Introducción a la Ciencia de Datos

Regresión y Clasificación

ugr | Universidad de Granada

Máster en Ciencia de Datos e Ingeniería de los Computadores

Besay Montesdeoca Santana Graduado en Ingeniería Informática

REGRESIÓN

1- Cálculo de media, desviación estándar, etc.

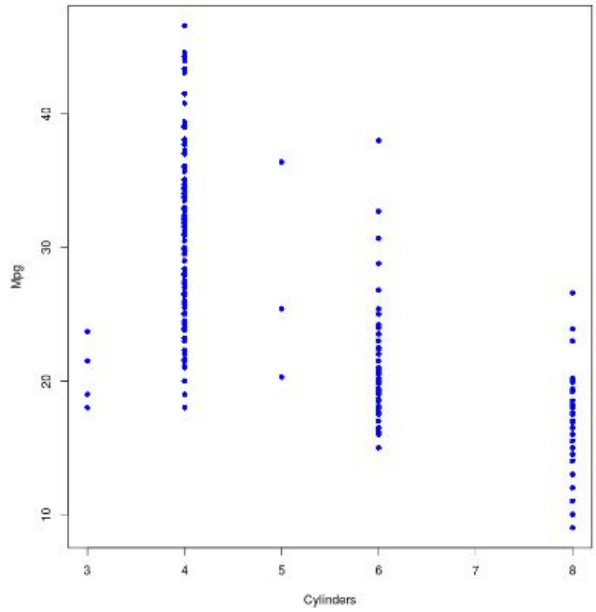
```
# Primero que nada leemos correctamente nuestro dataset y le añadimos los nombres de cada variable
autoFull <- read.csv("autoMPG8/autoMPG8.dat", comment.char="@")
names(autoFull) = c("Cylinders", "Displacement", "Horse_power", "Weight", "Acceleration", "Model_year", "Origin", "Mpg")
# Función que calcula todas las medidas estadísticas
analisisDatos = function(x){
  m = mean(x)
  s = sd(x)
  md = median(x)
  ma = max(x)
  mi = min(x)
  return(c(m,s, md, ma, mi))
}

a = apply(autoFull, 2, analisisDatos)
a = cbind(c("Media", "Desviación Estándar", "Mediana", "Máximo", "Mínimo"), a)
```

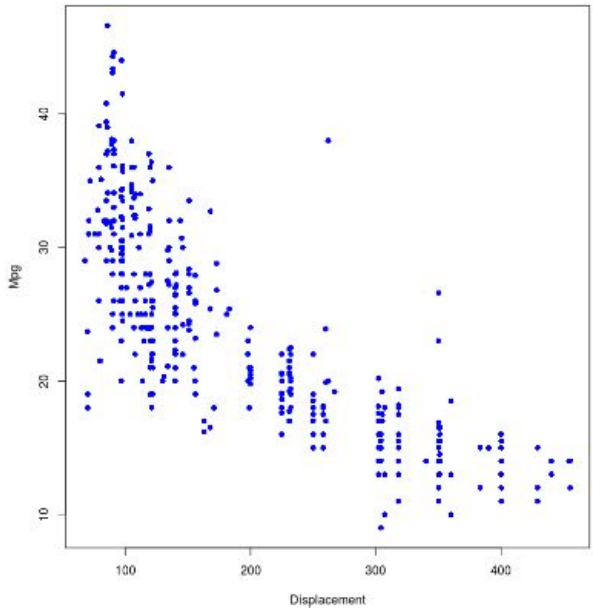
	Cylinders	Displacement	Horse_power	Weight
"Media"	"5.47570332480818"	"194.622762148338"	"104.511508951407"	"2978.53708439898"
"Desviación Estándar"	"1.7063374944868"	"104.694728486509"	"38.5314290034844"	"850.281020230847"
"Mediana"	"4"	"151"	"94"	"2807"
"Máximo"	"8"	"455"	"230"	"5140"
"Mínimo"	"3"	"68"	"46"	"1613"

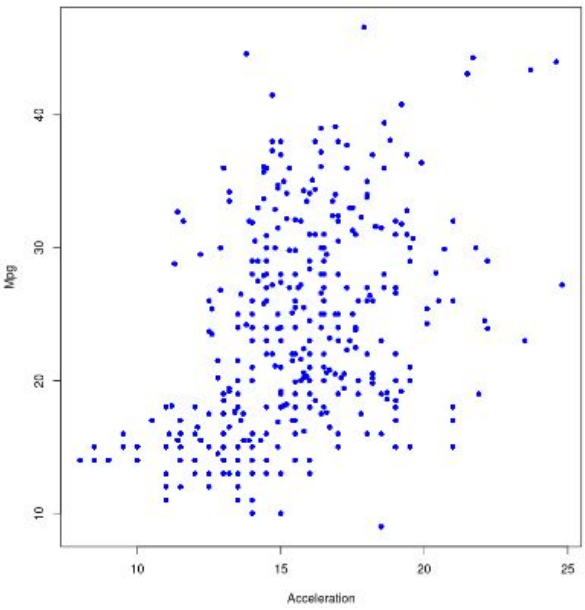
Acceleration	Model_year	Origin	Mpg
"15.5309462915601"	"75.9641943734015"	"1.57800511508951"	"23.4342710997442"
"2.75472347249747"	"3.67580401852913"	"0.806020366208571"	"7.81159571460068"
"15.5"	"76"	"1"	"22.5"
"24.8"	"82"	"3"	"46.6"
"8"	"70"	"1"	"9"

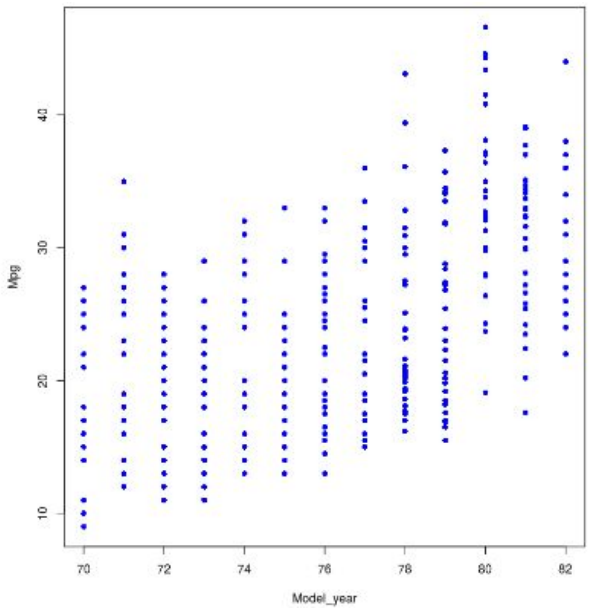
2- Gráficos que permitan visualizar los datos adecuadamente.

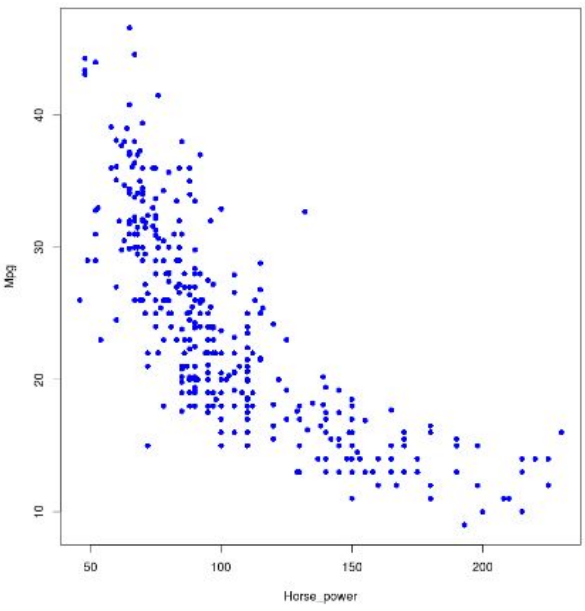
```
# Función para visualizar los datos mediante plot de forma más automática
temp <- autoFull
plotY <- function (x,y) {
 plot(temp[,y]~temp[,x], xlab=names(temp)[x], ylab=names(temp)[y], pch = 16, col = "Blue")
}
par(mfrow=c(3,3))
x <- sapply(1:(dim(temp)[2]-1), plotY, dim(temp)[2])
par(mfrow=c(1,1))
```

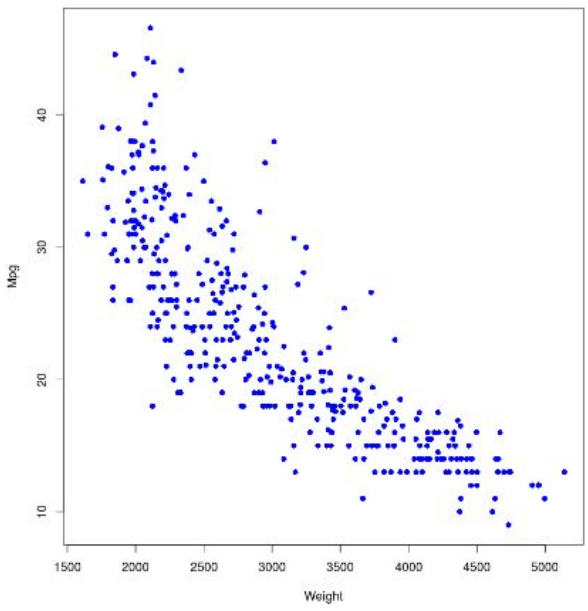

Mpg
10
20
30
40
3
4
5
6
7
8
Cylinders

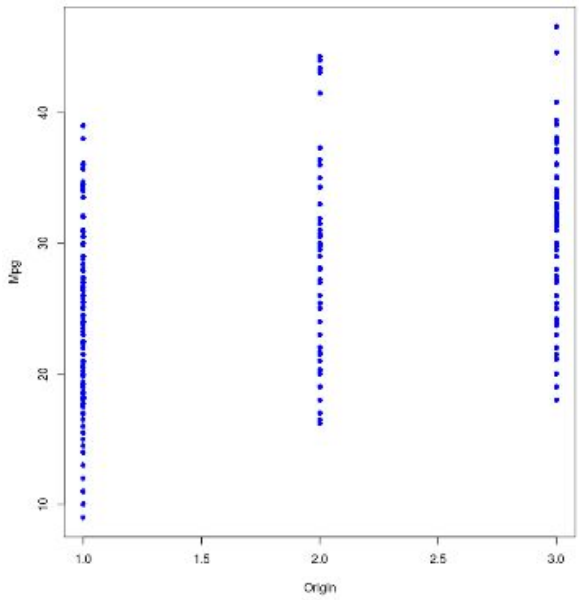

Mpg
10
20
30
40
100
200
300
400
Displacement


Mpg
10
20
30
40
10
15
20
25
Acceleration


Mpg
10
20
30
40
70
72
74
76
78
80
82
Model_year


Mpg
10
20
30
40
50
100
150
200
Horse_power


Mpg
10
20
30
40
1500
2000
2500
3000
3500
4000
4500
5000
Weight



3- Descripción del conjunto de datos a partir de los puntos anteriores.

Los datos se refieren al consumo de combustible del ciclo urbano en millas por galón. El objetivo consiste en predecir este valor a través de 7 atributos (número de cilindros, desplazamiento, caballos de potencia, peso, aceleración, año de modelo y origen).

Observando las gráficas se pueden apreciar que algunas de las variables son interesantes de cara a la construcción de los modelos. Por ejemplo las variables *displacements*, *weight* y *horsepower* tienen una evolución en la gráfica, a priori, susceptible de ser modelada con regresión linear o knn. En menor medida, la variable *acceleration* tambien podria ser una a tener en cuenta pero las otras restantes (*cylinders, model_year y origin*) si observamos sus gráficas se puede deducir que realmente son variables categóricas y por tanto difícilmente se podrán ajustar con rectas u otros métodos.

4- Regresión lineal Simple

```
# Asignación automática, facilita el acceso a los campos
n <- length(names(autoFull)) - 1
names(autoFull)[1:n] <- paste ("x", 1:n, sep="")
names(autoFull)[n+1] <- "y"
# Construimos los modelos con cada una de las variables
apply(autoFull[-8], 2, function(xn) summary(lm(y~xn)))
```

Modelo	Std.Err	t value	Pr(>|t|)	Res.Stand.Err	Ad.R-squared
Cylinders(x1)	0.146	-24.38	<2e-16	4.92	0.6033
Displacements(x2)	0.002	-26.76	<2e-16	4.641	0.6447
HorsePower(x3)	0.006	-24.45	<2e-16	4.911	0.6048
Weight(x4)	0.002	-29.60	<2e-16	4.337	0.6917
Acceleration(x5)	0.13	9.194	<2e-16	7.089	0.1764
ModelYear(x6)	0.087	14.05	<2e-16	6.37	0.3359
Origin(x7)	0.404	13.57	<2e-16	6.443	0.3196

Construimos los modelos lineales simples con cada una de las variables. En este punto es difícil determinar con exactitud qué variable será importante para la predicción sin realizar cálculos de errores más profundos con la parte de test. En la tabla observamos resaltados en colores los modelos, en un principio, más relevantes, y para llegar a esta conclusión nos hemos apoyado básicamente en la medida de Adjusted R-squared ya que nos dice el porcentaje escrutado o el porcentaje que conoce de la salida el modelo.

6- Regresión Lineal Múltiple

Para construir el modelo en un principio partimos de aquel que incluye todas las variables, y según el resultado de tanto los p-value de cada variable y del Adjusted R-square, fuimos descartando variables hasta llegar al modelo siguiente:

```
summary(lm(y~x4+x6+x7, data = autoFull))
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-9.9352 -2.1105 -0.0401  1.7219 13.2681

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.816e+01  4.012e+00  -4.526 8.02e-06 ***
x4          -5.998e-03  2.545e-04 -23.568  < 2e-16 ***
x6           7.590e-01  4.853e-02  15.640  < 2e-16 ***
x7           1.143e+00  2.599e-01   4.398 1.42e-05 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.351 on 387 degrees of freedom
Multiple R-squared:  0.8174,    Adjusted R-squared:  0.816
F-statistic: 577.5 on 3 and 387 DF,  p-value: < 2.2e-16
```

El siguiente paso fue investigar la no linealidad de la variables del modelo, y para ello nos hemos guiado por la evolución que tienen dichas variables en las gráficas anteriores, donde se observa cierta curvatura en los datos:

```
summary(lm(y~x4+I(x4^2)+x6+I(x6^2)+I(x6^3)+x7+I(x7^2), data = autoFull))
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-8.5091 -1.6561  0.0829  1.5320 12.5139

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.745e+03  1.595e+03   3.603 0.000356 ***
x4          -2.036e-02  1.470e-03 -13.847  < 2e-16 ***
I(x4^2)      2.242e-06  2.228e-07  10.063  < 2e-16 ***
x6          -2.217e+02  6.309e+01  -3.515 0.000493 ***
I(x6^2)      2.866e+00  8.311e-01   3.448 0.000627 ***
I(x6^3)     -1.228e-02  3.645e-03  -3.368 0.000835 ***
x7           4.882e+00  1.626e+00   3.003 0.002852 **
I(x7^2)     -1.105e+00  4.072e-01  -2.715 0.006932 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.859 on 383 degrees of freedom
Multiple R-squared:  0.8685,    Adjusted R-squared:  0.8661
F-statistic: 361.2 on 7 and 383 DF,  p-value: < 2.2e-16
```

Por último se probó las interacciones que tienen las variables entre ellas, para ello partimos de la variable que a priori era la más relevante (x4 = weight):

```
summary(lm(y~x4*x6, data = autoFull))
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-8.0432 -1.9850 -0.0943  1.6509 12.9677

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.108e+02  1.296e+01  -8.554 2.81e-16 ***
x4           2.760e-02  4.415e-03   6.251 1.08e-09 ***
x6           2.046e+00  1.720e-01  11.896  < 2e-16 ***
x4:x6       -4.586e-04  5.909e-05  -7.761 7.58e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.194 on 387 degrees of freedom
Multiple R-squared:  0.8341,    Adjusted R-squared:  0.8328
F-statistic: 648.6 on 3 and 387 DF,  p-value: < 2.2e-16
```

Aunque se ha obtenido una medida de Adjusted R-squared de 0.03 menos con respecto al último modelo, es evidente que este modelo es mucho más sencillo que el anterior y por tanto, en algunos casos puede que sea preferible.

7- Aplicación de k-NN

En este apartado se aplicará el método de lo k-vecinos más cercanos sobre el dataset para realizar la predicción de los datos. En este caso se realizará el entrenamiento y el cálculo de errores con la técnica de la validación cruzada para obtener unas medidas de error más fiables.

De paso se calcularán los errores para el modelo lineal múltiple con el objetivo realizar una primera comparación de los dos metodos de prediccion.

```
# Función donde se aplica el método de validación cruzada para realizar el
entrenamiento. Según el valor del parámetro alg
# se utilizará el algoritmo de regresión lineal o k-nn
nombre <- "autoMPG8/autoMPG8"
crossV_lm_knn <- function(i, x, tt = "test", alg) {
 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
 }
 if (alg == 0){
   fitMulti = lm(y~.,x_tra)
   yprime = predict(fitMulti,test)
   sum(abs(test$y-yprime)^2)/length(yprime) ##MSE
 }else{
   fitMulti=kknn(y~.,x_tra,test)
   yprime=fitMulti$fitted.values
   sum(abs(test$y-yprime)^2)/length(yprime) ##MSE
 }
}

lmMSEtrain = mean(sapply(1:5,crossV_lm_knn,nombre,"train", 0))
lmMSEtest = mean(sapply(1:5,crossV_lm_knn,nombre,"test", 0))
print(c(paste("lmMSEtrain =", lmMSEtrain, "lmMSEtest =", lmMSEtest, sep = " ")))
```

```
knnMSEtrain = mean(sapply(1:5,crossV_lm_knn,nombre, "train", alg = 1))
knnMSEtest = mean(sapply(1:5,crossV_lm_knn,nombre, "test", alg = 1))
print(c(paste("knnMSEtrain =", knnMSEtrain, "knnMSEtest =", knnMSEtest, sep = " ")))
```

	Train	Test
lm	7.984	8.389
k-NN	3.551	8.106

Podemos observar que el método knn ofrece una tasa de error medio cuadrático menor con respecto al modelo lineal múltiple, aunque si observamos el error MSE del modelo knn en la parte de entrenamiento podemos intuir que realiza un modelo quizás demasiado ajustado a los datos.

8- Comparativa de algoritmos

En primer lugar se compararán el algoritmo de regresión lineal contra el algoritmo de los k-vecinos más cercanos. Para ello utilizaremos el conocido test de Wilcoxon. Los datos que se analizarán son errores medios cuadráticos obtenidos al aplicar dichos algoritmos sobre 18 bases de datos diferentes bajo las mismas condiciones.

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

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

# Comparativa por pares
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])
rownames(wilc_1_2) = c(rownames(tablatst))

# Aplicamos el test de Wilcoxon
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
print(paste("LM(R+) vs KNN(R-) -> R+ =", Rmas, "R- =", Rmenos, "p-value =", pvalue))
```

	R+	R-	p-value
LM(R+) vs KNN(R-)	78	93	0.766029357910156

Observando los resultados del test podemos afirmar que no existen diferencias significativas entre ambos algoritmos, puesto que según el resultado del p-value, solo hay una confianza de 23.4% de que los algoritmos comparados sean diferentes.

En la segunda parte de este apartado realizaremos una comparativa múltiple donde se implicarán los dos algoritmos anteriores junto con otro nuevo(M5). En este caso al haber más de dos algoritmos no se puede utilizar el test anterior. Para esta comparativa se utilizará el test de Friedman seguido de post-hoc Holm.

```
# Realizamos el test de Friedman
test_friedman = friedman.test(as.matrix(tablatst))

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

```
        Friedman rank sum test

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

Este resultado nos indica que existen diferencias significativas al menos entre dos de los algoritmos.

Cuando aplicamos post-hoc Holm obtenemos los siguiente:

```
  1     2
2 0.580 -
3 0.081 0.108

P value adjustment method: holm
```

Técnicamente no existen diferencias significativas entre los algoritmos, pero si es cierto que existen ciertas diferencias significativas a favor del tercer algoritmo(M5) con aproximadamente 90% de confianza.

CLASIFICACIÓN

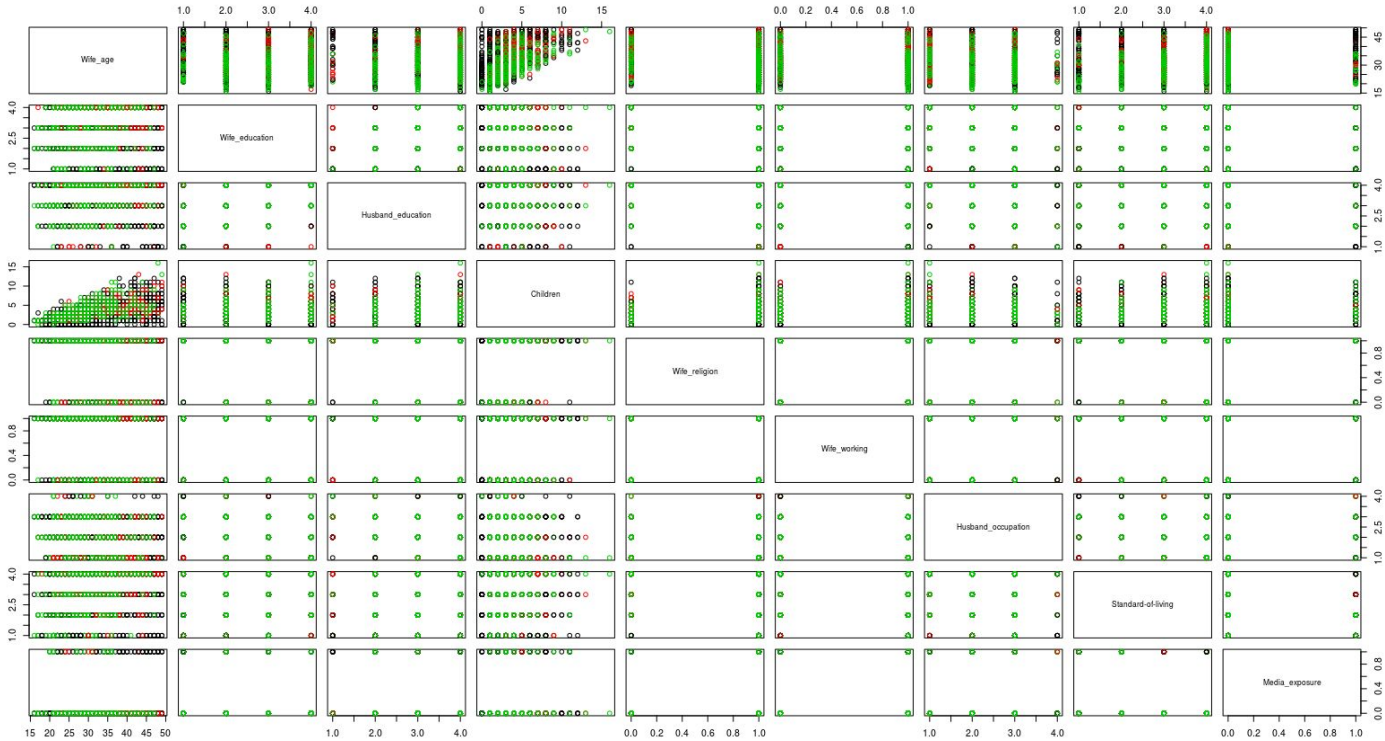
1- Cálculo de media, desviación estándar, etc.

```
# Leemos el dataset y añadimos los nombres de las estadisticas variables
con <- read.csv("contraceptive/contraceptive.dat", comment.char="@")
names(con) = c("Wife_age", "Wife_education", "Husband_education", "Children",
"Wife_religion", "Wife_working", "Husband_occupation", "Standard-of-living",
"Media_exposure", "Contraceptive_method")
a = apply(con, 2, analisisDatos)
a = cbind(c("Media", "Desviación Estándar", "Mediana", "Máximo", "Mínimo"), a)
```

```
                    Wife_age            Wife_education      Husband_education   Children
"Media"             "32.5441576086956"  "2.95923913043478"  "3.43002717391304"  "3.26154891304348"
"Desviación Estándar" "8.22702721037277" "1.01503126836001" "0.816549087676513" "2.35934055997844"
"Mediana"           "32"                "3"                 "4"                 "3"
"Máximo"            "49"                "4"                 "4"                 "16"
"Mínimo"            "16"                "1"                 "1"                 "0"
                    Children            Wife_religion       Wife_working        Husband_occupation
"Media"             "3.26154891304348"  "0.85054347826087"  "0.749320652173913" "2.13790760869565"
"Desviación Estándar" "2.35934055997844" "0.356659062348813" "0.433551504794334" "0.865143722782841"
"Mediana"           "3"                 "1"                 "1"                 "2"
"Máximo"            "16"                "1"                 "1"                 "4"
"Mínimo"            "0"                 "0"                 "0"                 "1"
                    Husband_occupation  Standard-of-living  Media_exposure
"Media"             "2.13790760869565"  "3.13383152173913"  "0.0740489130434783"
"Desviación Estándar" "0.865143722782841" "0.976486017495368" "0.261939464621011"
"Mediana"           "2"                 "3"                 "0"
"Máximo"            "4"                 "4"                 "1"
"Mínimo"            "1"                 "1"                 "0"
```

2- Gráficos que permitan visualizar los datos adecuadamente.

```
pairs(con[-10], col = con[,10])
```



3- Descripción del conjunto de datos a partir de los puntos anteriores.

Los datos de los que disponemos para realizar las distintas tareas de clasificación se refieren a muestras de mujeres casadas que no estaban embarazadas o no sabían si estaban en el momento de la entrevista. El objetivo es predecir el tipo de método anticonceptivo basándonos en una serie de características demográficas y socioeconómicas como pueden ser la edad, si ella o su pareja han recibido educación, el número de hijos, si actualmente trabaja y demás.

Observando las gráficas de todas las variables enfrentadas con todas es bastante difícil determinar qué algoritmo puede ir mejor para la clasificación, puesto que al tener tantas variables categóricas, visualmente no se aprecian agrupaciones de los datos por clase.

4- Aplicación del algoritmo k-NN.

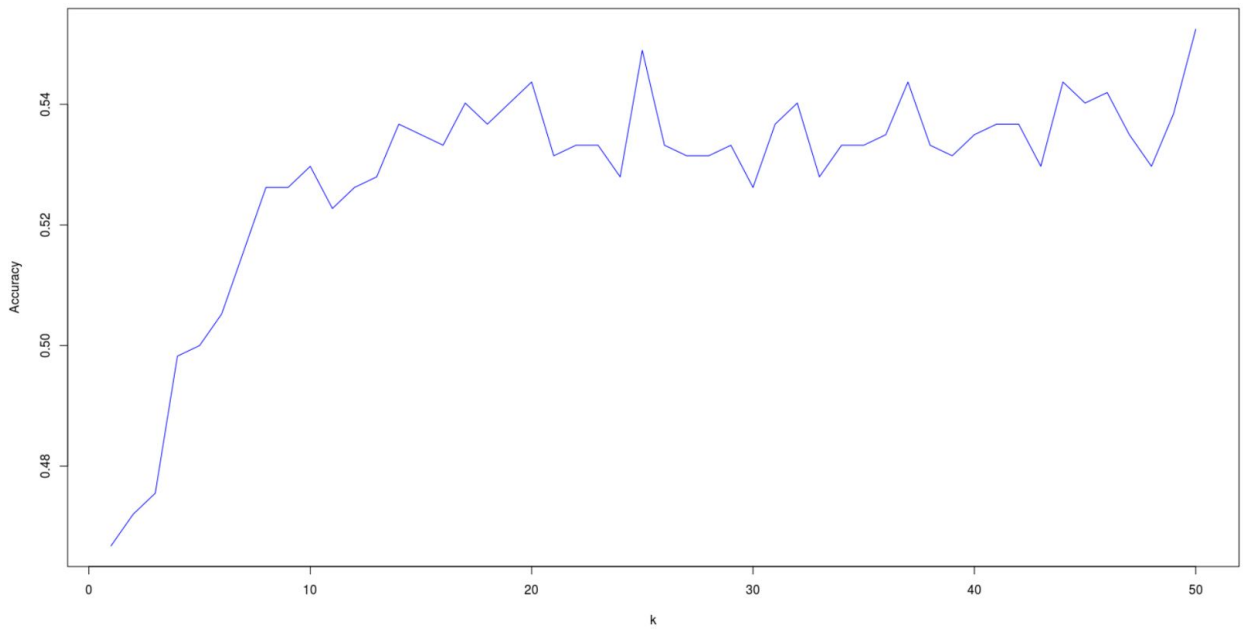
En esta sección probaremos el algoritmo de los k-vecinos más cercanos para realizar la clasificación del tipo de método anticonceptivo en nuestro dataset.

Pare ello nos apoyaremos en el paquete *Caret,* el cual dispone de un gran número de funciones que ayudan a agilizar el proceso de creación de modelos predictivos

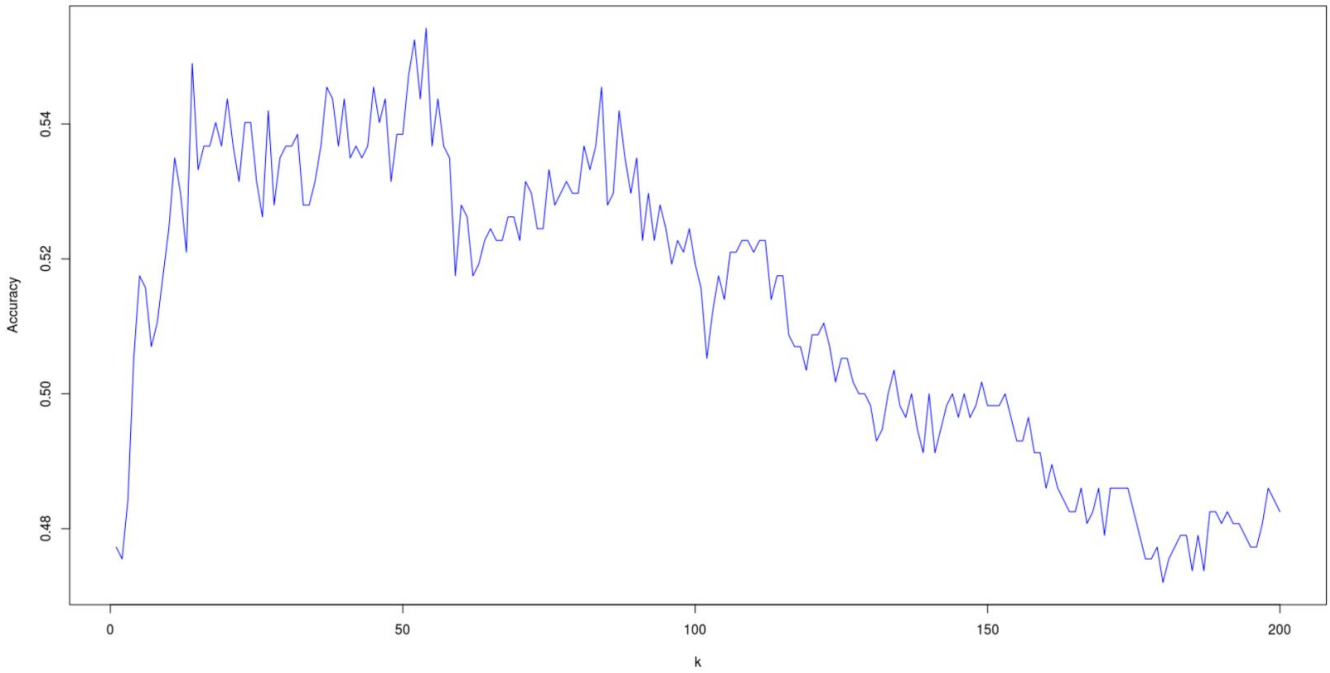
tanto de clasificación como de regresión.

Se realizará un pequeño estudio de la influencia de el número de vecinos(k) utilizados para la ejecución del algoritmo.

```
# División del conjunto de datos en train y test
con = con[sample(nrow(con)), ] # Shuffle de los datos
con_train = con[1:900,1:9]
con_test = con[901:1472,1:9]
con_train_labels = factor(con[1:900,10], levels = c(1, 2, 3), labels = c("No-use", "Long-term", "Short-term"))
con_test_labels = factor(con[901:1472,10], levels = c(1, 2, 3), labels = c("No-use", "Long-term", "Short-term"))


library(class)
require(caret)
# Función que realiza un entrenamiento con el algoritmo k-NN con un k especificado por parámetro. Devuelve la precisión(Accuracy) del modelo.
train_knn = function(nk){
 m = train(con_train, con_train_labels, method="knn", metric="Accuracy", tuneGrid = data.frame(.k=nk))
 knnPred <- predict(m, newdata = con_test)
 postResample(pred = knnPred, obs = con_test_labels)[1]
}
acc = sapply(1:50, train_knn)
plot(1:50, acc, type = "l", col = "blue", xlab = "k", ylab = "Accuracy")
```



Observamos que a partir de un k en torno a 15 el algoritmo ofrece porcentajes de precisión similares, pero como se ve en la siguiente gráfica, al aumentar el número de vecinos la precisión tiende a bajar.



En definitiva, el valor *k* mas adecuado para este conjunto de datos estaría en torno a 15. Escoger un valor mayor no implicaría alcanzar mejores porcentajes de acierto sino que obtendríamos mayores tiempos de ejecución o incluso, para valores muy grandes de *k*, peores tasas de precisión.

```
# Mejor modelo knn con un valor de k en torno a 15
knnModel = train(con_train, con_train_labels, method="knn", metric="Accuracy",
tuneGrid = data.frame(.k=15))
knnPred <- predict(knnModel, newdata = con_test)
prop.table(table(knnPred, con_test_labels))
postResample(pred = knnPred, obs = con_test_labels)
```

```
> prop.table(table(knnPred, con_test_labels))
           con_test_labels
knnPred          No-use  Long-term Short-term
  No-use     0.24125874 0.05944056 0.07517483
  Long-term  0.04545455 0.09090909 0.06293706
  Short-term 0.13461538 0.06993007 0.22027972
> postResample(pred = knnPred, obs = con_test_labels)
 Accuracy     Kappa
0.5524476 0.3066338
```

5- Aplicación del algoritmo LDA

En este quinto apartado probaremos cómo de precisa es la clasificación usando el algoritmo LDA. Nuevamente utilizaremos el paquete caret pero además se entrenará aplicando el método de validación cruzada, el cual permite evaluar mejor los resultados y garantizar que son independientes de la partición entre los datos de entrenamiento y de prueba.

```
library(MASS)

library(ISLR)

library(klaR)

con_labels = factor(con[,10], levels = c(1, 2, 3), labels = c("No-use", "Long-term",
"Short-term"))


ldaModel <- train(con[-10], con_labels, method = "lda", preProcess = c("center",
"scale"), tuneLength = 10, trControl = trainControl(method = "cv"))

confusionMatrix(ldaModel)
```

```
Cross-Validated (10 fold) Confusion Matrix

(entries are percentual average cell counts across resamples)

            Reference
Prediction   No-use Long-term Short-term
  No-use       26.5       6.5       12.0
  Long-term     3.1       8.2        5.8
  Short-term   13.0       7.9       16.9

 Accuracy (average) : 0.5156
```

6- Aplicación del algoritmo QDA

En el sexto apartado probaremos la variante cuadrática del algoritmo anterior. Al igual que anteriormente se utilizara caret y la validación cruzada para entrenar al modelo QDA.

```
qdaModel <- train(con[-10], con_labels, method = "qda", preProcess = c("center",
"scale"), tuneLength = 10, trControl = trainControl(method = "cv"))

confusionMatrix(qdaModel)
```

```
Cross-Validated (10 fold) Confusion Matrix

(entries are percentual average cell counts across resamples)

            Reference
Prediction   No-use Long-term Short-term
  No-use       19.7       3.3        5.6
  Long-term    10.4      13.5       10.5
  Short-term   12.6       5.8       18.7

 Accuracy (average) : 0.5183
```

7- Comparación de algoritmos

En el último apartado realizaremos una comparación de los anteriores algoritmos, para ello ejecutaremos el test de Friedman seguido de post-hoc Holm puesto que queremos comparar más de dos algoritmos. Los datos que utilizaremos para realizar dicha comparación consisten en medidas de precisión tomadas de 20 dataset distintos sobre los cuales se ha aplicado los tres algoritmos en cuestión bajo las mismas condiciones.

```
# Lectura del dataset donde se encuentran el resultado de las pruebas
resultados = read.csv("clasif_test_alumnos.csv")
tablatst <- cbind(resultados[,2:dim(resultados)[2]])
colnames(tablatst) <- names(resultados)[2:dim(resultados)[2]]
rownames(tablatst) <- resultados[,1]


# Realizamos el test de Friedman
test_friedman = friedman.test(as.matrix(tablatst))
test_friedman
```

```
        Friedman rank sum test

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

Observamos que obtenemos un p-value superior a 0.05 lo que indica que no existen diferencias significativas entre ninguno de los algoritmos a comparar. En este punto es innecesario realizar post-hoc Holm puesto que ya sabemos que no hay diferencias pero aún así se realizará el test.

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

```
        Pairwise comparisons using Wilcoxon signed rank test

data:  as.matrix(tablatst) and groups

  1    2
2 1.00 -
3 0.53 1.00

P value adjustment method: holm
```

Como era de esperar no existen diferencias significativas al comparar cada algoritmo dos a dos.

APÉNDICE: CÓDIGO

```
#************#
# REGRESIÓN #
#************#


# Primero que nada leemos correctamente nuestro dataset y le añadimos los nombres de cada variable
autoFull <- read.csv("autoMPG8/autoMPG8.dat", comment.char="@")
names(autoFull) = c("Cylinders", "Displacement", "Horse_power", "Weight", "Acceleration", "Model_year", "Origin", "Mpg")


# Función que calcula todas las medidas estadísticas
analisisDatos = function(x){
  m = mean(x)
  s = sd(x)
  md = median(x)
  ma = max(x)
  mi = min(x)
  return(c(m,s, md, ma, mi))
}

a = apply(autoFull, 2, analisisDatos)
a = cbind(c("Media", "Desviación Estándar", "Mediana", "Máximo", "Mínimo"), a)

# Función para visualizar los datos mediante plot de forma más automática
temp <- autoFull
plotY <- function (x,y) {
 plot(temp[,y]~temp[,x], xlab=names(temp)[x], ylab=names(temp)[y], pch = 16, col = "Blue")
}

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

```r
# REGRESIÓN LINEAL SIMPLE

# Asignación automática, facilita el acceso a los campos
n <- length(names(autoFull)) - 1
names(autoFull)[1:n] <- paste ("x", 1:n, sep="")
names(autoFull)[n+1] <- "y"

# Construimos los modelos con cada una de las variables
apply(autoFull[-8], 2, function(xn) summary(lm(y~xn)))

# El mejor modelo basándonos en la medida de 'Adjusted R-squared'
fit1 = lm(y~x4)
print(summary(fit1))
plot(y~x4,autoFull, pch = 16, col = "blue", xlab = "Weight", ylab = "Mpg")
abline(fit1,col="red")

# Regresión Lineal Múltiple
summary(lm(y~x4+x6+x7, data = autoFull))
summary(lm(y~x4+I(x4^2)+x6+I(x6^2)+I(x6^3)+x7+I(x7^2), data = autoFull))
summary(lm(y~x4*x6, data = autoFull))


# REGRESIÓN k-NN

# Función donde se aplica el método de validación cruzada para realizar el
entrenamiento. Según el valor del parámetro alg
# se utilizará el algoritmo de regresión lineal o k-nn
nombre <- "autoMPG8/autoMPG8"
crossV_lm_knn <- function(i, x, tt = "test", alg) {
 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
 }
 if (alg == 0){
   fitMulti = lm(y~.,x_tra)
   yprime = predict(fitMulti,test)
   sum(abs(test$y-yprime)^2)/length(yprime) ##MSE
 }else{
   fitMulti=kknn(y~.,x_tra,test)
   yprime=fitMulti$fitted.values
   sum(abs(test$y-yprime)^2)/length(yprime) ##MSE
 }
}

lmMSEtrain = mean(sapply(1:5,crossV_lm_knn,nombre,"train", 0))
lmMSEtest = mean(sapply(1:5,crossV_lm_knn,nombre,"test", 0))
print(c(paste("lmMSEtrain =", lmMSEtrain, "lmMSEtest =", lmMSEtest, sep = " ")))

knnMSEtrain = mean(sapply(1:5,crossV_lm_knn,nombre, "train", alg = 1))
knnMSEtest = mean(sapply(1:5,crossV_lm_knn,nombre, "test", alg = 1))
print(c(paste("knnMSEtrain =", knnMSEtrain, "knnMSEtest =", knnMSEtest, sep = " ")))


# COMPARACIÓN DE ALGORITMOS

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

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

```r
colnames(tablatra) <- names(resultados)[2:dim(resultados)[2]]
rownames(tablatra) <- resultados[,1]

# Comparativa por pares
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])
rownames(wilc_1_2) = c(rownames(tablatst))

# Aplicamos el test de Wilcoxon
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
print(paste("LM(R+) vs KNN(R-) -> R+ =", Rmas, "R- =", Rmenos, "p-value =", pvalue))

# Realizamos el test de Friedman
test_friedman = friedman.test(as.matrix(tablatst))

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



#***************#
# CLASIFICACIÓN #
#***************#

# Leemos el dataset y añadimos los nombres de las estadisticas variables
con <- read.csv("contraceptive/contraceptive.dat", comment.char="@")

names(con) = c("Wife_age", "Wife_education", "Husband_education", "Children",
"Wife_religion",
```

```
                    "Wife_working", "Husband_occupation", "Standard-of-living",
"Media_exposure",
                    "Contraceptive_method")

a = apply(con, 2, analisisDatos)
a = cbind(c("Media", "Desviación Estándar", "Mediana", "Máximo", "Mínimo"), a)

pairs(con[-10], col = con[,10])



con_labels = factor(con[,10], levels = c(1, 2, 3), labels = c("No-use", "Long-term",
"Short-term"))

# k-NN

# División del conjunto de datos en train y test
con = con[sample(nrow(con)), ] # Shuffle de los datos
con_train = con[1:900,1:9]
con_test = con[901:1472,1:9]
con_train_labels = factor(con[1:900,10], levels = c(1, 2, 3), labels = c("No-use",
"Long-term", "Short-term"))
con_test_labels = factor(con[901:1472,10], levels = c(1, 2, 3), labels = c("No-use",
"Long-term", "Short-term"))

library(class)
require(caret)

# Función que realiza un entrenamiento con el algoritmo k-NN con un k especificado
por parámetro. Devuelve la
# precisión(Accuracy) del modelo.
train_knn = function(nk){
 m = train(con_train, con_train_labels, method="knn", metric="Accuracy", tuneGrid =
data.frame(.k=nk))
 knnPred <- predict(m, newdata = con_test)
 postResample(pred = knnPred, obs = con_test_labels)[1]
}

acc = sapply(1:50, train_knn)
```

```r
plot(1:50, acc, type = "l", col = "blue", xlab = "k", ylab = "Accuracy")

# Mejor modelo knn con un valor de k en torno a 15
knnModel = train(con_train, con_train_labels, method="knn", metric="Accuracy",
tuneGrid = data.frame(.k=15))
knnPred <- predict(knnModel, newdata = con_test)
prop.table(table(knnPred, con_test_labels))
postResample(pred = knnPred, obs = con_test_labels)

# LDA
library(MASS)
library(ISLR)
library(klaR)

con_labels = factor(con[,10], levels = c(1, 2, 3), labels = c("No-use", "Long-term",
"Short-term"))

ldaModel <- train(con[-10], con_labels,
                  method = "lda",
                  preProcess = c("center", "scale"),
                  tuneLength = 10,
                  trControl = trainControl(method = "cv"))
confusionMatrix(ldaModel)

# QDA
qdaModel <- train(con[-10], con_labels,
                    method = "qda",
                    preProcess = c("center", "scale"),
                    tuneLength = 10,
                    trControl = trainControl(method = "cv"))
confusionMatrix(qdaModel)

# COMPARACIÓN DE ALGORITMOS

# Lectura del dataset donde se encuentran el resultado de las pruebas
resultados = read.csv("clasif_test_alumnos.csv")
tablatst <- cbind(resultados[,2:dim(resultados)[2]])
```

```r
colnames(tablatst) <- names(resultados)[2:dim(resultados)[2]]
rownames(tablatst) <- resultados[,1]

# Realizamos el test de Friedman
test_friedman = friedman.test(as.matrix(tablatst))
test_friedman

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