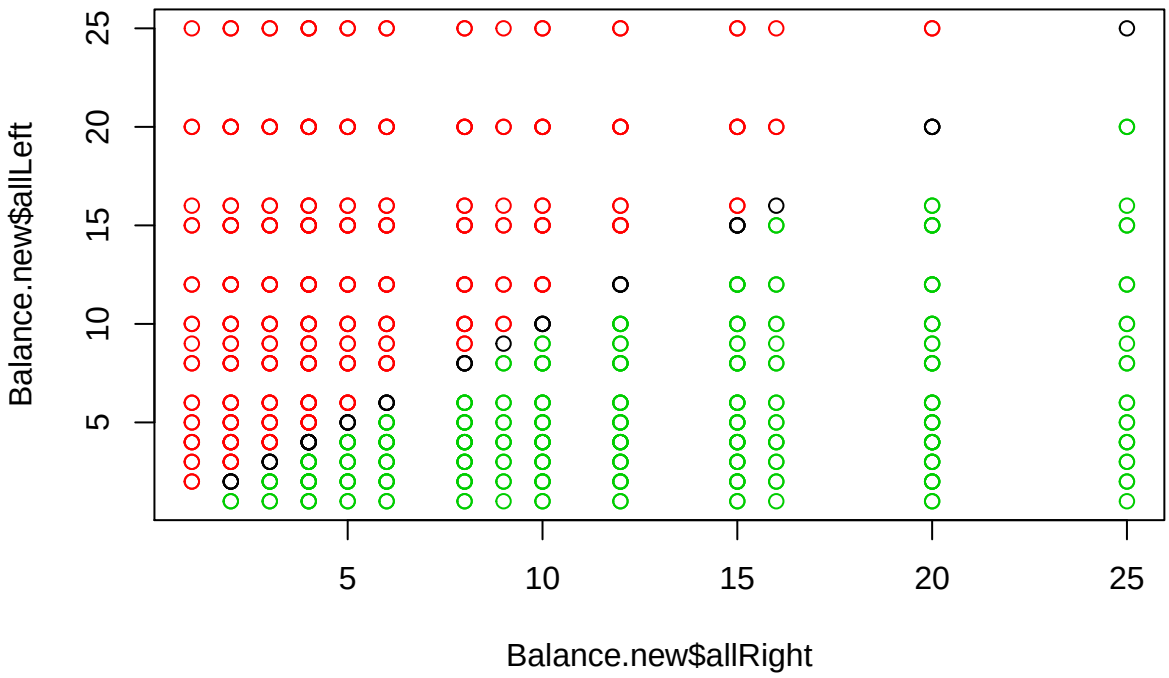
```
## 3            1             1             1              4             R
## 4            1             1             1              5             R
## 5            1             1             2              1             R
## 6            1             1             2              2             R
##   allLeft allRight
## 1       1        2
## 2       1        3
## 3       1        4
## 4       1        5
## 5       1        2
## 6       1        4
```

```
plot(Balance.new[,-5],col=Balance.new$Balance.scale)
```



Como podemos ver, ahora la mejor clasificación que obtenemos es usando las variables allLeft y allRight. A la hora de generar los modelos, usaremos las demás variables y compararemos los resultados con los modelos obtenidos con esta última variable.

```
plot(Balance.new$allLeft~Balance.new$allRight,col=Balance.new$Balance.scale)
```



B. CLASIFICACIÓN

C-1.Knn

Vamos a ver en primer lugar, una tabla con la cantidad de ejemplos de cada una de las clases que hay en nuestro dataset.

```
table(Balance$Balance.scale)
```

```
##
##   B   L   R
##  48 288 288
```

A continuación, escalaremos los datos del conjunto. Esto es es necesario porque knn es un método por distancias y si no están escaldas tendremos algunos valores muy grandes que influyan mucho más que otros. Además, como utilizaremos la neuva variable creada, el conjunto Balance inicial pasará a ser Balance.new.

```
Balance.new[,-5] <- as.data.frame(lapply(Balance.new[,-5], scale, center = TRUE, scale = TRUE))
Balance <- Balance.new
```

En primer lugar, para poder aplicar el algoritmo necesitamos dividir nuestro conjunto de datos en dos: uno para training y otro para test. Para ello sacaremos una muestra del 80% de los datos para el conjunto de train y otra de un 20% para el de test. Con cada una de esas porciones crearemos dos nuevos datasets: Balance_train será el que usaremos para entrenar nuestro modelo y Balance_test para ver como se comporta en nuevos conjuntos de datos. Estos dos ultimos conjuntos solo tendrán las variables utilizadas como descriptores, los

valores de la variable de clasificación para cada uno de los ejemplos la almacenaremos en dos factores a parte a los que llamaremos: Balance_train_labels y Balance_test_labels.

```
shuffle_ds <- sample(dim(Balance)[1])
eightypct <- (dim(Balance)[1] * 80) %/% 100
Balance_train <- Balance[shuffle_ds[1:eightypct],-5 ]
Balance_test <- Balance[shuffle_ds[(eightypct+1):dim(Balance)[1]],-5 ]
Balance_train_labels <- Balance[shuffle_ds[1:eightypct], 5]
Balance_test_labels <- Balance[shuffle_ds[(eightypct+1):dim(Balance)[1]], 5]
```

Para crear nuestro modelo usando el método Knn podemos hacerlo de la siguiente manera: aplicaremos el comando knn al que le especificaremos por parámetro el conjunto que tendremos de train (train) y las etiquetas de todos sus ejemplos (cl), el que tenemos de test (test), y el número de vecinos que queremos tener en cuenta (k).

```
Balance_test_pred <- knn(train = Balance_train, test = Balance_test, cl = Balance_train_labels)
summary(Balance_test_pred)
```

```
##  B  L  R
##  9 63 53
```

A continuación, veremos la matriz de confusión de los ejemplos clasificados por nuestro modelo knn. Si observamos la fila de los clasificados como B (balanceadas), vemos que no hay ninguno clasificado como tal, pero si observamos la columna de las etiquetas que eran de clase B , vemos que hay 4 etiquetas que se han clasificado como L en vez de B y otras dos como R en vez de B. El en caso de las clases L y R vemos que lo ha hecho mucho mejor, ha fallado solamente 1 en la clase L y otra en la clase R. Que se haya equivocado tanto en la clase B puede ser porque el número de ejemplos que teníamos de esa clase era mucho menor que las demás (puede verse en la tabla de frecuencias más arriba). Otro aspecto que puede influir es que sea más difícil de clasificar que los otros dos porque está entre uno y otro.

```
table(Balance_test_pred,Balance_test_labels)
```

```
##                   Balance_test_labels
## Balance_test_pred  B  L  R
##                 B  0  6  3
##                 L  6 57  0
##                 R  3  1 49
```

```
postResample(pred = Balance_test_pred, obs = Balance_test_labels)
```

```
##  Accuracy     Kappa
## 0.8480000 0.7287574
```

Con la libreria caret también podemos hacer un modelo knn de la siguiente manera:

En primer lugar creamos el modelo con el método train, especificándole el conjunto de ejemplos de train con sus etiquetas y el método que queremos que se utilice (knn en este caso). Además, de esta manera podemos especificar un número de vecinos k variable. De todos los k que use el mismo método elegirá el mejor tomando como valor de referencuia el accuracy.

El número de vecinos es una variable de la que va a depender mucho el modelo que creemos. Por ello, tendremos que elegir el valor idóneo para ella. En este caso he tomado valores de k desde 1 hasta 100 y me quedaré con el que mejor accuracy tenga.

```
require(caret)
knn <- train(x = Balance_train, y = Balance_train_labels, method = "knn",tuneGrid = data.frame(k=1:100))
knn
```

```
## k-Nearest Neighbors
##
```

```
## 499 samples
##   6 predictor
##   3 classes: ' B', ' L', ' R'
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 499, 499, 499, 499, 499, 499, ...
## Resampling results across tuning parameters:
##
##   k    Accuracy   Kappa
##     1  0.8167542  0.6771360
##     2  0.8050961  0.6582508
##     3  0.8092099  0.6648282
##     4  0.8204070  0.6829223
##     5  0.8329567  0.7034456
##     6  0.8466856  0.7261436
##     7  0.8551315  0.7395263
##     8  0.8646942  0.7556338
##     9  0.8711647  0.7662001
##    10  0.8773042  0.7761579
##    11  0.8821645  0.7844976
##    12  0.8852702  0.7898924
##    13  0.8867720  0.7920767
##    14  0.8880674  0.7941842
##    15  0.8868526  0.7921393
##    16  0.8910387  0.7993408
##    17  0.8876428  0.7929730
##    18  0.8901511  0.7974542
##    19  0.8932566  0.8030617
##    20  0.8938891  0.8041320
##    21  0.8945691  0.8051867
##    22  0.8923040  0.8007599
##    23  0.8921143  0.8005143
##    24  0.8906610  0.7976862
##    25  0.8909009  0.7980284
##    26  0.8902694  0.7967363
##    27  0.8896295  0.7954609
##    28  0.8892188  0.7944713
##    29  0.8893571  0.7946767
##    30  0.8889427  0.7938805
##    31  0.8872757  0.7907654
##    32  0.8886046  0.7931932
##    33  0.8894339  0.7948065
##    34  0.8892351  0.7943156
##    35  0.8919746  0.7993906
##    36  0.8907885  0.7972328
##    37  0.8918382  0.7990599
##    38  0.8937595  0.8025559
##    39  0.8930156  0.8013063
##    40  0.8913406  0.7981170
##    41  0.8923991  0.7999515
##    42  0.8904193  0.7962548
##    43  0.8913782  0.7979924
##    44  0.8919703  0.7990843
```

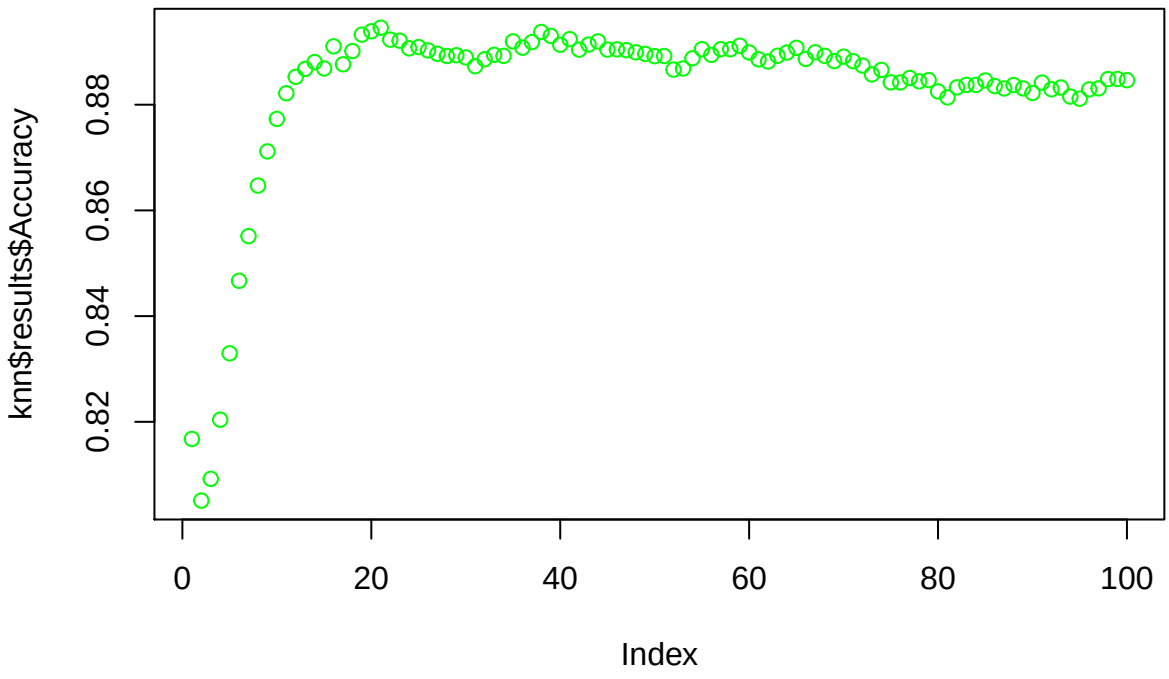
```
##     45  0.8904228  0.7962131
##     46  0.8904694  0.7962758
##     47  0.8903154  0.7960194
##     48  0.8899093  0.7952373
##     49  0.8896070  0.7947104
##     50  0.8891792  0.7938894
##     51  0.8892141  0.7939731
##     52  0.8866426  0.7891728
##     53  0.8868453  0.7895557
##     54  0.8887816  0.7931561
##     55  0.8905040  0.7963293
##     56  0.8894357  0.7943865
##     57  0.8905039  0.7963494
##     58  0.8904951  0.7963768
##     59  0.8911595  0.7975641
##     60  0.8898767  0.7952355
##     61  0.8885827  0.7928298
##     62  0.8881736  0.7920458
##     63  0.8892564  0.7940736
##     64  0.8898630  0.7951596
##     65  0.8907563  0.7967822
##     66  0.8886605  0.7928890
##     67  0.8899126  0.7952902
##     68  0.8892351  0.7940113
##     69  0.8882496  0.7921268
##     70  0.8890843  0.7936930
##     71  0.8882361  0.7920687
##     72  0.8874084  0.7905157
##     73  0.8857171  0.7874261
##     74  0.8865844  0.7891063
##     75  0.8842214  0.7846972
##     76  0.8842062  0.7846110
##     77  0.8850628  0.7862393
##     78  0.8844133  0.7850020
##     79  0.8846519  0.7854516
##     80  0.8825120  0.7814168
##     81  0.8813770  0.7793043
##     82  0.8832823  0.7828525
##     83  0.8837500  0.7837098
##     84  0.8837416  0.7837186
##     85  0.8845867  0.7852285
##     86  0.8835139  0.7832714
##     87  0.8830693  0.7824478
##     88  0.8837144  0.7836337
##     89  0.8830791  0.7825283
##     90  0.8822132  0.7809102
##     91  0.8841839  0.7846025
##     92  0.8829147  0.7822210
##     93  0.8832520  0.7828881
##     94  0.8815419  0.7797278
##     95  0.8811381  0.7789922
##     96  0.8828952  0.7821711
##     97  0.8830985  0.7825780
##     98  0.8848466  0.7858408
```

```
##    99  0.8848624  0.7858598
##   100  0.8846305  0.7854480
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 21.
```

En este caso he realizado el modelo con vecinos desde 1 hasta 100. Si observamos la siguiente gráfica, vemos que la variable accuracy aumenta mucho en el primer número de vecinos mientras que a partir de un k se estanca dicho aumento (puede que después en algun valor de k mejore algo pero no en exceso). Por otro lado vemos que con el kappa pasa algo muy similar.

A continuación, pintaré un gráfico que representa muy bien el comportamiento del algoritmo con respecto al k que establecemos. Como podemos ver, para números de k menores que aproximadamente 15, el algoritmo mejora muchísimo conforme vamos subiendo el número de vecinos. Sin embargo, cuando ya se aumentan los vecinos a más de 15, esta subida se detiene y comienza a dar peores resultados. Después de dicho valor, la precisión del modelo ya no mejora tanto pero sí que se siguen dando mejores modelos.

```
all_Accuracies <- knn$results$Accuracy
plot(knn$results$Accuracy,col="green")
```



Una vez que tenemos el modelo creado, vamos a ver como nos clasifica los nuevos datos. Para ello usaremos el método predict pasándole como argumento nuestro modelo y el conjunto de test.

```
knnPred <- predict(knn, newdata = Balance_test)
knnPred
```

```
##   [1]  R  R  R  L  R  L  R  L  L  R  L  R  L  R  R  L  L  L  L  R  R  R  L
##  [24]  L  R  L  R  L  R  L  R  L  L  L  R  L  L  L  L  R  L  R  R  R  L  R
##  [47]  R  R  L  L  L  R  L  R  R  R  L  L  L  L  L  L  L  L  L  R  L  R  L
```

```
##  [70]  L  L  L  R  L  L  R  L  R  L  R  R  R  R  L  L  R  R  R  R  R  L  R
##  [93]  L  L  R  L  L  R  R  R  R  L  R  L  L  R  R  L  R  L  L  L  L  R  R
## [116]  R  L  L  R  R  L  R  R  L  R
## Levels:  B  L  R
```

En este caso, en vez de mostrar la matriz de confusión para ver como se ha comportado el método, vamos a obtener los valores accuracy y kappa. Con accuracy nos referimos a la precisión del ajuste definida por un valor entre 0 (nada preciso) y 1(preciso al 100%). El segundo valor que tenemos es kappa que se refiere al porcentaje de mejoría con respecto a un modelo aletaorio.

```
table(knnPred,Balance_test_labels)
```

```
##        Balance_test_labels
## knnPred  B  L  R
##       B  0  0  0
##       L  4 61  0
##       R  5  3 52
```

```
info.knn = postResample(pred = knnPred, obs = Balance_test_labels)
info.knn
```

```
##  Accuracy     Kappa
## 0.9040000 0.8202516
```

En este caso con respecto del anterior hemos conseguido mejorar el modelo en accuracy y en kappa un 2%. Como hemos podido apreciar, la creación del modelo teniendo en cuenta un mayor número de posibilidades de k tarda mucho más que si no consideramos esa opción. Es por eso que observar el comportamiento que tiene el algoritmo con respecto al k nos puede aportar tanta información. De hecho, como hemos visto que en los primeros k aumenta el accuracy de forma muy rápida y luego el aumento es mínimo, no haremos tantas evaluaciones de k, ya que es preferible que tarde menos y quedarnos con un k más pequeño, que tener un mayor tiempo de ejecución para mejorar en muy poco el modelo.

C-2.LINEAR DISCRIMINANT ANALYSIS (LDA)

Como ya sabemos, este método funciona bien cuando las varianzas de las clases son similares. Vamos a ver qué varianzas tienen las clases para cada una de las variables que usaremos.

```
var(Balance[Balance$Balance.scale == " L",]$Left.distance)
```

```
## [1] 0.7533914
```

```
var(Balance[Balance$Balance.scale == " R",]$Left.distance)
```

```
## [1] 0.8869033
```

```
var(Balance[Balance$Balance.scale == " B",]$Left.distance)
```

```
## [1] 0.9891452
```

```
var(Balance[Balance$Balance.scale == " L",]$Right.distance)
```

```
## [1] 0.8869033
```

```
var(Balance[Balance$Balance.scale == " R",]$Right.distance)
```

```
## [1] 0.7533914
```

```
var(Balance[Balance$Balance.scale == " B",]$Right.distance)
```

```
## [1] 0.9891452
```

```
var(Balance[Balance$Balance.scale == " L",]$Left.weight)
```

```
## [1] 0.7533914
```

```
var(Balance[Balance$Balance.scale == " R",]$Left.weight)
```

```
## [1] 0.8869033
```

```
var(Balance[Balance$Balance.scale == " B",]$Left.weight)
```

```
## [1] 0.9891452
```

```
var(Balance[Balance$Balance.scale == " L",]$Right.weight)
```

```
## [1] 0.8869033
```

```
var(Balance[Balance$Balance.scale == " R",]$Right.weight)
```

```
## [1] 0.7533914
```

```
var(Balance[Balance$Balance.scale == " B",]$Right.weight)
```

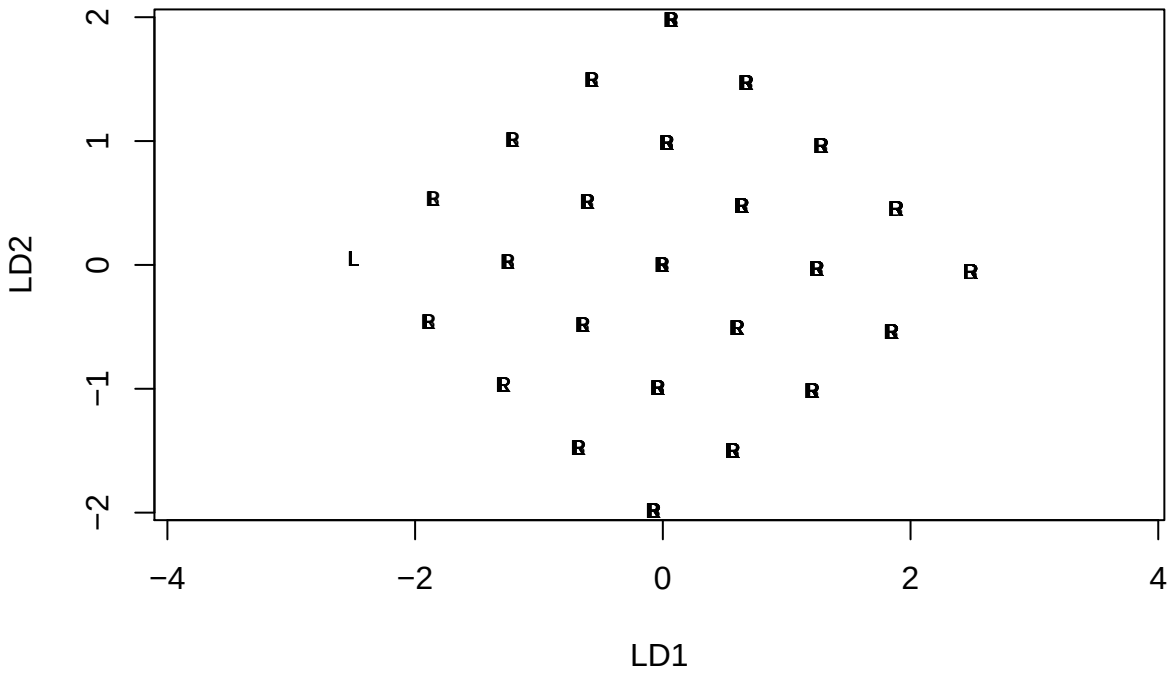
```
## [1] 0.9891452
```

Las varianzas entre clases son más o menos parecidas en todas ellas, por lo que el algortimo debe funcionar bien.

Vamos a crear el modelo a continación para ver que resultados nos da. Crearé un conjunto de training con la variable de clasificación para poder pasarla como argumento en la creación del modelo. El primer modelo que crearé será con las variables que tenía el conjunto inicial y después con las nuevas creadas.

Dentro de las variables iniciales, primero usaré las de la distancia, luego las de peso y después las 4 juntas.

```
training.data <- Balance_train
training.data$Balance.scale <- Balance_train_labels
lda.fit1 <- lda(Balance.scale ~ Right.distance + Left.distance ,data=training.data)
plot(lda.fit1)
```



Para ver su comportamiento, vamos a crear el predictor con el comando predict.

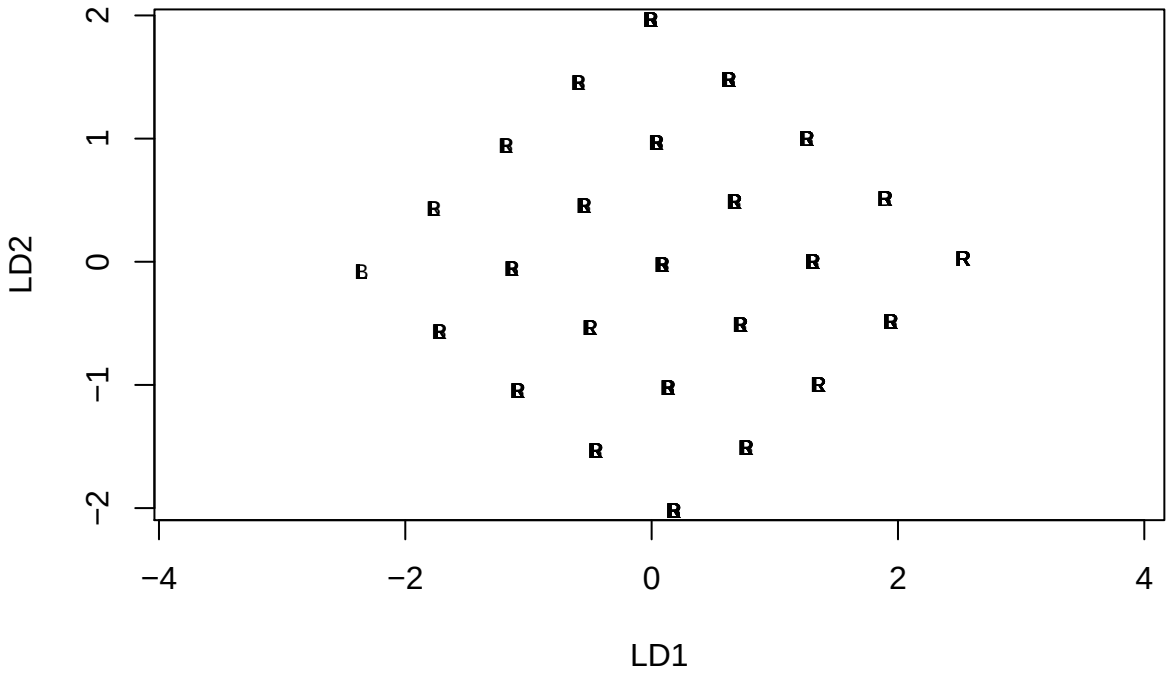
```
test.data <- Balance_test
test.data$Balance.scale <- Balance_test_labels

lda.pred1 <- predict(lda.fit1,test.data)

table(lda.pred1$class,test.data$Balance.scale)
```

```
##
##        B  L  R
##    B   0  0  0
##    L   3 42 10
##    R   6 22 42
```

```
info.lda1 <- postResample(lda.pred1$class,Balance_test_labels)
info.lda1
```

```
##  Accuracy     Kappa
## 0.6720000 0.3945659
```

```
training.data <- Balance_train
training.data$Balance.scale <- Balance_train_labels
lda.fit2 <- lda(Balance.scale ~ Right.weight + Left.weight ,data=training.data)
plot(lda.fit2)
```



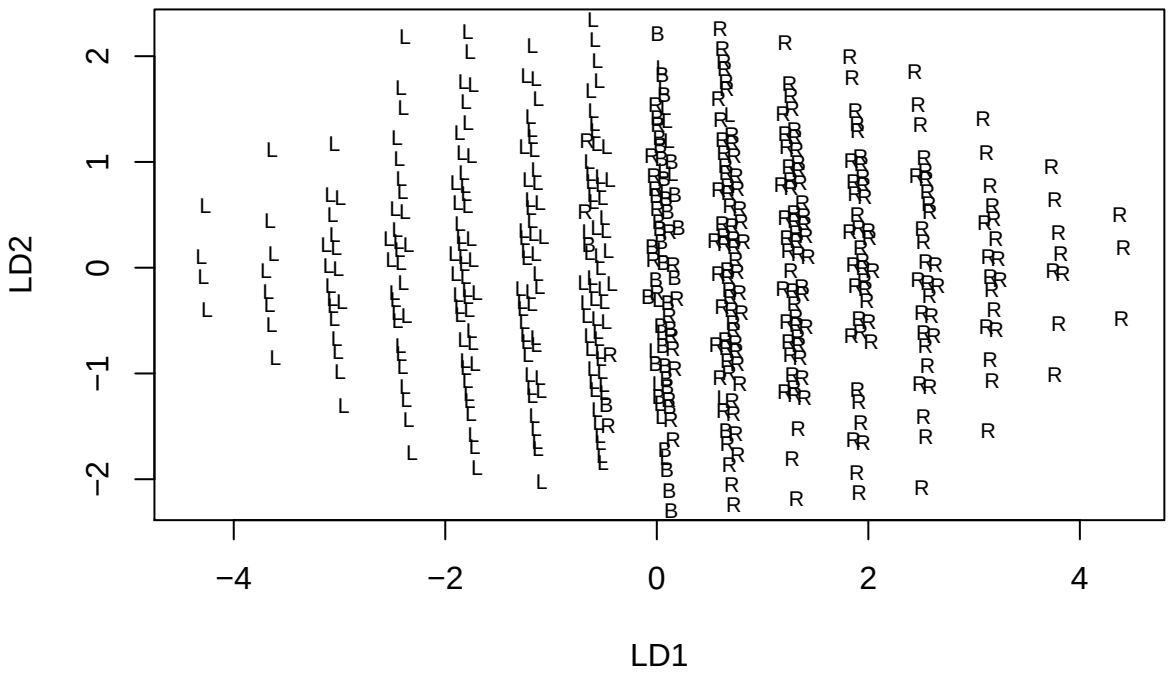

```
test.data <- Balance_test
test.data$Balance.scale <- Balance_test_labels

lda.pred2 <- predict(lda.fit2,test.data)

table(lda.pred2$class,test.data$Balance.scale)
```

```
##
##       B  L  R
##    B  0  0  0
##    L  2 40  7
##    R  7 24 45
```

```
info.lda2 <- postResample(lda.pred2$class,Balance_test_labels)
info.lda2
```

```
##  Accuracy     Kappa
## 0.6800000 0.4143142
```

```
training.data <- Balance_train
training.data$Balance.scale <- Balance_train_labels
lda.fit3 <- lda(Balance.scale ~ Right.weight + Left.weight + Right.distance + Left.distance ,data=train
plot(lda.fit3)
```




```
test.data <- Balance_test
test.data$Balance.scale <- Balance_test_labels

lda.pred3 <- predict(lda.fit3,test.data)

table(lda.pred3$class,test.data$Balance.scale)
```

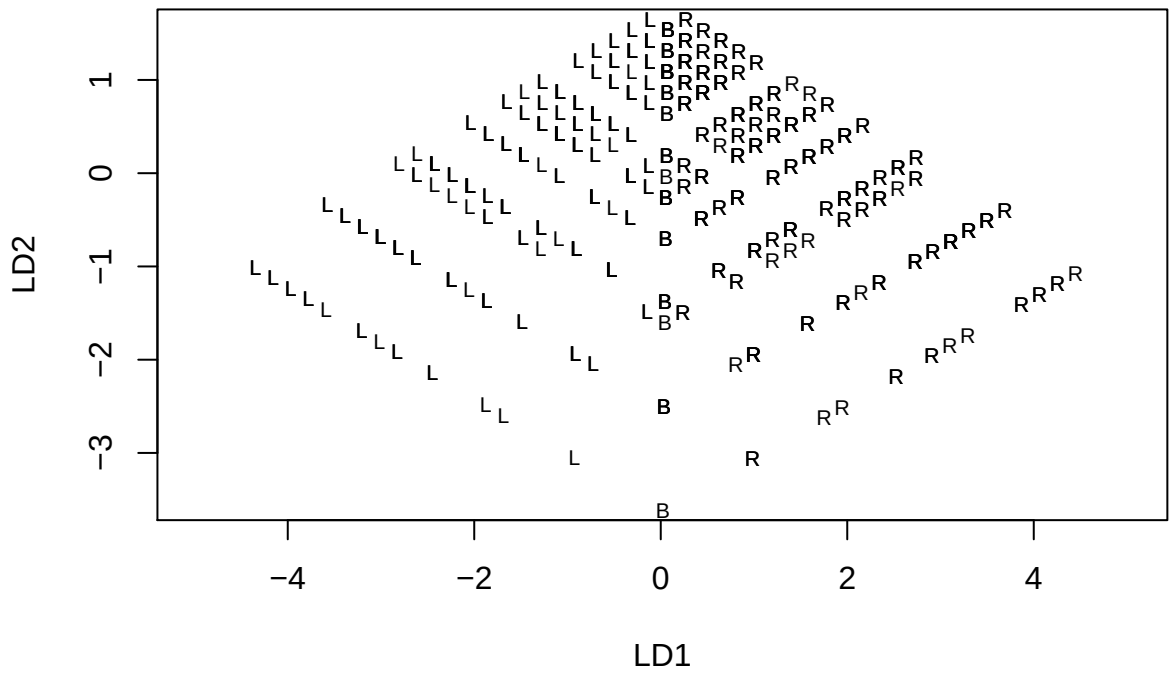
```
##
##       B  L  R
##    B  0  0  0
##    L  2 58  2
##    R  7  6 50
```

```
info.lda3 <- postResample(lda.pred3$class,Balance_test_labels)
info.lda3
```

```
##  Accuracy     Kappa
## 0.8640000 0.7464503
```

En este caso hemos visto un aumento de accuracy bastante grande con respecto a los dos modelos anteriores. Vamos a ver si mejora el modelo con las variables nuevas creadas.

```
training.data <- Balance_train
training.data$Balance.scale <- Balance_train_labels
lda.fit4 <- lda(Balance.scale ~ allLeft + allRight ,data=training.data)
plot(lda.fit4)
```




```
test.data <- Balance_test
test.data$Balance.scale <- Balance_test_labels

lda.pred4 <- predict(lda.fit4,test.data)

table(lda.pred3$class,test.data$Balance.scale)
```

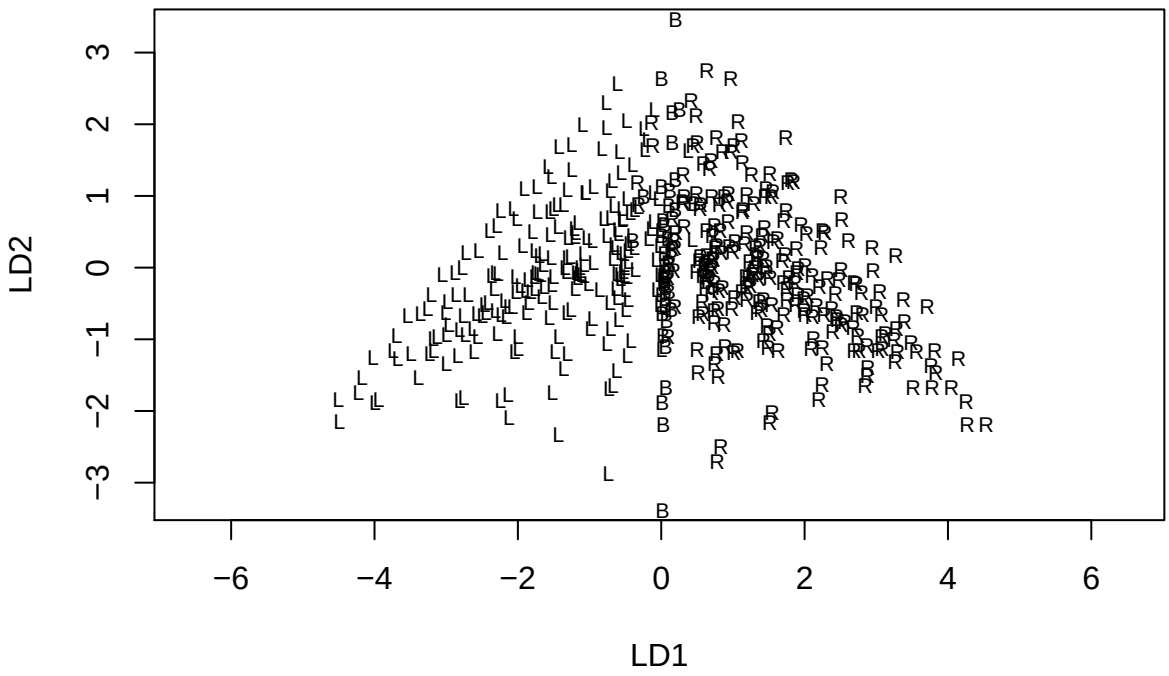
```
##
##       B  L  R
##    B  0  0  0
##    L  2 58  2
##    R  7  6 50
```

```
info.lda4 <- postResample(lda.pred4$class,Balance_test_labels)
info.lda4
```

```
##  Accuracy     Kappa
## 0.9280000 0.8653823
```

Como podemos observar, este modelo es el mejor obtenido hasta ahora con diferencia. Veamos que ocurre cuando usamos todas las variables que tenemos, incluyendo las dos nuevas creadas.

```
training.data <- Balance_train
training.data$Balance.scale <- Balance_train_labels
lda.fit5 <- lda(Balance.scale ~.,data=training.data)
plot(lda.fit5)
```




```
test.data <- Balance_test
test.data$Balance.scale <- Balance_test_labels

lda.pred5 <- predict(lda.fit5,test.data)

table(lda.pred3$class,test.data$Balance.scale)
```

```
##
##       B  L  R
##    B  0  0  0
##    L  2 58  2
##    R  7  6 50
```

```
info.lda5 <- postResample(lda.pred5$class,Balance_test_labels)
info.lda5
```

```
##  Accuracy     Kappa
## 0.9120000 0.8357032
```

Incluso usando todas las variables, el mejor modelo es el formado por las dos nuevas variables creadas, por tanto será ese el que nos quedemos para realizar la comparativa.

C-3.QUADRATIC DISCRIMINANT ANALYSIS (QDA)

El siguiente algoritmo que vamos a aplicar es muy parecido al anterior con la diferencia de que, al contrario que LDA, QDA asume que las varianzas de cada clase son diferentes.

Utilizaré la variable training data creada anteriormente para crear el modelo. Además, el procedimiento para realizar los modelos será usando el mismo orden de aplicación de las variables usado con LDA.

```
qda.fit1 <- qda(Balance.scale~ Right.distance + Left.distance, data=training.data)
qda.fit1
```

```
## Call:
## qda(Balance.scale ~ Right.distance + Left.distance, data = training.data)
##
## Prior probabilities of groups:
##          B          L          R
## 0.07815631 0.44889780 0.47294589
##
## Group means:
##    Right.distance Left.distance
##  B     0.03399561   -0.07479033
##  L    -0.41895537    0.45546018
##  R     0.42619350   -0.43372247
```

Con el predictor creado vamos a ver los datos que predice con respecto a nuestro conjunto de test.

```
qda.pred1 <- predict(qda.fit1,test.data)
info.qda1 <- postResample(qda.pred1$class,Balance_test_labels)
info.qda1
```

```
## Accuracy    Kappa
## 0.680000 0.405116
```

```
qda.fit2 <- qda(Balance.scale~ Right.weight + Left.weight, data=training.data)
qda.fit2
```

```
## Call:
## qda(Balance.scale ~ Right.weight + Left.weight, data = training.data)
##
## Prior probabilities of groups:
##          B          L          R
## 0.07815631 0.44889780 0.47294589
##
## Group means:
##    Right.weight Left.weight
##  B   -0.1110523  0.01586462
##  L   -0.4694631  0.43336303
##  R    0.4112124 -0.40376024
```

```
qda.pred2 <- predict(qda.fit2,test.data)
info.qda2 <- postResample(qda.pred2$class,Balance_test_labels)
info.qda2
```

```
##  Accuracy     Kappa
## 0.6720000 0.4030285
```

```
qda.fit3 <- qda(Balance.scale~ Right.weight + Left.weight + Right.distance + Left.distance, data=traini
qda.fit3
```

```
## Call:
## qda(Balance.scale ~ Right.weight + Left.weight + Right.distance +
##     Left.distance, data = training.data)
##
```

```
## Prior probabilities of groups:
##          B          L          R
## 0.07815631 0.44889780 0.47294589
##
## Group means:
##    Right.weight Left.weight Right.distance Left.distance
##  B   -0.1110523  0.01586462     0.03399561   -0.07479033
##  L   -0.4694631  0.43336303    -0.41895537    0.45546018
##  R    0.4112124 -0.40376024     0.42619350   -0.43372247
```

```
qda.pred3 <- predict(qda.fit3,test.data)
info.qda3 <- postResample(qda.pred3$class,Balance_test_labels)
info.qda3
```

```
##  Accuracy     Kappa
## 0.9280000 0.8754291
```

```
#qda.fit4 <- qda(Balance.scale~ allLeft + allRight, data=training.data)
#qda.fit4
```

```
#qda.fit5 <- qda(Balance.scale~ ., data=training.data)
#qda.fit5
```

Cuando añadimos laa dos nuevas variables, el método qda nos da un error asociado a dicha ecuación. Esto es debido a que tiene poca información sobre un grupo y es muy dificil para el método obtener un buen modelo.

Por lo tanto, nos quedaremos con el modelo formado por las cuatro variables iniciales que teniamos, ya que es el que mayor accuraacy nos ha proporcionado.

```
nombre<-"./balance/balance"
run_qda_fold<-function(i, x, tt= "test") {
file <-paste(x, "-10-", i, "tra.dat", sep="")
x_tra<-read.csv(file, comment.char="@")
file <-paste(x, "-10-", 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
}

x_tra$Y <- as.numeric(x_tra$Y)
test$Y <- as.numeric(test$Y)
fitMulti=qda(Y ~ X1 + X2 + X3 + X4, data=x_tra)
yprime=predict(fitMulti,test)
sum(abs(test$Y-as.numeric(yprime$class))^2)/length(yprime) ##MSE
mean(yprime$class!=test$Y)
}
qdaMSEtrain<-mean(sapply(1:10,run_qda_fold,nombre,"train"))
qdaMSEtest<-mean(sapply(1:10,run_qda_fold,nombre,"test"))
```

```r
print(paste("Train:", qdaMSEtrain))
```

```
## [1] "Train: 0.0833482935793036"
```

```r
print(paste("Test:",qdaMSEtest))
```

```
## [1] "Test: 0.0845628415300546"
```

C-4.Comparativa de algoritmos y conclusiones

En este apartado realizaré una comparativa de los mejores modelos obtenidos teniendo en cuenta su accuracy y su kappa.

```r
data.frame(metodo = c("knn","lda","qda"),
           accuracy = c(info.knn[1],info.lda4[1],info.qda3[1]) ,
           kappa =c(info.knn[2],info.lda4[2],info.qda3[2])
           )
```

```
##   metodo accuracy     kappa
## 1    knn    0.904 0.8202516
## 2    lda    0.928 0.8653823
## 3    qda    0.928 0.8754291
```

Tomando como referencia la medida de accuracy podemos decir que los mejores modelos son los creados con lda y qda, con un mismo valor.Pese a ello, nos quedaríamos con el modelo QDA ya que parece que sobreajusta menos el modelo.

En cualquier caso, vamos a comparar los algoritmos con unos tests estadísticos para ver si existen diferencias significativas entre ellos.

```r
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]

#leemosla tablacon los erroresmediosde 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]
```

Veamos con el test de Friedman si existen diferencias significativas entre los 3 algoritmos.

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

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

En este caso, no podemos decir que los modelos sean diferentes ya que la confianza que nos devuelve el método es muy cercana a 0.5, es decir, casi como decirlo al azar. De todas formas, veamos los resultados obtenidos con el test de post Holm para cada pareja de modelos.

```r
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 0.65 -
## 3 0.59 0.53
##
## P value adjustment method: holm
```

Como podemos ver, los pvalor asociados a todos los pares de modelos rondan el 0.5, por lo que no podemos decir que sean diferentes.