“Trabajo Teórico/Práctico Integrador - Introducción a la Ciencia Datos”

“Cristian Rohr”

Contents

Introducción

Este trabajo consta de tres apartados: Análisis de datos, Regresión y Clasificación.

Se analizaran dos sets de datos, uno para el apartado de clasificación y otro para el apartado de regresión. Ambos datasets tendrán un análisis exploratorio previo.

El dataset a utilizar para el apartado de clasificación es wisconsin y para el apartado de regresión ANACALT.

Objetivo

El objetivo del trabajo es aplicar las técnicas y algoritmos aprendidos durante el curso para analizar los sets de datos y comparar la performance de diferentes algoritmos sobre ellos.

Apartado Análisis de Datos

El objetivo del análisis exploratorio de datos es resumir y visualizar los datos para poder determinar cuales variables son las más importantes para predecir la variable respuesta. Además de identificar relaciones o distribuciones que nos ayuden a seleccionar un modelo, o optimizar sus parámetros.

Los datos se encuentran almacenados en ficheros con formato keel. Se leen a un dataframe de R y se renombran las columnas.

Set de datos regresión (ANACALT)

Inspección del dataset y generación de hipótesis

```
# Lectura dataset de regresión: ANACALT
datasetR <- read.csv("DATOS/Datasets Regresion/ANACALT/ANACALT.dat",
                     comment.char = "@")
# Asignación automática de nombres a las variables
n <- length(names(datasetR)) - 1
names(datasetR)[1:n] <- paste("X", 1:n, sep = "") # Variables predictoras
names(datasetR)[n+1] <- "Y" # Variable respuesta

class(datasetR)  # Donde se almacenan los datos
dim(datasetR)    # Dimensiones del dataset
str(datasetR)    # Estructura del dataset, tipos de datos atómicos
# Inspección de las primeras 10 filas
head(datasetR)
# Inspección de las últimas 10 filas
tail(datasetR)
```

Estadística descriptiva

Utilizo las funciones `summary` y `describe` para obtener estadísticas descriptivas del dataset.

```
# Estadísticas descriptivas básicas
summary(datasetR)
```

```
##        X1                X2               X3                X4
##  Min.   : 0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.00000
##  1st Qu.: 0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.00000
##  Median : 0.0000   Median :1.0000   Median :0.00000   Median :0.00000
##  Mean   : 0.1089   Mean   :0.5196   Mean   :0.07825   Mean   :0.02296
##  3rd Qu.: 0.0000   3rd Qu.:1.0000   3rd Qu.:0.00000   3rd Qu.:0.00000
##  Max.   :11.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.00000
##        X5                X6              X7                Y
##  Min.   :0.0000   Min.   :1953   Min.   :0.0000   Min.   :0.000
##  1st Qu.:0.0000   1st Qu.:1964   1st Qu.:0.0000   1st Qu.:2.080
##  Median :0.0000   Median :1973   Median :0.0000   Median :2.300
##  Mean   :0.3374   Mean   :1972   Mean   :0.2254   Mean   :2.034
##  3rd Qu.:1.0000   3rd Qu.:1981   3rd Qu.:0.0000   3rd Qu.:2.300
##  Max.   :1.0000   Max.   :1988   Max.   :1.0000   Max.   :2.300
```

Nota: Resultados del comando *describe* no son mostrados debido a la longitud de los mismos.

```r
# No muestro los resultados debido a la longitud de los mismos
describe(datasetR)
```

- El set de datos consta de 4051 observaciones de 8 variables. Al inspeccionar los datos cargados todas las variables toman valores numéricos.
- 5 variables (X2, X3, X4, X5 y X7) toman valores dos valores posible 0 o 1, y deben ser convertidas a categóricas (factor en R).
- X1 y X6 son contínuas. La variable independiente Y toma valores continuos.

Los nombres originales de las variables son los siguientes:

- Actions_taken: [0.0, 11.0]
- Liberal: [0.0, 1.0]
- Unconstitutional: [0.0, 1.0]
- Precedent_alteration: [0.0, 1.0]
- Unanimous: [0.0, 1.0]
- Year_of_decision: [1953.0, 1988.0]
- Lower_court_disagreement: [0.0, 1.0]
- Log_exposure: [0.0, 2.3]

El objetivo es predecir las decisiones de la corte. Los datos son de los años 1953 a 1988.

Algunas hipótesis a priori sobre los datos

- Para la variable X3 (Unconstitutional) si asumimos 0 como negativo y 1 como positivo, es de esperar una baja cantidad de valores 1 para la variable, ya que serían decisiones inconstitucionales.
- Para la variable X4 (Precedent_alteration) si asumimos 0 como negativo y 1 como positivo, es de esperar una baja cantidad de valores 1 para la variable, ya que las decisiones de la corte generalmente tienen en cuenta la jurisprudencia previa, y no se suelen alterar las decisiones.
- Para la variable X7 (Lower_court_disagreement) si asumimos 0 como negativo y 1 como positivo, es de esperar una baja cantidad de valores 1 para la variable, ya que generalmente hay coincidencia entre las cámaras.
- Con respecto a la variable X6 (Year_of_decision) esperaríamos una mayor cantidad de decisiones a medida que incrementan los años, asumiendo que un incremento poblacional acarrea mas juicios.
- ¿Las decisiones que suponen una alteración de los precedentes son inconstitucionales?. Es decir ¿si X4 es 1, entonces X3 es 1?.

Valores faltantes y duplicados

```r
# Calculo la cantidad de duplicados
nrow(datasetR[duplicated(datasetR), ])
```

```
## [1] 3442
```

- Los resultados de summary y describe indican que no existen valores faltantes. En caso de haber existido, se podría haber realizado un análisis más profundo con el paquete *VIM*.
- Existe una gran cantidad de valores duplicados (3442), por este motivo asumo que se trata de observaciones independientes con valores iguales.

Visualización del dataset

```
# Para facilitar el uso de ggplot2, genero un nuevo dataframe en
#formato `long` en lugar de `wide`.
datasetR_long <- melt(data = datasetR)

# Histograma
plothistR <- ggplot(datasetR_long, aes(x = value)) +
   geom_histogram() +
   facet_wrap(~variable, scales = "free") +
   xlab("Valor") + ylab("Conteo") +
   theme_light()

# barplot variable year
plotbaryear <- ggplot(data = datasetR, aes(X6)) +
  geom_bar() + ylab("Conteo") + theme_light()

# qq-plots
plotqqR <- ggplot(datasetR_long, aes(sample = value)) +
  stat_qq() +
  stat_qq_line() +
  xlab("Teórico") +
  ylab("Muestra") +
  facet_wrap(~variable, scale = "free") +
  theme_light()

# boxplots
plotboxR <- ggplot(data = datasetR_long, aes(x = variable, y = value)) +
  geom_boxplot() + xlab("Variable") +
  ylab("Valor") +
  theme_light() + facet_wrap(~variable, scales = "free")

# ggpairs plot
plotpairsR <- ggpairs(data = datasetR)
```

Gráficos Univariados

1. Histogramas: chequeo de la distribución de los datos. Ver Figura 1.

Esta primer visualización nos permite chequear algunas de las hipótesis a priori sobre el dataset

- Para el caso de las variables X3 (Unconstitutional) y X4 (Precedent_alteration) se cumple la hipótesis.
- Para la variable X7 (Lower_court_disagreement) también se cumple, pero no en la proporción que imaginaba.
- Con respecto a la variable X6 (Year_of_decision) se observa un incremento del número de decisiones con el correr de los años como se esperaba. Se observa la presencia de un patrón que exploré en mayor detalle en la Figura 2. Esta figura muestra que anteriormente solo veíamos un artefacto por el tamaño de bin en el histograma.

2. Q-Q plot, chequeo la normalidad de los datos. Ver Figura 3.

- Los datos no siguen una distribución normal. Al ser datos categóricos en su mayoría, no es factible aplicar una transformación para convertirlos a una distribución normal.

3. Diagramas de caja (boxplots), permiten la identificación visual de estadísticas y presencia de outliers. Ver Figura 4.

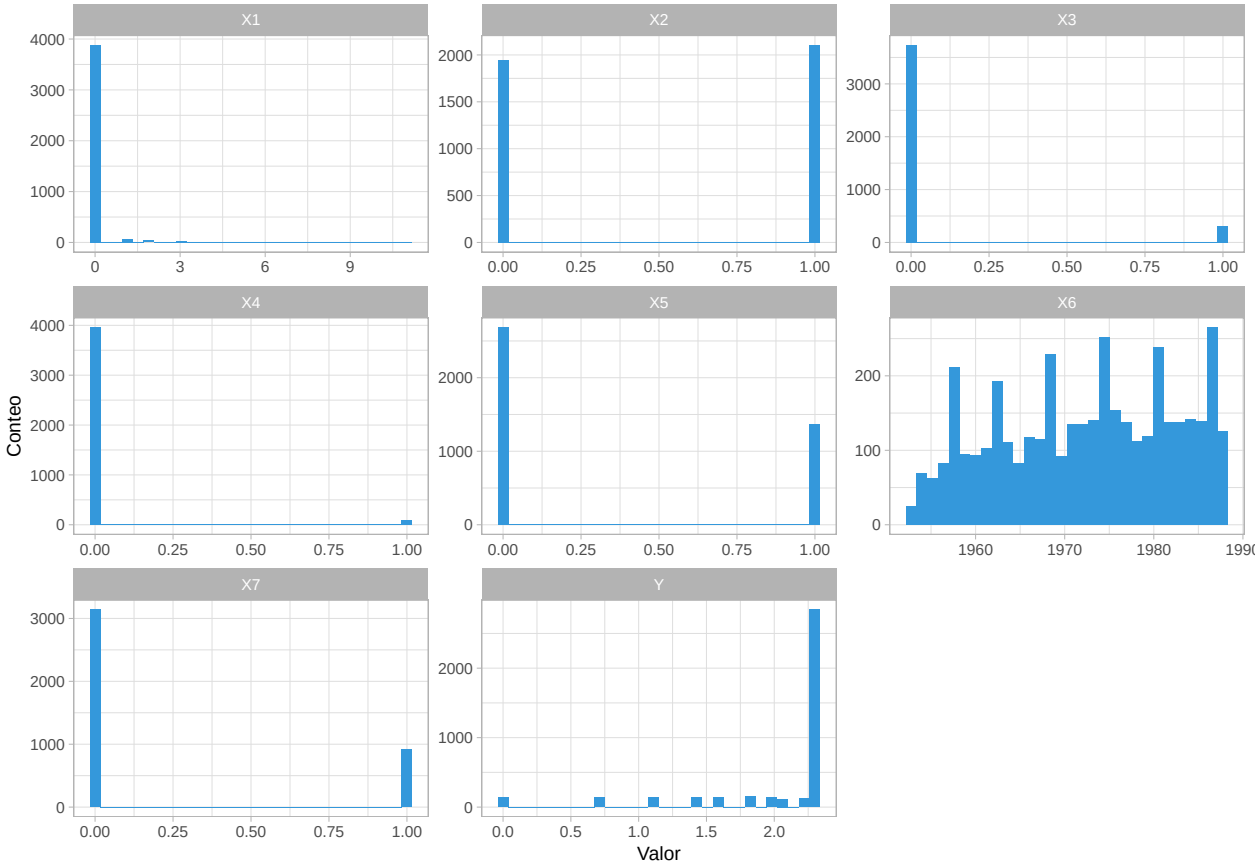


Figure 1: Histogramas para cada variable del dataset de regresión.

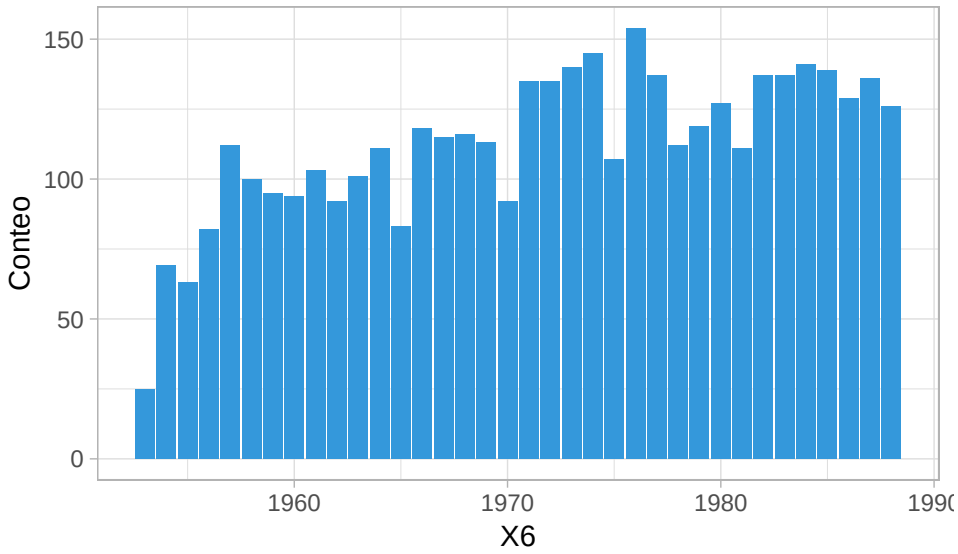


Figure 2: Cantidad de decisiones por año.

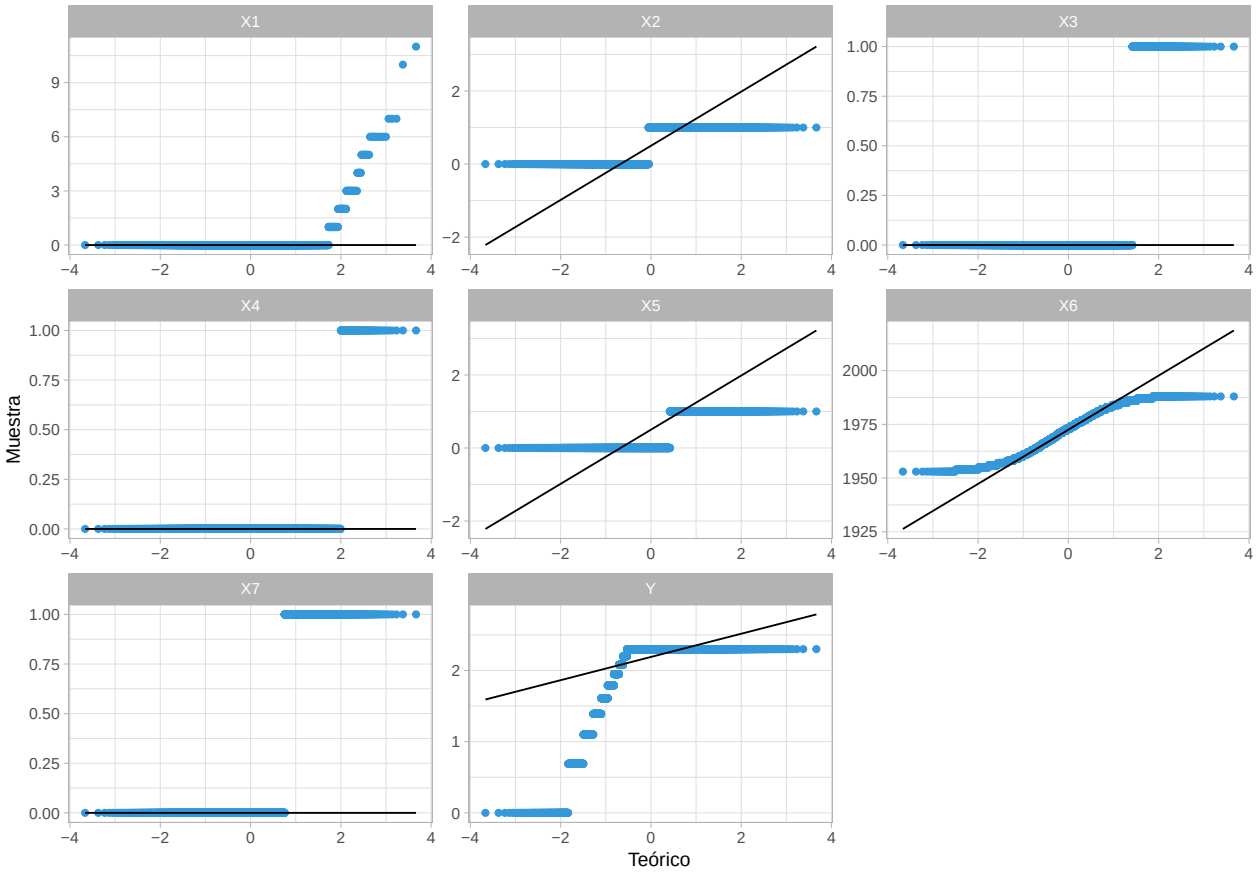


Figure 3: Q-Q plot para cada variable del dataset de regresión.

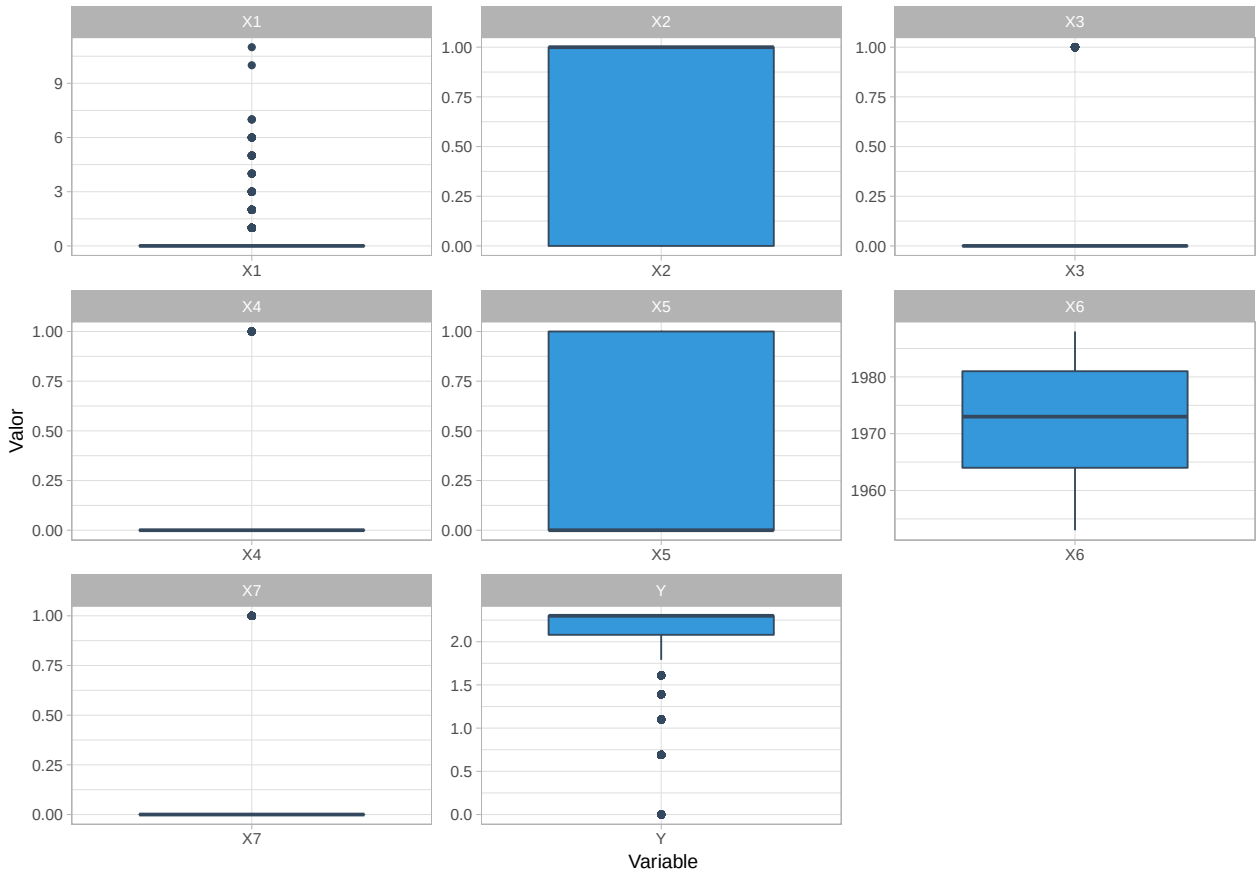


Figure 4: Boxplots para cada variable del dataset de regresión.

Gráficos multivariados

1. Análisis de correlación entre las variables. Ver Figura 5.

```
datasetR.cor <- cor(datasetR)
corrplot(datasetR.cor, method = "circle")
```

- No existe una correlación elevada entre variables.

2. Uso de la función *ggpairs* de la librería *GGally* para detectar posibles relaciones entre las variables. Ver Figura 6.

- De acuerdo a los valores de correlación y los gráficos exploratorios, la variable con mejor posibilidad de estar relacionada a la variable Y, es la variable X6.
- Debido a la naturaleza del dataset, con la presencia de varias variables categóricas es difícil indicar a priori cual de las otras variables tienen una posible relación lineal con la variable respuesta.

3. Visualización de pares de variables por escala de grises. Ver Figura 7.

```
temp <- datasetR
plotgrayR <- plot(temp[,-dim(temp)[2]],pch=16,
     col=gray(1-(temp[,dim(temp)[2]]/max(temp[,dim(temp)[2]]))))
```

Chequeo de hipótesis

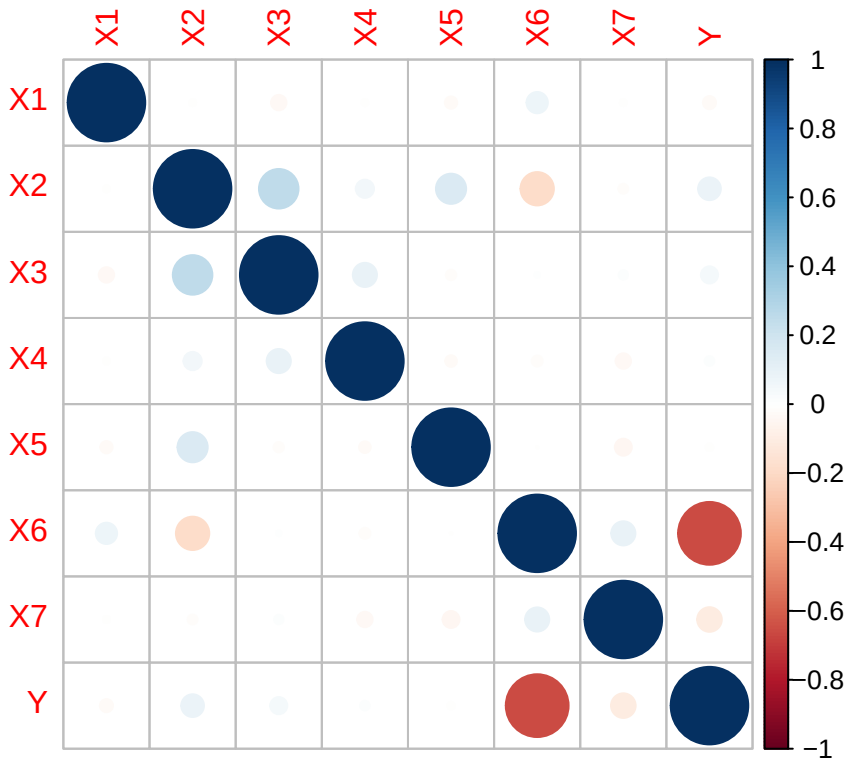


Figure 5: Correlación entre variables del dataset de regresión.

Chequeo la hipótesis de que las decisiones que no tienen en cuenta las decisiones previas son inconstitucionales. Para esto utilizo un test chi cuadrado.

```
chis <- chisq.test(table(datasetR$X3, datasetR$X4))
chis
```

```
##
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table(datasetR$X3, datasetR$X4)
## X-squared = 35.356, df = 1, p-value = 2.746e-09
```

- El p-valor significativo indica que ambas variables estan relacionadas.

Preprocesamiento dataset de Regresión

Luego del análisis exploratorio de los datos, voy a proceder a realizar un tratamiento de los mismos, para luego avanzar con los modelos de clasificación.

- Convierto a factor las variables categóricas.

```
datasetR$X2 <- as.factor(datasetR$X2)
datasetR$X3 <- as.factor(datasetR$X3)
datasetR$X4 <- as.factor(datasetR$X4)
datasetR$X5 <- as.factor(datasetR$X5)
datasetR$X7 <- as.factor(datasetR$X7)
```

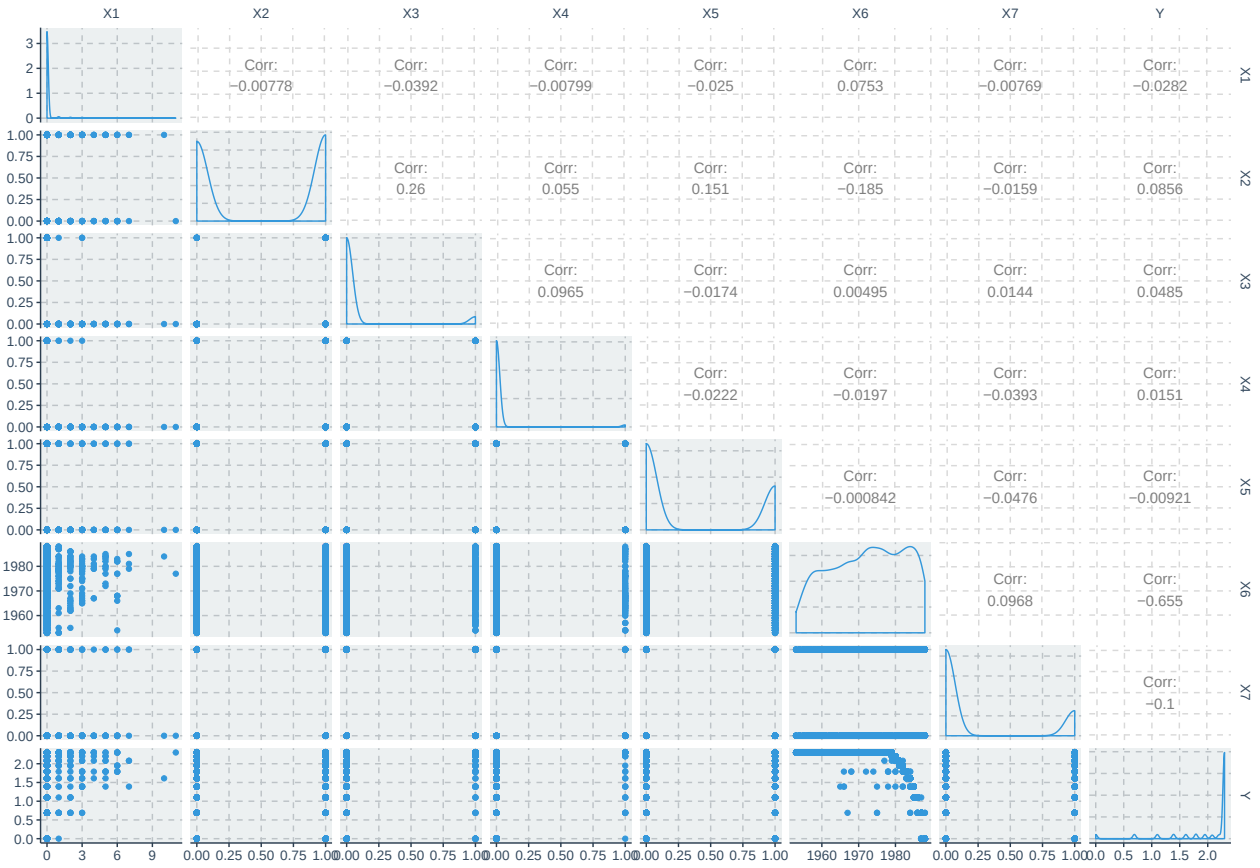


Figure 6: Gráfico utilizando la funcion ggpairs para explorar relaciones entre variables del datset de regresión.

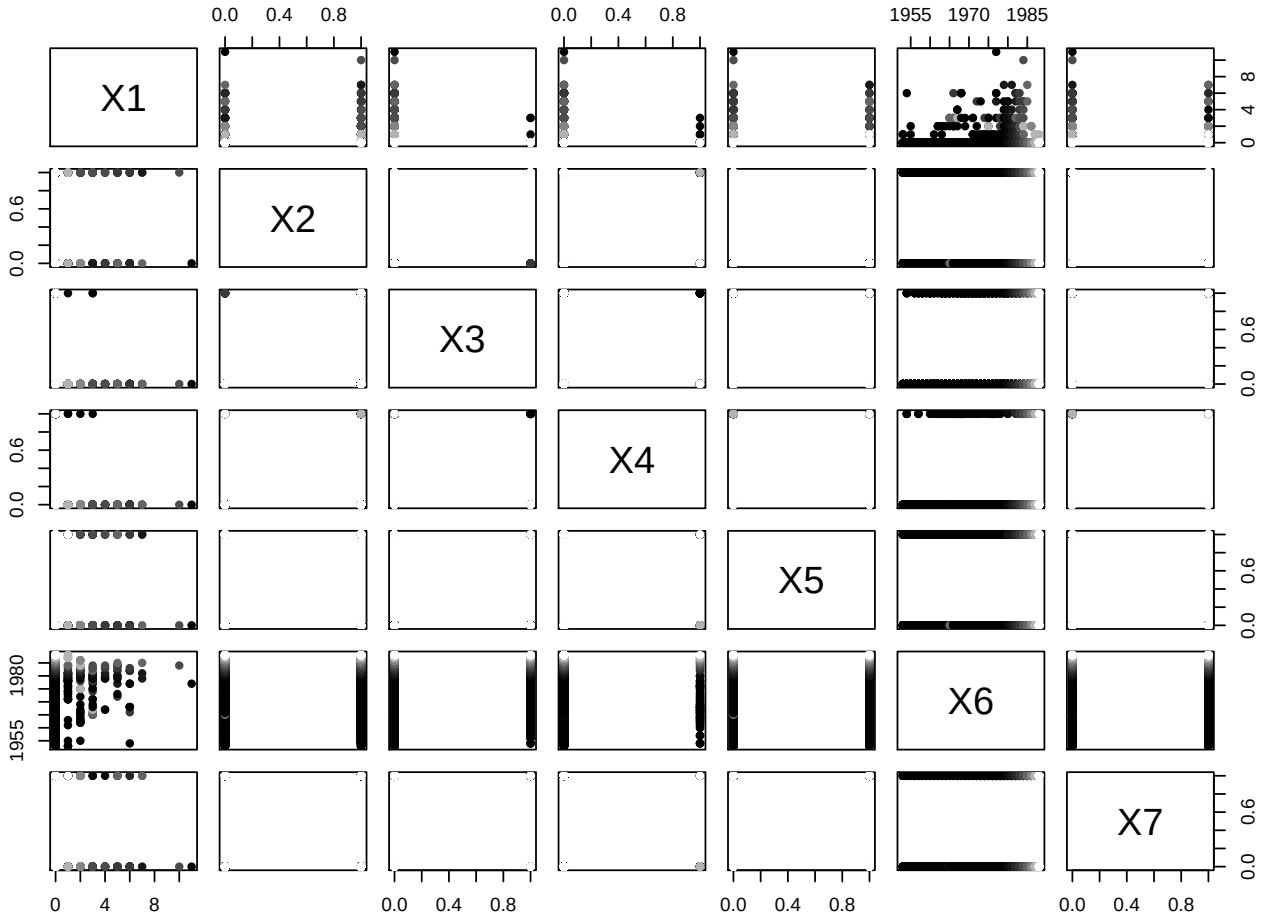


Figure 7: Visualización de pares de variables por escala de grises, para el dataset de regresión.

Set de datos clasificación (wisconsin)

Inspección del dataset

```
# Lectura dataset clasificación: wisconsin
datasetC <- read.csv("DATOS/Datasets Clasificacion/wisconsin/wisconsin.dat",
                     comment.char = "@")
# Asignación automática de nombres a las variables
n <- length(names(datasetC)) - 1
names(datasetC)[1:n] <- paste("X", 1:n, sep = "") # Variables predictoras
names(datasetC)[n+1] <- "Y" # Variable respuesta

# Guardo un dataset sin procesar para comparar la cross validation
datasetC_sin_procesar <- datasetC
datasetC_sin_procesar$Y <- as.factor(datasetC_sin_procesar$Y)

class(datasetC)  # Donde se almacenan
dim(datasetC)    # Dimensiones del dataset
str(datasetC)    # Estructura del dataset, tipos de datos atómicos
# Inspección de las primeras 10 filas
head(datasetC)
# Inspección de las ultimas 10 filas
tail(datasetC)
```

Estadística descriptiva

Utilizo las funciones `summary` y `describe` para obtener estadísticas descriptivas del dataset.

Nota: los resultados de la función describe no se muestran debido a la longitud de los mismos.

```
describe(datasetC) # Estadísticas descriptivas básicas
```

```
summary(datasetC) # Estadísticas descriptivas básicas
```

```
##        X1              X2              X3              X4
##  Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.: 1.000
##  Median : 4.000   Median : 1.000   Median : 1.000   Median : 1.000
##  Mean   : 4.441   Mean   : 3.154   Mean   : 3.218   Mean   : 2.833
##  3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 5.000   3rd Qu.: 4.000
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000
##        X5              X6              X7              X8
##  Min.   : 1.000   Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.: 2.000   1st Qu.: 1.000   1st Qu.: 2.000   1st Qu.: 1.000
##  Median : 2.000   Median : 1.000   Median : 3.000   Median : 1.000
##  Mean   : 3.236   Mean   : 3.548   Mean   : 3.446   Mean   : 2.872
##  3rd Qu.: 4.000   3rd Qu.: 6.000   3rd Qu.: 5.000   3rd Qu.: 4.000
##  Max.   :10.000   Max.   :10.000   Max.   :10.000   Max.   :10.000
##        X9              Y
##  Min.   : 1.000   Min.   :2.000
##  1st Qu.: 1.000   1st Qu.:2.000
##  Median : 1.000   Median :2.000
##  Mean   : 1.604   Mean   :2.701
##  3rd Qu.: 1.000   3rd Qu.:4.000
```

```
##  Max.   :10.000   Max.   :4.000
```

```
table(as.factor(datasetC$Y)) # Distribución de la variable respuesta
```

```
##
##   2   4
## 443 239
```

- Es un dataset multivariado.
- El dataset consta de 682 observaciones de 10 variables.
- Cada una de las variables en las 10 columnas corresponde a un tipo de dato entero (int) de R al momento de cargar los datos. Son cuantitativas numéricas y toman valores discretos.
- Las 9 variables independientes (predictoras) tienen valores en el rango 1-10
- La variable dependiente (respuesta) toma dos valores 2-4, por lo tanto es categórica. Es necesario convertirla de tipo `int` a `factor`.
- La variable respuesta esta desbalanceada, un clasificador tonto que siempre asigne todas las observaciones a la clase mayoritaria obtendría un 64,95% de acierto.
- No hay datos faltantes.

Los nombres originales de las variables son los siguientes:

- ClumpThickness
- CellSize
- CellShape
- MarginalAdhesion
- EpithelialSize
- BareNuclei
- BlandChromatin
- NormalNucleoli
- Mitoses
- Class (2 benigno, 4 maligno)

Chequeo la existencia de duplicados

```
nrow(datasetC[duplicated(datasetC), ])
```

```
## [1] 233
```

Existe 233 observaciones duplicadas. Al ser tantas muestras duplicadas, decido no realizar ninguna acción sobre ellas, asumiendo que se trata de observaciones independientes.

Visualización del dataset

```
# Para facilitar el uso de ggplot2, genero un nuevo dataframe en
# formato `long` en lugar de `wide`.
datasetC$Y <- as.factor(datasetC$Y)
datasetC_long <- melt(data = datasetC)

# Histograma
plothistC <- ggplot(datasetC_long, aes(x = value)) +
    geom_histogram(aes(y =..density..),
                   breaks = seq(0, 10, by = 2.5)) +
stat_function(fun = dnorm,
              args = list(mean = mean(datasetC_long$value),
                          sd = sd(datasetC_long$value))) +
```

```
   facet_wrap(~variable) + xlab("Valor") + ylab("Conteo") +
   theme_light()

# Grafico de densidad respecto a la variable Y
plotdensidadC <- ggplot(datasetC_long, aes(x = value, fill = Y, colour = Y)) +
   geom_density(alpha = 0.1) +
   facet_wrap(~variable) +
   xlab("Valor") + ylab("Densidad") +
   theme_light()

# Q-Q plot
plotqqC <- ggplot(datasetC_long, aes(sample = value)) +
  stat_qq() +
  stat_qq_line() +
  xlab("Teórico") + ylab("Muestra") +
  facet_wrap(~variable) +
  theme_light()

# Boxplot
plotboxC <- ggplot(data = datasetC_long, aes(x = variable, y = value)) +
  geom_boxplot() +
  xlab("Variable") + ylab("Valor") +
  theme_light()

# Pairs plot GGally library
plotpairsC <- ggpairs(data = datasetC)
```

Gráficos Univariados

1. Histogramas: chequeo de la distribución de los datos. Ver Figura 8.
2. Gráficos de densidad, considerando las categorías de la variable respuesta. Ver Figura 9.

- Se puede observar que la distribución de las variables predictoras, es diferente en función de la variable respuesta.

3. q-q plots, chequeo de la normalidad de los datos. Ver Figura 10.
4. Boxplot de los datos, identificación de outliers. Ver Figura 11.

- La visualización de los boxplots indica la existencia de outliers en 5 variables: X4, X5, X7, X8 y X9. Siendo X9 la más afectada por los outliers.

Gráficos multivariados

1. Análisis de correlación entre las variables. Ver Figura 17.

```
datasetC.cor <- cor(datasetC[, 1:9])
corrplot(datasetC.cor, method = "number")
```

- Hay variables que se encuentran correlacionadas. Estas variables contienen información redundante. Es recomendable remover este tipo de variables, para evitar bias en los modelos.

2. Inspección de los datos para detectar posibles relaciones entre las variables. Ver Figura 13.

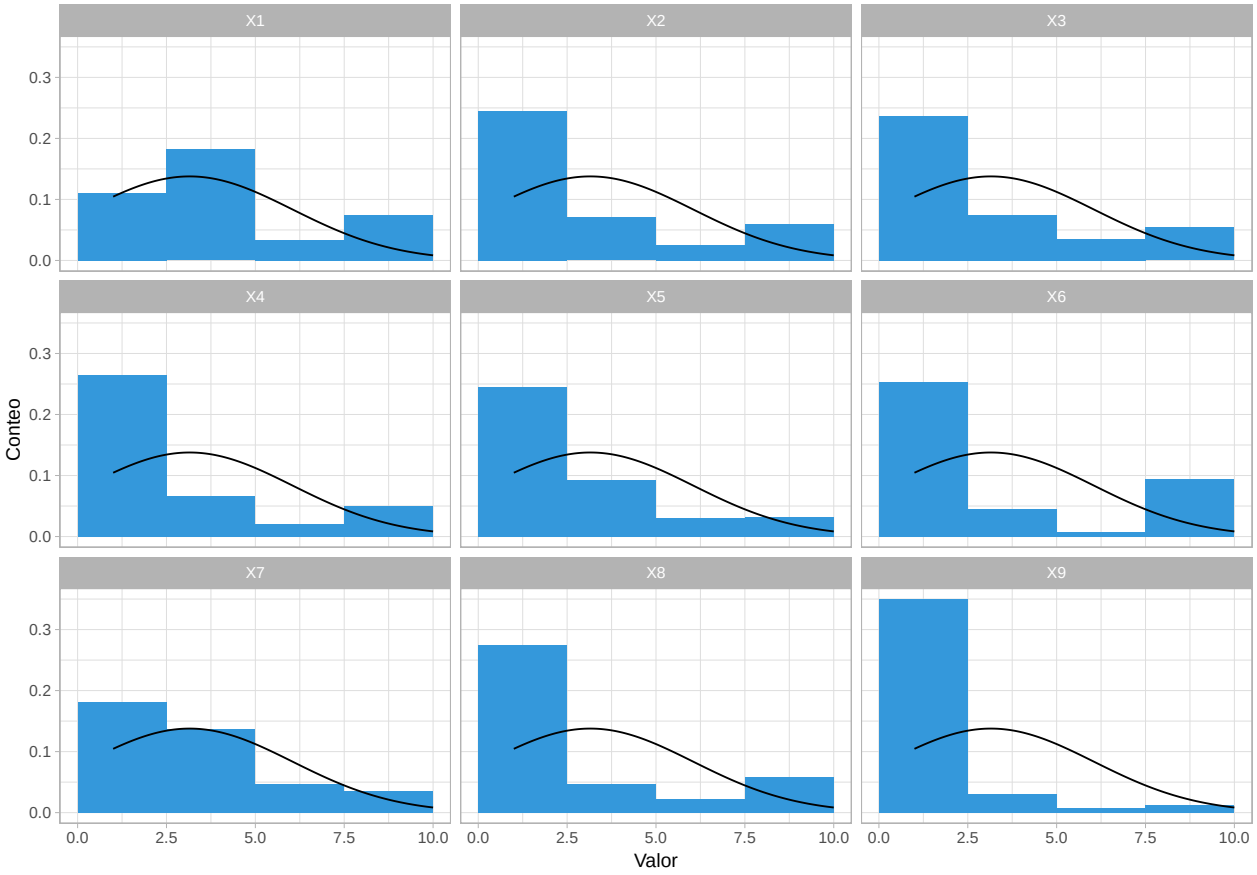


Figure 8: Histogramas para cada variable del dataset de clasificación.

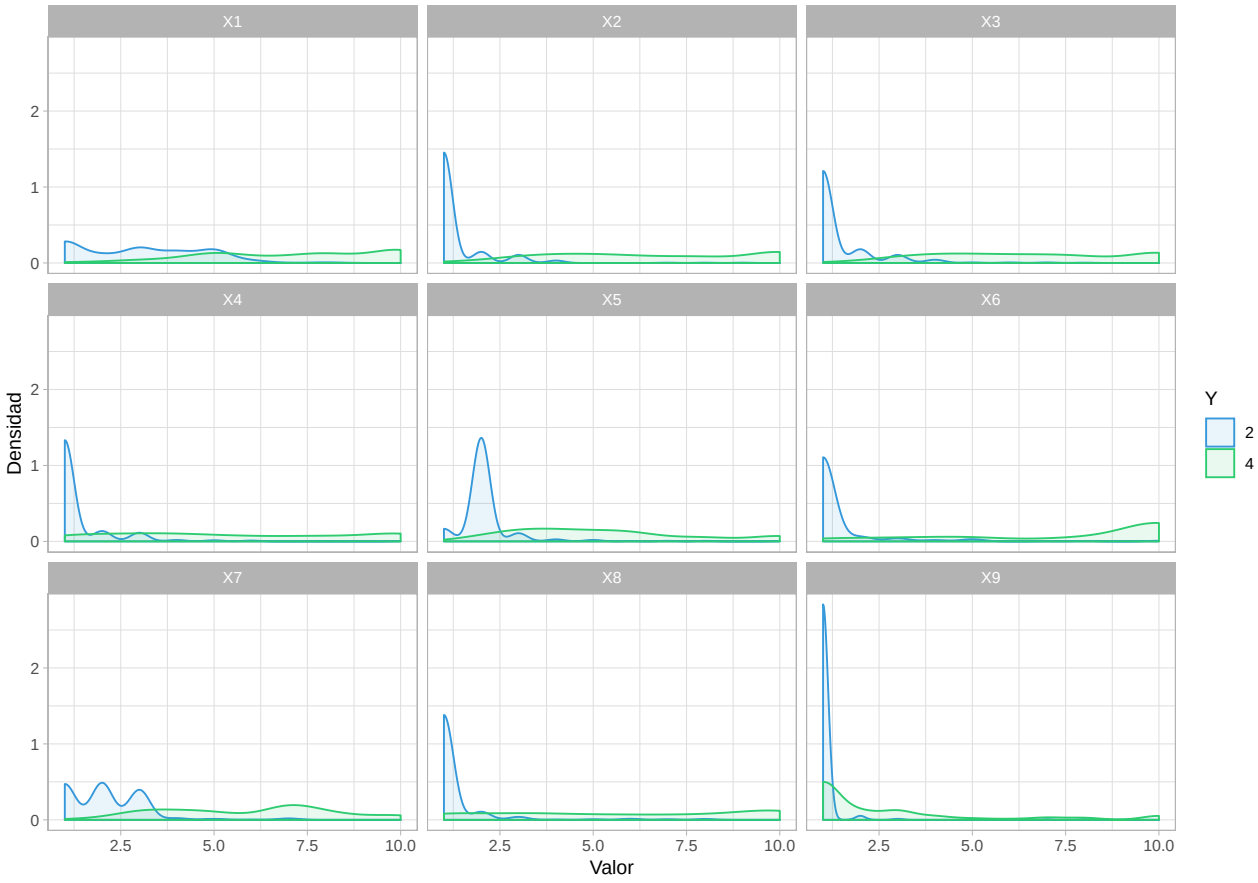


Figure 9: Gráfico de densidad para cada variable del dataset de clasificación.

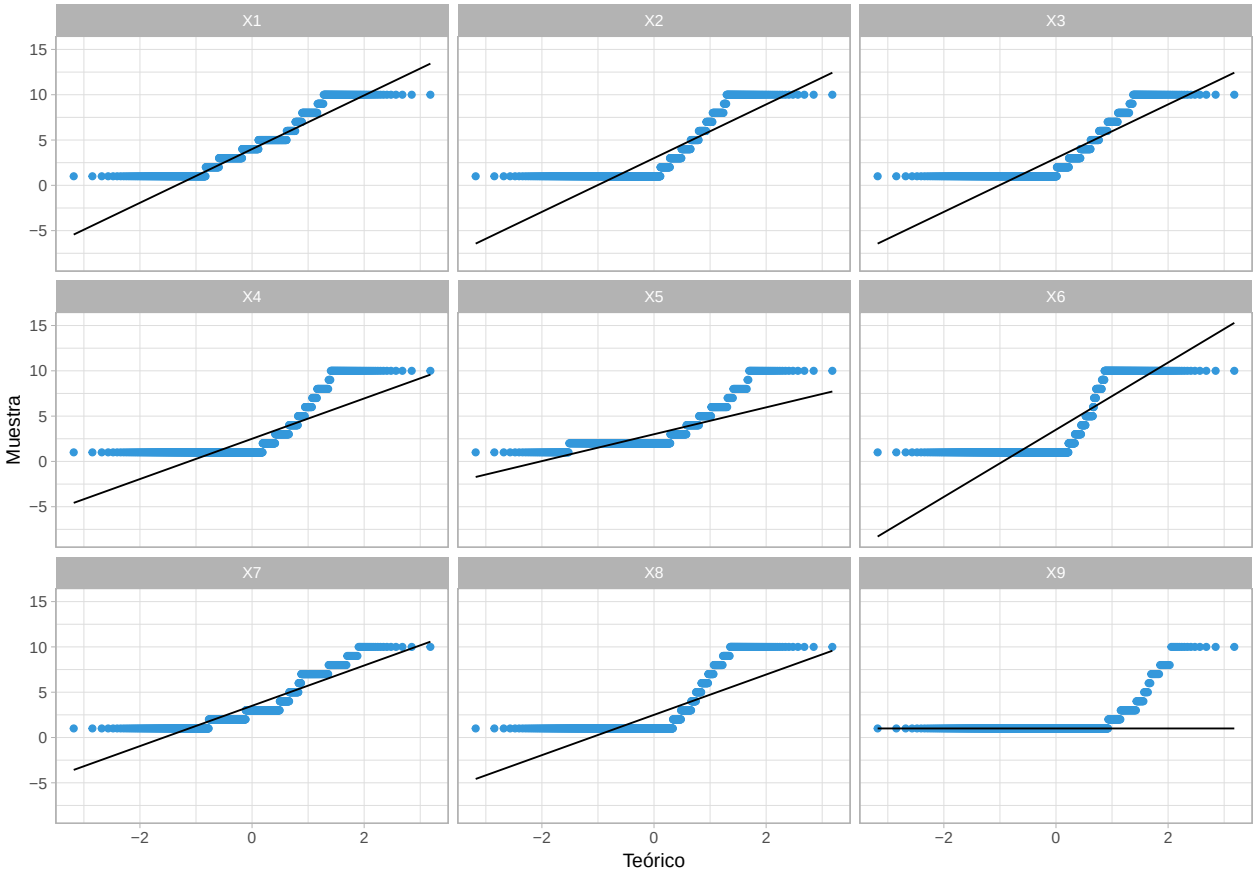


Figure 10: q-q plots para las variables del dataset de clasificación

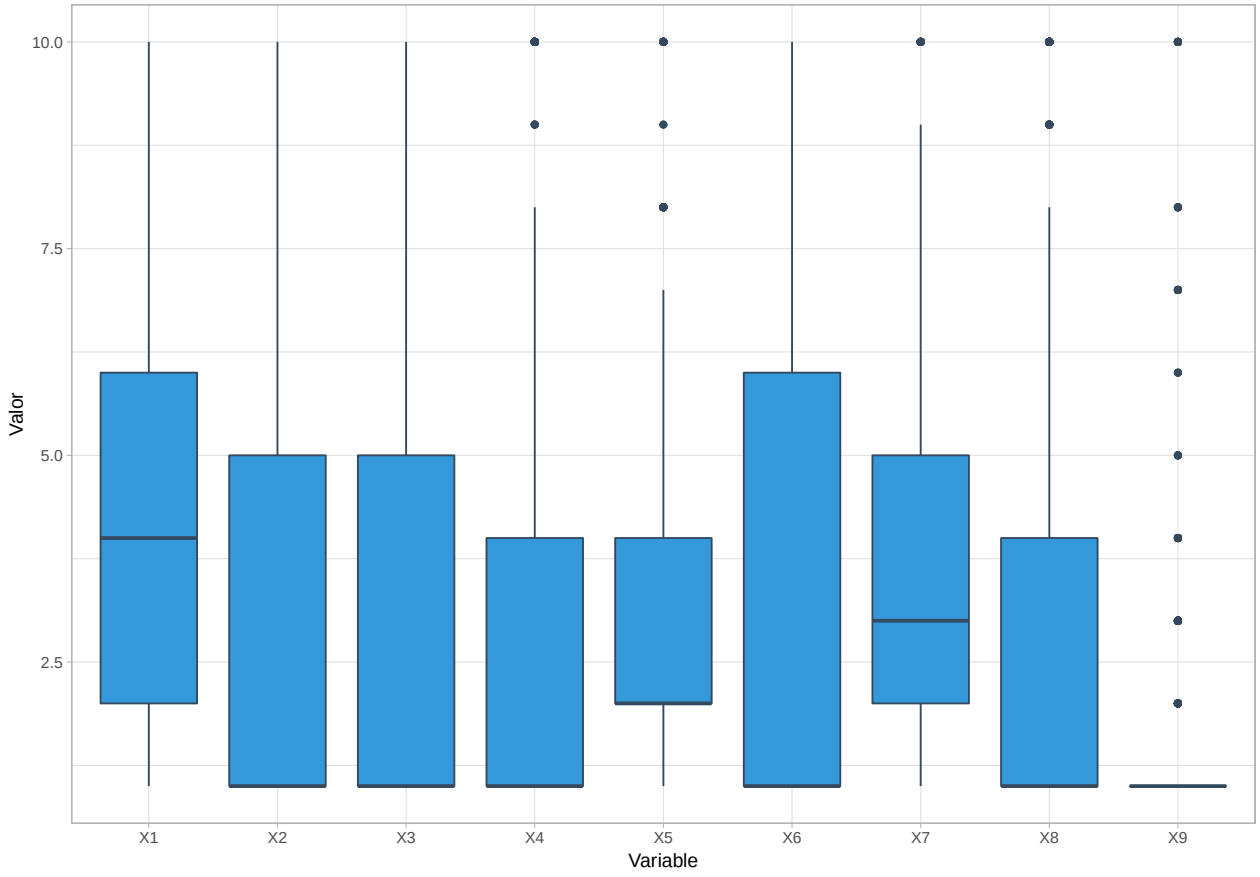


Figure 11: Boxplots para las variables del dataset de clasificación

Figure 12: Gráfico de correlación entre las variables del dataset de clasificación

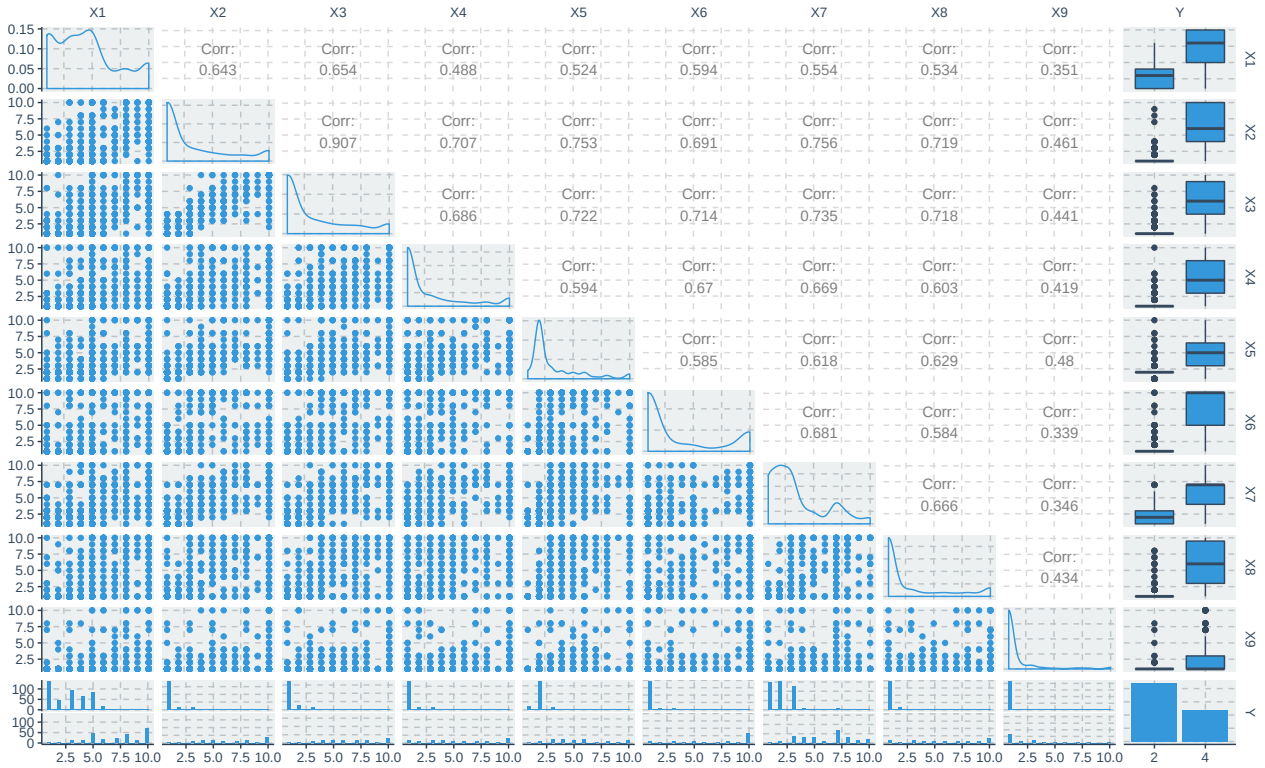


Figure 13: Gráfico con la función ggpairs para explorar posibles relaciones entre las variables

Preprocesamiento dataset de Clasificación

Luego del análisis exploratorio de los datos, voy a proceder a realizar un tratamiento de los mismos, para luego avanzar con los modelos de clasificación.

Asigno la variable respuesta a tipo factor

```
datasetC$Y <- as.factor(datasetC$Y)
```

- Remoción de variables con correlación. Defino un threshold de 0.9, remuevo variables con una correlación mayor, dejando la de menor media con respecto al resto de variables.

```
highCor <- colnames(datasetC)[findCorrelation(datasetC.cor, cutoff = 0.9, verbose = TRUE)]
```

```
## Compare row 2  and column  3 with corr  0.907
##   Means:  0.705 vs 0.59 so flagging column 2
## All correlations <= 0.9
```

```
datasetC <- datasetC[, which(!colnames(datasetC) %in% highCor)]
```

En este caso X2 y X3 tienen una correlación de 0.91, se elimina X2 ya que la suma de correlaciones es 6.63, y la de X3 6.57.

- Transformación de outliers: En los gráficos boxplot se pudo observar la presencia de outliers para las variables X4, X5, X7, X8 y principalmente X9. Inicialmente chequeo la distribución de valores con respecto a la variable dependiente, para elegir los rangos adecuados.

```
table(datasetC$X9, datasetC$Y)
```

```
##
```

```
##        2   4
##   1  430 132
##   2    8  27
##   3    2  31
##   4    0  12
##   5    1   5
##   6    0   3
##   7    1   8
##   8    1   7
##   10   0  14
```

```
table(datasetC$X5, datasetC$Y)
```

```
##
##        2   4
##   1   43   1
##   2  354  21
##   3   28  43
##   4    7  41
##   5    5  34
##   6    1  39
##   7    2   9
##   8    2  19
##   9    0   2
##   10   1  30
```

```
table(datasetC$X4, datasetC$Y)
```

```
##
##        2   4
##   1  362  30
##   2   37  21
##   3   31  27
##   4    5  28
##   5    4  19
##   6    3  18
##   7    0  13
##   8    0  25
##   9    0   4
##   10   1  54
```

```
table(datasetC$X7, datasetC$Y)
```

```
##
##        2   4
##   1  148   2
##   2  153   7
##   3  124  36
##   4    7  32
##   5    4  30
##   6    1   8
##   7    6  65
##   8    0  28
##   9    0  11
##   10   0  20
```

```r
table(datasetC$X8, datasetC$Y)
```

```
##
##        2   4
##   1  390  41
##   2   30   6
##   3   11  31
##   4    1  17
##   5    2  17
##   6    4  18
##   7    2  14
##   8    3  20
##   9    0  15
##   10   0  60
```

Binning de variables, para eliminar outliers.

```r
datasetC$X9 <- as.integer(as.character(cut(datasetC$X9,
                                           breaks = c(0,1,10), labels =c(1, 2))))
datasetC$X5 <- as.integer(as.character(cut(datasetC$X5,
                                           breaks = c(0,2,10), labels =c(1, 2))))
datasetC$X4 <- as.integer(as.character(cut(datasetC$X4,
                                           breaks = c(0,1,3,10), labels =c(1, 2, 3))))
datasetC$X7 <- as.integer(as.character(cut(datasetC$X7,
                                           breaks = c(0,2,3,10), labels =c(1, 2, 3))))
datasetC$X8 <- as.integer(as.character(cut(datasetC$X8,
                                           breaks = c(0,2,10), labels =c(1, 2))))
```

- Normalización: En este punto tengo dos sets de datos, al que le realize el pre-procesamiento (remoción de variable con alta correlación y binning), y el original. Voy a normalizar ambos, luego utilizaré cross-validation para determinar si el pre-procesamiento mejora los modelos o no.

```r
# Dataset preprocesado
Y_values = datasetC$Y
datasetC <- as.data.frame(scale(datasetC[,-ncol(datasetC)], center = TRUE, scale = TRUE))
datasetC$Y <- Y_values

# Dataset sin preprocesar
datasetC_sin_procesar <- as.data.frame(scale(
  datasetC_sin_procesar[,-ncol(datasetC_sin_procesar)],
  center = TRUE, scale = TRUE))
datasetC_sin_procesar$Y <- Y_values
```

Apartado Regresión

Debido a la naturaleza del dataset, con la presencia de varias variables categóricas, la variable mas factible de estar relacionada es X6, con un valor de correlación de -0.655. Voy a preseleccionar y probar 5 variables.

Probare todas excepto X4 y X5, ya que luego del análisis exploratorio de datos, teniendo en cuenta los valores de correlación, son las que aparentan ser menos significativas para realizar un modelo.

Comenzaré un proceso de ***forward selection*** creando modelos lineales simples. A través del análisis del estadístico F y el estadístico t, determinaré si una variable tiene relacion lineal con la variable respuesta. A través del coeficiente de determinación R^2 (r cuadrado) sabré que porcentaje de la varianza de la variable respuesta esta explicada por una variable independiente. La raíz del error cuadrático medio (RMSE) sera utilizada para calcular la diferencia entre los valores predichos y los valores reales observados.

Construcción de modelos lineales simples

```
# Variables para almacenar los resultados y luego comparar con KNN
modelos_comparar <- c()
metodos_comparar <- c()
RMSE_comparar <- c()

# Construyo un modelo de regresión para la variable X1
fit1 <- lm(Y~X1, datasetR)
# Obtengo información del modelo
fit1
```

```
##
## Call:
## lm(formula = Y ~ X1, data = datasetR)
##
## Coefficients:
## (Intercept)           X1
##     2.03625     -0.02404
```

```
summary(fit1)
```

```
##
## Call:
## lm(formula = Y ~ X1, data = datasetR)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.03625  0.04375  0.26375  0.26375  0.52821
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.036251   0.008754 232.604   <2e-16 ***
## X1          -0.024042   0.013392  -1.795   0.0727 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5494 on 4049 degrees of freedom
## Multiple R-squared:  0.0007953,  Adjusted R-squared:  0.0005486
## F-statistic: 3.223 on 1 and 4049 DF,  p-value: 0.07269
```

```r
# Cálculo manual del RMSE
RMSEfit1 <- sqrt(sum(fit1$residuals^2)/(length(fit1$residuals)-2))
```

Repito para el resto de las variables

```r
fit2 <- lm(Y~X2, datasetR)
fit2
```

```
##
## Call:
## lm(formula = Y ~ X2, data = datasetR)
##
## Coefficients:
## (Intercept)          X21
##     1.98472      0.09414
```

```r
summary(fit2)
```

```
##
## Call:
## lm(formula = Y ~ X2, data = datasetR)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.07886  0.00114  0.22114  0.31528  0.31528
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.98472    0.01241 159.884  < 2e-16 ***
## X21          0.09414    0.01722   5.467 4.86e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5476 on 4049 degrees of freedom
## Multiple R-squared:  0.007326,   Adjusted R-squared:  0.007081
## F-statistic: 29.88 on 1 and 4049 DF,  p-value: 4.864e-08
```

```r
RMSEfit2 <- sqrt(sum(fit2$residuals^2)/(length(fit2$residuals)-2))
```

```r
fit3 <- lm(Y~X3, datasetR)
fit3
```

```
##
## Call:
## lm(formula = Y ~ X3, data = datasetR)
##
## Coefficients:
## (Intercept)          X31
##     2.02586      0.09931
```

```r
summary(fit3)
```

```
##
## Call:
## lm(formula = Y ~ X3, data = datasetR)
##
## Residuals:
```

```
##      Min       1Q   Median       3Q      Max
## -2.12517  0.05414  0.27414  0.27414  0.27414
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 2.025862   0.008984 225.501   <2e-16 ***
## X31         0.099311   0.032115   3.092    0.002 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.549 on 4049 degrees of freedom
## Multiple R-squared:  0.002356,   Adjusted R-squared:  0.00211
## F-statistic: 9.562 on 1 and 4049 DF,  p-value: 0.001999
```

```
RMSEfit3 <- sqrt(sum(fit3$residuals^2)/(length(fit3$residuals)-2))
```

```
fit6 <- lm(Y~X6, datasetR)
fit6
```

```
##
## Call:
## lm(formula = Y ~ X6, data = datasetR)
##
## Coefficients:
## (Intercept)           X6
##    74.15916     -0.03657
```

```
summary(fit6)
```

```
##
## Call:
## lm(formula = Y ~ X6, data = datasetR)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.53939 -0.18537  0.07061  0.29002  0.50943
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 74.1591602  1.3066706   56.75   <2e-16 ***
## X6          -0.0365683  0.0006625  -55.20   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4152 on 4049 degrees of freedom
## Multiple R-squared:  0.4294, Adjusted R-squared:  0.4292
## F-statistic:  3047 on 1 and 4049 DF,  p-value: < 2.2e-16
```

```
RMSEfit6 <- sqrt(sum(fit6$residuals^2)/(length(fit6$residuals)-2))
```

```
fit7 <- lm(Y~X7, datasetR)
fit7
```

```
##
## Call:
## lm(formula = Y ~ X7, data = datasetR)
##
```

```
## Coefficients:
## (Intercept)          X71
##      2.0634      -0.1321
```

```
summary(fit7)
```

```
##
## Call:
## lm(formula = Y ~ X7, data = datasetR)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.0634  0.0166  0.2366  0.2366  0.3687
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.063397   0.009762 211.373  < 2e-16 ***
## X71         -0.132061   0.020563  -6.422  1.5e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5468 on 4049 degrees of freedom
## Multiple R-squared:  0.01008,    Adjusted R-squared:  0.00984
## F-statistic: 41.25 on 1 and 4049 DF,  p-value: 1.497e-10
```

```
RMSEfit7 <- sqrt(sum(fit7$residuals^2)/(length(fit7$residuals)-2))
```

Genero una tabla de resultados

```
# Genero un vector de variables pre seleccionadas
variables_elegidas <- c("X1", "X2", "X3", "X6", "X7")

# Genero un vector con los p-valores del estadístico t para cada modelo
p_valores_elegidas <- c(summary(fit1)$coefficients[,4][2],
               summary(fit2)$coefficients[,4][2],
               summary(fit3)$coefficients[,4][2],
               summary(fit6)$coefficients[,4][2],
               summary(fit7)$coefficients[,4][2]
               )

# Genero un vector con los valores de R2 para cada modelo
r2_elegidas <- c(summary(fit1)$r.squared,
               summary(fit2)$r.squared,
               summary(fit3)$r.squared,
               summary(fit6)$r.squared,
               summary(fit7)$r.squared
               )

rmse_elegidas <- c(RMSEfit1,
                   RMSEfit2,
                   RMSEfit3,
                   RMSEfit6,
                   RMSEfit7
                   )

# Armo el dataframe
```

Table 1: P-valor del estadístico t, R2 y RMSE para las 5 variables pre-seleccionadas.

Variable	Pval	R2	RMSE
X1	0.0726898	0.0007953	0.5493996
X2	0.0000000	0.0073264	0.5476012
X3	0.0019995	0.0023561	0.5489704
X6	0.0000000	0.4293879	0.4151754
X7	0.0000000	0.0100842	0.5468400

```
df_elegidas <- data.frame(Variable = variables_elegidas,
                          Pval = p_valores_elegidas,
                          R2 = r2_elegidas,
                          RMSE = rmse_elegidas)

# Sin nombre de filas
rownames(df_elegidas) <- c()

# Muestro la tabla
kable(df_elegidas,
      format = "latex",
      booktabs = TRUE,
      caption="\\label{tab:tab0}P-valor del estadístico t, R2 y RMSE
      para las 5 variables pre-seleccionadas.")
```

Selección del mejor modelo lineal simple

El estadístico t, testea la hipótesis nula de que no hay asociación entre la variable predictora y respuesta. Existen 4 variables (X6, X7, X2 y X3) que poseen un p-valor significativo, por lo tanto se rechaza la hipótesis nula de que las variables no estan asociadas, y aceptamos la hipótesis alternativa de que hay una relación entre la variable predictora y respuesta (**Tabla 1**). La variable X1 no fue significativa en el test de hipótesis con un alfa = 0.05.

De los 5 modelos pre-seleccionados el que tiene un menor RMSE es el que contempla a la variable X6, RMSE=0.4152. Su R2 adjustado es de 0.429, es decir que explica el 42,9% de la variabilidad de la variable dependiente Y, y su p-valor significativo para el estadístico t, indica que la relación no es al azar (ver **Tabla 1**).

Visualización del modelo para X6.

Ver Figura. 14

```
plot(Y~X6, datasetR) # Gráfico de dispersion Y en función de X6
# Añado la línea de regresión
abline(fit6, col="red")
```

```
confint(fit6)
```

```
##                 2.5 %      97.5 %
## (Intercept) 71.5973671 76.72095334
## X6          -0.0378671 -0.03526943
```

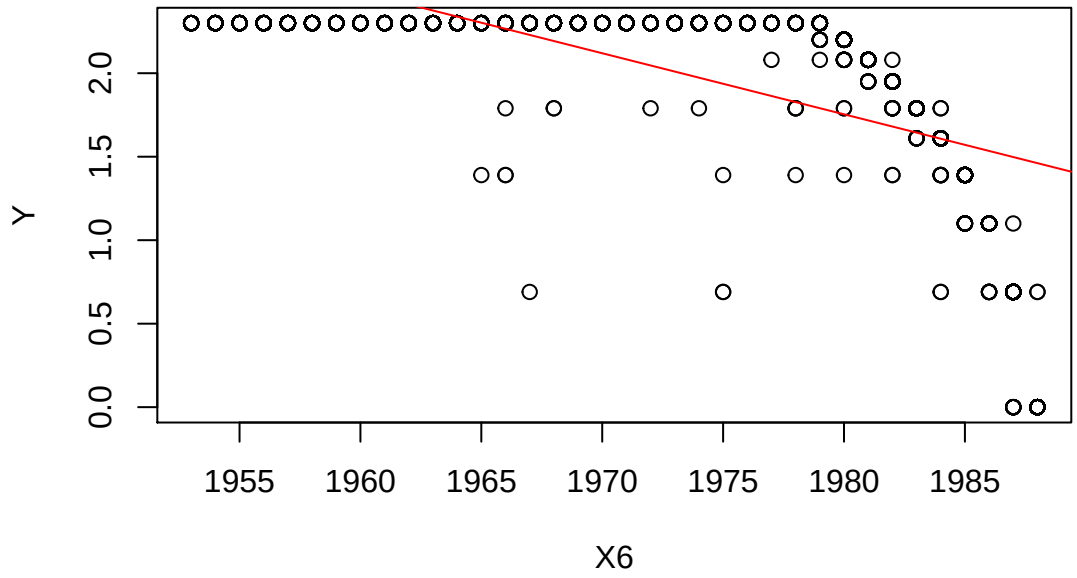


Figure 14: Visualización del modelo para X6.

```
# Guardo la información para comparar
modelos_comparar <- c(modelos_comparar, ("X6"))
metodos_comparar <- c(metodos_comparar, ("LM"))
RMSE_comparar <- c(RMSE_comparar, RMSEfit6)
```

Construcción de modelos lineales múltiples

Comenzaré con un modelo que contemple todas las variables, y luego realizaré ***backward selection***, el objetivo es determinar si se puede obtener un modelo más preciso incorporando mayor complejidad al mismo.

```
# Construyo un modelo lineal múltiple con todas las varibles y muestro
# información del mismo
fitmul1 <- lm(Y~., datasetR)
fitmul1
```

```
##
## Call:
## lm(formula = Y ~ ., data = datasetR)
##
## Coefficients:
## (Intercept)           X1          X21          X31          X41
##   75.133210     0.020157    -0.060054     0.138595    -0.010443
##         X51           X6          X71
##   -0.001883    -0.037047    -0.050028
```

```
summary(fitmul1)
```

```
##
## Call:
## lm(formula = Y ~ ., data = datasetR)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -1.57269 -0.20355  0.07478  0.29706  0.59426
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 75.1332095  1.3357916  56.246  < 2e-16 ***
## X1           0.0201566  0.0101060   1.995  0.04616 *
## X21         -0.0600539  0.0138989  -4.321 1.59e-05 ***
## X31          0.1385946  0.0252109   5.497 4.09e-08 ***
## X41         -0.0104435  0.0435839  -0.240  0.81064
## X51         -0.0018829  0.0139347  -0.135  0.89252
## X6          -0.0370467  0.0006769 -54.731  < 2e-16 ***
## X71         -0.0500277  0.0156296  -3.201  0.00138 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4128 on 4043 degrees of freedom
## Multiple R-squared:  0.4367, Adjusted R-squared:  0.4358
## F-statistic: 447.8 on 7 and 4043 DF,  p-value: < 2.2e-16
```

```
RMSEfitmul1 <- sqrt(sum(fitmul1$residuals^2)/(length(fitmul1$residuals)-2))
```

El estadístico F con un valor de 447 nos indica que al menos 1 de las variables independientes tiene una relación lineal con la variable dependiente, esto ya lo sabíamos de los modelos lineales simples.

A través de la inspección de los p-valores individuales para el estadístico t de cada variable independiente, podemos observar que las 5 variables seleccionadas en el paso anterior (regresiones lineales simples) son significativas con un alfa = 0.05. En este caso X1, también esta por debajo del alfa. El R2 ajustado del modelo lineal mútiple es 0.4358, es decir que el 43,6% de la variabilidad de la variable dependiente se explica con este modelo.

Como parte del proceso de backward selection comenzaré a eliminar variables del modelo y analizar el resultado. Iré eliminando la variable con el p-valor más alto. Consideraré dos cosas para continuar con este proceso, a) eliminar todas las variables cuyos p-valores no sean significativos, o b) que se produzca una perdida significativa en el R2.

Inicialmente construiré un modelo con todas las variables excepto X5.

```
fitmul2 <- lm(Y~.-X5, datasetR)
fitmul2
```

```
##
## Call:
## lm(formula = Y ~ . - X5, data = datasetR)
##
## Coefficients:
## (Intercept)           X1          X21          X31          X41
##    75.13948      0.02020     -0.06036      0.13880     -0.01028
##          X6          X71
##    -0.03705     -0.04992
```

```
summary(fitmul2)
```

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

```
## -1.57199 -0.20486  0.07528  0.29758  0.59312
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 75.1394815  1.3348229  56.292  < 2e-16 ***
## X1           0.0201973  0.0101003   2.000  0.04560 *
## X21         -0.0603649  0.0137054  -4.404 1.09e-05 ***
## X31          0.1387961  0.0251636   5.516 3.69e-08 ***
## X41         -0.0102807  0.0435620  -0.236  0.81344
## X6          -0.0370502  0.0006763 -54.782  < 2e-16 ***
## X71         -0.0499234  0.0156087  -3.198  0.00139 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4127 on 4044 degrees of freedom
## Multiple R-squared:  0.4367, Adjusted R-squared:  0.4359
## F-statistic: 522.6 on 6 and 4044 DF,  p-value: < 2.2e-16
```

```
RMSEfitmul2 <- sqrt(sum(fitmul2$residuals^2)/(length(fitmul2$residuals)-2))
```

- El valor de R2 practicamente no se modifico. Ahora eliminare también X4, ya que no es significativa.

```
fitmul3 <- lm(Y~.-X5-X4, datasetR)
fitmul3
```

```
##
## Call:
## lm(formula = Y ~ . - X5 - X4, data = datasetR)
##
## Coefficients:
## (Intercept)           X1          X21          X31           X6
##    75.13599      0.02021     -0.06045      0.13828     -0.03705
##         X71
##    -0.04978
```

```
summary(fitmul3)
```

```
##
## Call:
## lm(formula = Y ~ . - X5 - X4, data = datasetR)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.57174 -0.20470  0.07547  0.29776  0.59324
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 75.1359918  1.3345852  56.299  < 2e-16 ***
## X1           0.0202063  0.0100990   2.001  0.04548 *
## X21         -0.0604548  0.0136985  -4.413 1.04e-05 ***
## X31          0.1382836  0.0250669   5.517 3.67e-08 ***
## X6          -0.0370485  0.0006762 -54.789  < 2e-16 ***
## X71         -0.0497793  0.0155949  -3.192  0.00142 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.4127 on 4045 degrees of freedom
## Multiple R-squared:  0.4367, Adjusted R-squared:  0.436
## F-statistic: 627.3 on 5 and 4045 DF,  p-value: < 2.2e-16
```

```
RMSEfitmul3 <- sqrt(sum(fitmul3$residuals^2)/(length(fitmul3$residuals)-2))
```

Todas las variables del modelo son significativas para el estadístico t.

Finalizo el proceso de backward selection, ya que llegue al criterio de que todas las variables predictoras del modelo sean significativas.

```
# Agrego la información al dataframe de comparaciones
modelos_comparar <- c(modelos_comparar, ".-X5-X4")
metodos_comparar <- c(metodos_comparar, ("LM"))
RMSE_comparar <- c(RMSE_comparar, RMSEfitmul3)
```

Conclusiones parciales

Un modelo lineal múltiple que contempla a las variables X1, X2, X3, X6 y X7 explica casi la misma variabilidad de la variable dependiente que un modelo lineal simple que solo contempla a la variable X6, 43.6% vs 42.9% La diferencia es la interpretabilidad de ambos modelos. El modelo que solo contempla a X6 es más facil de interpretar que el modelo que contempla a 5 variables. El RMSE de ambos modelos es 0.41.

Construcción de modelos considerando interacciones y no linealidad

Probare modelos no lineales e interacciones, para tratar de mejorar el modelo. A partir de los resultados anteriores, la intuición indica que lo más interesante a probar son modelos no lineales que trabajen sobre la variable X6.

```
# Construyo un modelo no linea y muestro información
fitinl1 <- lm(Y~ X6 + I(sqrt(X6)), datasetR)
fitinl1
```

```
##
## Call:
## lm(formula = Y ~ X6 + I(sqrt(X6)), data = datasetR)
##
## Coefficients:
## (Intercept)           X6  I(sqrt(X6))
##   -55591.91       -28.27      2507.41
```

```
summary(fitinl1)
```

```
##
## Call:
## lm(formula = Y ~ X6 + I(sqrt(X6)), data = datasetR)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.81680 -0.16665 -0.01119  0.19507  0.44376
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.559e+04  7.060e+02  -78.74   <2e-16 ***
## X6          -2.827e+01  3.581e-01  -78.94   <2e-16 ***
## I(sqrt(X6))  2.507e+03  3.180e+01   78.84   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2608 on 4048 degrees of freedom
## Multiple R-squared:  0.775,  Adjusted R-squared:  0.7748
## F-statistic:  6970 on 2 and 4048 DF,  p-value: < 2.2e-16
```

```
# Repito el proceso para distintos modelos no lineales y/o con interacciones
fitinl2 <- lm(Y~ X6 + I(log(X6)), datasetR)
fitinl2
```

```
##
## Call:
## lm(formula = Y ~ X6 + I(log(X6)), data = datasetR)
##
## Coefficients:
## (Intercept)           X6   I(log(X6))
##  -183146.51       -14.15     27817.48
```

```
summary(fitinl2)
```

```
##
## Call:
## lm(formula = Y ~ X6 + I(log(X6)), data = datasetR)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.81688 -0.16691 -0.01143  0.19544  0.44394
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.831e+05  2.327e+03  -78.71   <2e-16 ***
## X6          -1.415e+01  1.792e-01  -78.95   <2e-16 ***
## I(log(X6))   2.782e+04  3.533e+02   78.74   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.261 on 4048 degrees of freedom
## Multiple R-squared:  0.7746, Adjusted R-squared:  0.7745
## F-statistic:  6956 on 2 and 4048 DF,  p-value: < 2.2e-16
```

```
fitinl3 <- lm(Y~ X6 + I(X6^2), datasetR)
fitinl3
```

```
##
## Call:
## lm(formula = Y ~ X6 + I(X6^2), data = datasetR)
##
## Coefficients:
## (Intercept)           X6      I(X6^2)
##  -1.387e+04    1.411e+01   -3.586e-03
```

```
summary(fitinl3)
```

```
##
## Call:
## lm(formula = Y ~ X6 + I(X6^2), data = datasetR)
```

```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.81653 -0.16589 -0.01049  0.19398  0.44322
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.387e+04  1.761e+02  -78.72   <2e-16 ***
## X6           1.411e+01  1.787e-01   78.94   <2e-16 ***
## I(X6^2)     -3.586e-03  4.532e-05  -79.14   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2602 on 4048 degrees of freedom
## Multiple R-squared:  0.776,  Adjusted R-squared:  0.7759
## F-statistic:  7012 on 2 and 4048 DF,  p-value: < 2.2e-16
```

```
fitinl4 <- lm(Y~ X6 + I(X6^2) + X1 + I(X1^2) + X6*X1, datasetR)
fitinl4
```

```
##
## Call:
## lm(formula = Y ~ X6 + I(X6^2) + X1 + I(X1^2) + X6 * X1, data = datasetR)
##
## Coefficients:
## (Intercept)           X6      I(X6^2)           X1      I(X1^2)
##  -1.391e+04    1.415e+01   -3.598e-03   -1.233e+01    1.062e-03
##       X6:X1
##   6.233e-03
```

```
summary(fitinl4)
```

```
##
## Call:
## lm(formula = Y ~ X6 + I(X6^2) + X1 + I(X1^2) + X6 * X1, data = datasetR)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.63509 -0.16977 -0.01498  0.19506  1.17616
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.391e+04  1.753e+02 -79.347  < 2e-16 ***
## X6           1.415e+01  1.778e-01  79.567  < 2e-16 ***
## I(X6^2)     -3.598e-03  4.510e-05 -79.775  < 2e-16 ***
## X1          -1.233e+01  1.752e+00  -7.034 2.35e-12 ***
## I(X1^2)      1.062e-03  2.461e-03   0.432    0.666
## X6:X1        6.233e-03  8.873e-04   7.025 2.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2586 on 4045 degrees of freedom
## Multiple R-squared:  0.7788, Adjusted R-squared:  0.7786
## F-statistic:  2849 on 5 and 4045 DF,  p-value: < 2.2e-16
```

```r
fitinl5 <- lm(Y~ X6 + X3 + X6*X3 + I(X6^2), datasetR)
fitinl5
```

```
##
## Call:
## lm(formula = Y ~ X6 + X3 + X6 * X3 + I(X6^2), data = datasetR)
##
## Coefficients:
## (Intercept)           X6          X31      I(X6^2)       X6:X31
##  -1.391e+04    1.415e+01   -1.318e+01   -3.598e-03    6.671e-03
```

```r
summary(fitinl5)
```

```
##
## Call:
## lm(formula = Y ~ X6 + X3 + X6 * X3 + I(X6^2), data = datasetR)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.82070 -0.16504 -0.01585  0.20651  0.48149
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.391e+04  1.769e+02 -78.623  < 2e-16 ***
## X6           1.415e+01  1.795e-01  78.839  < 2e-16 ***
## X31         -1.318e+01  3.674e+00  -3.586 0.000340 ***
## I(X6^2)     -3.598e-03  4.552e-05 -79.044  < 2e-16 ***
## X6:X31       6.671e-03  1.863e-03   3.581 0.000346 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2598 on 4046 degrees of freedom
## Multiple R-squared:  0.7768, Adjusted R-squared:  0.7766
## F-statistic:  3520 on 4 and 4046 DF,  p-value: < 2.2e-16
```

```r
fitinl6 <- lm(Y~ X6 + I(X6^2) + I(X6^3), datasetR)
fitinl6
```

```
##
## Call:
## lm(formula = Y ~ X6 + I(X6^2) + I(X6^3), data = datasetR)
##
## Coefficients:
## (Intercept)           X6      I(X6^2)      I(X6^3)
##   2.070e+06   -3.158e+03    1.606e+00   -2.721e-04
```

```r
summary(fitinl6)
```

```
##
## Call:
## lm(formula = Y ~ X6 + I(X6^2) + I(X6^3), data = datasetR)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.70980 -0.08513  0.02722  0.11643  0.40083
##
```

```
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.070e+06  2.167e+04   95.54   <2e-16 ***
## X6          -3.158e+03  3.298e+01  -95.75   <2e-16 ***
## I(X6^2)      1.606e+00  1.673e-02   95.97   <2e-16 ***
## I(X6^3)     -2.721e-04  2.829e-06  -96.18   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1435 on 4047 degrees of freedom
## Multiple R-squared:  0.9318, Adjusted R-squared:  0.9318
## F-statistic: 1.844e+04 on 3 and 4047 DF,  p-value: < 2.2e-16
```

```
fitinl6b <- lm(Y~ X6 + I(X6^2) + I(X6^3) + I(X6^4), datasetR)
fitinl6b
```

```
##
## Call:
## lm(formula = Y ~ X6 + I(X6^2) + I(X6^3) + I(X6^4), data = datasetR)
##
## Coefficients:
## (Intercept)           X6      I(X6^2)      I(X6^3)      I(X6^4)
##   2.070e+06   -3.158e+03    1.606e+00   -2.721e-04           NA
```

```
summary(fitinl6b)
```

```
##
## Call:
## lm(formula = Y ~ X6 + I(X6^2) + I(X6^3) + I(X6^4), data = datasetR)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.70980 -0.08513  0.02722  0.11643  0.40083
##
## Coefficients: (1 not defined because of singularities)
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.070e+06  2.167e+04   95.54   <2e-16 ***
## X6          -3.158e+03  3.298e+01  -95.75   <2e-16 ***
## I(X6^2)      1.606e+00  1.673e-02   95.97   <2e-16 ***
## I(X6^3)     -2.721e-04  2.829e-06  -96.18   <2e-16 ***
## I(X6^4)             NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1435 on 4047 degrees of freedom
## Multiple R-squared:  0.9318, Adjusted R-squared:  0.9318
## F-statistic: 1.844e+04 on 3 and 4047 DF,  p-value: < 2.2e-16
```

R-2. Conclusiones parciales

Luego de probar modelos con interacciones y terminos no lineales, llego a la conclusion que la mayor ganancia en precisión en la etapa de modelado, se da al incorporar términos no lineales para la variable X6. El R2 de un modelo con un termino cuadrático para X6 se incrementa a 0.77 con un RMSE de 0.26. El R2 trepa hasta 0.93 con la incoporación de un termino cúbico y el RMSE baja a 0.14. No se produce una mejora en el R2 al agregar un termino de grado 4 para X6.

Calculo el RMSE para los dos modelos que explican la mayor variabilidad de acuerdo al R2.

```
# Guardo la información para comparar
modelos_comparar <- c(modelos_comparar, "X6+X6^2", "X6+X6^2+X6^3")
metodos_comparar <- c(metodos_comparar, c("LM", "LM"))
RMSEfitinl3 <- sqrt(sum(fitinl3$residuals^2)/(length(fitinl3$residuals)-2))
RMSEfitinl6 <- sqrt(sum(fitinl6$residuals^2)/(length(fitinl6$residuals)-2))
RMSE_comparar <- c(RMSE_comparar, RMSEfitinl3, RMSEfitinl6)
```

Conclusiones finales regresiones

Luego de realizar el análisis con regresión lineal simple, múltiple y no lineal + interacciones, selecciono 4 modelos:

- Y ~ X6: Modelo lineal simple que explica el 42,9% de la variabilidad de Y.
- Y ~ . -X5 -X4: Modelo lineal múltiple, es un poco más preciso y explica el 43,6% de la variabilidad de Y. Sin embargo es más complejo que el modelo que solo contempla a X6, y no genera una ganancia en precisión.
- Y ~ X6 + I(X6^2) y Y ~ X6 + I(X6^2) + I(X6^3): Con el agregado de termino polinómicos mejoramos la precision, disminuyendo el RMSE y mejorando la variabilidad explicada. Selecciono estos dos modelos para analizar si el modelo con el termino de grado 3, no sobreajustada demasiado con respecto al modelo de grado 2.

k-NN

Para comenzar con k-NN voy a probar los modelos más interesantes obtenidos con regresiones lineales y no lineales, pero utilizando k-NN, y compararé la raíz del error cuadrático medio (RMSE) del mismo modelo con los dos métodos.

```
fitknn1 <- kknn(Y ~ X6, datasetR, datasetR) # Construyo el modelo
fitknn1 # Printeo informacion del modelo
```

```
##
## Call:
## kknn(formula = Y ~ X6, train = datasetR, test = datasetR)
##
## Response: "continuous"
```

```
yprime <- fitknn1$fitted.values # Obtengo los valores predichos por el modelo
RMSEknn1 <- sqrt(sum((datasetR$Y-yprime)^2)/length(yprime)) # calculo del RMSE
RMSEknn1 # Muestro el RMSE
```

```
## [1] 0.0823474
```

```
# Almaceno la información para comparar
modelos_comparar <- c(modelos_comparar, "X6")
metodos_comparar <- c(metodos_comparar, "k-NN")
RMSE_comparar <- c(RMSE_comparar, RMSEknn1)

# Repito el proceso para el resto de modelos seleccionados con regresiones
fitknn2 <- kknn(Y ~ . -X5 -X4, datasetR, datasetR)
fitknn2
```

```
##
## Call:
## kknn(formula = Y ~ . - X5 - X4, train = datasetR, test = datasetR)
##
## Response: "continuous"
```

```r
yprime <- fitknn2$fitted.values
RMSEknn2 <- sqrt(sum((datasetR$Y-yprime)^2)/length(yprime)) #RMSE
RMSEknn2
```

```
## [1] 0.07425714
```

```r
modelos_comparar <- c(modelos_comparar, ".-X5-X4")
metodos_comparar <- c(metodos_comparar, "k-NN")
RMSE_comparar <- c(RMSE_comparar, RMSEknn2)

fitknn3 <- kknn(Y ~ X6 + I(X6^2), datasetR, datasetR)
fitknn3
```

```
##
## Call:
## kknn(formula = Y ~ X6 + I(X6^2), train = datasetR, test = datasetR)
##
## Response: "continuous"
```

```r
yprime <- fitknn3$fitted.values
RMSEknn3 <- sqrt(sum((datasetR$Y-yprime)^2)/length(yprime)) #RMSE
RMSEknn3
```

```
## [1] 0.08442032
```

```r
modelos_comparar <- c(modelos_comparar, "X6+X6^2")
metodos_comparar <- c(metodos_comparar, "k-NN")
RMSE_comparar <- c(RMSE_comparar, RMSEknn3)

fitknn4 <- kknn(Y ~ X6 + I(X6^2) + I(X6^3), datasetR, datasetR)
fitknn4
```

```
##
## Call:
## kknn(formula = Y ~ X6 + I(X6^2) + I(X6^3), train = datasetR,     test = datasetR)
##
## Response: "continuous"
```

```r
yprime <- fitknn4$fitted.values
RMSEknn4 <- sqrt(sum((datasetR$Y-yprime)^2)/length(yprime)) #RMSE
RMSEknn4
```

```
## [1] 0.08620104
```

```r
modelos_comparar <- c(modelos_comparar, "X6+X6^2+X6^3")
metodos_comparar <- c(metodos_comparar, "k-NN")
RMSE_comparar <- c(RMSE_comparar, RMSEknn4)
```

```r
# Creo un dataframe para almacenar los resultados y grafico
df_cv_R <- data.frame(Algoritmo = metodos_comparar,
                      Modelo = modelos_comparar, RMSE = RMSE_comparar)

# Genero una tabla de resultados
kable(df_cv_R, format = "latex", booktabs = TRUE,
      caption = "\\label{tab:tab1}Comparación modelos utilizando regresión vs k-NN")
```

```r
# grafico los resultados
ggplot(data = df_cv_R, aes(x = Modelo, y = RMSE, fill = Algoritmo)) +
  geom_bar(stat = "identity", position = "dodge") +
```

Table 2: Comparación modelos utilizando regresión vs k-NN

Algoritmo	Modelo	RMSE
LM	X6	0.4151754
LM	.-X5-X4	0.4124937
LM	X6+X6^2	0.2601262
LM	X6+X6^2+X6^3	0.1435029
k-NN	X6	0.0823474
k-NN	.-X5-X4	0.0742571
k-NN	X6+X6^2	0.0844203
k-NN	X6+X6^2+X6^3	0.0862010

```
  theme_light() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) + ggtitle("RMSE LM vs k-NN")
```

R-3. Conclusiones parciales

- Se puede observar que el RMSE es menor cuando comparo el mismo modelo modificando el algoritmo (Ver **Tabla 2**), es decir el mismo modelo si uso k-NN tiene un menor RMSE que utilizando regresión lineal o múltiple.
- El modelo con menor RMSE es el que contempla a todas las variables menos X5 y X4. La Figura 15 y **Tabla 2** compara 4 modelos para las mismas variables utilizando algoritmos de regresión lineal y múltiple versus k-NN.

Prueba de modelos k-NN

Los mejores modelos para k-NN no necesariamente serán los mismos que para regresion lineal, múltiple o no lineal, por lo tanto probaré algunos modelos, y los evaluare calculando el RMSE.

```
fitknn5 <- kknn(Y ~ X6 + X1, datasetR, datasetR) # Construyo el modelo
yprime <- fitknn5$fitted.values # Obtengo los valores predichos por el modelo
RMSEknn5 <- sqrt(sum((datasetR$Y-yprime)^2)/length(yprime)) # calculo del RMSE
RMSEknn5 # Muestro el RMSE
```

```
## [1] 0.07032727
```

```
fitknn6 <- kknn(Y ~ X6 + X2, datasetR, datasetR) # Construyo el modelo
yprime <- fitknn6$fitted.values # Obtengo los valores predichos por el modelo
RMSEknn6 <- sqrt(sum((datasetR$Y-yprime)^2)/length(yprime)) # calculo del RMSE
RMSEknn6 # Muestro el RMSE
```

```
## [1] 0.08218099
```

```
fitknn7 <- kknn(Y ~ X6 + X3, datasetR, datasetR) # Construyo el modelo
yprime <- fitknn7$fitted.values # Obtengo los valores predichos por el modelo
RMSEknn7 <- sqrt(sum((datasetR$Y-yprime)^2)/length(yprime)) # calculo del RMSE
RMSEknn7 # Muestro el RMSE
```

```
## [1] 0.08373089
```

```
fitknn8 <- kknn(Y ~ X6 + X4, datasetR, datasetR) # Construyo el modelo
yprime <- fitknn8$fitted.values # Obtengo los valores predichos por el modelo
```

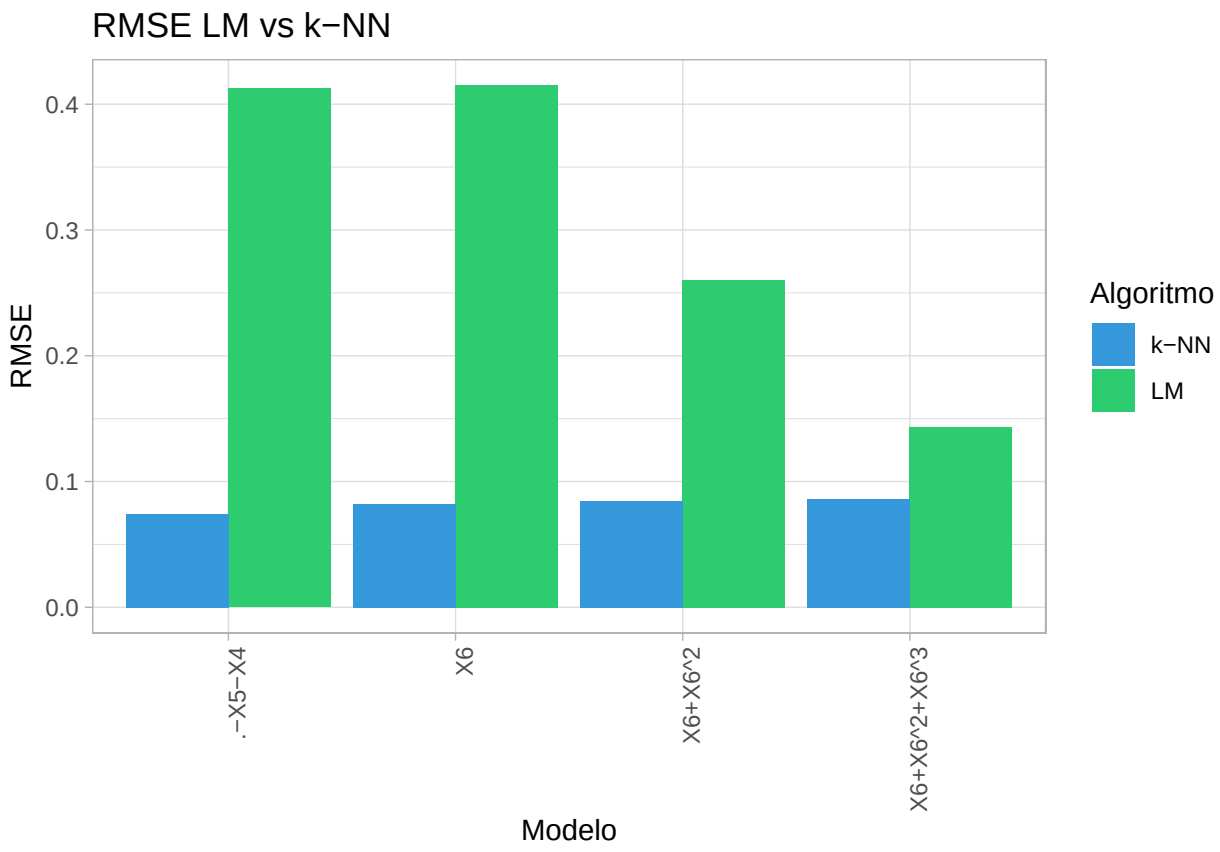


Figure 15: Comparación diferentes modelos empleando regresión lineal y k-NN.

```
RMSEknn8 <- sqrt(sum((datasetR$Y-yprime)^2)/length(yprime)) # calculo del RMSE
RMSEknn8 # Muestro el RMSE
```

```
## [1] 0.08391004
```

```
fitknn9 <- kknn(Y ~ X6 + X5, datasetR, datasetR) # Construyo el modelo
yprime <- fitknn9$fitted.values # Obtengo los valores predichos por el modelo
RMSEknn9 <- sqrt(sum((datasetR$Y-yprime)^2)/length(yprime)) # calculo del RMSE
RMSEknn9 # Muestro el RMSE
```

```
## [1] 0.08665816
```

```
fitknn10 <- kknn(Y ~ X6 + X7, datasetR, datasetR) # Construyo el modelo
yprime <- fitknn10$fitted.values # Obtengo los valores predichos por el modelo
RMSEknn10 <- sqrt(sum((datasetR$Y-yprime)^2)/length(yprime)) # calculo del RMSE
RMSEknn10 # Muestro el RMSE
```

```
## [1] 0.08598704
```

```
fitknn11 <- kknn(Y ~ X6*X1, datasetR, datasetR) # Construyo el modelo
yprime <- fitknn11$fitted.values # Obtengo los valores predichos por el modelo
RMSEknn11 <- sqrt(sum((datasetR$Y-yprime)^2)/length(yprime)) # calculo del RMSE
RMSEknn11 # Muestro el RMSE
```

```
## [1] 0.07261029
```

Conclusiones parciales

Los modelos X6+X1 y X6*X1 tienen un RMSE menor al menor modelo obtenido anteriormente, son los dos mejores modelos obtenidos con k-NN.

Cross validation

Finalmente para observar que tan bien generalizan los modelos construidos en los pasos anteriores, voy a realizar 5-Fold Cross Validation.

Voy a probar los dos mejores modelos obtenidos con regresión y los dos mejores modelos obtenidos con k-NN, teniendo en cuenta el RMSE. Los modelos seleccionados son los siguientes:

Regresión lineal, múltiple o interacciones

- Y ~ X6 + I(X6^2)
- Y ~ X6 + I(X6^2) + I(X6^3)

k-NN

- Y ~ X6+X1
- Y ~ X6*X1

Voy a calcular el **error cuadrático medio** tanto para training como testing de los 4 modelos seleccionados empleando modelos lineales simples y múltiples, además de k-NN.

```
# Función para ejecutar 5 fold cross validation con modelos lineales
# Toma como entradas:
# -i: indice del fold
# -x: nombre del archivo con los datos
# -formula: variables para construir el modelo
# -tt: dataset de train o test, para calcular el MSE
# Retorna: MSE
```

```r
run_lm_fold <- function(i, x, formula, 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(paste("Y ~ ", formula, sep = ""), data = x_tra)
  yprime = predict(fitMulti, test)
  sum(abs(test$Y - yprime)^2)/length(yprime) #MSE
}

# Función para ejecutar 5 fold cross validation con KNN
# Toma como entradas:
# -i: indice del fold
# -x: nombre del archivo con los datos
# -formula: variables para construir el modelo
# -tt: dataset de train o test, para calcular el MSE
# Retorna: MSE
run_knn_fold <- function(i, x, formula, 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( paste("Y~", formula, sep = ""), x_tra, test)
  yprime=fitMulti$fitted.values
  sum(abs(test$Y-yprime)^2)/length(yprime) # MSE
}

# Variables para almacenar resultados
modelo <- c()
método <- c()
MSEtrain <- c()
MSEtest <- c()

# Nombre del dataset
nombre <- "DATOS/Datasets\ Regresion/ANACALT/ANACALT"
```

```
# Pruebo el modelo Y~X6+I(X6^2)
# MSE train
lmMSEtrain<-mean(sapply(1:5, run_lm_fold,
                        nombre, "X6+I(X6^2)", "train"))
# MSE test
lmMSEtest<-mean(sapply(1:5, run_lm_fold,
                       nombre, "X6+I(X6^2)", "test"))
# Guardo la información de resultados
modelo <- c(modelo, "X6+I(X6^2)")
método <- c(método, "LM")
MSEtrain <- c(MSEtrain, lmMSEtrain)
MSEtest <- c(MSEtest, lmMSEtest)

# Pruebo el modelo Y~X6+I(X6^2)+I(X6^3)
lmMSEtrain<-mean(sapply(1:5, run_lm_fold, nombre, "X6+I(X6^2)+I(X6^3)", "train"))
lmMSEtest<-mean(sapply(1:5,run_lm_fold, nombre, "X6+I(X6^2)+I(X6^3)", "test"))
modelo <- c(modelo, "X6+I(X6^2)+I(X6^3)")
método <- c(método, "LM")
MSEtrain <- c(MSEtrain, lmMSEtrain)
MSEtest <- c(MSEtest, lmMSEtest)

# ------------------------------------------------
# KNN
# Pruebo el modelo Y~X6+X1
knnMSEtrain<-mean(sapply(1:5, run_knn_fold, nombre, "X6+X1", "train"))
knnMSEtest<-mean(sapply(1:5,run_knn_fold, nombre, "X6+X1", "test"))
modelo <- c(modelo, "X6+X1")
método <- c(método, "k-NN")
MSEtrain <- c(MSEtrain, knnMSEtrain)
MSEtest <- c(MSEtest, knnMSEtest)

# Pruebo el modelo Y~X6*X1
knnMSEtrain<-mean(sapply(1:5, run_knn_fold, nombre, "X6*X1", "train"))
knnMSEtest<-mean(sapply(1:5,run_knn_fold, nombre, "X6*X1", "test"))
modelo <- c(modelo, "X6*X1")
método <- c(método, "k-NN")
MSEtrain <- c(MSEtrain, knnMSEtrain)
MSEtest <- c(MSEtest, knnMSEtest)

# Creo un dataframe para graficar los resultados
método <- as.factor(método)
modelo <- as.factor(modelo)
df <- data.frame(Método = método, Modelo = modelo,
                 MSEtrain = MSEtrain, MSEtest = MSEtest)

# Genero tabla de resultados
kable(df, format="latex", booktabs = TRUE,
      caption = "\\label{tab:tab2}Resultados cross-validation regresión")
```

```
# Grafico los resultados
df <- melt(df)
```

```
## Using Método, Modelo as id variables
```

Table 3: Resultados cross-validation regresión

Método	Modelo	MSEtrain	MSEtest
LM	X6+I(X6^2)	0.0676304	0.0681243
LM	X6+I(X6^2)+I(X6^3)	0.0205900	0.0207504
k-NN	X6+X1	0.0044152	0.0069101
k-NN	X6*X1	0.0042808	0.0069361

```
ggplot(data = df, aes(x = Modelo, y = value, fill = variable)) +
  geom_bar(stat = "identity", position = "dodge") +
  facet_wrap(~Método, scales = "free_x") +
  theme_light() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  ylab("MSE")
```

Conclusiones parciales

Se puede observar que el algoritmo k-NN obtiene los mejores resultados de cross-validation. El mejor resultado es para el modelo X6+X1 que tiene un MSE en test de 0.0069. Ver **Tabla 3**.

Comparación estadística de algoritmos regresión (R.4)

Procedere a comparar diferentes algoritmos. Para comenzar los resultados de los modelos de regresion lineal vs k-NN y luego agregaré a la comparación el algoritmo M5'.

```
# Calculo los errores medios de train y test, para k-NN y LM utilizando todas las variables,
# para reemplazar en la tabla
lmMSEtrainAll<-mean(sapply(1:5, run_lm_fold, nombre, ".", "train"))
lmMSEtestAll<-mean(sapply(1:5,run_lm_fold, nombre, ".", "test"))

knnMSEtrainAll<-mean(sapply(1:5, run_knn_fold, nombre, ".", "train"))
knnMSEtestAll<-mean(sapply(1:5,run_knn_fold, nombre, ".", "test"))


# Lectura de los datos
#leemos la tabla con los errores medios de test
resultados <- read.csv("regr_test_alumnos.csv")
tablatst <- cbind(resultados[,2:dim(resultados)[2]])
colnames(tablatst) <- names(resultados)[2:dim(resultados)[2]]
rownames(tablatst) <- resultados[,1]
# Reemplazo por mis valores
tablatst['ANACALT', 1] <- lmMSEtestAll # lm
tablatst['ANACALT', 2] <- knnMSEtestAll # KNN

kable(tablatst, format = "latex", booktabs = TRUE, caption =
        "\\label{tab:tabCAR1}Errores medios de test para los algoritmos k-NN, lm y M5'")
```

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

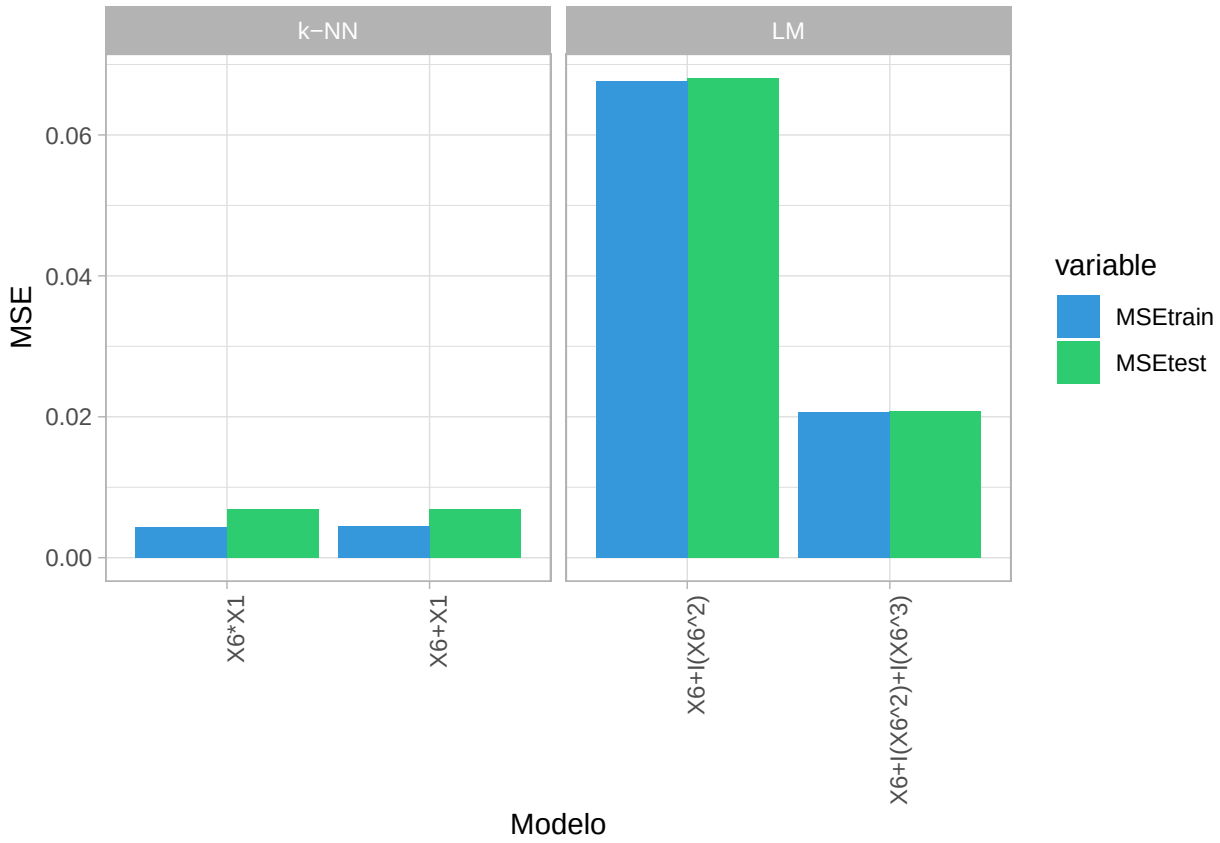


Figure 16: Comparación 5-fold cross validation LM vs k-NN

Table 4: Errores medios de test para los algoritmos k-NN, lm y M5'

	out_test_lm	out_test_kknn	out_test_m5p
abalone	4.950000e+00	5.400000e+00	4.680000e+00
ANACALT	1.711604e-01	1.156870e-02	7.000000e-03
autoMPG6	1.162000e+01	7.740000e+00	8.240000e+00
autoMPG8	1.140000e+01	8.110000e+00	8.350000e+00
baseball	5.366760e+05	5.661130e+05	5.464640e+05
california	4.844366e+09	3.845914e+09	3.158145e+09
concrete	1.090000e+02	6.835600e+01	3.800000e+01
dee	1.705200e-01	1.732600e-01	1.699600e-01
delta_ail	0.000000e+00	0.000000e+00	0.000000e+00
delta_elv	2.100000e-06	2.400000e-06	2.000000e-06
forestFires	4.060940e+03	5.841000e+03	4.071040e+03
friedman	7.298700e+00	3.196100e+00	5.349100e+00
house	2.072908e+09	1.425915e+09	1.305419e+09
mortgage	1.484100e-02	3.003600e-02	1.448300e-02
stock	5.510000e+00	4.500000e-01	1.000000e+00
treasury	6.082100e-02	4.743900e-02	8.124800e-02
wankara	2.490000e+00	6.790000e+00	1.650000e+00
wizmir	1.605000e+00	6.060000e+00	1.449000e+00

Table 5: Errores medios de train para los algoritmos k-NN, lm y M5'

	out_train_lm	out_train_kknn	out_train_m5p
abalone	4.820000e+00	2.220000e+00	4.250000e+00
ANACALT	1.699982e-01	6.250400e-03	5.900000e-03
autoMPG6	1.129000e+01	3.530000e+00	6.870000e+00
autoMPG8	1.079000e+01	3.550000e+00	6.600000e+00
baseball	4.481590e+05	2.020880e+05	3.925890e+05
california	4.826190e+09	1.560869e+09	2.558518e+09
concrete	1.070000e+02	2.870000e+01	3.000000e+01
dee	1.618800e-01	7.611000e-02	1.620100e-01
delta_ail	0.000000e+00	0.000000e+00	0.000000e+00
delta_elv	2.100000e-06	1.000000e-06	2.000000e-06
forestFires	3.945000e+03	2.206000e+03	3.980000e+03
friedman	7.230000e+00	1.420000e+00	4.360000e+00
house	2.061567e+09	5.259870e+08	9.384299e+08
mortgage	1.354300e-02	8.827000e-03	1.101500e-02
stock	5.350000e+00	1.800000e-01	5.900000e-01
treasury	5.520300e-02	1.599800e-02	4.040400e-02
wankara	2.430000e+00	2.740000e+00	1.510000e+00
wizmir	1.565000e+00	2.538000e+00	1.358000e+00

```
rownames(tablatra) <- resultados[,1]
# Reemplazo por mis valores
tablatra['ANACALT', 1] <- lmMSEtrainAll # lm
tablatra['ANACALT', 2] <- knnMSEtrainAll # KNN

kable(tablatra, format = "latex", booktabs = TRUE, caption =
        "\\label{tab:tabCAR1}Errores medios de train para los algoritmos k-NN, lm y M5'")
```

Wilcoxon

Primero realizare una comparativa de a pares entre los algoritmos de LM y k-NN utilizando el test de Wilcoxon. Este test contrasta la hipótesis de que las medianas son iguales frente a que son distintas (también caben otras alternativas) y únicamente se tienen en cuenta las situaciones relativas (ranks) de los valores (no se tiene en cuenta las magnitudes de las diferencias).

Primero es necesario normalizar las medidas del error. En este caso estamos utilizando el error cuadrático medio, y este valor depende de cada dataset en particular. Por esto utilizare la siguiente normalización[1]:

DIFF = (Mean(Other) - Mean(Reference Algorithm)) / Mean(Other)

El algoritmo de referencia es el que creemos que es mejor, de acuerdo las medidas de error que obtuvimos durante los análisis, en este caso k-NN.

Datos de test

```
# Normalización del error
```

[1]M.J. Gacto, R. Alcalá, F. Herrera, Integration of an Index to Preserve the Semantic Interpretability in the Multi-Objective Evolutionary Rule Selection and Tuning of Linguistic Fuzzy Systems. IEEE Transactions on Fuzzy Systems 18:3 (2010) 515-531

```
# Se monta una nueva tabla donde para cada conjunto de datos, si la
# diferencia es positiva se pone 0 para Other y abs(DIFF) para Ref.Alg., y
# a la inversa si es negativa

##lm (other) vs knn (ref)
# + 0.1 porque wilcox R falla para valores == 0 en la tabla
difs <- (tablatst[,1] - tablatst[,2]) / tablatst[,1]
wilc_1_2 <- cbind(ifelse (difs<0, abs(difs)+0.1, 0+0.1), ifelse (difs>0, abs(difs)+0.1, 0+0.1))
colnames(wilc_1_2) <- c(colnames(tablatst)[1], colnames(tablatst)[2])
```

Datos de train

```
##lm (other) vs knn (ref)
difs_train <- (tablatra[,1] - tablatra[,2]) / tablatra[,1]
wilc_1_2_train <- cbind(ifelse (difs_train<0, abs(difs_train)+0.1, 0+0.1),
                        ifelse (difs_train>0, abs(difs_train)+0.1, 0+0.1))
colnames(wilc_1_2_train) <- c(colnames(tablatra)[1], colnames(tablatra)[2])
```

Aplico el test de Wilcoxon

Datos de test

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

```
##  V
## 78
```

```
Rmenos
```

```
##  V
## 93
```

```
pvalue
```

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

- El p-valor no es significativo. No existen diferencias significativas entre ambos algoritmos.
- Sólo hay un (1-0.7660) *100 = 23.4% de confianza en que sean distintos.

Datos de train

```
# LM(R+) vs KNN(R-)
LMvsKNNtra <- wilcox.test(wilc_1_2_train[,1], wilc_1_2_train[,2],
                          alternative = "two.sided", paired=TRUE)
Rmastra <- LMvsKNNtra$statistic
pvaluetra <- LMvsKNNtra$p.value
LMvsKNNtra <- wilcox.test(wilc_1_2_train[,2], wilc_1_2_train[,1],
                          alternative = "two.sided", paired=TRUE)
Rmenostra <- LMvsKNNtra$statistic
Rmastra
```

```
##  V
## 10
```

```
Rmenostra
```

```
##   V
## 161
```

```
pvaluetra
```

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

- El p-valor es significativo. Existen diferencias significativas entre ambos algoritmos.
- Hay un (1-0.00032)*100 = 99.96% de confianza en que sean distintos.

Conclusiones

- Sobre datos de testing no hay diferencias entre LM y k-NN. Si se observan diferencias significativas en favor de k-NN sobre datos de train. En el dataset ANACALT en particular no parece existir un problema de sobreajuste de k-NN, ya que pudimos observar que el algoritmo generaliza bien sobre test. Pero los resultados de este test, pueden ser indicativos que k-NN esta sobreajustando sobre otros datasets que formaron parte de la comparativa.

Comparativas múltiples

Usaremos el test Friedman que es la extensión para mas de dos variables del test de Wilcoxon. El test de Friedman trabaja asignando rankings a los resultados obtenidos por cada algoritmo en cada problema. La hipótesis nula indica que todos los algoritmos se comportan similarmente, por lo que sus rankings deben ser similares.

Aplicamos el test de Friedman

Datos de test

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

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

- El p-valor de 0.01 indica que debemos rechazar la hipótesis nula, y que existen diferencias significativas al menos entre un par de algoritmos.

Datos de train

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

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

- El p-valor de 3.84e-05 indica que debemos rechazar la hipótesis nula, y que existen diferencias significativas al menos entre un par de algoritmos.

Como el test de Friedman fue significativo tanto para train como para test, tenemos que realizar un post-hoc que nos diga qué algoritmos pueden considerarse similares entre sí. En este caso utilizaré el test de Holm.

Procederemos a aplicar el test de Holm

Datos de test

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

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

- Existen diferencias significativas a favor de M5' (3vs1 0.081 y 3vs2 0.108, con aproximasdamente 90% de confianza) mientras que los otros dos pueden ser considerados equivalentes.

Datos de train

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

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

- Existen diferencias significativas a favor de k-NN vs LM y M5', y de M5' vs LM.

Apartado Clasificación

k-NN

Uso del paquete class

Se utilizara el algoritmo k-NN, testeando diferentes valores de K, para seleccionar el modelo con el valor de K más adecuado para el dataset.

Para comparar los diferentes valores de K, realizare 10-Fold Cross Validation, utilizando la métrica F1 para seleccionar el valor de K más adecuado. Generaré graficos de varias métricas, calculadas tanto en training como testing, a fines observacionales.

```
set.seed(11) # Semilla para reproducibilidad de resultados

K <- 10 # Valor de K para cross validation

data_use <- datasetC

# Creo los folds para cross validation
data_folds_C <- createFolds(data_use[, "Y"], k = K,
                            list = TRUE, returnTrain = FALSE)

# createFolds, crea folds balanceados, aca se puede comprobar
sapply(data_folds_C, function(i) table(data_use$Y[i]))
```

```
##   Fold01 Fold02 Fold03 Fold04 Fold05 Fold06 Fold07 Fold08 Fold09 Fold10
## 2     45     44     45     44     45     44     44     44     44     44
## 4     24     24     24     24     24     24     24     24     24     23
```

```
# Creo una funcion que toma como parámetros:
# Kval: Valor de K
# data: dataset
# data_folds: data_folds creados con la funcion createFolds
# response_name: Nombre de la columna con la variable dependiente
# predict_on: pasar train como parametro si queremos calcular el training error.
FitPredictCVKNN <- function(Kval, data, data_folds, response_name, predict_on = "test") {

  folds <- length(data_folds)

  # Variables para almacenar los resultados
  accuracy_list <- list() # Accuracy: = Number of correct predictions / Total number of predictions
  error_list <- list()    # Error rate: 1 - Accuracy
  recall_list <- list()   # Recall= TP / (TP + FN)
  precision_list <- list()# Precision= TP / (TP + FP)
  f1_list <- list()       # 2*( (precision*recall) / (precision + recall))

    # Itero por la cantidad de folds
    for(i in 1:folds){

      # Obtengo los indices y
      fold_index <- data_folds[[i]]

      # Separo el dataset en train y test
      X_train <- data[-fold_index, !(colnames(data) %in% response_name)]
```

```
      X_test <- data[fold_index, !(colnames(data) %in% response_name)]
      y_train <- data[-fold_index, response_name]
      y_test <- data[fold_index, response_name]

      # Para ver si calculo el error de train o test
      if(predict_on == "test") {
        pred <- knn(k = Kval, train = X_train, test = X_test, cl = y_train)
        cm <- as.matrix(table(Predicted = pred, Actual = y_test))
      } else { # predict on train
        pred <- knn(k = Kval, train = X_train, test = X_train, cl = y_train)
        cm <- as.matrix(table(Predicted = pred, Actual = y_train))
      }

      # Calculo estadísticas del fold
      accuracy <- sum(diag(cm))/sum(cm)
      TP <- cm[1, 1]
      FN <- cm[2, 1]
      FP <- cm[1, 2]
      TN <- cm[2, 2]
      recall <- TP / (TP + FN)
      precision <- TP / (TP + FP)
      f1 <- 2*( (precision*recall) / (precision + recall))
      error_list[[i]] <- 1 - accuracy
      accuracy_list[[i]] <- accuracy
      recall_list[[i]] <- recall
      precision_list[[i]] <- precision
      f1_list[[i]] <- f1

    }

  # Obtengo la media de las métricas de Cross Validation
  accuracy <- mean(unlist(accuracy_list))
  recall <- mean(unlist(recall_list))
  precision <- mean(unlist(precision_list))
  f1 <- mean(unlist(f1_list))
  error_rate <- mean(unlist(error_list))

  # Genero un vector de resultados y lo devuelvo
  vec <- list(Accuracy = accuracy,
              Recall = recall,
              Precision = precision,
              F1 = f1,
              ErrorRate = error_rate)
  vec
}

# Defino los valores de K a probar, desde 1 hasta la raiz cuadrada de la cantidad de muestras
# en pasos de 2, para que sean valores impares de K
k_values <- seq(1, sqrt(nrow(datasetC)), 2)

# Obtengo métricas de cross validation para train y test
train_metrics <- lapply(k_values, FitPredictCVKNN, data_use, data_folds_C, "Y", predict_on = "train")
test_metrics <- lapply(k_values, FitPredictCVKNN, data_use, data_folds_C, "Y", predict_on = "test")
```

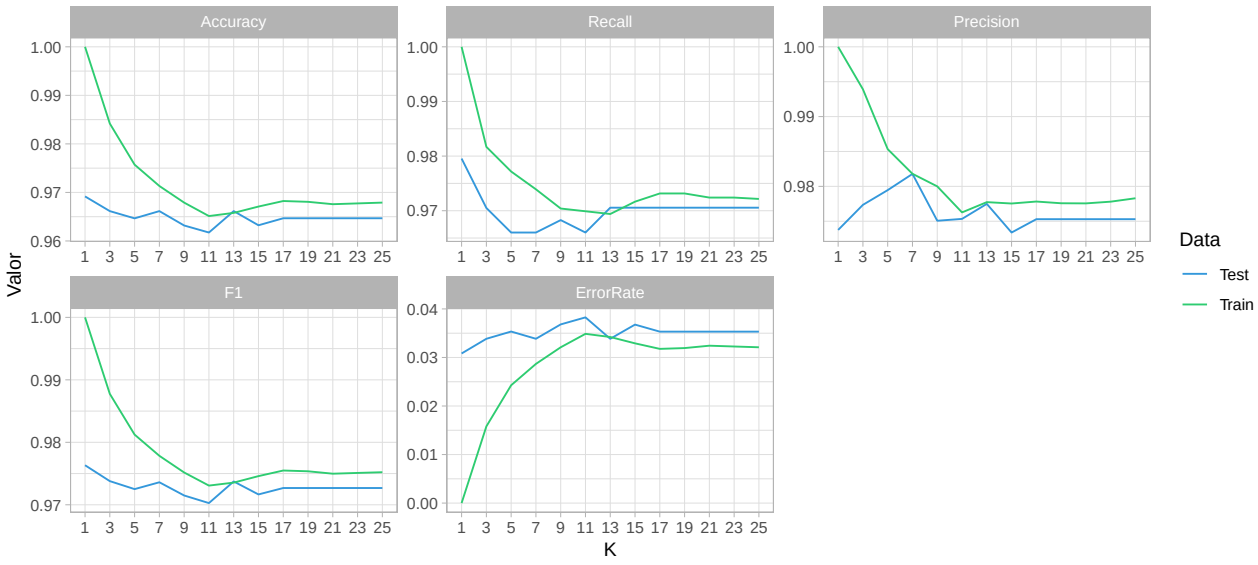


Figure 17: Diferentes métricas calculadas para diferentes valores de K para el algoritmo KNN, tanto en training como testing utilizando 10-Fold Cross Validation

```
# Creo un dataframe con los resultados de train
train_df <- as.data.frame(do.call(rbind, train_metrics))
colnames(train_df) <- c("Accuracy", "Recall", "Precision", "F1", "ErrorRate")
train_df$K <- as.factor(k_values)
train_df$Data <- "Train"

# Creo un dataframe con los resultados de test
test_df <- as.data.frame(do.call(rbind, test_metrics))
colnames(test_df) <- c("Accuracy", "Recall", "Precision", "F1", "ErrorRate")
test_df$K <- as.factor(k_values)
test_df$Data <- "Test"

# Mergeo el dataframe
df <- rbind(train_df, test_df)
df$Accuracy <- as.numeric(df$Accuracy)
df$Recall <- as.numeric(df$Recall)
df$Precision <- as.numeric(df$Precision)
df$F1 <- as.numeric(df$F1)
df$ErrorRate <- as.numeric(df$ErrorRate)
df_long <- melt(df)
```

```
## Using K, Data as id variables
```

```
kable(df, format = "latex", booktabs = TRUE, caption =
        "\\label{tab:tabKNNC}Estadísticas para diferentes valores de K del algoritmo k-NN, cálculadas u
```

```
# Grafico los resultados de diferentes métricas, para diferentes valores de K,
# tanto en train como test durante el proceso de cross validation
ggplot(data = df_long, aes(x = K, y = value, group = Data, color = Data)) +
  ylab("Valor") + geom_line() + theme_light() +
  facet_wrap(~variable, scales = "free")
```

Table 6: Estadísticas para diferentes valores de K del algoritmo k-NN, cálculadas utilizando 10-Fold Cross Validation

Accuracy	Recall	Precision	F1	ErrorRate	K	Data
1.0000000	1.0000000	1.0000000	1.0000000	0.0000000	1	Train
0.9841958	0.9816898	0.9939124	0.9877587	0.0158042	3	Train
0.9757244	0.9771760	0.9853348	0.9812353	0.0242756	5	Train
0.9713259	0.9739166	0.9818013	0.9778378	0.0286741	7	Train
0.9679039	0.9704034	0.9799958	0.9751715	0.0320961	9	Train
0.9651335	0.9699015	0.9762677	0.9730738	0.0348665	11	Train
0.9657853	0.9694003	0.9777387	0.9735500	0.0342147	13	Train
0.9670890	0.9716578	0.9775417	0.9745896	0.0329110	15	Train
0.9682296	0.9731622	0.9778240	0.9754859	0.0317704	17	Train
0.9680662	0.9731622	0.9775773	0.9753636	0.0319338	19	Train
0.9675773	0.9724090	0.9775601	0.9749767	0.0324227	21	Train
0.9677405	0.9724090	0.9778063	0.9750993	0.0322595	23	Train
0.9679031	0.9721590	0.9782934	0.9752140	0.0320969	25	Train
0.9691797	0.9795455	0.9737532	0.9763319	0.0308203	1	Test
0.9661520	0.9705051	0.9773491	0.9737903	0.0338480	3	Test
0.9646594	0.9660101	0.9794606	0.9725102	0.0353406	5	Test
0.9661520	0.9660101	0.9817835	0.9736046	0.0338480	7	Test
0.9631889	0.9682828	0.9750690	0.9714911	0.0368111	9	Test
0.9617396	0.9660101	0.9753362	0.9702731	0.0382604	11	Test
0.9661520	0.9705556	0.9774885	0.9737341	0.0338480	13	Test
0.9632315	0.9705556	0.9733833	0.9716569	0.0367685	15	Test
0.9646808	0.9705556	0.9752965	0.9726864	0.0353192	17	Test
0.9646808	0.9705556	0.9752965	0.9726864	0.0353192	19	Test
0.9646808	0.9705556	0.9752965	0.9726864	0.0353192	21	Test
0.9646808	0.9705556	0.9752965	0.9726864	0.0353192	23	Test
0.9646808	0.9705556	0.9752965	0.9726864	0.0353192	25	Test

Conclusión

- A partir de la inspeccion de los resultados (Figura 17, Tabla 6), decido que el valor de K = 1 es el más apropiado para este dataset. Presenta el F1 mas alto calculado sobre testing para 10-Fold CV. Además posee el menor error rate en testing.

Habiendo definido el valor de K = 1, voy a repetir el calculo solo para este K, para obtener el accuracy de cada fold de cross validation, luego compararé estos resultados con los algoritmos de LDA y QDA para ver cual funciona mejor sobre el dataset.

Además ejecutare la cross validation sobre los mismos folds, pero con el dataset sin preprocesar, solo normalizado, para chequear si el preprocesamiento de datos mejora el modelo.

```
getKNNaccuracy <- function(folds, data, data_folds, Kval, response_name = "Y") {

      # Obtengo los indices y
      fold_index <- data_folds[[folds]]

      # Separo el dataset en train y test
      X_train <- data[-fold_index, !(colnames(data) %in% response_name)]
      X_test <- data[fold_index, !(colnames(data) %in% response_name)]
      y_train <- data[-fold_index, response_name]
      y_test <- data[fold_index, response_name]

      # Para ver si calculo el error de train o test
      pred <- knn(k = Kval, train = X_train, test = X_test, cl = y_train)
      cm <- as.matrix(table(Predicted = pred, Actual = y_test))

      # Calculo estadísticas del fold
      accuracy <- sum(diag(cm))/sum(cm)
      accuracy
}

knn_accuracy_comparar <- unlist(lapply(seq(1:K), getKNNaccuracy, data_use, data_folds_C, 1))
knn_accuracy_comparar
```

```
##  [1] 1.0000000 0.9558824 0.9565217 0.9852941 0.9565217 0.9852941 0.9264706
##  [8] 1.0000000 0.9705882 0.9552239
```

Repito para el dataset que solo fue normalizado como preprocesamiento.

```
knn_accuracy_comparar_no_procesado <- unlist(lapply(seq(1:K),
                                                    getKNNaccuracy,
                                                    datasetC_sin_procesar,
                                                    data_folds_C,
                                                    1))
knn_accuracy_comparar_no_procesado
```

```
##  [1] 0.9855072 0.9705882 0.9130435 0.9558824 0.9420290 0.9705882 0.9117647
##  [8] 0.9705882 0.9411765 0.9402985
```

Uso del paquete caret

A modo comparativo con el proceso manual anterior, donde utilize la funcion knn del paquete class, ahora realizare 10-Fold Cross Validation con diferentes valores de K, pero utilizando el paquete *caret*. Puedo utilizar las mismas particiones para el 10-Fold crss validation.

El paquete caret me permite realizar las tareas que realize manualmente de una forma más automatizada.

```
# 10-Fold cross validation

# Uso las mismas particiones que en el modelo anterior (data_folds_C)
trControl <- trainControl(method  = "cv",
                          index = data_folds_C)

# Entreno el modelo
model_knn_caret <- train(Y ~ .,
             method     = "knn",
             # Utilizo los mismos valores de K
             tuneGrid   = expand.grid(k = k_values),
             trControl  = trControl,
             metric     = "Accuracy",
             data       = data_use
             )
model_knn_caret
```

```
## k-Nearest Neighbors
##
## 682 samples
##   8 predictor
##   2 classes: '2', '4'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 69, 68, 69, 68, 69, 68, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    1  0.9530838  0.8960050
##    3  0.9623656  0.9172204
##    5  0.9631799  0.9190978
##    7  0.9633409  0.9194070
##    9  0.9633394  0.9193017
##   11  0.9644808  0.9217519
##   13  0.9621996  0.9166185
##   15  0.9615478  0.9150795
##   17  0.9597560  0.9109720
##   19  0.9586157  0.9084218
##   21  0.9561722  0.9028358
##   23  0.9548695  0.8997825
##   25  0.9524260  0.8941186
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 11.
```

```
summary(model_knn_caret)
```

```
##             Length Class      Mode
## learn       2      -none-     list
## k           1      -none-     numeric
## theDots     0      -none-     list
## xNames      8      -none-     character
```

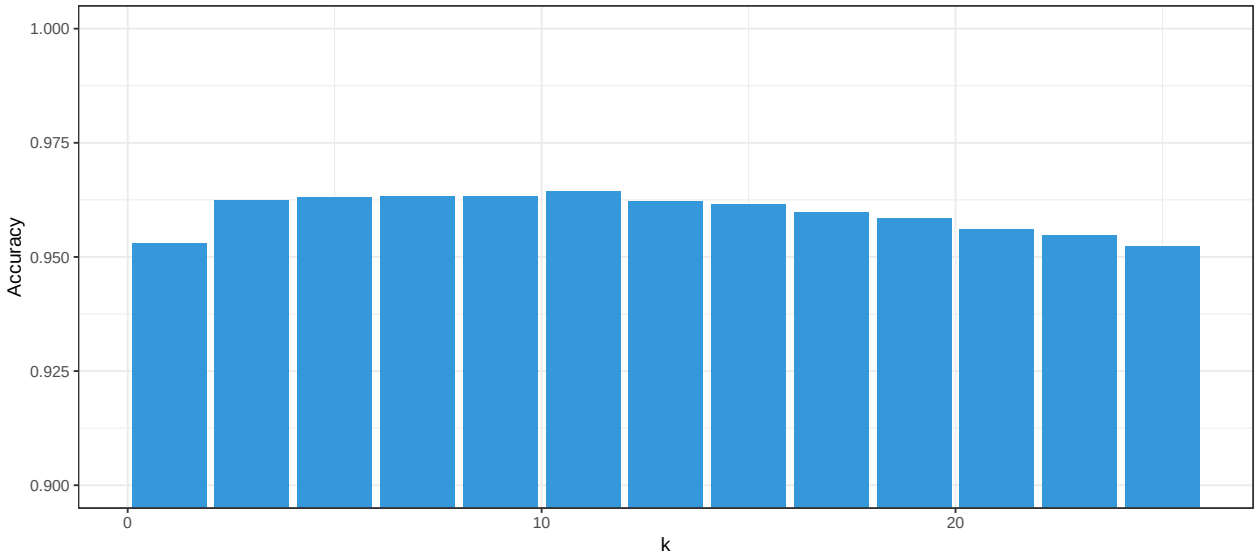


Figure 18: Comparación de diferentes valores de K para kNN en caret

```
## problemType 1      -none-     character
## tuneValue   1      data.frame list
## obsLevels   2      -none-     character
## param       0      -none-     list
```

```
print(paste0("El mejor modelo es el que posee un K de : ",
             model_knn_caret$bestTune))
```

```
## [1] "El mejor modelo es el que posee un K de : 11"
```

```
# Graficos las estadísticas
ggplot(data = model_knn_caret$results, aes(x = k, y = Accuracy)) +
  geom_bar(stat = "identity") +
  coord_cartesian(ylim=c(0.9, 1)) +
  theme_bw()
```

Conclusión

- Utilizando la implementación del paquete caret, el mejor valor de K segun la accuracy calculada luego de 10-Fold CV es K = 11.

LDA

Chequeo asumpciones LDA y QDA

Análisis de normalidad univariante, normalidad multivariante y homogeneidad de varianza

Las condiciones que se deben cumplir para que un Análisis Discriminante Lineal sea válido son:

- Cada predictor que forma parte del modelo se distribuye de forma normal en cada una de las clases de la variable respuesta. En el caso de múltiples predictores, las observaciones siguen una distribución normal multivariante en todas las clases. Para chequear la normalidad univariante de los predictores se puede utilizar el test de normalidad Shapiro-Wilk para cada variable predictora en cada clase de la

Table 7: Test de normalidad de Shapiro-Wilk

Y	variable	p_value_Shapiro.test
2	X1	0
2	X3	0
2	X4	0
2	X5	0
2	X6	0
2	X7	0
2	X8	0
2	X9	0
4	X1	0
4	X3	0
4	X4	0
4	X5	0
4	X6	0
4	X7	0
4	X8	0
4	X9	0

variable dependiente. Para testear la normalidad multivariante se pueden emplear los test de Royston y Henze-Zirkler

- La varianza del predictor es igual en todas las clases de la variable respuesta. En el caso de múltiples predictores, la matriz de covarianza es igual en todas las clases. Si esto no se cumple se recurre a Análisis Discriminante Cuadrático (QDA).

```
# Test de normalidad Shapiro-Wilk para cada variable predictora
# en cada clase de la variable dependiente
data_tidy <- melt(data_use, value.name = "valor")
```

```
## Using Y as id variables
```

```
kable(data_tidy %>% group_by(Y, variable) %>%
        summarise(p_value_Shapiro.test = shapiro.test(valor)$p.value),
      format = "latex", booktabs = TRUE, caption =
        "\\label{tab:tab3}Test de normalidad de Shapiro-Wilk")
```

```
# Test de normalidad multivariante Royston
royston_test <- mvn(data = data_use[,-9], mvnTest = "royston", multivariatePlot = "qq")
```

```
royston_test$multivariateNormality
```

```
##      Test        H       p value MVN
## 1 Royston 1071.547 1.420205e-226  NO
```

```
royston_test$univariateNormality
```

```
##           Test  Variable Statistic   p value Normality
## 1 Shapiro-Wilk    X1        0.9025  <0.001      NO
## 2 Shapiro-Wilk    X3        0.7460  <0.001      NO
## 3 Shapiro-Wilk    X4        0.6987  <0.001      NO
## 4 Shapiro-Wilk    X5        0.6173  <0.001      NO
## 5 Shapiro-Wilk    X6        0.6700  <0.001      NO
## 6 Shapiro-Wilk    X7        0.7543  <0.001      NO
```

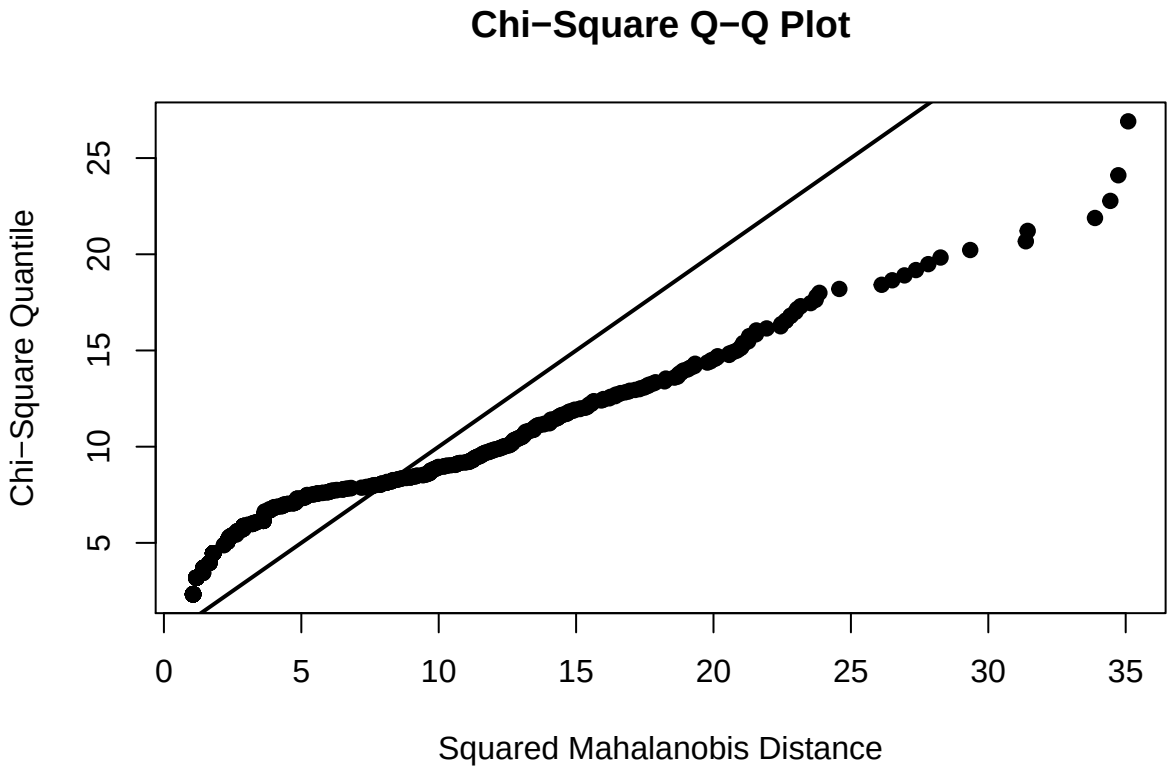


Figure 19: Imagen generada por la función que realiza el Test de normalidad multivariante Royston

```
## 7 Shapiro-Wilk    X8        0.5849  <0.001      NO
## 8 Shapiro-Wilk    X9        0.4613  <0.001      NO
```

```
# Test de normalidad multivariante de Henze-Zirkler
hz_test <- mvn(data = data_use[,-9], mvnTest = "hz")
hz_test$multivariateNormality
```

```
##            Test       HZ p value MVN
## 1 Henze-Zirkler 36.82453       0  NO
```

Conclusión:

- Todas las variables predictoras fueron significativas en el test Shapiro Wilk (tabla 7). Se puede concluir que no hay normalidad univariante en todas las variables empleadas como predictores.
- Los test de normalidad multivariante tambien fueron significativos, indicando que no hay normalidad multivariante.

Se puede utilizar el test M de Box (1949) para determinar la homogeneidad de las matrices de covarianza obtenidas a partir de datos normales multivariados según uno o más factores de clasificación.

- H_o = Matriz de covarianza de la variable dependiente es igual en todos los grupos
- H_a = Matriz de covarianza de la variable dependiente es diferente en al menos uno de los grupos

```
# Chequeo homogeneidad varianza
unlist(lapply(X = data_use[, -9], FUN = var))
```

```
## X1 X3 X4 X5 X6 X7 X8 X9
##  1  1  1  1  1  1  1  1
```

```
# Test de homogeneidad matriz covarianza
# Utilizo el test M de Box (1949). Tiene como hipótesis nula que las matrices de covarianza son iguales
```

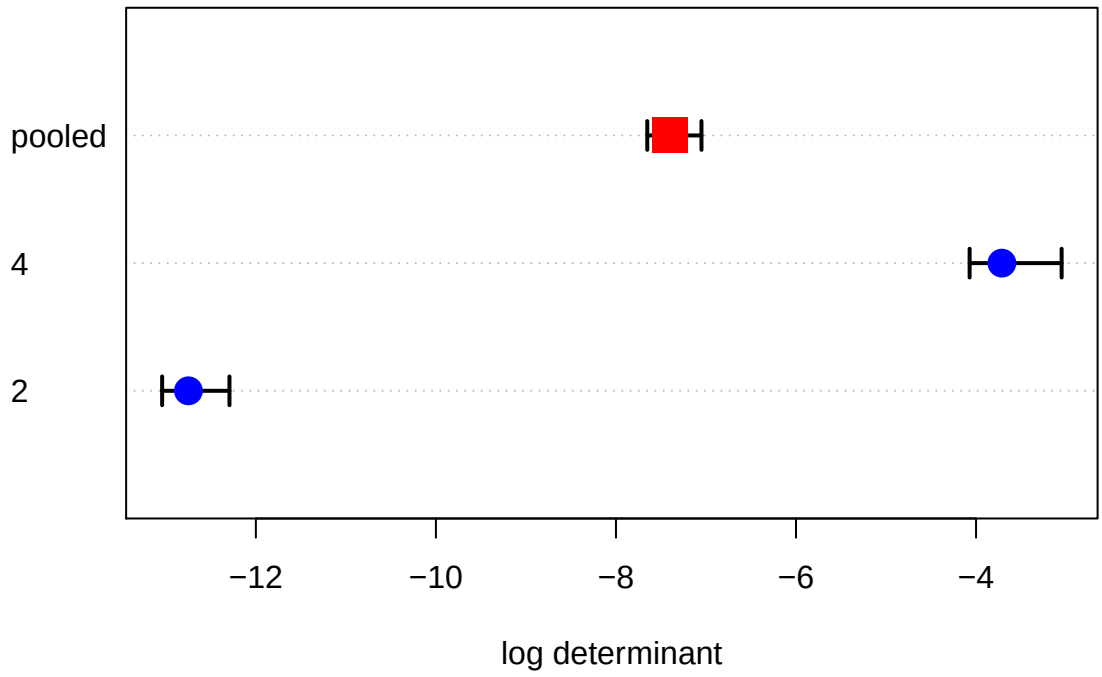


Figure 20: Imagen generada por la función que realiza el Test M de Box

```
# Importante: sensible a la falta de normalidad multivariante
boxm <- heplots::boxM(data_use[, -9], data_use[, 9])
boxm
```

```
##
##  Box's M-test for Homogeneity of Covariance Matrices
##
## data:  data_use[, -9]
## Chi-Sq (approx.) = 1463.3, df = 36, p-value < 2.2e-16
```

```
plot(boxm)
```

Conclusiones:

- El p-valor significativo, indica que debemos rechazar la hipótesis nula, y aceptar la hipótesis alternativa.
- Los datos indican que la matriz de covarianza no es estable para los dos grupos.

Los resultados obtenidos indican que sería apropiado aplicar QDA en lugar de LDA, pero en este caso ante la falta de normalidad multivariante en los datos, el test podría verse visto afectado por ello.

Uso del paquete MASS

Ejecutare la función lda del paquete MASS, obtendre los valores de accuracy de cada fold para 10-Fold CV, y luego los compararé con los resultados de KNN y QDA.

```
runLDA <- function(data, data_folds, response_name, predict_on = "test") {

  folds <- length(data_folds)

    # Variables para almacenar los resultados
```

```
    accuracy_list <- list()

    # Itero por la cantidad de folds
    for(i in 1:folds){

      # Obtengo los indices y
      fold_index <- data_folds[[i]]

      # Separo el dataset en train y test
      X_train <- data[-fold_index, ]
      X_test <- data[fold_index, ]
      y_train <- data[-fold_index, response_name]
      y_test <- data[fold_index, response_name]

      # Para ver si calculo el error de train o test
      if(predict_on == "test") {
        lda.fit <- lda(Y~., data=X_train)
        lda.pred <- predict(lda.fit, X_test)
        cm <- as.matrix(table(Predicted = lda.pred$class, Actual = y_test))
      } else { # predict on train
        lda.fit <- lda(Y~., data=X_train)
        lda.pred <- predict(lda.fit, X_train)
        cm <- as.matrix(table(Predicted = lda.pred$class, Actual = y_train))
      }

      # Calculo estadísticas del fold
      accuracy <- sum(diag(cm))/sum(cm)
      accuracy_list[[i]] <- accuracy
    }
  accuracy_list
}

# Obtengo métricas de cross validation para test
lda_accuracy_comparar <- unlist(runLDA(data_use, data_folds_C, "Y", predict_on = "test"))
lda_accuracy_comparar
```

```
##  [1] 1.0000000 0.9852941 0.9420290 0.9852941 0.9710145 0.9705882 0.9264706
##  [8] 1.0000000 0.9411765 0.9552239
```

Repito para el dataset que solo fue normalizado como preprocesamiento

```
# Obtengo métricas de cross validation para test
lda_accuracy_comparar_no_preprocesado <- unlist(runLDA(datasetC_sin_procesar,
                                                       data_folds_C,
                                                       "Y",
                                                       predict_on = "test"))
lda_accuracy_comparar_no_preprocesado
```

```
##  [1] 1.0000000 0.9705882 0.9275362 0.9558824 0.9565217 0.9705882 0.9264706
##  [8] 1.0000000 0.9558824 0.9402985
```

Uso del paquete caret

Utilizo la implementación de LDA en el paquete caret.

```r
# 10-Fold cross validation
# Uso las mismas particiones que en los casos anteriores
# Entreno el modelo
model_lda_caret <- train(Y ~ .,
             method     = "lda",
             trControl  = trControl,
             metric     = "Accuracy",
             data       = data_use
             )
model_lda_caret
```

```
## Linear Discriminant Analysis
##
## 682 samples
##   8 predictor
##   2 classes: '2', '4'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 69, 68, 69, 68, 69, 68, ...
## Resampling results:
##
##   Accuracy   Kappa
##   0.9563377  0.9035463
```

```r
summary(model_lda_caret)
```

```
##             Length Class      Mode
## prior        2     -none-     numeric
## counts       2     -none-     numeric
## means       16     -none-     numeric
## scaling      8     -none-     numeric
## lev          2     -none-     character
## svd          1     -none-     numeric
## N            1     -none-     numeric
## call         3     -none-     call
## xNames       8     -none-     character
## problemType  1     -none-     character
## tuneValue    1     data.frame list
## obsLevels    2     -none-     character
## param        0     -none-     list
```

```r
print(paste0("Method: ", model_lda_caret$method, " -- 10-fold CV Mean Accuracy = ",
             round(100*model_lda_caret$results$Accuracy, 3), "%"))
```

```
## [1] "Method: lda -- 10-fold CV Mean Accuracy = 95.634%"
```

QDA

Uso del paquete MASS

Ejecutare la función qda del paquete MASS, obtendre los valores de accuracy de cada fold para 10-Fold CV, y luego los compararé con los resultados de k-NN y LDA.

```r
runQDA <- function(data, data_folds, response_name, predict_on = "test") {

  folds <- length(data_folds)

    # Variables para almacenar los resultados
    accuracy_list <- list()

    # Itero por la cantidad de folds
    for(i in 1:folds){

      # Obtengo los indices y
      fold_index <- data_folds[[i]]

      # Separo el dataset en train y test
      X_train <- data[-fold_index, ]
      X_test <- data[fold_index, ]
      y_train <- data[-fold_index, response_name]
      y_test <- data[fold_index, response_name]

      # Para ver si calculo el error de train o test
      if(predict_on == "test") {
        qda.fit <- qda(Y~., data=X_train)
        qda.pred <- predict(qda.fit, X_test)
        cm <- as.matrix(table(Predicted = qda.pred$class, Actual = y_test))
      } else { # predict on train
        qda.fit <- qda(Y~., data=X_train)
        qda.pred <- predict(qda.fit, X_train)
        cm <- as.matrix(table(Predicted = qda.pred$class, Actual = y_train))
      }

      # Calculo estadísticas del fold
      accuracy <- sum(diag(cm))/sum(cm)
      accuracy_list[[i]] <- accuracy
    }
  accuracy_list
}

# Obtengo métricas de cross validation para test
qda_accuracy_comparar <- unlist(runQDA(data_use, data_folds_C, "Y", predict_on = "test"))
qda_accuracy_comparar
```

```
##  [1] 1.0000000 0.9558824 0.9855072 0.9852941 0.9420290 0.9705882 0.9264706
##  [8] 0.9264706 0.9705882 0.9104478
```

Repito para el dataset que solo fue normalizado como preprocesamiento

```r
# Obtengo métricas de cross validation para test
qda_accuracy_comparar_no_preprocesado <- unlist(runQDA(datasetC_sin_procesar,
                                                       data_folds_C,
                                                       "Y",
                                                       predict_on = "test"))
qda_accuracy_comparar_no_preprocesado
```

```
##  [1] 1.0000000 0.9558824 0.9710145 0.9411765 0.9130435 0.9705882 0.8970588
##  [8] 0.9411765 0.9705882 0.9253731
```

Uso del paquete caret

```
# 10-Fold cross validation
# Uso las mismas particiones que en los modelos anteriores
# Entreno el modelo
model_qda_caret <- train(Y ~ .,
             method     = "qda",
             trControl  = trControl,
             metric     = "Accuracy",
             data       = data_use)
model_qda_caret
```

```
## Quadratic Discriminant Analysis
##
## 682 samples
##   8 predictor
##   2 classes: '2', '4'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 69, 68, 69, 68, 69, 68, ...
## Resampling results:
##
##   Accuracy  Kappa
##   0.944067  0.8816465
```

```
summary(model_qda_caret)
```

```
##             Length Class      Mode
## prior         2    -none-     numeric
## counts        2    -none-     numeric
## means        16    -none-     numeric
## scaling     128    -none-     numeric
## ldet          2    -none-     numeric
## lev           2    -none-     character
## N             1    -none-     numeric
## call          3    -none-     call
## xNames        8    -none-     character
## problemType   1    -none-     character
## tuneValue     1    data.frame list
## obsLevels     2    -none-     character
## param         0    -none-     list
```

```
print(paste0("Method: ", model_qda_caret$method, " -- 10-fold CV Mean Accuracy = ",
             round(100*model_qda_caret$results$Accuracy, 3), "%"))
```

```
## [1] "Method: qda -- 10-fold CV Mean Accuracy = 94.407%"
```

Comparación de modelos

Comparación de modelos obtenidos a mano

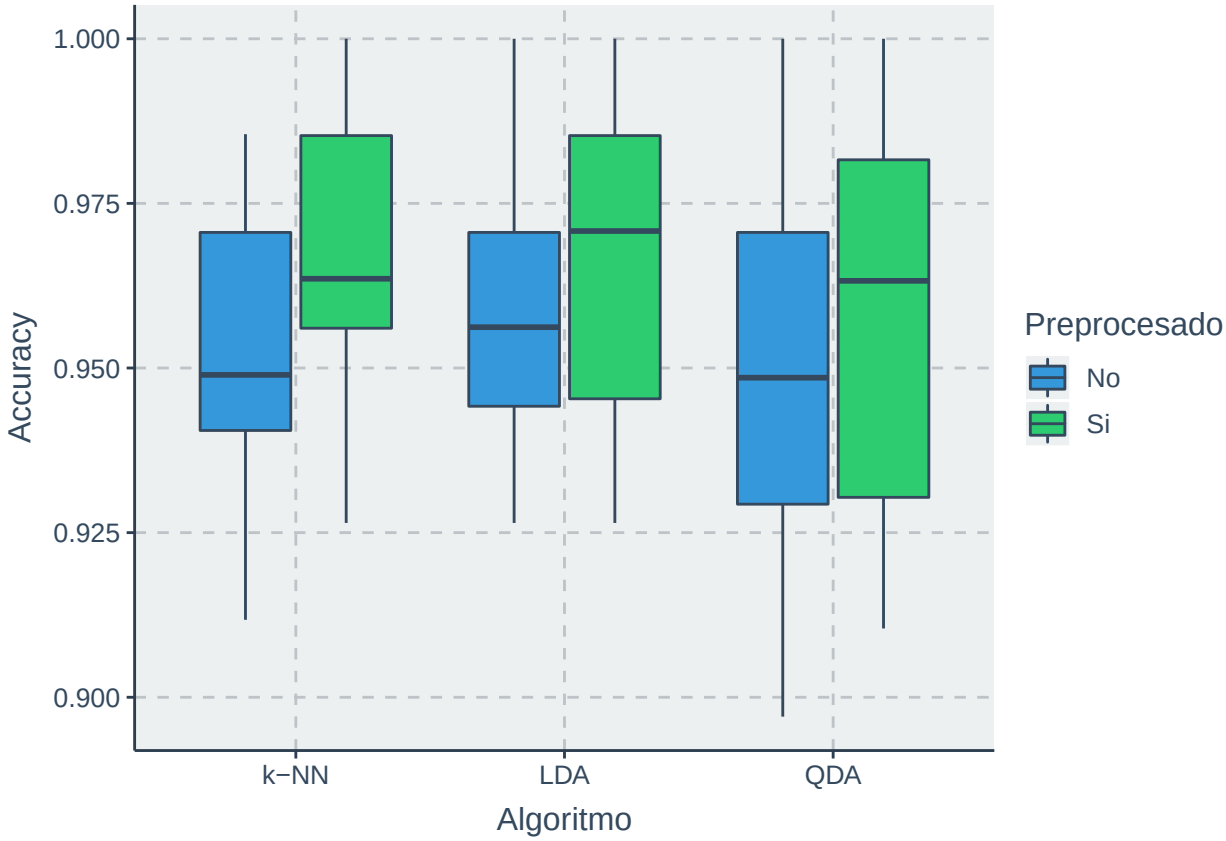
A continuación realizare una comparación de los modelos de k-NN, LDA y QDA obtenidos de forma manual, es decir sin utilizar caret. Compararé los resultados de 10-Fold Cross Validation de cada algoritmo sobre el

dataset preprocesado, y el que como preprocesamiento solo recibio normalización.

```
# Genero vectores de resultados para el dataframe
algoritmo <- c(rep("k-NN", 20),
               rep("LDA", 20),
               rep("QDA", 20))

preproc <- c(rep("Si", 10),
             rep("No", 10),
             rep("Si", 10),
             rep("No", 10),
             rep("Si", 10),
             rep("No", 10))

acc <- c(knn_accuracy_comparar,
         knn_accuracy_comparar_no_procesado,
         lda_accuracy_comparar,
         lda_accuracy_comparar_no_preprocesado,
         qda_accuracy_comparar,
         qda_accuracy_comparar_no_preprocesado)

# Dataframe de resultados para graficar
df <- data.frame(Algoritmo = algoritmo,
                 Preprocesado = preproc,
                 Accuracy = acc)

# Grafico
ggplot(data = df, aes(x = Algoritmo, y = Accuracy, fill = Preprocesado)) + geom_boxplot()
```

Table 8: Comparación exactitud (accuracy) promedio 10-Fold CV

Algoritmo	Preprocesado	AccuracyPromedio
kNN	Yes	0.9691797
kNN	No	0.9501466
LDA	Yes	0.9677091
LDA	No	0.9603768
QDA	Yes	0.9573278
QDA	No	0.9485902




```
# Obtengo la exactitud promedio
mean.acc <- c(mean(unlist(knn_accuracy_comparar)),
         mean(unlist(knn_accuracy_comparar_no_procesado)),
         mean(unlist(lda_accuracy_comparar)),
         mean(unlist(lda_accuracy_comparar_no_preprocesado)),
         mean(unlist(qda_accuracy_comparar)),
         mean(unlist(qda_accuracy_comparar_no_preprocesado)))

# Genero un dataframe para mostrar como tabla
df <- data.frame(Algoritmo = c("kNN", "kNN", "LDA", "LDA", "QDA", "QDA"),
                 Preprocesado = c("Yes", "No", "Yes", "No", "Yes", "No"),
                 AccuracyPromedio = mean.acc)

kable(df, format = "latex", booktabs = TRUE,
      caption = "\\label{tab:tab4}Comparación exactitud (accuracy) promedio 10-Fold CV")
```

Conclusiones

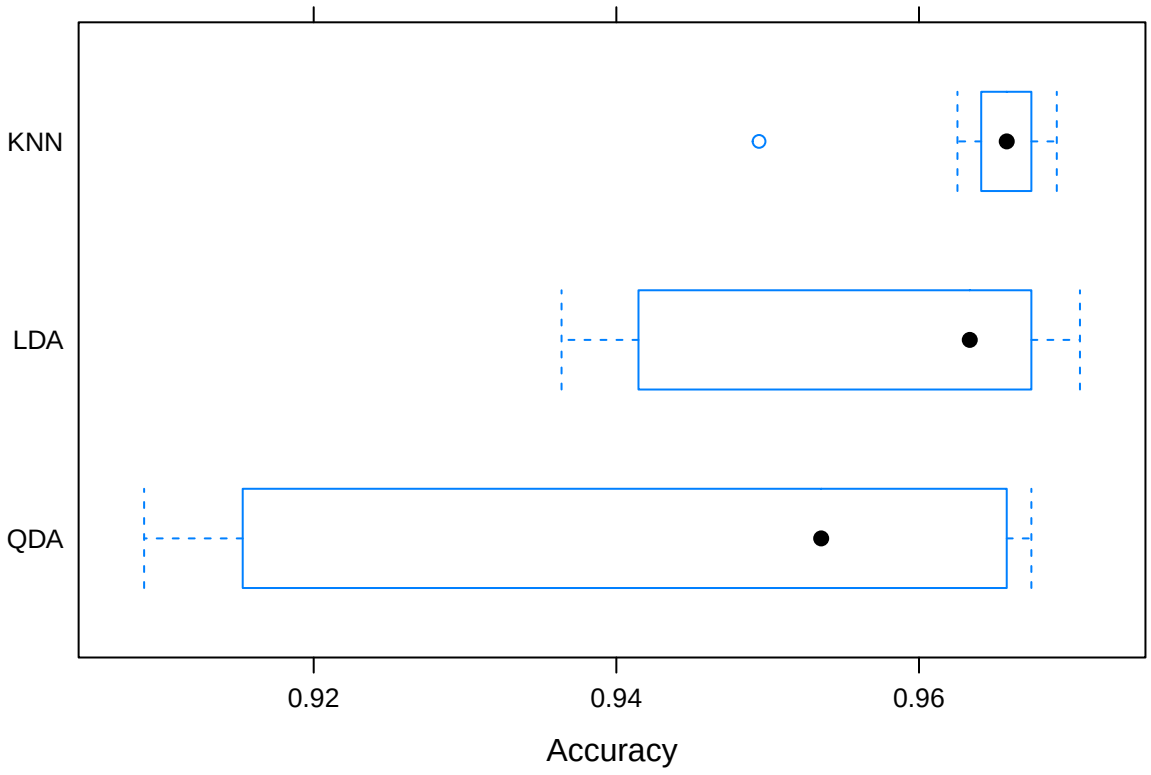
- El preprocesamiento de los datos produjos mejores resultados, teniendo en cuenta la exactitud (accuracy) realizando 10-fold cross validation, para los 3 algoritmos probados (tabla 8).
- k-NN obtuvo la mayor exactitud (accuracy) media en 10-Fold cross validation, seguido de LDA y por último QDA.

Comparación de modelos obtenidos con caret

```
# Modelos generados con caret
model_list <- list(KNN = model_knn_caret,
                   LDA= model_lda_caret,
                   QDA= model_qda_caret)

# Obtengo estadísticas del 10-Fold CV
resamples <- resamples(model_list)

# Printeo la media de Accuracy obtenida en 10-Fold CV
summary(resamples)$statistics$Accuracy[,4]
```

```
##       KNN       LDA       QDA
## 0.9644808 0.9563377 0.9440670
```

```
# Boxplot de resultados
bwplot(resamples, metric = "Accuracy")
```



Conclusiones

Al igual que con los modelos manuales, k-NN obtuvo la mayor exactitud (accuracy) media en 10-Fold cross

Table 9: Errores medios de test para los algoritmos k-NN, LDA y QDA

	out_train_lm	out_train_kknn	out_train_m5p
abalone	4.820000e+00	2.220000e+00	4.250000e+00
ANACALT	1.699982e-01	6.250400e-03	5.900000e-03
autoMPG6	1.129000e+01	3.530000e+00	6.870000e+00
autoMPG8	1.079000e+01	3.550000e+00	6.600000e+00
baseball	4.481590e+05	2.020880e+05	3.925890e+05
california	4.826190e+09	1.560869e+09	2.558518e+09
concrete	1.070000e+02	2.870000e+01	3.000000e+01
dee	1.618800e-01	7.611000e-02	1.620100e-01
delta_ail	0.000000e+00	0.000000e+00	0.000000e+00
delta_elv	2.100000e-06	1.000000e-06	2.000000e-06
forestFires	3.945000e+03	2.206000e+03	3.980000e+03
friedman	7.230000e+00	1.420000e+00	4.360000e+00
house	2.061567e+09	5.259870e+08	9.384299e+08
mortgage	1.354300e-02	8.827000e-03	1.101500e-02
stock	5.350000e+00	1.800000e-01	5.900000e-01
treasury	5.520300e-02	1.599800e-02	4.040400e-02
wankara	2.430000e+00	2.740000e+00	1.510000e+00
wizmir	1.565000e+00	2.538000e+00	1.358000e+00

validation, seguido de LDA y por último QDA. Además los resultados obtenidos con k-NN fueron más estables en los distintos folds de cross validation. QDA mostro la mayor varianza en los resultados de cada fold de cross validation.

Comparación estadística de algoritmos clasficación

Procedere a comparar los 3 diferentes algoritmos empleados en el trabajo.

```
# Lectura de los datos

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

kable(tablatra, format = "latex", booktabs = TRUE, caption =
        "\\label{tab:tabCACTST}Errores medios de test para los algoritmos k-NN, LDA y QDA")
```

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

kable(tablatra, format = "latex", booktabs = TRUE, caption =
        "\\label{tab:tabCACTRA}Errores medios de train para los algoritmos k-NN, LDA y QDA")
```

Table 10: Errores medios de train para los algoritmos k-NN, LDA y QDA

	out_train_knn	out_train_lda	out_train_qda
appendicitis	0.8834602	0.8815461	0.8690241
australian	0.7277419	0.8605475	0.8072464
balance	0.9072122	0.8791122	0.9167999
bupa	0.7405521	0.7024224	0.6447628
contraceptive	0.6168944	0.5236485	0.5314180
haberman	0.7795116	0.7519934	0.7567115
hayes-roth	0.6475524	0.5604167	0.7361111
heart	0.7342975	0.8576132	0.8777778
iris	0.9791045	0.9800000	0.9814815
led7digit	0.7636971	0.7635556	0.7680556
mammographic	0.8160856	0.8274465	0.8196843
monk-2	0.9793684	0.7821826	0.9303010
newthyroid	0.9158409	0.9183457	0.9700283
pima	0.7791914	0.7792266	0.7633125
tae	0.5263460	0.5584858	0.5688072
titanic	0.7892319	0.7760111	0.7732851
vehicle	0.7213300	0.7989229	0.9123989
vowel	0.8378652	0.6457912	0.9701459
wine	0.7745126	1.0000000	0.9956250
wisconsin	0.9739304	0.9614471	0.9588436

Wilcoxon

Primero realizare una comparativa de a pares entre los diferentes algoritmos utilizando el test de Wilcoxon.

k-NN vs LDA

Datos de test

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

```
##  V
## 90
```

```
Rmenos
```

```
##   V
## 120
```

```
pvalue
```

```
## [1] 0.5958195
```

- No existen diferencias significativas entre ambos algoritmos.
- Sólo hay un (1-0.5958) *100 = 40.4% de confianza en que sean distintos

Datos de train

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

```
##   V
## 118
```

```
Rmenos
```

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

```
pvalue
```

```
## [1] 0.6476555
```

- No existen diferencias significativas entre ambos algoritmos.
- Sólo hay un (1-0.6476) *100 = 35.2% de confianza en que sean distintos

k-NN vs QDA

Datos de test

```
KNNvsQDAtst <- wilcox.test(tablatst[,1], tablatst[,3], alternative = "two.sided", paired=TRUE)
Rmas <- KNNvsQDAtst$statistic
pvalue <- KNNvsQDAtst$p.value
QDAvsKNNtst <- wilcox.test(tablatst[,3], tablatst[,1], alternative = "two.sided", paired=TRUE)
Rmenos <- QDAvsKNNtst$statistic
Rmas
```

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

```
Rmenos
```

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

```
pvalue
```

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

- No existen diferencias significativas entre ambos algoritmos.
- hay un (1-0.1769) *100 = 82.3% de confianza en que sean distintos

Datos de train

```
KNNvsQDAtra <- wilcox.test(tablatra[,1], tablatra[,3], alternative = "two.sided", paired=TRUE)
Rmas <- KNNvsQDAtra$statistic
pvalue <- KNNvsQDAtra$p.value
QDAvsKNNtra <- wilcox.test(tablatra[,3], tablatra[,1], alternative = "two.sided", paired=TRUE)
Rmenos <- QDAvsKNNtra$statistic
Rmas
```

```
##  V
## 76
```

```
Rmenos
```

```
##   V
## 134
```

```
pvalue
```

```
## [1] 0.2942524
```

- No existen diferencias significativas entre ambos algoritmos.
- hay un (1-0.2942) *100 = 70.6% de confianza en que sean distintos

LDA vs QDA

Datos de test

```
LDAvsQDAtst <- wilcox.test(tablatst[,2], tablatst[,3], alternative = "two.sided", paired=TRUE)
Rmas <- LDAvsQDAtst$statistic
pvalue <- LDAvsQDAtst$p.value
QDAvsLDAtst <- wilcox.test(tablatst[,3], tablatst[,2], alternative = "two.sided", paired=TRUE)
Rmenos <- QDAvsLDAtst$statistic
Rmas
```

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

```
Rmenos
```

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

```
pvalue
```

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

- No existen diferencias significativas entre ambos algoritmos.
- hay un (1-0.8408) *100 = 15.9% de confianza en que sean distintos

Datos de train

```
LDAvsQDAtra <- wilcox.test(tablatra[,2], tablatra[,3], alternative = "two.sided", paired=TRUE)
Rmas <- LDAvsQDAtra$statistic
pvalue <- LDAvsQDAtra$p.value
QDAvsLDAtra <- wilcox.test(tablatra[,3], tablatra[,2], alternative = "two.sided", paired=TRUE)
Rmenos <- QDAvsLDAtra$statistic
Rmas
```

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

```
Rmenos
```

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

```
pvalue
```

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

- No existen diferencias significativas entre ambos algoritmos.
- Hay un (1-0.1768) *100 = 82.3% de confianza en que sean distintos

Comparativas múltiples

Usaremos Friedman y como post-hoc Holm (los rankings se calculan por posiciones de los algoritmos para cada problema y no hace falta normalización)

Test de Friedman **Datos de test**

```
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
```

- El p-valor de 0.7 indica que debemos aceptar la hipótesis nula, no existen diferencias significativas entre los 3 algoritmos.

Datos de train

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

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

- El p-valor de 0.5 indica que debemos aceptar la hipótesis nula, no existen diferencias significativas entre los 3 algoritmos.

Procederemos a aplicar el test de Holm, para corroborar lo que indica el test de Friedman, aunque no es necesario en este caso.

Datos de test

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

```
##
##  Pairwise comparisons using Wilcoxon signed rank test
##
## data:  as.matrix(tablatst) and groups
##
##   1    2
## 2 1.00 -
## 3 0.53 1.00
##
## P value adjustment method: holm
```

- No existen diferencias significativas en favor de ninguno de los 3 algoritmos.

Datos de train

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

```
##
##  Pairwise comparisons using Wilcoxon signed rank test
##
## data:  as.matrix(tablatra) and groups
##
##   1    2
## 2 0.65 -
## 3 0.59 0.53
##
## P value adjustment method: holm
```

- No existen diferencias significativas en favor de ninguno de los 3 algoritmos.

Apéndice 1

Librerías necesarias para el análisis

```
library(ggplot2)
library(reshape2)
library(ISLR)
library(caret)
library(reshape2)
library(corrplot)
library(Hmisc)
library(car)
library(MVN)
library(dplyr)
library(class)
library(kknn)
library(VIM)
library(heplots)
library(MASS)
library(GGally)
library(kableExtra)
library(ggthemr)
library(knitr)

# Tema para gráficos con ggplot2
ggthemr('flat')
```