Introducción a la Ciencia de Datos
Trabajo Teórico/Práctico Integrador

Cristian González Guerrero

24 de agosto de 2017

Índice

1. Introducción

Este es el Trabajo Teórico/Pŕactico Integrador de la asignatura *Introducción a la Ciencia de Datos*, impartida en el *Máster en Ciencia de Datos e Ingeniería de Computadores* (DATCOM). En el mismo, se desarrollan los ejercicios propuestos por los profesores, aplicando los conocimientos adquiridos en la asignatura.

La realización de este trabajo se ha llevado a cabo usando *RStudio* [1, 2], *en su versión 0.99.903 - R version 3.3.2 (2016-10-31)* [3]. La mayor parte de los gráficos han sido generados usando *ggplot2*[4, 5], *en su versión 2.2.1*, lo cuál se ha considerado una ampliación del apartado de visualización.

Este trabajo, realizado de forma original por Cristian González Guerrero, se presenta para su valoración en la evaluación extraordinaria de Septiembre de 2017.

2. Presentación y visualización de los datos

Los conjuntos de datos asignados por los profesores son los siguientes.

abalone se trata de un conjunto de datos que presenta distintas variables fisiológicas del abulón, un marisco muy preciado en la cocina oriental, junto con el número de anillos que presenta, del cuál puede deducirse directamente su edad [6, 7]. El ejercicio consiste en la estimación de la edad del abulón a partir de las otras variables fisiológicas medidas a través de la regresión.

tae (Teaching Assistant Evaluation) se trata de un conjunto de datos que recoge las evaluaciones del desempeño docente de 150 asistentes en el Departamento de Estadística de la Universidad de Wisconsin-Madison [8, 9, 6]. Las puntuaciones fueron divididas en tres grupos de tamaño similar. Estos grupos o clases son el objetivo del ejercicio de clasificación.

A continuación se presenta un estudio más pormenorizado de los citados datasets. El código fuente utilizado para obtener los datos aquí presentados puede consultarse en section §A.

2.1. abalone

El conjunto de datos *abalone* corresponde a un dataframe con una dimensión de 4176 filas y 9 columnas. Esto corresponde a 4176 muestras en las que se han medido 9 propiedades. Estas propiedades o campos se presentan en table 1, no apareciendo en ninguna valores omitidos (*missing values*). Como puede observarse en la tabla, todos los datos son de tipo numérico, salvo el sexo, que es una variable categórica que sólo toma tres valores. El tipo de datos de la variable de salida es un número entero, que se tratará como una variable continua a la hora de realizar la regresión.

Las medidas estadísticas de las distintas variables pueden observarse en table 2 y en table 3. Estas medidas estadísticas nos hablan de las distribuciones de las distintas variables, que pueden visualizarse mediante un diagrama de caja y bigotes como el mostrado en figure 1.

En el ejercicio de regresión, que usará estos datos, será conveniente hacerse una idea de la correlación entre las distintas variables. Para ello, se ha calculado el coeficiente de correlación de Pearson entre las distintas variables numéricas, obteniendo así la matriz de correlación de table 4. Esta matriz nos da una idea de la relación lineal que tienen las variables entre sí. Otra forma de visualizar esta relación es con un diagrama de dispersión. En figure 2 puede observarse la relación que existe entre las distintas variables cuantitativas de entrada y la variable de salida (fila en negrita en table 4).

En vista de las medidas de correlación realizadas, podemos concluir que este conjunto de datos presenta muchas variables correlacionadas entre sí. Esto quiere decir que muchas de las variables de entrada aportan información redundante que puede causar sobreajuste en los modelos de regresión lineal. Por otro lado, la correlación de las distintas variables de entrada con respecto a la salida es considerablemente alta, por lo que un modelo sencillo podría servir para predecir la edad del abulón con un error relativamente pequeño.

Attribute	Domain	Data type	Description
`Sex`	[1,3]	`factor`	1 = M 2 = F 3 = I (infant)
`Length`	[0.075, 0.815]	`numeric`	Longest shell measurement (mm)
`Diameter`	[0.055, 0.65]	`numeric`	Diameter perpendicular to length (mm)
`Height`	[0.0, 1.13]	`numeric`	Height with meat in shell (mm)
`Whole_weight`	[0.0020, 2.8255]	`numeric`	Weight of the whole abalone (grams)
`Shucked_weight`	[0.0010, 1.488]	`numeric`	Weight of meat (grams)
`Viscera_weight`	[0.0005, 0.76]	`numeric`	Gut weight after bleeding (grams)
`Shell_weight`	[0.0015, 1.005]	`numeric`	Shell weight after being dried (grams)
`Rings`	[1,29]	`integer`	+1.5 gives the age in years

Cuadro 1: Descripción de los atributos presentes en *abalone* [6, 10].

Variable	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Var.	SD
`Length`	0.075	0.450	0.545	0.524	0.615	0.815	0.014	0.120
`Diameter`	0.055	0.350	0.425	0.408	0.480	0.650	0.010	0.099
`Height`	0.000	0.115	0.140	0.140	0.165	1.130	0.002	0.042
`Whole_weight`	0.002	0.442	0.800	0.829	1.153	2.826	0.240	0.490
`Shucked_weight`	0.001	0.186	0.336	0.359	0.502	1.488	0.049	0.222
`Viscera_weight`	0.001	0.094	0.171	0.181	0.253	0.760	0.012	0.110
`Shell_weight`	0.002	0.130	0.234	0.239	0.329	1.005	0.019	0.139
`Rings`	1.000	8.000	9.000	9.934	11.000	29.000	10.396	3.224

Cuadro 2: Medidas estadísticas sobre las variables cuantitativas de *abalone*.

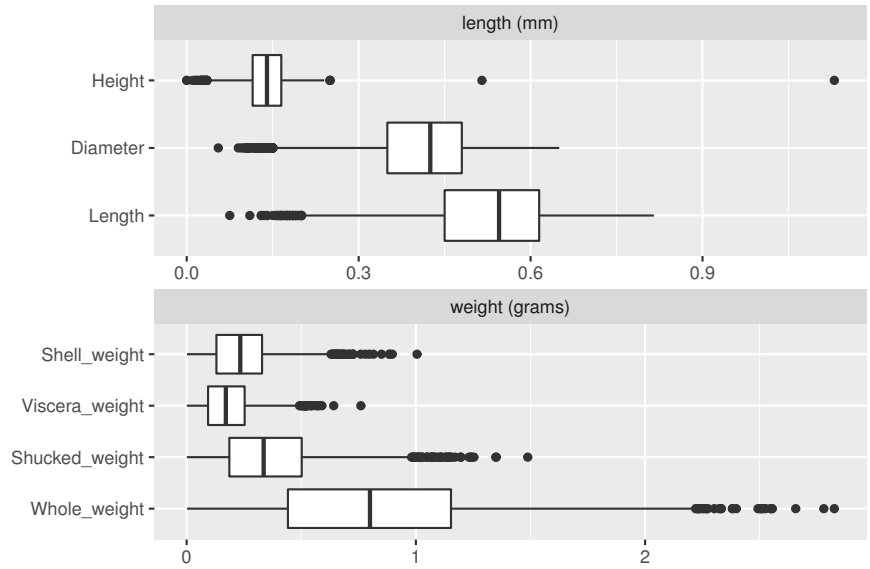
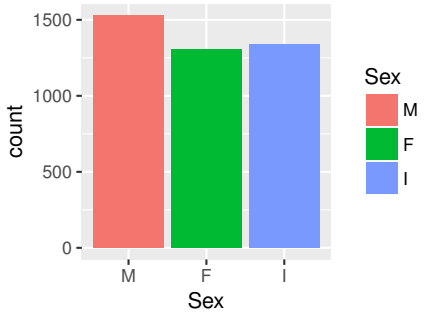


Figura 1: Distribución de las variables cuantitativas de entrada en *abalone*.

Sex	Freq.	Rel. Freq.
M	1528	0.366
F	1307	0.313
I	1341	0.321



Cuadro 3: Tabla de frecuencias de las variables categóricas de *abalone*, junto con su diagrama de barras.

Correlation	Length	Diameter	Height	Whole _weight	Shucked _weight	Viscera _weight	Shell _weight	Rings
Length	1.000	0.987	0.828	0.925	0.898	0.903	0.898	0.557
Diameter	0.987	1.000	0.834	0.925	0.893	0.900	0.905	0.575
Height	0.828	0.834	1.000	0.819	0.775	0.798	0.817	0.557
Whole_weight	0.925	0.925	0.819	1.000	0.969	0.966	0.955	0.540
Shucked_weight	0.898	0.893	0.775	0.969	1.000	0.932	0.883	0.421
Viscera_weight	0.903	0.900	0.798	0.966	0.932	1.000	0.908	0.504
Shell_weight	0.898	0.905	0.817	0.955	0.883	0.908	1.000	0.627
Rings	**0.557**	**0.575**	**0.557**	**0.540**	**0.421**	**0.504**	**0.627**	**1.000**

Cuadro 4: Correlación entre las variables cuantitativas en *abalone*.

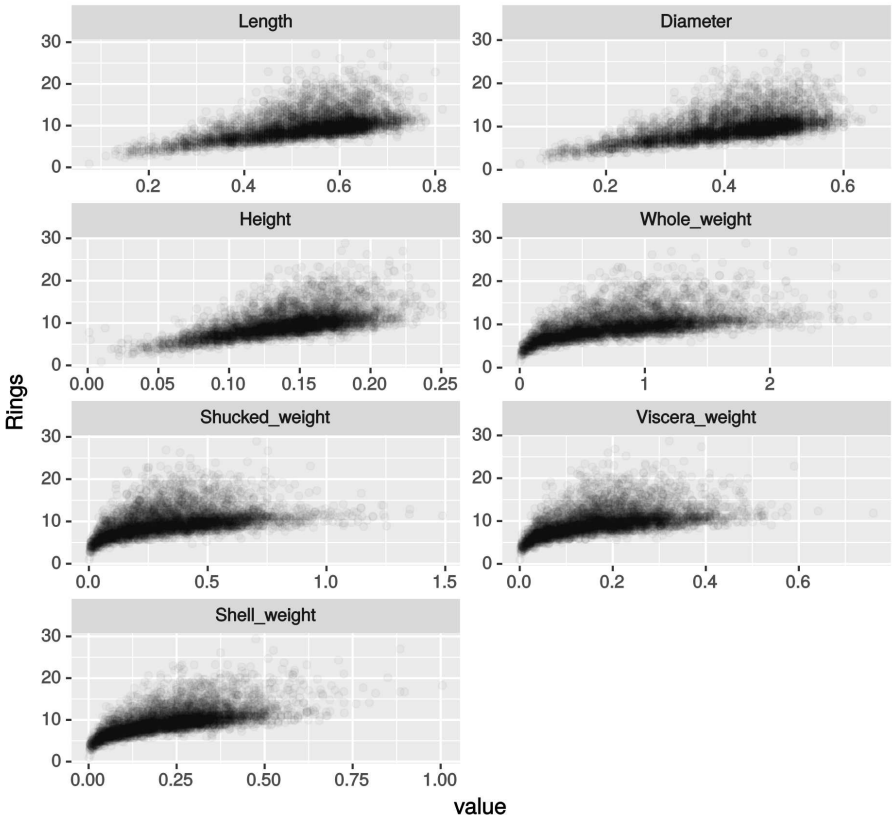


Figura 2: Nube de puntos de las variables de entrada con respecto a la salida en *abalone*.

Attribute	Domain	Data type	Description
`Native`	[1, 2]	`factor`	whether of not the TA is a native English speaker: 1 = English speaker 2 = non-English speaker
`Instructor`	[1, 25]	`factor`	course instructor (25 categories)
`Course`	[1, 26]	`factor`	26 categories
`Semester`	[1, 2]	`factor`	summer or regular semester: 1 = Summer 2 = Regular
`Size`	[3, 66]	`integer`	class size (numerical)
`Class`	{1, 2, 3}	`factor`	TA score: 1 = low 2 = medium 3 = high

Cuadro 5: Descripción de los atributos presentes en *tae* [6, 10].

Variable	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Var.	SD
`Size`	3	19	27	27.93	37	66	166.83	12.92
`Instructor`	1	8	13	13.58	20	25	46.31	6.81
`Course`	1	3	4.5	8.14	15	26	49.49	7.03

Cuadro 6: Medidas estadísticas sobre la única variables cuantitativa de *tae*. Las dos últimas filas representan las mismas medidas estadísticas sobre dos variables categóricas, a las que se han asignado etiquetas numéricas empezando en el 1.

2.2. tae (Teaching Assistant Evaluation)

El conjunto de datos *tae* corresponde a un dataframe con una dimensión de 150 filas y 6 columnas que no presentan valores omitidos. Esto corresponde a 150 muestras en las que se han medido 6 propiedades tal y como se muestra en table 5. Como puede observarse en la tabla, todas las variables son categóricas, salvo el tamaño de la clase, que es un entero. El tipo de datos de la variable de salida es también categórico, pudiendo tomar tres valores distintos: *low*, *medium* y *high*, por lo que tendremos que realizar una clasificación con tres clases.

Las medidas estadísticas sobre la variable numérica `Size` pueden observarse en table 6. Junto a ellas, se han calculado los momentos estadísticos de las variables `Instructor` y `Course`, que, a pesar de ser categóricas, tienen asignados valores numéricos enteros. No obstante, esta representación numérica no es apropiada, ya que los números han sido dados de forma arbitraria y no tienen un significado concreto. Esto se pone de manifiesto en la figura figure 3, en la que puede observarse que la distribución de estas variables no es normal, ni sigue ningún patrón conocido. Más bien se trata de unos puntos aleatorios, ya que cada instructor, o, equivalentemente cada curso, tendrá un número de asistentes determinado, independiente del número asignado al instructor o al curso. La figura figure 4, en cambio, muestra la distribución de la variable `Size`. Esta variable tiene una distribución más cercana a la normal, aunque existen diferencias en función de la clase a la que pertenecen los datos.

La figura figure 5a muestra la frecuencia absoluta de las variables categóricas, separándolas según la clase de salida. Como puede observarse, existe una diferencia perceptible entre los asistentes nativos y no nativos, y también entre el tipo de semestre. Por un lado, hay menos asistentes nativos, pero suelen obtener mejores resultados. Lo mismo pasa con el semestre de verano: a pesar de haber menos asistentes, los resultados son mejores. Podría pensarse que esto viene dado por el número de alumnos, que tal vez sea menor en verano, y también es posible que sólo los más privilegiados puedan tener un asistente nativo. Sin embargo, un test de Kolmogorov–Smirnov bilateral nos indica

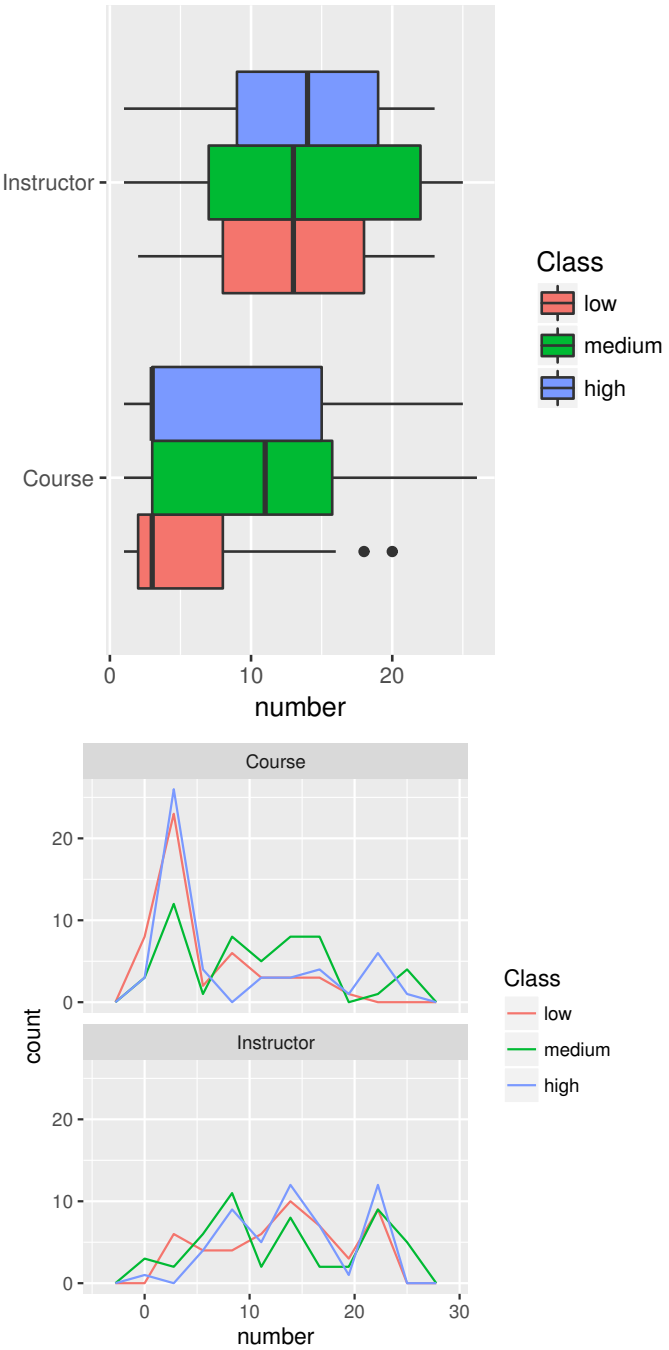


Figura 3: Diagrama de caja con bigotes de los valores numéricos de las variables `Course` e `Instructor`, junto a su distribución de frecuencias.

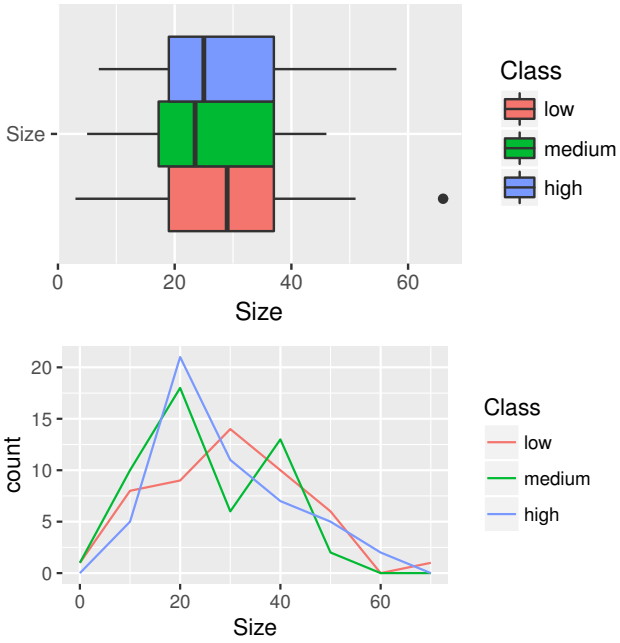


Figura 4: Diagrama de caja con bigotes de la variable numérica `Size` junto a su distribución de frecuencias.

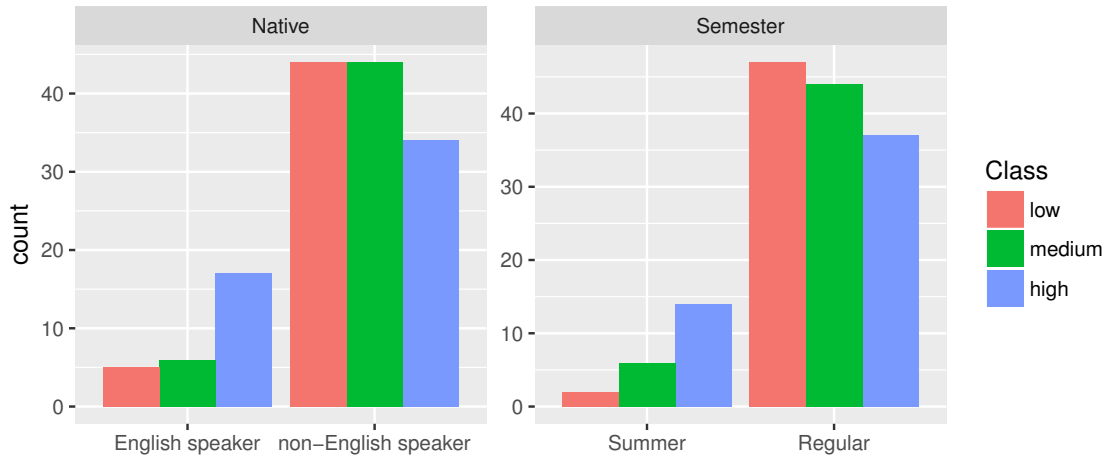
que no hay diferencia significativa en la distribución del tamaño del aula con asistentes nativos o no nativos. Sin embargo, el mismo test indica que sí existe una diferencia significativa en la distribución del tamaño del aula entre los semestres de verano e invierno.

Como se describía en el enunciado del problema, los tres grupos en los que se debe llevar a cabo la clasificación tienen una proporción parecida de registros, correspondientes a un tercio cada una. Esto se pone de manifiesto en la figura figure 5.

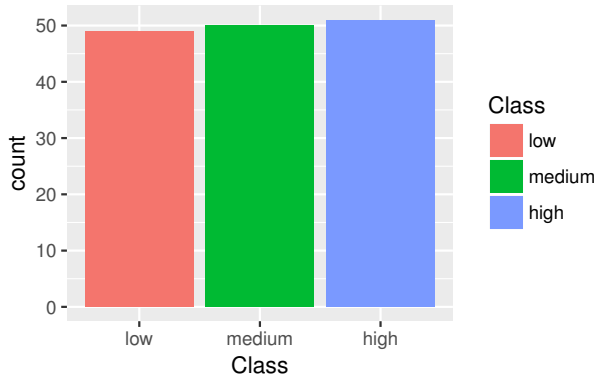
Volviendo a las variables categóricas comentadas anteriormente, debe decirse que el presente estudio ha sido realizado en 26 cursos, con 25 instructores diferentes. El número de asistentes que ha impartido clases en cada curso y con cada instructor ha sido variable. El valor numérico de cada curso y de cada instructor ha sido dado de manera arbitraria, de forma que los datos no presentan un orden. En la figura figure 6 se ha representado la frecuencia de las ocurrencias de `Course`, esto es, el número de asistentes que ha impartido clases en cada curso, ordenadas de mayor a menor. Como puede observarse, no hay un orden específico en el eje de abscisas, ni tampoco una distribución clara de los datos. Junto a la frecuencia absoluta, se ha representado la frecuencia relativa de cada clase, es decir, el número de asistentes con rendimiento alto, medio y bajo en proporción al número de asistentes en cada curso. En este caso, puede observarse que el rendimiento no depende del número de asistentes que haya tenido el curso, puesto que los valores no siguen ninguna tendencia en concreto. Este mismo análisis se ha realizado para la variable `Instructor`, obteniendo las mismas conclusiones (ver figura figure 7).

Un hecho que puede llamar la atención de este análisis es la existencia de cursos e instructores con resultados 100 % buenos (barritas azules), 100 % malos (barritas rojas), 100 % medios (barritas verdes) o mezcla (barritas de varios colores). Esto se debe, en parte, al número de asistentes que han dado clase en cada curso o con cada instructor. Cuando se tiene un número elevado de registros (muchos asistentes), los resultados de rendimiento suelen estar divididos. La tabla table 7 muestra un recuento de estos resultados, basándose en las observaciones de las figuras figure 6 y figure 7.

Como conclusión, hay que decir que se trata de un ejercicio de clasificación muy complicado. Esto se debe a que la mayor parte de las entradas son categóricas, mientras que los algoritmos de clasificación suelen esperar variables numéricas, y a la poca relación vista entre las distintas variables. Teniendo esto en cuenta, es posible predecir un rendimiento pobre en la clasificación.



(a) Variables de entrada.



(b) Variable de salida.

Figura 5: Conteo de variables categóricas.

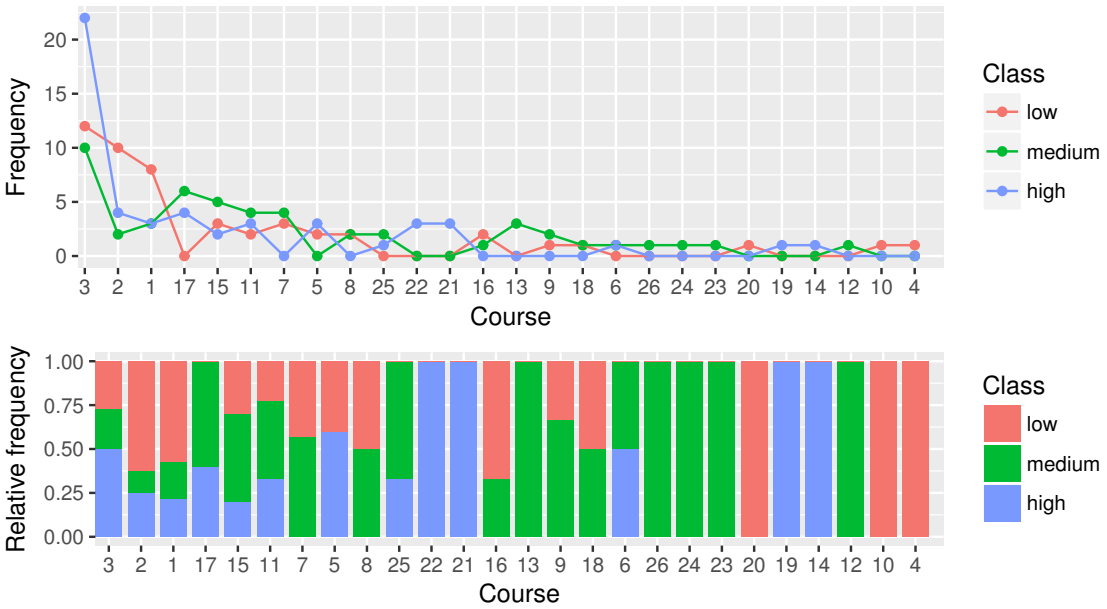


Figura 6: Frecuencia absoluta y relativa de las ocurrencias de `Course` en función de su clase. Los datos han sido ordenados para que las mayores frecuencias queden a la izquierda.

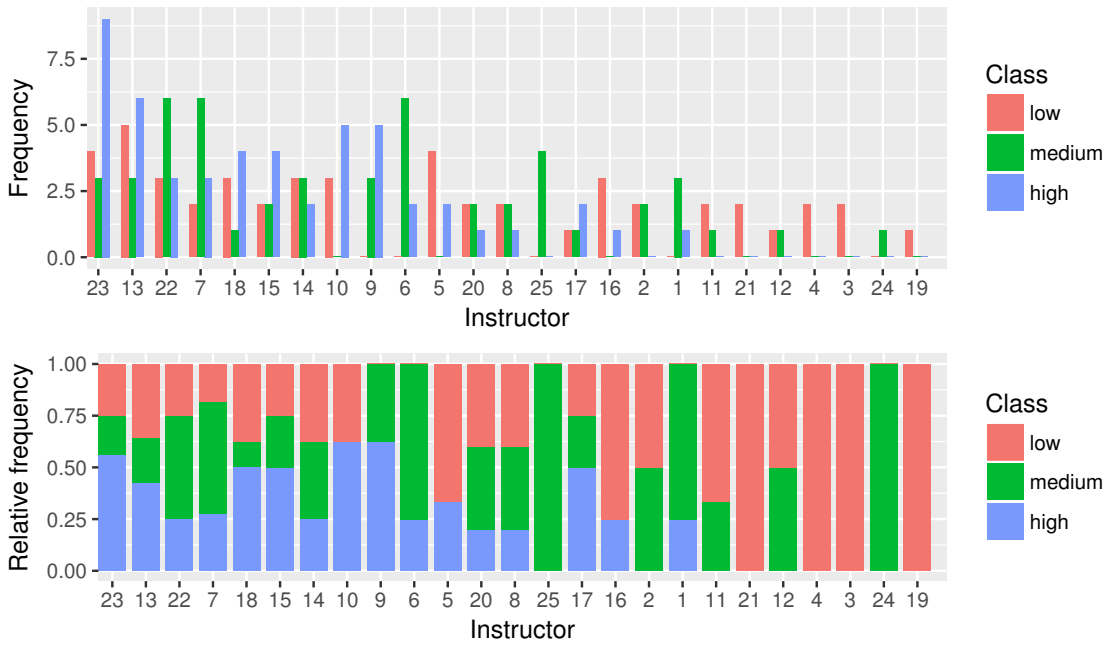


Figura 7: Frecuencia absoluta y relativa de las ocurrencias de `Instructor` en función de su clase. Los datos han sido ordenados para que las mayores frecuencias queden a la izquierda.

Class (performance)	Course	Instructor
low	3	4
medium-low	6	3
medium-low-high	5	10
medium	5	2
low-high	1	3
medium-high	2	3
high	4	0

Cuadro 7: Recuento del número de grupos e instructores en función del rendimiento.

Regressor	Summary of fit				Cross validation	
	Linear model ($\hat{Y}(X)$)	MSE	R^2	p-value	$\overline{MSE}_{tra}$	$\overline{\mathbf{MSE}}_{\mathbf{tst}}$
`Sex`	$10{,}71 + 0{,}42\,X_F - 2{,}81\,X_I$	8.39	0.1929	< 2.2e-16	8.39	**8.40**
`Length`	$2{,}10 + 14{,}94\,X$	7.17	0.3098	< 2.2e-16	7.17	**7.18**
`Diameter`	$2{,}32 + 18{,}67\,X$	6.96	0.3301	< 2.2e-16	6.96	**6.97**
`Height`	$3{,}94 + 42{,}96\,X$	7.16	0.3107	< 2.2e-16	7.15	**7.33**
`Whole_weight`	$6{,}99 + 3{,}55\,X$	7.36	0.2919	< 2.2e-16	7.36	**7.37**
`Shucked_weight`	$7{,}74 + 6{,}11\,X$	8.55	0.1771	< 2.2e-16	8.55	**8.57**
`Viscera_weight`	$7{,}26 + 14{,}82\,X$	7.76	0.2537	< 2.2e-16	7.76	**7.77**
`Shell_weight`	$6{,}46 + 14{,}53\,X$	6.30	0.3937	< 2.2e-16	6.30	**6.32**

Cuadro 8: Regresiones lineales simples, con cada regresor.

3. Regresión sobre el conjunto de datos abalone

Tras haber visualizado los datos y haber explorado las posibles relaciones entre las distintas variables de entrada (ver sección 2.1), procedemos a aplicar la regresión lineal sobre el conjunto de datos. Para ello, se han explorado diversos modelos, que van desde la regresión lineal simple hasta modelos que contemplan las no linealidades e interacciones entre las variables. Posteriormente, se ha experimentado la regresión con el algoritmo k-NN. En este caso, se han evaluado los modelos encontrados con la regresión lineal.

En todos los casos, la bondad de las regresiones se ha medido usando validación cruzada con el método *k-fold cross validation*. De esta forma, ha sido posible medir el error cuadrático medio sobre el conjunto de test en cada regresión MSE_{tst}, realizando una media sobre los distintos conjuntos de test $\overline{MSE}_{tst}$. El error cuadrático medio sobre el conjunto de entrenamiento MSE_{tra} ha sido de utilidad para la evaluación del sobreajuste (*overfitting*). Para la ejecución de la validación cruzada se ha hecho uso de las particiones ya existentes.

3.1. Regresión lineal simple

Se ha realizado un modelo de regresión lineal sobre cada regresor. Los datos de los mismos pueden verse en la tabla 8. Como puede observarse, el regresor que ha dado mejor resultado ha sido `Shell_weight`, seguido por `Diameter`. Esta elección se ha basado en minimizar el error cuadrático medio sobre el conjunto de test ($\overline{MSE}_{tst}$), ya que consideraremos que el modelo tiene que tener la habilidad de generalizar (evitar sobreajuste). Llama la atención que los errores MSE medidos sobre los conjuntos de entrenamiento y el conjunto de test son muy parecidos. Esto se debe a que se está usando un modelo de regresión con una única variable, que difícilmente puede sobreajustar. En los apartados siguientes veremos un comportamiento bastante distinto, en el que $\overline{MSE}_{tst}$ será claramente mayor a $\overline{MSE}_{tra}$.

Observando las gráficas de la figura 2, podemos intuir que existen no linealidades entre algunas de las variables de entrada y la de salida. En concreto, es claro que todos los pesos (variables `_weight`) siguen una curva parecida a una raíz o un logaritmo. Se han llevado a cabo diversas experiencias, aplicando operaciones sencillas a los datos de entrada. En particular, se ha probado la raíz cuadrada, cúbica y el logaritmo. Estas tres operaciones tienen un sentido físico sobre algunas de las variables. En concreto, el volumen ocupado (directamente proporcional al peso) es una función cúbica del radio, mientras que la superficie de contacto con el exterior es una función cuadrática. Esto es un factor limitante del crecimiento. Por otro lado, hay especies que crecen de forma exponencial, como las colonias de bacterias. De entre las operaciones probadas, el logaritmo ha dado el mejor resultado, al aumentar el índice de correlación lineal de Pearson y mejorar el ajuste lineal, como puede verse en la tabla 9 y la figura 9.

3.2. Regresión lineal múltiple

A continuación, se ha realizado la regresión lineal teniendo en cuenta todos los regresores, sin contemplar interacciones entre ellos. Se ha visto, mediante *backward selection* que los regresores que

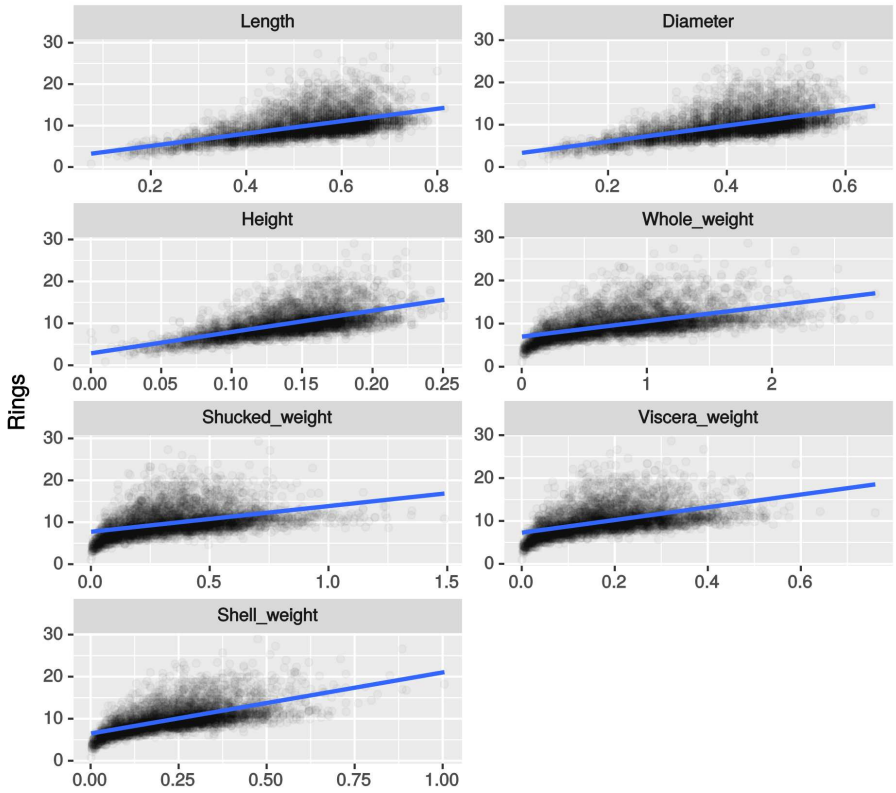


Figura 8: Regresiones lineales simples, con cada regresor.

Regressor	Correlation with `Rings`	Summary of fit			Cross validation	
		$\hat{Y}(X)$	R^2	p-value	$\overline{MSE}_{tra}$	$\overline{\mathbf{MSE}}_{\mathbf{tst}}$
`Whole_weight`	0.540	$6{,}99 + 3{,}55\,X$	0.2919	$<$ 2.2e-16	7.36	**7.37**
log (`Whole_weight`)	0.584	$10{,}93 + 2{,}25\,X$	0.3411	$<$ 2.2e-16	6.85	**6.86**
`Shucked_weight`	0.421	$7{,}74 + 6{,}11\,X$	0.1771	$<$ 2.2e-16	8.55	**8.57**
$\sqrt{\texttt{Shucked\_weight}}$	0.477	$5{,}49 + 7{,}85\,X$	0.2270	$<$ 2.2e-16	8.03	**8.04**
$\sqrt[3]{\texttt{Shucked\_weight}}$	0.492	$3{,}43 + 9{,}64\,X$	0.2423	$<$ 2.2e-16	7.87	**7.89**
log (`Shucked_weight`)	0.510	$12{,}40 + 1{,}91\,X$	0.2604	$<$ 2.2e-16	7.69	**7.70**
`Viscera_weight`	0.504	$7{,}26 + 14{,}82\,X$	0.2537	$<$ 2.2e-16	7.76	**7.77**
log (`Viscera_weight`)	0.566	$14{,}17 + 2{,}15\,X$	0.3198	$<$ 2.2e-16	7.07	**7.08**
`Shell_weight`	0.627	$6{,}46 + 14{,}53\,X$	0.3937	$<$ 2.2e-16	6.30	**6.32**
$\sqrt[3]{\texttt{Shell\_weight}}$	*0.649*	$0{,}87 + 15{,}31\,X$	*0.4212*	*$<$ 2.2e-16*	*6.02*	**6.03**
log (`Shell_weight`)	0.634	$14{,}14 + 2{,}52\,X$	0.4015	$<$ 2.2e-16	6.22	**6.23**

Cuadro 9: Corrección de no linealidades en las distintas variables.

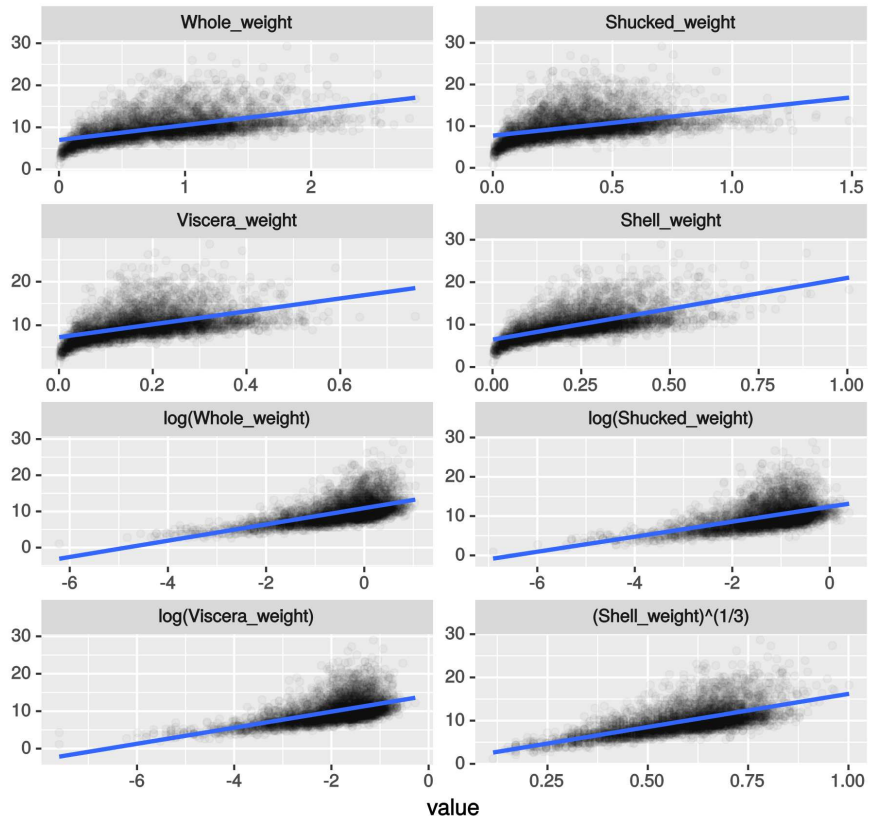


Figura 9: Regresión lineal simple en las variables en las que se han teniendo en cuenta las no linealidades. Claramente, el modelo que aplica preprocesamiento tiene un mejor ajuste.

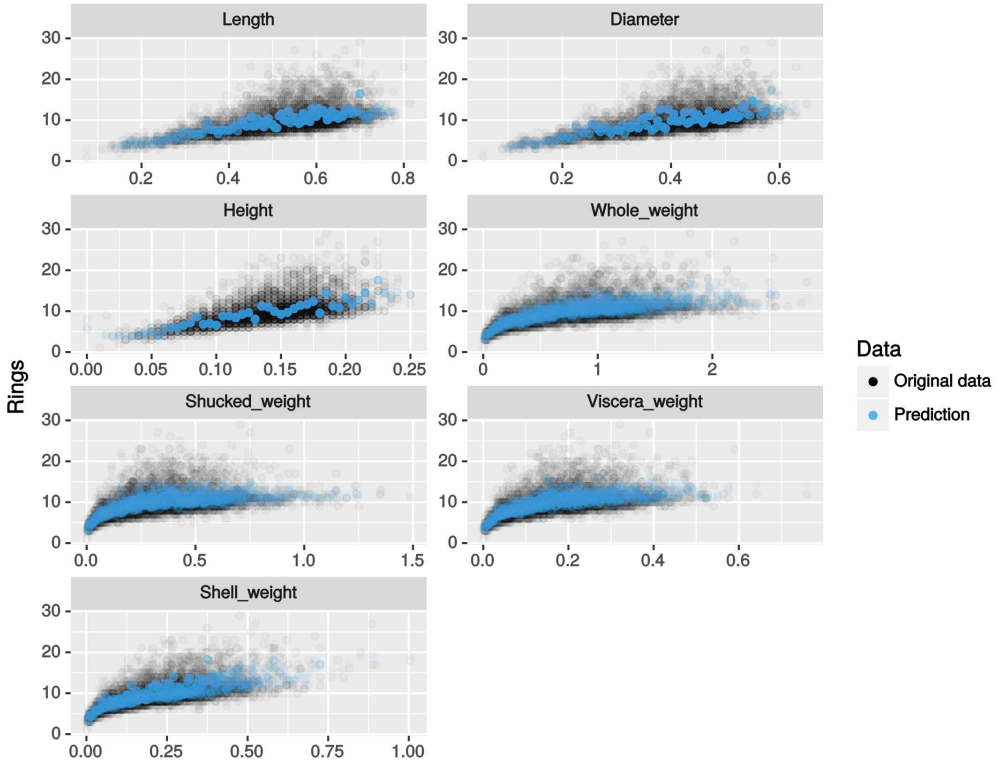


Figura 10: Regresiones con el algoritmo k-NN, con cada regresor de forma individual ($k = 7$).

producen mejor resultados son `Diameter`, `Height` y todos los `_weight`, pudiéndose descartar `Sex` y `Length` por presentar un p-value muy elevado. En la tabla 10 se puede contemplar los resultados de esta segunda aproximación, así como el de varias iteraciones en las que se han ido eliminando cada vez más regresores.

Así mismo, se ha realizado la regresión múltiple tratando de corregir las no linealidades vistas en el apartado anterior. En concreto, se han corregido las variables que miden los distintos pesos usando funciones logarítmicas y raíces cúbicas. Adicionalmente se han ensayado modelos con interacciones entre las variables. Al usar interacciones, se ha comprobado que es muy fácil conseguir modelos que produzcan sobreajuste, por lo que hay que seleccionar las interacciones con cuidado.

En este apartado hemos visto modelos que ajustaban muy bien el conjunto de *training*, pero en cambio fallaban al ser evaluados con el conjunto de test. Esto se debe a que los modelos que tienen en cuenta muchos parámetros tienen una gran varianza, perdiendo de vista las principales relaciones que hay entre las distintas variables. Esto hace que se sobreajusten al ruido presente en conjunto de datos, lo que se conoce como *overfitting*, y es el efecto contrario al producido por el sesgo que produce tener en cuenta pocas variables de entrada, o *underfitting*.

3.3. Regresión mediante el algoritmo k-NN

El algoritmo de los k vecinos más cercanos (k-Nearest Neighbours) puede usarse para regresión.

Comparando las figuras 8 y 10, podría parecer que k-NN con un único regresor ofrece resultados muy superiores a los del modelo lineal. No obstante, los resultados son muy dependientes del parámetro k. Para el ejemplo de las gráficas, con $k = 7$, el ajuste lineal tiene, en realidad, mejores resultados. En general, parece que k-NN produce más sobreajuste, al tener una diferencia mayor entre $\overline{MSE}_{tra}$ y $\overline{MSE}_{tst}$. Esto puede verse claramente en la figura 11. En la misma, puede observarse que la diferencia del error en los sets de entrenamiento y de test es muy elevada, en especial con valores pequeños de k. Esta diferencia se acentúa en el caso de tener un modelo más complejo, en cuyo caso un valor bajo de k produce un sobreajuste muy importante.

Regressors	Summary of fit		Cross validation		Comments
	R^2	p-value	$\overline{MSE}_{tra}$	$\overline{\mathbf{MSE}}_{\mathbf{tst}}$	
`Diameter + Height + Whole_weight + Shucked_weight + Viscera_weight + Shell_weight`	0.528	$<$ 2.2e-16	4.90	**5.03**	Modelo lineal eliminando las variables menos relevantes.
`Length + Height + Whole_weight.log + Shucked_weight.log + Viscera_weight.log + Shell_weight.3`	0.544	$<$ 2.2e-16	4.74	**4.78**	Modelo lineal con preprocesamiento para eliminar las no linealidades.
`poly(Length, 2) + poly(Height, 2) + poly(Whole_weight, 2) + poly(Shucked_weight, 2) + poly(Viscera_weight, 2) + poly(Shell_weight, 2)`	0.565	$<$ 2.2e-16	4.51	**4.83**	Modelo lineal usando polinomios de segundo grado para tratar de corregir las no linealidades. A pesar de que R^2 aumenta y $\overline{MSE}_{tra}$ disminuye, el error en el conjunto de test crece, por lo que podemos sospechar que el modelo produce overfitting.
`poly(Length, 3) + poly(Height, 3) + poly(Whole_weight, 3) + poly(Shucked_weight, 3) + poly(Viscera_weight, 3) + poly(Shell_weight, 3)`	0.568	$<$ 2.2e-16	4.48	**5.68**	Modelo lineal usando polinomios de grado 3 para tratar de corregir las no linealidades. El modelo produce overfitting de forma clara, lo cuál es evidente si miramos la diferencia entre $\overline{MSE}_{tra}$ y $\overline{MSE}_{tst}$.
`Length + Height + Whole_weight.log * Shucked_weight.log * Viscera_weight * Shell_weight.3`	0.574	$<$ 2.2e-16	4.42	**4.50**	Modelo lineal teniendo en cuenta las interacciones entre algunas de las variables. El modelo produce mejores resultados que los anteriores, al reducir el $\overline{MSE}_{tst}$.

Cuadro 10: Regresiones lineales múltiples.

Regressor	k-NN ($k = 3$)		k-NN ($k = 7$)	
	$\overline{MSE}_{tra}$	$\mathbf{\overline{MSE}_{tst}}$	$\overline{MSE}_{tra}$	$\mathbf{\overline{MSE}_{tst}}$
`Sex`	9.39	**9.35**	8.81	**8.81**
`Length`	9.00	**9.34**	7.82	**8.12**
`Diameter`	9.49	**9.82**	7.67	**7.97**
`Height`	8.28	**8.46**	7.42	**7.54**
`Whole_weight`	3.96	**9.34**	5.17	**7.79**
`Shucked_weight`	5.70	**10.56**	5.97	**8.88**
`Viscera_weight`	6.75	**9.87**	5.97	**8.22**
`Shell_weight`	5.91	**7.74**	5.45	**6.71**
log (`Whole_weight`)	3.95	**9.30**	5.17	**7.77**
log (`Shucked_weight`)	5.72	**10.57**	5.96	**8.91**
log (`Viscera_weight`)	6.77	**9.94**	5.98	**8.26**
$\sqrt[3]{\texttt{Shell\_weight}}$	5.90	**7.65**	5.42	**6.64**

Regressor	k-NN ($k = 15$)		Linear regression	
	$\overline{MSE}_{tra}$	$\mathbf{\overline{MSE}_{tst}}$	$\overline{MSE}_{tra}$	$\mathbf{\overline{MSE}_{tst}}$
`Sex`	9.06	**9.10**	8.39	**8.40**
`Length`	7.24	**7.59**	7.17	**7.18**
`Diameter`	7.08	**7.36**	6.96	**6.97**
`Height`	7.00	**7.15**	7.15	**7.33**
`Whole_weight`	5.98	**7.29**	7.36	**7.37**
`Shucked_weight`	6.72	**8.20**	8.55	**8.57**
`Viscera_weight`	6.31	**7.56**	7.76	**7.77**
`Shell_weight`	5.43	**6.26**	6.30	**6.32**
log (`Whole_weight`)	5.98	**7.29**	6.85	**6.86**
log (`Shucked_weight`)	6.72	**8.21**	7.69	**7.70**
log (`Viscera_weight`)	6.31	**7.59**	7.07	**7.08**
$\sqrt[3]{\texttt{Shell\_weight}}$	5.41	**6.23**	6.02	**6.03**

Cuadro 11: Regresiones simples usando k-NN, con cada regresor. Comparativa con regresión lineal simple.

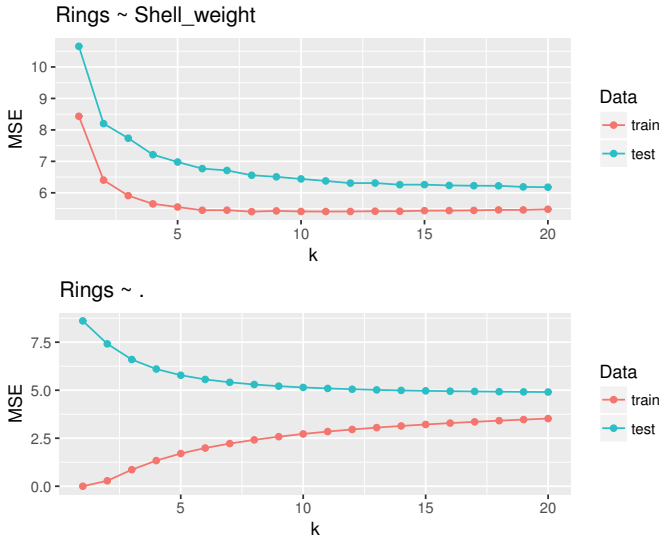


Figura 11: Comparación de diferentes modelos construidos con k-NN, cambiando el valor de k. Los errores representados son $\overline{MSE}_{tra}$ y $\overline{MSE}_{tst}$.

Una de las mayores ventajas de k-NN es que se ajusta a las no linealidades sin necesidad de preprocesamiento, devolviendo resultados muy parecidos en ambos casos. Por este motivo, puede prescindirse de esta etapa al trabajar con modelos más complejos.

La figura 12 muestra una comparación del error cuadrático medio en modelos complejos. Al igual que ocurría en la regresión lineal, el modelo que tiene en cuenta las interacciones produce resultados muy buenos, siendo el que produce los mejores resultados con $k > 6$.

3.4. Comparación de los resultados

En este apartado se comparan los resultados obtenidos con el ajuste lineal, el algoritmo k-NN aplicado a la regresión. Más tarde, se comparan de forma genérica los modelos de ajuste lineal, k-NN y el modelo de regresión M5'. Para ello, se ha realizado un contraste de hipótesis sobre los resultados de la aplicación de estos métodos a los distintos conjuntos de datos usados en estas prácticas.

3.4.1. Comparación entre los modelos lineales y regresiones con k-NN para el caso estudiado

A la vista de las tablas 10 y 12, es posible decir que ambos métodos arrojan unos resultados parecidos, siempre que k sea lo suficientemente elevado. La figura 13 muestra los errores cometidos para 20 muestras de un conjunto de test, que no ha sido usado en el entrenamiento del modelo. En la misma puede apreciarse que ambos modelos cometen errores parecidos para la mayoría de los puntos. En cambio, hay puntos para los que los errores cometidos son considerablemente distintos. A pesar de estas diferencias, la media de los errores cuadráticos es muy parecida.

La figura 14 muestra una comparación de las distribuciones del error cuadrático para el mismo conjuto de test. Como puede observarse, las dos distribuciones son muy parecidas. Esto induce a pensar que ninguno de los dos métodos usados sea mejor que el otro. En efecto, un test de *Kolmogorov–Smirnov* no puede negar, con un p-value de 0.77, que las muestras hayan sido extraídas de la misma distribución. El test de *Mann–Whitney–Wilcoxon* arroja un resultado parecido, con un p-value de 0.61.

A la hora de elegir un modelo para explicar el conjunto de datos estudiado y realizar predicciones, debería usarse el modelo lineal que considera las interacciones (`Rings ~ Length + Height + Whole_weight.log * Shucked_weight.log * Viscera_weight * Shell_weight.3`). Para realizar esta elección no sólo se ha tenido en cuenta que el MSE medido ha sido ligeramente más bajo que el del correspondiente modelo k-NN, pues ya hemos visto que la diferencia no es significativa y ésta se debe, probablemente, a las particiones realizadas para la validación cruzada. En cambio, la elección

Regressors	k-NN ($k = 3$)		k-NN ($k = 7$)		k-NN ($k = 15$)	
	$\overline{MSE}_{tra}$	$\overline{\mathbf{MSE}}_{\mathbf{tst}}$	$\overline{MSE}_{tra}$	$\overline{\mathbf{MSE}}_{\mathbf{tst}}$	$\overline{MSE}_{tra}$	$\overline{\mathbf{MSE}}_{\mathbf{tst}}$
. *(sum of every variable)*	0.86	**6.60**	2.22	**5.41**	3.22	**4.97**
`Diameter + Height + Whole_weight + Shucked_weight + Viscera_weight + Shell_weight`	0.92	**6.53**	2.36	**5.48**	3.33	**5.02**
`Length + Height + Whole_weight.log + Shucked_weight.log + Viscera_weight.log + Shell_weight.3`	0.95	**6.56**	2.40	**5.53**	3.45	**5.09**
`poly(Length, 2) + poly(Height, 2) + poly(Whole_weight, 2) + poly(Shucked_weight, 2) + poly(Viscera_weight, 2) + poly(Shell_weight, 2)`	0.93	**6.59**	2.32	**5.46**	3.31	**5.01**
`poly(Length, 3) + poly(Height, 3) + poly(Whole_weight, 3) + poly(Shucked_weight, 3) + poly(Viscera_weight, 3) + poly(Shell_weight, 3)`	0.94	**6.48**	2.34	**5.42**	3.31	**4.99**
`Length + Height + Whole_weight.log * Shucked_weight.log * Viscera_weight * Shell_weight.3`	0.68	**6.40**	1.92	**5.36**	2.89	**4.90**

Cuadro 12: Regresiones kNN múltiples.

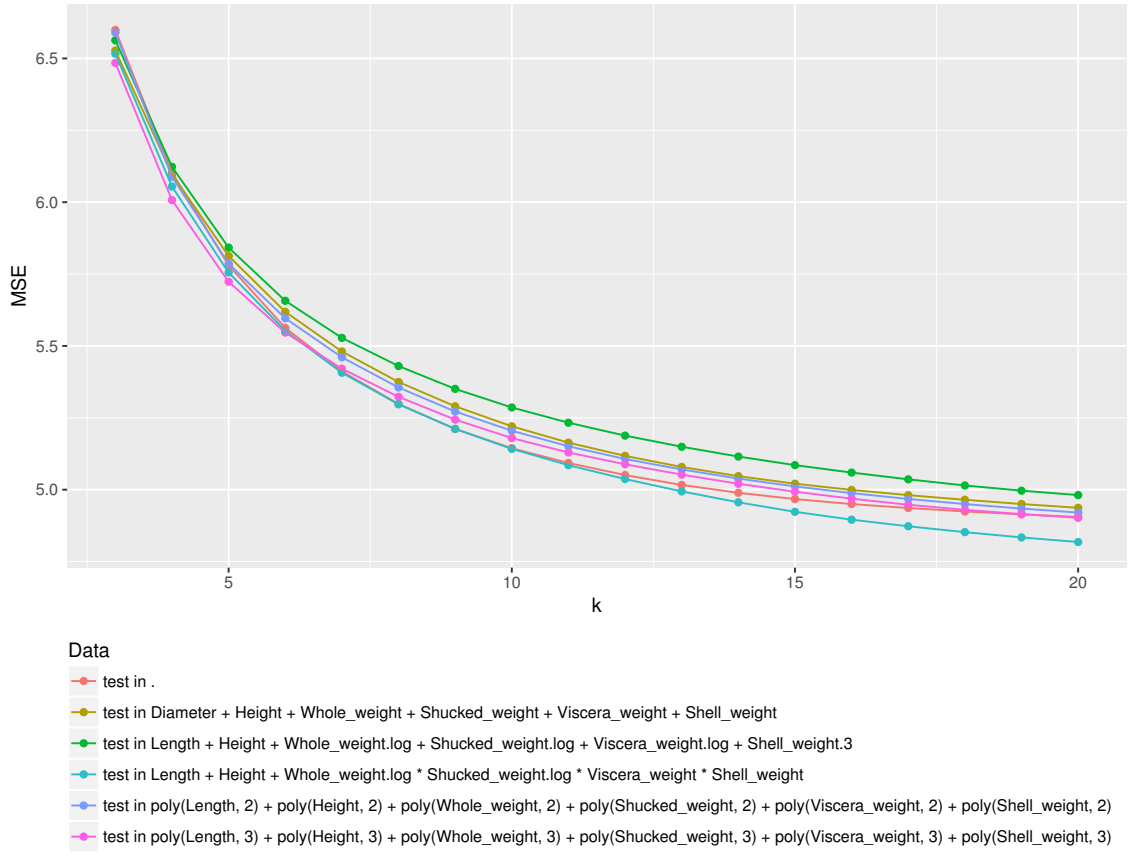


Figura 12: Comparación de diferentes modelos construidos con k-NN, cambiando el valor de k. El error representado es $\overline{MSE}_{tst}$.

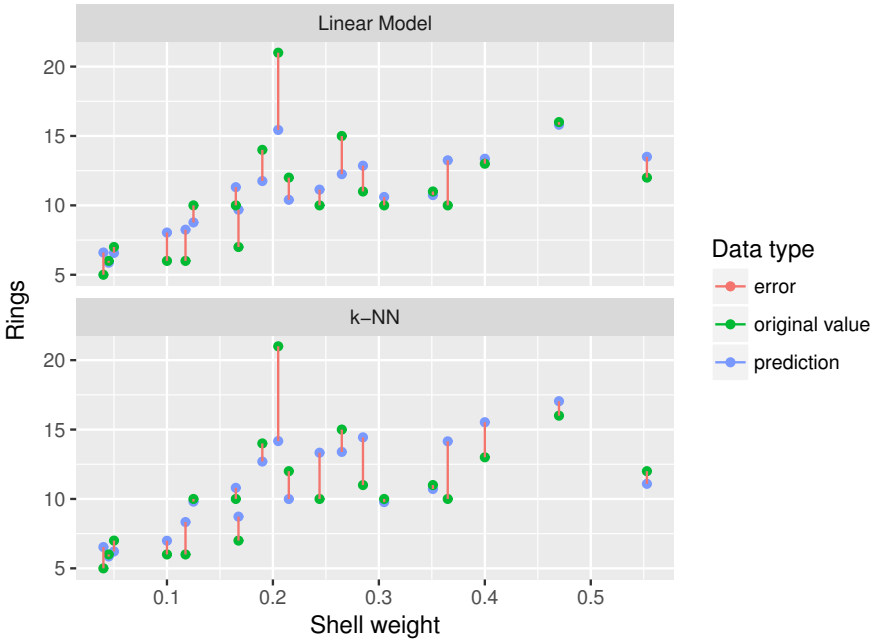


Figura 13: Comparación entre 20 datos reales sus predicciones con k-NN y LM para el modelo `Rings ~ Length + Height + Whole_weight.log * Shucked_weight.log * Viscera_weight * Shell_weight.3` ($k = 7$). Aunque los errores tienen distintos valores para distintos puntos, la media es muy parecida.

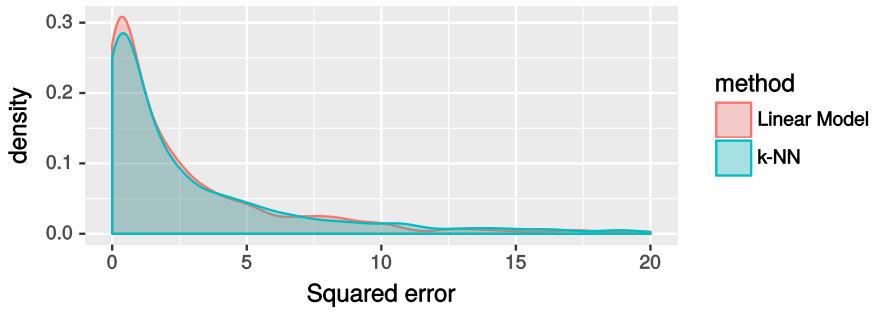


Figura 14: Comparación entre las distribuciones de error cuadrático con k-NN y LM para el modelo `Rings ~ Length + Height + Whole_weight.log * Shucked_weight.log * Viscera_weight * Shell_weight.3` ($k = 7$).

R+	R-	p-value
78	93	0.7660

Cuadro 13: Resultado de aplicar el test de los rangos con signo de Wilcoxon a los resultados de MSE estudiados.

p-value	LM	k-NN
k-NN	0.67	—
M5'	0.16	0.16

$1-p$-value	LM	k-NN
k-NN	0.33	—
M5'	0.84	0.84

Cuadro 14: Resultados del test de los rangos con signo de Wilcoxon por parejas, aplicado a los resultados de *Accuracy* obtenidos por los diferentes algoritmos de regresión.

del modelo lineal se debe al hecho de que éste requiere un menor coste computacional, especialmente en la predicción, y que produce un modelo real que podría ser interpretado por un experto. También cabe destacar que dicho modelo produce menos sobreajuste que su equivalente k-NN, lo cuál puede verse al comparar los errores cuadráticos medios en los conjunto de entrenamiento y de test.

3.4.2. Comparación genérica entre los modelos lineal, k-NN y M5'

Para comparar si existen diferencias significativas entre los modelos lineal, k-NN y M5', se han realizado diversos test estadísticos sobre los resultados de validación cruzada de modelos que usan todas las variables en distintos conjuntos de datos.

En primer lugar, se ha estudiado si existe diferencia significativa entre el modelo lineal y k-NN. Para ello, se ha realizado un test de los rangos con signo de Wilcoxon. El resultado, con un p-value de 0,77, es que no puede asegurarse que ninguno de los dos métodos produzca menor MSE.

Para comprobar si alguno de los tres métodos produce resultados significativamente superiores al resto se ha procedido a realizar un test de Friedman. En este caso, el p-value obtenido es de 0,0338, por lo que podemos negar la hipótesis nula que mantiene que no existe diferencia significativa entre el $\overline{MSE}_{tst}$ medido con los métodos mencionados para el conjunto de datasets estudiado. Dicho esto, será necesario comparar los tres métodos entre sí, para lo que se hará uso de un test de los rangos con signo de Wilcoxon por parejas, cuyos resultado puede observarse en la tabla 14. Observando estos resultados, podemos decir que sólo existe un 33 % de significancia de que LM produzca resultados distintos a k-NN. En cambio, tenemos un 84 % de significancia de que M5' produzca resultados distintos tanto a LM como a k-NN.

4. Clasificación sobre el conjunto de datos tae

Tras haber visualizado los datos y haber explorado las posibles relaciones entre las distintas variables de entrada (ver sección 2.2), procedemos a aplicar distintos algoritmos de clasificación sobre el conjunto de datos. Para ello, se han utilizado en todo caso las particiones ya existentes, que serán también utilizadas para la validación cruzada.

4.1. Clasificación mediante el algoritmo k-NN

El algoritmo de los k vecinos más cercanos (k-NN) es un método no paramétrico que puede ser usado en clasificación. Tiene la principal ventaja de que suele conseguir muy buenas predicciones y la fase de entrenamiento es muy eficiente. Sin embargo, k-NN no produce un modelo para explicar el conjunto de datos, y la etapa de predicción es computacionalmente costosa.

Cabe destacar que, puesto que el principio básico de k-NN supone medir la distancia de cada elemento a sus k elementos más cercanos, es necesario definir una métrica y es conveniente que las variables de entrada estén normalizadas. Por este motivo, el primer paso en el procesamiento de los datos será normalizar la variable `Size`. Adicionalmente, podría procederse a normalizar el resto de variables, que, a pesar de ser categóricas, tienen un valor numérico. Esto supone prescindir de

Variable	Native	Instructor	Course	Semester	Size
Scale factor	x4	x4	x4	x1	*normalized*

Cuadro 15: Factores de normalización aplicados a las variables de entrada antes del algoritmo k-NN.

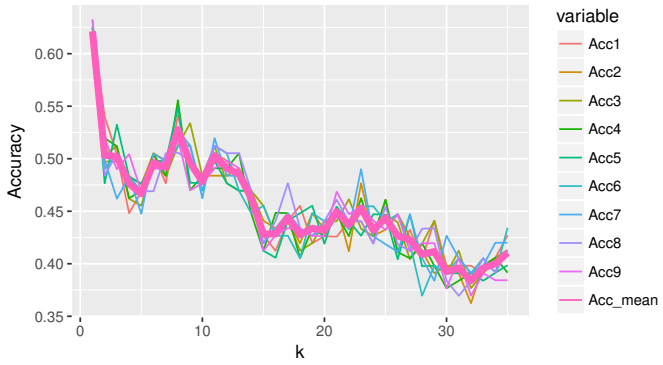


Figura 15: *Accuracy* obtenido con el algoritmo k-NN en función del parámetro k. Cada punto ha sido calculado haciendo la media de los resultados obtenidos para cada partición de validación cruzada (*k-fold*). La experiencia se ha repetido 9 veces para observar la variabilidad de los resultados producida por la componente aleatoria del algoritmo k-NN. La media se ha representado con una línea magenta más gruesa.

la transformación en *factors* de las variables de entrada. Aunque este paso no tiene un fundamento claro, se ha comprobado que mejora los resultados de la predicción. No obstante, tras probar con distintos factores de escalado en cada variable de entrada, se ha comprobado que la mayor mejora no se produce con una normalización[1], sino con un escalado como el de la tabla 15.

Con el fin de obtener los mejores resultados posibles, se ha realizado un estudio del rendimiento del algoritmo k-NN con distintos valores del parámetro k. Para ello, se ha calculado el *Accuracy* medio de las predicciones realizadas para cada partición de validación cruzada (*k-fold*) con distintos valores de k. Esta experiencia se ha repetido nueve veces, con el fin de observar la variabilidad de los resultados producida por la componente aleatoria del algoritmo k-NN. Por último, se ha calculado la media de los resultados de Accuracy observados, seleccionando la cifra más alta junto al valor de k que la produce. Los resultados óptimos, representados en la figura 15, se han conseguido para $k = 1$, llegando a un *Accuracy* medio de 0,621, con una desviación típica de 0,009.

4.2. Clasificación mediante análisis discriminante lineal (LDA)

LDA asume que los datos han sido extraídos de una distribución normal. Por esto, LDA no funcionará bien con datos que no sigan una distribución normal. Para comprobar si las variables de entrada siguen una distribución normal, se ha procedido a realizar un test de Shapiro-Wilk sobre las mismas. La tabla 16 muestra los resultados de este test, que deben interpretarse del siguiente modo: para cada una de las variables, debe rechazarse la hipótesis de que los datos provengan de una distribución normal. La figura 16 muestra los gráficos Q-Q de estas variables, pudiendo comprobarse que no se ajustan a la distribución normal. Tal vez las variables Size e Instructor tengan una forma más parecida a la recta esperada, cosa que también se pone de manifiesto en el p-value del test de Shapiro-Wilk, al ser éste un poco más elevado que el resto. Con estos resultados sobre la mesa, podemos asegurar que no se cumplen los requisitos establecidos por LDA, y que por tanto cabe esperar un rendimiento muy bajo del método.

Los resultados obtenidos con LDA pueden verse en la tabla 17, donde se comparan los resultados de los tres métodos estudiados.

[1] La normalización llevada a cabo se rige por la función $f(x) = \frac{x - \text{mín}(x)}{\text{máx}(x) - \text{mín}(x)}$

Variable	Statistic	p-value
`Native`	0.474	5.51e-21
`Instructor`	0.949	**2.87e-05**
`Course`	0.843	2.29e-11
`Semester`	0.421	6.89e-22
`Size`	0.971	**2.71e-03**

Cuadro 16: Test de normalidad Shapiro-Wilk aplicado a las variables de entrada del conjunto de datos *tae*.

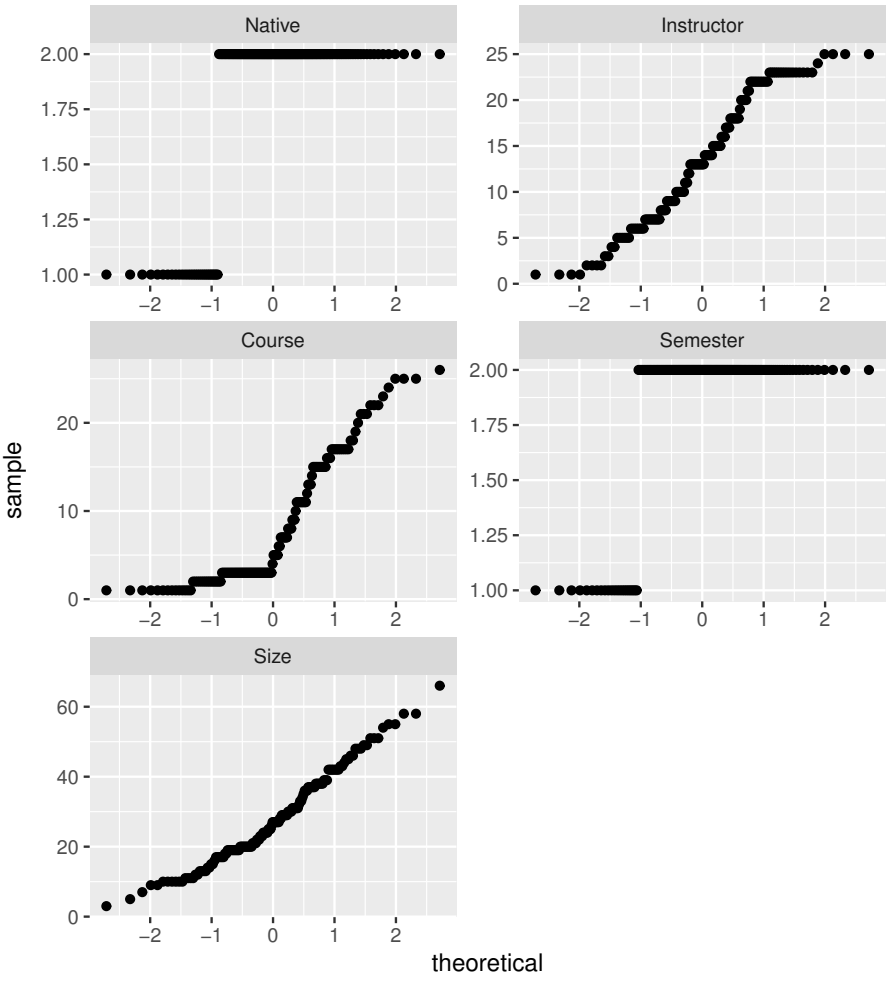


Figura 16: Gráficos Q-Q de las distintas variables de entrada.

k-fold	k-NN ($k = 1$)		LDA		QDA	
	Accuracy	Kappa	Accuracy	Kappa	Accuracy	Kappa
1	0.467	0.200	0.333	0.000	0.267	-0.100
2	0.714	0.569	0.500	0.222	0.214	-0.108
3	0.786	0.679	0.571	0.333	0.286	0.000
4	0.714	0.563	0.500	0.222	0.357	0.045
5	0.643	0.462	0.357	0.000	0.143	-0.244
6	0.603	0.408	0.357	0.000	0.286	-0.014
7	0.429	0.170	0.357	0.080	0.357	0.023
8	0.667	0.491	0.429	0.111	0.286	-0.029
9	0.571	0.349	0.429	0.188	0.286	-0.069
10	0.540	0.305	0.429	0.125	0.429	0.111
mean	**0.613**	**0.420**	**0.426**	**0.128**	**0.291**	**-0.039**

Cuadro 17: Comparación de los algoritmos de clasificación para cada uno de los *k-folds*.

p-value	k-NN	LDA
LDA	0.011	—
QDA	0.011	0.011

Cuadro 18: Resultados del test de los rangos con signo de Wilcoxon por parejas, aplicado a los resultados de Accuracy obtenidos por los diferentes algoritmos de clasificación.

4.3. Clasificación mediante análisis discriminante cuadrático (QDA)

QDA es un método muy parecido a LDA, que también requiere que los datos provengan de distribuciones normales. La principal ventaja de QDA sobre LDA es que el primero no requiere que los datos tengan la misma varianza ni covarianzas, puesto que éstas son estimadas de forma independiente. No obstante, el caso que nos ocupa no presenta ninguna distribución normal, por lo que no se esperan buenos resultados de este método.

Los resultados obtenidos con QDA pueden verse en la tabla 17, donde se comparan los resultados de los tres métodos estudiados.

4.4. Comparación de los algoritmos

Con el objetivo de mejorar los resultados de la clasificación, se ha probado a poner en una matriz de ocurrencias las variables `Instructor` y `Course`. A pesar de ello, las medidas de *Accuracy* con el algoritmo k-NN han sido muy similares a las obtenidas anteriormente. Es más, esta transformación no ha permitido el uso de los métodos LDA y QDA, ya que las matrices de ocurrencias tienen muchas columnas igual a cero, que son por tanto linealmente dependientes.

Los resultados obtenidos, tanto de *Accuracy* como *Kappa* son los mostrados en la tabla 17, cuyas medias están representadas gráficamente en la figura 17. Como puede observarse, k-NN es muy superior a los otros dos algoritmos en el caso estudiado. Para comprobar que efectivamente existen diferencias significativas entre los resultados de los distintos algoritmos, se han aplicado el test de Friedman y el test de los rangos con signo de Wilcoxon sobre la variable *Accuracy*. En el primero de los tests, se ha obtenido un p-value de 2.56e-05, comprobándose que al menos uno de los métodos funciona mejor que el resto. El test de Wilcoxon revela que, en realidad, existen diferencias significativas entre los resultados de *Accuracy* de los tres algoritmos, al arrojar los p-value mostrados en la tabla 18.

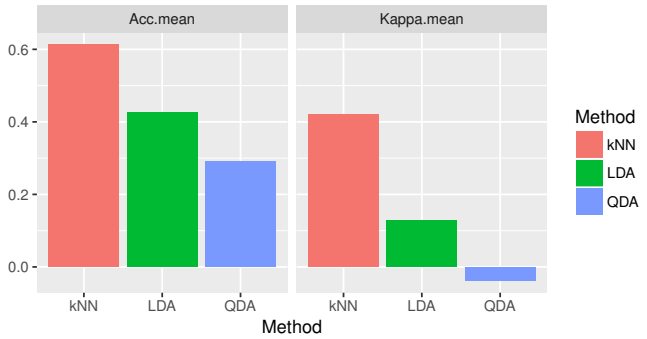


Figura 17: Comparación de los algoritmos de clasificación

Referencias

[1] Inc. RStudio. Rstudio - free and open-source ide for r. https://www.rstudio.com/, 2017.

[2] Jeffrey S Racine. Rstudio: A platform-independent ide for r and sweave. *Journal of Applied Econometrics*, 27(1):167–172, 2012.

[3] Ross Ihaka and Robert Gentleman. R: a language for data analysis and graphics. *Journal of computational and graphical statistics*, 5(3):299–314, 1996.

[4] Hadley Wickham. ggplot2 - a plotting system for r. http://ggplot2.org/, 2013.

[5] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis (Use R!)*. Springer, 2009.

[6] Jesús Alcalá-Fdez, Alberto Fernández, Julián Luengo, Joaquín Derrac, Salvador García, Luciano Sánchez, and Francisco Herrera. Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework. *Journal of Multiple-Valued Logic & Soft Computing*, 17, 2011.

[7] Marine Resources Division and Sam Waugh. Abalone data set. http://archive.ics.uci.edu/ml/datasets/Abalone, 1994.

[8] Keel-dataset - description for teaching assistant evaluation data set. http://sci2s.ugr.es/keel/dataset.php?cod=188, 2015.

[9] Wei-Yin Loh and Tjen-Sien Lim. Teaching assistant evaluation data set. http://archive.ics.uci.edu/ml/datasets/Teaching+Assistant+Evaluation, 1997.

[10] M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013.

[11] Keel-dataset - description for abalone (regression version) data set. http://sci2s.ugr.es/keel/dataset.php?cod=96, 2015.

[12] WJ Nash, TL Sellers, SR Talbot, AJ Cawthorn, and WB Ford. The population biology of abalone (haliotis species). *Blacklip Abalone (H. rubra) from the North Coast and Islands of Bass Strait. Sea Fisheries Division Technical Report*, 48, 1994.

A. Código fuente: Análisis de datos

A.1. Código fuente usado en la visualización de los datos abalone

```
################################################
#       Introducción a la Ciencia de Datos      #
#               ANÁLISIS DE DATOS               #
#                                               #
# (C) Cristian González Guerrero                #
################################################

# Build the workspace (load variables and functions)
source("../regression/build-workspace.R")


# Get the data type and dimension
class(abalone)
dim(abalone)

# Get the data types of every field
lapply(abalone, class)

# Check if there are missing values
colSums(is.na(abalone))

# Get the statistical measurements
# (numeric variables) and save them into a.s.m
# (abalone.statistical.measurements matrix)
abalone2 = subset(abalone, select = -Sex)

a.s.m = matrix(
  nrow = ncol(abalone2),
  ncol = 8
)
colnames(a.s.m) = c(
  "Min.",
  "1st Qu.",
  "Median",
  "Mean",
  "3rd Qu.",
  "Max.",
  "Var.",
  "SD"
)
rownames(a.s.m) = names(abalone2)

a.s.m[,1] = sapply(abalone2, min)              # Minimum
a.s.m[,2] = sapply(abalone2, quantile, 0.25)   # 1st Quart.
a.s.m[,3] = sapply(abalone2, median)           # Median
a.s.m[,4] = sapply(abalone2, mean)             # Mean
a.s.m[,5] = sapply(abalone2, quantile, 0.75)   # 3rd Quart.
a.s.m[,6] = sapply(abalone2, max)              # Max
a.s.m[,7] = sapply(abalone2, var)              # Variance
a.s.m[,8] = sapply(abalone2, sd)               # Std. dev.

# Get the statistical measurements
# (categorical variables)
a.s.m2 = as.data.frame( table(abalone$Sex) )
names(a.s.m2)[1] = "Sex"
a.s.m2$relFreq = a.s.m2$Freq/nrow(abalone)

```

```
# Get summarized information
str(abalone)       # Structure
summary(abalone)   # Statistical measurements

# Get correlation matrix
cor.matrix = cor(abalone2)

# By taking a look to the file, it is clear that
# the output variable is Class, thus we have to
# classify the data by this category

# PLOTS

# Categorical variables (barplots)
ggplot(abalone, aes(x = Sex, fill = Sex)) +
  geom_bar(stat = "count")

# Numeric variables Boxplots
meltData1 = melt.data.frame(data = abalone[,2:4])
meltData1$Measure = "length␣(mm)"

meltData2 = melt.data.frame(data = abalone[,5:8])
meltData2$Measure = "weight␣(grams)"

meltData = rbind(meltData1, meltData2)
ggplot(meltData, aes(factor(variable), value)) +
  xlab("") + ylab("") + geom_boxplot() + coord_flip() +
  facet_wrap( ~ Measure, ncol=1, scales="free")

# Distribution of rings
ggplot(abalone, aes(Rings, fill = Sex, color = Sex)) +
  geom_freqpoly(bins = 29)
ggplot(abalone, aes(Sex, Rings, fill = Sex)) + geom_boxplot()
ggplot(abalone, aes(Rings, fill = Sex, color = Sex)) +
  stat_ecdf() + ylab("CDF")
ggplot(abalone, aes(Rings, fill = Sex, color = Sex)) +
  stat_density(alpha = .8)

# Distribution of numeric variables against sex
# Remove outlayers
abalone.reshaped = melt(abalone[abalone$Height<0.3,])
ggplot(abalone.reshaped, aes(value, colour = Sex)) +
  geom_freqpoly(bins = 29) +
  facet_wrap( ~ variable, ncol = 2, scales="free")

# Histograms show that Male-Female distinction is not relevant,
# and does not characterize any of the above variables.
# Nonetheless, Infants' dimensions are usually lower,
# as is their age.
# Statistical test is needed to confirm this statement.


# Scatterplots
myData = melt.data.frame(
  abalone[abalone$Height<0.3,],
  id.vars=c("Sex", "Rings")
)
myData[myData$variable=="Height",] =
  within(myData[myData$variable=="Height",], {
    value = jitter(value, factor = 3)
  }
```

```
)
ggplot(myData, aes(x = value, y = Rings)) +
  geom_jitter(alpha = 0.03) +
  facet_wrap( ~ variable, ncol = 2, scales = "free")


# Check whether the Rings distributions are the same
mean(abalone[abalone$Sex=="F",]$Rings)
summary(abalone[abalone$Sex=="F",]$Rings)
mean(abalone[abalone$Sex=="M",]$Rings)
summary(abalone[abalone$Sex=="M",]$Rings)

ks.test(
  abalone[abalone$Sex=="M",]$Rings,
  abalone[abalone$Sex=="F",]$Rings
)
# In the studied sample, there are more old females than
# old males. Actually, the distributions can be proved not
# to be the same.
# This can indicate that females usually live longer than males.
# This fact should be taken into account to test if there is
# any secondary sex charasteristic, i.e. it does not suffice to
# test wheter length distribution in females is the same than
# length distribution in males, it is necessary to test this
# taking age into account.
# From the minimum age of both female and male, it can be
# suspected that primary sex characteristics are easier
# detected on males.
```

End Of File 'abalone.R'

A.2. Código fuente usado en la visualización de los datos tae

```
################################################
#       Introducción a la Ciencia de Datos     #
#               ANÁLISIS DE DATOS              #
#                                              #
# (C) Cristian González Guerrero               #
################################################

# Build the workspace
source("../classification/build-workspace.R")


# Get the data type and dimension
class(tae)
dim(tae)

# Get the data types of every field
lapply(tae, class)

# Check if there are missing values
colSums(is.na(tae))

# Get the statistical measurements (numeric variables)
attach(tae)
min       (Size)         # Minimum
quantile  (Size, 0.25)   # 1st Quartile
median    (Size)         # Median
mean      (Size)         # Mean
quantile  (Size, 0.75)   # 3rd Quartile
max       (Size)         # Max

var       (Size)         # Variane
sd        (Size)         # Standard deviation
detach(tae)

# Get summarized information
str(tae)       # Structure
summary(tae)   # Statistical measurements
summary(tae.original)

# By taking a look to the file, it is clear that
# the output variable is Class, thus we have to
# classify the data by this category


# Visualize numerical variables
myData = melt.data.frame(
  data = subset(tae, select = c(Class, Size, Native, Semester)),
  id.vars = c("Class", "Native", "Semester")
)

# Boxplots of numeric variables
ggplot(myData, aes(factor(variable), value, fill = Class)) +
  xlab("") + ylab("Size") + geom_boxplot() + coord_flip()

# Distributions of Size
ggplot(myData, aes(value)) +
```

```
  geom_freqpoly(bins = 8, aes(colour = Class)) +
  xlab("Size") + xlim(c(0,70))

ggplot(myData, aes(value)) +
  geom_freqpoly(bins = 8, aes(colour = Semester)) +
  xlab("Size") + xlim(c(0,70))
ggplot(myData, aes(value)) +
  geom_freqpoly(bins = 8, aes(colour = Native)) +
  xlab("Size") + xlim(c(0,70))
ks.test(
  subset(tae, subset = Semester=="Summer")$Size,
  subset(tae, subset = Semester=="Regular")$Size
)
ks.test(
  subset(tae, subset = Native=="English␣speaker")$Size,
  subset(tae, subset = Native=="non-English␣speaker")$Size
)

# Visualize pseudo-numeric variables
myData = melt.data.frame(
  data = subset(
    tae.original,
    select = c(Class, Course, Instructor)
  ),
  id.vars = "Class"
)
ggplot(myData, aes(factor(variable), value, fill = Class)) +
  xlab("") + ylab("number") + geom_boxplot() + coord_flip()
ggplot(myData, aes(value, colour = Class)) +
  geom_freqpoly(bins = 10) + xlab("number") +
  facet_wrap(~ variable, ncol = 1)


########################
# Barplots of factors #
########################

## Native & Semester
myData = melt.data.frame(
  data = subset(tae, select = c(-Size, -Instructor, -Course)),
  id.vars = "Class"
)

ggplot(myData, aes(value, fill = Class)) +
  geom_bar(stat = "count", position = "dodge") +
  xlab("") +
  facet_wrap(~ variable, ncol = 2, scales = "free")


## Class
ggplot(tae, aes(Class, fill = Class)) +
  geom_bar(stat = "count")


## Course
freq = ave(rep(1, times=nrow(tae)), tae$Course, FUN=sum)
myData = tae[order(freq, tae$Course, decreasing = TRUE), ]
myData = melt.data.frame(
  data = subset(myData, select = c(Class, Course)),
  id.vars = "Class"
)
```

```
118
119 ggplot(myData, aes(value, fill = Class)) +
120   geom_bar(stat = "count", position = "dodge") +
121   xlab("Course") +
122   scale_x_discrete(
123     limits = as.factor(
124       order(
125         count(tae, "Course")$freq,
126         decreasing = T
127       )
128     )
129   )
130
131
132 myData = as.data.frame(table(tae$Class, tae$Course))
133 names(myData) = c("Class", "Course", "freq")
134 freq = ave(myData$freq, myData$Course, FUN = sum)
135 myData = myData[order(freq, myData$Course, decreasing = TRUE), ]
136 ggplot(myData, aes(x = Course, y = freq, color = Class)) +
137   geom_point() +
138   geom_line(aes(group = Class)) +
139   scale_x_discrete(limits = myData$Course) +
140   ylab("Frequency")
141
142 myData$rel.freq =
143   myData$freq/freq[order(freq, myData$Course, decreasing = T)]
144 ggplot(myData, aes(x = Course, y = rel.freq, fill = Class)) +
145   geom_col(width = 2.5) +
146   scale_x_discrete(limits = myData$Course) +
147   ylab("Relative␣frequency")
148
149
150 ## Instructor
151 myData = as.data.frame(table(tae$Class, tae$Instructor))
152 names(myData) = c("Class", "Instructor", "freq")
153 freq = ave(myData$freq, myData$Instructor, FUN = sum)
154 myData =
155   myData[order(freq, myData$Instructor, decreasing = T), ]
156 myData[myData$freq==0,]$freq = 0.03
157 ggplot(myData, aes(x = Instructor, y = freq))  +
158   geom_col(width = 2, position = "dodge", aes(fill = Class)) +
159   scale_x_discrete(limits = myData$Instructor) +
160   ylab("Frequency")
161
162 myData[myData$freq==0.03,]$freq = 0
163 myData$rel.freq =
164   myData$freq/freq[order(freq, myData$Instructor, decreasing = T)]
165 ggplot(myData, aes(x = Instructor, y = rel.freq, fill = Class)) +
166   geom_col(width = 2.5) +
167   scale_x_discrete(limits = myData$Instructor) +
168   ylab("Relative␣frequency")
169
170 ggplot(tae, aes(as.integer(Instructor), color = Class)) +
171   geom_density()
172 ggplot(tae, aes(as.integer(Course), color = Class)) +
173   geom_density()
```

End Of File 'tae.R'

B. Código fuente: Regresión

B.1. Código fuente de las funciones usadas en los ejercicios de regresión

```
##################################################
#       Introducción a la Ciencia de Datos       #
#      FUNCIONES Y VARIABLES PARA REGRESIÓN      #
#                                                #
# (C) Cristian González Guerrero                 #
##################################################

# Load required libraries
library(utils)
library(stats)
library(foreign)
library(ggplot2)
library(reshape)
library(kknn)

# ABALONE

# Load the dataset and provide it with the
# structure from Keel
abalone = read.csv(
  "datasets/abalone/abalone.dat",
  comment.char="@"
)
names(abalone) = c(
  "Sex",
  "Length",
  "Diameter",
  "Height",
  "Whole_weight",
  "Shucked_weight",
  "Viscera_weight",
  "Shell_weight",
  "Rings"
)
abalone$Sex = factor(
  abalone$Sex,
  levels = c(1, 2, 3),
  labels = c("M", "F", "I")
)

# Load traning data and test data
abalone.tra = list()
abalone.tst = list()
for (i in 1:5) {
  for (j in 1:2) {
    filename = paste(
      "datasets/abalone/abalone-5-",
      as.character(i),
      ifelse(j==1, "tra", "tst"),
      ".dat",
      sep = ""
    )
    x = read.csv(
      filename,
```

```
      comment.char="@"
    )
    names(x) = names(abalone)
    x$Sex = factor(
      x$Sex,
      levels = c(1, 2, 3),
      labels = c("M", "F", "I")
    )
    if(j==1) {
      abalone.tra[[i]] = x
    } else {
      abalone.tst[[i]] = x
    }
  }
}


# FUNCTIONS

# Run k-fold cross validation on LM fit
run_lm_fold = function(
  i,
  tra = abalone.tra,
  tst = abalone.tst,
  model = Rings~.,
  tt = "test"
) {
  x_tra = tra[[i]]
  x_tst = tst[[i]]

  if (tt != "test") {
    test = x_tra
  } else {
    test = x_tst
  }

  # Perform LM fit
  lm.fit = lm(model, x_tra)
  output.var = as.character(model[2])

  # Get MSE Error
  yprime = predict(lm.fit, test)
  sum(abs(test[,output.var]-yprime)^2)/length(yprime)
}


run_knn_fold = function(
  i,
  tra = abalone.tra,
  tst = abalone.tst,
  model = Rings~.,
  tt = "test",
  ...
) {
  x_tra = tra[[i]]
  x_tst = tst[[i]]

  if (tt != "test") {
    test = x_tra
  } else {
    test = x_tst
```

```
116   }
117
118   # Perform knn fit
119   knn.fit = kknn(model, x_tra, test, ...)
120   output.var = as.character(model[2])
121
122   # Get MSE Error
123   yprime = knn.fit$fitted.values
124   sum(abs(test[,output.var]-yprime)^2)/length(yprime)
125 }
126
127 # Add non linearities to abalone training/test set
128 add.non.linearities = function(df) {
129   # Remove new fields
130   df = df[,1:9]
131
132   df$Whole_weight.2     = (df$Whole_weight)^(1/2)
133   df$Whole_weight.3     = (df$Whole_weight)^(1/3)
134   df$Whole_weight.log   = log(df$Whole_weight)
135   df$Shucked_weight.2   = (df$Shucked_weight)^(1/2)
136   df$Shucked_weight.3   = (df$Shucked_weight)^(1/3)
137   df$Shucked_weight.log = log(df$Shucked_weight)
138   df$Viscera_weight.2   = (df$Viscera_weight)^(1/2)
139   df$Viscera_weight.3   = (df$Viscera_weight)^(1/3)
140   df$Viscera_weight.log = log(df$Viscera_weight)
141   df$Shell_weight.2     = (df$Shell_weight)^(1/2)
142   df$Shell_weight.3     = (df$Shell_weight)^(1/3)
143   df$Shell_weight.log   = log(df$Shell_weight)
144
145   return(df)
146 }
147
148 rm(i, j, filename, x)
```

End Of File 'build-workspace.R'

B.2. Código fuente usado en el ejercicio de regresión lineal

```
##################################################
#       Introducción a la Ciencia de Datos       #
#                 REGRESIÓN LINEAR               #
#                                                #
# (C) Cristian González Guerrero                 #
##################################################

# Build the workspace
source("../regression/build-workspace.R")


# Get simple linear regression models
simple.linear.fit = list()
for (i in 1:(length(abalone)-1)) {
  simple.linear.fit[[i]] = lm(abalone$Rings~abalone[,i])
  names(simple.linear.fit)[i] = names(abalone)[i]
}

summary(simple.linear.fit$Sex)
summary(simple.linear.fit$Length)
summary(simple.linear.fit$Diameter)
summary(simple.linear.fit$Height)
summary(simple.linear.fit$Whole_weight)
summary(simple.linear.fit$Shucked_weight)
summary(simple.linear.fit$Viscera_weight)
summary(simple.linear.fit$Shell_weight)


MSE = matrix(nrow = 8, ncol = 2)
colnames(MSE) = c("train", "test")
rownames(MSE) = names(abalone)[1:8]
for (tt in colnames(MSE)) {
  for (i in rownames(MSE)) {
    MSE[i, tt] = mean(sapply(
      1:5,
      run_lm_fold,
      model = Rings~eval(parse(text = i)),
      tt = tt
    ))
  }
}

# Using the shell weight seems to be the best model,
# since it produces less error.

# Scatterplots
myData = melt.data.frame(
  abalone[abalone$Height<0.3,],
  id.vars=c("Sex", "Rings")
)
myData[myData$variable=="Height",] =
  within(myData[myData$variable=="Height",], {
    value = jitter(value, factor = 3)
  }
  )
ggplot(myData, aes(x = value, y = Rings)) +
```

```
  geom_jitter(alpha = 0.03) +
  geom_smooth(method = lm) +
  facet_wrap( ~ variable, ncol = 2, scales = "free") +
  xlab("")


# Analysis of preprocessing
abalone = add.non.linearities(abalone)
abalone.tra = lapply(abalone.tra, add.non.linearities)
abalone.tst = lapply(abalone.tst, add.non.linearities)


## Calculate correlations
Y.vars = grepl("Rings", names(abalone))
X.vars = grepl("_weight", names(abalone))
cor.Shucked_weight = cor(
  subset(abalone, select = X.vars),
  subset(abalone, select = Y.vars)
)

## New models, based on new variables
simple.linear.fit = list()
for (i in 1:(length(abalone))) {
  simple.linear.fit[[i]] = lm(abalone$Rings~abalone[,i])
  names(simple.linear.fit)[i] = names(abalone)[i]
}

summary(simple.linear.fit$Whole_weight.log)
summary(simple.linear.fit$Shucked_weight.2)
summary(simple.linear.fit$Shucked_weight.3)
summary(simple.linear.fit$Shucked_weight.log)
summary(simple.linear.fit$Viscera_weight.log)
summary(simple.linear.fit$Shell_weight.3)
summary(simple.linear.fit$Shell_weight.log)

MSE = matrix(nrow = ncol(abalone), ncol = 2)
colnames(MSE) = c("train", "test")
rownames(MSE) = names(abalone)
for (tt in colnames(MSE)) {
  for (i in rownames(MSE)) {
    MSE[i, tt] = mean(sapply(
      1:5,
      run_lm_fold,
      model = Rings~eval(parse(text = i)),
      tt = tt
    ))
  }
}


# Plot
myData = melt.data.frame(
  abalone[abalone$Height<0.3,],
  id.vars=c("Sex", names(abalone)[Y.vars])
)

selected.fields = c(
  "Whole_weight",
  "log(Whole_weight)",
  "Shucked_weight",
  "log(Shucked_weight)",
```

```
  "Viscera_weight",
  "log(Viscera_weight)",
  "Shell_weight",
  "(Shell_weight)^(1/3)"
)
names(selected.fields) = c(
  "Whole_weight",
  "Whole_weight.log",
  "Shucked_weight",
  "Shucked_weight.log",
  "Viscera_weight",
  "Viscera_weight.log",
  "Shell_weight",
  "Shell_weight.3"
)

ggplot(
  subset(myData, variable %in% names(selected.fields)),
  aes(x = value, y = Rings)
) +
  geom_jitter(alpha = 0.03) + geom_smooth(method = lm) +
  facet_wrap(
    ~ variable,
    ncol = 2,
    scales = "free",
    labeller = as_labeller(selected.fields)
  ) +
  xlab("")

# Linear regression using multiple variables
## Only linear models
abalone = abalone[,1:9]

myFit = lm(Rings~., abalone)
summary(myFit)

myModel = Rings ~
  Diameter +
  Height +
  Whole_weight +
  Shucked_weight +
  Viscera_weight +
  Shell_weight
myFit = lm(myModel, abalone)
summary(myFit)


MSE = matrix(nrow = 1, ncol = 2)
colnames(MSE) = c("train", "test")
for (tt in colnames(MSE)) {
  MSE[1, tt] = mean(sapply(
    1:5,
    run_lm_fold,
    model = myModel,
    tt = tt
  ))
}

# Add non-linearities
abalone = add.non.linearities(abalone)

```

```
myModel = Rings ~
  Length +
  Height +
  Whole_weight.log +
  Shucked_weight.log +
  Viscera_weight.log +
  Shell_weight.3
myFit = lm(Rings~., abalone)
summary(myFit)

myFit = lm(myModel, abalone)
summary(myFit)

MSE = matrix(nrow = 1, ncol = 2)
colnames(MSE) = c("train", "test")
for (tt in colnames(MSE)) {
  MSE[1, tt] = mean(sapply(
    1:5,
    run_lm_fold,
    model = myModel,
    tt = tt
  ))
}

myModel = Rings ~
  poly(Length, 2) +
  poly(Height, 2) +
  poly(Whole_weight, 2) +
  poly(Shucked_weight, 2) +
  poly(Viscera_weight, 2) +
  poly(Shell_weight, 2)
myFit = lm(myModel, abalone)
summary(myFit)

MSE = matrix(nrow = 1, ncol = 2)
colnames(MSE) = c("train", "test")
for (tt in colnames(MSE)) {
  MSE[1, tt] = mean(sapply(
    1:5,
    run_lm_fold,
    model = myModel,
    tt = tt
  ))
}

# Add interactions
myModel = Rings ~
  Length +
  Height +
  Whole_weight.log *
  Shucked_weight.log *
  Viscera_weight *
  Shell_weight.3

myFit = lm(myModel, abalone)
summary(myFit)

MSE = matrix(nrow = 1, ncol = 2)
colnames(MSE) = c("train", "test")
for (tt in colnames(MSE)) {
  MSE[1, tt] =
```

```
240     mean(sapply(1:5, run_lm_fold, model = myModel, tt = tt))
241 }
```

End Of File 'linear-regression.R'

B.3. Código fuente usado en el ejercicio de regresión k-NN

```
#################################################
#       Introducción a la Ciencia de Datos      #
#                 REGRESIÓN k-NN                #
#                                               #
# (C) Cristian González Guerrero                #
#################################################

# Build the workspace
source("../regression/build-workspace.R")

abalone = abalone[abalone$Height<0.3,]

# Get simple knn models
simple.knn.fit = list()
for (i in 1:(length(abalone)-1)) {
  simple.knn.fit[[i]] =
    kknn(abalone$Rings~abalone[,i], abalone, abalone)
  names(simple.knn.fit)[i] = names(abalone)[i]
}


## Scatterplots
myData1 = melt.data.frame(
  abalone,
  id.vars=c("Sex", "Rings")
)

myData2 = data.frame()
for (variable in names(simple.knn.fit)[2:8]) {
  Prediction = simple.knn.fit[[variable]]$fitted.values
  myData2 = rbind(myData2, data.frame(Prediction))
}

myData = cbind(myData1, myData2)

Data = "Original␣data"
b = "Prediction"
colorPalette <- c("#000000", "#56B4E9")

ggplot(myData) +
  geom_point(
    aes(x = value, y = Rings, color = Data),
    alpha = 0.03
  ) +
  geom_point(
    aes(x = value, y = Prediction, color = b),
    alpha = 0.03
  ) +
  facet_wrap( ~ variable, ncol = 2, scales = "free") +
  xlab("") +
  scale_colour_manual(values=colorPalette) +
  guides(colour = guide_legend(override.aes = list(alpha = 1)))

## MSE
MSE = matrix(nrow = 8, ncol = 2)
colnames(MSE) = c("train", "test")
```

```r
rownames(MSE) = names(abalone)[1:8]
for (tt in colnames(MSE)) {
  for (i in rownames(MSE)) {
    MSE[i, tt] = mean(sapply(
      1:5,
      run_knn_fold,
      model = Rings~eval(parse(text = i)),
      tt = tt,
      k = 7    # Modify k
    ))
  }
}

## Comparative k
k = 1:20
MSE.Shell_weight = data.frame(k)
for (tt in c("train", "test")) {
  for (i in k) {
    MSE.Shell_weight[i,tt] = mean(sapply(
      1:5,
      run_knn_fold,
      model = Rings~Shell_weight,
      tt = tt,
      k = i
    ))
  }
}

Data = "test"
b = "train"
ggplot(MSE.Shell_weight) +
  geom_point(aes(x = k, y = test,  color = Data)) +
  geom_line (aes(x = k, y = test,  color = Data)) +
  geom_point(aes(x = k, y = train, color = b)) +
  geom_line (aes(x = k, y = train, color = b)) +
  ylab("MSE") +
  scale_colour_manual(values=colorPalette)



# Non-linearities
abalone = add.non.linearities(abalone)
abalone.tra = lapply(abalone.tra, add.non.linearities)
abalone.tst = lapply(abalone.tst, add.non.linearities)

MSE = matrix(nrow = ncol(abalone), ncol = 2)
colnames(MSE) = c("train", "test")
rownames(MSE) = names(abalone)
for (tt in colnames(MSE)) {
  for (i in rownames(MSE)) {
    MSE[i, tt] = mean(sapply(
      1:5,
      run_knn_fold,
      model = Rings~eval(parse(text = i)),
      tt = tt,
      k = 7    # Modify k
    ))
  }
}


```

```r
# Multiple variables
abalone = abalone[,1:9]
abalone.tra = lapply(abalone.tra, function(df){return(df[,1:9])})
abalone.tst = lapply(abalone.tst, function(df){return(df[,1:9])})
MSE.record = data.frame()

knn.MSE.data.frame = function(
  model = Rigns~.,
  k = 1:20, i = 1:5,
  tt = c("train", "test")
) {
  MSE = data.frame(k)
  tt_ = tt
  i_  = i
  for (tt in tt_) {
    for (i in k) {
      MSE[i, tt] = mean(sapply(
        i_,
        run_knn_fold,
        model = model,
        tt = tt,
        k = i    # Modify k
      ))
    }
  }
  MSE$model = as.character(myModel)[3]

  return(MSE)
}

plot.MSE.data.frame = function(MSE, model = "") {
  model = as.character(myModel)
  if(length(model) == 3) {
    model = paste(model[2], model[1], model[3])
  }
  myData = melt.data.frame(
    MSE,
    id.vars = c("k", "model"),
    variable_name = "Data"
  )

  ggplot(myData, aes(x = k, y = value, color = Data)) +
    geom_point() + geom_line () + ylab("MSE") +
    ggtitle(model)
}

# Simple regression
myModel = Rings~Shell_weight

MSE = knn.MSE.data.frame(myModel)
MSE.record = rbind(MSE.record, MSE)
plot.MSE.data.frame(MSE, myModel)

# Every variable
myModel = Rings~.

MSE = knn.MSE.data.frame(myModel)
MSE.record = rbind(MSE.record, MSE)
plot.MSE.data.frame(MSE, myModel)


```

```
## No interactions
myModel = Rings ~
  Diameter +
  Height +
  Whole_weight +
  Shucked_weight +
  Viscera_weight +
  Shell_weight

MSE = knn.MSE.data.frame(myModel)
MSE.record = rbind(MSE.record, MSE)
plot.MSE.data.frame(MSE)


## Non-linearities
abalone = add.non.linearities(abalone)
abalone.tra = lapply(abalone.tra, add.non.linearities)
abalone.tst = lapply(abalone.tst, add.non.linearities)

myModel = Rings ~
  Length +
  Height +
  Whole_weight.log +
  Shucked_weight.log +
  Viscera_weight.log +
  Shell_weight.3

MSE = knn.MSE.data.frame(myModel)
MSE.record = rbind(MSE.record, MSE)
plot.MSE.data.frame(MSE)


## Non-linearities (poly)
myModel = Rings ~
  poly(Length, 2) +
  poly(Height, 2) +
  poly(Whole_weight, 2) +
  poly(Shucked_weight, 2) +
  poly(Viscera_weight, 2) +
  poly(Shell_weight, 2)

MSE = knn.MSE.data.frame(myModel)
MSE.record = rbind(MSE.record, MSE)
plot.MSE.data.frame(MSE)

#--
myModel = Rings ~
  poly(Length, 3) +
  poly(Height, 3) +
  poly(Whole_weight, 3) +
  poly(Shucked_weight, 3) +
  poly(Viscera_weight, 3) +
  poly(Shell_weight, 3)

MSE = knn.MSE.data.frame(myModel)
MSE.record = rbind(MSE.record, MSE)
plot.MSE.data.frame(MSE)


## Interactions
myModel = Rings ~
```

```
  Length +
  Height +
  Whole_weight.log *
  Shucked_weight.log *
  Viscera_weight *
  Shell_weight.3

MSE = knn.MSE.data.frame(myModel)
MSE.record = rbind(MSE.record, MSE)
plot.MSE.data.frame(MSE)

## Compare k
myData = subset(MSE.record, select = -train)
myData = myData[myData$k>2,]
myData = melt.data.frame(
  myData,
  id.vars = c("k", "model"),
  variable_name = "Data"
)
myData$Data = paste(myData$Data, "␣in␣", myData$model)

ggplot(myData, aes(x = k, y = value, color = Data)) +
  geom_point() + geom_line () + ylab("MSE") +
  theme(legend.position = "bottom", legend.direction = "vertical")
```

End Of File 'knn-regression.R'

B.4. Código fuente de la comparación entre los distintos algoritmos de regresión

```
################################################
#       Introducción a la Ciencia de Datos     #
#            COMPARATIVA DE ALGORITMOS         #
#                                              #
# (C) Cristian González Guerrero               #
################################################

# Build the workspace
source("../regression/build-workspace.R")

abalone.tra = lapply(abalone.tra, add.non.linearities)
abalone.tst = lapply(abalone.tst, add.non.linearities)


# Comparison of data points

myModel = Rings ~
  Length +
  Height +
  Whole_weight.log *
  Shucked_weight.log *
  Viscera_weight *
  Shell_weight.3

myTrain = abalone.tra[[1]]
myTest  = abalone.tst[[1]][sample(1:nrow(abalone.tst[[1]]), 20), ]

## Linear regression
# Plot some points of the last model
myFit = lm(myModel, myTrain)
myResult = predict(myFit, myTest)

myData1 = myTest
myData1$Rings = myResult
myData1 = cbind(
  myData1,
  data.type = "prediction",
  method = "Linear Model"
)
myData1 = rbind(
  myData1,
  cbind(
    myTest,
    data.type = "original value",
    method = "Linear Model"
  )
)
myData1 = melt.data.frame(
  data = myData1,
  id.vars = c("Shell_weight", "data.type", "method"),
  measure.vars = "Rings"
)

## k-NN
# Plot some points of the last model
myResult = kknn(myModel, myTrain, myTest, k = 7)
```

```r

myData2 = myTest
myData2$Rings = myResult$fitted.values
myData2 = cbind(
  myData2,
  data.type = "prediction",
  method = "k-NN"
)
myData2 = rbind(
  myData2,
  cbind(
    myTest,
    data.type = "original value",
    method = "k-NN"
  )
)
myData2 = melt.data.frame(
  data = myData2,
  id.vars = c("Shell_weight", "data.type", "method"),
  measure.vars = "Rings"
)


## Plot
myData = rbind(myData1, myData2)

ggplot(
  myData,
  aes(x = Shell_weight, y = value, color = data.type)
) +
  geom_point() +
  ylab("Rings") +
  xlab("Shell weight") +
  geom_line(aes(group = Shell_weight, color = "error")) +
  labs(color = "Data type") +
  facet_wrap( ~ method, ncol = 1)


# Error distribution
myTest = abalone.tst[[1]]
lmErrors  =
  myTest$Rings-predict(myFit, myTest)
knnErrors =
  myTest$Rings-kknn(myModel, myTrain, myTest)$fitted.values

### Tests
wilcox.test(lmErrors^2, knnErrors^2, paired = FALSE)
ks.test(lmErrors^2, knnErrors^2)

### Plot
myData1 = data.frame(method = "Linear Model", Err = lmErrors)
myData2 = data.frame(method = "k-NN", Err = knnErrors)
myData = melt.data.frame(
  data = rbind(myData1, myData2),
  id.vars = "method"
)
ggplot(myData, aes(x=value^2)) +
  geom_density(aes(color = method, fill = method), alpha = 0.3) +
  xlab("Squared error") + xlim(0,20)


```

```r
# General algorithm comparison (using MSE from every database)
## Read data
testResults = read.csv("../regression/regr_test_alumnos.csv")
testTable = testResults[, 2:ncol(testResults)]
rownames(testTable) = testResults[,1]

trainResults = read.csv("../regression/regr_train_alumnos.csv")
trainTable = trainResults[, 2:ncol(testResults)]
rownames(trainTable) = trainResults[,1]

## Comparison between LM and kNN
## (kNN is assumed to provide better results)
difs = (testTable[,1] - testTable[,2])/testTable[,1]

wilc_1_2 = cbind(
  ifelse(difs<0, abs(difs)+0.1, 0.1),
  ifelse(difs>0, abs(difs)+0.1, 0.1)
)
colnames(wilc_1_2) = colnames(testTable)[1:2]

### Wikcoxon tests
LMvsKNNtst = wilcox.test(
  wilc_1_2[,1],
  wilc_1_2[,2],
  alternative = "two.sided",
  paired = TRUE
)

Rplus  = LMvsKNNtst$statistic
pvalue = LMvsKNNtst$p.value

LMvsKNNtst = wilcox.test(
  wilc_1_2[,2],
  wilc_1_2[,1],
  alternative = "two.sided",
  paired = TRUE
)

Rminus = LMvsKNNtst$statistic

WilcoxonTestOutput = cbind(Rplus, Rminus, pvalue)


## Comparison amongst LM, kNN and M5'
friedman.test(as.matrix(testTable))

#### p-value < 0.05 => significative difference exist

groups = rep(1:ncol(testTable), each = nrow(testTable))
pairwise.wilcox.test(
  as.matrix(testTable),
  groups,
  p.adjust.method = "holm",
  paired = TRUE
)



#### M5' seems to work better (confidence: 84%)
#### LM and kNN seems not to present any difference
```

End Of File 'comparison.R'

C. Código fuente: Clasificación

C.1. Código fuente de de las funciones usadas en el ejercicio de clasificación

```
################################################
#       Introducción a la Ciencia de Datos     #
#    FUNCIONES Y VARIABLES PARA CLASIFICACIÓN  #
#                                              #
# (C) Cristian González Guerrero               #
################################################

# Load required libraries
library(utils)
library(stats)
library(foreign)
library(ggplot2)
library(reshape)
library(GGally)
library(class)
library(caret)
library(MASS)
library(plyr)
library(ISLR)
library(klaR)

#TAE

# Load the dataset and provide it with the
# structure from Keel
tae = read.csv(
  "datasets/tae/tae.dat",
  comment.char="@"
)
names(tae) = c(
  "Native",
  "Instructor",
  "Course",
  "Semester",
  "Size",
  "Class"
)


tae$Class      = factor(
  tae$Class,
  levels = c(1,2,3),
  labels = c("low", "medium", "high")
)
tae.original = tae
tae$Native     = factor(
  tae$Native,
  levels = c(1,2),
  labels = c(
    "English␣speaker",
    "non-English␣speaker"
  )
)
tae$Semester   = factor(
```

```
  tae$Semester,
  levels = c(1,2),
  labels = c("Summer", "Regular")
)
tae$Instructor = factor(tae$Instructor)
tae$Course     = factor(tae$Course)


# Load traning data and test data
tae.tra = list()
tae.tra.cl = list()
tae.tra.dt = list()
tae.tst = list()
tae.tst.cl = list()
tae.tst.dt = list()
for (i in 1:10) {
  for (j in 1:2) {
    filename = paste(
      "datasets/tae/tae-10-",
      as.character(i),
      ifelse(j==1, "tra", "tst"),
      ".dat",
      sep = ""
    )
    x = read.csv(
      filename,
      comment.char="@"
    )
    names(x) = names(tae)
    # Everything except the Class will be
    # treated as numeric for classification
    x$Class      = factor(
      x$Class,
      levels = c(1,2,3),
      labels = c("low", "medium", "high")
    )

    if(j==1) {
      tae.tra[[i]]    = x
      tae.tra.dt[[i]] = subset(x, select = -Class)
      tae.tra.cl[[i]] = x$Class
    } else {
      tae.tst[[i]]    = x
      tae.tst.dt[[i]] = subset(x, select = -Class)
      tae.tst.cl[[i]] = x$Class
    }
  }
}


# FUNCTIONS
normalize = function(x) {
  return((x-min(x))/(max(x)-min(x)))
}

normalize.tae = function(df) {
  df$Instructor = df$Instructor*4
  df$Course     = df$Course*4
  df$Native     = df$Native*4
  df$Semester   = df$Semester
  df$Size       = normalize(df$Size)
```

```
116
117   return(df)
118 }
119
120 # Accuracy function
121 accuracy = function(ConfusionMatrix) {
122   return(sum(diag(ConfusionMatrix))/sum(ConfusionMatrix))
123
124   meanAccuracy      = 0
125   for (k in 1:ncol(ConfusionMatrix)) {
126     TP = ConfusionMatrix[k,k]
127     FP = sum(ConfusionMatrix[k,]) - TP
128     FN = sum(ConfusionMatrix[,k]) - TP
129     TN = sum(ConfusionMatrix[-k,-k])
130     Accuracy = (TP+TN)/(TP+TN+FP+FN)
131
132     meanAccuracy = meanAccuracy + Accuracy/ncol(ConfusionMatrix)
133   }
134   return(meanAccuracy)
135 }
136
137 set.seed(1)
138
139 rm(i, j, filename, x)
```

End Of File 'build-workspace.R'

C.2. Código fuente usado en el ejercicio de clasificación

```
################################################
#       Introducción a la Ciencia de Datos     #
#                 CLASIFICACIÓN                #
#                                              #
# (C) Cristian González Guerrero               #
################################################

# Build the workspace
source("../classification/build-workspace.R")

# Some info about the classes
round(prop.table(table(tae$Class)) * 100, digits = 1)

# Normalize numeric data
tae$Size = normalize(tae$Size)
tae.tra.dt = lapply(tae.tra.dt, normalize.tae)
tae.tst.dt = lapply(tae.tst.dt, normalize.tae)



################################
# k Nearest Neighbourgs (k-NN) #
################################

# Apply k-NN algorithm to the first set
tae_test_pred = knn(
  train = tae.tra.dt[[1]], # Dataframe
  test  = tae.tst.dt[[1]], # Dataframe
  cl    = tae.tra.cl[[1]], # Vector (factor)
  k     = 21
)

# See the results on a contingency table (confusion matrix)
table(tae_test_pred, tae.tst.cl[[1]])

# Apply k-NN algorithm to every dataset and check the
# overall accuracy for several values of k. Repeat this process
# m times, to get the random part working.
k.max = 35
Acc = data.frame(k = 1:k.max)
for (m in 1:9) {
  for (k in 1:k.max) {
    acc = rep(0, 10)
    for (i in 1:10) {
      tae_test_pred = knn(
        train = tae.tra.dt[[i]], # Dataframe
        test  = tae.tst.dt[[i]], # Dataframe
        cl    = tae.tra.cl[[i]], # Vector (factor)
        k     = k
      )
      acc[i] = accuracy(
        table(tae_test_pred, tae.tst.cl[[i]])
      )
    }
    Acc[k, paste("Acc", m, sep = "")] = mean(acc)
  }
```

```
}

Acc$Acc_mean = rowMeans(Acc[,-1])

myData = melt.data.frame(
  data = Acc,
  id.vars = "k"
)
myData$lineSize = 1
myData[myData$variable=="Acc_mean",]$lineSize = 2

max(Acc$Acc_mean)
which.max(Acc$Acc_mean)

ggplot(
  myData,
  aes(x = k, y = value, color = variable, size = lineSize)
) +
  geom_line() + scale_size(range = c(0.5, 2), guide="none") +
  ylab("Accuracy")
## It seems that k=1 performs the best, with Acc = 0.61
# This plot could also be done with training data
# in order to test for overfitting


# Fill in the table in the document
Acc.knn = matrix(ncol = 10, nrow = 10)
rownames(Acc.knn) = paste("K", 1:10, sep = "")
colnames(Acc.knn) =
  c(paste("Acc", 1:9, sep = ""), "Acc_mean")
Kappa.knn = matrix(ncol = 10, nrow = 10)
rownames(Kappa.knn) = paste("K", 1:10, sep = "")
colnames(Kappa.knn) =
  c(paste("Kappa", 1:9, sep = ""), "Kappa_mean")
for (m in 1:9) {
    myAcc   = rep(0, 10)
    myKappa = rep(0, 10)
    for (i in 1:10) {
      tae_test_pred = knn(
        train = tae.tra.dt[[i]], # Dataframe
        test  = tae.tst.dt[[i]], # Dataframe
        cl    = tae.tra.cl[[i]], # Vector (factor)
        k     = 1
      )
      cm = confusionMatrix(
        table(tae_test_pred, tae.tst.cl[[i]])
      )
      myAcc[i]   = cm$overall[1]
      myKappa[i] = cm$overall[2]
    }
    Acc.knn[,m]   = myAcc
    Kappa.knn[,m] = myKappa
}
Acc.knn[,10]   = rowMeans(Acc.knn[,1:9])
Kappa.knn[,10] = rowMeans(Kappa.knn[,1:9])


#######################################
# Linear Discriminant Analysis (LDA) #
#######################################

```

```r
# Check for normality of (pseudo-)numeric variables
sapply(tae.original[1:5], shapiro.test)

myData = melt.data.frame(
  data = tae.original,
  measure.vars = c(
    "Native",
    "Instructor",
    "Course",
    "Semester",
    "Size"
  )
)
ggplot(myData) +
  stat_qq(aes(sample = value)) +
  facet_wrap(~variable, ncol = 2, scales = "free")

# LDA fit
lda.fit = lda(
  Class ~ .,
  data = tae.tra[[1]]
)

lda.pred = predict(lda.fit, tae.tst.dt[[1]])

table(lda.pred$class, tae.tst.cl[[1]])

# Do it for every dataset in k-fold CV
acc = rep(0, 10)
for (i in 1:10) {
  lda.fit = lda(
    Class ~ .,
    data = tae.tra[[i]]
  )
  lda.pred = predict(lda.fit, tae.tst.dt[[i]])

  acc[i] = accuracy(
    table(lda.pred$class, tae.tst.cl[[i]])
  )
}
Acc.LDA = data.frame(fold = 1:10, Acc = acc)

mean(Acc.LDA$Acc)

# LDA performs poorly if every variable is considered.
# However, there are not real numeric variables, only
# ordered factors, with no normal distribution.
# Other variables should be computed from available
# information (and pray for them to have a quasi-normal
# distribution). Otherwise, LDA will not perform correctly.
# Actually, mean(acc) equals 0.348 if the model considers
# solely the "numeric" variables, i.e. Course+Instructor+Size,
# meaning a loss of less than 5% in accuracy.


# Fill in the table in the document
Acc.lda = matrix(ncol = 10, nrow = 10)
rownames(Acc.lda) = paste("K", 1:10, sep = "")
colnames(Acc.lda) =
  c(paste("Acc", 1:9, sep = ""), "Acc_mean")
Kappa.lda = matrix(ncol = 10, nrow = 10)
```

```
rownames(Kappa.lda) = paste("K", 1:10, sep = "")
colnames(Kappa.lda) =
  c(paste("Kappa", 1:9, sep = ""), "Kappa_mean")
for (m in 1:9) {
  myAcc   = rep(0, 10)
  myKappa = rep(0, 10)
  for (i in 1:10) {
    lda.fit = lda(
      Class ~ .,
      data = tae.tra[[i]]
    )
    lda.pred = predict(lda.fit, tae.tst.dt[[i]])

    cm = confusionMatrix(
      table(lda.pred$class, tae.tst.cl[[i]])
    )
    myAcc[i]   = cm$overall[1]
    myKappa[i] = cm$overall[2]
  }
  Acc.lda[,m]   = myAcc
  Kappa.lda[,m] = myKappa
}
Acc.lda[,10] = rowMeans(Acc.lda[,1:9])
Kappa.lda[,10] = rowMeans(Kappa.lda[,1:9])


## Some plots
partimat(
  Class~Size+Instructor,
  data = tae.tra[[1]],
  method = "lda"
)
partimat(
  Class~Size+Course,
  data = tae.tra[[1]],
  method = "lda"
)
partimat(
  Class~Instructor+Course,
  data = tae.tra[[1]],
  method = "lda"
)



##########################################
# Quadratic Discriminant Analysis (QDA) #
##########################################
qda.fit = qda(
  Class ~ .,
  data = tae.tra[[1]]
)

qda.pred = predict(qda.fit, tae.tst.dt[[1]])

table(qda.pred$class, tae.tst.cl[[1]])

# Do it for every dataset in k-fold CV
acc = rep(0, 10)
for (i in 1:10) {
  qda.fit = qda(
```

```
    Class ~ .,
    data = tae.tra[[i]]
  )
  qda.pred = predict(qda.fit, tae.tst.dt[[i]])

  acc[i] = accuracy(
    table(qda.pred$class, tae.tst.cl[[i]])
  )
}
Acc.QDA = data.frame(fold = 1:10, Acc = acc)

mean(Acc.QDA$Acc)

# QDA performs poorly if every variable is considered.


# Fill in the table in the document
Acc.qda = matrix(ncol = 10, nrow = 10)
rownames(Acc.qda) = paste("K", 1:10, sep = "")
colnames(Acc.qda) =
  c(paste("Acc", 1:9, sep = ""), "Acc_mean")
Kappa.qda = matrix(ncol = 10, nrow = 10)
rownames(Kappa.qda) = paste("K", 1:10, sep = "")
colnames(Kappa.qda) =
  c(paste("Kappa", 1:9, sep = ""), "Kappa_mean")
for (m in 1:9) {
  myAcc   = rep(0, 10)
  myKappa = rep(0, 10)
  for (i in 1:10) {
    qda.fit = qda(
      Class ~ .,
      data = tae.tra[[i]]
    )
    qda.pred = predict(qda.fit, tae.tst.dt[[i]])

    cm = confusionMatrix(
      table(qda.pred$class, tae.tst.cl[[i]])
    )
    myAcc[i]   = cm$overall[1]
    myKappa[i] = cm$overall[2]
  }
  Acc.qda[,m]   = myAcc
  Kappa.qda[,m] = myKappa
}
Acc.qda[,10] = rowMeans(Acc.qda[,1:9])
Kappa.qda[,10] = rowMeans(Kappa.qda[,1:9])


## Some plots
partimat(
  Class~Size+Instructor,
  data = tae.tra[[1]],
  method = "qda"
)
partimat(
  Class~Size+Course,
  data = tae.tra[[1]],
  method = "qda"
)
partimat(
  Class~Instructor+Course,
```

```
  data = tae.tra[[1]],
  method = "qda"
)



#########################
# Algorithm comparison #
#########################

# Acc and Kappa for every k-fold
Acc = cbind(Acc.knn[,10], Acc.lda[,10], Acc.qda[,10])
Acc = rbind(Acc, mean = colMeans(Acc))
colnames(Acc) = c("kNN", "LDA", "QDA")
Kappa = cbind(Kappa.knn[,10], Kappa.lda[,10], Kappa.qda[,10])
Kappa = rbind(Kappa, mean = colMeans(Kappa))
colnames(Kappa) = c("kNN", "LDA", "QDA")

# Check whether there is a method outperforming the others
friedman.test(as.matrix(Acc))

## Low p-value clearly shows that there's a method
## outperforming the others

groups = rep(1:ncol(Acc), each = nrow(Acc))
pairwise.wilcox.test(
  Acc,
  groups,
  p.adjust.method = "holm",
  paired = TRUE
)

# There are significant differences amongst the three methods,
# a plot will clarify it

myAcc = as.data.frame(cbind(Acc, k.fold = 1:11))[1:10,]
myData = melt.data.frame(
  data = myAcc,
  id.vars = c("k.fold"),
  variable_name = "Method"
)
names(myData)[3] = "Acc"
ggplot(myData, aes(x = k.fold, y = Acc, fill = Method)) +
  geom_col(position = "dodge")

myData = data.frame(Measure = c("Acc.mean", "Kappa.mean"))
myData = cbind(myData, rbind(t(Acc[11,]), t(Kappa[11,])))
myData = melt.data.frame(
  data = as.data.frame(myData),
  id.vars = "Measure",
  variable_name = "Method"
)
ggplot(myData, aes(x = Method, y = value, fill = Method)) +
  geom_col(position = "dodge") + ylab("") +
  facet_wrap(~Measure, nrow = 1)
```

End Of File 'classification.R'