Introducción a la Ciencia de Datos

Trabajo Final Teórico/Práctico

Antonio Manuel Milán Jiménez

4 de Enero de 2019

Contents

Análisis de Datos

Vamos a empezar cargando los dos 'datasets'.

```
#Carga de los 'datasets'
library(ggplot2)
heart <- read.csv("./heart/heart.dat", comment.char = "@", header = FALSE)

names(heart) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
"FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced", "Oldpeak",
"Slope", "MajorVessels", "Thal", "Class")

deltaAil <- read.csv("./delta_ail/delta_ail.dat",comment.char = "@", header = FALSE)

names(deltaAil) <- c("RollRate", "PitchRate", "currPitch", "currRoll", "diffRollRate", "Sa")
```

Análisis del Dataset 'heart'

En este 'dataset' se trata de poder predecir una enfermedad en el corazón a partir de otros factores tales como la edad, el sexo, colesterol, etc. Por lo tanto se trata de un problema de clasificación.

```
#Dimensiones del dataset heart
dim(heart)
```

```
## [1] 270  14
```

A partir de las dimensiones del 'dataset' descubrimos que tenemos un total de 13 factores para hacer la predicción (la última variable es la predicción en sí). También sabemos que tenemos un total de 270 observaciones.

Estos son los nombres de las variables:

```
#Nombre de las variables
colnames(heart)
```

```
##  [1] "Age"                    "Sex"
##  [3] "ChestPainType"          "RestBloodPressure"
##  [5] "SerumCholestoral"       "FastingBloodSugar"
##  [7] "ResElectrocardiographic" "MaxHeartRate"
##  [9] "ExerciseInduced"        "Oldpeak"
## [11] "Slope"                  "MajorVessels"
## [13] "Thal"                   "Class"
```

Y aquí el tipo de dato atómico de cada una de ellas:

```
#Tipo atómico de cada variable
unlist(lapply(heart,class))
```

```
##                     Age                     Sex           ChestPainType
##               "integer"               "integer"               "integer"
##       RestBloodPressure       SerumCholestoral       FastingBloodSugar
##               "integer"               "integer"               "integer"
## ResElectrocardiographic           MaxHeartRate         ExerciseInduced
##               "integer"               "integer"               "integer"
```

```
##                  Oldpeak                    Slope            MajorVessels
##                "numeric"                "integer"                "integer"
##                     Thal                    Class
##                "integer"                "integer"
```

Descubrimos que todas son de tipo 'entero' excepto la variable 'Oldpeak' que es 'real'.

```
#Estructura del dataset
class(heart)
```

```
## [1] "data.frame"
```

```
unique(heart$Class)
```

```
## [1] 2 1
```

Además, sabemos que el dataset es un dataframe y que variable a predecir es binaria, 1 y 2; indicando un 1 que hay enfermedad y un 2 indicando que no la hay. Para hacer más comprensible esta variable de salida, la reconvertimos a un "factor" con los valores apropiados:

```
#Conversión de la variable de salida a un factor
heart$Class <- factor(heart$Class, levels = c(1, 2), labels = c("Si", "No"))
```

Eliminación de 'missing values' y observaciones repetidas

Antes de empezar a trabajar con los datos, debemos tratar los 'missing values' de nuestro conjunto de datos pues será necesarios para según qué cálculos.

```
#Comprobación de missing values
which(is.na(heart) == TRUE)
```

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

Sin embargo, descubrimos que no hay ningún 'missing value' en los datos por lo que no será necesario preocuparnos por ello.

Otro caso que podría suceder en el conjunto de datos es que haya observaciones repetidas que deberán ser eliminadas.

```
#Comprobación de valores duplicados
which(duplicated(heart) == TRUE)
```

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

Observamos que tampoco hay observaciones repetidas en el 'dataset' así que no tenemos que eliminar ninguna.

Datos de salida equilibrados

Otro punto a tener en cuenta es que estén los datos equilibrados respecto a la variable a predecir, es decir, que haya un número de observaciones parecidas para una clase y para otra.

```
#Proporción de la variable de salida
round(prop.table(table(heart$Class)) * 100, digits = 1)
```

```
##
##   Si   No
## 55.6 44.4
```

Tenemos una proporción de 55.6% frente a 44.4% en la variable de salida por lo que hay un cierto equilibrio en las observaciones. Si quisiesmos que estuviesen más balanceados podríamos simplemente eliminar observaciones de la clase "Si".

Medidas de distribución sobre el conjunto de datos

A continuación estudiamos la distribución de los datos para las diferentes variables del conjunto de datos. Con esto aprenderemos más de los datos y podremos tratar los datos en el caso de que no estén bien distribuidos.

Empezamos por las variables categóricas, estudiando si están bien distribuidas.

```
#Proporción de las variables discretas predictoras
round(prop.table(table(heart$Sex)) * 100, digits = 1)
```

```
##
##    0    1
## 32.2 67.8
```

```
round(prop.table(table(heart$ChestPainType)) * 100, digits = 1)
```

```
##
##    1    2    3    4
##  7.4 15.6 29.3 47.8
```

```
round(prop.table(table(heart$FastingBloodSugar)) * 100, digits = 1)
```

```
##
##    0    1
## 85.2 14.8
```

```
round(prop.table(table(heart$ResElectrocardiographic)) * 100, digits = 1)
```

```
##
##    0    1    2
## 48.5  0.7 50.7
```

```
round(prop.table(table(heart$ExerciseInduced)) * 100, digits = 1)
```

```
##
##  0  1
## 67 33
```

```
round(prop.table(table(heart$Slope)) * 100, digits = 1)
```

```
##
##    1    2    3
## 48.1 45.2  6.7
```

```
round(prop.table(table(heart$MajorVessels)) * 100, digits = 1)
```

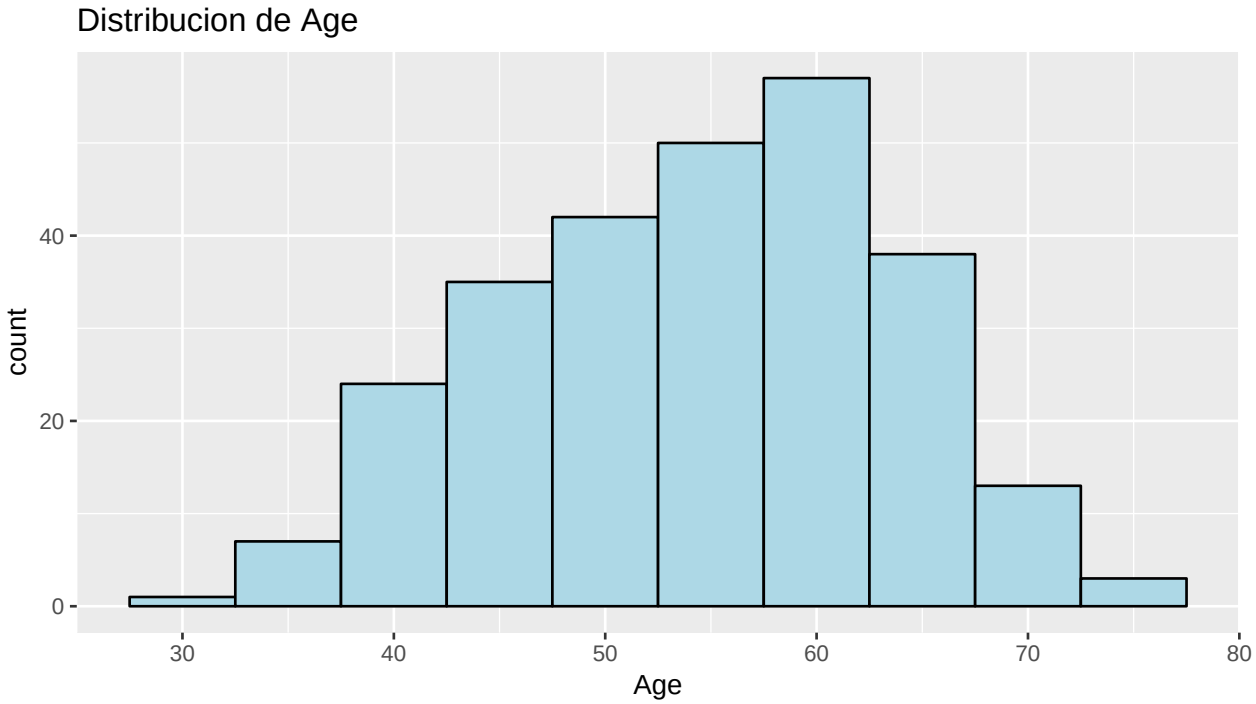
```
##
##    0    1    2    3
## 59.3 21.5 12.2  7.0
```

```
round(prop.table(table(heart$Thal)) * 100, digits = 1)
```

```
##
##    3    6    7
## 56.3  5.2 38.5
```

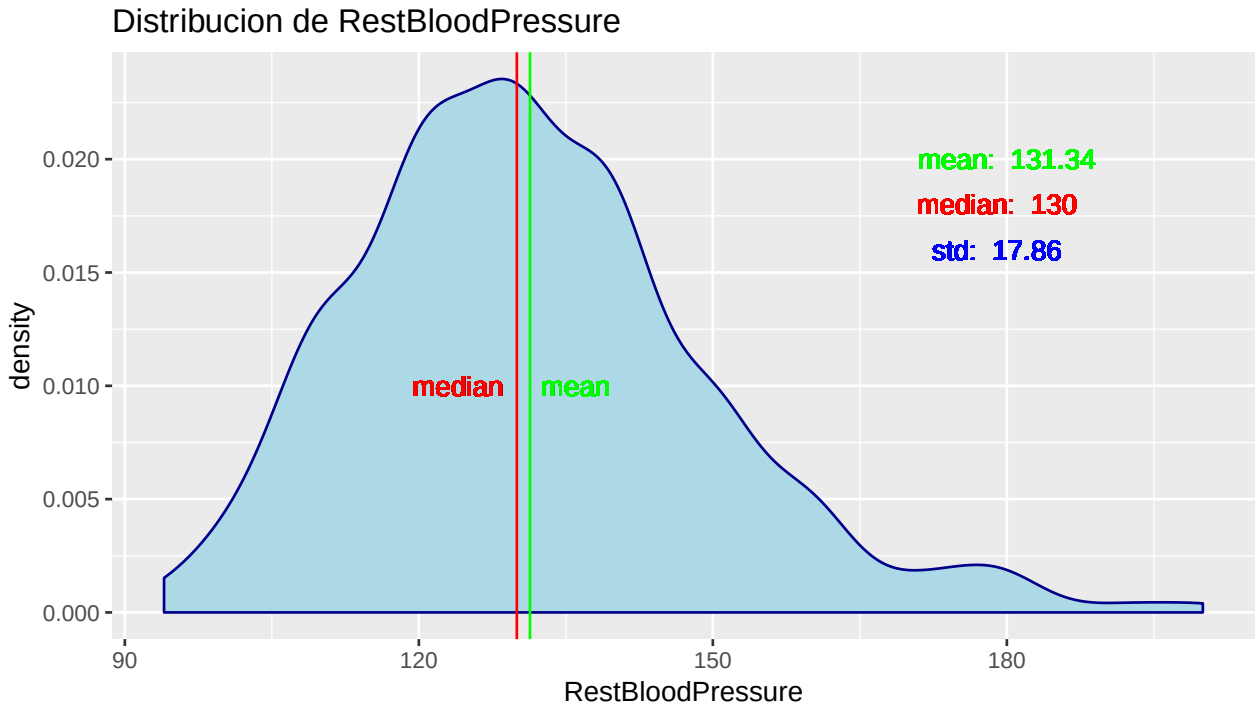
Vemos que no están bien balanceadas las clases lo cuál podría influir negativamente para futuras predicciones en las que no se tengan suficientes datos de entrenamiento para las clases más escasas.

Estudiando ahora las variables continuas, empezamos con la variable de "Edad":

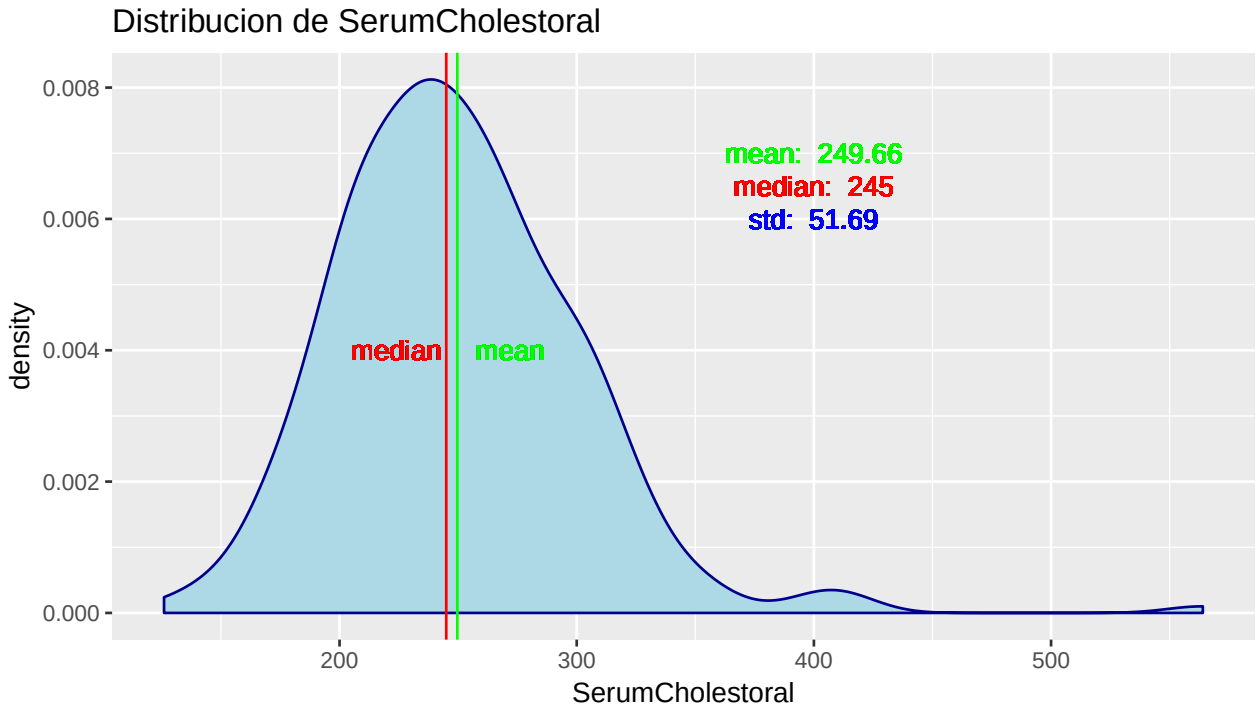


Observamos una distribución normal por lo que podemos decir que los datos para la variable "Edad" están muy bien distribuidos.

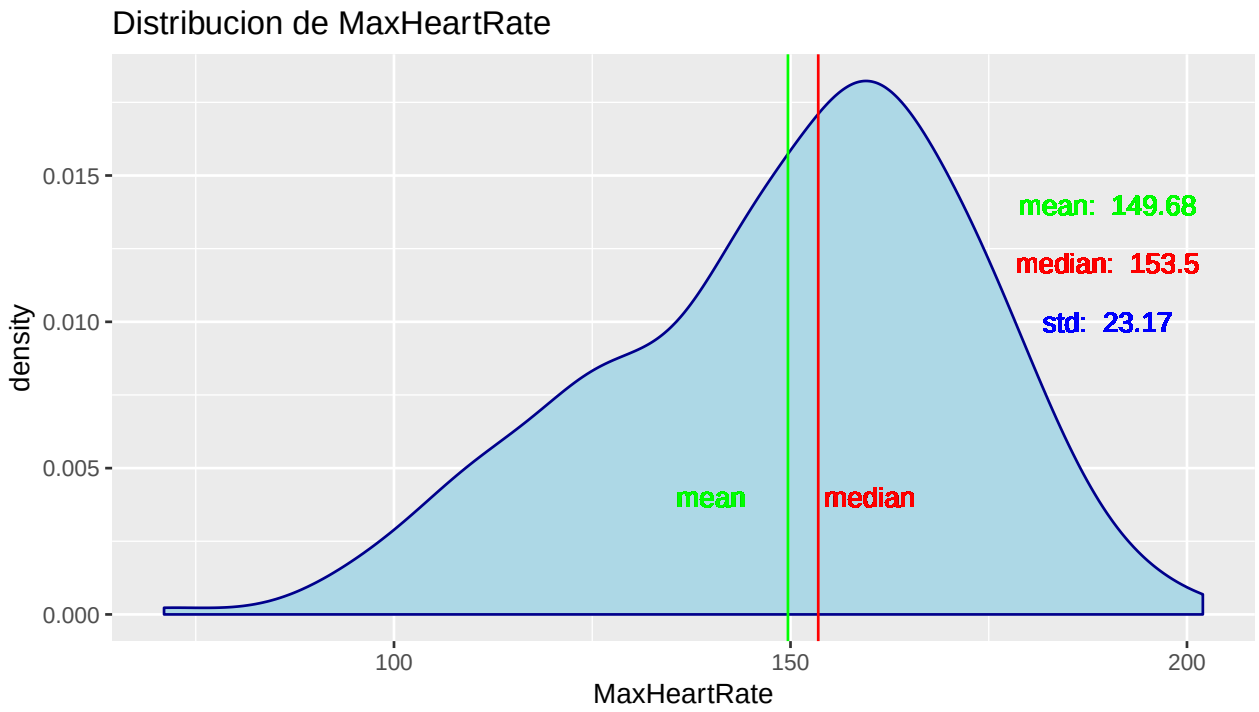
Siguiendo con las distribuciones del resto de variables continuas:



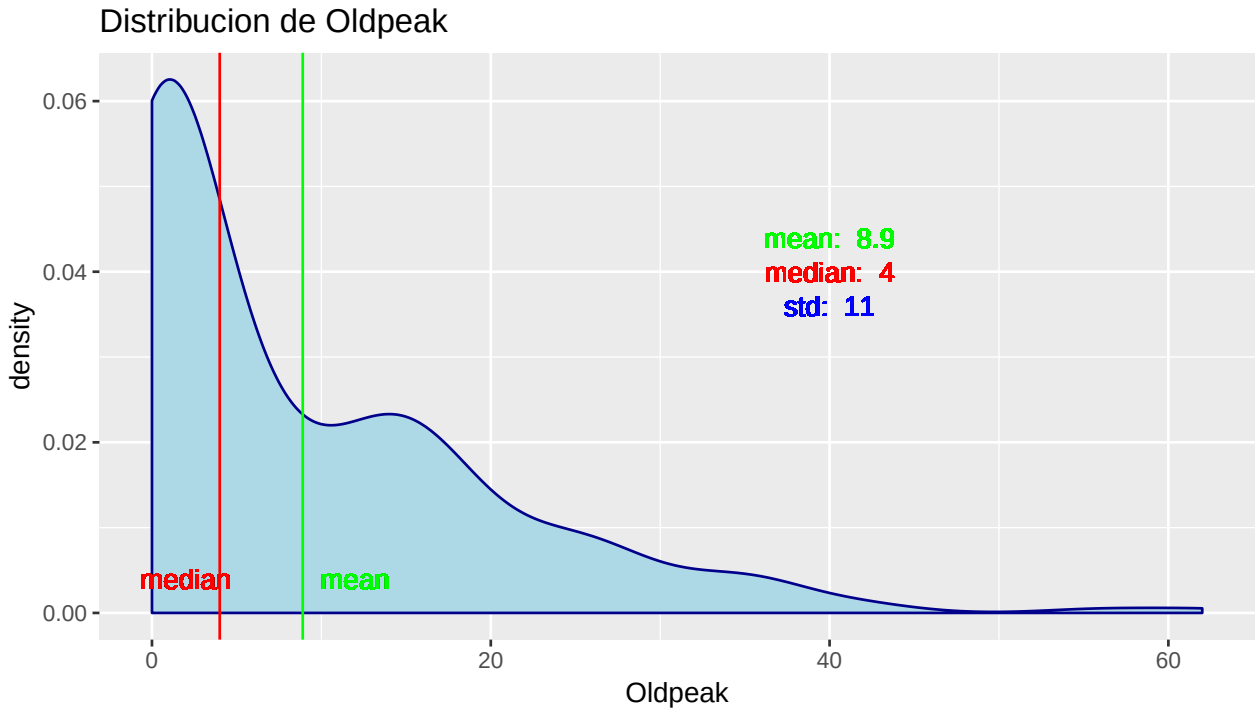
Se presenta una distribución aceptable, una distribución normal con cola a la derecha.



Se presenta una buena distribución normal con un elemento en el extremo derecho que seguramente se tratará de un 'outlier'. Sin embargo, dado que ese dato puede ser un valor 'natural', es decir, no que haya sido un error de medición, se decide mantenerlo en el conjunto de datos.



Para esta variable se tiene una distribución más pobre, con una ligera cola a la izquierda.



Por último, esta variable posee la peor distribución de todas, muy alejada de una distribución normal.

Normalización

Observando que no todas las distribuciones de las variables son todo lo buenas que quisiésemos vamos a normalizar estas variables continuas, creando un nuevo 'dataset' modificado, buscando si se comporta mejor los modelos que construyamos con éstas.

```
#Normalización del dataset
heartNormalizado <- data.frame(heart)
heartNormalizado$RestBloodPressure <- scale(heartNormalizado$RestBloodPressure)
heartNormalizado$SerumCholestoral <- scale(heartNormalizado$SerumCholestoral)
heartNormalizado$MaxHeartRate <- scale(heartNormalizado$MaxHeartRate)
heartNormalizado$Oldpeak <- scale(heartNormalizado$Oldpeak)
```

Así conseguimos que la desviación típica para estas variables continuas sea de 1.

Análisis de correlaciones

Por último vamos a estudiar las correlaciones que existen entre las diferentes variables del dataset a excepción de la variable de salida. Esto se realiza para buscar si hay alguna variable muy correlacionada con otra y que pueda ser redundante, pudiendo así prescindir de ella.

```
#Maxima correlacion entre las variables predictoras
max(abs(cor(heart[-14]))%%1)
```

```
## [1] 0.5261103
```

Sin embargo, descubrimos que la mayor correlación entre alguna de las variables, claramente que no sean las mismas entre ellas, es tan sólo de 0.52 por lo que no hay redundancia en este aspecto y no se prescinde de ninguna a priori.

Análisis del Dataset 'delta_ail'

Este 'dataset' está relacionado con el control de los alerones de los aviones. Así, a partir de otras variables se trata de estimar una variación sobre dichos alerones, siendo así un problema de regresión.

```
#Dimensiones del dataset delta_Ail
dim(deltaAil)
```

```
## [1] 7129    6
```

Se observa que tenemos un total de 7129 observaciones con 5 factores y una variable a predecir, 'Sa'.

Estos son los nombres de las variables:

```
#Nombre de las variables del dataset
colnames(deltaAil)
```

```
## [1] "RollRate"     "PitchRate"    "currPitch"    "currRoll"
## [5] "diffRollRate" "Sa"
```

Y aquí el tipo de dato atómico de cada una de ellas:

```
#Tipo atomico de cada variable
unlist(lapply(deltaAil,class))
```

```
##     RollRate    PitchRate    currPitch     currRoll diffRollRate
##    "numeric"    "numeric"    "numeric"    "numeric"    "numeric"
##           Sa
##    "numeric"
```

En esta ocasión todas las variables son 'reales'.

```
#Estructura del dataset y de la variable de salida
class(deltaAil)
```

```
## [1] "data.frame"
```

```
min(deltaAil$Sa)
```

```
## [1] -0.0021
```

```
max(deltaAil$Sa)
```

```
## [1] 0.0022
```

Nuevamente, la estructura del 'dataset' es de un dataframe. Dado que este 'dataset' es utilizado como un problema de regresión, la variable de salida es ahora continua, desde -0.0021 hasta 0.0022.

Buscando una primera idea sobre las variables predictoras respecto a la variable de salida:

```
#Correlación respecto a la variable de salida
cor(deltaAil)[6,]
```

```
##     RollRate    PitchRate    currPitch     currRoll diffRollRate
##  -0.77230803  -0.02063738   0.17135286   0.06018298  -0.56296620
##           Sa
##   1.00000000
```

En esta ocasión observamos que la variable 'RollRate' tiene una correlación, aunque negativa, del 77% por lo que será bastante interesante empezar a trabajar por esta variable. Estudiando el resto de las correlaciones, la variable 'diffRollRate' se sitúa en el 56% y el resto tienen unas correlaciones muy bajas por lo que, en principio, las variables 'RollRate' y 'diffRollRate' serán el 'eje central' de los modelos que se construyan.

Eliminación de 'missing values' y observaciones repetidas

Nuevamente comprobaremos si hay algún 'missing value' en nuestro dataset que debiese ser tratado:

```
#Comprobación de missing values
which(is.na(deltaAil) == TRUE)
```

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

Sin embargo, no hay ningún dato perdido en el conjunto de datos. Al igual que para el dataset anterior, es importante también comprobar si hay alguna observación repetida:

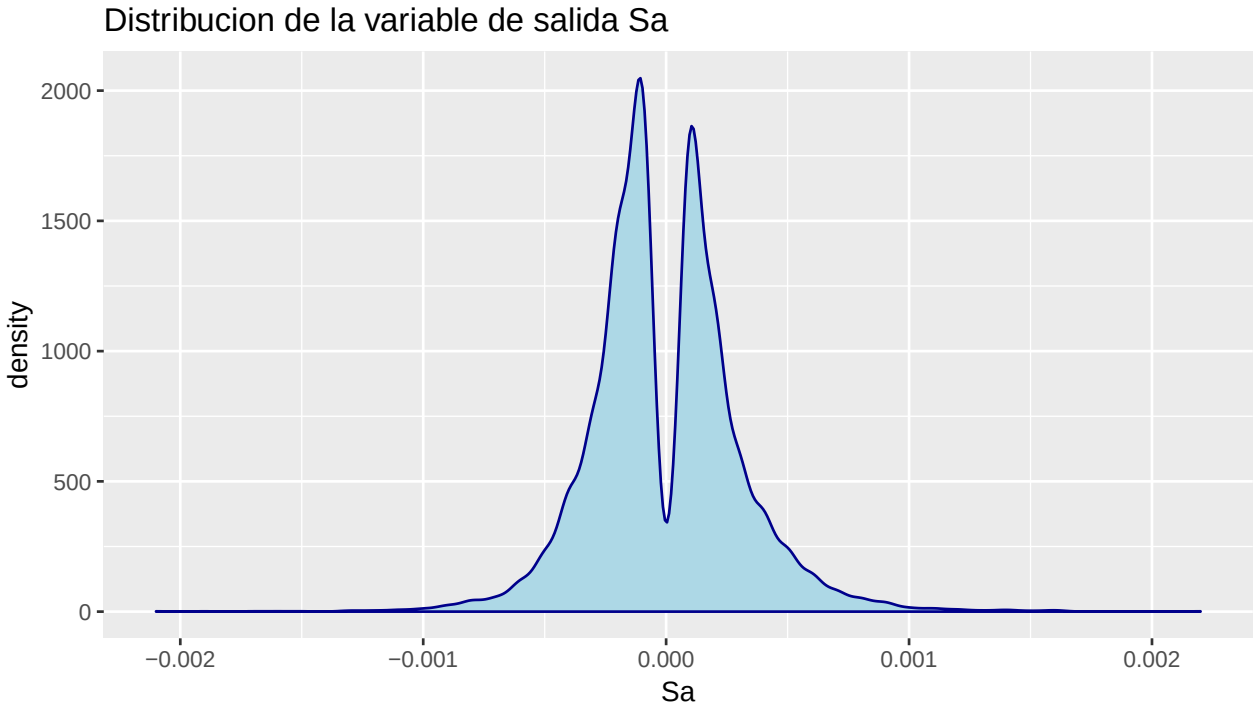
```
#Comprobación de valores duplicados
which(duplicated(deltaAil) == TRUE)
```

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

No tenemos que eliminar ninguna pues todas las observaciones son únicas.

Datos de salida equilibrados

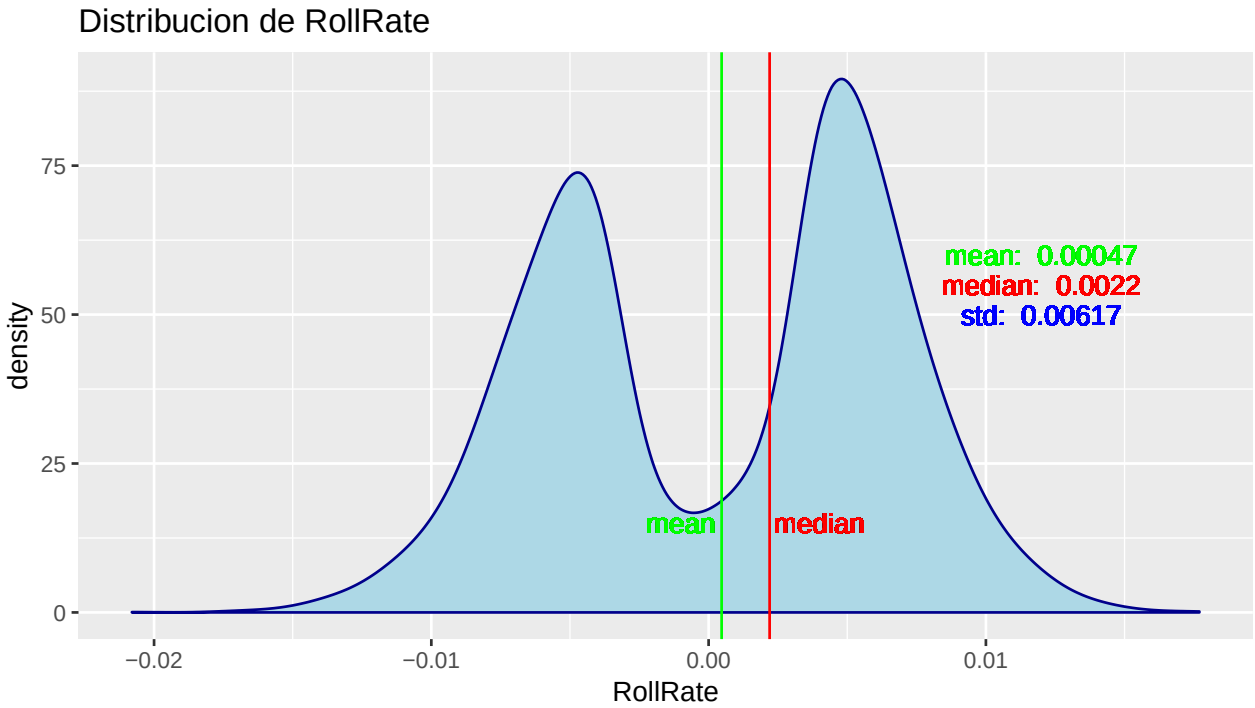
Dado que ahora se está trabajando con un problema de regresión, lo ideal sería que, respecto a la variable de salida, los datos siguiesen una distribución normal.



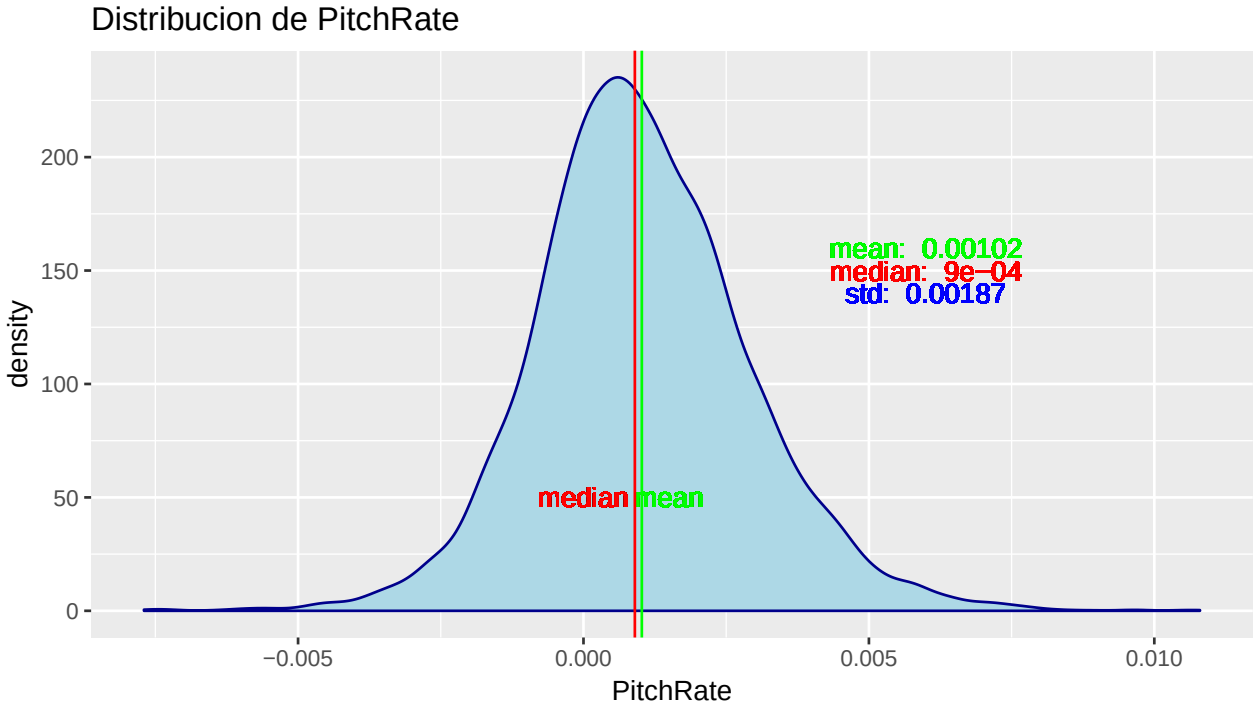
Esta distribución se corresponde con una distribución bimodal, teniendo dos "modas" en ambos máximos de la gráfica. Además, ambas modas se corresponden con que se haga una modificación positiva o negativa en los alerones, por lo que podemos decir que están bastante bien diferenciados los datos respecto al tipo de modificación, aunque al ser un problema de regresión, nuestro objetivo es saber cuánta modificación se debe hacer.

Medidas de distribución sobre el conjunto de datos

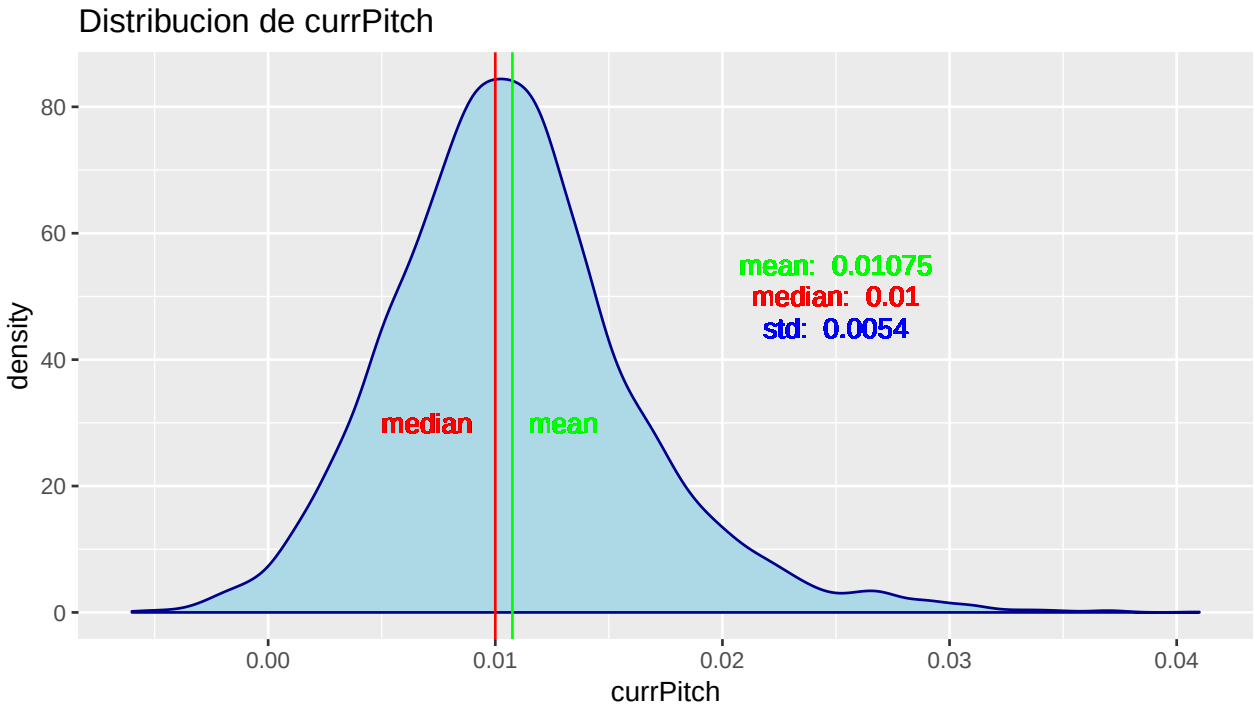
Dado que todas las variables de este conjunto de datos son continuas, procedemos a estudiar las distribuciones de cada una de ellas:



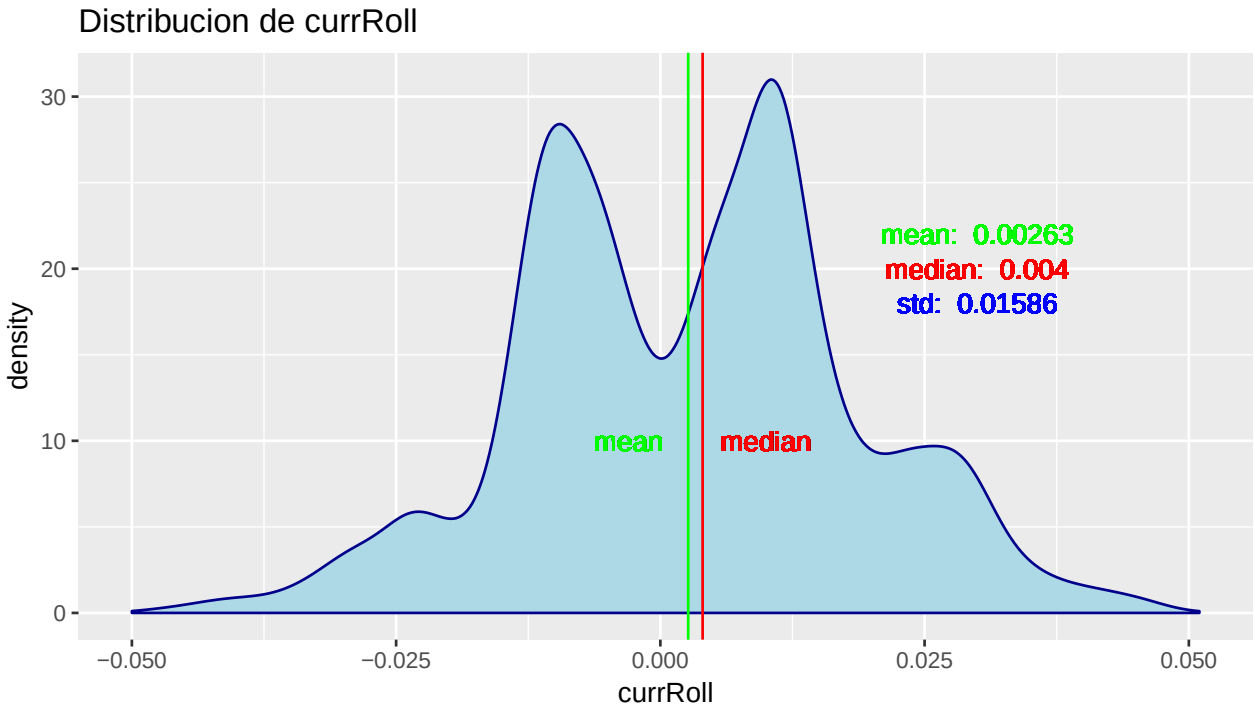
Nuevamente nos encontramos con una distribución bimodal, bastante diferenciada en si el valor es negativo o positivo.



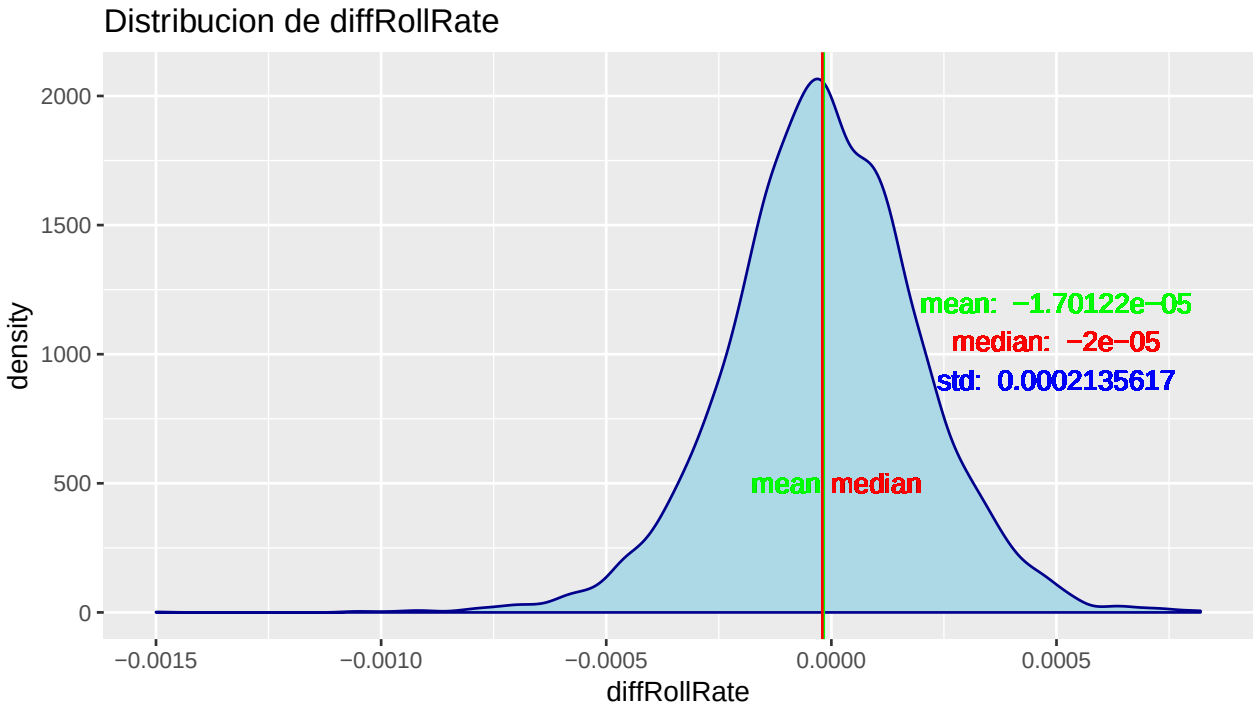
Para la variable 'PitchRate' sí que se tiene una distribución normal.



Esta variable también presenta una buena distribución normal con una ligera cola a la derecha.



Al contrario de lo que se estaba descubriendo anteriormente, para esta variable no tenemos una buena distribución de los datos.



Por último, para esta variable volvemos a tener una distribución normal con una ligera cola a la izquierda.

Normalización

Al igual que hicimos para el problema anterior, vamos a realizar una normalización sobre los datos para conseguir que su desviación típica sea de 1 y estudiar si con estos datos se consiguen modelos con un mejor comportamiento.

```
#Normalización del dataset
deltaAilNormalizado <- data.frame(deltaAil)

deltaAilNormalizado$RollRate <- scale(deltaAil$RollRate)
deltaAilNormalizado$PitchRate <- scale(deltaAil$PitchRate)
deltaAilNormalizado$currPitch <- scale(deltaAil$currPitch)
deltaAilNormalizado$currRoll <- scale(deltaAil$currRoll)
deltaAilNormalizado$diffRollRate <- scale(deltaAil$diffRollRate)
```

Análisis de correlaciones

Por último, vamos a analizar las correlaciones entre las variables predictoras para descubrir si hay alguna que nos esté dando la misma información que otra y por lo tanto no sea relevante:

```
#Correlación entre variable predictoras
max(abs(cor(deltaAil[-6]))%%1)
```

```
## [1] 0.6389178
```

Descubrimos que la máxima correlación encontrada no es muy relevante por lo que podemos concluir que no hay dos variables demasiado similares entre ellas.

Regresión

Una vez estudiados y tratados los datos de 'deltaAil', procedemos a construir los modelos que puedan precedir futuras observaciones en este problema de regresión.

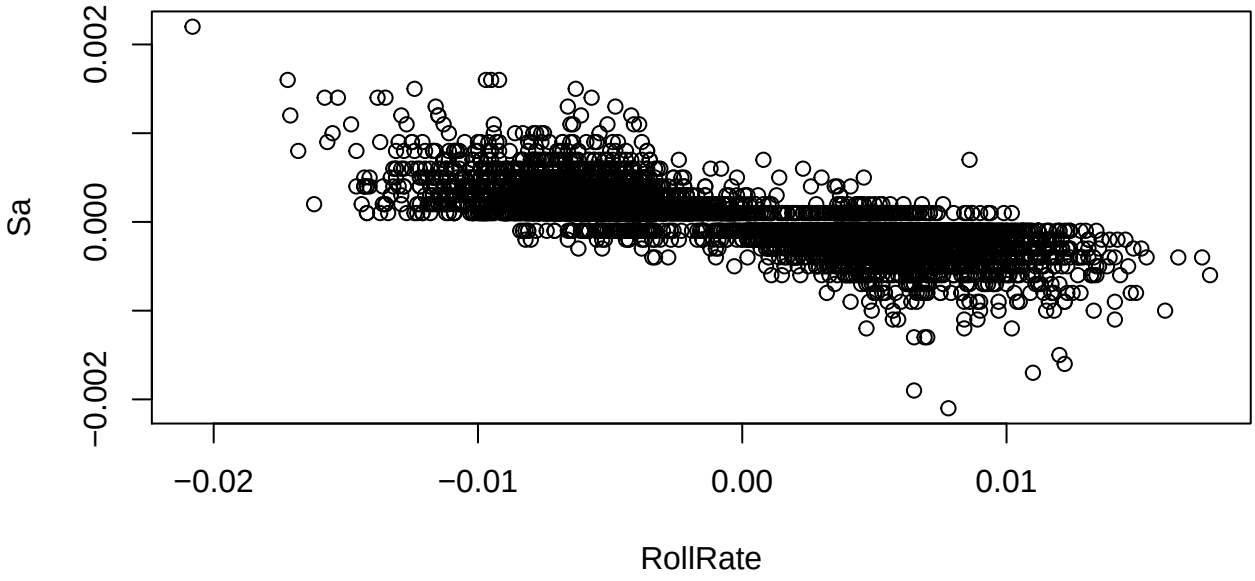
Regresión lineal simple

Empezamos por el algoritmo más simple de todos, la regresión lineal simple. Aunque al tener 5 regresores en el conjunto de datos se realizará un modelo simple para cada uno de ellos, podemos saber por dónde empezar gracias al estudio de correlaciones que se hizo en el anterior apartado de análisis de datos.

```
#Correlación de las variables respecto a la variable de salida
cor(deltaAil)[6,]
```

```
##     RollRate    PitchRate     currPitch     currRoll diffRollRate
##  -0.77230803  -0.02063738   0.17135286   0.06018298  -0.56296620
##           Sa
##   1.00000000
```

Destaca 'RollRate' con un 77.2% de correlación negativa. De hecho, podemos visualizar esta correlación:



Construcción de los modelos lineales simples

```
#Construcción modelo simple con RollRate
fit1=lm(Sa~RollRate,data=deltaAil)
summary(fit1)
```

```
##
## Call:
## lm(formula = Sa ~ RollRate, data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.815e-03 -1.025e-04 -1.660e-06  9.776e-05  1.401e-03
##
```

```
## Coefficients:
##               Estimate Std. Error  t value Pr(>|t|)
## (Intercept)  1.070e-05  2.285e-06    4.682 2.89e-06 ***
## RollRate    -3.790e-02  3.693e-04 -102.636  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001924 on 7127 degrees of freedom
## Multiple R-squared:  0.5965, Adjusted R-squared:  0.5964
## F-statistic: 1.053e+04 on 1 and 7127 DF,  p-value: < 2.2e-16
```

Acierto del 59.64%

```
#Construcción modelo simple con PitchRate
fit2=lm(Sa~PitchRate,data=deltaAil)
summary(fit2)
```

```
##
## Call:
## lm(formula = Sa ~ PitchRate, data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -2.069e-03 -1.920e-04 -8.766e-05  2.010e-04  2.215e-03
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -3.657e-06  4.086e-06  -0.895   0.3709
## PitchRate   -3.340e-03  1.917e-03  -1.743   0.0814 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0003028 on 7127 degrees of freedom
## Multiple R-squared:  0.0004259,  Adjusted R-squared:  0.0002856
## F-statistic: 3.037 on 1 and 7127 DF,  p-value: 0.08144
```

Acierto del 0.02%

```
#Construcción modelo simple con currPitch
fit3=lm(Sa~currPitch,data=deltaAil)
summary(fit3)
```

```
##
## Call:
## lm(formula = Sa ~ currPitch, data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -2.076e-03 -1.761e-04 -2.806e-05  1.759e-04  2.185e-03
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.104e-04  7.873e-06  -14.02   <2e-16 ***
## currPitch    9.604e-03  6.541e-04   14.68   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0002984 on 7127 degrees of freedom
## Multiple R-squared:  0.02936,    Adjusted R-squared:  0.02923
## F-statistic: 215.6 on 1 and 7127 DF,  p-value: < 2.2e-16
```

Acierto del 2.92%

```
#Construcción modelo simple con currRoll
fit4=lm(Sa~currRoll,data=deltaAil)
summary(fit4)
```

```
##
## Call:
## lm(formula = Sa ~ currRoll, data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -2.082e-03 -1.888e-04 -7.726e-05  1.871e-04  2.188e-03
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.010e-05  3.630e-06  -2.782  0.00542 **
## currRoll     1.149e-03  2.258e-04   5.090 3.67e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0003023 on 7127 degrees of freedom
## Multiple R-squared:  0.003622,   Adjusted R-squared:  0.003482
## F-statistic: 25.91 on 1 and 7127 DF,  p-value: 3.673e-07
```

Acierto del 0.34%

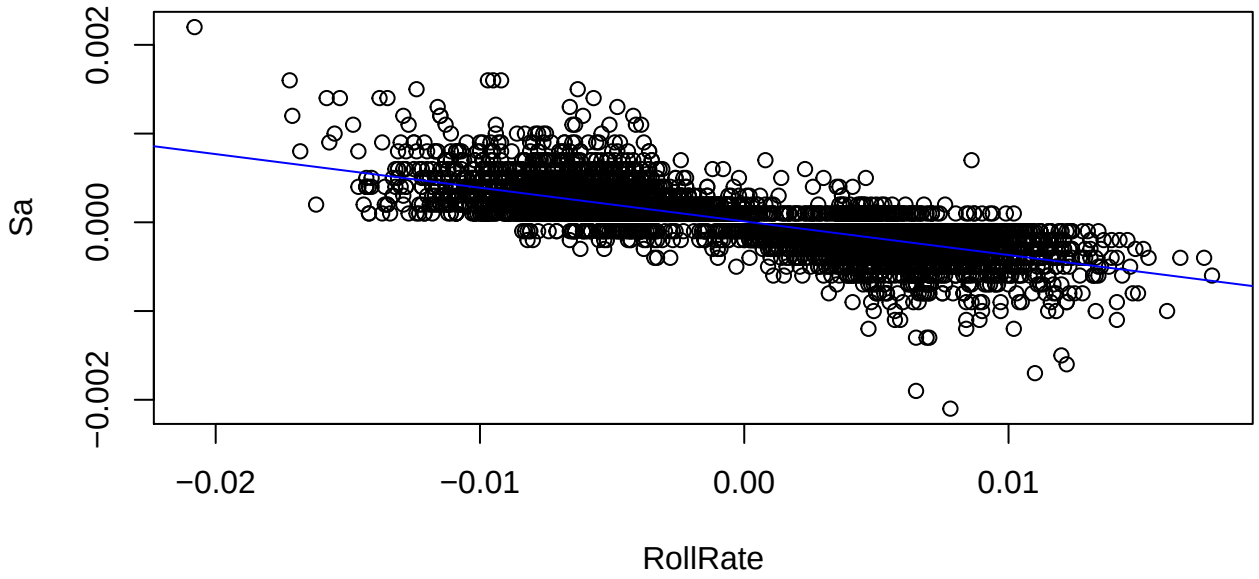
```
#Construcción modelo simple con diffRollRate
fit5=lm(Sa~diffRollRate,data=deltaAil)
summary(fit5)
```

```
##
## Call:
## lm(formula = Sa ~ diffRollRate, data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.640e-03 -1.554e-04 -7.080e-06  1.610e-04  1.589e-03
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -2.065e-05  2.974e-06  -6.944 4.15e-12 ***
## diffRollRate -7.983e-01  1.388e-02 -57.505  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0002503 on 7127 degrees of freedom
```

```
## Multiple R-squared:  0.3169, Adjusted R-squared:  0.3168
## F-statistic:  3307 on 1 and 7127 DF,  p-value: < 2.2e-16
```

Acierto del 31.68%

Se obtienen en general una modelos bastantes pobres utilizando regresión lineal simple. El que obtiene un mejor resultado es el construido con la variable 'RollRate' con un 59.64% de acierto, coincidiendo con la variable que mostró anteriormente una mayor correlación.




```
##                     2.5 %        97.5 %
## (Intercept)  6.220158e-06  1.517974e-05
## RollRate    -3.862594e-02 -3.717812e-02
```

También podemos observar la predicción que haría este modelo para nuevos datos así como hacer el cálculo manual del RMSE para el conjunto de test aunque se utilice aquí el propio conjunto de entrenamiento:

```
#Ejemplo de predicción del modelo simple
predict(fit1,data.frame(RollRate=c(0.01,-0.01,1)))
```

```
##             1             2             3
## -0.0003683203  0.0003897202 -0.0378913293
```

```
#Cálculo del RMSE para el modelo simple construido
yprime=predict(fit1,data.frame(RollRate=deltaAil$RollRate))
sqrt(sum(abs(deltaAil$Sa-yprime)^2)/length(yprime))
```

```
## [1] 0.000192371
```

En esta regresión lineal simple no es necesario utilizar datos normalizados pues se obtendrán los mismos resultados, al fin y al cabo, se tienen las mismas correlaciones:

```
#Modelo simple con datos normalizados
fit1n=lm(Sa~RollRate,data=deltaAilNormalizado)
summary(fit1n)
```

```
##
## Call:
## lm(formula = Sa ~ RollRate, data = deltaAilNormalizado)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.815e-03 -1.025e-04 -1.660e-06  9.776e-05  1.401e-03
##
## Coefficients:
##               Estimate Std. Error  t value Pr(>|t|)
## (Intercept) -7.070e-06  2.279e-06   -3.103  0.00193 **
## RollRate    -2.339e-04  2.279e-06 -102.636  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001924 on 7127 degrees of freedom
## Multiple R-squared:  0.5965, Adjusted R-squared:  0.5964
## F-statistic: 1.053e+04 on 1 and 7127 DF,  p-value: < 2.2e-16
```

Regresión lineal múltiple

Es este tipo de regresión se realiza una combinación de diferentes variables predictoras. Si hacemos un breve estudio de las correlaciones entre las diferentes variables predictoras, podremos descubrir "parejas" de variables interesantes con las que, si alguna de las variables es interesante para el modelo, posiblemente la otra variable de la pareja también lo sea:

```
#Corerlaciones entre las variables predictoras
cor(deltaAil)[-6,-6]
```

```
##                RollRate   PitchRate   currPitch    currRoll diffRollRate
## RollRate      1.00000000  0.06640862 -0.16003341 -0.04180944    0.4803962
## PitchRate     0.06640862  1.00000000  0.17910836  0.02400816   -0.1111336
## currPitch    -0.16003341  0.17910836  1.00000000  0.02642593   -0.1538780
## currRoll     -0.04180944  0.02400816  0.02642593  1.00000000   -0.6389178
## diffRollRate  0.48039620 -0.11113356 -0.15387803 -0.63891775    1.0000000
```

Los valores más elevados son las correlaciones de "diffRollRate" con "currRoll" y "diffRollRate" con "RollRate" por lo que serán parejas interesantes de predictores a tener en cuenta en este tipo de modelos.

Dado que estudiar cada una de las combinaciones posibles de variables sería demasiado costoso, se procede a hacer el estudio con todas las variables y, a continuación, ir presciendiendo de las menos relevantes con el fin de conseguir un modelo más simple con el mismo acierto.

```
#Construcción de modelo lineal múltiple
fit6 = lm(Sa~.,data=deltaAil)
summary(fit6)
```

```
##
## Call:
## lm(formula = Sa ~ ., data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.530e-03 -8.761e-05  3.150e-06  9.516e-05  1.211e-03
```

```
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.653e-06  4.652e-06  -0.355 0.722403
## RollRate     -2.729e-02  4.179e-04 -65.312  < 2e-16 ***
## PitchRate    -5.339e-03  1.133e-03  -4.714 2.47e-06 ***
## currPitch     1.362e-03  3.902e-04   3.491 0.000483 ***
## currRoll     -4.900e-03  1.832e-04 -26.750  < 2e-16 ***
## diffRollRate -6.519e-01  1.573e-02 -41.443  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.000172 on 7123 degrees of freedom
## Multiple R-squared:  0.6777, Adjusted R-squared:  0.6775
## F-statistic:  2995 on 5 and 7123 DF,  p-value: < 2.2e-16
```

Se consigue una importante mejora al utilizar todas las variables, pasando del 59.64% de acierto hasta un 67.75%. Ahora debemos ir eliminando las variables más irrelevantes, fijándonos en los p-values más altos:

```
#Construcción de modelo lineal múltiple
fit7 = lm(Sa~.-currPitch,data=deltaAil)
summary(fit7)
```

```
##
## Call:
## lm(formula = Sa ~ . - currPitch, data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.534e-03 -8.831e-05  2.660e-06  9.441e-05  1.237e-03
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.236e-05  2.353e-06   5.252 1.55e-07 ***
## RollRate     -2.744e-02  4.162e-04 -65.930  < 2e-16 ***
## PitchRate    -4.646e-03  1.116e-03  -4.163 3.17e-05 ***
## currRoll     -4.925e-03  1.832e-04 -26.883  < 2e-16 ***
## diffRollRate -6.557e-01  1.570e-02 -41.752  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001721 on 7124 degrees of freedom
## Multiple R-squared:  0.6771, Adjusted R-squared:  0.677
## F-statistic:  3735 on 4 and 7124 DF,  p-value: < 2.2e-16
```

Compensa que haya habido una ligera reducción en el acierto a cambio de un modelo más interpretable.

```
#Construcción de modelo lineal múltiple
fit8 = lm(Sa~.-currPitch-PitchRate,data=deltaAil)
summary(fit8)
```

```
##
## Call:
## lm(formula = Sa ~ . - currPitch - PitchRate, data = deltaAil)
```

```
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.572e-03 -8.976e-05  3.100e-06  9.489e-05  1.254e-03
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   7.728e-06  2.076e-06   3.722 0.000199 ***
## RollRate     -2.774e-02  4.101e-04 -67.643  < 2e-16 ***
## currRoll     -4.828e-03  1.819e-04 -26.540  < 2e-16 ***
## diffRollRate -6.423e-01  1.539e-02 -41.734  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001723 on 7125 degrees of freedom
## Multiple R-squared:  0.6764, Adjusted R-squared:  0.6762
## F-statistic:  4963 on 3 and 7125 DF,  p-value: < 2.2e-16
```

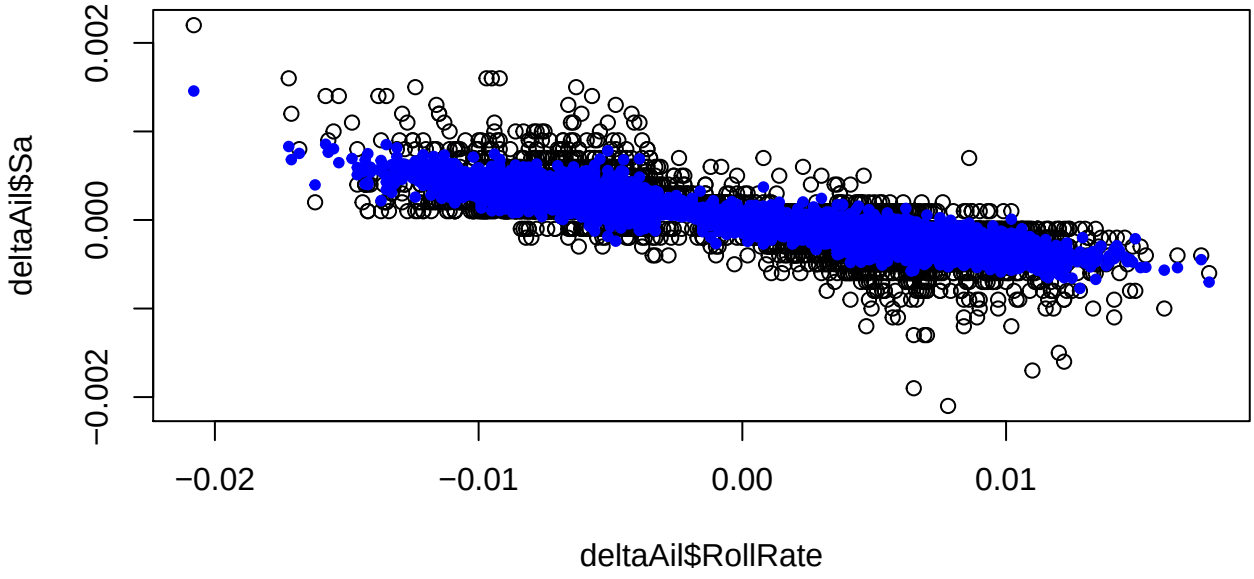
Similar al caso anterior, se ha perdido muy poco en el acierto. Hemos conseguido un acierto del 67.62% con un modelo "sencillo" con tan solo 3 variables. Dado que se indica ahora que para todas las variables restante hay un p-value muy bajo, todas ellas son relevantes para el modelo y no hay que prescindir ya de ninguna.

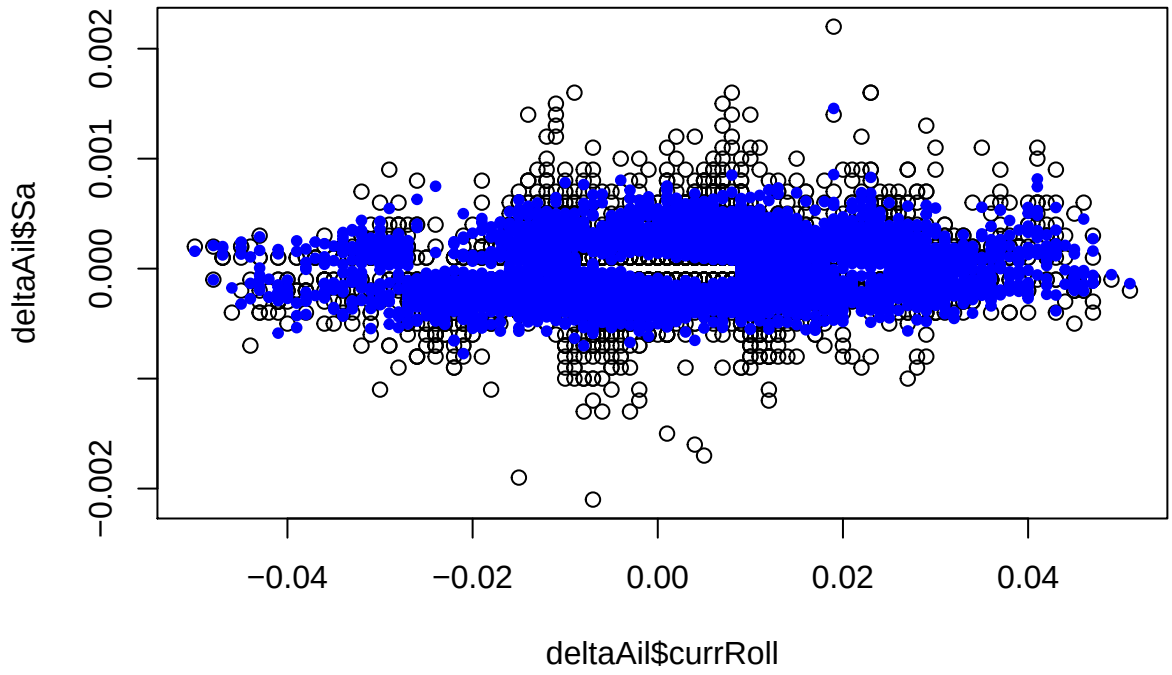
Calculando ahora el error RMSE:

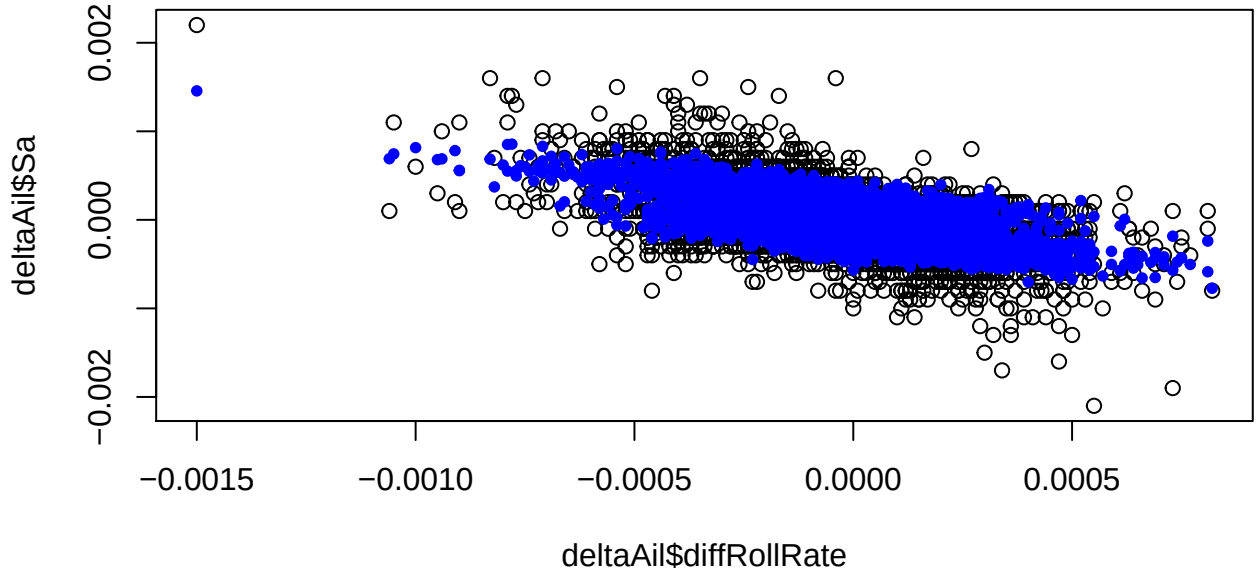
```
#Cálculo de RMSE para el modelo lineal múltiple 8
yprime=predict(fit8,deltaAil)
sqrt(sum(abs(deltaAil$Sa-yprime)^2)/length(yprime))
```

```
## [1] 0.0001722785
```

Podemos incluso dibujar este modelo, en cada una de las variables, para observar su comportamiento:







Interacciones y no-linealidad

En este apartado se trata de descubrir otro tipo de interacciones más complejas entre las variables del modelo que ya tenemos con el fin de que un comportamiento más complejo se traduzca en un mayor acierto.

```
#Construcción de modelo múltiple con interacciones
fit9 = lm(Sa~RollRate+I(RollRate^2),data=deltaAil)
summary(fit9)
```

```
##
## Call:
## lm(formula = Sa ~ RollRate + I(RollRate^2), data = deltaAil)
```

```
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.826e-03 -9.914e-05  1.930e-06  9.764e-05  1.250e-03
##
## Coefficients:
##                 Estimate Std. Error  t value Pr(>|t|)
## (Intercept)   -7.811e-06  3.348e-06   -2.333   0.0197 *
## RollRate      -3.790e-02  3.678e-04 -103.041  < 2e-16 ***
## I(RollRate^2)  4.834e-01  6.411e-02    7.541 5.26e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001916 on 7126 degrees of freedom
## Multiple R-squared:  0.5997, Adjusted R-squared:  0.5995
## F-statistic:  5337 on 2 and 7126 DF,  p-value: < 2.2e-16
```

Se obtiene un acierto de 59.95% con una sola variable. Cuando se construyó el modelo simple para esta misma variable se obtuvo un acierto del 59.64% por lo que se ha conseguido una ligera mejorar al incluir una interacción consigo misma.

Así, se procede a ir construyendo diferentes modelos con diferentes interacciones, buscando mejorar el acierto siempre que el modelo no sea demasiado complejo. Al igual que en las construcciones de los modelos anteriores, nos iremos fijando también en los p-values para descubrir variables que no estén aportando al modelo:

```
#Construcción de modelo múltiple con interacciones
fit10 = lm(Sa~RollRate+diffRollRate+I(RollRate^2)+I(diffRollRate^2)+I((RollRate+diffRollRate)^2),
           data=deltaAil)
summary(fit10)
```

```
##
## Call:
## lm(formula = Sa ~ RollRate + diffRollRate + I(RollRate^2) + I(diffRollRate^2) +
##     I((RollRate + diffRollRate)^2), data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.704e-03 -9.487e-05  6.450e-06  9.886e-05  1.267e-03
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                   -1.343e-05  3.239e-06  -4.148  3.4e-05 ***
## RollRate                      -3.178e-02  4.015e-04 -79.164  < 2e-16 ***
## diffRollRate                  -3.501e-01  1.168e-02 -29.984  < 2e-16 ***
## I(RollRate^2)                 -3.371e+00  1.003e+00  -3.360 0.000785 ***
## I(diffRollRate^2)              3.880e+01  3.256e+01   1.192 0.233460
## I((RollRate + diffRollRate)^2) 3.603e+00  9.772e-01   3.687 0.000229 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001798 on 7123 degrees of freedom
## Multiple R-squared:  0.6477, Adjusted R-squared:  0.6475
## F-statistic:  2619 on 5 and 7123 DF,  p-value: < 2.2e-16
```

Acierto de 64.75%.

```r
#Construcción de modelo múltiple con interacciones
fit11 = lm(Sa~RollRate+diffRollRate+currRoll+I((RollRate+diffRollRate+currRoll)^2),
           data=deltaAil)
summary(fit11)
```

```
##
## Call:
## lm(formula = Sa ~ RollRate + diffRollRate + currRoll + I((RollRate +
##     diffRollRate + currRoll)^2), data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.569e-03 -8.911e-05  3.610e-06  9.470e-05  1.252e-03
##
## Coefficients:
##                                           Estimate Std. Error t value
## (Intercept)                               4.893e-06  2.533e-06   1.932
## RollRate                                 -2.780e-02  4.110e-04 -67.631
## diffRollRate                             -6.430e-01  1.539e-02 -41.773
## currRoll                                 -4.901e-03  1.856e-04 -26.399
## I((RollRate + diffRollRate + currRoll)^2)  1.056e-02  5.407e-03   1.954
##                                           Pr(>|t|)
## (Intercept)                                 0.0534 .
## RollRate                                    <2e-16 ***
## diffRollRate                                <2e-16 ***
## currRoll                                    <2e-16 ***
## I((RollRate + diffRollRate + currRoll)^2)   0.0508 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001723 on 7124 degrees of freedom
## Multiple R-squared:  0.6765, Adjusted R-squared:  0.6763
## F-statistic:  3725 on 4 and 7124 DF,  p-value: < 2.2e-16
```

Acierto de 67.63%.

```r
#Construcción de modelo múltiple con interacciones
fit12 = lm(Sa~RollRate+PitchRate+currPitch+currRoll+diffRollRate+I((RollRate+PitchRate+currPitch
          +currRoll+diffRollRate)^2),data=deltaAil)
summary(fit12)
```

```
##
## Call:
## lm(formula = Sa ~ RollRate + PitchRate + currPitch + currRoll +
##     diffRollRate + I((RollRate + PitchRate + currPitch + currRoll +
##     diffRollRate)^2), data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.530e-03 -8.738e-05  3.130e-06  9.519e-05  1.211e-03
##
## Coefficients:
##                                                                  Estimate
```

```
## (Intercept)                                                    -1.628e-06
## RollRate                                                       -2.728e-02
## PitchRate                                                      -5.327e-03
## currPitch                                                       1.370e-03
## currRoll                                                       -4.892e-03
## diffRollRate                                                   -6.519e-01
## I((RollRate + PitchRate + currPitch + currRoll + diffRollRate)^2) -2.772e-04
##                                                                Std. Error
## (Intercept)                                                     4.671e-06
## RollRate                                                        4.410e-04
## PitchRate                                                       1.150e-03
## currPitch                                                       4.135e-04
## currRoll                                                        2.327e-04
## diffRollRate                                                    1.574e-02
## I((RollRate + PitchRate + currPitch + currRoll + diffRollRate)^2)  4.763e-03
##                                                                t value
## (Intercept)                                                     -0.349
## RollRate                                                       -61.870
## PitchRate                                                       -4.630
## currPitch                                                        3.314
## currRoll                                                       -21.024
## diffRollRate                                                   -41.426
## I((RollRate + PitchRate + currPitch + currRoll + diffRollRate)^2)  -0.058
##                                                                Pr(>|t|)
## (Intercept)                                                    0.727424
## RollRate                                                        < 2e-16
## PitchRate                                                      3.72e-06
## currPitch                                                      0.000924
## currRoll                                                        < 2e-16
## diffRollRate                                                    < 2e-16
## I((RollRate + PitchRate + currPitch + currRoll + diffRollRate)^2) 0.953588
##
## (Intercept)
## RollRate                                                       ***
## PitchRate                                                      ***
## currPitch                                                      ***
## currRoll                                                       ***
## diffRollRate                                                   ***
## I((RollRate + PitchRate + currPitch + currRoll + diffRollRate)^2)
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.000172 on 7122 degrees of freedom
## Multiple R-squared:  0.6777, Adjusted R-squared:  0.6774
## F-statistic:  2496 on 6 and 7122 DF,  p-value: < 2.2e-16
```

Acierto de 67.74%.

```
#Construcción de modelo múltiple con interacciones
fit13 = lm(Sa~RollRate+currRoll+diffRollRate+RollRate*currRoll+RollRate*diffRollRate,data=deltaAil)
summary(fit13)
```

```
##
## Call:
```

```
## lm(formula = Sa ~ RollRate + currRoll + diffRollRate + RollRate *
##     currRoll + RollRate * diffRollRate, data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.611e-03 -8.681e-05  5.450e-06  9.523e-05  1.250e-03
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)             1.029e-06  2.373e-06   0.434 0.664473
## RollRate               -2.770e-02  4.126e-04 -67.135  < 2e-16 ***
## currRoll               -4.726e-03  1.827e-04 -25.860  < 2e-16 ***
## diffRollRate           -6.336e-01  1.544e-02 -41.030  < 2e-16 ***
## RollRate:currRoll       9.604e-02  2.826e-02   3.398 0.000682 ***
## RollRate:diffRollRate   1.093e+01  1.895e+00   5.768 8.37e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001719 on 7123 degrees of freedom
## Multiple R-squared:  0.6779, Adjusted R-squared:  0.6776
## F-statistic:  2998 on 5 and 7123 DF,  p-value: < 2.2e-16
```

Acierto de 67.76%.

```
#Construcción de modelo múltiple con interacciones
fit14 = lm(Sa~RollRate+currRoll+diffRollRate+RollRate*currRoll+RollRate*diffRollRate+I(RollRate^2)
          +I(currRoll^2)+I(diffRollRate^2),data=deltaAil)
summary(fit14)
```

```
##
## Call:
## lm(formula = Sa ~ RollRate + currRoll + diffRollRate + RollRate *
##     currRoll + RollRate * diffRollRate + I(RollRate^2) + I(currRoll^2) +
##     I(diffRollRate^2), data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.592e-03 -8.812e-05  5.870e-06  9.408e-05  1.246e-03
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)            -3.881e-06  3.406e-06  -1.139 0.254545
## RollRate               -2.759e-02  4.175e-04 -66.078  < 2e-16 ***
## currRoll               -4.902e-03  1.899e-04 -25.816  < 2e-16 ***
## diffRollRate           -6.444e-01  1.598e-02 -40.320  < 2e-16 ***
## I(RollRate^2)           3.359e-02  6.720e-02   0.500 0.617177
## I(currRoll^2)           2.278e-02  6.544e-03   3.480 0.000504 ***
## I(diffRollRate^2)      -9.017e+01  3.417e+01  -2.639 0.008330 **
## RollRate:currRoll       1.131e-01  3.035e-02   3.728 0.000194 ***
## RollRate:diffRollRate   1.438e+01  2.595e+00   5.540 3.14e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001718 on 7120 degrees of freedom
```

```
## Multiple R-squared:  0.6785, Adjusted R-squared:  0.6782
## F-statistic:  1878 on 8 and 7120 DF,  p-value: < 2.2e-16
```

Acierto de 67.82%.

```
#Construcción de modelo múltiple con interacciones
fit15 = lm(Sa~RollRate+currRoll+diffRollRate+((RollRate+currRoll+diffRollRate)^2)+I(RollRate^2)
           +I(currRoll^2)+I(diffRollRate^2),data=deltaAil)
summary(fit15)
```

```
##
## Call:
## lm(formula = Sa ~ RollRate + currRoll + diffRollRate + ((RollRate +
##     currRoll + diffRollRate)^2) + I(RollRate^2) + I(currRoll^2) +
##     I(diffRollRate^2), data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.586e-03 -8.824e-05  6.380e-06  9.441e-05  1.245e-03
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -3.023e-06  3.437e-06  -0.880  0.37912
## RollRate              -2.762e-02  4.177e-04 -66.114  < 2e-16 ***
## currRoll              -4.890e-03  1.900e-04 -25.741  < 2e-16 ***
## diffRollRate          -6.443e-01  1.598e-02 -40.320  < 2e-16 ***
## I(RollRate^2)          7.858e-03  6.861e-02   0.115  0.90882
## I(currRoll^2)          7.993e-03  1.033e-02   0.774  0.43887
## I(diffRollRate^2)     -1.780e+02  5.849e+01  -3.044  0.00234 **
## RollRate:currRoll      1.573e-01  3.861e-02   4.075 4.65e-05 ***
## RollRate:diffRollRate  1.773e+01  3.166e+00   5.602 2.20e-08 ***
## currRoll:diffRollRate -2.712e+00  1.465e+00  -1.851  0.06424 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001718 on 7119 degrees of freedom
## Multiple R-squared:  0.6787, Adjusted R-squared:  0.6783
## F-statistic:  1671 on 9 and 7119 DF,  p-value: < 2.2e-16
```

Acierto de 67.83%

```
#Construcción de modelo múltiple con interacciones
fit16 = lm(Sa~RollRate+currRoll+diffRollRate+currPitch+PitchRate+(.^2)+I(RollRate^2)+I(currRoll^2)
           +I(diffRollRate^2),data=deltaAil)
summary(fit16)
```

```
##
## Call:
## lm(formula = Sa ~ RollRate + currRoll + diffRollRate + currPitch +
##     PitchRate + (.^2) + I(RollRate^2) + I(currRoll^2) + I(diffRollRate^2),
##     data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
```

```
## -1.547e-03 -8.458e-05  7.960e-06  9.088e-05  1.060e-03
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)             -2.311e-05  5.698e-06  -4.056 5.06e-05 ***
## RollRate                -2.051e-02  9.718e-04 -21.110  < 2e-16 ***
## currRoll                -2.539e-03  4.136e-04  -6.138 8.82e-10 ***
## diffRollRate            -3.625e-01  3.626e-02  -9.999  < 2e-16 ***
## currPitch                2.051e-03  4.433e-04   4.628 3.77e-06 ***
## PitchRate               -8.183e-03  2.298e-03  -3.560 0.000373 ***
## I(RollRate^2)           -9.761e-04  6.895e-02  -0.014 0.988706
## I(currRoll^2)           -2.980e-03  1.038e-02  -0.287 0.774144
## I(diffRollRate^2)       -2.234e+02  6.216e+01  -3.595 0.000327 ***
## RollRate:PitchRate      -1.570e-01  2.033e-01  -0.772 0.439885
## RollRate:currPitch      -5.989e-01  7.991e-02  -7.495 7.42e-14 ***
## RollRate:currRoll        1.469e-01  3.909e-02   3.759 0.000172 ***
## RollRate:diffRollRate    1.325e+01  3.276e+00   4.043 5.34e-05 ***
## currPitch:PitchRate      2.341e-01  1.725e-01   1.357 0.174755
## currRoll:PitchRate      -1.285e-01  9.311e-02  -1.380 0.167513
## diffRollRate:PitchRate   2.344e+00  7.669e+00   0.306 0.759854
## currRoll:currPitch      -2.100e-01  3.359e-02  -6.251 4.32e-10 ***
## diffRollRate:currPitch -2.722e+01  2.987e+00  -9.112  < 2e-16 ***
## currRoll:diffRollRate   -4.482e+00  1.516e+00  -2.956 0.003125 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0001674 on 7110 degrees of freedom
## Multiple R-squared:  0.6953, Adjusted R-squared:  0.6946
## F-statistic: 901.6 on 18 and 7110 DF,  p-value: < 2.2e-16
```

Acierto del 69.46%. Aunque hasta ahora presenta el mayor acierto, es bastante más complejo que el resto de modelos.

```
#Construcción de modelo múltiple con interacciones
fit17 = lm(Sa~RollRate+currRoll+diffRollRate+currPitch+PitchRate+(.^3)+I(RollRate^3)+I(currRoll^3)
           +I(diffRollRate^3)+I(PitchRate^3)+I(currPitch^3),data=deltaAil)
summary(fit17)
```

```
##
## Call:
## lm(formula = Sa ~ RollRate + currRoll + diffRollRate + currPitch +
##     PitchRate + (.^3) + I(RollRate^3) + I(currRoll^3) + I(diffRollRate^3) +
##     I(PitchRate^3) + I(currPitch^3), data = deltaAil)
##
## Residuals:
##        Min         1Q     Median         3Q        Max
## -1.517e-03 -8.294e-05  7.770e-06  9.065e-05  1.037e-03
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    -1.952e-05  7.507e-06  -2.600  0.00933
## RollRate                       -1.935e-02  1.218e-03 -15.884  < 2e-16
## currRoll                       -3.629e-03  5.020e-04  -7.229 5.37e-13
## diffRollRate                   -2.926e-01  4.202e-02  -6.962 3.67e-12
```

```
## currPitch                          1.480e-03  7.879e-04   1.879  0.06030
## PitchRate                         -5.997e-04  2.681e-03  -0.224  0.82301
## I(RollRate^3)                      1.678e+01  6.499e+00   2.582  0.00983
## I(currRoll^3)                      1.315e+00  2.269e-01   5.796 7.10e-09
## I(diffRollRate^3)                  1.920e+04  5.192e+04   0.370  0.71147
## I(PitchRate^3)                    -7.089e+01  5.643e+01  -1.256  0.20907
## I(currPitch^3)                     1.681e+00  9.423e-01   1.784  0.07454
## RollRate:PitchRate                -1.338e+00  4.384e-01  -3.053  0.00227
## RollRate:currPitch                -6.883e-01  9.343e-02  -7.367 1.94e-13
## RollRate:currRoll                 -1.175e-02  7.058e-02  -0.166  0.86779
## RollRate:diffRollRate              8.149e-01  4.874e+00   0.167  0.86723
## currPitch:PitchRate               -1.796e-01  1.834e-01  -0.979  0.32745
## currRoll:PitchRate                -2.905e-01  1.851e-01  -1.569  0.11666
## diffRollRate:PitchRate            -4.474e+01  1.447e+01  -3.092  0.00199
## currRoll:currPitch                -2.184e-01  3.925e-02  -5.565 2.72e-08
## diffRollRate:currPitch            -3.544e+01  3.513e+00 -10.088  < 2e-16
## currRoll:diffRollRate             -1.112e+00  1.247e+00  -0.892  0.37245
## RollRate:currPitch:PitchRate       9.513e+01  3.409e+01   2.791  0.00527
## RollRate:currRoll:PitchRate        2.355e+01  1.439e+01   1.636  0.10180
## RollRate:diffRollRate:PitchRate    1.668e+02  9.581e+02   0.174  0.86182
## RollRate:currRoll:currPitch        6.660e+00  5.708e+00   1.167  0.24331
## RollRate:diffRollRate:currPitch    3.382e+02  3.917e+02   0.863  0.38790
## RollRate:currRoll:diffRollRate     5.095e+02  9.192e+01   5.543 3.08e-08
## currRoll:currPitch:PitchRate       1.381e+01  1.405e+01   0.983  0.32569
## diffRollRate:currPitch:PitchRate   4.824e+03  1.089e+03   4.429 9.59e-06
## currRoll:diffRollRate:PitchRate    1.740e+02  2.820e+02   0.617  0.53726
## currRoll:diffRollRate:currPitch    2.396e+01  1.099e+02   0.218  0.82747
##
## (Intercept)                       **
## RollRate                          ***
## currRoll                          ***
## diffRollRate                      ***
## currPitch                         .
## PitchRate
## I(RollRate^3)                     **
## I(currRoll^3)                     ***
## I(diffRollRate^3)
## I(PitchRate^3)
## I(currPitch^3)                    .
## RollRate:PitchRate                **
## RollRate:currPitch                ***
## RollRate:currRoll
## RollRate:diffRollRate
## currPitch:PitchRate
## currRoll:PitchRate
## diffRollRate:PitchRate            **
## currRoll:currPitch                ***
## diffRollRate:currPitch            ***
## currRoll:diffRollRate
## RollRate:currPitch:PitchRate      **
## RollRate:currRoll:PitchRate
## RollRate:diffRollRate:PitchRate
## RollRate:currRoll:currPitch
## RollRate:diffRollRate:currPitch
```

```
## RollRate:currRoll:diffRollRate   ***
## currRoll:currPitch:PitchRate
## diffRollRate:currPitch:PitchRate ***
## currRoll:diffRollRate:PitchRate
## currRoll:diffRollRate:currPitch
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.000166 on 7098 degrees of freedom
## Multiple R-squared:  0.7008, Adjusted R-squared:  0.6995
## F-statistic: 554.1 on 30 and 7098 DF,  p-value: < 2.2e-16
```

Un acierto del 69.95%. Un modelo mucho más complejo que los demás que, sin embargo, no obtiene una mejora considerable.

Como conclusión final de este apartado, el modelo finalmente elegido sería el construido únicamente con las variables "RollRate", "currRoll" y "diffRollRate" sin ningún tipo de interacción. Este modelo, que obtuvo un acierto del 67.62%, es mucho más sencillo que este último y, sin embargo, su acierto es únicamente un 2% inferior por lo que se convierte en el modelo más interesante del estudio realizado.

Knn

En este apartado se seguirá trabajando sobre el problema de regresión utilizando un modelo Knn, "vecino más cercano".

Para las construcción de los diferentes modelos de Knn se pueden utilizar las variables que dieron mejores resultado en el apartado anterior. No obstante, la mejor selección de predictores para un modelo de regresión lineal no implica que también lo sea para Knn por lo que se deberán probar otras posibles variables.

```
#Construcción de modelo utlizando kNN y obtención de RMSE
require(kknn)
```

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

```
fitknn1 <- kknn(Sa ~ .-currPitch-PitchRate, deltaAil, deltaAil)
yprime = fitknn1$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))
```

```
## [1] 0.0001223048
```

Aquí se utiliza el que fue el mejor modelo en regresión y se consigue bajar el RMSE utilizando Knn, bajando del 0.000172 que se consiguió a 0.000122, empleando las mismas variables.

```
#Construcción de modelo utlizando kNN y obtención de RMSE
fitknn2 <- kknn(Sa ~ ., deltaAil, deltaAil)
yprime = fitknn2$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))
```

```
## [1] 0.0001176461
```

Utilizando todas las variables se baja a 0.000117.

```
#Construcción de modelo utlizando kNN y obtención de RMSE
fitknn3 <- kknn(Sa ~ RollRate, deltaAil, deltaAil)
```

```
yprime = fitknn3$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))
```

```
## [1] 0.0001948336
```

Con el modelo más simple posible, utilizando únicamente la variable "RollRate" se obtiene un RMSE de 0.00019.

Probamos ahora con modelos más complejos utilizando interacciones entre las variables. Recordar que en el apartado anterior, este tipo de modelos apenas conseguían mejoras a cambio de una complejidad bastante mayor.

```
#Construcción de modelo utlizando kNN y obtención de RMSE
fitknn4 <- kknn(Sa ~ RollRate+diffRollRate+currRoll+RollRate*diffRollRate, deltaAil, deltaAil)
yprime = fitknn4$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))
```

```
## [1] 0.0001165896
```

Añaniendo la primera interacción la mejora es mínima, de 0.000117 a 0.000116.

```
#Construcción de modelo utlizando kNN y obtención de RMSE
fitknn5 <- kknn(Sa ~ RollRate+diffRollRate+currRoll+RollRate*diffRollRate+I(RollRate^2)
                +I(diffRollRate^2), deltaAil, deltaAil)
yprime = fitknn5$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))
```

```
## [1] 0.0001143888
```

Nuevamente con un modelo más complejo la mejora es muy leve.

```
#Construcción de modelo utlizando kNN y obtención de RMSE
fitknn6 <- kknn(Sa ~ RollRate+diffRollRate+currRoll+currPitch+PitchRate+(.^2),
                deltaAil, deltaAil)
yprime = fitknn6$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))
```

```
## [1] 0.0001050256
```

Se consigue bajar ahora hasta 0.000105.

```
#Construcción de modelo utlizando kNN y obtención de RMSE
fitknn7 <- kknn(Sa ~ RollRate+diffRollRate+currRoll+currPitch+PitchRate+(.^3), deltaAil,
                deltaAil)
yprime = fitknn7$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))
```

```
## [1] 0.0001027079
```

Nuevamente, un modelo mucho más complejo con una mejora mínima.

Con estos resultados se podría diferenciar 2 modelos interesantes. El segundo modelo construido con todas las variables pero sin ninguna interacción con un RMSE de 0.000117, un modelo bastante simple. El sexto modelo, con interacciones entre variables de todas contra todas con un RMSE de 0.000105 a cambio de una complejidad mayor.

Dado que en esta ocasión no es tan complejo este último modelo, se decide tomarlo finalmente como modelo final.

Podemos ahora utilizar este mismo modelo con los datos normalizados aunque previsiblemente no se conseguirán mejorar los resultados:

```
#Construcción de modelo utlizando kNN con datos normalizados y obtención de RMSE
fitknnNorm <- kknn(Sa ~ RollRate+diffRollRate+currRoll+currPitch+PitchRate+(.^2),
                   deltaAilNormalizado, deltaAilNormalizado)
yprime = fitknnNorm$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))
```

```
## [1] 0.0001062433
```

Efectivamente, el error es muy similar, incluso algo mayor, al utilizar datos normalizados.

Comparación

En este último apartado se procede a la comparación entre los algoritmos de regresión utilizando los resultados obtenidos para este 'dataset'.

Antes de empezar, debemos obtener el MSE para ambos algoritmos utilizando validación cruzada con el fin de evitar un sobreajuste por parte del modelo.

```
#Cálculo de MSE para el algoritmo "lm" utilizando validación cruzada
setwd("./delta_ail")

nombre <- "delta_ail"
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~.-X2-X3,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"))
```

```
#Cálculo de MSE para el algoritmo "knn" utilizando validación cruzada
setwd("./delta_ail")

nombre <- "delta_ail"
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~.+(.^2),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"))
```

Estos son los resultados obtenidos:

```
#Resultados de train y test para los algoritmos lm y kNN
lmMSEtrain
```

```
## [1] 2.968163e-08
```

```
lmMSEtest
```

```
## [1] 2.971013e-08
```

```
knnMSEtrain
```

```
## [1] 1.118305e-08
```

```
knnMSEtest
```

```
## [1] 3.055722e-08
```

Gracias al haber realizado validación cruzada tenemos una conclusión interesante sobre ambos modelos. Cuando se construyó el modelo final para “Knn” se obtuvo un mejor resultado que con “lm” a cambio de un modelo más complejo. Sin embargo, descubrimos ahora que el modelo para “Knn” hizo sobreaprendizaje pues el error para los datos de test es superior al de train, haciendo incluso que ahora el modelo de “lm” sea mejor, tiene un error menor con mucha menos complejidad.

Se actualizan los ficheros ".csv" correspondientes con estos nuevos resultados y se comparan ya en sí los algoritmos:

```
#Preparación tablas para tests de comparación
setwd("./delta_ail")

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]

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]
```

```
#Test de Wilcoxon
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,]  69.6882353     0.1000000
## [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
```

Y aquí tenemos los valores finales:

```
#Resultados de test de Wilcoxon
Rmas
```

```
##  V
## 92
```

```
Rmenos
```

```
##  V
## 79
```

```
pvalue
```

```
## [1] 0.7987061
```

```
#Confianza
(1-pvalue)*100
```

```
## [1] 20.12939
```

Con estos resultados del Test de Wilcoxon podemos concluir que no existen diferencias relevantes entre los algoritmos de "Knn" y "lm" pues sólo se tiene un 20.12% de confianza de que sean distintos.

Se puede realizar el mismo experimento para los datos de entrenamiento aunque, por lo que ha sucedido, podemos interpretar que los valores que se obtengan pueden no ser realmente fiables:

```
#Test de Wilcoxon para resultados de entrenamiento
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
```

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

Rmas
```

```
## V
## 9
```

```
Rmenos
```

```
##   V
## 162
```

```
pvalue
```

```
## [1] 0.00025177
```

```
#Confianza
(1-pvalue)*100
```

```
## [1] 99.97482
```

Con los datos de "train" el Test nos indica totalmente lo opuesto, que está confiado al 99.97% de que los algoritmos son distintos. Sin embargo, dado que a nosotros nos interesa los resultados de test, pues hemos descubierto que puede existir sobreaprendizaje en los modelos, concluimos que ambos algoritmos son similares con un 80% de confianza.

Por último se realizará una comparativa múltiple entre los algoritmos de "lm", "Knn" y "M5" utilizando el test de Friedman y el de Post-hoc Holm.

```
#Test de Friedman
test_friedman <- friedman.test(as.matrix(tablatst))
test_friedman
```

```
##
```

```
##  Friedman rank sum test
##
## data:  as.matrix(tablatst)
## Friedman chi-squared = 5.3333, df = 2, p-value = 0.06948
```

Con un p-value de tan solo 0.06, el Test de Friedman indica que hay diferencias significativas entre, al menos, dos de los algoritmos.

```
#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 0.97 -
## 3 0.27 0.27
##
## P value adjustment method: holm
```

Este último Test de Post-hoc Holm nos indica que no hay diferencias entre los algoritmos de "Knn" y "lm" pues se tiene tan solo un 3% de confianza de que sean distintos. Sin embargo, sí que hay diferencias a favor del algoritmos M5' pues, para ambos algoritmos, se obtiene una confianza del 73% de que son distintos.

Clasificación

Nuestro problema de clasificación trata con el conjunto de datos 'Heart' ya estudiado. En este problema se trata de poder construir y entrenar un modelo de clasificación a partir de este 'dataset' que, posteriormente, sea capaz de indicar para nuevas observaciones si se trata de una enfermedad del corazón o no. Se construirán diferentes modelos con diferentes algoritmos para poder a continuación comparar sus resultados y decidir qué modelo se comporta mejor.

Algoritmo Knn

El primero modelo se construirá utilizando el algoritmo de "vecino más cercano". Dado que su comportamiento varía en función del parámetro "k" utilizado, se probarán diferentes para encontrar así el mejor modelo posible.

Empezamos dividiendo el 'dataset' en datos de entrenamiento y de test en una relación de 80-20:

```
#División del dataset en entrenamiento y test
set.seed(10)
shuffled <- sample(dim(heart)[1])
eightypct <- (dim(heart)[1] * 80) %/% 100
heart_train <- heart[shuffled[1:eightypct], 1:13]
heart_test <- heart[shuffled[(eightypct+1):dim(heart)[1]], 1:13]

heart_train_labels <- heart[shuffled[1:eightypct], 14]
heart_test_labels <- heart[shuffled[(eightypct+1):dim(heart)[1]], 14]
```

Se realiza la misma división para los datos normalizados:

```
#División de los datos normalizados en entrenamiento y test
set.seed(10)
shuffledN <- sample(dim(heartNormalizado)[1])
heartN_train <- heartNormalizado[shuffledN[1:eightypct], 1:13]
heartN_test <- heartNormalizado[shuffledN[(eightypct+1):dim(heartNormalizado)[1]], 1:13]

heartN_train_labels <- heartNormalizado[shuffledN[1:eightypct], 14]
heartN_test_labels <- heartNormalizado[shuffledN[(eightypct+1):dim(heartNormalizado)[1]], 14]
```

Ya sí procedemos a construir los diferentes modelos empleando diferentes "k":

```
#Construcción de modelos utilizando kNN con diferentes "k"
library(class)
require(caret)
```

```
## Loading required package: caret
```

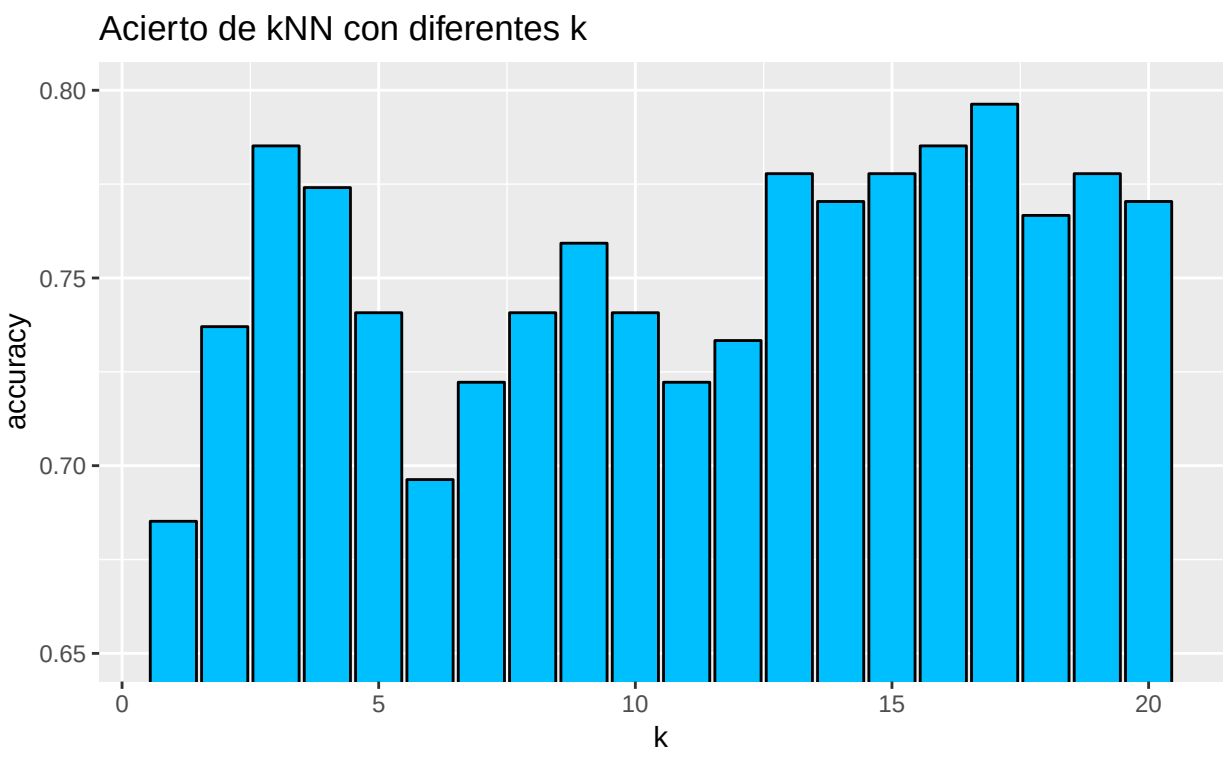
```
## Loading required package: lattice
```

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

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

```r
set.seed(10)
getKnn <- function(miK){

  heart_test_pred <- knn(train = heart_train, test = heart_test, cl = heart_train_labels, k=miK)

  postResample(pred = heart_test_pred, obs = heart[shuffled[(eightypct+1):dim(heart)[1]], 14])
}
result1 <- lapply(1:20,getKnn)
result1 <- unlist(result1)[1:20*2-1]
#Dado que existe aleatoriedad en este proceso se realiza 5 veces y se trata con la media
result2 <- lapply(1:20,getKnn)
result2 <- unlist(result2)[1:20*2-1]
result3 <- lapply(1:20,getKnn)
result3 <- unlist(result3)[1:20*2-1]
result4 <- lapply(1:20,getKnn)
result4 <- unlist(result4)[1:20*2-1]
result5 <- lapply(1:20,getKnn)
result5 <- unlist(result5)[1:20*2-1]

result<-unlist(Map("+",unlist(Map("+",unlist(Map("+",unlist(Map("+",result1,result2)),
                                                 result3)),result4)),result5))

result<-result/5
```

```r
#Gráfica de resultados obtenidos con diferentes "k"
df <- data.frame(k=1:20,accuracy=result)
ggplot(df, aes(x=k,y=accuracy)) + geom_histogram(stat="identity",color="black",fill="deepskyblue"
              )+coord_cartesian(ylim=c(0.65,0.8))+ labs(title="Acierto de kNN con diferentes k")
```



```r
#Mejor resultado obtenido
max(result)
```

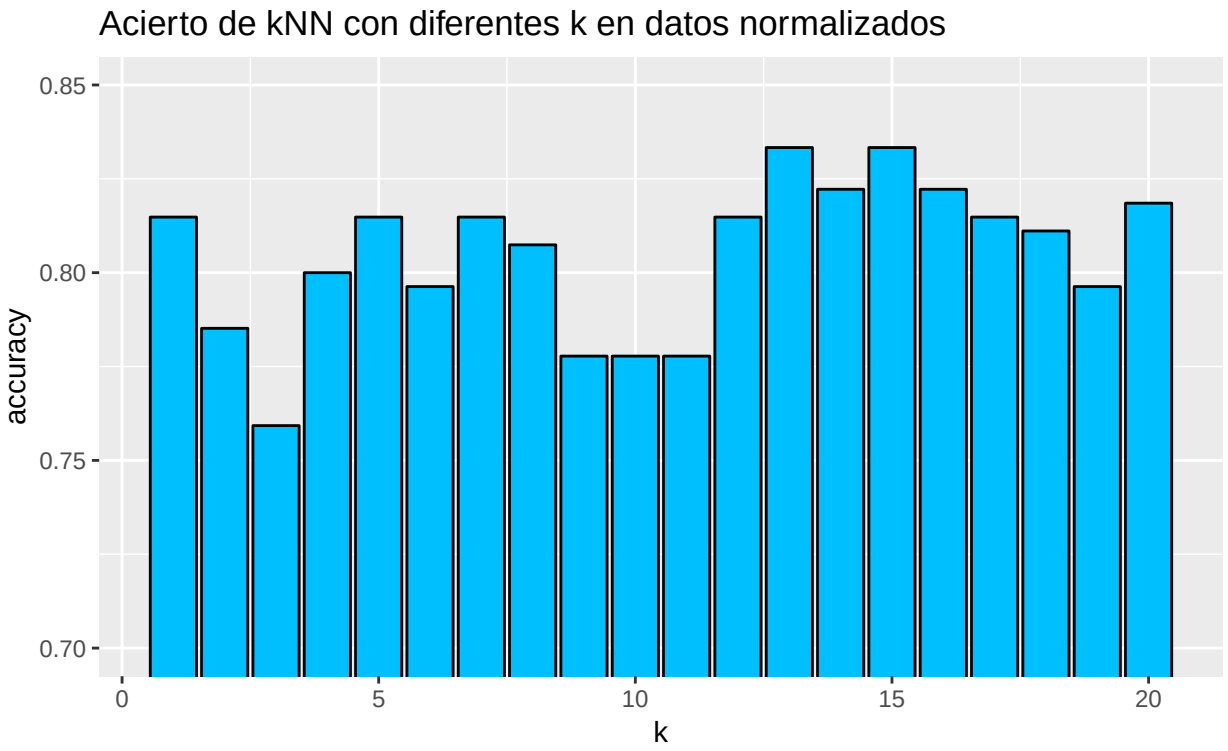
```
## [1] 0.7962963
```

Dado que existe aletoriedad en el proceso de obtención de resultados, se ha repetido el proceso 5 veces para, a continuación, hacer la media y tener un cálculo más aproximado. Gracias al gráfico obtenido sabemos que el acierto mayor ha sido de 79.62% que se ha alcanzado con k=17 por lo que éste sería el modelo que se elegiría.

A continuación se realiza el mismo experimento con los datos normalizados para descubrir posibles cambios en el comportamiento del modelo:

```r
#Construcción de modelos con kNN con datos normalizados y diferentes "k"
set.seed(10)
getKnn <- function(miK){

  heart_test_pred <- knn(train = heartN_train, test = heartN_test, cl = heartN_train_labels, k=miK)

  postResample(pred = heart_test_pred, obs =
                 heartNormalizado[shuffledN[(eightypct+1):dim(heartNormalizado)[1]], 14])
}

result1 <- lapply(1:20,getKnn)
result1 <- unlist(result1)[1:20*2-1]
#Dado que existe aleatoriedad en este proceso se realiza 5 veces y se trata con la media
result2 <- lapply(1:20,getKnn)
result2 <- unlist(result2)[1:20*2-1]
result3 <- lapply(1:20,getKnn)
result3 <- unlist(result3)[1:20*2-1]
result4 <- lapply(1:20,getKnn)
result4 <- unlist(result4)[1:20*2-1]
result5 <- lapply(1:20,getKnn)
result5 <- unlist(result5)[1:20*2-1]

result<-unlist(Map("+",unlist(Map("+",unlist(Map("+",unlist(Map("+",result1,result2)),
                                                 result3)),result4)),result5))

result<-result/5
```

```r
#Gráfica con resultados de kNN y sus diferentes "k"
df <- data.frame(k=1:20,accuracy=result)

ggplot(df, aes(x=k,y=accuracy)) + geom_histogram(stat="identity",color="black",fill="deepskyblue"
  )+coord_cartesian(ylim=c(0.7,0.85))+ labs(title="Acierto de kNN con diferentes k en datos normalizados
```




```
#Mejor resultado obtenido
max(result)
```

```
## [1] 0.8333333
```

Gracias a que se normalizasen los datos continuos del 'dataset', el modelo se ha comportado mejor con estos datos y se ha conseguido una mejora hasta alcanzar el 83.3% de acierto. En esta ocasión, los mejores resultados han sido con k=13 y k=15 que, dado que la complejidad es prácticamente la misma, el modelo final sería con cualquierda de los dos "k" si se decidiese utilizar los datos normalizados.

Algoritmo LDA

A continuación vamos a estudiar otros modelos empleando el algoritmo LDA.

Lo primero será dividir el conjunto de datos en datos de entrenamiento y de test. Al igual que se hizo en el algoritmo anterior, la relación será de 80%-20%.

```
#División de los datos en entrenamiento y test
set.seed(10)
shuffled <- sample(dim(heart)[1])
eightypct <- (dim(heart)[1] * 80) %/% 100
heart_train <- heart[shuffled[1:eightypct], ]
heart_test <- heart[shuffled[(eightypct+1):dim(heart)[1]], ]
```

Ahora sí construimos el modelo:

```
#Construcción del modelo con LDA
library(MASS)
```

```
library(ISLR)
ldaFit <- lda(Class ~.,data=heart_train)
ldaFit
```

```
## Call:
## lda(Class ~ ., data = heart_train)
##
## Prior probabilities of groups:
##       Si       No
## 0.537037 0.462963
##
## Group means:
##         Age       Sex ChestPainType RestBloodPressure SerumCholestoral
## Si 53.02586 0.5775862      2.827586          128.9655         242.1293
## No 56.83000 0.8200000      3.580000          135.2400         256.4700
##    FastingBloodSugar ResElectrocardiographic MaxHeartRate ExerciseInduced
## Si         0.1551724               0.8189655      158.069       0.1637931
## No         0.1400000               1.1700000      139.900       0.5300000
##      Oldpeak    Slope MajorVessels     Thal
## Si  6.077586 1.413793    0.2672414 3.818966
## No 12.690000 1.820000    1.1600000 5.890000
##
## Coefficients of linear discriminants:
##                                   LD1
## Age                       -0.008251986
## Sex                        0.564233438
## ChestPainType              0.369942511
## RestBloodPressure          0.009439521
## SerumCholestoral           0.002651639
## FastingBloodSugar         -0.343693957
## ResElectrocardiographic    0.176880775
## MaxHeartRate              -0.009484944
## ExerciseInduced            0.405270159
## Oldpeak                    0.012040942
## Slope                      0.283556664
## MajorVessels               0.595519017
## Thal                       0.258410316
```

Con el modelo ya entrenado se realiza la predicción utilizando los datos de test:

```
#Resultados del modelo construido con LDA
ldaPred <- predict(ldaFit,heart_test)

table(ldaPred$class,heart_test$Class)
```

```
##
##      Si No
##   Si 31  3
##   No  3 17
```

```
resultLDA <- mean(ldaPred$class==heart_test$Class)
resultLDA
```

```
## [1] 0.8888889
```

Se obtiene un acierto del 88.9%, una notable mejora respecto al algoritmo kNN.

Dado que se consiguió una mejora interesante al emplear datos normalizados, igualmente se realiza ahora el mismo experimento con estos datos pues este tipo de algoritmos asumen que los datos siguen unas distribución normal:

```
#Construcción y resultados de un modelo LDA sobre datos normalizados
set.seed(10)
shuffledN <- sample(dim(heartNormalizado)[1])
heartN_train <- heartNormalizado[shuffledN[1:eightypct], ]
heartN_test <- heartNormalizado[shuffledN[(eightypct+1):dim(heartNormalizado)[1]], ]

ldaFitN <- lda(Class ~.,data=heartN_train)
ldaFitN
```

```
## Call:
## lda(Class ~ ., data = heartN_train)
##
## Prior probabilities of groups:
##       Si       No
## 0.537037 0.462963
##
## Group means:
##         Age       Sex ChestPainType RestBloodPressure SerumCholestoral
## Si 53.02586 0.5775862      2.827586        -0.1331866       -0.1456858
## No 56.83000 0.8200000      3.580000         0.2180966        0.1317709
##    FastingBloodSugar ResElectrocardiographic MaxHeartRate ExerciseInduced
## Si         0.1551724               0.8189655    0.3622244       0.1637931
## No         0.1400000               1.1700000   -0.4220797       0.5300000
##       Oldpeak    Slope MajorVessels     Thal
## Si -0.2564913 1.413793    0.2672414 3.818966
## No  0.3444222 1.820000    1.1600000 5.890000
##
## Coefficients of linear discriminants:
##                                     LD1
## Age                       -0.008251986
## Sex                        0.564233438
## ChestPainType              0.369942511
## RestBloodPressure          0.168605018
## SerumCholestoral           0.137053242
## FastingBloodSugar         -0.343693957
## ResElectrocardiographic    0.176880775
## MaxHeartRate              -0.219725525
## ExerciseInduced            0.405270159
## Oldpeak                    0.132497756
## Slope                      0.283556664
## MajorVessels               0.595519017
## Thal                       0.258410316
```

```
ldaPredN <- predict(ldaFitN,heartN_test)
table(ldaPredN$class,heartN_test$Class)
```

```
##
##      Si No
##   Si 31  3
```

```
##   No  3 17
```

```
resultLDAN <- mean(ldaPredN$class==heartN_test$Class)
resultLDAN
```

```
## [1] 0.8888889
```

Se obtiene exactamente el mismo acierto, 88.9%. Por lo tanto descubrimos que no ha variado el comportamiento del algoritmo para este conjunto de datos ya estén normalizados o no.

Algoritmo QDA

De igual forma se realiza este experimento con el algoritmo QDA:

Entrenamos el modelo:

```
#Construcción del modelo con QDA
qdaFit <- qda(Class ~.,data=heart_train)
qdaFit
```

```
## Call:
## qda(Class ~ ., data = heart_train)
##
## Prior probabilities of groups:
##       Si       No
## 0.537037 0.462963
##
## Group means:
##         Age       Sex ChestPainType RestBloodPressure SerumCholestoral
## Si 53.02586 0.5775862      2.827586          128.9655         242.1293
## No 56.83000 0.8200000      3.580000          135.2400         256.4700
##    FastingBloodSugar ResElectrocardiographic MaxHeartRate ExerciseInduced
## Si         0.1551724               0.8189655      158.069       0.1637931
## No         0.1400000               1.1700000      139.900       0.5300000
##      Oldpeak    Slope MajorVessels     Thal
## Si  6.077586 1.413793    0.2672414 3.818966
## No 12.690000 1.820000    1.1600000 5.890000
```

```
#Resultados del modelo construido con QDA
qdaPred <- predict(qdaFit,heart_test)

table(qdaPred$class,heart_test$Class)
```

```
##
##      Si No
##   Si 31  3
##   No  3 17
```

```
resultQDA <- mean(qdaPred$class==heart_test$Class)
resultQDA
```

```
## [1] 0.8888889
```

El resultado obtenido para QDA ha sido el mismo que para LDA, 88.9% de acierto. Aunque son algoritmos diferentes, al fin y al cabo, QDA es una versión más genérica de LDA por lo que existe la posibilidad de que para un conjunto de datos concreto se obtenga el mismo resultado.

Podemos realizar el mismo experimento con los datos normalizados aunque previsiblemente se obtendrá el mismo resultado:

```
#Construcción y resultados del modelo QDA sobre datos normalizados
qdaFitN <- qda(Class ~.,data=heartN_train)
qdaFitN
```

```
## Call:
## qda(Class ~ ., data = heartN_train)
##
## Prior probabilities of groups:
##       Si       No
## 0.537037 0.462963
##
## Group means:
##         Age       Sex ChestPainType RestBloodPressure SerumCholestoral
## Si 53.02586 0.5775862      2.827586        -0.1331866       -0.1456858
## No 56.83000 0.8200000      3.580000         0.2180966        0.1317709
##    FastingBloodSugar ResElectrocardiographic MaxHeartRate ExerciseInduced
## Si         0.1551724               0.8189655    0.3622244       0.1637931
## No         0.1400000               1.1700000   -0.4220797       0.5300000
##       Oldpeak    Slope MajorVessels     Thal
## Si -0.2564913 1.413793    0.2672414 3.818966
## No  0.3444222 1.820000    1.1600000 5.890000
```

```
qdaPredN <- predict(qdaFitN,heartN_test)
table(qdaPredN$class,heartN_test$Class)
```

```
##
##      Si No
##   Si 31  3
##   No  3 17
```

```
resultQDAN <- mean(qdaPredN$class==heartN_test$Class)
resultQDAN
```

```
## [1] 0.8888889
```

Efectivamente se obtiene el mismo resultado.

Comparativa entre algorimos

En esta última sección se hara una comparativa entre los 3 algoritmos vistos anteriormente.

Lo primero será utilizar validación cruzada para tener unos resultados más veraces de los algoritmos:

Validación cruzada kNN

Se utilizarán los datos normalizados ya que propiciaron un mejor comportamiento para el algoritmo kNN:

```r
#Obtención de resultados de kNN utilizando validación cruzada y diferentes "k"
setwd("./heart")

nombre <- "heart"
run_knn_fold <- function(i, x, tt = "test",miK) {
  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="@")
  if (tt == "train") {
    test <- x_tra
  }
  else {
    test <- x_tst
  }


  #Normalizacion
  heartN_train <- data.frame(x_tra)
  names(heartN_train) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
  "FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced",
  "Oldpeak", "Slope", "MajorVessels", "Thal", "Class")

  heartN_train$RestBloodPressure <- scale(heartN_train$RestBloodPressure)
  heartN_train$SerumCholestoral <- scale(heartN_train$SerumCholestoral)
  heartN_train$MaxHeartRate <- scale(heartN_train$MaxHeartRate)
  heartN_train$Oldpeak <- scale(heartN_train$Oldpeak)


  heartN_test <- data.frame(test)
  names(heartN_test) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
  "FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced", "Oldpeak",
  "Slope", "MajorVessels", "Thal", "Class")

  heartN_test$RestBloodPressure <- scale(heartN_test$RestBloodPressure)
  heartN_test$SerumCholestoral <- scale(heartN_test$SerumCholestoral)
  heartN_test$MaxHeartRate <- scale(heartN_test$MaxHeartRate)
  heartN_test$Oldpeak <- scale(heartN_test$Oldpeak)

  #Preparacion datos
  heartN_train$Class <- factor(heartN_train$Class, levels = c(1, 2), labels = c("Si", "No"))
  heartN_test$Class <- factor(heartN_test$Class, levels = c(1, 2), labels = c("Si", "No"))

  heartN_train_labels <- heartN_train$Class
  heartN_test_labels <- heartN_test$Class

  heartN_train <- heartN_train[,1:13]
  heartN_test <- heartN_test[,1:13]

  #Construcción modelo kNN y resultado
  heart_test_pred <- knn(train = heartN_train, test = heartN_test, cl = heartN_train_labels, k=miK)
  postResample(pred = heart_test_pred, obs = heartN_test_labels)
}
```

```r
#Función utilizada para construir modelos con diferentes "k"
mejorK <- function(miK){

  knnAcctrain<-mean(sapply(1:10,run_knn_fold,nombre,"train",miK))
  knnAcctest<-mean(sapply(1:10,run_knn_fold,nombre,"test",miK))
  list(knnAcctrain,knnAcctest)
}

resultados <- lapply(1:20,mejorK)
```

Aunque ya se hizo el estudio de con qué "k" se comportaba mejor el modelo, dado que ahora se está tratando con otra distribución de los datos, es posible que cambie la mejor "k" para el algoritmo. Por ello, se realiza la validación cruzada con diferentes "k" para escoger finalmente el mejor resultado de los datos de "test":

```r
#Obtención de la mejor "k"
which.max(unlist(resultados)[1:20*2])
```

```
## [1] 7
```

Descubrimos que con k=7 se obtiene el mejor resultado.

```r
#Resultados obtenidos
knnAcctrain <- unlist(resultados[7])[1]
knnAcctest <- unlist(resultados[7])[2]
knnAcctrain
```

```
## [1] 0.7539717
```

```r
knnAcctest
```

```
## [1] 0.7009483
```

Finalmente tenemos para los datos de entrenamiento un acierto del 75.39% y para test un 70.09%. Por lo tanto ha habido algo de sobreaprendizaje por parte del modelo, pues el resultado para test es algo inferior al de "train".

Validación cruzada LDA

Dado que descubrimos que el comportamiento del algoritmo con los datos normalizados o no era exactamente el mismo, no es necesario entonces que hagamos normalización para obtener estos nuevos resultados.

```r
#Resultado de modelo LDA utilizando validación cruzada
setwd("./heart")

nombre <- "heart"
run_lda_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="@")
  if (tt == "train") {
    test <- x_tra
  }
  else {
```

```
    test <- x_tst
  }


  #Preparación datos
  heartN_train <- data.frame(x_tra)
  names(heartN_train) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
  "FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced", "Oldpeak",
  "Slope", "MajorVessels", "Thal", "Class")


  heartN_test <- data.frame(test)
  names(heartN_test) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
  "FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced", "Oldpeak",
  "Slope", "MajorVessels", "Thal", "Class")


  heartN_train$Class <- factor(heartN_train$Class, levels = c(1, 2), labels = c("Si", "No"))
  heartN_test$Class <- factor(heartN_test$Class, levels = c(1, 2), labels = c("Si", "No"))

  #Construcción modelo y resultado
  ldaFit <- lda(Class ~.,data=heartN_train)
  ldaPred <- predict(ldaFit,heartN_test)

  table(ldaPred$class,heartN_test$Class)
  resultLDA <- mean(ldaPred$class==heartN_test$Class)
  resultLDA

}

#Resultados obtenidos
ldaAcctrain<-mean(sapply(1:10,run_lda_fold,nombre,"train"))
ldaAcctest<-mean(sapply(1:10,run_lda_fold,nombre,"test"))
ldaAcctrain
```

```
## [1] 0.8578512
```

```
ldaAcctest
```

```
## [1] 0.8576923
```

Para el algoritmo LDA obtenemos un acierto de 85.78% para "train" y un acierto de 85.76% para test, un buen resultado que además nos muestra que no ha existido sobreaprendizaje pues es un valor muy similar al obtenido en el entrenamiento.

Validación cruzada QDA

```
#Resultados del modelo QDA utilizando validación cruzada
setwd("./heart")

nombre <- "heart"
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="@")
  if (tt == "train") {
    test <- x_tra
  }
  else {
    test <- x_tst
  }


  #Preparación datos
  heartN_train <- data.frame(x_tra)
  names(heartN_train) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
  "FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced", "Oldpeak",
  "Slope", "MajorVessels", "Thal", "Class")


  heartN_test <- data.frame(test)
  names(heartN_test) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
  "FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced", "Oldpeak",
  "Slope", "MajorVessels", "Thal", "Class")


  heartN_train$Class <- factor(heartN_train$Class, levels = c(1, 2), labels = c("Si", "No"))
  heartN_test$Class <- factor(heartN_test$Class, levels = c(1, 2), labels = c("Si", "No"))

  #Construcción de modelo
  qdaFit <- qda(Class ~.,data=heartN_train)
  qdaPred <- predict(qdaFit,heartN_test)

  table(qdaPred$class,heartN_test$Class)
  resultQDA <- mean(qdaPred$class==heartN_test$Class)
  resultQDA

}


#Resultados obtenidos
qdaAcctrain<-mean(sapply(1:10,run_qda_fold,nombre,"train"))
qdaAcctest<-mean(sapply(1:10,run_qda_fold,nombre,"test"))
qdaAcctrain
```

```
## [1] 0.8785124
```

```
qdaAcctest
```

```
## [1] 0.8346154
```

Para el algoritmo QDA se obtiene 87.85% de acierto para los datos de entrenamiento y 83.46% para el test. Si bien siguen siendo unos buenos resultados, en esta ocasión sí que ha existido algo de sobreentrenamiento pues el resultado para test es algo inferior.

Con estos resultados finales podemos decir que el algoritmo con mejores resultados para este 'dataset' ha sido LDA con un 85.76% de acierto. Está seguido de QDA con 83.46% y ya notablemente inferior ha sido kNN con 70.09% de acierto.

Por último quedaría comprobar cómo de diferentes son estos algoritmos entre ellos empleando los resultados de los algoritmos de otros 'datasets' y los resultados que acabamos de obtener:

Test de Wilcoxon

Empezamos realizando el test de Wilcoxon realizando comparaciones 1 vs 1 para saber cómo de diferentes son los algoritmos entre ellos. Estas comparaciones sólo se harán con los datos de test pues dado que hemos visto que puedo haber sobreaprendizaje en el entrenamiento, el resultado de los tests podría llevarnos a confusión.

```
#Preparación tablas para comparación de algoritmos
setwd("./heart")

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]
```

```
#Test de Wilcoxon
#kNN vs LDA
difs <- (tablatst[,1] - tablatst[,2]) / tablatst[,1]
wilc_1_2 <- cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1), ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) <- c(colnames(tablatst)[1], colnames(tablatst)[2])
head(wilc_1_2)
```

```
##      out_test_knn out_test_lda
## [1,]    0.1000000    0.1307536
## [2,]    0.3546673    0.1000000
## [3,]    0.1000000    0.1443729
## [4,]    0.1000000    0.1040566
## [5,]    0.1000000    0.1655377
## [6,]    0.1026335    0.1000000
```

```
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
## 123
```

```
Rmenos
```

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

```
pvalue
```

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

```r
#Confianza
(1-pvalue)*100
```

```
## [1] 47.83268
```

El test de Wilcoxon nos indica que hay un 47.83% de confianza de que sean distintos kNN y LDA.

```r
#Test de Wilcoxon
#kNN vs QDA
difs <- (tablatst[,1] - tablatst[,3]) / tablatst[,1]
wilc_1_3 <- cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1), ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_3) <- c(colnames(tablatst)[1], colnames(tablatst)[3])
head(wilc_1_3)
```

```
##      out_test_knn out_test_qda
## [1,]    0.1000000    0.1956404
## [2,]    0.2741312    0.1000000
## [3,]    0.1158854    0.1000000
## [4,]    0.1000000    0.2273004
## [5,]    0.1000000    0.1505725
## [6,]    0.1068124    0.1000000
```

```r
LMvsKNNtst <- wilcox.test(wilc_1_3[,1], wilc_1_3[,2], alternative = "two.sided", paired=TRUE)
Rmas <- LMvsKNNtst$statistic
pvalue <- LMvsKNNtst$p.value
LMvsKNNtst <- wilcox.test(wilc_1_3[,2], wilc_1_3[,1], alternative = "two.sided", paired=TRUE)
Rmenos <- LMvsKNNtst$statistic
Rmas
```

```
##   V
## 142
```

```r
Rmenos
```

```
##  V
## 68
```

```r
pvalue
```

```
## [1] 0.1768532
```

```r
#Confianza
(1-pvalue)*100
```

```
## [1] 82.31468
```

Se muestra mucha seguridad de que los algoritmos de kNN y QDA son distintos, con un 82.31% de confianza sobre ellos.

```r
#Test de Wilcoxon
#LDA vs QDA
difs <- (tablatst[,2] - tablatst[,3]) / tablatst[,2]
wilc_2_3 <- cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1), ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_2_3) <- c(colnames(tablatst)[2], colnames(tablatst)[3])
head(wilc_2_3)
```

```
##      out_test_lda out_test_qda
```

```
## [1,]    0.1000000    0.1669456
## [2,]    0.1000000    0.1641892
## [3,]    0.1630563    0.1000000
## [4,]    0.1000000    0.2237458
## [5,]    0.1160149    0.1000000
## [6,]    0.1041679    0.1000000
```

```
LMvsKNNtst <- wilcox.test(wilc_2_3[,1], wilc_2_3[,2], alternative = "two.sided", paired=TRUE)
Rmas <- LMvsKNNtst$statistic
pvalue <- LMvsKNNtst$p.value
LMvsKNNtst <- wilcox.test(wilc_2_3[,2], wilc_2_3[,1], alternative = "two.sided", paired=TRUE)
Rmenos <- LMvsKNNtst$statistic
Rmas
```

```
##   V
## 114
```

```
Rmenos
```

```
##  V
## 96
```

```
pvalue
```

```
## [1] 0.7561665
```

```
#Confianza
(1-pvalue)*100
```

```
## [1] 24.38335
```

En el caso de LDA vs QDA, el test de Wilcoxon no ha detectado diferencias relevantes entre ellos pues sólo nos muestra un 24.38% de confianza de que sean distintos.

Comparación múltiple

El test de Friedman nos indicará si hay diferencias entre algunos de los algoritmos

```
#Test de Friedman
test_friedman <- friedman.test(as.matrix(tablatst))
test_friedman
```

```
##
##  Friedman rank sum test
##
## data:  as.matrix(tablatst)
## Friedman chi-squared = 0.7, df = 2, p-value = 0.7047
```

Con un p-value de 0.7047, este test indica que no parece que haya algoritmos distintos entre ellos pues sólo tiene un 30% de confianza de que por lo menos 2 de los algoritmos sean distintos.

Por último se realiza el test de Post-hoc Holm:

```
#Test de post-hoc Holm
tam <- dim(tablatst)
```

```r
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 1.00 -
## 3 0.53 1.00
##
## P value adjustment method: holm
```

Este test nos indica un 100% confianza de que el algoritmo LDA es igual a kNN y a QDA. Si bien sorprende que muestre que kNN y LDA son iguales, puede ser que para otros 'datasets' se hayan obtenido resultados muy similares para ambos algoritmos. Por último, sí que muestra que hay diferencias entre kNN y QDA con 47% de confianza.

Apéndice

En este apéndice se muestra todo el código utilizado en este trabajo.

Análisis de datos

```
knitr::opts_chunk$set(echo = TRUE)

## ------------------------------------------------------------------------
#Carga de los 'datasets'
library(ggplot2)
heart <- read.csv("./heart/heart.dat",
          comment.char = "@", header = FALSE)

names(heart) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
"FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced", "Oldpeak",
"Slope", "MajorVessels", "Thal", "Class")

deltaAil <- read.csv("./delta_ail/delta_ail.dat",comment.char = "@", header = FALSE)

names(deltaAil) <- c("RollRate", "PitchRate", "currPitch", "currRoll", "diffRollRate", "Sa")
## ------------------------------------------------------------------------

########################################################
#Análisis dataset heart
########################################################

## ------------------------------------------------------------------------
#Dimensiones del dataset heart
dim(heart)

#Nombre de las variables
colnames(heart)

#Tipo atómico de cada variable
unlist(lapply(heart,class))

#Estructura del dataset
class(heart)
unique(heart$Class)

#Conversión de la variable de salida a un factor
heart$Class <- factor(heart$Class, levels = c(1, 2), labels = c("Si", "No"))
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Comprobación de missing values
which(is.na(heart) == TRUE)

#Comprobación de valores duplicados
which(duplicated(heart) == TRUE)
```

```
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Proporción de la variable de salida
round(prop.table(table(heart$Class)) * 100, digits = 1)

#Proporción de las variables discretas predictoras
round(prop.table(table(heart$Sex)) * 100, digits = 1)
round(prop.table(table(heart$ChestPainType)) * 100, digits = 1)
round(prop.table(table(heart$FastingBloodSugar)) * 100, digits = 1)
round(prop.table(table(heart$ResElectrocardiographic)) * 100, digits = 1)
round(prop.table(table(heart$ExerciseInduced)) * 100, digits = 1)
round(prop.table(table(heart$Slope)) * 100, digits = 1)
round(prop.table(table(heart$MajorVessels)) * 100, digits = 1)
round(prop.table(table(heart$Thal)) * 100, digits = 1)
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Distribución de la variable Age
ggplot(heart, aes(x=Age)) + geom_histogram(binwidth = 5,color="black",fill="lightblue")+labs(
  title="Distribucion de Age")

#Distribución de la variable RestBloodPressure
ggplot(heart, aes(x=RestBloodPressure))+geom_density(color="darkblue",fill="lightblue")+geom_vline(
  aes(xintercept=mean(RestBloodPressure)),color="green")+labs(title="Distribucion de RestBlood
  Pressure")+geom_text(aes(x=136,y=0.01),label="mean",color="green")+geom_vline(aes(xintercept
  =median(RestBloodPressure)),color="red")+geom_text(aes(x=124,y=0.01),label="median",color=
  "red")+geom_text(aes(x=180,y=0.02),label=paste("mean: ",toString(round(mean(
  heart$RestBloodPressure),2))),color="green") +geom_text(aes(x=179,y=0.018),label=paste("median: "
  ,toString(median(heart$RestBloodPressure))),color="red") + geom_text(aes(x=179,y=0.016),label=
  paste("std: ",toString(round(sd(heart$RestBloodPressure),2))),color="blue")

#Distribución de la variable SerumCholestoral
ggplot(heart, aes(x=SerumCholestoral))+geom_density(color="darkblue",fill="lightblue")+geom_vline(
  aes(xintercept=mean(SerumCholestoral)),color="green")+labs(title="Distribucion de SerumCholestor
  al")+geom_text(aes(x=272,y=0.004),label="mean",color="green")+geom_vline(aes(xintercept=median(
  SerumCholestoral)),color="red")+geom_text(aes(x=224,y=0.004),label="median",color="red"
  )+geom_text(aes(x=400,y=0.007),label=paste("mean: ",toString(round(mean(heart$SerumCholestoral),
  2))),color="green") +geom_text(aes(x=400,y=0.0065),label=paste("median: ",toString(median(
  heart$SerumCholestoral))),color="red") + geom_text(aes(x=400,y=0.006),label=paste("std: ",
  toString(round(sd(heart$SerumCholestoral),2))),color="blue")

#Distribución de la variable MaxHeartRate
ggplot(heart, aes(x=MaxHeartRate))+geom_density(color="darkblue",fill="lightblue")+geom_vline(
  aes(xintercept=mean(MaxHeartRate)),color="green")+labs(title="Distribucion de MaxHeartRate"
  )+geom_text(aes(x=140,y=0.004),label="mean",color="green")+geom_vline(aes(xintercept=median(
  MaxHeartRate)),color="red")+geom_text(aes(x=160,y=0.004),label="median",color="red"
  )+geom_text(aes(x=190,y=0.014),label=paste("mean: ",toString(round(mean(heart$MaxHeartRate),
  2))),color="green") +geom_text(aes(x=190,y=0.012),label=paste("median: ",toString(median(
  heart$MaxHeartRate))),color="red") + geom_text(aes(x=190,y=0.01),label=paste("std: ",
  toString(round(sd(heart$MaxHeartRate),2))),color="blue")
```

```r
#Distribución de la variable Oldpeak
ggplot(heart, aes(x=Oldpeak))+geom_density(color="darkblue",fill="lightblue")+geom_vline(
  aes(xintercept=mean(Oldpeak)),color="green")+labs(title="Distribucion de Oldpeak")+geom_text(
  aes(x=12,y=0.004),label="mean",color="green")+geom_vline(aes(xintercept=median(Oldpeak)),
  color="red")+geom_text(aes(x=2,y=0.004),label="median",color="red")+geom_text(aes(x=40,y=0.044),
  label=paste("mean: ",toString(round(mean(heart$Oldpeak),2))),color="green") +geom_text(aes(x=40,
  y=0.04),label=paste("median: ",toString(median(heart$Oldpeak))),color="red") + geom_text(aes(
  x=40,y=0.036),label=paste("std: ",toString(round(sd(heart$Oldpeak),2))),color="blue")
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Normalización del dataset
heartNormalizado <- data.frame(heart)
heartNormalizado$RestBloodPressure <- scale(heartNormalizado$RestBloodPressure)
heartNormalizado$SerumCholestoral <- scale(heartNormalizado$SerumCholestoral)
heartNormalizado$MaxHeartRate <- scale(heartNormalizado$MaxHeartRate)
heartNormalizado$Oldpeak <- scale(heartNormalizado$Oldpeak)

#Maxima correlacion entre las variables predictoras
max(abs(cor(heart[-14]))%%1)
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Dimensiones del dataset delta_Ail
dim(deltaAil)
## ------------------------------------------------------------------------



#########################################################
#Análisis dataset deltaAil
#########################################################

## ------------------------------------------------------------------------
#Nombre de las variables del dataset
colnames(deltaAil)

#Tipo atomico de cada variable
unlist(lapply(deltaAil,class))

#Estructura del dataset y de la variable de salida
class(deltaAil)
min(deltaAil$Sa)
max(deltaAil$Sa)

#Correlación respecto a la variable de salida
cor(deltaAil)[6,]
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
```

```r
#Comprobación de missing values
which(is.na(deltaAil) == TRUE)

#Comprobación de valores duplicados
which(duplicated(deltaAil) == TRUE)
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Distribución de la variable de salida Sa
ggplot(deltaAil, aes(x=Sa))+geom_density(color="darkblue",fill="lightblue")+labs(
  title="Distribucion de la variable de salida Sa")

#Distribución de la variable RollRate
ggplot(deltaAil, aes(x=RollRate))+geom_density(color="darkblue",fill="lightblue")+geom_vline(
  aes(xintercept=mean(RollRate)),color="green")+labs(title="Distribucion de RollRate")+geom_text(
  aes(x=-0.001,y=15),label="mean",color="green")+geom_vline(aes(xintercept=median(RollRate)),
  color="red")+geom_text(aes(x=0.004,y=15),label="median",color="red")+geom_text(aes(x=0.012,y=60),
  label=paste("mean: ",toString(round(mean(deltaAil$RollRate),5))),color="green") +geom_text(
  aes(x=0.012,y=55),label=paste("median: ",toString(median(deltaAil$RollRate))),color="red"
  )+geom_text(aes(x=0.012,y=50),label=paste("std: ",toString(round(sd(deltaAil$RollRate),5))),color
  ="blue")

#Distribución de la variable PitchRate
ggplot(deltaAil, aes(x=PitchRate))+geom_density(color="darkblue",fill="lightblue")+geom_vline(
  aes(xintercept=mean(PitchRate)),color="green")+labs(title="Distribucion de PitchRate")+geom_text(
  aes(x=0.0015,y=50),label="mean",color="green")+geom_vline(aes(xintercept=median(PitchRate)),color
  ="red")+geom_text(aes(x=0,y=50),label="median",color="red")+geom_text(aes(x=0.006,y=160),label=
  paste("mean: ",toString(round(mean(deltaAil$PitchRate),5))),color="green") +geom_text(aes(x=
  0.006,y=150),label=paste("median: ",toString(median(deltaAil$PitchRate))),color="red")+geom_text(
  aes(x=0.006,y=140),label=paste("std: ",toString(round(sd(deltaAil$PitchRate),5))),color="blue")

#Distribución de la variable currPitch
ggplot(deltaAil, aes(x=currPitch))+geom_density(color="darkblue",fill="lightblue")+geom_vline(
  aes(xintercept=mean(currPitch)),color="green")+labs(title="Distribucion de currPitch")+geom_text(
  aes(x=0.013,y=30),label="mean",color="green")+geom_vline(aes(xintercept=median(currPitch)),
  color="red")+geom_text(aes(x=0.007,y=30),label="median",color="red")+geom_text(aes(x=0.025,y=55),
  label=paste("mean: ",toString(round(mean(deltaAil$currPitch),5))),color="green") +geom_text(
  aes(x=0.025,y=50),label=paste("median: ",toString(median(deltaAil$currPitch))),color="red"
  )+geom_text(aes(x=0.025,y=45),label=paste("std: ",toString(round(sd(deltaAil$currPitch),5))),
  color="blue")

#Distribución de la variable currRoll
ggplot(deltaAil, aes(x=currRoll))+geom_density(color="darkblue",fill="lightblue")+geom_vline(aes(
  xintercept=mean(currRoll)),color="green")+labs(title="Distribucion de currRoll")+geom_text(aes(
  x=-0.003,y=10),label="mean",color="green")+geom_vline(aes(xintercept=median(currRoll)),color=
  "red")+geom_text(aes(x=0.01,y=10),label="median",color="red")+geom_text(aes(x=0.03,y=22),label=
  paste("mean: ",toString(round(mean(deltaAil$currRoll),5))),color="green") +geom_text(aes(x=0.03,
  y=20),label=paste("median: ",toString(median(deltaAil$currRoll))),color="red") + geom_text(aes(
  x=0.03,y=18),label=paste("std: ",toString(round(sd(deltaAil$currRoll),5))),color="blue")

#Distribución de la variable diffRollRate
ggplot(deltaAil, aes(x=diffRollRate))+geom_density(color="darkblue",fill="lightblue")+geom_vline(
```

```
  aes(xintercept=mean(diffRollRate)),color="green")+labs(title="Distribucion de diffRollRate"
  )+geom_text(aes(x=-0.0001,y=500),label="mean",color="green")+geom_vline(aes(xintercept=median(
  diffRollRate)),color="red")+geom_text(aes(x=0.0001,y=500),label="median",color="red")+geom_text(
  aes(x=0.0005,y=1200),label=paste("mean: ",toString(round(mean(deltaAil$diffRollRate),10))),color
  ="green") +geom_text(aes(x=0.0005,y=1050),label=paste("median: ",toString(median(
  deltaAil$diffRollRate))),color="red")+geom_text(aes(x=0.0005,y=900),label=paste("std: ",toString(
  round(sd(deltaAil$diffRollRate),10))),color="blue")
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Normalización del dataset
deltaAilNormalizado <- data.frame(deltaAil)

deltaAilNormalizado$RollRate <- scale(deltaAil$RollRate)
deltaAilNormalizado$PitchRate <- scale(deltaAil$PitchRate)
deltaAilNormalizado$currPitch <- scale(deltaAil$currPitch)
deltaAilNormalizado$currRoll <- scale(deltaAil$currRoll)
deltaAilNormalizado$diffRollRate <- scale(deltaAil$diffRollRate)
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Correlación entre variable predictoras
max(abs(cor(deltaAil[-6]))%%1)
## ------------------------------------------------------------------------
```

Regresión

```
#########################################################
#Problema de Regresión
#########################################################


## ------------------------------------------------------------------------
#Correlación de las variables respecto a la variable de salida
cor(deltaAil)[6,]
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Correlación entre Sa y RollRate
plot(Sa~RollRate,deltaAil)

#Construcción modelo simple con RollRate
fit1=lm(Sa~RollRate,data=deltaAil)
summary(fit1)

#Construcción modelo simple con PitchRate
fit2=lm(Sa~PitchRate,data=deltaAil)
summary(fit2)
```

```r
#Construcción modelo simple con currPitch
fit3=lm(Sa~currPitch,data=deltaAil)
summary(fit3)

#Construcción modelo simple con currRoll
fit4=lm(Sa~currRoll,data=deltaAil)
summary(fit4)

#Construcción modelo simple con diffRollRate
fit5=lm(Sa~diffRollRate,data=deltaAil)
summary(fit5)

#Modelo simple construido con RollRate
plot(Sa~RollRate,data=deltaAil)
abline(fit1,col="blue")
confint(fit1)
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Ejemplo de predicción del modelo simple
predict(fit1,data.frame(RollRate=c(0.01,-0.01,1)))

#Cálculo del RMSE para el modelo simple construido
yprime=predict(fit1,data.frame(RollRate=deltaAil$RollRate))
sqrt(sum(abs(deltaAil$Sa-yprime)^2)/length(yprime))
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Modelo simple con datos normalizados
fit1n=lm(Sa~RollRate,data=deltaAilNormalizado)
summary(fit1n)
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Correlaciones entre las variables predictoras
cor(deltaAil)[-6,-6]

#Construcción de modelo lineal múltiple
fit6 = lm(Sa~.,data=deltaAil)
summary(fit6)

#Construcción de modelo lineal múltiple
fit7 = lm(Sa~.-currPitch,data=deltaAil)
summary(fit7)

#Construcción de modelo lineal múltiple
fit8 = lm(Sa~.-currPitch-PitchRate,data=deltaAil)
summary(fit8)

#Cálculo de RMSE para el modelo lineal múltiple 8
```

```r
yprime=predict(fit8,deltaAil)
sqrt(sum(abs(deltaAil$Sa-yprime)^2)/length(yprime))
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Comportamiento del modelo lineal múltiple construido para la variable RollRate
plot(deltaAil$Sa~deltaAil$RollRate)
points(deltaAil$RollRate,fitted(fit8),col="blue",pch=20)

#Comportamiento del modelo lineal múltiple construido para la variable currRoll
plot(deltaAil$Sa~deltaAil$currRoll)
points(deltaAil$currRoll,fitted(fit8),col="blue",pch=20)

#Comportamiento del modelo lineal múltiple construido para la variable diffRollRate
plot(deltaAil$Sa~deltaAil$diffRollRate)
points(deltaAil$diffRollRate,fitted(fit8),col="blue",pch=20)
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Construcción de modelo múltiple con interacciones
fit9 = lm(Sa~RollRate+I(RollRate^2),data=deltaAil)
summary(fit9)

#Construcción de modelo múltiple con interacciones
fit10 = lm(Sa~RollRate+diffRollRate+I(RollRate^2)+I(diffRollRate^2)+I((RollRate+diffRollRate)^2),
           data=deltaAil)
summary(fit10)

#Construcción de modelo múltiple con interacciones
fit11 = lm(Sa~RollRate+diffRollRate+currRoll+I((RollRate+diffRollRate+currRoll)^2),
           data=deltaAil)
summary(fit11)

#Construcción de modelo múltiple con interacciones
fit12 = lm(Sa~RollRate+PitchRate+currPitch+currRoll+diffRollRate+I((RollRate+PitchRate+currPitch
          +currRoll+diffRollRate)^2),data=deltaAil)
summary(fit12)

#Construcción de modelo múltiple con interacciones
fit13 = lm(Sa~RollRate+currRoll+diffRollRate+RollRate*currRoll+RollRate*diffRollRate,data=deltaAil)
summary(fit13)

#Construcción de modelo múltiple con interacciones
fit14 = lm(Sa~RollRate+currRoll+diffRollRate+RollRate*currRoll+RollRate*diffRollRate+I(RollRate^2)
           +I(currRoll^2)+I(diffRollRate^2),data=deltaAil)
summary(fit14)

#Construcción de modelo múltiple con interacciones
fit15 = lm(Sa~RollRate+currRoll+diffRollRate+((RollRate+currRoll+diffRollRate)^2)+I(RollRate^2)
           +I(currRoll^2)+I(diffRollRate^2),data=deltaAil)
summary(fit15)
```

```r
#Construcción de modelo múltiple con interacciones
fit16 = lm(Sa~RollRate+currRoll+diffRollRate+currPitch+PitchRate+(.^2)+I(RollRate^2)+I(currRoll^2)
           +I(diffRollRate^2),data=deltaAil)
summary(fit16)

#Construcción de modelo múltiple con interacciones
fit17 = lm(Sa~RollRate+currRoll+diffRollRate+currPitch+PitchRate+(.^3)+I(RollRate^3)+I(currRoll^3)
           +I(diffRollRate^3)+I(PitchRate^3)+I(currPitch^3),data=deltaAil)
summary(fit17)
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Construcción de modelo utlizando kNN y obtención de RMSE
require(kknn)
fitknn1 <- kknn(Sa ~ .-currPitch-PitchRate, deltaAil, deltaAil)
yprime = fitknn1$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))

#Construcción de modelo utlizando kNN y obtención de RMSE
fitknn2 <- kknn(Sa ~ ., deltaAil, deltaAil)
yprime = fitknn2$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))

#Construcción de modelo utlizando kNN y obtención de RMSE
fitknn3 <- kknn(Sa ~ RollRate, deltaAil, deltaAil)
yprime = fitknn3$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))

#Construcción de modelo utlizando kNN y obtención de RMSE
fitknn4 <- kknn(Sa ~ RollRate+diffRollRate+currRoll+RollRate*diffRollRate, deltaAil, deltaAil)
yprime = fitknn4$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))

#Construcción de modelo utlizando kNN y obtención de RMSE
fitknn5 <- kknn(Sa ~ RollRate+diffRollRate+currRoll+RollRate*diffRollRate+I(RollRate^2)
                +I(diffRollRate^2), deltaAil, deltaAil)
yprime = fitknn5$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))

#Construcción de modelo utlizando kNN y obtención de RMSE
fitknn6 <- kknn(Sa ~ RollRate+diffRollRate+currRoll+currPitch+PitchRate+(.^2),
                deltaAil, deltaAil)
yprime = fitknn6$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))

#Construcción de modelo utlizando kNN y obtención de RMSE
fitknn7 <- kknn(Sa ~ RollRate+diffRollRate+currRoll+currPitch+PitchRate+(.^3), deltaAil,
                deltaAil)
yprime = fitknn7$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))

#Construcción de modelo utlizando kNN con datos normalizados y obtención de RMSE
```

```
fitknnNorm <- kknn(Sa ~ RollRate+diffRollRate+currRoll+currPitch+PitchRate+(.^2),
                   deltaAilNormalizado, deltaAilNormalizado)
yprime = fitknnNorm$fitted.values
sqrt(sum((deltaAil$Sa-yprime)^2)/length(yprime))
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Cálculo de MSE para el algoritmo "lm" utilizando validación cruzada
setwd("./delta_ail")

nombre <- "delta_ail"
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~.-X2-X3,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"))
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Cálculo de MSE para el algoritmo "knn" utilizando validación cruzada
setwd("./delta_ail")

nombre <- "delta_ail"
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~.+(.^2),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"))
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Resultados de train y test para los algoritmos lm y kNN
lmMSEtrain
lmMSEtest
knnMSEtrain
knnMSEtest
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Preparación tablas para tests de comparación
setwd("./delta_ail")

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]

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]
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Test de Wilcoxon
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)

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

#Resultados de test de Wilcoxon
```

```
Rmas
Rmenos
pvalue
#Confianza
(1-pvalue)*100
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Test de Wilcoxon para resultados de entrenamiento
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)

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

Rmas
Rmenos
pvalue
#Confianza
(1-pvalue)*100
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Test de Friedman
test_friedman <- friedman.test(as.matrix(tablatst))
test_friedman

#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)
## ------------------------------------------------------------------------
```

Clasificación

```
######################################################
#Problema de Clasificación
######################################################


## ------------------------------------------------------------------------
#División del dataset en entrenamiento y test
set.seed(10)
shuffled <- sample(dim(heart)[1])
eightypct <- (dim(heart)[1] * 80) %/% 100
```

```
heart_train <- heart[shuffled[1:eightypct], 1:13]
heart_test <- heart[shuffled[(eightypct+1):dim(heart)[1]], 1:13]

heart_train_labels <- heart[shuffled[1:eightypct], 14]
heart_test_labels <- heart[shuffled[(eightypct+1):dim(heart)[1]], 14]

#División de los datos normalizados en entrenamiento y test
set.seed(10)
shuffledN <- sample(dim(heartNormalizado)[1])
heartN_train <- heartNormalizado[shuffledN[1:eightypct], 1:13]
heartN_test <- heartNormalizado[shuffledN[(eightypct+1):dim(heartNormalizado)[1]], 1:13]

heartN_train_labels <- heartNormalizado[shuffledN[1:eightypct], 14]
heartN_test_labels <- heartNormalizado[shuffledN[(eightypct+1):dim(heartNormalizado)[1]], 14]
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Construcción de modelos utilizando kNN con diferentes "k"
library(class)
require(caret)
set.seed(10)
getKnn <- function(miK){

  heart_test_pred <- knn(train = heart_train, test = heart_test, cl = heart_train_labels, k=miK)

  postResample(pred = heart_test_pred, obs = heart[shuffled[(eightypct+1):dim(heart)[1]], 14])
}
result1 <- lapply(1:20,getKnn)
result1 <- unlist(result1)[1:20*2-1]
#Dado que existe aleatoriedad en este proceso se realiza 5 veces y se trata con la media
result2 <- lapply(1:20,getKnn)
result2 <- unlist(result2)[1:20*2-1]
result3 <- lapply(1:20,getKnn)
result3 <- unlist(result3)[1:20*2-1]
result4 <- lapply(1:20,getKnn)
result4 <- unlist(result4)[1:20*2-1]
result5 <- lapply(1:20,getKnn)
result5 <- unlist(result5)[1:20*2-1]

result<-unlist(Map("+",unlist(Map("+",unlist(Map("+",unlist(Map("+",result1,result2)),
                                                 result3)),result4)),result5))

result<-result/5
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Gráfica de resultados obtenidos en diferentes "k"
df <- data.frame(k=1:20,accuracy=result)
ggplot(df, aes(x=k,y=accuracy)) + geom_histogram(stat="identity",color="black",fill="deepskyblue"
  )+coord_cartesian(ylim=c(0.65,0.8))+ labs(title="Acierto de kNN con diferentes k")
```

```r
#Mejor resultado obtenido
max(result)
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Construcción de modelos con kNN con datos normalizados y diferentes "k"
set.seed(10)
getKnn <- function(miK){

  heart_test_pred <- knn(train = heartN_train, test = heartN_test, cl = heartN_train_labels, k=miK)

  postResample(pred = heart_test_pred, obs =
                 heartNormalizado[shuffledN[(eightypct+1):dim(heartNormalizado)[1]], 14])
}

result1 <- lapply(1:20,getKnn)
result1 <- unlist(result1)[1:20*2-1]
#Dado que existe aleatoriedad en este proceso se realiza 5 veces y se trata con la media
result2 <- lapply(1:20,getKnn)
result2 <- unlist(result2)[1:20*2-1]
result3 <- lapply(1:20,getKnn)
result3 <- unlist(result3)[1:20*2-1]
result4 <- lapply(1:20,getKnn)
result4 <- unlist(result4)[1:20*2-1]
result5 <- lapply(1:20,getKnn)
result5 <- unlist(result5)[1:20*2-1]

result<-unlist(Map("+",unlist(Map("+",unlist(Map("+",unlist(Map("+",result1,result2)),
                                                 result3)),result4)),result5))
print(result)

result<-result/5
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Gráfica con resultados de kNN y sus diferentes "k"
df <- data.frame(k=1:20,accuracy=result)

ggplot(df, aes(x=k,y=accuracy)) + geom_histogram(stat="identity",color="black",fill="deepskyblue"
  )+coord_cartesian(ylim=c(0.7,0.85))+ labs(title="Acierto de kNN con diferentes k en datos
  normalizados")

#Mejor resultado obtenido
max(result)
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#División de los datos en entrenamiento y test
set.seed(10)
shuffled <- sample(dim(heart)[1])
```

```
eightypct <- (dim(heart)[1] * 80) %/% 100
heart_train <- heart[shuffled[1:eightypct], ]
heart_test <- heart[shuffled[(eightypct+1):dim(heart)[1]], ]
## ------------------------------------------------------------------------



## ------------------------------------------------------------------------
#Construcción del modelo con LDA
library(MASS)
library(ISLR)
ldaFit <- lda(Class ~.,data=heart_train)
ldaFit

#Resultados del modelo construido con LDA
ldaPred <- predict(ldaFit,heart_test)

table(ldaPred$class,heart_test$Class)
resultLDA <- mean(ldaPred$class==heart_test$Class)
resultLDA
## ------------------------------------------------------------------------



## ------------------------------------------------------------------------
#Construcción y resultados de un modelo LDA sobre datos normalizados
set.seed(10)
shuffledN <- sample(dim(heartNormalizado)[1])
heartN_train <- heartNormalizado[shuffledN[1:eightypct], ]
heartN_test <- heartNormalizado[shuffledN[(eightypct+1):dim(heartNormalizado)[1]], ]

ldaFitN <- lda(Class ~.,data=heartN_train)
ldaFitN

ldaPredN <- predict(ldaFitN,heartN_test)
table(ldaPredN$class,heartN_test$Class)
resultLDAN <- mean(ldaPredN$class==heartN_test$Class)
resultLDAN
## ------------------------------------------------------------------------



## ------------------------------------------------------------------------
#Construcción del modelo con QDA
qdaFit <- qda(Class ~.,data=heart_train)
qdaFit

#Resultados del modelo construido con QDA
qdaPred <- predict(qdaFit,heart_test)

table(qdaPred$class,heart_test$Class)
resultQDA <- mean(qdaPred$class==heart_test$Class)
resultQDA
## ------------------------------------------------------------------------
```

```
## ------------------------------------------------------------------------
#Construcción y resultados del modelo QDA sobre datos normalizados
qdaFitN <- qda(Class ~.,data=heartN_train)
qdaFitN

qdaPredN <- predict(qdaFitN,heartN_test)
table(qdaPredN$class,heartN_test$Class)
resultQDAN <- mean(qdaPredN$class==heartN_test$Class)
resultQDAN
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Obtención de resultados de kNN utilizando validación cruzada y diferentes "k"
setwd("./heart")

nombre <- "heart"
run_knn_fold <- function(i, x, tt = "test",miK) {
  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="@")
  if (tt == "train") {
    test <- x_tra
  }
  else {
    test <- x_tst
  }


  #Normalizacion
  heartN_train <- data.frame(x_tra)
  names(heartN_train) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
  "FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced",
  "Oldpeak", "Slope", "MajorVessels", "Thal", "Class")

  heartN_train$RestBloodPressure <- scale(heartN_train$RestBloodPressure)
  heartN_train$SerumCholestoral <- scale(heartN_train$SerumCholestoral)
  heartN_train$MaxHeartRate <- scale(heartN_train$MaxHeartRate)
  heartN_train$Oldpeak <- scale(heartN_train$Oldpeak)


  heartN_test <- data.frame(test)
  names(heartN_test) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
  "FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced", "Oldpeak",
  "Slope", "MajorVessels", "Thal", "Class")

  heartN_test$RestBloodPressure <- scale(heartN_test$RestBloodPressure)
  heartN_test$SerumCholestoral <- scale(heartN_test$SerumCholestoral)
  heartN_test$MaxHeartRate <- scale(heartN_test$MaxHeartRate)
  heartN_test$Oldpeak <- scale(heartN_test$Oldpeak)

  #Preparacion datos
```

```
  heartN_train$Class <- factor(heartN_train$Class, levels = c(1, 2), labels = c("Si", "No"))
  heartN_test$Class <- factor(heartN_test$Class, levels = c(1, 2), labels = c("Si", "No"))

  heartN_train_labels <- heartN_train$Class
  heartN_test_labels <- heartN_test$Class

  heartN_train <- heartN_train[,1:13]
  heartN_test <- heartN_test[,1:13]

  #Construcción modelo kNN y resultado
  heart_test_pred <- knn(train = heartN_train, test = heartN_test, cl = heartN_train_labels, k=miK)
  postResample(pred = heart_test_pred, obs = heartN_test_labels)

}

#Función utilizada para construir modelos con diferentes "k"
mejorK <- function(miK){

  knnAcctrain<-mean(sapply(1:10,run_knn_fold,nombre,"train",miK))
  knnAcctest<-mean(sapply(1:10,run_knn_fold,nombre,"test",miK))
  list(knnAcctrain,knnAcctest)
}

resultados <- lapply(1:20,mejorK)
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Obtención de la mejor "k"
which.max(unlist(resultados)[1:20*2])

#Resultados obtenidos
knnAcctrain <- unlist(resultados[7])[1]
knnAcctest <- unlist(resultados[7])[2]
knnAcctrain
knnAcctest
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Resultado de modelo LDA utilizando validación cruzada
setwd("./heart")

nombre <- "heart"
run_lda_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="@")
  if (tt == "train") {
    test <- x_tra
  }
  else {
```

```
    test <- x_tst
  }


  #Preparación datos
  heartN_train <- data.frame(x_tra)
  names(heartN_train) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
  "FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced", "Oldpeak",
  "Slope", "MajorVessels", "Thal", "Class")


  heartN_test <- data.frame(test)
  names(heartN_test) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
  "FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced", "Oldpeak",
  "Slope", "MajorVessels", "Thal", "Class")


  heartN_train$Class <- factor(heartN_train$Class, levels = c(1, 2), labels = c("Si", "No"))
  heartN_test$Class <- factor(heartN_test$Class, levels = c(1, 2), labels = c("Si", "No"))

  #Construcción modelo y resultado
  ldaFit <- lda(Class ~.,data=heartN_train)
  ldaPred <- predict(ldaFit,heartN_test)

  table(ldaPred$class,heartN_test$Class)
  resultLDA <- mean(ldaPred$class==heartN_test$Class)
  resultLDA

}

#Resultados obtenidos
ldaAcctrain<-mean(sapply(1:10,run_lda_fold,nombre,"train"))
ldaAcctest<-mean(sapply(1:10,run_lda_fold,nombre,"test"))
ldaAcctrain
ldaAcctest
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Resultados del modelo QDA utilizando validación cruzada
setwd("./heart")

nombre <- "heart"
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="@")
  if (tt == "train") {
    test <- x_tra
  }
  else {
    test <- x_tst
```

```
  }


  #Preparación datos
  heartN_train <- data.frame(x_tra)
  names(heartN_train) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
  "FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced", "Oldpeak",
  "Slope", "MajorVessels", "Thal", "Class")


  heartN_test <- data.frame(test)
  names(heartN_test) <- c("Age", "Sex", "ChestPainType", "RestBloodPressure", "SerumCholestoral",
  "FastingBloodSugar", "ResElectrocardiographic", "MaxHeartRate", "ExerciseInduced", "Oldpeak",
  "Slope", "MajorVessels", "Thal", "Class")


  heartN_train$Class <- factor(heartN_train$Class, levels = c(1, 2), labels = c("Si", "No"))
  heartN_test$Class <- factor(heartN_test$Class, levels = c(1, 2), labels = c("Si", "No"))

  #Construcción de modelo
  qdaFit <- qda(Class ~.,data=heartN_train)
  qdaPred <- predict(qdaFit,heartN_test)

  table(qdaPred$class,heartN_test$Class)
  resultQDA <- mean(qdaPred$class==heartN_test$Class)
  resultQDA

}


#Resultados obtenidos
qdaAcctrain<-mean(sapply(1:10,run_qda_fold,nombre,"train"))
qdaAcctest<-mean(sapply(1:10,run_qda_fold,nombre,"test"))
qdaAcctrain
qdaAcctest
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Preparación tablas para comparación de algoritmos
setwd("./heart")

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]
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Test de Wilcoxon
#kNN vs LDA
difs <- (tablatst[,1] - tablatst[,2]) / tablatst[,1]
```

```r
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)

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
Rmenos
pvalue
#Confianza
(1-pvalue)*100
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Test de Wilcoxon
#kNN vs QDA
difs <- (tablatst[,1] - tablatst[,3]) / tablatst[,1]
wilc_1_3 <- cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1), ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_3) <- c(colnames(tablatst)[1], colnames(tablatst)[3])
head(wilc_1_3)

LMvsKNNtst <- wilcox.test(wilc_1_3[,1], wilc_1_3[,2], alternative = "two.sided", paired=TRUE)
Rmas <- LMvsKNNtst$statistic
pvalue <- LMvsKNNtst$p.value
LMvsKNNtst <- wilcox.test(wilc_1_3[,2], wilc_1_3[,1], alternative = "two.sided", paired=TRUE)
Rmenos <- LMvsKNNtst$statistic
Rmas
Rmenos
pvalue
#Confianza
(1-pvalue)*100
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Test de Wilcoxon
#LDA vs QDA
difs <- (tablatst[,2] - tablatst[,3]) / tablatst[,2]
wilc_2_3 <- cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1), ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_2_3) <- c(colnames(tablatst)[2], colnames(tablatst)[3])
head(wilc_2_3)

LMvsKNNtst <- wilcox.test(wilc_2_3[,1], wilc_2_3[,2], alternative = "two.sided", paired=TRUE)
Rmas <- LMvsKNNtst$statistic
pvalue <- LMvsKNNtst$p.value
LMvsKNNtst <- wilcox.test(wilc_2_3[,2], wilc_2_3[,1], alternative = "two.sided", paired=TRUE)
Rmenos <- LMvsKNNtst$statistic
Rmas
Rmenos
```

```r
pvalue
#Confianza
(1-pvalue)*100
## ------------------------------------------------------------------------


## ------------------------------------------------------------------------
#Test de Friedman
test_friedman <- friedman.test(as.matrix(tablatst))
test_friedman

#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)
## ------------------------------------------------------------------------
```