UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Máster Universitario en Ingeniería del Software, Métodos Formales y Sistemas de Información

---

INCIDENCIA DEL RUIDO EN LOS DATOS DE TEST SOBRE LA PRECISIÓN DE MODELOS DE CLASIFICACIÓN Y REGRESIÓN

Daniel Borao Bermúdez

Dirigida por:

César Ferri Ramírez

Valencia, Septiembre de 2013

Resumen

Las fuentes de datos hoy en día son heterogéneas y de un tamaño enorme, gracias al adelanto tecnológico experimentado, se nos ha abierto la posibilidad de acceder a tal cantidad de información, pero el problema ha pasado a ser ahora el de manejar correctamente dicho exceso de información. Un aspecto a tener presente es que estos datos suelen venir acompañados de ruido y valores incompletos o inconsistentes, por lo que una tarea fundamental antes de trabajar con ellos, es minimizar estos errores o falta de precisión al máximo, siendo conscientes que es imposible asegurar que han desaparecido por completo y en un problema real siempre estarán presentes.

La minería de datos puede entenderse como el "*Proceso de extracción de información desconocida con anterioridad, válida y potencialmente útil de grandes bases de datos, para usarla con posterioridad para tomar decisiones importantes de negocio*", para ello es muy importante que su precisión sea la mas elevada posible y el ruido puede acabar siendo un serio impedimento, de aquí la importancia de ser capaces de conocer en detalle y prever su comportamiento.

Este trabajo pretende aproximar estos dos conceptos, estudiando como se comportan los distintos modelos de predicción ante la presencia de ruido, para ello se ha realizado una serie de experimentos, donde se ha introducido ruido artificialmente en los datos de test de una serie de datasets emulando situaciones posibles y se han analizado sus resultados, obteniendo una visión de cual nos ofrece una mayor robustez o cual es mas sensible frente a la presencia de este incomodo pero inseparable elemento de las fuentes de información actuales, lo que puede ser importante a la hora de tomar decisiones en la resolución de un problema real.

Keywords: ruido, robustez, regresión, clasificación, precisión, error relativo, error cuadrático, minería de datos.

Agradecimientos

Quería agradecer en primer lugar a toda mi familia y amigos, pos su comprensión cuando no les he podido dedicar todo el tiempo necesario durante la realización de este máster. Y en especial me gustaría nombrar a dos personas:

- A César Ferri con el que ha sido un placer trabajar y del que he aprendido mucho.
- A Bea que es la que realmente me lió para embarcarme a realizar este máster.

Por último agradecer a mis compañeros de trabajo, que siempre me han facilitado el adaptar el horario a mis estudios, pese a que en ocasiones ha podido resultar problemático.

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

Los últimos avances tecnológicos han hecho que las capacidades para generar y almacenar datos se incrementen día a día, hasta llegar al punto de sobrepasar con creces las capacidades humanas para su análisis, hemos pasado a tener un exceso de información del que tenemos que ser capaces de sacar partido. Las técnicas de minería de datos persiguen como objetivo esencial el descubrimiento automático del conocimiento contenido en la información almacenada, descubriendo patrones, perfiles, tendencias y otras relaciones presentes en la información, pero ocultas si no se trata adecuadamente. El ruido, que siempre esta presente en datos reales y se encarga de enmascarar todavía mas el conocimiento subyacente en la información, complicando su tratamiento. Es conveniente entender como se comportan los distintos modelos de clasificación y regresión ante su presencia y este es el objetivo del trabajo, para ello se ha introducido ruido artificialmente en una serie de datasets de testeo y se han analizado los resultados al observar los cambios producidos en su precisión, para comprobar cual nos ofrece una mayor robustez frente a la presencia de ruido.

1.1. Minería de datos

Cada día generamos una gran cantidad de información, algunas veces conscientes de que lo hacemos y otras veces inconscientes de ello porque lo desconocemos. Nos damos cuenta de que generamos información cuando

entramos en un servidor para ver nuestro correo, cuando pagamos con una tarjeta de crédito o cuando reservamos un billete de avión. Otras veces no nos damos cuenta de que generamos información, como cuando conducimos por una vía donde están contabilizando el número de automóviles que pasan por minuto, cuando se sigue nuestra navegación por Internet o cuando nos sacan una fotografía del rostro al haber pasado cerca de una oficina gubernamental.

¿Con qué finalidad queremos generar información? Son muchos los motivos que nos llevan a generar información, ya que nos pueden ayudar a controlar, optimizar, administrar, examinar, investigar, planificar, predecir, someter, negociar o tomar decisiones de cualquier ámbito según el dominio en que nos desarrollemos. La información por sí misma está considerada un bien patrimonial. De esta forma, si una empresa tiene una pérdida total o parcial de información provoca bastantes perjuicios. Es evidente que la información debe ser protegida, pero también explotada [6].

¿Qué nos ha permitido poder generar tanta información? En los últimos años, debido al desarrollo tecnológico a niveles exponenciales tanto en el área de cómputo como en la de transmisión de datos, ha sido posible que se gestionen de una mejor manera el manejo y almacenamiento de la información. Existen cuatro factores importantes que nos han llevado a este suceso:

- El abaratamiento de los sistemas de almacenamiento tanto temporal como permanente.
- El incremento de las velocidades de cómputo en los procesadores.
- Las mejoras en la confiabilidad y aumento de la velocidad en la transmisión de datos.
- El desarrollo de sistemas administradores de bases de datos más potentes.

La cantidad de información que nos llega cada día es tan inmensa que es imposible de asimilar. Basta con ir a cualquier buscador y ver la cantidad de sitios que nos facilitan información sobre prácticamente cualquier tema. Por lo tanto, existe una clara necesidad de disponer de tecnologías que nos ayuden en nuestros procesos de búsqueda y, aún más, de tecnologías que nos ayuden a comprender su contenido.

La minería de datos surge como una tecnología que intenta ayudar a comprender el contenido de una fuente de datos, trabajando en un nivel superior buscando patrones, comportamientos, agrupaciones, secuencias, tendencias o asociaciones que puedan generar algún modelo que nos permita comprender mejor el dominio para ayudar en una posible toma de decisión.

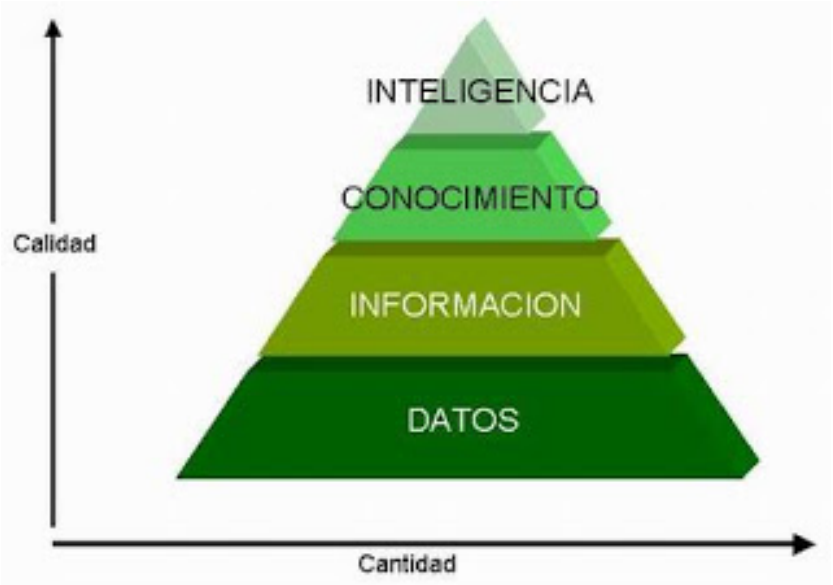

Figura 1.1: Pirámide del conocimiento. [2]

Como vemos en la imagen se puede formar una pirámide con estos cuatro conceptos: datos, información, conocimiento e inteligencia. Según escalamos por la pirámide aumentamos su valor y disminuimos su cantidad. Es costoso pero necesario pasar por todos los pasos, partiendo de un cúmulo de datos imposibles de manejar por una persona y que por si solos nos dicen muy poco, podemos a llegar a obtener el conocimiento e inteligencia necesarios sobre un área, que nos permita tomar decisiones acertadas o marcar diferencias sobre la competencia.

- Datos: Valores recolectados del mundo real, estos no tienen contexto ni relación entre si. Ejemplo: 8, Matemáticas, Pedro.
- Información: Cuando los datos son sometidos a un proceso el cual los relaciona entre si. Ejemplo. Pedro aprobó matemáticas con un 8.

- Conocimiento: Se genera cuando la información es sometida al uso y hay experiencia gracias a la vivencia. Ejemplo. Pedro es buen estudiante porque saco un 8 en matemáticas, pero debería estudiar un poco mas para acercarse al 10.
- Inteligencia: Nace gracias a que tenemos conocimiento sobre el tema y podemos predecir o imaginarnos con un grado aceptable de certeza acerca de acontecimientos futuros, esta relacionada directamente con la capacidad de aprender. Ejemplo: Como Pedro estudia matemáticas los sábados y ha sacado un 8, yo voy a estudiar también los Domingos para sacar un 10.

1.2. Concepto de ruido

En cualquier tipo de comunicación, el ruido es algo que hay que evitar, ya que ensucia el mensaje que se está transmitiendo.

Cuando nos referimos a un modelo de predicción, podemos asumir que las variables independientes transmiten información sobre la variable dependiente. Dicho de otro modo, existen variables que comunican información que nos será útil para predecir la variable de interés.

Se suele decir que los datos contienen ruido y esto interfiere con la creación de un buen modelo. Pero ¿qué es el ruido?. [5]

Podemos definir el ruido como una señal aleatoria que se superpone a la señal original y confunde al destinatario. Veamos un ejemplo viendo primero una señal sin ruido y luego con ruido superpuesto.

Supongamos que tenemos la siguiente relación entre dos variables

$y = x^2$–x+2

Una tabla de datos que contenga esta relación tendrá dos variables X e Y. La variable X será la variable independiente e Y la dependiente. Podemos asumir que X transmite información acerca de Y y un buen modelo será capaz de usar la información de X para estimar Y.

Si graficamos esta relación veremos que para cada valor de X existe solamente un valor de Y posible 1.2.

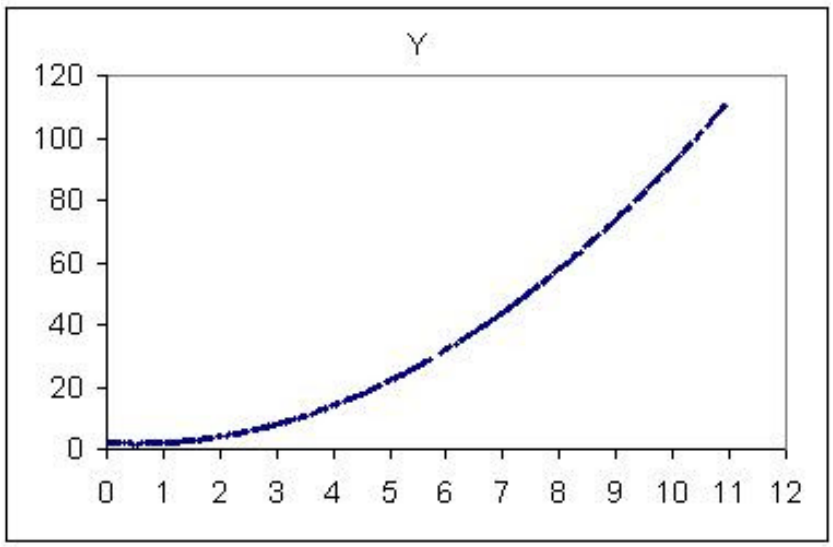

Figura 1.2: Gráfica función $y = x^2$–x+2.

Los datos podrán contener varios valores de X iguales, por ejemplo, varias filas en donde X es 5, pero en cada una de estas filas, la variable Y tendrá el mismo valor: 22 (que es el resultado de la ecuación dada más arriba).

Supongamos ahora que los datos tienen la siguiente particularidad: para un mismo valor de X pueden existir varios valores de Y. O sea, ahora si existen varias filas en donde X es 5, no necesariamente el valor de Y será en todas ellas 22. Podría ser que en algunas sea 20, en otras 22 y en otras 30.

El gráfico de estos nuevos datos se verá así 1.3:

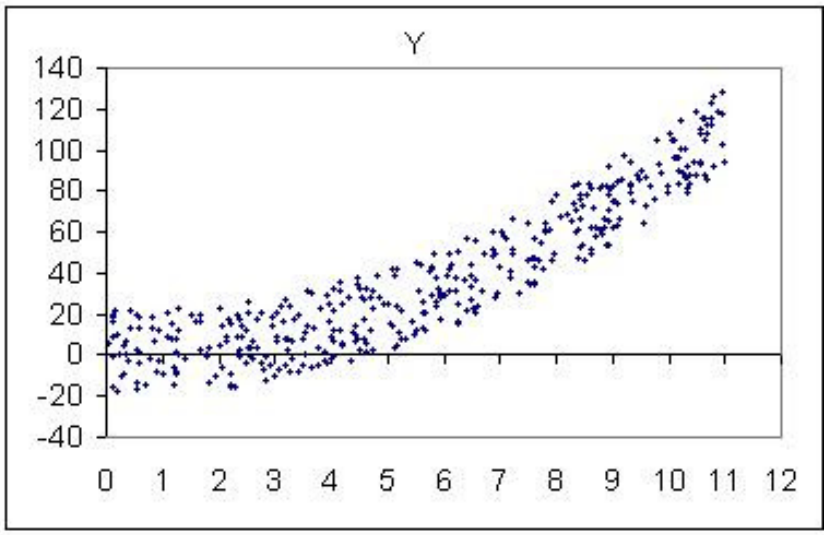

Figura 1.3: Gráfica función $y = x^2$–x+2*conruido*.

Estos nuevos datos contienen ruido porque para una misma señal (en nuestro ejemplo la señal es el valor de X) existen distintos valores que puede tomar la variable a predecir.

Un modelo construido con estos datos nunca será lo suficientemente preciso debido a que parte de la información contiene ruido. Es importante notar que no importa el tipo de herramienta usada para construir el modelo, si los datos contienen ruido, el modelo no será perfecto, y el grado de precisión dependerá justamente del nivel de ruido.

1.3. Conceptos previos

La fase de minería de datos es la fase central del proceso de descubrimiento en la que se aplican los algoritmos de búsqueda de conocimiento a los datos previamente preprocesados. En realidad este paso es inseparable del siguiente en la cadena que es el análisis de resultados. De hecho a menudo el análisis de los resultados obtenidos provoca que se retroceda de nuevo a la fase de preparación de cara a obtención de más datos o más atributos.

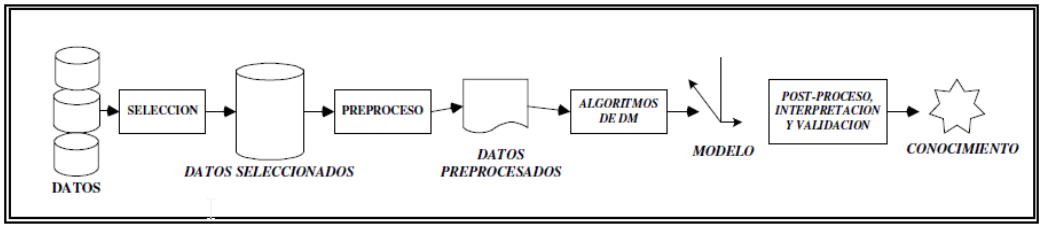

Figura 1.4: Descubrimiento de Conocimiento a partir de Bases de Datos (KDD)

Si bien es difícil establecer una clasificación de los posibles tipos de problemas a resolver que nos podemos encontrar en minería de datos, es aún más difícil encontrar un procedimiento que establezca el mejor algoritmo a aplicar para cada tipo de problema. Básicamente consideraremos estos dos tipos de problemas:

- Problemas descriptivos. Entendemos en este contexto como un problema descriptivo aquellos cuya meta es simplemente encontrar una descripción de los datos de estudio. Pertenecen a este tipo de problemas el ejemplo de conocer cuales son los clientes de una organización (características de los mismos), o el encontrar los productos que frecuentemente se compran juntos o síntomas de enfermedades que se presentan juntos. Para este tipo de problemas usaremos un aprendizaje no supervisado, es decir no contamos con conocimiento a priori, dentro de este área situaríamos el clustering o agrupamiento.

- Problemas predictivos. Por otra parte, existen consultas de data mining cuya meta es obtener un modelo que en un futuro pueda ser aplicado para predecir comportamientos. Este tipo de problemas es el que denominamos problemas predictivos o en entornos de inteligencia Artificial se denominan problemas de aprendizaje supervisado. Son entonces problemas de tipo predictivo encontrar el perfil del comprador del producto A, calcular el valor potencial de un cliente, o la probabilidad de que un cliente devuelva un préstamo. En este tipo de problemas aplicamos un aprendizaje Supervisado, que quiere decir que se cuenta con un conocimiento a priori, es decir para la tarea de clasificar un objeto dentro de una categoría o clase contamos con modelos ya clasificados (objetos agrupados que tienen características comunes).

Según el tipo de variables que el modelo debe predecir, dentro de los problemas predictivos tendremos:

- Problemas de clasificación: Hacen referencia a los problemas en los que la variable a predecir tiene un número finito de valores, esto es la variable es categórica. Un ejemplo de este tipo de problemas sería encontrar un modelo que a la vista de un histórico de clientes clasificados como "buenos" , "regulares" y "malos", establezca qué tipo de cliente es uno nuevo.
- Problemas de regresión. Se refieren a los problemas en los que la variable a predecir es numérica. En este caso como ejemplo podríamos tener el caso de encontrar un modelo que me establezca la probabilidad de que un cliente que está pidiendo un préstamo lo devolverá o no el hacer esta distinción es importante pues dependiendo del tipo de problema así será la técnica que se utilizará para solucionarlo.

Una de las fases mas costosas del proceso es la conocida como preparación de los datos, donde se integran, seleccionan, limpian y transforman los datos recogidos para obtener una vista minable o dataset con la que poder trabajar. En esta vista minable o dataset, se han debido solucionar los problemas iniciales de los datos, así como su heterogeneidad, en la medida de lo posible, proporcionándonos una única matriz o tabla con la que poder trabajar, donde cada fila corresponde a una nueva instancia o registro del tipo de datos tratado, y cada columna supondría un atributo descriptor de dicho registro, por ejemplo si la vista minable corresponde a los clientes de una empresa, cada fila seria un cliente diferente y cada uno de sus columnas aportaría información sobre el mismo, como su nombre, edad, estado civil, etc...

id_paciente	rango_edad	ge	etnia_pa	AF	AF	AF	AF	AF	AF	AF	CE	DIS	PA	ED	DA
2	Rango2	M	Mestiza	No	No	No	No	No	Si	No	Si	No	No	No	No
3	Rango2	F	Mestiza	No	No	No	No	No	No	No	Si	No	No	No	No
4	Rango1	M	Negra	No	No	No	Si	No	No	No	Si	No	No	No	No
5	Rango2	M	Negra	No	No	No	Si	No	No	No	Si	No	No	No	No
6	Rango2	M	Blanca	No	No	No	No	No	No	No	Si	No	No	No	No
7	Rango1	F	Blanca	No	No	No	Si	No	No	No	Si	No	No	No	No
8	Rango1	F	Mestiza	No	No	No	No	No	No	Si	Si	No	No	No	No
9	Rango2	M	Blanca	No	No	No	No	No	No	No	No	No	No	No	No
10	Rango2	M	Blanca	No	No	No	No	No	No	No	Si	No	No	No	No
11	Rango3	F	Negra	No	No	No	Si	No	No	No	No	No	No	No	No
12	Rango3	M	Negra	No	No	No	No	No	No	Si	Si	No	No	No	No

Figura 1.5: Ejemplo de vista minable

Una vez se tiene la vista minable, se realiza un división en la misma, en dos partes, una nos servirá como entrenamiento para los modelos y la otra como conjunto de pruebas o testeo para poder realizar predicciones con datos no usados durante el entrenamiento y comprobar su precisión.

En los problemas predictivos que vamos a tratar se busca el estimar los valores de un determinado atributo, en base a los valores del resto de atributos. Dicho atributo se denomina etiqueta o label. Sus valores, como se ha dicho, han de ser conocidos en el conjunto de entrenamiento que se utiliza para extraer el modelo. Se denomina modelo a la representación, del tipo que sea, que permite la estimación de los valores del atributo etiqueta en base al resto de valores de atributos.

Un tipo habitual de estos modelos son los modelos basados en reglas, por ejemplo:

$$Si\ (MODELO = \text{“SP02”}\ Y\ GASTO\_MENSUAL > 105) \rightarrow OFERTA = true$$

$$Si\ (EDAD < 18\ Y\ GASTO\_MENSUAL > 35) \rightarrow OFERTA = true$$

$$Si\ (MODELO = \text{“TT12”}\ Y\ CASADO = SI) \rightarrow OFERTA = false$$

Existen diferentes tipos de modelos pero todos ellos una vez extraídos son usados para predecir el valor del atributo etiqueta en los casos en los que se

desconoce. La necesidad de la existencia de la etiqueta en la fase de construcción del modelo, diferencia estas técnicas, denominadas en conjunto como de aprendizaje supervisado, de las que no usan dicho valor o de aprendizaje no supervisado.

La diferencia entre las técnicas denominadas de clasificación de las de predicción es una cuestión únicamente de la naturaleza de los atributos etiqueta. En la clasificación el atributo etiqueta es de carácter nominal (o discreto), por ejemplo, si hay abandono o no, el modelo de coche que se compra, el tipo de hipoteca a contratar, etc. Mientras que en la regresión el valor a estimar es de tipo numérico o continuo, por ejemplo, el gasto en llamadas del próximo mes, los beneficios de producción de un determinado producto, etc. Muchas de las técnicas utilizadas para ambos casos son similares y la diferencia, como se puede ver, radica en las características del valor a predecir.

Una vez se han entrenado los diferentes modelos y se ha realizado la estimación correspondiente a los datos de test, se puede calcular la precisión obtenida por los mismos, si estos datos de test disponen del valor etiqueta real, es decir evaluamos el promedio de aciertos frente a los errores en el caso de la clasificación por ser un atributo discreto y el promedio de cercanía con el valor real para los casos de regresión donde el atributo es continuo.

1.4. Plan de la tesis

Este estudio esta organizado de la siguiente forma:

- **Capítulo 1: Introducción.** Introducción a la problemática asociada a este trabajo, así como la introducción de una serie de conceptos previos útiles sobre el mismo.
- **Capítulo 2: Efecto del ruido en la precisión de los modelos.** En esta sección especificamos los experimentos a realizar, realizando una definición del estudio así como la tecnología utilizada y un estudio previo de los tipos de modelos usados.
- **Capítulo 3: Resultados.** Detallamos los resultados parciales obtenidos en los experimentos, comentándolos brevemente, y finalmente viendo sus resultados globalmente tanto para regresión como para clasificación.

- **Capítulo 4: Conclusiones.** Evaluamos los resultados obtenidos, para obtener conclusiones y encaminar posibles estudios futuros.

Finalmente, la **bibliografía** y algunos **apéndices** con la implementación utilizada para los experimentos realizados.

Capítulo 2

Efecto del ruido en la precisión de los modelos

Para conocer el comportamiento de los modelos de predicción, tanto de clasificación como de regresión, frente al ruido sobre los datos de test, hemos realizado este estudio sobre su incidencia en varios datasets. En este apartado detallamos los experimentos realizados, así como las herramientas y técnicas empleadas. Para complementar este estudio, se ha incluido en la definición de los modelos, unas gráficas para observar el comportamiento y aspecto que presentan los diferentes modelos, frente a datasets generados artificialmente, que describen unas formas muy marcadas.

2.1. Trabajos relacionados

Consultando trabajos previos sobre el tema, se ha observado que existen mas estudios sobre la incidencia de ruido, en modelos de clasificación ([9], [10]) que sobre modelos de regresión [1] y con diferentes enfoques. Los experimentos de este estudio son tanto para clasificación como para regresión.

En la mayoría de artículos consultados se introduce el ruido sobre la vista minable completa incluyendo los datos de entrenamiento y de test, con lo que se puede producir un sobreajuste (overfitting) de los modelos a estos datos con el error presente, en nuestro caso partimos de unos modelos entrenados

con datos correctos y es únicamente en los datos de test donde se introduce el ruido.

Existen multitud de formas de introducir ruido:

Hay artículos donde experimentan con 3 tipos de ruido [15]: por atributo, etiqueta [8] o en ambos. En nuestro caso hemos optado por hacerlo para todos los atributos, salvo la etiqueta.

También hay diversas formas de elegir el nivel de ruido a emplear, en algunos casos utilizan un porcentaje fijo como el 40 % [15], nosotros hemos optado por analizar el incremento progresivo del efecto del mismo desde su ausencia hasta un máximo de 50 %.

En algunos casos se centran en estudiar más exhaustivamente el comportamiento de un único modelo (Naive Bayes en [16]), o de una problemática mas concreta enfocada al problema a resolver, como la genética [13], en nuestro caso optamos por una visión mas genérica, comparando de varios modelos, intentando abarcar las principales propuestas para resolver problemas de minería de datos.

2.2. Descripción del problema

Como primer paso leemos el dataset sobre el que realizar el experimento, este será dividido en dos partes, una se utilizara para realizar el entrenamiento del modelo y otro para realizar los testeos posteriores, en una proporción del 50 % para cada uno.

El objetivo es construir una gráfica para ver cómo afecta el ruido. En esta gráfica colocaremos los clasificadores o regresores, según corresponda y representamos cómo va variando su precisión con respecto al ruido.

Para los casos de clasificación, debido a que los datos pueden no estar balanceados, es decir, existir mas registros de una clase resultante que de otra, o que a la hora de hacer una partición inicial aleatoria, su proporción sea descompensada, de manera que por ejemplo, para una clase resultante binaria Si/No, nos podemos encontrar con el caso en que se entrene el modelo únicamente con registros que sean Si, y el testeo se realice con registros

de tipo *No*. Para la separación de los datos se ha hecho uso de la función "*strata*", para la estratificación de los mismo, es decir para mantener la proporción de los mismos si existen 8 *Si* y 4 *No*, en el dataset de train, al hacerse al 50 % habrían 4 *Si* y 2 *No*, de igual forma que en el testeo, de otra forma podría llegar a darse el caso de que todas las instancias fuesen *Si* para el entrenamiento y *No* para el testeo. En el caso de la regresión se hace una partición aleatoria.

Una vez hecho esto y tomando como clase referencia para realizar la predicción, la última de las columnas, aunque esto podría ser configurable, discernimos si nos encontramos con un problema de clasificación o de regresión, según sea un atributo numérico o nominal, con lo que adecuaremos su ejecución, básicamente por los modelos a usar y el tipo de error calculado.

- Modelos de clasificación. Para medir su precisión utilizamos el porcentaje de error, es decir la proporción del total en que la predicción ha fallado, ya que al ser un modelo de clasificación, se puede indicar si se ha acertado o no. Nos dará un valor comprendido entre 0-100, donde 0 es la precisión máxima, acertando en todos los casos y con 100 se fallan todas las predicciones realizadas. La lógica nos indica que lo normal es que este valor aumente conforme se introduzca más ruido en los dataset.

- Modelos de regresión. A la hora de considerar que error utilizar en estos modelos, donde si no se acierta una predicción hay que cuantificar en cuanto se ha alejado del valor real para saber si es aceptable o no, primero se utilizó el error cuadrático medio (raíz cuadrada de la suma de los cuadrados de los errores individuales de las predicciones) pero como los valores de error se disparaban tanto en algunos casos, las gráficas dejaban de ser útiles al no poder visualizar correctamente los resultados, se optó por usar el error relativo medio o desviación relativa media (es el cociente entre el error absoluto medio y el que damos como representativo (la media aritmética)), donde el valor promedio vale 100 y donde un valor bajo indica una mejor precision y cuanto mayor sea su distancia positiva con el 100, mayor será su error.

Una vez se ha entrenado el modelo se realiza una predicción con cada modelo sin presencia de ruido y se calcula la precisión obtenida con el mismo para los datos de testeo propuestos, según el tipo de problema obtendremos

un tipo de precisión, en ambos casos y para unificar las gráficas se mostrará un valor de error ascendente según aumente el nivel de ruido.

Una vez llegados a este punto, se comienza a introducir ruido en los datos de test para ver el comportamiento con que nos encontramos, para ello partiendo de un porcentaje de error del 5 %, hasta llegar al 50 %, con incrementos sucesivos de 5 %, esto es 5 %, 10 %, 15 %, etc.

El ruido se introduce en todas las columnas, salvo lógicamente la de resultado a predecir, dentro de cada una de ellas se introducirá aleatoriamente en el porcentaje de ruido en que nos encontremos, por ejemplo si estamos en un nivel de ruido del 5 %, y hay 200 filas totales de test, se introducirá en 10 filas, conforme aumenta el nivel de ruido, aumenta el número de filas afectadas, importante indicar que la selección de filas afectadas por el ruido en una columna, no tiene porque coincidir con la siguiente columna, es decir si en la columna de edad, se introduce ruido en las 3 primeras filas y en la ultima, en la siguiente columna lo puede hacer en la primera fila y en las 3 últimas por ejemplo; aunque el número de filas totales afectadas para cada columna coincida, por ejemplo 15/30 filas, las filas concretas varían en cada columna, con lo que se torna muy difícil que hayan filas a las que el ruido no afecte en ninguno de sus atributos.

El ruido a introducir varía en función del tipo de columna tratada:

- Numérica. Se utiliza el error gaussiano con una distribución normal, teniendo en cuenta que la amplitud se calculará en función del valor máximo y mínimo que hayan en cada columna, el valor resultante sustituirá al que había previamente. Para definir el ruido tomamos los valores de un atributo, por ejemplo tenemos 150 valores entre 2 y 8, tomamos la diferencia entre 2 y 8, en este caso 6, lo dividimos entre un alfa (por ejemplo 2), dando 3 (llamamos a este valor amplitud) en este caso, y ese valor lo tomamos para estimar una distribución normal. El siguiente paso es inyectar ruido, tomamos un valor de los 150, por ejemplo nos cae uno en 4.5, y lo que debemos hacer estimar el valor con ruido construyendo una normal de media en 4.5 y desviación típica en el valor de la amplitud. Asignamos al valor con ruido, el valor de estimar un dato siguiendo esa probabilidad.
- Nominal. Se cambia el valor actual de la celda, por uno cualquiera de los valores válidos para esta columna, es decir si la columna indica el

tipo de cliente con A/B/C, si el tipo de cliente indicaba que era A, se cambia aleatoriamente por una B.

Consideramos 3 tipos de introducción de ruido:

- Filas. Se introduce ruido en todas las filas del dataset. Dentro de cada una de estas filas, en un numero de columnas determinado por el porcentaje de ruido, si es un 10 % de ruido, y hay 100 columnas, en cada fila afectara a 10 columnas, pero su distribución será aleatoria, es decir, el mismo numero total de columnas afectadas, pero no tienen porque ser las mismas en cada fila.

	Columna 1	**Columna 2**	**Columna 3**	**Columna 4**	50 % ruido
Fila 1		X	X		2/4 columnas(2, 5)
Fila 2	X			X	2/4 columnas (1, 4)
Fila 3	X	X			2/4 columnas (1, 2)
...	...	...	...	...	2/4 columnas (i, j)
Fila n		X		X	2/4 columnas (2, 4)

Tabla 2.1: Distribución ruido por filas.

- Columnas. Se introduce ruido en todas las columnas del dataset. Dentro de cada una de estas columnas, en un numero de filas determinado por el porcentaje de ruido, si es un 10 % de ruido, y hay 100 filas, en cada columna afectara a 10 filas, pero su distribución será aleatoria, es decir, el mismo numero total de filas afectadas, pero no tienen porque ser las mismas en cada columna.

	Columna 1	**Columna 2**	**Columna 3**	**...**	**Columna n**
Fila 1		X	X	...	
Fila 2	X		X	...	
Fila 3	X	X		...	X
Fila 4				...	X
50 % ruido	2/4 filas (2,3)	2/4 filas (1,3)	2/4 filas (1,2)	2/4 filas (i,j)	2/4 filas (3,4)

Tabla 2.2: Distribución ruido por columnas.

- Matricial. Tomando el dataset como una matriz y siendo el total de elementos, el numero de columnas por el número de filas, aplicaremos el porcentaje de ruidos a los elementos de la matriz independientemente de su fila y columna, es decir si hay 2 filas y 2 columnas, habrá 4

elementos o celdas, si se tiene un porcentaje de ruido del 25 % afectara a 1 elemento o intersección fila por columna.

	Columna 1	**Columna 2**	**Columna 3**	
Fila 1	X			3 celdas de 6 ((1,1),(2,2)
Fila 2	X	X		
	3 celdas de 6 ((1,1),(2,2))			50 % ruido

Tabla 2.3: Distribución ruido por matriz.

Una vez hecho esto, para cada nivel de ruido, se reitera n veces (dependiendo del número de registros 20 o 100 iteraciones) de manera que se tenga una idea clara y estable de la precisión obtenida por el modelo para este nivel de ruido.

Finalmente con los resultados obtenidos, se genera una gráfica con lattice (haciendo uso de sus funciones *xyplot y* bwplot, con la precisión obtenida frente al ruido aplicado como ejes y una agrupación por modelo), del estilo de la siguiente:

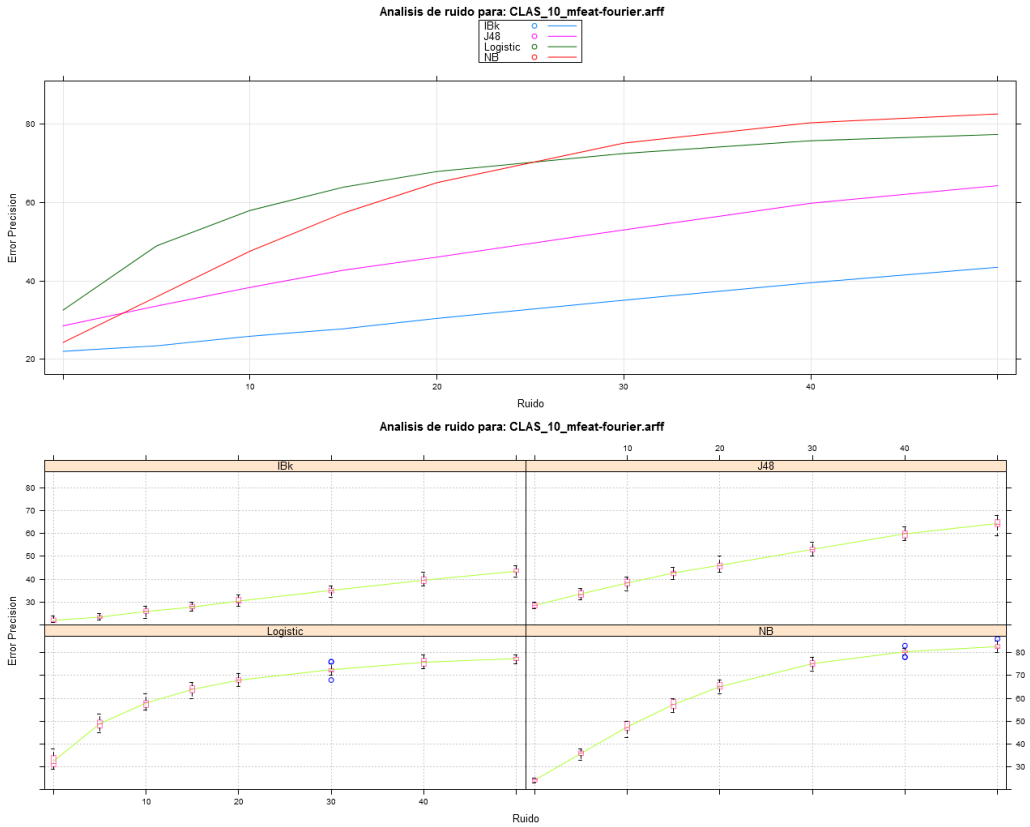

Figura 2.1: Ejemplo Gráfica

Concretamente son dos gráficas, la primera muestra la media de la precisión, conforme se aumenta el ruido, y la segunda desglosa el comportamiento de cada modelo, con diagramas de Caja-Bigotes (boxplots o box and whiskers) en los que se puede ver información mas detallada, como su dispersión, simetría, cuartiles, etc...

También se genera una tabla con los resultados, que nos será útil al final para poder realizar un test estadístico final que nos ayude a sacar conclusiones.

En el caso de la regresión, dependiendo de la partición inicial las gráficas pueden variar mucho, así que se hace el mismo experimento n veces (normalmente 20) con particiones distintas y se muestran los resultados totales medios. En el caso de la clasificación no afecta este hecho ya que los da-

tos están estratificados y el comportamiento no se distancia mucho por la partición inicial.

Resumiendo:

```
Realizar 20 particiones.

  Leer el dataset, partiendo el 50\% para train y 50\% restante test.

  Entrenar los modelos correspondientes a clasificación o regresión.

  Con el test estimar los valores de precisión inicial obtenidos.

  Bucle for n=5\% until 50\%
    Bucle foreach (tipo de ruido en (Matriz, filas, columnas))
      repeat (100)
        Inyectar ruido al n\% de valores de los atributos no etiqueta.
        Calcular precisión para los clasificadores o regresores.
      end repeat
    Construir gráfica y tablas temporales intermedias.
    end bucle
  end bucle

Construir gráfica y tablas finales con los resultados.
```

2.3. Tecnologías

Las tecnologías que se han usado son básicamente el sistema R y en especial su exportación de datos mediante gráficas o ficheros csv y modelos de weka para realizar predicciones.

2.3.1. Lenguaje R.

R es un lenguaje y entorno de programación para análisis estadístico y gráfico.

Se trata de un proyecto de software libre, resultado de la implementación GNU del premiado lenguaje S. R y S-Plus -versión comercial de S- son, probablemente, los dos lenguajes más utilizados en investigación por la comunidad estadística, siendo además muy populares en el campo de la investigación biomédica, la bioinformática y las matemáticas financieras. A esto contribuye la posibilidad de cargar diferentes bibliotecas o paquetes con finalidades específicas de cálculo o gráfico.

R se distribuye bajo la licencia GNU GPL y está disponible para los sistemas operativos Windows, Macintosh, Unix y GNU/Linux.

R proporciona un amplio abanico de herramientas estadísticas (modelos lineales y no lineales, tests estadísticos, análisis de series temporales, algoritmos de clasificación y agrupamiento, etc.) y gráficas.

Al igual que S, se trata de un lenguaje de programación, lo que permite que los usuarios lo extiendan definiendo sus propias funciones. De hecho, gran parte de las funciones de R están escritas en el mismo R, aunque para algoritmos computacionalmente exigentes es posible desarrollar bibliotecas en C, C++ o Fortran que se cargan dinámicamente. Los usuarios más avanzados pueden también manipular los objetos de R directamente desde código desarrollado en C. R también puede extenderse a través de paquetes desarrollados por su comunidad de usuarios.

R también puede usarse como herramienta de cálculo numérico, campo en el que puede ser tan eficaz como otras herramientas específicas tales como GNU Octave y su equivalente comercial, MATLAB. Se ha desarrollado una interfaz, RWeka para interactuar con Weka que permite leer y escribir ficheros en el formato arff y enriquecer R con los algoritmos de minería de datos de dicha plataforma.

- RWeka. Una interfaz para poder usar los modelos Weka (Versión 3.7.9) en R. Se comenta que es Weka en un apartado posterior.
- Lattice. Lattice es un potente y elegante sistema de visualización de datos de alto nivel, con especial énfasis en los datos multivariable, suficiente para las necesidades de gráficas mas usuales, aunque también es suficientemente flexible como para manejar los requerimientos mas específicos.
- Sampling. Funciones extra para crear y tratar muestras.

- Cairo. Este paquete proporciona un mecanismo para exportar gráficas con mayor calidad, en este caso lo utilizamos para añadir un filtro de antialiasing a las gráficas resultantes como imágenes.
- Kernlab. Proporciona entre otras cosas, modelos de maquina vector soporte basados en núcleo o kernel, tanto para clasificación como para regresión, en nuestro caso lo usaremos únicamente para los casos de regresión.
- gridExtra. Funciones de alto nivel para el uso de cuadrículas o grids.

2.3.2. Weka

Weka (Waikato Environment for Knowledge Analysis - Entorno para Análisis del Conocimiento de la Universidad de Waikato) es una plataforma de software para aprendizaje automático y minería de datos escrito en Java y desarrollado en la Universidad de Waikato. Weka es un software libre distribuido bajo licencia GNU-GPL.

El paquete Weka contiene una colección de herramientas de visualización y algoritmos para análisis de datos y modelado predictivo, unidos a una interfaz gráfica de usuario para acceder fácilmente a sus funcionalidades. La versión original de Weka fue un front-end en TCL/TK para modelar algoritmos implementados en otros lenguajes de programación, más unas utilidades para preprocesamiento de datos desarrolladas en C para hacer experimentos de aprendizaje automático. Esta versión original se diseñó inicialmente como herramienta para analizar datos procedentes del dominio de la agricultura,pero la versión más reciente basada en Java (WEKA 3), que empezó a desarrollarse en 1997, se utiliza en muchas y muy diferentes áreas, en particular con finalidades docentes y de investigación.

Los puntos fuertes de Weka son:

- Está disponible libremente bajo la licencia pública general de GNU.
- Es muy portable porque está completamente implementado en Java y puede correr en casi cualquier plataforma.
- Contiene una extensa colección de técnicas para preprocesamiento de datos y modelado.

- Es fácil de utilizar por un principiante gracias a su interfaz gráfica de usuario.

Weka soporta varias tareas estándar de minería de datos, especialmente, preprocesamiento de datos, clustering, clasificación, regresión, visualización, y selección. Todas las técnicas de Weka se fundamentan en la asunción de que los datos están disponibles en un fichero plano (flat file) o una relación, en la que cada registro de datos está descrito por un número fijo de atributos (normalmente numéricos o nominales, aunque también se soportan otros tipos). Weka también proporciona acceso a bases de datos vía SQL gracias a la conexión JDBC (Java Database Connectivity) y puede procesar el resultado devuelto por una consulta hecha a la base de datos. No puede realizar minería de datos multi-relacional, pero existen aplicaciones que pueden convertir una colección de tablas relacionadas de una base de datos en una única tabla que ya puede ser procesada con Weka.

2.4. Datasets artificiales

En la definición de los modelos se incluye un estudio comparativo con Datasets generados artificialmente.

Para el estudio de los modelos de clasificación utilizamos los datasets publicados por Tom Fawcett en su web `http://home.comcast.net/~tom.fawcett/public_html/ML-gallery/pages/index.html`. Para el estudio de los modelos de regresión, se han creado tres datasets, uno mediante la función seno, otro con la función arcotangente y otro con la función inversa de x.

La base común de este experimento es mostrar 6 gráficas, detallando aquí que representa cada una:

- Entrenamiento. En esta gráfica se muestran los puntos que sirven como entrenamiento al modelo, diferenciando con un color distinto, cada uno de los 2 valores posibles.
- Test. En esta gráfica se muestran los puntos que sirven como test al modelo, diferenciando con un color distinto, cada uno de los 2 valores posibles y mostrando su resultado real.

- Modelo Ruido 0. En esta gráfica se muestra, con colores distintos al resto de gráficas, la predicción que realiza el modelo sobre los puntos de test sin ruido, diferenciando con un color distinto, cada uno de los 2 valores posibles y mostrando su resultado predicho, que puede coincidir mas o menos con los datos reales anteriores. La precisión viene dada por un porcentaje entre paréntesis. Para clasificación, si la precision es del 100 % esta gráfica coincide con la anterior, si la precisión es del 0 % la gráfica será inversa a la anterior. En Regresión lo que se muestra es el error relativo medio, con lo que un valor optimo y completamente alejado de la media podría llegar a ser negativo, y un valor pésimo correspondería a un valor muy alto.
- Modelo Ruido 10. Muestra el efecto del ruido al 10 % sobre los datos de test. No se mostrara la predicción del modelo, únicamente la precisión obtenida entre paréntesis en el título.
- Modelo Ruido 20. Muestra el efecto del ruido al 20 % sobre los datos de test. No se mostrara la predicción del modelo, únicamente la precisión obtenida entre paréntesis en el título.
- Modelo Ruido 30. Muestra el efecto del ruido al 20 % sobre los datos de test. No se mostrara la predicción del modelo, únicamente la precisión obtenida entre paréntesis en el título.

2.5. Clasificación

Una clasificación se puede ver como el esclarecimiento de una dependencia, en la que el atributo dependiente puede tomar un valor entre varias clases, ya conocidas.

2.5.1. J48

Los algoritmos más comúnmente utilizados en problemas de clasificación son los algoritmos basados en árboles de decisión, ya que son muy fáciles de representar y comprender. Estos algoritmos generan estructuras en forma de árbol, donde cada uno de sus nodos tienen una condición, siendo los hijos de dicho nodo los diferentes valores que puede tomar dicha decisión. En las

hojas se encuentran los valores posibles del atributo etiqueta. La evaluación de estos modelos consiste en ir descendiendo por el árbol de decisión, guiado por las condiciones de los nodos hasta encontrar una hoja que determina el valor predicho por el modelo.

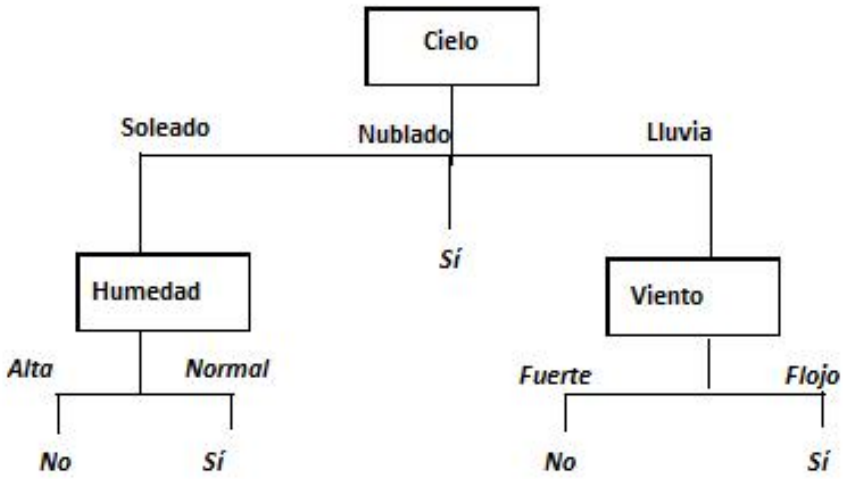

Figura 2.2: Árbol de decisión

Para los experimentos usaremos el J48, una implementación libre en Java del algoritmo C4.5 (algoritmo usado para generar un árbol de decisión, desarrollado por Ross Quinlan [11]). C4.5 es una extensión del algoritmo previo ID3 de Quinlan. Es un método "divide y vencerás", basado en criterios de partición derivados de la ganancia (GainRatio).

Al testear una distribución en forma de cuadros, se ve como se consigue una precisión perfecta, ya que es sencillo para el J48 marcar reglas que supongan dividir el espacio en dos zonas tales que uno sean de un tipo y en la otra del otro tipo.Una vez se comienza a generar ruido lógicamente disminuye la precisión y lo hace dependiendo directamente de que el valor con ruido se desplace dentro de una zona del mismo color o invada otra, con lo que se provocaría un error de predicción en este punto.

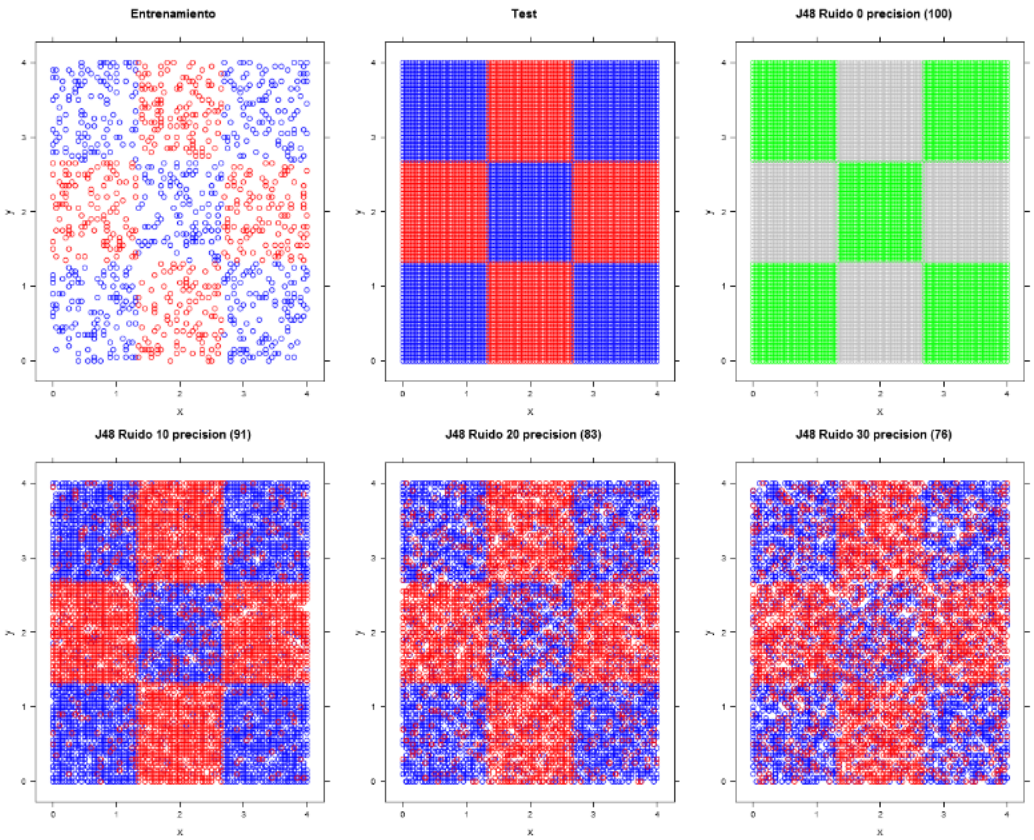

Figura 2.3: Ruido sobre clasificador binario tablero J48

Para el caso de zonas delimitadas por curvas, en lugar de rectas, el J48 obtiene lógicamente una menor precisión inicial, ya que no es el comportamiento deseable para este modelo, pero su estabilidad frente al ruido se mantiene en unos indices similares y vendrá afectado por la misma casuística.

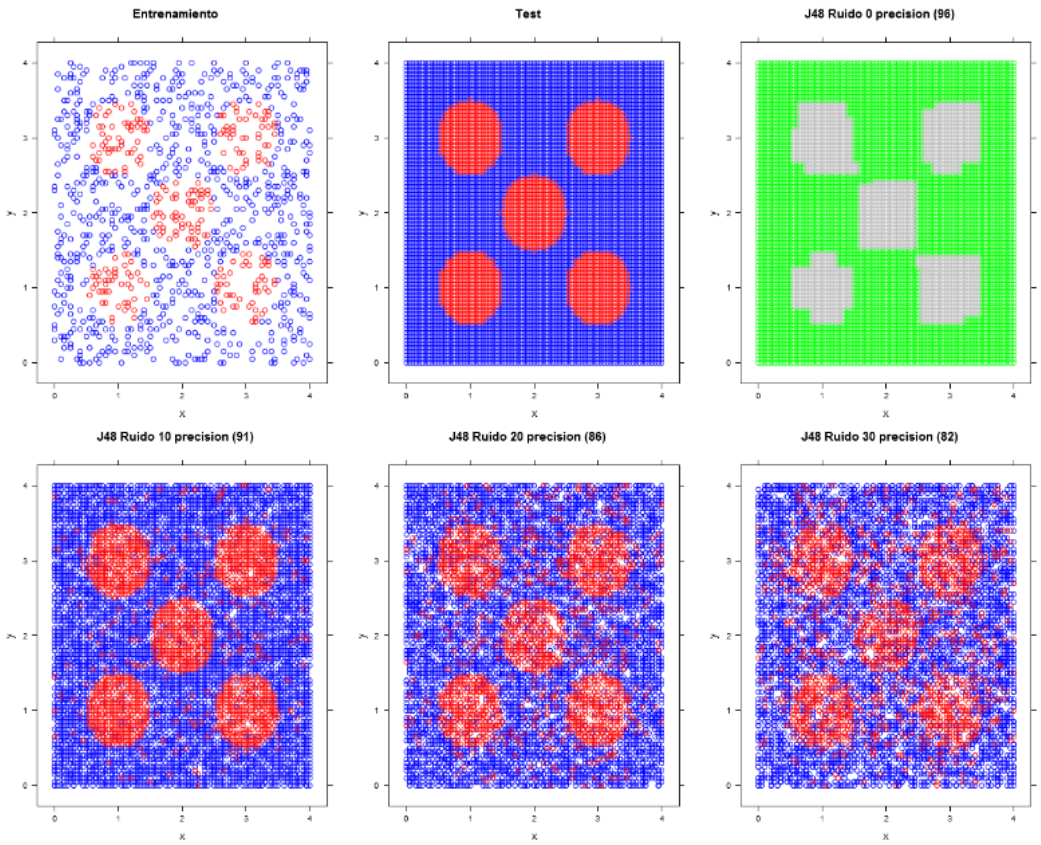

Figura 2.4: Ruido sobre clasificador binario circulos J48

2.5.2. Regresión logística

El Análisis de Regresión es una técnica estadística que tiene como objetivo establecer modelos matemáticos para representar formalmente las relaciones de dependencia existente entre un conjunto de variables estadísticas.

Tanto en el caso de dos variables (regresión simple) como en el de más de dos variables (regresión múltiple), el análisis de regresión lineal puede utilizarse para explorar y cuantificar la relación entre una variable llamada dependiente o criterio (Y) y una o más variables llamadas independientes o predictoras (X_1, X_2, ..., X_k), así como para desarrollar una ecuación lineal con fines predictivos.

Un diagrama de dispersión ofrece una idea bastante aproximada sobre el tipo de relación existente entre dos variables. Pero, además, un diagrama de dispersión también puede utilizarse como una forma de cuantificar el grado de relación lineal existente entre dos variables: basta con observar el grado en el que la nube de puntos se ajusta a una línea recta.

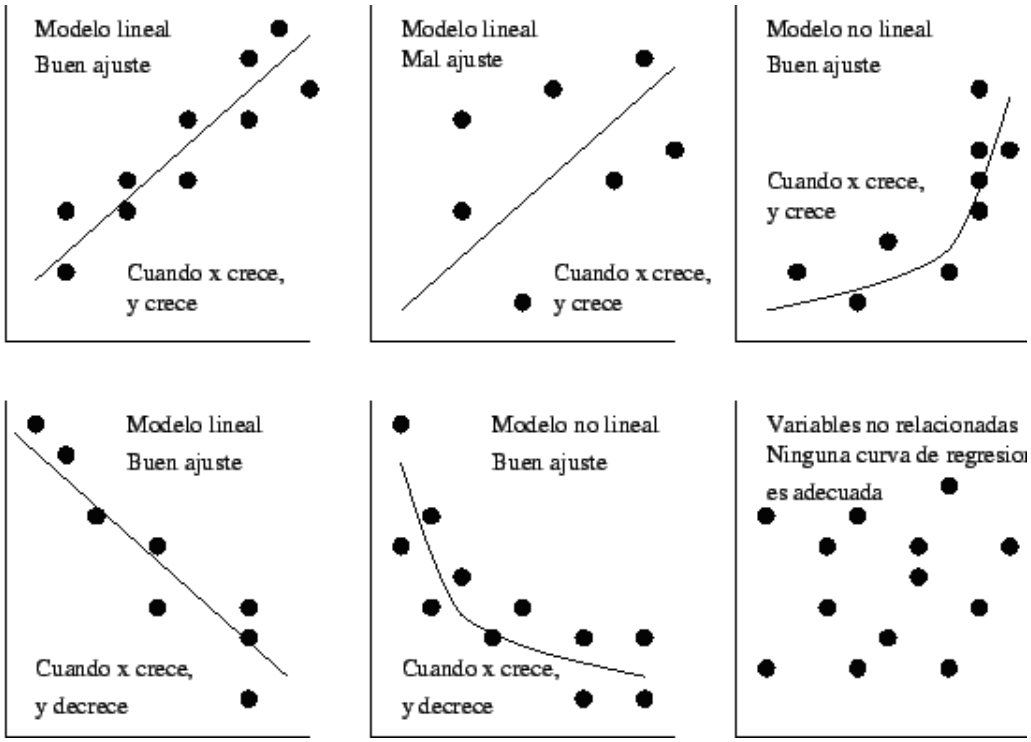

Figura 2.5: Ajuste del diagrama de dispersión

La distribución logística es una distribución de probabilidad continua cuya función de distribución es la función logística, que aparece en el contexto de la regresión logística y determinados tipos de redes neuronales. Se parece a la distribución normal en su forma, pero tiene colas más pesadas.

Al testear una distribución en forma de cuadros, la regresión logística se comporta dando siempre un mismo valor, con lo que la precisión no es muy alta, pero se mantiene invariable por el ruido, siempre que no cambie la proporción de las clases.

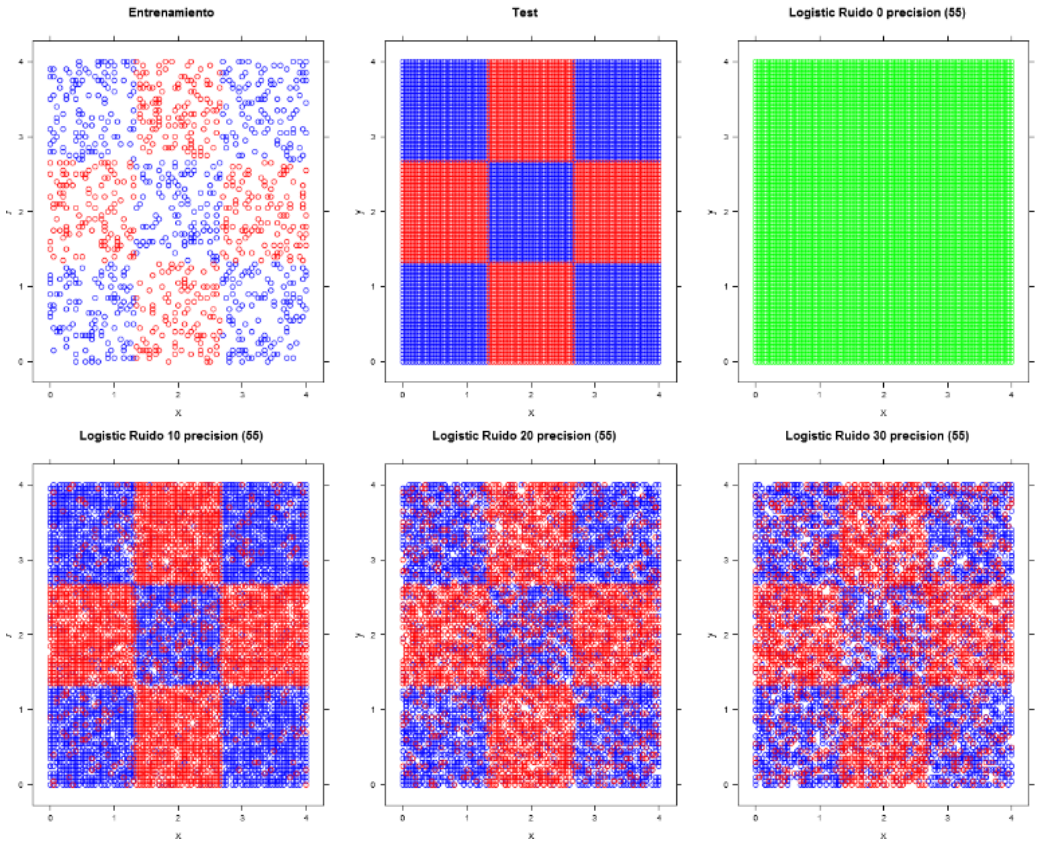

Figura 2.6: Ruido sobre clasificador binario tablero Logistic

Al tratar con formas curvas, la regresión logística se comporta de igual forma, dando siempre un mismo valor, con lo que la precisión tampoco es muy alta, y también se mantiene invariable por el ruido, en este caso particular la precisión obtenida es mayor ya que existen muchas mas instancias de un tipo que de otro, con lo que siempre predice el valor mayoritario.

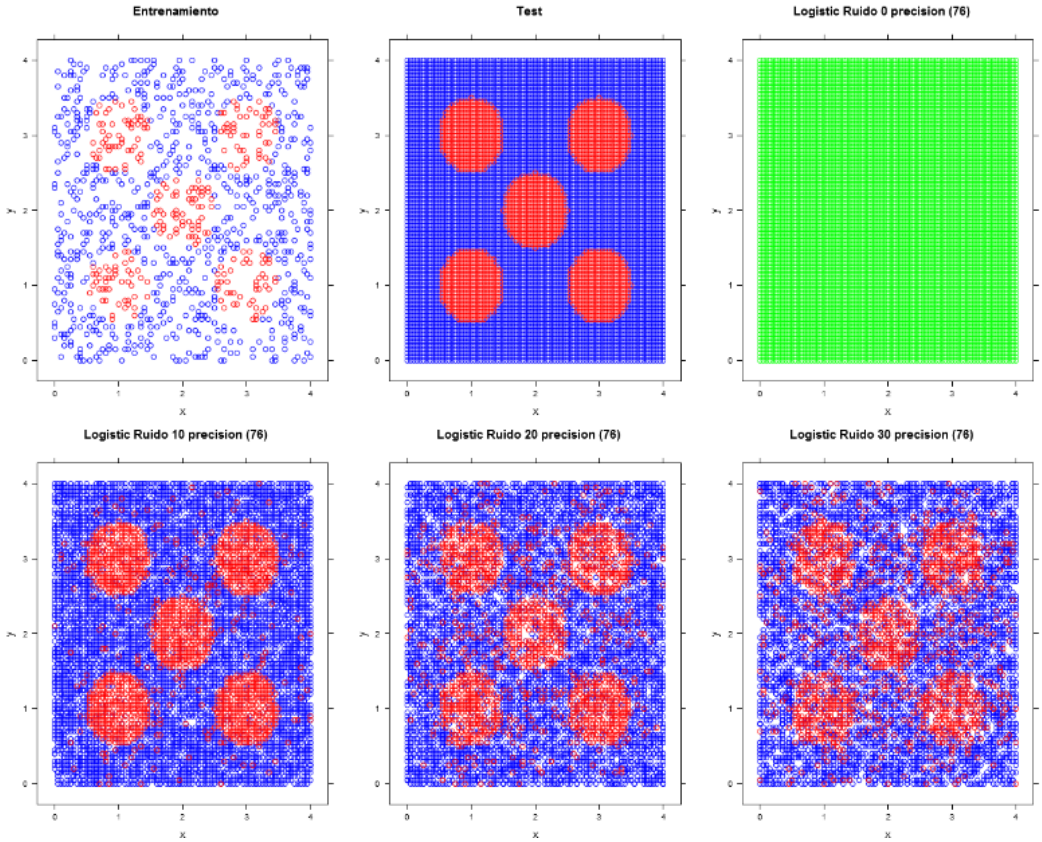

Figura 2.7: Ruido sobre clasificador binario circulos Logistic

De todas formas, para las distribuciones de estos datasets artificiales, la regresión logística no es un modelo que tenga un buen comportamiento

2.5.3. Naïve Bayes

Uno de los clasificadores más utilizados por su simplicidad y rapidez. Se trata del modelo más simple de clasificación con redes bayesianas, la estructura de la red es fija y sólo necesitamos aprender los parámetros (probabilidades). Constituye una técnica supervisada porque necesita tener ejemplos clasificados para que funcione. La principal desventaja de este mecanismo de clasificación se deriva de las dos simplificaciones que asume a la hora de calcular los modelos probabilísiticos:

- La independencia entre las variables: Rara vez es posible y en problemas reales es muy complicado que los propios datos no contengan variables redundantes o directamente dependientes de otras. En estos casos el modelo asignaría un mayor peso a las características originales que vengan representadas por más de una variable. Por ejemplo, si tenemos un conjunto de datos que son potenciales clientes y dos de las variables que conocemos de ellos son su Edad y Fecha de Nacimiento (dependiente una de la otra) el modelo Naive-Bayes extraíble otorgará el doble de peso a esta característica en relación a las demás. Esto hace que las predicciones vengan sesgadas por las variables redundantes.
- Estimación de variables continuas: El uso de distribuciones normales como suposición en el ajuste de valores continuos es un caso más de simplificación del modelo que puede provocar errores. Un ajuste más apropiado, si el número de datos lo permite, sería estudiar que distribución siguen dichos valores continuos.

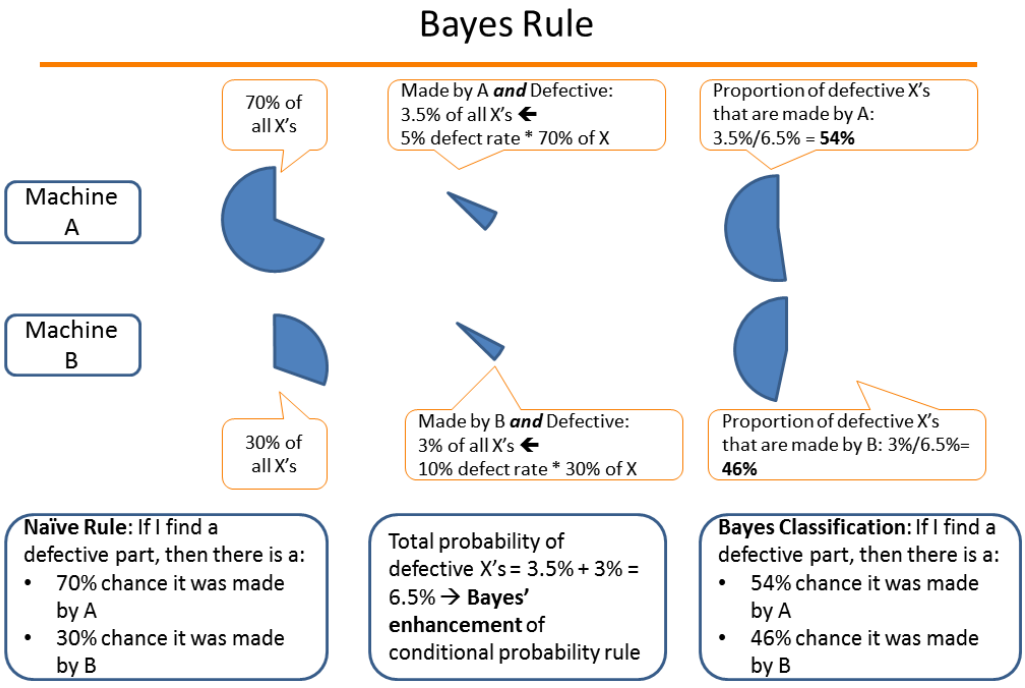

Figura 2.8: Naive Bayes sobre el dataset de Iris [3]

Al testear una distribución en forma de cuadros, se ve como se como Naive Bayes muestra un aspecto bastante diferente, con lo que la precisión es

bastante baja y esta se mantiene invariable con el ruido, ya que prácticamente es la misma que la probabilidad aleatoria.

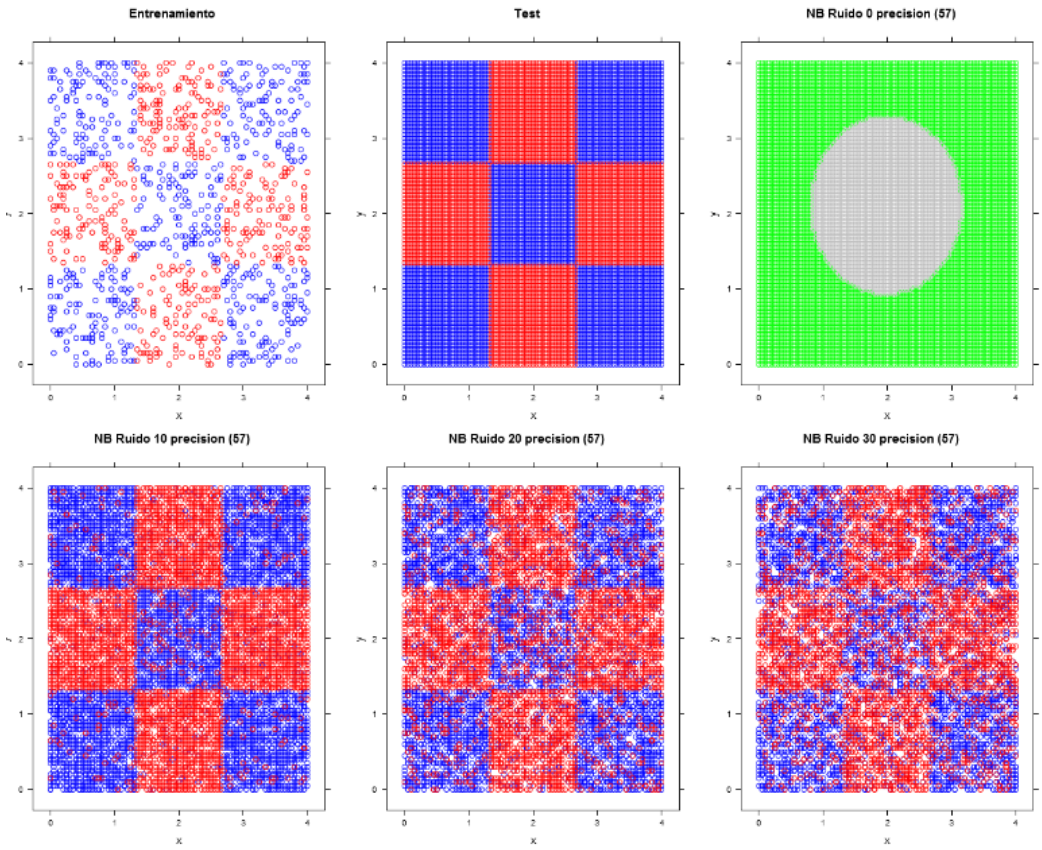

Figura 2.9: Ruido sobre clasificador binario tablero Naive Bayes

Para el caso de zonas delimitadas por curvas, en lugar de rectas, Naive Bayes se comporta prediciendo siempre el mismo valor, en este caso el valor predominante, con lo que se obtiene una precisión mayor que en el ejemplo anterior, invariante con el ruido, ya que haya el ruido que haya en los atributos siempre retornara la misma predicción, la única forma en que podría disminuir la precisión es que la clase predominante dejase de serlo, habiendo mas instancias de la otra clase resultante.

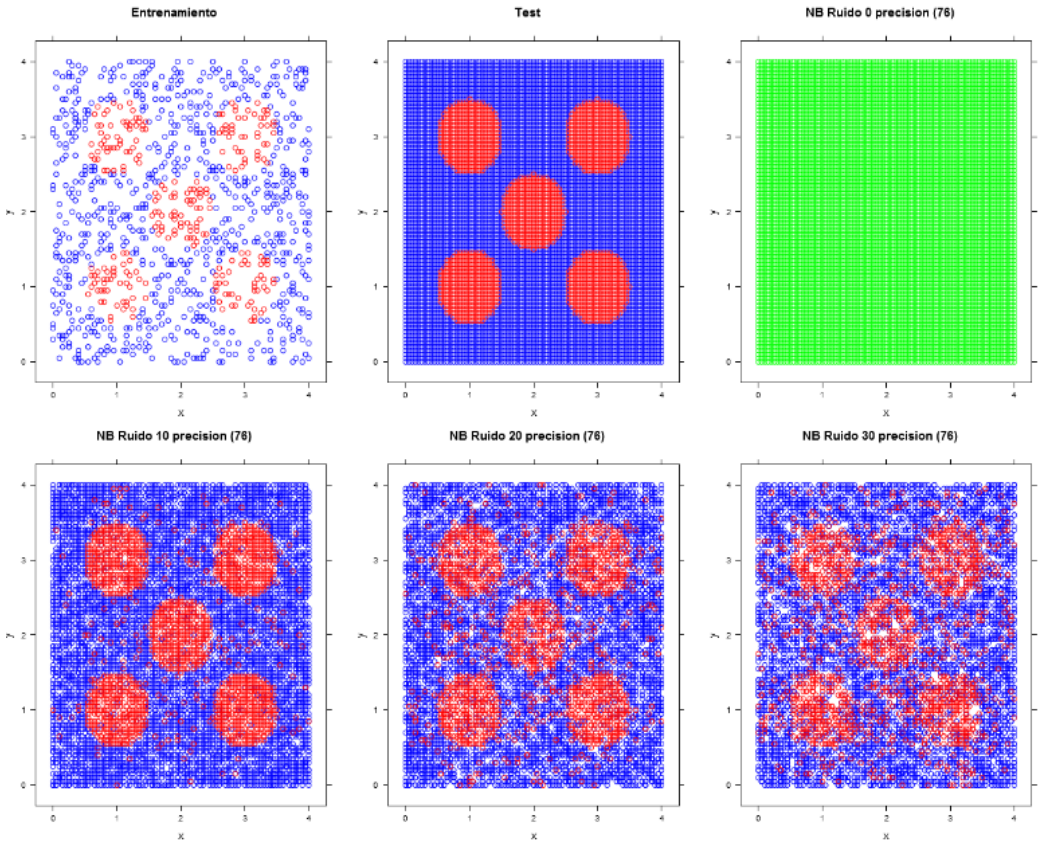

Figura 2.10: Ruido sobre clasificador binario circulos Naive Bayes

2.5.4. IBk

El método K-NN (K vecinos cercanos, Fix y Hodges, 1951), también conocido como IBk, es un método de clasificación no paramétrico supervisada. Este método permite clasificar un elemento x dentro de una de las categorías de la clase $C = \{c_1, c_2, \ldots, c_k\}$.

La idea básica sobre la que se fundamenta este paradigma es que un nuevo caso se va a clasificar en la clase más frecuente a la que pertenecen sus K vecinos más cercanos. El paradigma se fundamenta por tanto en una idea muy simple e intuitiva, lo que unido a su fácil implementación hace que sea un paradigma clasificatorio muy extendido.

En problemas prácticos donde se aplica esta regla de clasificación se acostumbra tomar un número k de vecinos impar para evitar posibles empates, aunque esta forma es cierta en problemas que poseen dos clases nada más. También, los empates pueden ser resueltos decidiendo aleatoriamente la clasificación de la muestra entre las clases empatadas o la clase donde la distancia media de sus vecinos sea inferior.

Para determinados problemas reales (es decir, con un número finito de muestras e incluso, en muchas ocasiones, un número relativamente pequeño), la aplicación de esta regla podría entenderse como una solución poco apropiada, debido a los pobres resultados que pudieran obtener, es decir, a su baja tasa de aciertos en el correspondiente proceso de clasificación. Este problema también está presente cuando el número de muestras de que se dispone puede considerarse pequeño comparado con la dimensionalidad intrínseca del espacio de representación, lo cual corresponde a una situación bastante habitual.

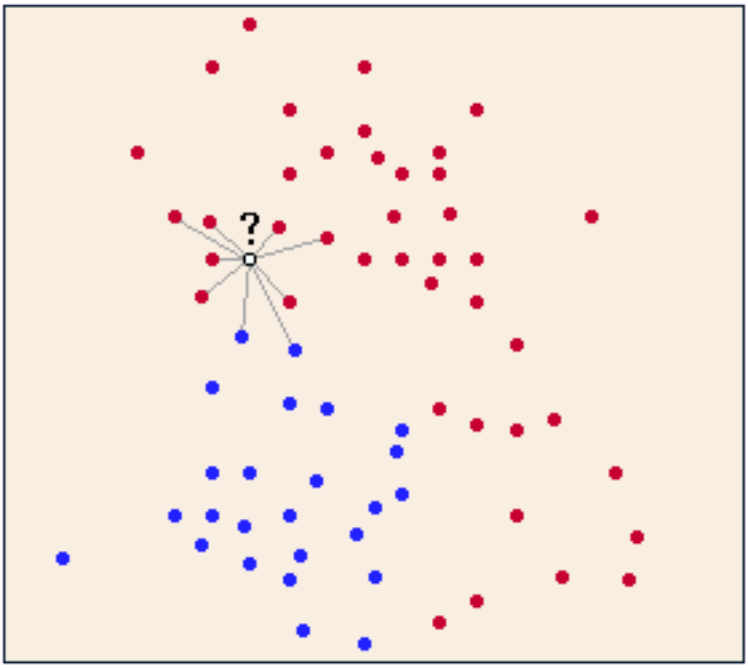

Figura 2.11: K vecinos mas próximos

Al testear una distribución en forma de cuadros, se ve como se como el aspecto de la predicción sin ruido en IBk varia ligeramente de los reales

de test, esto es debido a que se basa en los datos de entrenamiento y estos al no tener todas las instancias dejan huecos que el IBk no interpreta como lineas rectas, sino que depende cada caso particular. Se muestran las regiones basándose en el diagrama de Voronoi

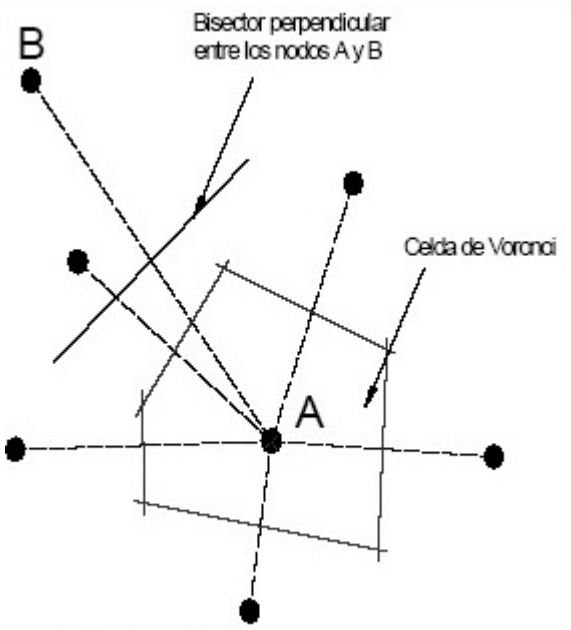

Figura 2.12: Celda de Voronoi para el nodo A

La idea del diagrama de Voronoi se basa fundamentalmente en la proximidad. Suponemos dado un conjunto finito de puntos en el plano $P = \{p_1,\ldots,p_n\}$ (con n mayor o igual que dos) y a cada p_j le asociamos aquellos puntos del plano que están más cerca o igual suya que de cualquier otro de los p_i con i distinto de j. Todo punto del plano queda así asociado a algún p_i, formándose conjuntos que recubren a éste. Existirán puntos que disten lo mismo de dos elementos de P y que formarán la frontera de cada región. Los conjuntos resultantes forman una teselación del plano, en el sentido de que son exhaustivos (todo punto del plano pertenece a alguno de ellos) y mutuamente excluyentes salvo en su frontera. Llamamos a esta teselación Diagrama de Voronoi plano (denotado $Vor(P)$). A cada una de las regiones resultantes las llamaremos regiones de Voronoi o polígonos de Voronoi (denotado $Vor(p_i)$). Los puntos del conjunto reciben el nombre de generadores del diagrama.

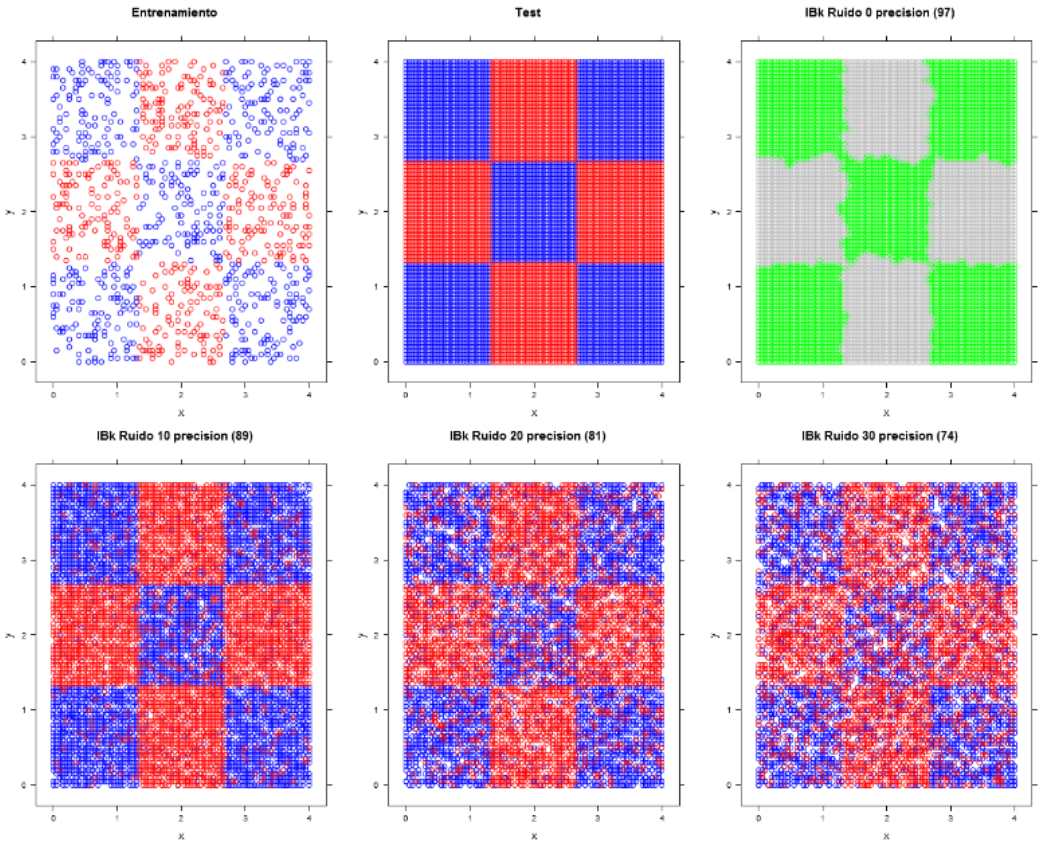

Figura 2.13: Ruido sobre clasificador binario tablero IBk

Al tratar con superficies curvas, el comportamiento es similar al de paredes planas, tanto en la predicción como en el efecto del ruido.

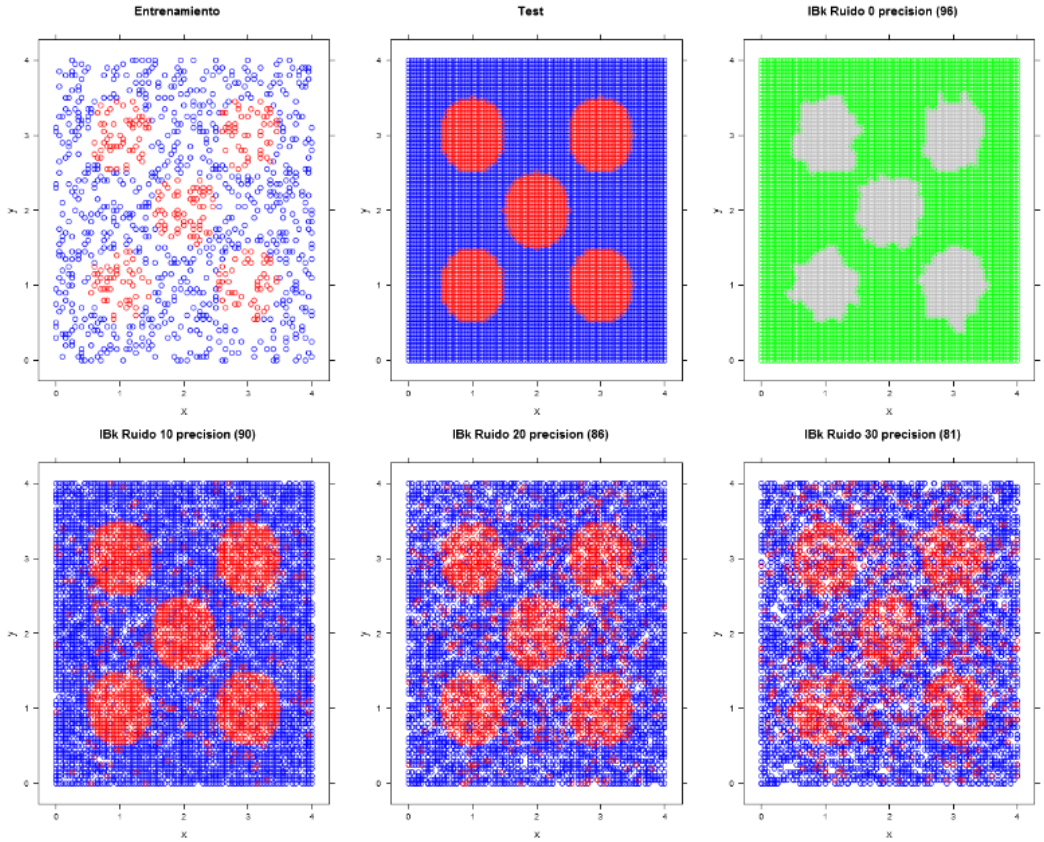

Figura 2.14: Ruido sobre clasificador binario circulos IBk

2.6. Regresión

El objetivo de la regresión es predecir los valores de una variable continua a partir de la evolución sobre otra variable continua, generalmente el tiempo.

2.6.1. M5P

Es un método de aprendizaje mediante árboles de decisión numéricos, utiliza el criterio estándar de poda M5. Construcción de árbol mediante algoritmo inductivo de árbol de decisión. Decisiones de enrutado en nodos tomadas a partir de valores de los atributos. Cada hoja tiene asociada una

clase que permite calcular el valor estimado de la instancia mediante una regresión lineal.

El algoritmo original M5 fue inventado por R. Quinlan [4] y mejorado por Yong Wang [17].

Podemos ver en qué consiste un árbol de decisión en el apartado correspondiente a la clasificación J48, sección 2.5.1.

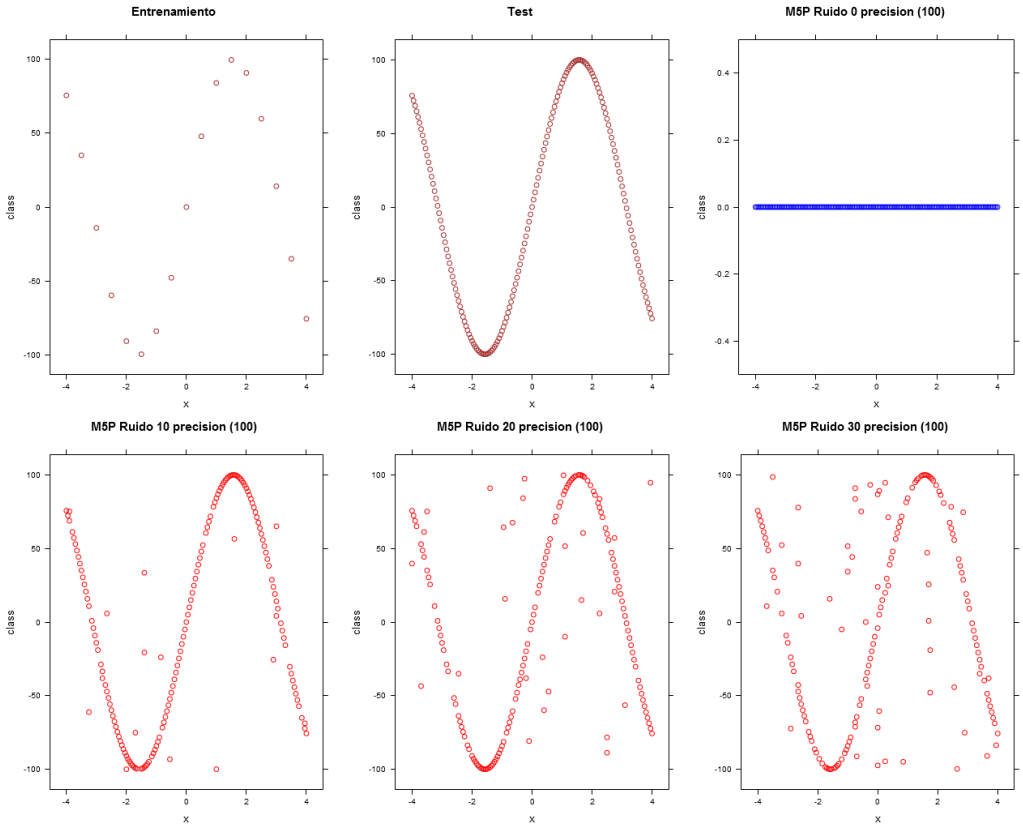

Figura 2.15: Función seno M5P

Observamos como para el caso de la función senoidal, el M5P no es capaz de predecir el comportamiento, y siempre retorna la media, con ruido o sin el, por lo que no es útil su predicción para gráficas de este estilo.

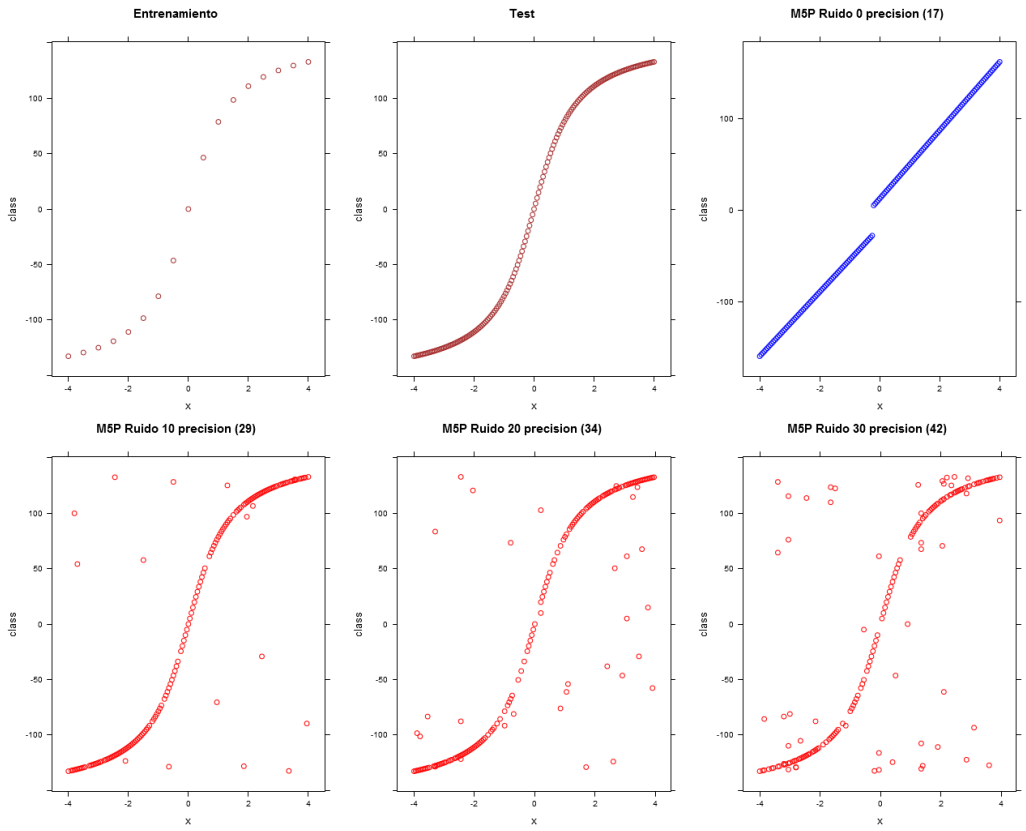

Figura 2.16: Función arcotangente M5P

En el caso de la arcotangente, M5P se comporta muchísimo mejor que frente al seno, ya que para esta función disminuye sensiblemente el error inicial, acercándose al aspecto que la gráfica debería tener, aumentando este como es lógico frente a la presencia de ruido.

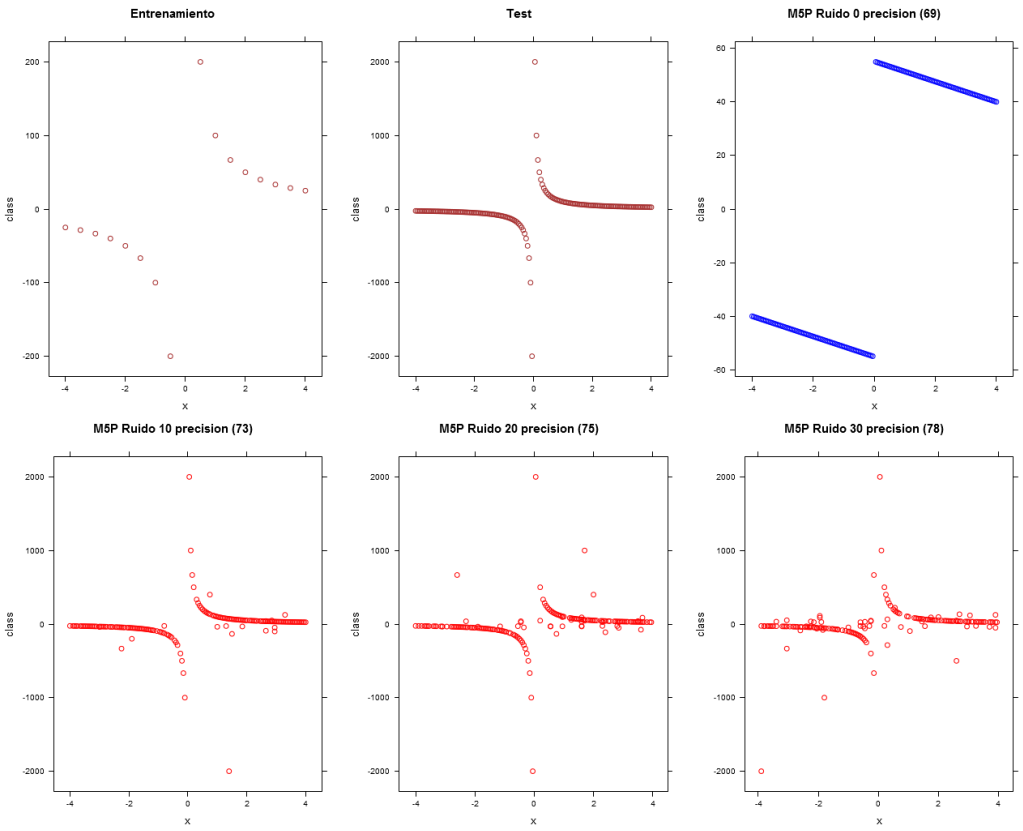

Figura 2.17: Función hipérbola equilátera M5P

Frente a la función inversa de x, el modelo M5P se comporta mejor que la media, aunque sin mejorar tanto como en el caso de la arcotangente, vemos que si diferencia dos zonas, aunque con aspecto de lineas rectas.

2.6.2. M5Rules

Genera un sistema de reglas de decisión para problemas de regresión usando "divide y vencerás". En cada iteración construye un modelo de árbol usando M5 y convierte la mejor hoja en una regla.

Este modelo tiene un comportamiento muy similar al anterior (M5P), aunque en este caso mejora ligeramente el error detectado.

Podemos ver en qué consiste un árbol de decisión en el apartado correspondiente a la clasificación J48, sección 2.5.1.

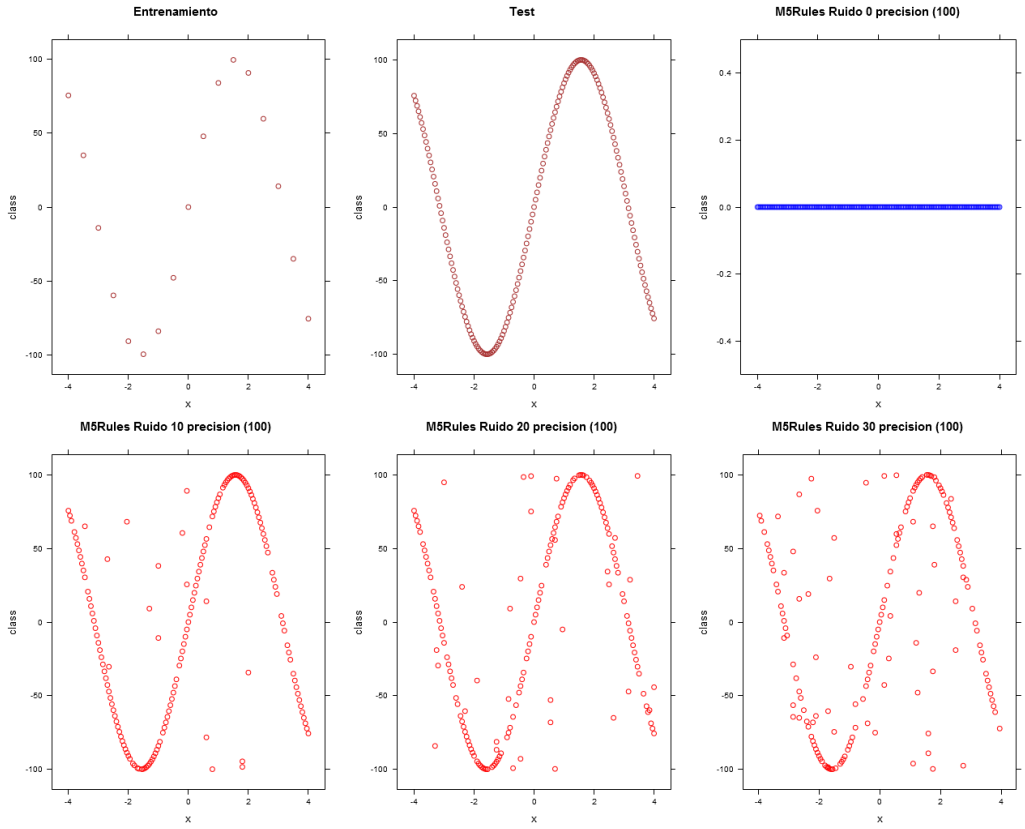

Figura 2.18: Función seno M5Rules

Como en el caso del M5P, no es capaz de seguir el comportamiento de la función, al representarse siempre con lineas rectas y adoptando en todos los casos el valor medio.

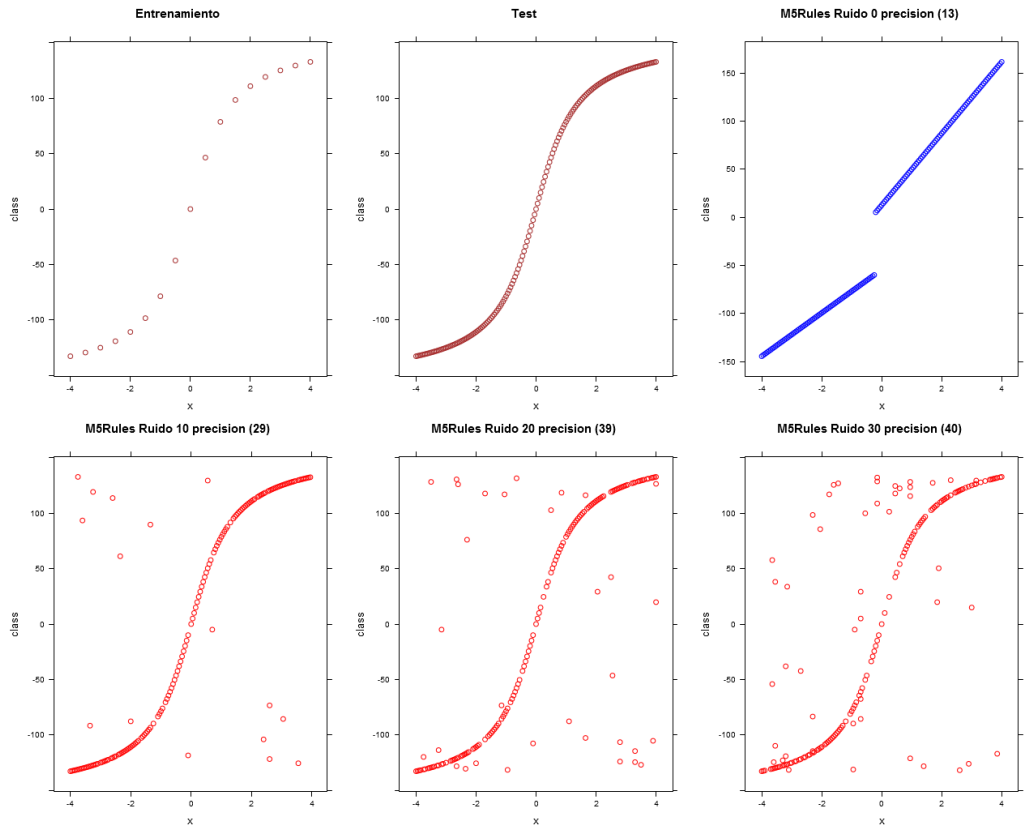

Figura 2.19: Función arcotangente M5Rules

En este caso se comporta muy bien, acercando mucho el aspecto e la gráfica y disminuyendo mucho el error inicial y ante la presencia de ruido.

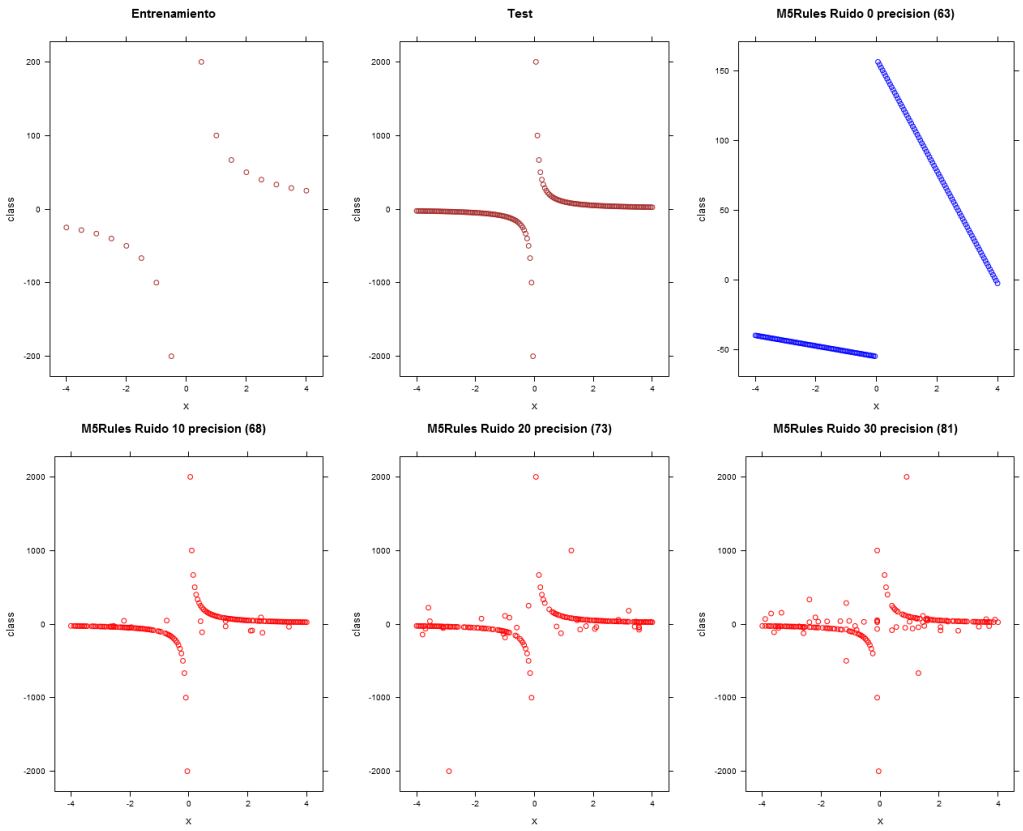

Figura 2.20: Función hipérbola equilátera M5Rules

Como en el caso anterior de M5P, mejora la precisión que la media, pero el aspecto de la gráfica sigue sin adaptarse demasiado al aspecto final.

2.6.3. Regresión lineal

El modelo de regresión lineal es el más utilizado a la hora de predecir los valores de una variable cuantitativa a partir de los valores de otra variable explicativa también cuantitativa. Una generalización de este modelo, el de regresión lineal múltiple, permite considerar más de una variable explicativa cuantitativa. Por otra parte es también posible incluir variables explicativas categóricas en un modelo de regresión lineal si se sigue una determinada estrategia en la codificación de los datos conocida como codificación ficticia.

Podemos ver algunos conceptos mas de regresión en el apartado correspondiente a la clasificación, aunque este caso corresponda a la regresión logística sección 2.5.2.

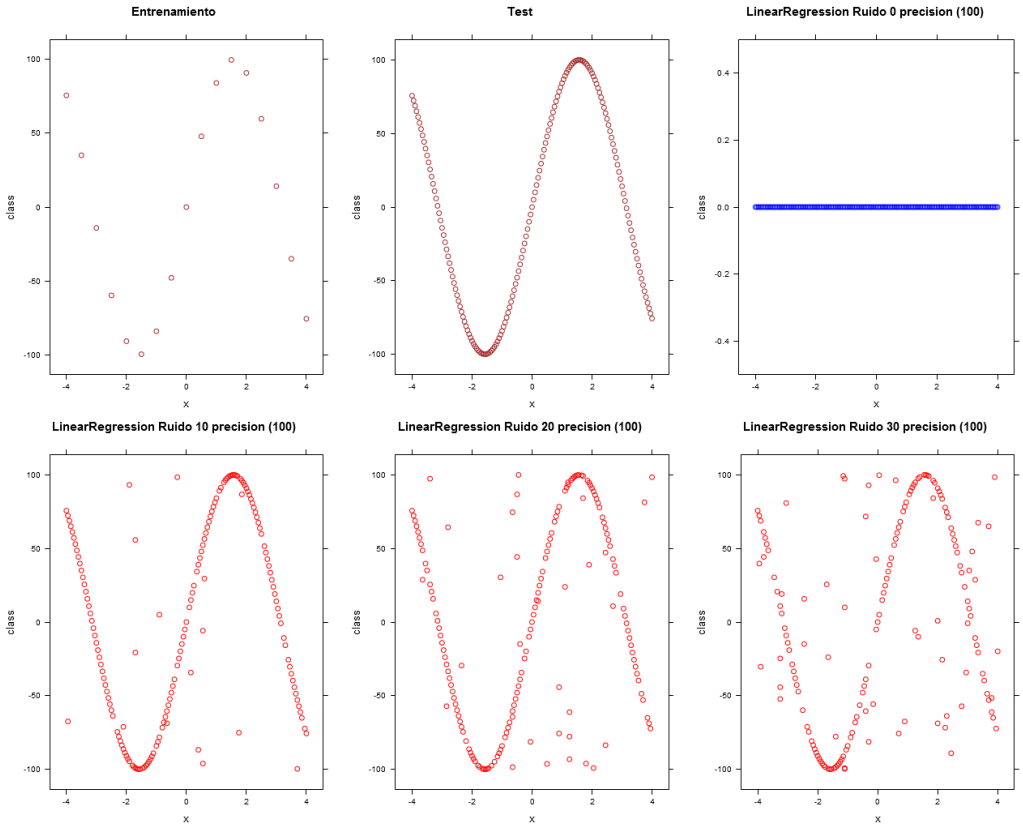

Figura 2.21: Función seno regresión lineal

La regresión lineal no se adapta a la forma senoidal de la función, obteniendo en todos los casos el valor medio como predicción.

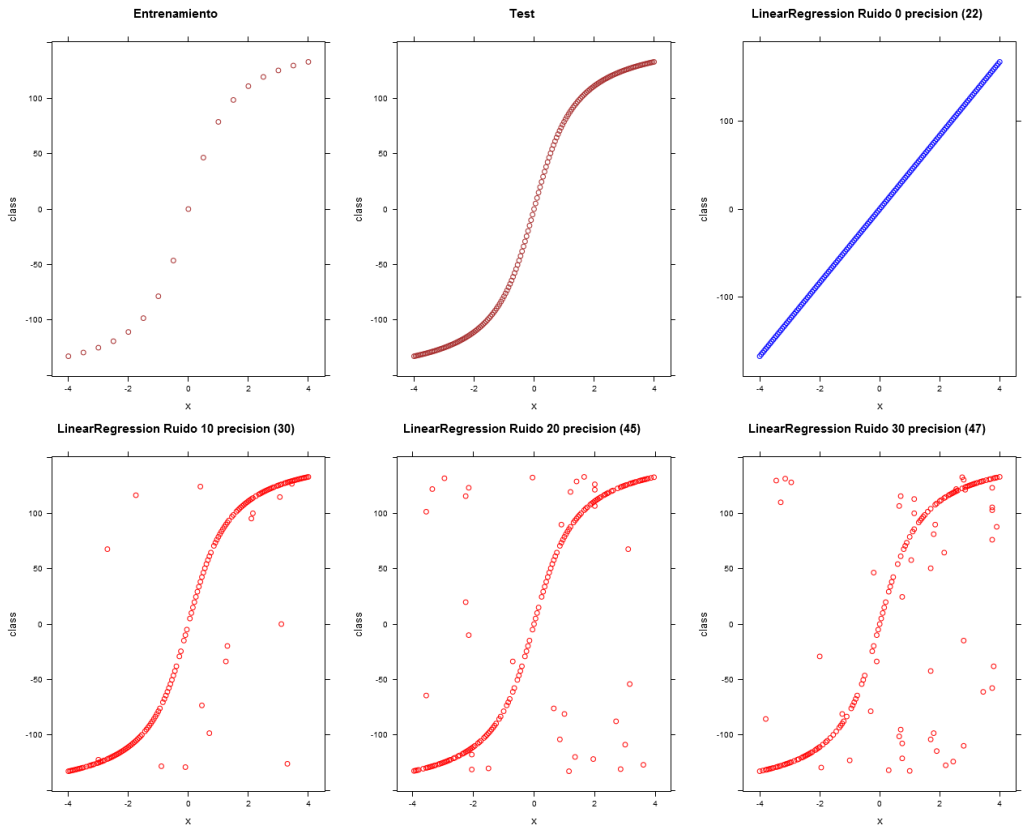

Figura 2.22: Función arcotangente regresión lineal

Para este caso y al formar siempre una linea recta, se obtiene errores mucho menores, con lo que la precisión si aumenta notablemente con respecto a la media.

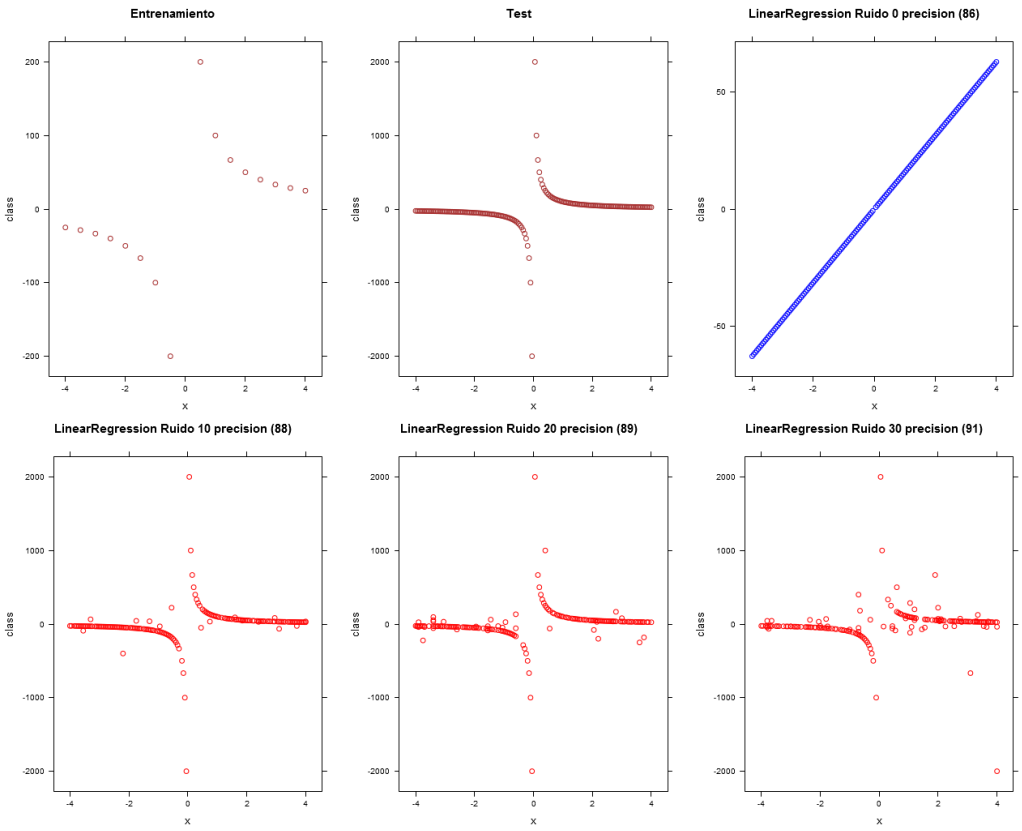

Figura 2.23: Función hipérbola equilátera regresión lineal

Para el caso de la función inversa de x, el modelo trata de acercarse, pero el resultado no es comparable a la arcotangente y obtiene unos valores de error inferiores a la media, pero por muy poco.

2.6.4. IBk

Este algoritmo es común para clasificación y regresión y ya ha sido comentado en el apartado previo de la sección 2.5.

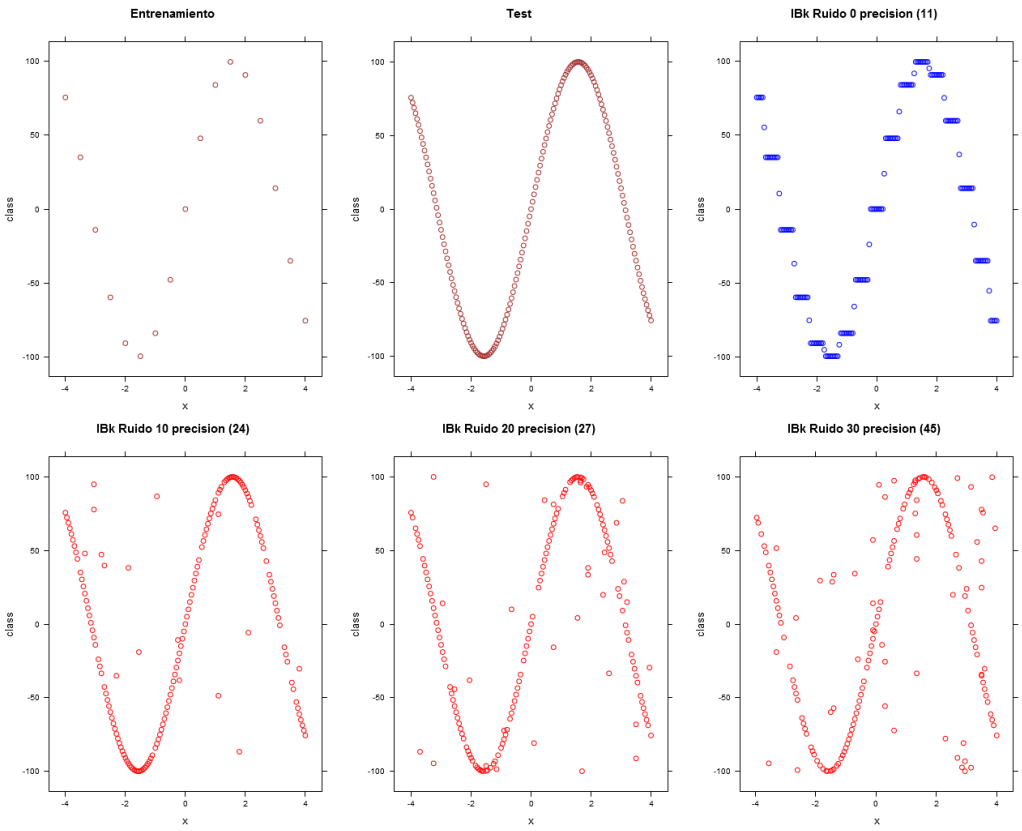

Figura 2.24: Función seno IBk

Los valores predichos por el modelo IBk, respecto a la función del seno, tienen bastante precisión y se adaptan aceptablemente a la forma que tiene la curva, para los escasos valores con los que se ha entrenado, obteniendo valores de error muy bajos, pero que aumentan considerablemente frente a la presencia de ruido.

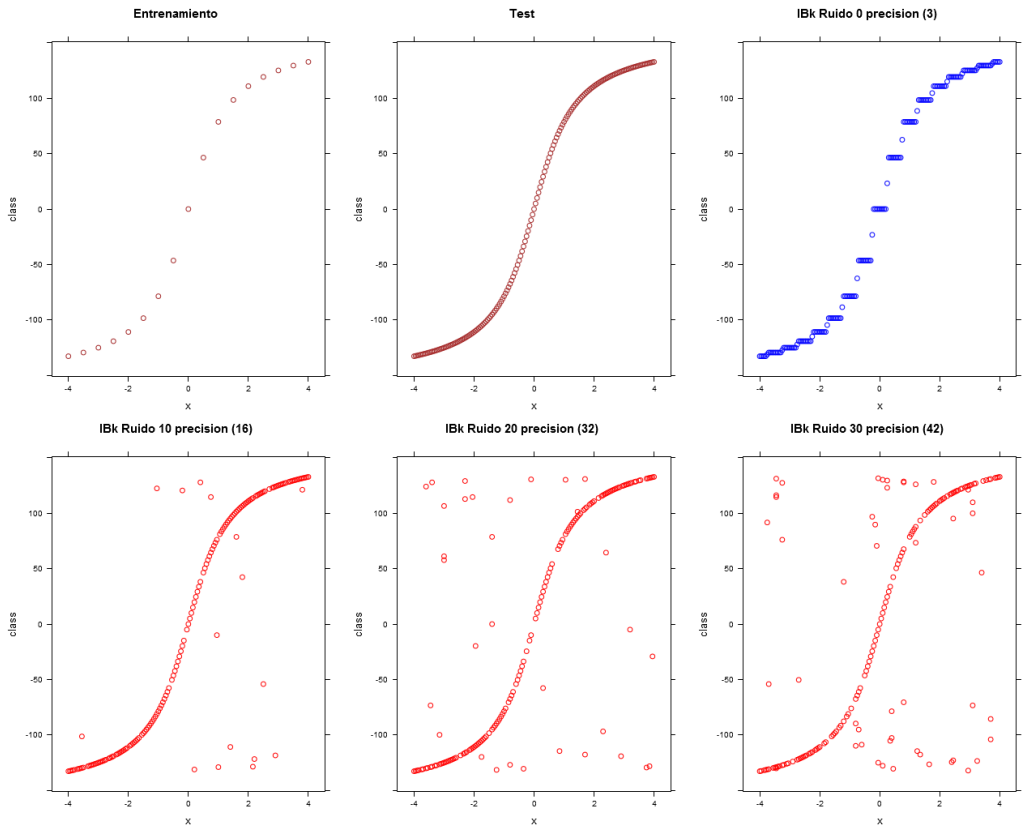

Figura 2.25: Función arcotangente IBk

Como en el caso de la función seno, para la función arcotangente el modelo se ajusta incluso mejor, llegando a obtener un error para el caso inicial de tan solo 3, sin presencia de ruido.

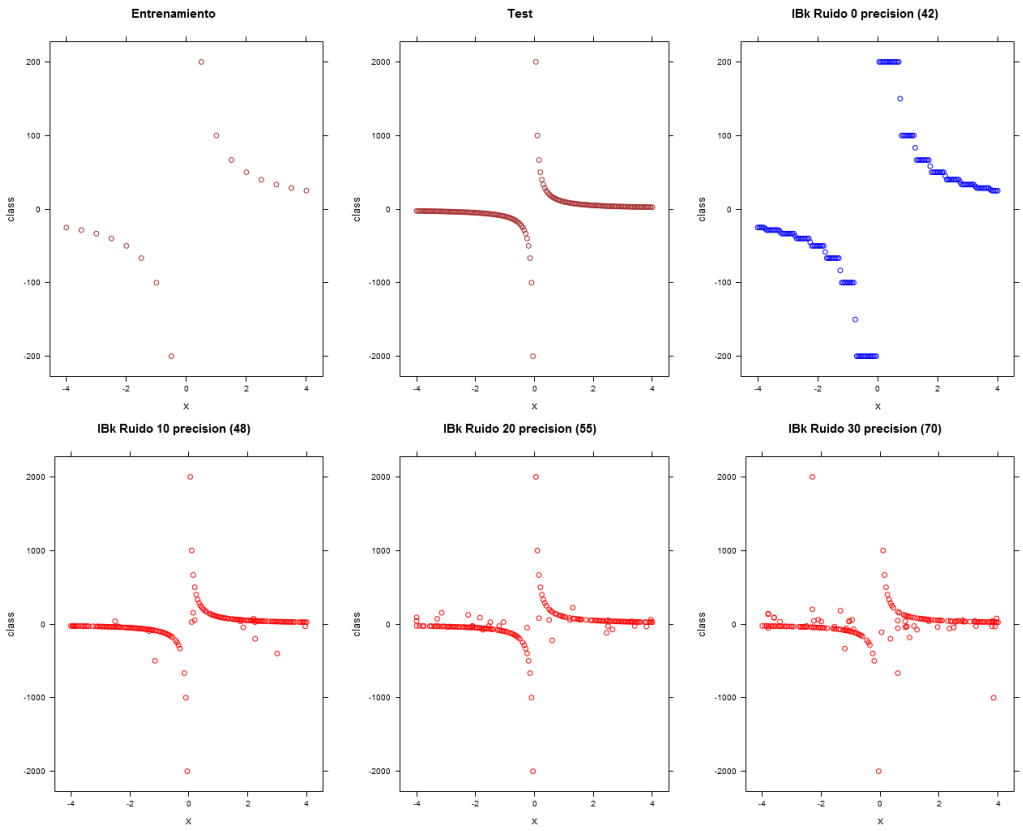

Figura 2.26: Función hipérbola equilátera IBk

Con IBk, la gráfica resultante se adapta muchísimo mejor que los modelos vistos hasta ahora, aunque el error no es tan bajo como para las otras función, es comprensible para los escasos datos de entrenamiento, sea como sea, su comportamiento es en todos los casos sensiblemente mejor a la media.

2.6.5. ZeroR

Modelo usado como referencia, únicamente predice la media para este caso de regresión.

No hace falta mostrar las gráficas resultantes de este modelo ya que este modelo que representa la media, tiene siempre el mismo aspecto y error de 100.

2.6.6. Ksvm

Las maquinas de vector soporte (SVM) son una excelente herramienta para clasificación y regresión.

Las SVM son clasificadores derivados de la teoría de aprendizaje estadístico postulada por Vapnik y Chervonenkis.

Las SVM fueron presentadas en 1992 y adquirieron fama cuando dieron resultados muy superiores a las redes neuronales en el reconocimiento de letra manuscrita, usando como entrada pixeles.

La máquina de vectores de soporte es un clasificador lineal que emplea la siguiente metodología:

- Mapear los puntos de entrenamiento a un espacio vectorial mayor.
- Construir un hiperplano que separe los puntos en sus clases respectivas.
- Clasificar un punto nuevo de acuerdo a su ubicación con respecto al hiperplano de separación.

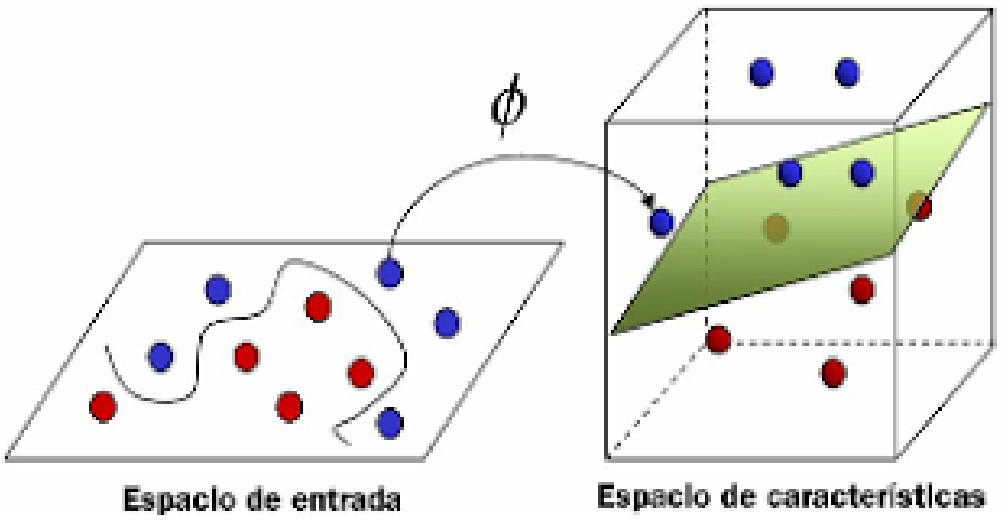

Figura 2.27: De esta manera se construye un hiplano de separación

El propósito del entrenamiento es maximizar la distancia entre los patrones y el hiperplano de separación (para obtener un clasificador más confiable). Pero dado que w es inversamente proporcional a dicha distancia, este mismo problema se puede expresar como la minimización de w.

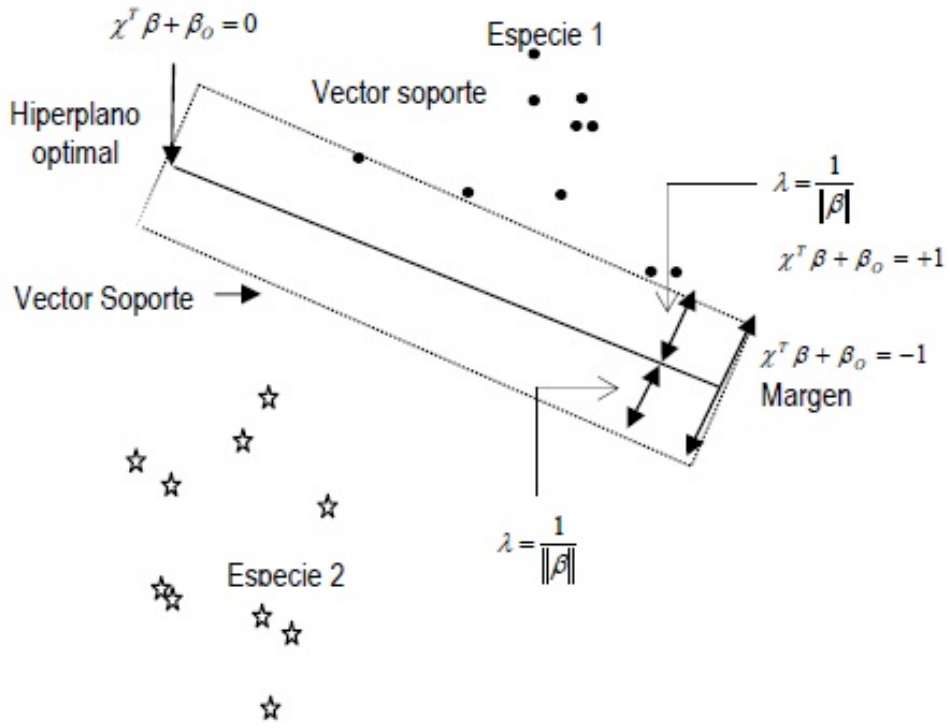

Figura 2.28: Distancia hiperplano optima

Cuando los datos no se pueden separar linealmente se hace un cambio de espacio mediante una función que transforme los datos de manera que se puedan separar linealmente. Tal función se llama Kernel.

También hay métodos para separar los datos (x_i, y_i) directamente aún no siendo separables linealmente, mediante funciones polinómicas y otro tipo de funciones, las Funciones de Base Radial (RBF).

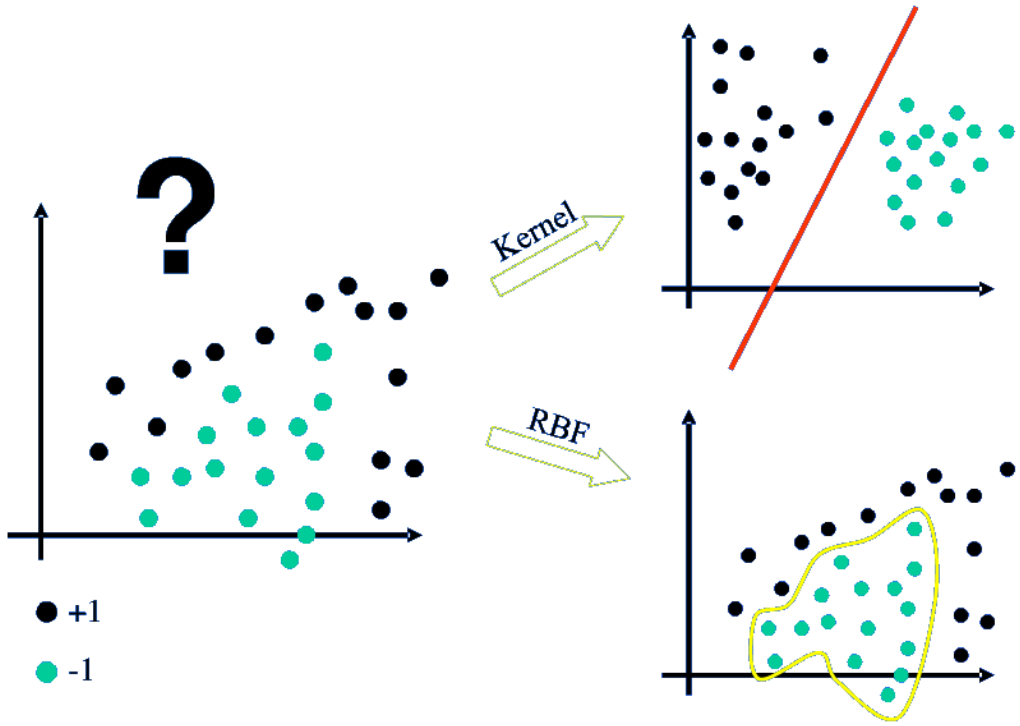

Figura 2.29: Kernel y Funciones de Base Radial

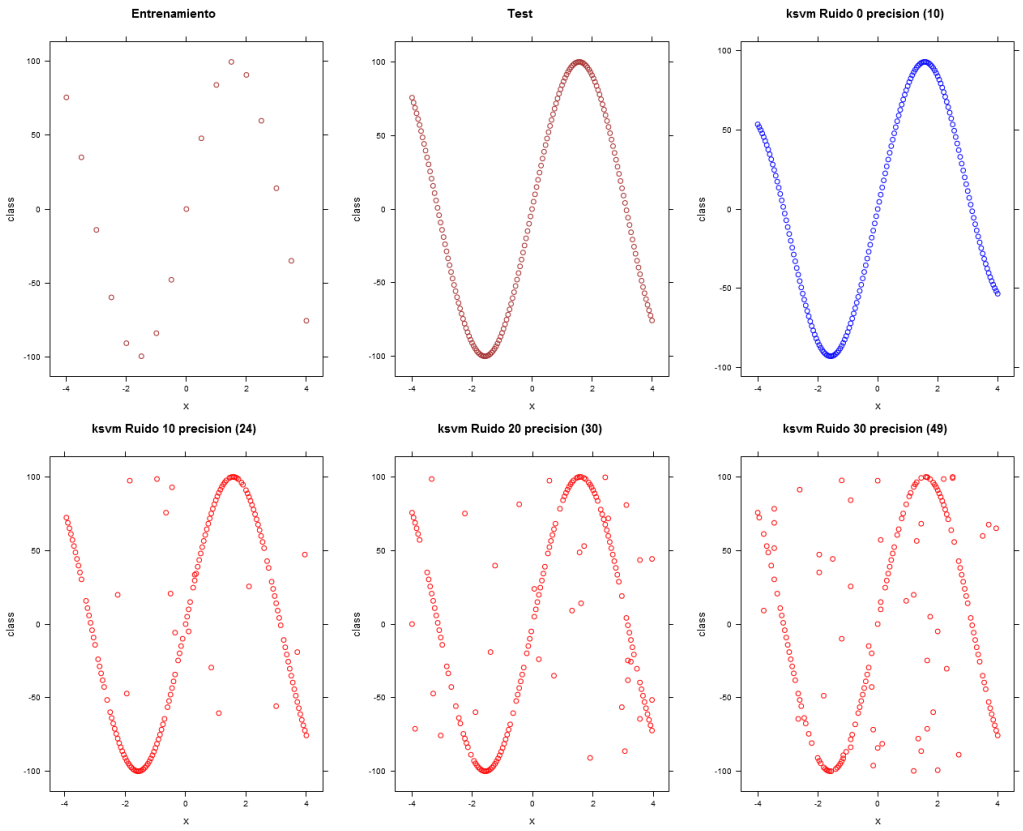

Figura 2.30: Función seno SVM

La gráfica generada con la predicción del modelo adopta una forma muy similar a la real, obteniendo errores muy bajos.

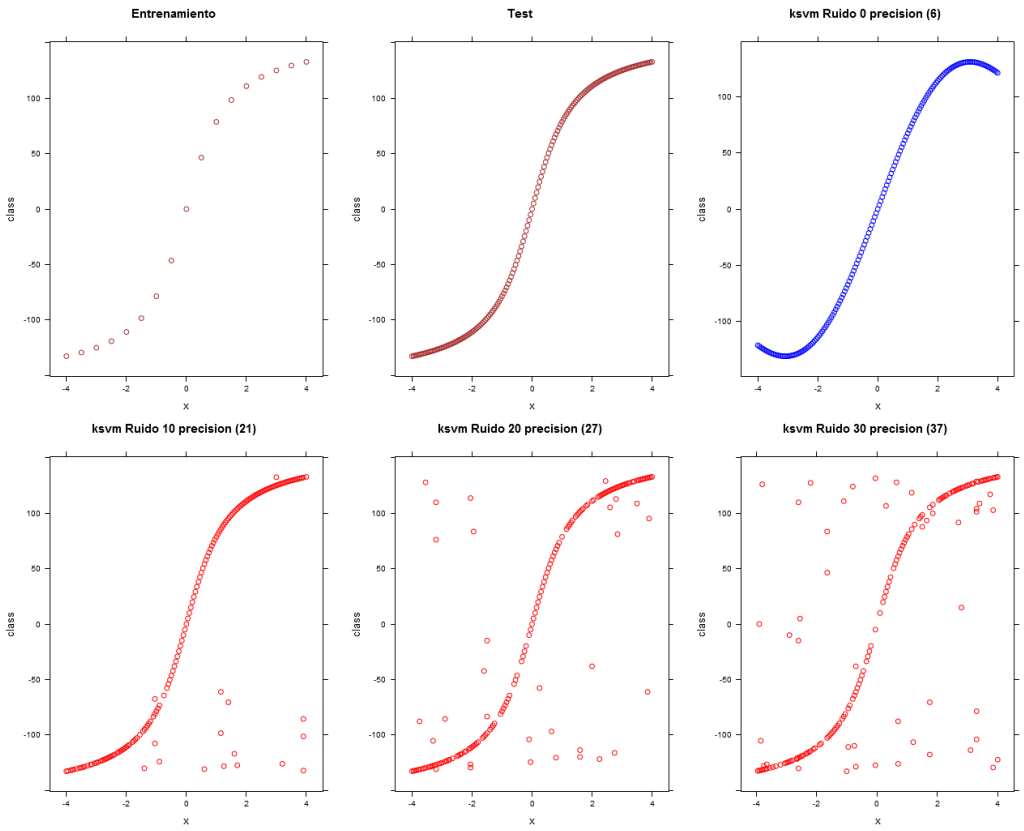

Figura 2.31: Función arcotangente SVM

Como en la función del seno, en el caso de la función arcotangente, la gráfica generada con la predicción del modelo también adopta una forma muy similar a la real, y se obtienen errores muy bajos.

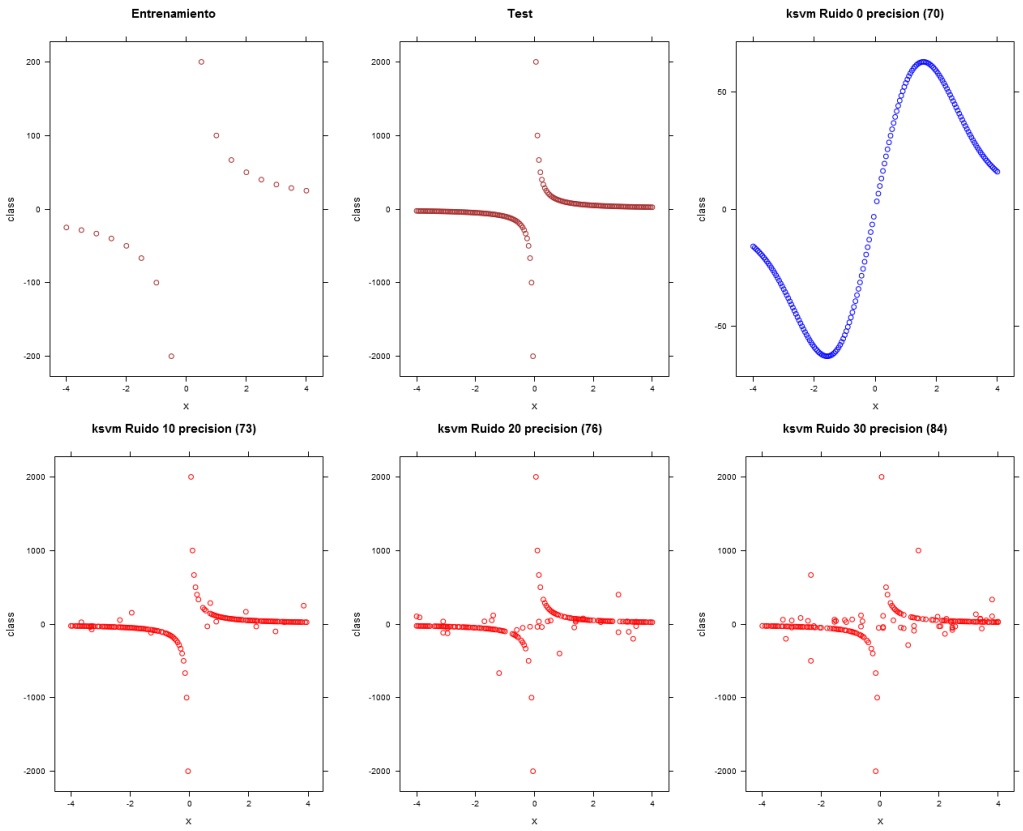

Figura 2.32: Función hipérbola equilátera SVM

El aspecto de la gráfica no es tan similar a la real y pese a partir de un error inicial de 69, no mantiene el nivel de precisión alcanzado con las otras dos funciones.

2.6.7. SMOreg

SMOreg implementa las maquinas de vector soporte para regresión. Los parámetros pueden ser aprendidos usando varios algoritmos. El algoritmo mas popular (RegSMOImproved) es debido a Shevade, Keerthi [14] y es el que se usa por defecto.

Este es el algoritmo para maquinas vector soporte que incluye weka y se comporta mucho peor que el anterior ksvm, puede que sea cuestión de adaptar

y precisar mejor sus parámetros de entrada, pero ksvm con los valores por defecto ha obtenido mejores resultados claramente.

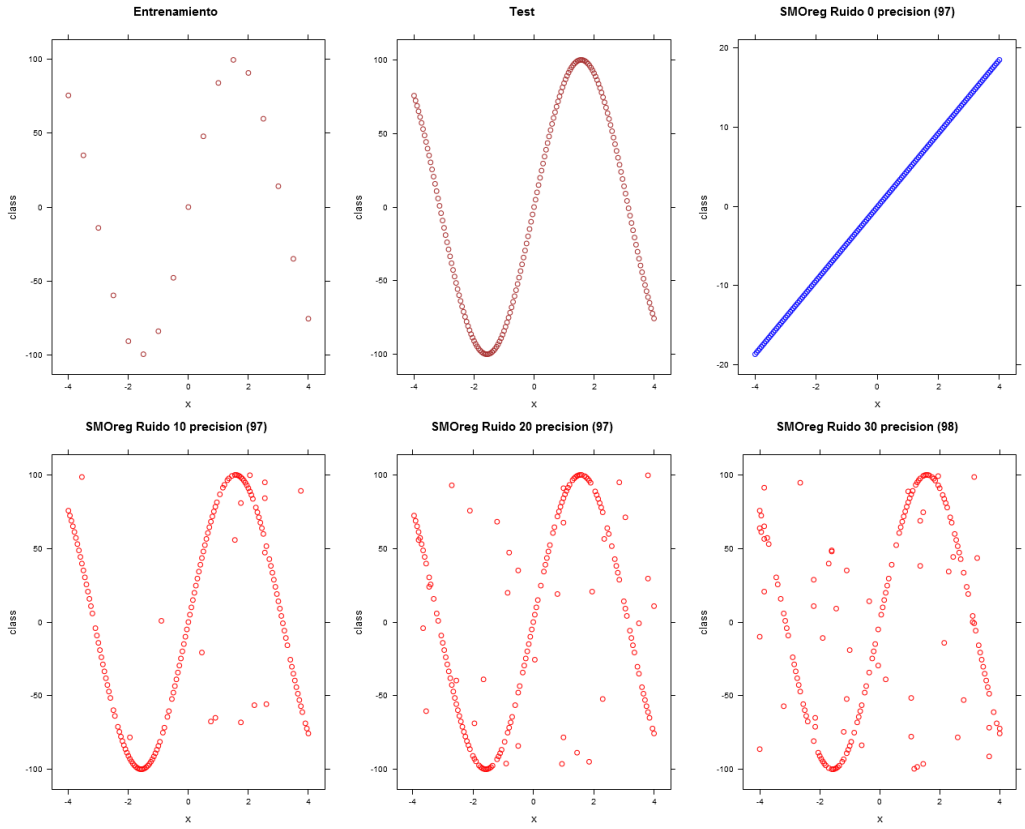

Figura 2.33: Función seno SMOreg

El modelo no se comporta bien con esta función, obteniendo valores similares a la media en todo momento, con lo que no se adapta para nada a la función descrita.

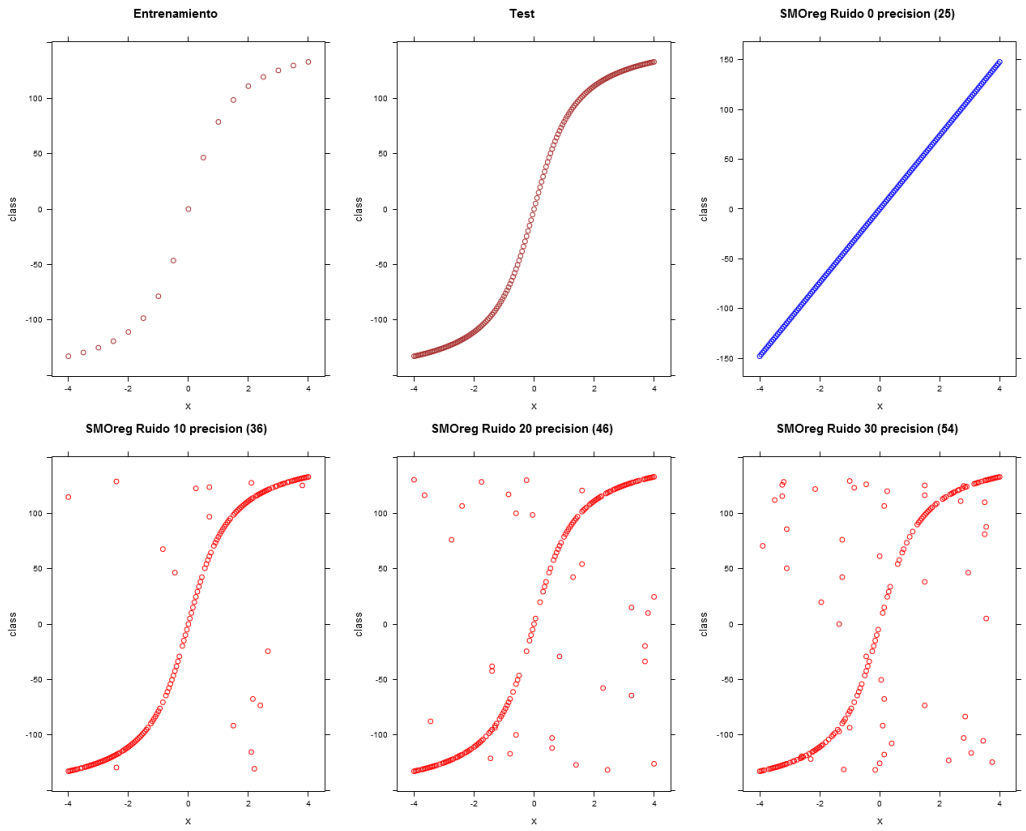

Figura 2.34: Función arcotangente SMOreg

En este caso de la función arcotangente, se consigue un error bajo con un aspecto de la gráfica similar a al que describe para la función de seno, pero que en este caso si se ajusta mucho con mucho mejor resultado.

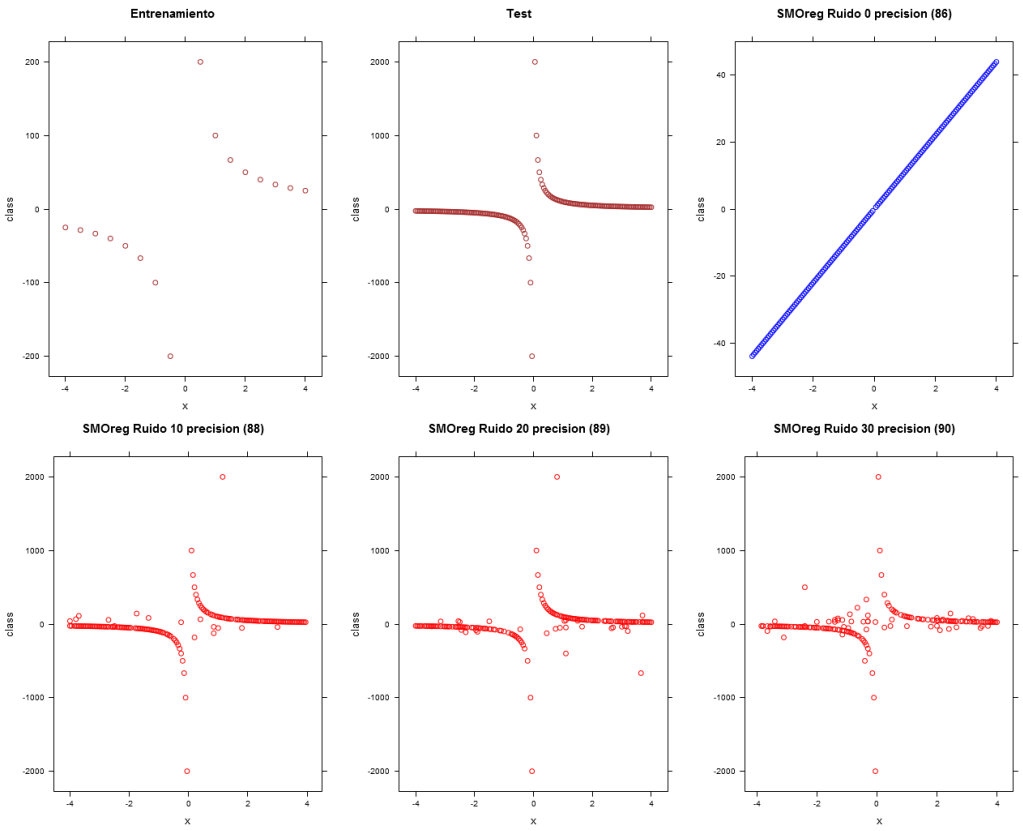

Figura 2.35: Función hipérbola equilátera SMOreg

De nuevo la forma de la gráfica generada es muy similar que las obtenidas para la función seno y arcontangente, y pese a mejorar la media, lo hace en unos niveles no demasiado distantes.

Capítulo 3

Resultados Experimentos

A la hora de realizar los estudios, han surgido problemas, como el diferente tratamiento de los nulos entre R y Weka, el tiempo de procesado de cada fichero, que en ocasiones era de varios días, pero finalmente se ha podido finalizar el estudio correctamente.

Se ha visto que el método de introducción de ruido por filas, columnas o matriz no tenia especialmente relevancia, salvo en algún dataset muy pequeño.

3.1. Metodología

Hemos analizado un total de 10 datasets para clasificación y otros 10 para regresión que a continuación detallamos con su número total de instancias y atributos, especificando cuantos de estos son de tipo nominal y cuantos de tipo numérico, para el caso de la clasificación se especifica también las clases o valores posibles para la etiqueta, al ser una variable discreta.

	Dataset	Instancias	Atributos	Nominal	Num.	Clases
1	Iris Plants Database	150	5	1	4	3
2	White Clover Persistence Trials	63	32	5	27	4
3	Hepatitis Domain	155	20	14	6	2
4	Breast cancer data	286	10	10	0	2
5	Vehicle silhouettes	846	19	18	1	4
6	German Credit data	1000	21	14	7	2
7	Primate gene sequences (DNA)	3190	62	61	1	3
8	Letter Image Recognition Data	20000	17	1	16	26
9	Waveform Database Generator	5000	41	1	40	3
10	Letter Image Recognition Data	2000	77	1	76	10

Tabla 3.1: Resumen de los 10 Datasets clasificación

Todos los datasets son públicos y accesibles vía web (`http://www.cs.waikato.ac.nz/ml/weka/datasets.html`). No entramos a detallar el motivo ni tipo de información que facilita cada dataset, pero puede consultarse su detalle descargando el fichero correspondiente.

	Dataset	Instancias	Atributos	Nominales	Numéricos
1	Heart Disease Databases	297	14	7	7
2	Percentage of Body Fat	252	15	0	15
3	Wisconsin Prognostic Breast Cancer	194	33	0	33
4	Daily stock prices	950	10	0	10
5	Results of Statlog project	264	22	2	20
6	The inhibition of dihydrofolate	186	61	0	61
7	Abalone data	4177	9	1	8
8	Computer Activity databases	8192	22	0	22
9	S&P Letters Data	20640	9	0	9
10	US Census Bureau	22784	9	0	9

Tabla 3.2: Resumen de los 10 Datasets regresión

Especificaremos en cada tipo de modelo de predicción las condiciones específicas de sus experimentos.

Para englobar y presentar los datos de una manera sencilla y directa a modo de conclusión final de los experimentos para cada modelo de predicción (regresión y clasificación), tras mostrar el desglose de los resultados de los experimentos por cada Dataset, se ha optado por recoger la robustez alcanzada por cada modelo frente a los distintos niveles de ruido de cada dataset en una única tabla resumen, ya que la cantidad de datos a mostrar es grande y serian muchas tablas. Para explicar dicho proceso veremos a continuación, tomando únicamente el primer dataset de regresión, cual ha sido el proceso

realizado (la tabla final contendrá los valores totales de todos los dataset no solo del primero como en este ejemplo).

Tomamos en primer lugar la tabla de resultados obtenidos para el primer Dataset de regresión, donde se observa para cada modelo y nivel de ruido cuál ha sido el error obtenido 3.3.

Ruido:	**0 %**	**5 %**	**10 %**	**15 %**	**20 %**	**30 %**	**40 %**	**50 %**
ZeroR	100	100	100	100	100	100	100	100
M5P	98	99	100	99	101	103	104	104
M5Rules	98	99	100	99	101	103	104	104
Linear Reg.	99	101	101	101	103	104	106	106
IBk	137	135	136	133	137	134	135	135
SVM	97	97	98	98	99	100	100	100
SMOreg	100	101	102	103	105	107	110	112

Tabla 3.3: 1) Precisión de modelos regresión para el primer Dataset

Eliminamos la fila del modelo ZeroR en regresión (en clasificación no lo hemos considerado), ya que simplemente se trata de la media por lo que mantiene siempre la misma precisión y no tiene cabida dentro de las conclusiones finales.

Eliminamos la columna sin ruido inicial (nivel de ruido 0 %), ya que no aporta información para la robustez frente al ruido. Para el resto de columnas, se obtiene la diferencia del error obtenido con respecto a la columna inicial sin ruido que hemos ocultado, para ver cuanto ha empeorado, es lo que se ve en la tabla 3.4 para el primer dataset de regresión. Si por ejemplo un modelo para su predicción sin ruido alcanza un error de 15, para un error del 5 % un error del 20, y para un error del 25 % un error del 36, no mostraríamos la columna inicial sin ruido, ya que sería la referencia inicial de precisión, pero si la siguiente con un valor de 5 (20-5) para el ruido al 5 % y la otra columna con un valor de 21 (36-15), que seria siempre la diferencia entre el error en la precisión para dicho nivel de ruido y la precisión inicial obtenida sin presencia de ruido. Cabe destacar que solo observamos el aumento de ruido, sin tener en cuenta la precisión inicial obtenida, así que un modelo de escasa precisión inicial para el Dataset puede acabar siendo considerado el mas robusto.

Una vez que hemos calculado estos incrementos de error producidos, otorgamos una puntuación a cada celda. Es decir, para cada nivel de ruido, obtenemos los incrementos de error en cada modelo y los ordenamos y agrupamos, siendo el menor error, el que mas robustez demuestre, y el mayor error el que

Ruido:	5 %	10 %	15 %	20 %	30 %	40 %	50 %
M5P	1	2	1	3	5	6	6
M5Rules	1	2	1	3	5	6	6
Linear Reg.	2	2	2	4	5	7	7
IBk	-2	-1	-4	0	-3	-2	-2
SVM	0	1	1	2	3	3	3
SMOreg	1	2	3	5	7	10	12

Tabla 3.4: 2) Incremento error de modelos regresión para el primer Dataset

se muestre mas sensible al ruido, con ello aportamos una puntuación ficticia donde los modelos con mayor error obtengan un cero y un punto mas que al anterior, para cada nivel de ruido inferior 3.5. Por ejemplo si para un nivel de ruido determinado, M5P tiene un incremento de 3 en su error, J48 un incremento de 4, IBk de 4 también y SVM de 1, entonces J48 e IBk obtendrían 0 puntos por ser los que mayor incremento de ruido han sufrido, después vendría M5P que obtendría un punto y SVM seria el mas robusto y obtendría 2 puntos.

Ruido:	5 %	10 %	15 %	20 %	30 %	40 %	50 %
M5P	1	0	2	2	1	2	2
M5Rules	1	0	2	2	1	2	2
Linear Reg.	0	0	1	1	1	1	1
IBk	3	2	3	4	3	4	4
SVM	2	1	2	3	2	3	3
SMOreg	1	0	0	0	0	0	0

Tabla 3.5: 3) Puntuación de modelos regresión para el primer Dataset

Con los resultados de esta tabla 3.5, podríamos obtener sumando las puntuaciones de todos los niveles de ruido, la puntuación total del modelo para este Dataset, que es la puntuación que debe aparecer en la tabla final con todos los Dataset, pudiendo ver que modelo ha sido el mas robusto y cual lo ha sido menos.

Una vez se ha obtenido esta puntuación, se realiza la misma operación para cada Dataset de regresión y clasificación y se agrupan estas puntuaciones por cada Dataset, pudiendo generar la tabla final con dicha información y un gráfico para visualizar gráficamente los resultados obtenidos.

3.2. Clasificación

Para el estudio sobre el comportamiento de modelos de clasificación, hemos usado los siguientes 10 datasets públicos, con las condiciones de 10 repeticiones, 5 particiones, errores (0, 5, 10, 15, 20, 30, 40, 50) para los tres tipos de introducción de ruido:

3.2.1. Dataset 1: Iris Plants Database

Las gráficas para el error por columnas o Matriz, prácticamente coinciden, mostrando la incidencia del ruido, algo mas acusada para los casos de Logistic y Naive Bayes:

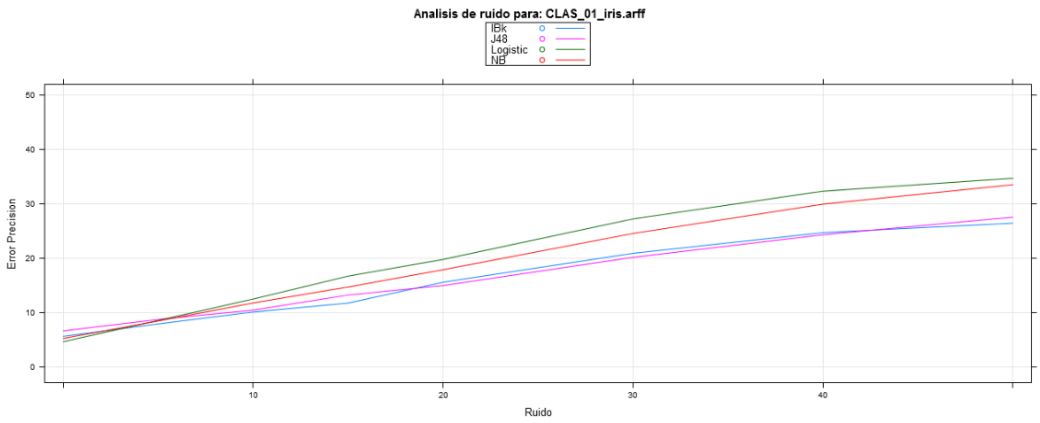

Figura 3.1: Efecto del ruido por columnas sobre el primer dataset de clasificación

Para el caso de ruido por filas, la gráfica difiere ya que el efecto del ruido se acusa muy pronto y luego se mantiene estable hasta el ruido al 50 %:

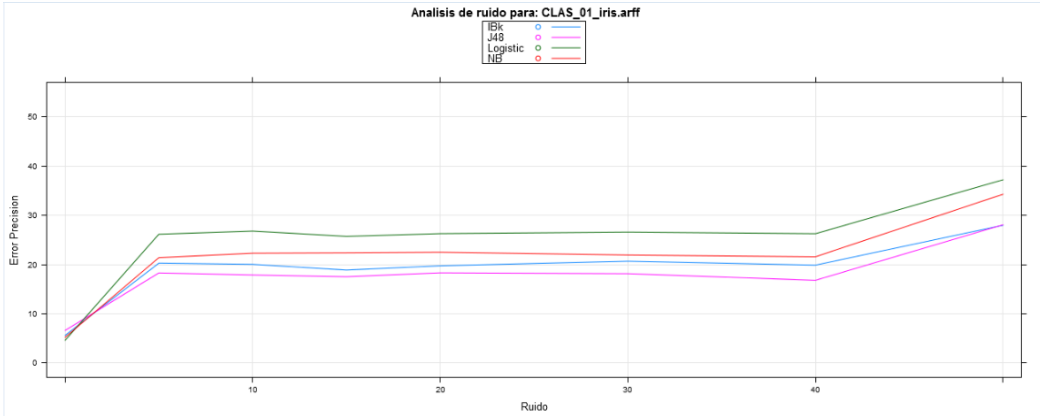

Figura 3.2: Efecto del ruido por filas sobre el primer dataset de clasificación

El motivo de la diferencia entre las gráficas es el número de campos al que afecta por ejemplo con un nivel de ruido al 5 %:

- Filas. 150 filas, 5 columnas. 150 filas por 5 % de 5 columnas (1 Columna). Afecta a 150 celdas.
- Columnas. 150 filas, 5 columnas. 5 columnas por 5 % filas (8 filas). Afecta a 40 celdas.
- Matriz. 150 filas, 5 columnas. 150 filas por 5 columnas son 750 celdas, su 5 % es 38 celdas.

La diferencia es clara, para un nivel de ruido bajo, en columnas y matriz afecta prácticamente al mismo numero de celdas, 38 y 40, frente al de filas para el que ya afecta a 150 registros.

Ruido:	**0 %**	**5 %**	**10 %**	**15 %**	**20 %**	**30 %**	**40 %**	**50 %**
J48	6	18	17	17	18	18	16	28
Logistic	4	26	26	25	26	26	26	37
NaiveBayes	5	21	22	22	22	21	21	34
IBk	5	20	20	18	19	20	19	27

Tabla 3.6: Precisión de Modelos, frente a ruido por Filas

Ruido:	0 %	5 %	10 %	15 %	20 %	30 %	40 %	50 %
J48	6	8	10	13	15	20	23	27
Logistic	4	7	12	16	19	25	31	34
NaiveBayes	5	7	10	14	17	22	28	33
IBk	5	7	10	12	15	20	23	27

Tabla 3.7: Precisión de Modelos, frente a ruido por Columnas

3.2.2. Dataset 2: White Clover Persistence Trials

Para este Dataset, las gráficas generadas para los tres tipos de error son parecidas, aunque tienen un comportamiento peculiar, muestran unos modelos muy estables frente al ruido, al único que parece afectar es al de Logistic que comienza siendo el que mas precisión tiene y acaba siendo uno de los que menos tiene.

Viendo el comportamiento de esta gráfica se puede entrever que los modelos de predicción usados para este Dataset, no son muy precisos, ya que desde el principio con los datos de test sin ruido ya tienen una precisión cercana al 50 %, la cual se mantiene con el ruido, ya que es prácticamente una predicción aleatoria. Haría falta un Dataset con mas registros.

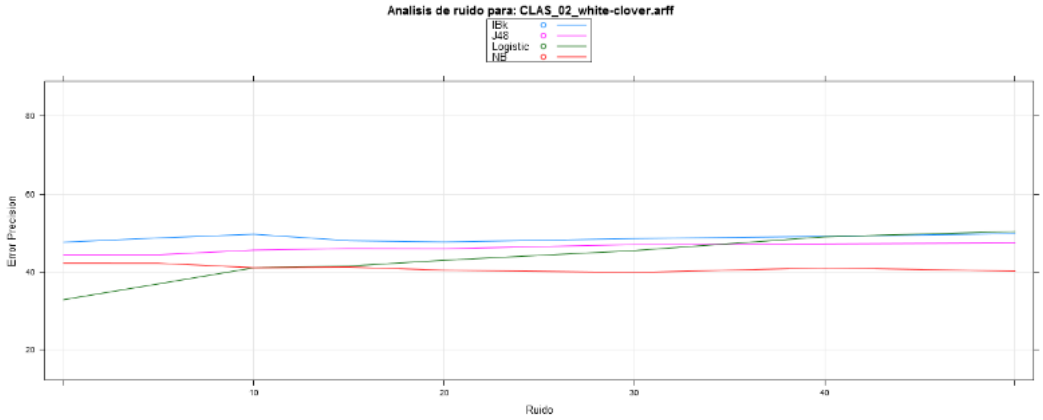

Figura 3.3: Efecto del ruido sobre el segundo dataset de clasificación

Ruido:	0 %	5 %	10 %	15 %	20 %	30 %	40 %	50 %
J48	44	44	45	45	45	46	47	47
Logistic	32	36	41	41	42	45	48	50
NaiveBayes	42	42	41	41	40	39	41	40
IBk	47	48	49	47	47	48	49	49

Tabla 3.8: Precisión de Modelos, frente a ruido por Matriz

3.2.3. Dataset 3: Hepatitis Domain

Para este Dataset se consiguen unos resultados parecidos al primer Dataset para los tipos de error matriz y columnas, salvo que al haber muchas mas columnas, no se produce la diferencia con el tipo de error por fila y en los 3 casos se comporta de una manera semejante. De nuevo los modelos a los que mas le afecta el ruido son Logistic y Naive Bayes.

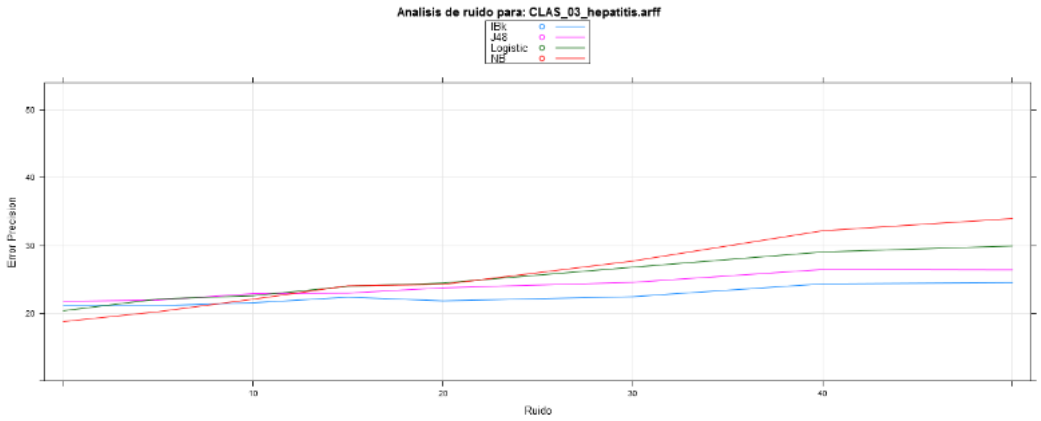

Figura 3.4: Efecto del ruido sobre el tercer dataset de clasificación

3.2.4. Dataset 4: Breast cancer data

Con este Dataset, no se aprecia mucha incidencia del ruido, es un dataset sobre un atributo de clasificación binario, con lo que la clase a acertar solo se pude acertar o fallar.

Ruido:	0 %	5 %	10 %	15 %	20 %	30 %	40 %	50 %
J48	28	29	29	29	29	29	31	32
Logistic	32	33	33	33	33	34	35	36
NaiveBayes	28	28	28	28	28	29	30	31
IBk	28	29	29	29	30	31	32	33

Tabla 3.9: Precisión de Modelos, frente a ruido por Filas

Ruido:	0 %	5 %	10 %	15 %	20 %	30 %	40 %	50 %
J48	28	28	28	29	29	30	31	32
Logistic	32	32	33	33	34	35	35	36
NaiveBayes	28	28	28	28	29	30	31	31
IBk	28	29	30	30	31	31	32	33

Tabla 3.10: Precisión de Modelos, frente a ruido por Columnas

Ruido:	0 %	5 %	10 %	15 %	20 %	30 %	40 %	50 %
J48	28	28	28	29	29	30	31	32
Logistic	32	32	33	34	34	34	36	37
NaiveBayes	28	28	28	29	30	29	31	32
IBk	28	29	29	30	31	31	33	33

Tabla 3.11: Precisión de Modelos, frente a ruido por Matriz

3.2.5. Dataset 5: Vehicle silhouettes

Sobre este Dataset se aprecia claramente la incidencia del ruido, para los tres tipos de error de una manera similar. De entre todos destaca como el menos estable frente al ruido el Logistic, ya que comienza como el mas preciso y acaba siendo el menos preciso, acusando gravemente el efecto del ruido, (de una precisión del casi 80 % pasa a una del 20

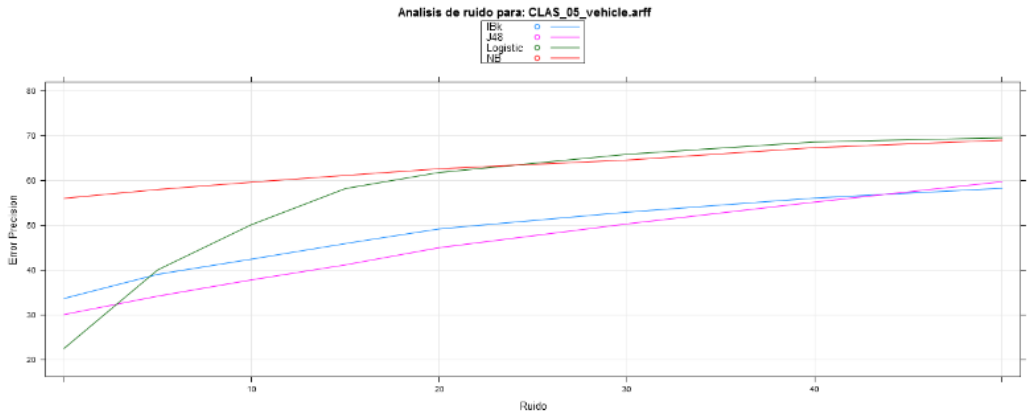

Figura 3.5: Efecto del ruido sobre el quinto dataset de clasificación

3.2.6. Dataset 6: German Credit data

Con este Dataset, también se ve claramente la incidencia del ruido, pero en este caso Naive Bayes es claramente el menos estable de todos.

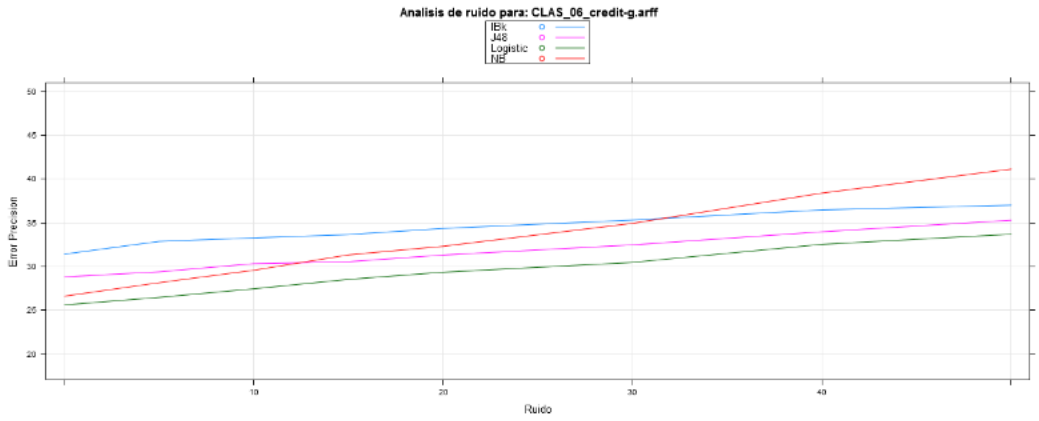

Figura 3.6: Efecto del ruido sobre el sexto dataset de clasificación

3.2.7. Dataset 7: Primate splice-junction gene sequences (DNA)

En este dataset se ve la incidencia del ruido y como afecta bastante a todos los modelos, de entre todos afecta adopta una pendiente mas acusada en el caso del J48.

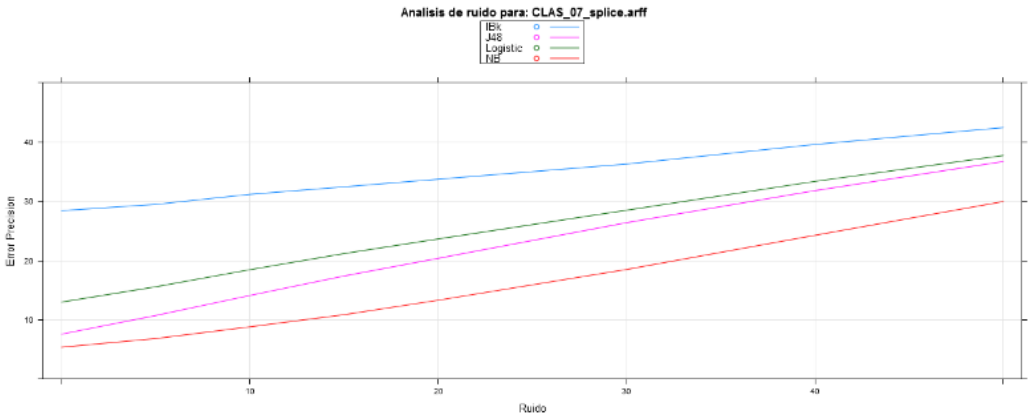

Figura 3.7: Efecto del ruido sobre el septimo dataset de clasificación

Ruido:	**0 %**	**5 %**	**10 %**	**15 %**	**20 %**	**30 %**	**40 %**	**50 %**
J48	7	10	14	17	20	26	31	36
Logistic	13	15	18	21	23	28	33	37
NaiveBayes	5	6	8	10	13	18	24	29
IBk	28	29	31	32	33	36	39	42

Tabla 3.12: Precisión de Modelos, frente a ruido por Matriz

3.2.8. Dataset 8: Letter Image Recognition Data

Se observa en este Dataset como los 4 modelos tienen un comportamiento similar, aunque entre los 4 resisten mejor el ruido Naive Bayes y J48.

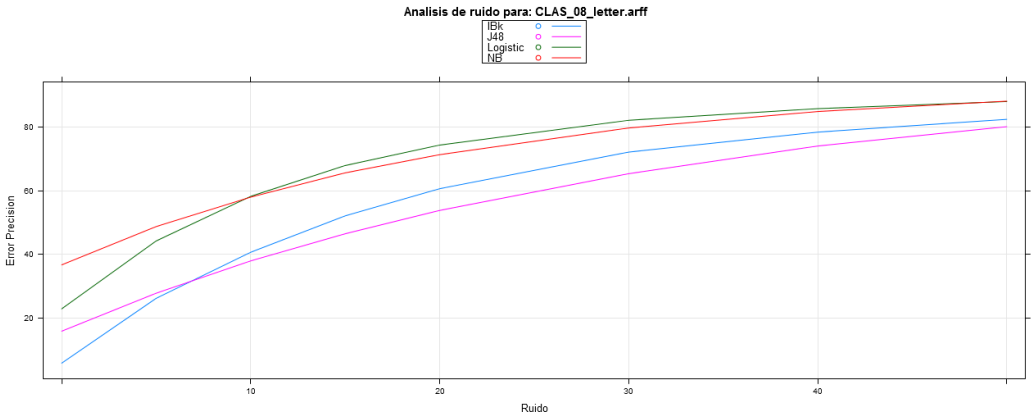

Figura 3.8: Efecto del ruido sobre el octavo dataset de clasificación

3.2.9. Dataset 9: Waveform Database Generator

Para este Dataset los 4 modelos mantienen una precisión bastante correcta, incluso a niveles de ruido altos, de entre ellos el que se muestra mas sensible al ruido es Naive Bayes en esta ocasión.

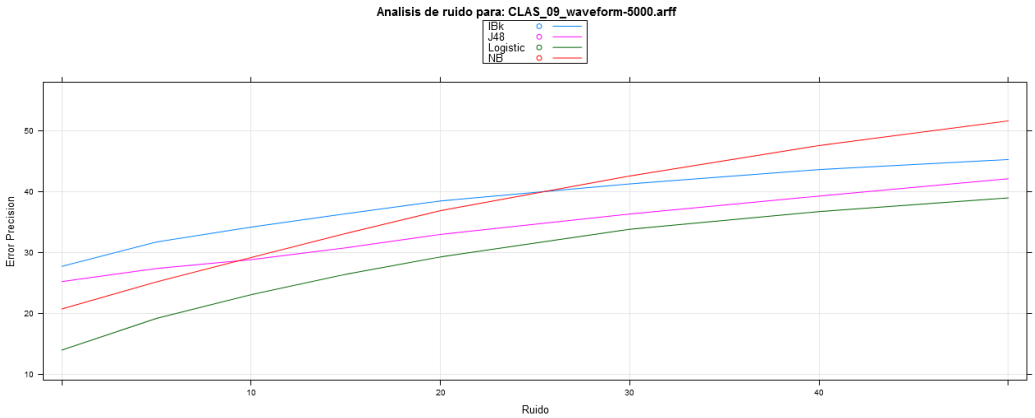

Figura 3.9: Efecto del ruido sobre el noveno dataset de clasificación

3.2.10. Dataset 10: Multi-feature digit

Viendo la gráfica de este dataset observamos como el ruido afecta de una manera mas clara y temprana para los modelos de Logistic y Naive Bayes.

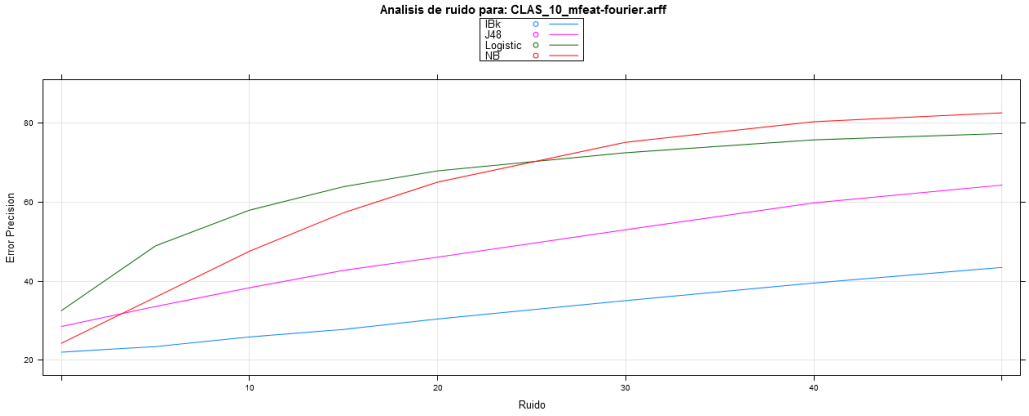

Figura 3.10: Efecto del ruido sobre el decimo dataset de clasificación

3.3. Resultados clasificación

En el estudio final de los modelos de clasificación se observa como el modelo mas robusto al ruido es el IBk, seguido muy de cerca por el J48, y el que peor se comporta con diferencia la regresión logística.

Como se comentó en el apartado previo de la sección de Metodología 3.1, resumimos en una única tabla final la puntuación obtenida por cada modelo y Dataset. Esta puntuación es el total obtenido por todos los niveles de ruido para cada dataset y en ella se refleja la robustez frente al ruido.

	J48	Logistic	NB	IBk		Vence
Dataset 1	18	0	7	15		J48
Dataset 2	9	0	20	12		NB
Dataset 3	13	6	0	19		IBk
Dataset 4	9	4	9	0		J48 / NB
Dataset 5	11	0	21	10		NB
Dataset 6	12	7	0	13		IBk
Dataset 7	0	7	13	18		IBk
Dataset 8	14	5	19	2		NB
Dataset 9	19	5	0	12		J48
Dataset 10	14	5	2	21		IBk
Total	119	39	91	122		IBk

Tabla 3.13: Puntuación final de modelos de clasificación para los Datasets

Con esta tabla podemos ordenar los resultados para el total de datasets, ordenando los modelos del más robusto frente al ruido al que lo es menos:

1. IBk. 4 victorias. 122 puntos.
2. J48. 3 victorias. 119 puntos.
3. NB. 4 victorias. 91 puntos.
4. Logistic. 39 puntos.

Este gráfico 3.11 sería una representación de la tabla anterior donde cada circulo corresponde a cada Dataset y en un color distinto la puntuación obtenida por el modelo. El primer dataset seria el circulo mas externo y el último dataset el de menor tamaño y mas interno.

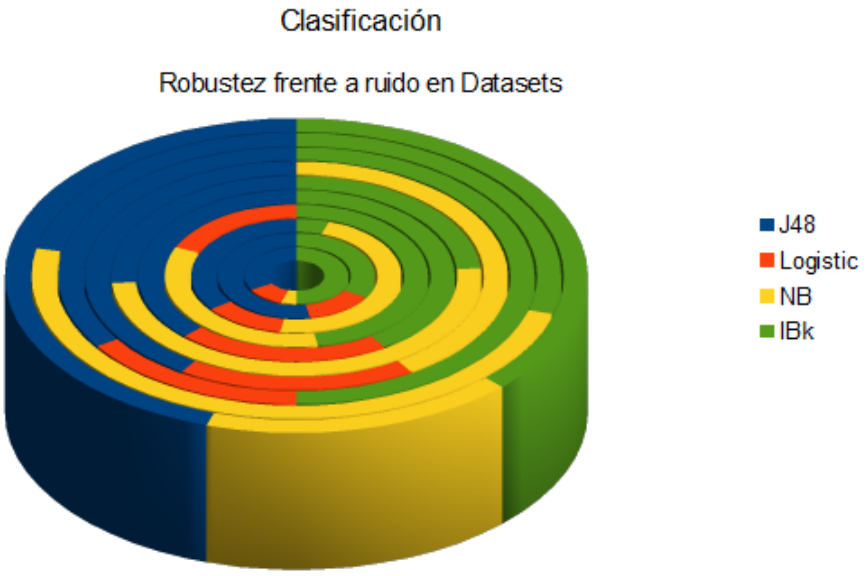

Figura 3.11: Promedio robustez ruido modelos clasificación en Datasets

3.4. Regresión

Para el estudio sobre el comportamiento de modelos de regresión, hemos usado los siguientes 10 datasets públicos, con las condiciones de 5 repeticiones, 4 particiones, errores (0, 5, 10, 15, 20, 30, 40, 50) para el tipo de introducción de ruido por matriz.

3.4.1. Dataset 1: Heart Disease Databases

Para este Dataset se ve como no hay ningún buen modelo predictivo, ya que sin presencia de ruido ya se tienen valores muy cercanos a la media, el error de los cuales al inyectarlo en seguida empeora los registros de ZeroR. Mención especial para IBk, que frente a este problema muestra un error elevado antes de introducir ruido, pero luego se mantiene inmune al mismo, habiendo ocasiones en las que incluso mejora su precisión.

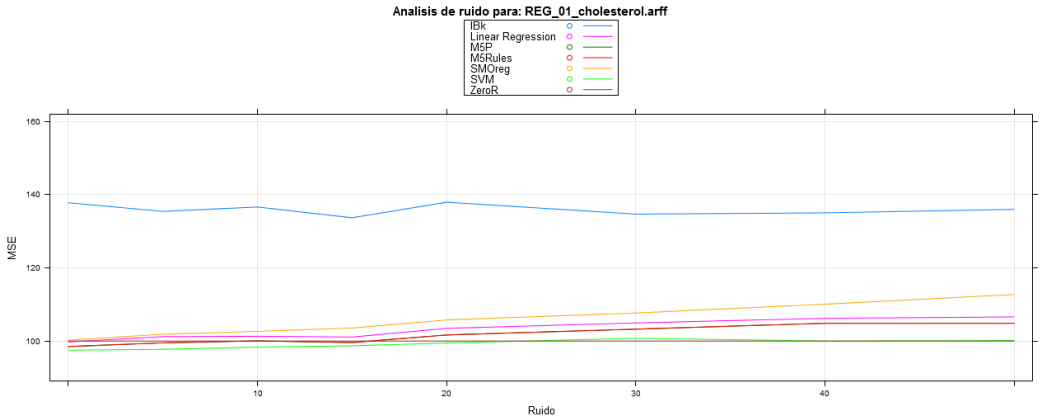

Figura 3.12: Efecto del ruido sobre el primer dataset de regresión

3.4.2. Dataset 2: Percentage of Body Fat

En este Dataset se obtienen resultados con los que sacar mas información provechosa, en ellos comprobamos como todos los modelos son claramente mejores que la media (ZeroR), sin la presencia de ruido, y cuando este empieza a aparecer va incrementándose su error hasta llegar a empeorar los resultados de la media a partir de un ruido al 30 %. En los casos de SVM e IBk, ni con ruido al 50 % empeoran los resultados de ZeroR, resulta especialmente estable frente al ruido IBk, ya que es el que tiene mayor error inicial (49) y es el que menos error tiene al final (99).

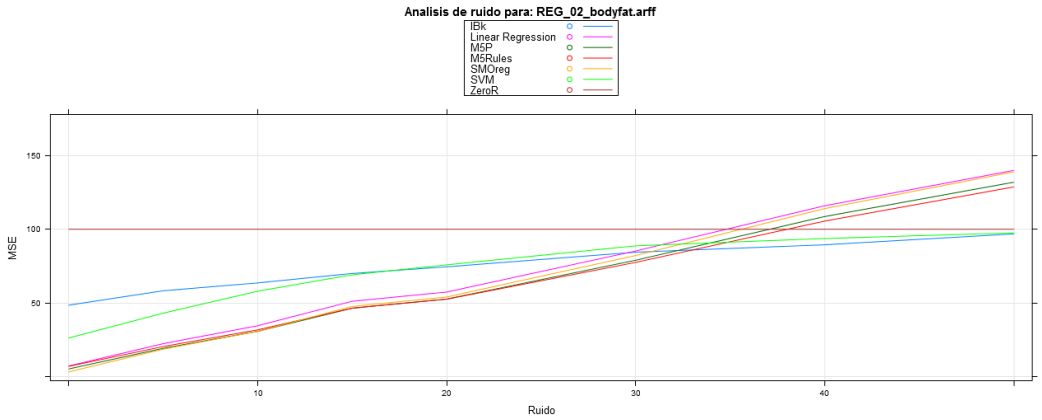

Figura 3.13: Efecto del ruido sobre el segundo dataset de regresión

3.4.3. Dataset 3: Wisconsin Prognostic Breast Cancer (WPBC)

En este caso, tenemos otro Dataset, para el que los modelos parten de un error similar a la media, con lo que su precision no es buena y esta empeora al inyectar ruido, sobre todo en el caso de la regresión lineal, que se dispara a valores superiores a 400 con el error al 50 %.

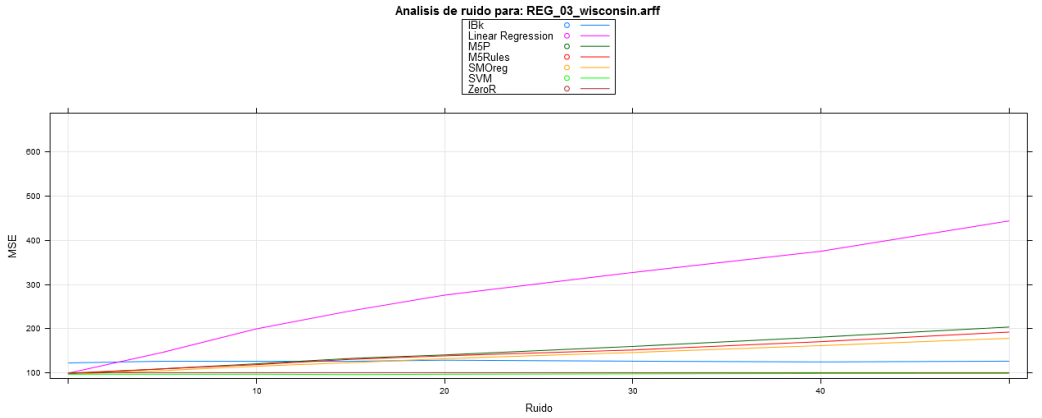

Figura 3.14: Efecto del ruido sobre el tercer dataset de regresión

3.4.4. Dataset 4: Daily stock prices for ten aerospace companies

Con este Dataset obtenemos resultados similares a los del segundo Dataset, en los que se parte de un error considerablemente menor a la media y que va incrementándose, según se aumenta el nivel de ruido, hasta llegar a empeorar a la media en todos los modelos salvo en el IBk y SVM.

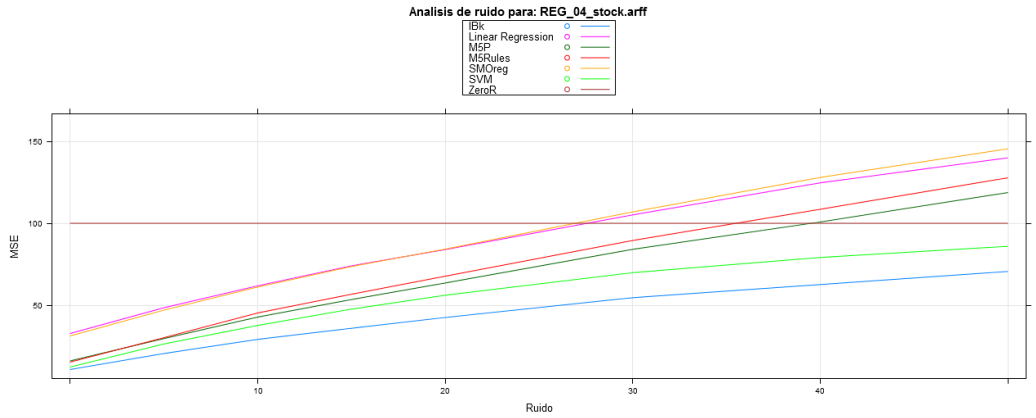

Figura 3.15: Efecto del ruido sobre el cuarto dataset de regresión

3.4.5. Dataset 5: Results of Statlog project

Para este Dataset nos encontramos con unos resultados que no nos ayudan a sacar muchas conclusiones, ya que el error inicial se mantiene prácticamente constantemente para todos los modelos, independientemente del nivel de ruido introducido. Digamos que en este caso la predicción realizada es bastante independiente con respecto a los datos usados para predecir.

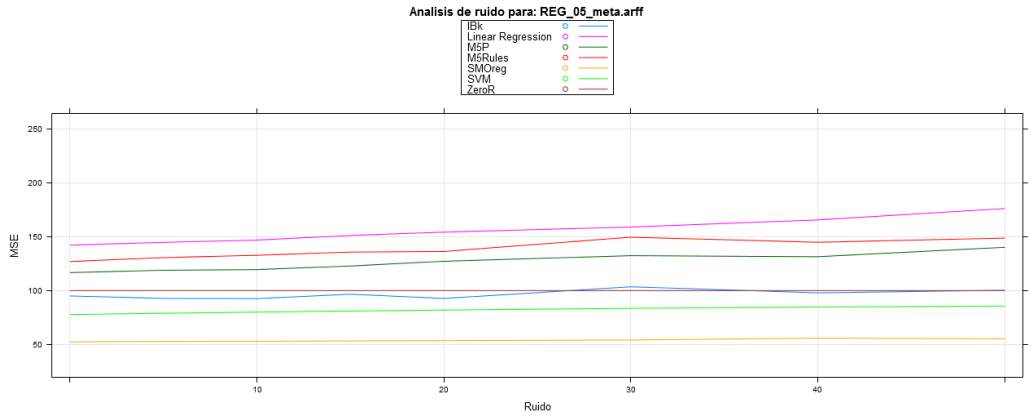

Figura 3.16: Efecto del ruido sobre el quinto dataset de regresión

3.4.6. Dataset 6: The inhibition of dihydrofolate reductase by triazines

Con este Dataset obtenemos una gráfica similar a la del tercer Dataset.

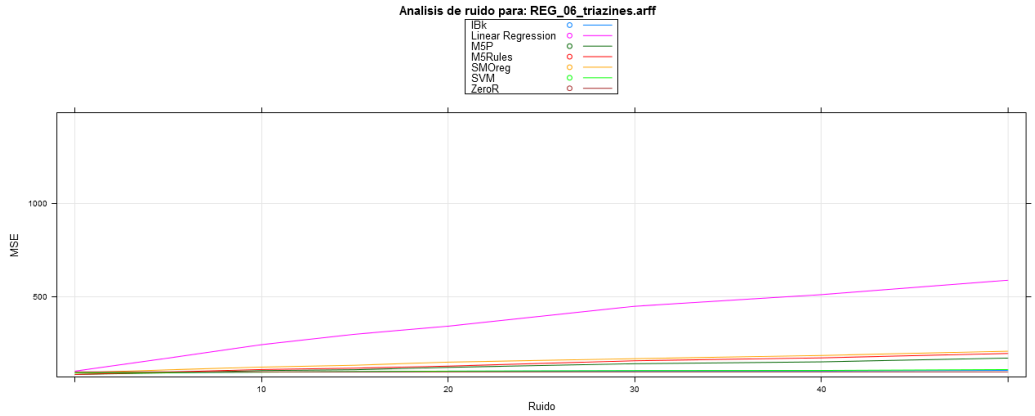

Figura 3.17: Efecto del ruido sobre el sexto dataset de regresión

En la tabla podemos observar que el error para la regresión lineal casi alcanza 600.

Ruido:	0 %	5 %	10 %	15 %	20 %	30 %	40 %	50 %
ZeroR	100	100	100	100	100	100	100	100
M5P	86	96	108	117	126	140	153	170
M5Rules	86	100	113	127	139	156	179	195
Linear Reg.	104	184	254	306	352	425	550	593
IBk	95	97	98	101	100	106	104	108
SVM	95	96	97	102	104	107	110	112
SMOreg	95	115	127	144	149	177	202	212

Tabla 3.14: Error relativo medio de los Modelos, frente a ruido por Matriz

3.4.7. Dataset 7: Abalone data

Para este Dataset observamos claramente como pese a iniciar con una precisión bastante baja, el error se dispara en todos los modelos salvo para IBk y SVM, una vez más.

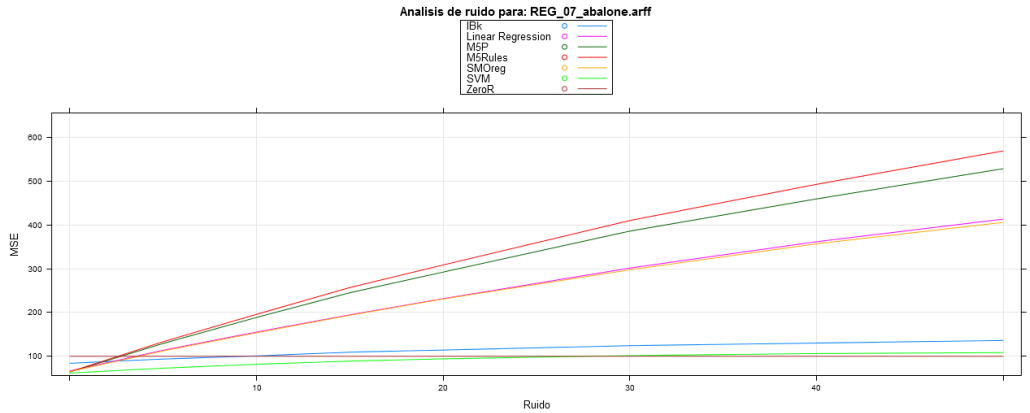

Figura 3.18: Efecto del ruido sobre el séptimo dataset de regresión

3.4.8. Dataset 8: Computer Activity databases

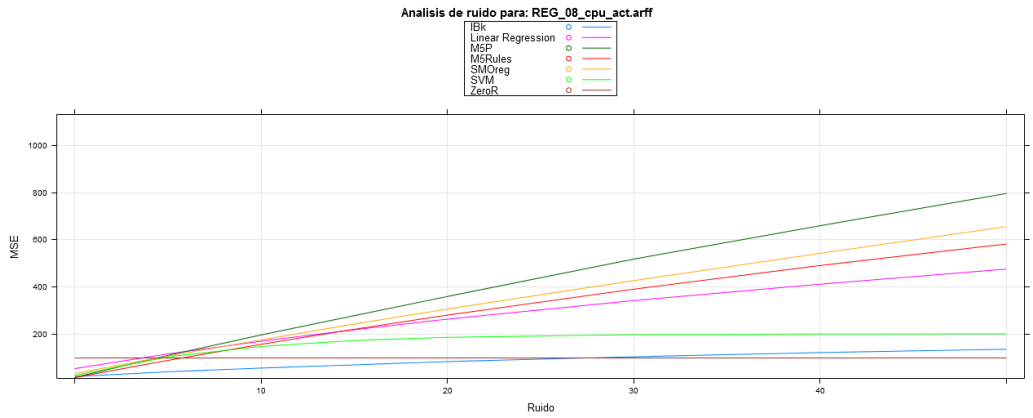

Figura 3.19: Efecto del ruido sobre el octavo dataset de regresión

Una vez mas el aspecto de la gráfica es similar, aunque en esta ocasión podemos destacar que el modelo de SVM, pese a mostrarse mas robustos que el resto de modelos, salvo el IBk, alcanza niveles de error de 200, como podemos ver en mas detalle en la tabla con los resultados.

Ruido:	**0 %**	**5 %**	**10 %**	**15 %**	**20 %**	**30 %**	**40 %**	**50 %**
ZeroR	100	100	100	100	100	100	100	100
M5P	18	111	197	279	360	518	659	796
M5Rules	17	89	157	221	281	391	491	582
Linear Reg.	55	117	171	219	264	342	412	476
IBk	21	41	57	70	84	105	122	136
SVM	26	104	149	174	187	198	201	202
SMOreg	34	108	176	243	307	427	542	655

Tabla 3.15: Error relativo medio de losModelos, frente a ruido por Matriz

3.4.9. Dataset 9: SP Letters Data

De nuevo se muestra una gráfica de aspecto similar, donde el error se dispara aún más y solo se muestra estable para los modelos IBk y SVM.

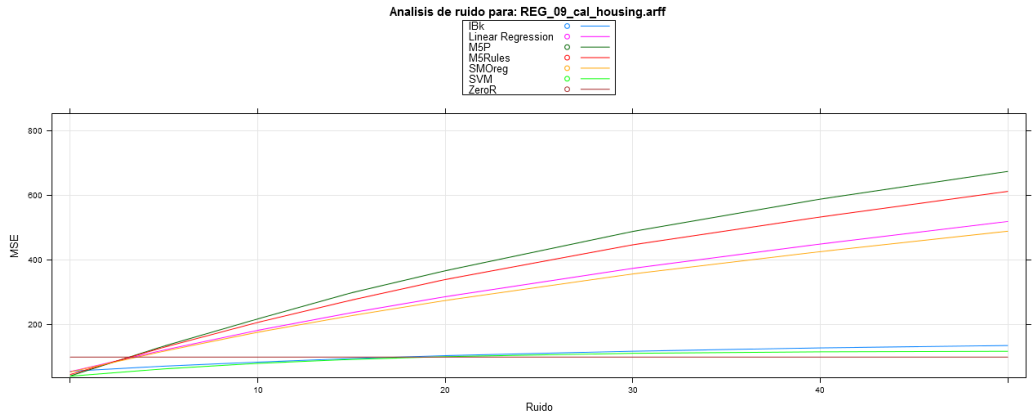

Figura 3.20: Efecto del ruido sobre el noveno dataset de regresión

3.4.10. Dataset 10: US Census Bureau

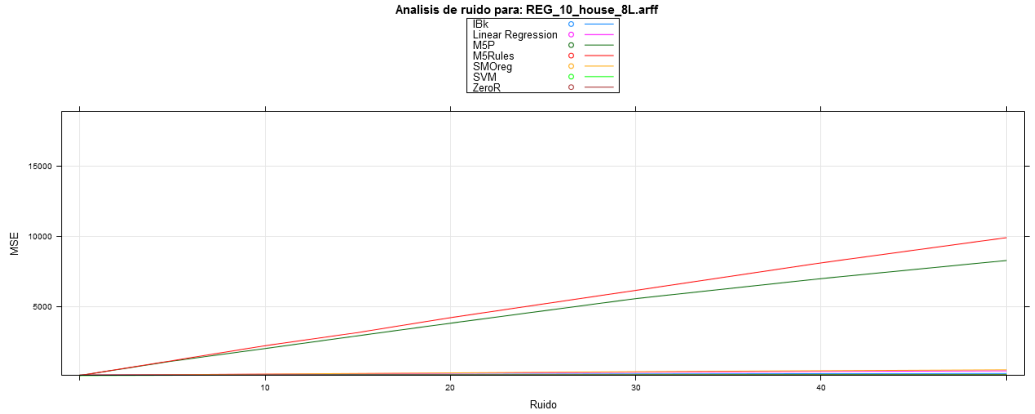

Figura 3.21: Efecto del ruido sobre el decimo dataset de regresión

En la gráfica del último Dataset, se dispara tantísimo el error para los modelos de M5Rules y M5P, que no se puede observar el comportamiento del resto de modelos, en la tabla se muestra lo pronto y en que magnitud crece el error, mientra en el resto de modelos se obtienen resultados mas habituales.

Ruido:	0 %	5 %	10 %	15 %	20 %	30 %	40 %	50 %
ZeroR	100	100	100	100	100	100	100	100
M5P	52	1083	1981	2891	3789	5549	6977	8271
M5Rules	54	1114	2183	3122	4187	6135	8102	9903
Linear Reg.	74	116	154	191	223	287	341	390
IBk	70	92	112	130	145	169	189	202
SVM	48	67	82	94	103	116	123	127
SMOreg	64	112	158	202	243	324	397	464

Tabla 3.16: Error relativo medio de los Modelos, frente a ruido por Matriz

3.5. Resultados regresión

Como en la clasificación, en el estudio final de los modelos se observa como el modelo mas robusto al ruido es el IBk, seguido de cerca por el SVM, y a una distancia considerable de los dos primeros, el resto de ellos.

Como se comentó en el apartado previo de la sección de Metodología 3.1, resumimos en una única tabla final la puntuación obtenida por cada modelo y Dataset. Esta puntuación es el total obtenido por todos los niveles de ruido para cada dataset y en ella se refleja la robustez frente al ruido.

	M5P	M5Rules	Linear Reg.	IBk	SVM	SMOreg		Vence
Dataset 1	10	10	5	23	16	1		IBk
Dataset 2	15	22	6	32	15	2		IBk
Dataset 3	7	14	0	28	34	21		SVM
Dataset 4	19	6	9	32	24	0		IBk
Dataset 5	10	6	2	28	19	27		IBk
Dataset 6	21	14	0	33	30	7		IBk
Dataset 7	7	0	15	30	28	18		IBk
Dataset 8	0	15	23	35	24	8		IBk
Dataset 9	0	7	14	34	29	21		IBk
Dataset 10	7	0	21	28	35	14		SVM
Total	96	94	95	303	254	119		IBk

Tabla 3.17: Puntuación de modelos clasificación para los Datasets

Con esta tabla podemos ordenar los resultados para el total de datasets, ordenando los modelos del más robusto frente al ruido al que lo es menos:

1. IBk. 8 victorias. 303 puntos.

2. SVM. 2 victorias. 254 puntos.

3. SMOreg. 119 puntos.

4. M5P. 96 puntos.

5. Linear Regression. 95 puntos.

6. M5Rules. 94 puntos.

Este gráfico 3.22 sería una representación de la tabla anterior donde cada circulo corresponde a cada Dataset y en un color distinto la puntuación obtenida por el modelo. El primer dataset seria el circulo mas externo y el último dataset el de menor tamaño y mas interno.

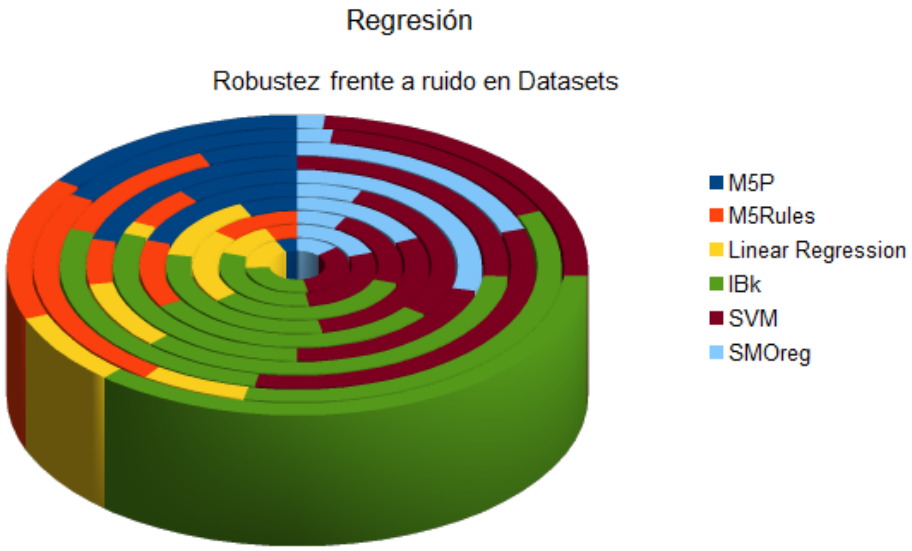

Figura 3.22: Promedio robustez ruido modelos regresión en Datasets

Capítulo 4

Conclusiones

En este trabajo hemos realizado una serie de experimentos para conocer como afecta a la precisión de los modelos de clasificación y regresión, la inyección incremental de ruido en sus datos de testeo, estos modelos engloban algunas de las principales técnicas de resolución de problemas de minería de datos, tales como arboles de decisión, regresión logística, maquinas vector soporte, etc. Los experimentos se han repetido varias veces para cada nivel de ruido, sobre un total de veinte datasets, realizando a su vez diferentes particiones iniciales de los datos de entrenamiento y testeo para lograr unos resultados mas genéricos y que no dependan tanto de la partición inicial.

Una vez se han obtenido los resultado y tenemos la comparativa del comportamiento de los modelos frente al ruido introducido, podemos realizar ciertas conclusiones.

- Los diferentes tipos de ruido empleados en general no tienen especial relevancia en el efecto del ruido, influye mas la diferencia en el numero de atributos afectados y de la importancia que el modelo atribuye al atributo para realizar la predicción.
- Con la implementación del algoritmo utilizada para las maquinas vector soporte (ksvm), se aprecia que el efecto provocado por el ruido y que provoca la perdida progresiva de precisión, llega un momento en que es controlada cuando el nivel de ruido es alto y pasa a actuar como si fuera la media, dando en todos los casos el mismo valor, de manera que

deja de verse afectada por el mismo aunque sus predicciones pierden su valor.

- De entre todos los modelos analizados, el que obtiene una mayor robustez frente al ruido, tanto en regresión como en clasificación es el IBk, extrapolando que generalmente, al afectar el ruido a atributos aleatoriamente, continua pudiendo ver semejanzas entre los registros para clasificarlos correctamente.

En el futuro seria conveniente reemprender este estudio con una maquina mas potente, para realizar muchas mas iteraciones y análisis de datasets en un tiempo mas razonable, y abarcar una mayor casuística. También seria interesante estudiar más técnicas de aprendizaje, así como nuevas maneras de generar error y se podría plantear el estudiar algún mecanismo para determinar que el nivel de error (o cambio del propio problema) es tan alto que convendría reentrenar el modelo.

Apéndices

Codigo fuente para procesar los datasets, introduciendo ruido y generando graficas:

```
# Funcion para escribir en logs
logger <- function(contenido, fichero) {
  print(paste(format(Sys.time(), "%X"), "-",contenido))
  ficheroLogger <- file(paste0(fichero,"/3_Log.txt"), "at") # open
      an output file connection
  cat(paste(format(Sys.time(), "%X"), "-",contenido), file =
      ficheroLogger, sep = "\n")
  close(ficheroLogger)
}

# Funcion global
analizarRuido <- function(
     nombreDatos=c("iris.arff","iris.arff","weather.arff","weather.arff"),
     dirDataset="C:/Users/Eclipse/Documents/Master/Tesis/SVN/Datasets/",
     dirTmp="C:/Users/Eclipse/Documents/Master/Tesis/SVN/tmp/",
     proporcionTrain=0.5,numRepeticiones=100,numParticiones=5,
     saltoError = c(0,5,10,15,20,30,40,50),ruidos=c(1,2,3)) {

  # Cargamos librerias
  # install.packages("RWeka")
  library("RWeka")
  # install.packages("lattice")
  library("lattice") # xyplot
  # install.packages("sampling")
  library("sampling") # Strata
  # install.packages(c("Cairo"), repos="http://cran.r-project.org" )
  # install.packages("Cairo")
```

```
library("Cairo") # Antialiasing
# install.packages("kernlab") # For ksvm
# install.packages("kernlab")
library("kernlab")
# install.packages("gridExtra")
require("gridExtra") # grid.arrange

# Funciones Internas...

# Estratificamos la coleccion, en funcion del numero de columna
    indicado, con una proporcion de 0 a 1
# retornamos los indices de la coleccion obtenidos de la
    estratificacion
stratificame <- function (esClasificacion, datos, colStrat,
    proporcion) {
  if (esClasificacion) {
    # Para clasificacion reealizamos strata

    # Ordenamos datos
    datosOrden <- datos[order(datos[,colStrat]),]
    # Calculamos proporciones iniciales
    proporcionesIni<-c()
    if (is.numeric(datos[,colStrat])) {
      # Columna a stratificar numerica, lo hacemos a mano
      valores<-unique(datos[,colStrat])
      for (valor in valores) {
        proporcionesIni<-c(proporcionesIni,
            length(datos[datos[,colStrat]==valor,1]))
      }
    } else {
      # Columna a stratificar no numerica, podemos usar el
          summary que las separa
      proporcionesIni<-summary(datos[,colStrat])
    }
    # Indicamos cuantos de cada tipo, si [25, 25, 25] y
        proporcion 0.5, entonces [12,12,12]
    proporciones <-
        c(trunc(as.numeric(proporcionesIni*proporcion)))
    # No puede haber valores a cero o peta, en estos casos
        ponemos 1
    i<-1
    for (proporcionPos in proporciones) {
      if (proporcionPos == 0) {
```

```
        proporciones[i] = 1
      }
      i<-(i+1)
    }
    # print(paste("hay:",proporcionesIni))
    # print(paste("(",proporcion,") separamos:",proporciones))

    # Realizamos estratificacion, CUIDADO CON LOS NOMBRES CON
       GUION STRATA FALLA!!!!!
    # names(datosOrden)[colStrat] = "WhiteClover94"
    # print(toString(names(datosOrden)[colStrat]))
    tmp<-strata(datosOrden,c(toString(names(datosOrden)[colStrat])),
                        size=proporciones,
                           method="srswor",description=FALSE)
    # Retornamos indices
    return(tmp$ID_uni)
  } else {
    # Para regresion separamos aleatorio en la proporcion indicada
    tam<-length(datos[,1]) #
    indices<-sample(1:tam) # indices desordenados de los datos
    mig<-trunc(proporcion*tam) # particion segun parametro
       proporcion
    return(indices[1:mig])
  }
}

# Funcion de analisis del ruido para los 4 modelos,
# retorna los datos para pintar la grafica
analizarRuidoInterna <- function(fichero) {

  # Funcion interna analizarRuidoInterna: Para rellenar un
     registro de los datos de la grafica
  rellenarGrafica <- function(ruido, precision, modelo, todo,
     testError, textoLog) {

    # Escribimos fichero con ruido y su resultado
    # write.arff(testError, paste0(dirTmp,fichero,"/4_Test_",
    #  ruido,"_",modelo,"_",trunc(precision),".arff"))
    logger(paste0(textoLog," ",modelo,": ",precision),
       dirParticion)

    precision<-as.numeric(precision)
    result <- list(Ruido=c(todo$Ruido,ruido),
```

```
                    Precision=c(todo$Precision,precision),
                         Modelo=c(todo$Modelo,modelo))
  return(result)
}

# Funcion interna analizarRuidoInterna; Para introducir ruido
    en los datos dados
generarRuido <- function(lista,
    ruido,rep,tipoRuido,descTipoRuido) {

  retornar<-lista
  if (ruido<=0) {
    # No metemos ruido
    write.arff(retornar, paste0(dirParticion,"/4_Test_",
                           descTipoRuido,"_",ruido,"_",rep,".arff"))
    return(retornar)
  }

  # Metemos ruido para los diversos tipos de resultado
  proporcionRuido<-ruido/100

  generarRuidoCelda <- function(retornar, i, j) {
    # Controlamos tipo de valores
    if (is.numeric(retornar[,j])){
      # Metemos ruido en los numericos
      if (esRuidoNumericoRandom) {
        # Calculo error Random como en nominales
        retornar[i,j]<-sample(retornar[,j],1)
      } else {
        # Calculo error gausssiano distribucion normal
        minParam<-summary(lista[,j])["Min."]
        maxParam<-summary(lista[,j])["Max."]
        amplitud<-(maxParam-minParam)/alfa
        # amplitud<-(amplitud*amplitud)
        # print(paste0("error->",retornar[j,i],",", amplitud))
        if (!is.na(retornar[i,j])) {
          # Si no es un valor valido lo dejo como esta
          valorPosteriori<-rnorm(1, mean=retornar[i,j],
              sd=amplitud)
          # print(paste(i,")antes",retornar[j,i], "despues",
              valorPosteriori))
          retornar[i,j]<-valorPosteriori
        }
```

```
    }
  } else if (ruidoEnNominales) {
    # Metemos ruido en nominales
    retornar[i,j]<-sample(retornar[,j],1)
  }
  return(retornar)
}

numFilas<-length(retornar[,1])
numColumnas<-(length(retornar[1,])-1)
if (tipoRuido == 1) {
  # Introducimos Error en todas las filas, en el porcentaje
      de ruido indicado para sus columnas
  filas<-c(1:numFilas)
  # Recorro filas
  for (i in filas) {
    # Obtenemos indices en proporcion de ruido dicha
    indices<-sample(1:numColumnas) # indices desordenados de
        los datos
    mig<-trunc(proporcionRuido*numColumnas) # particion segun
        parametro proporcion
    indices<-sort(indices[1:mig])
    if (FALSE) {
      print(paste("Introduciendo
          ruido",ruido,"iteracion",rep,"en fila",i,".
          ",length(indices),"/",length(retornar[1,])))
      print(paste(sort(indices)))
    }
    # Recorro columnas
    for (j in indices) {
      retornar<-generarRuidoCelda(retornar,i,j)
    }
  }
} else if (tipoRuido == 3) {
  # Introducimos Error en la matriz, en el porcentaje de
      ruido indicado
  tam<-(numFilas*numColumnas) #
  indices<-sample(1:tam) # indices desordenados de los datos
  mig<-trunc(proporcionRuido*tam) # particion segun parametro
      proporcion
  indices<-sort(indices[1:mig])
  if (FALSE) {
    print(paste("Introduciendo
```

```
        ruido",ruido,"iteracion",rep,".
        ",length(indices),"/",tam,".",numFilas,"*",numColumnas))
    print(paste(sort(indices)))
  }
  # Recorro matriz
  for (indice in indices) {
    # Obtengo valor fila y columna para localizar celda
    tmp<-indice/numColumnas
    tmpTrun<-trunc(tmp)
    if (tmp>tmpTrun) {
      i<-(tmpTrun+1)
    } else {
      i<-tmpTrun
    }
    j<-(indice-((i-1)*numColumnas))
    if (FALSE) {
      print(paste("Indice",indice,"=(",i,",",j,")"))
    }
    retornar<-generarRuidoCelda(retornar,i,j)
  }
} else if (tipoRuido == 2) {
  # Introducimos Error en todas las columnas, en el
     porcentaje de ruido indicado para sus filas

  # Recorro columnas
  for (j in paramInterRuido) {
    # Obtenemos indices en proporcion de ruido dicha
    indices<-sample(1:numFilas) # indices desordenados de los
       datos
    mig<-trunc(proporcionRuido*numFilas) # particion segun
       parametro proporcion
    indices<-sort(indices[1:mig])
    if (FALSE) {
      print(paste("Introduciendo
         ruido",ruido,"iteracion",rep,"en columna",i,".
         ",length(indices),"/",length(retornar[,1])))
      print(paste(sort(indices)))
    }

    # Recorro filas
    for (i in indices) {
      retornar<-generarRuidoCelda(retornar,i,j)
    }
```

```
    }
  }

  # Escribimos fichero con ruido y su resultado
  write.arff(retornar, paste0(dirParticion,"/4_Test_",
                    descTipoRuido,"_",ruido,"_",rep,".arff"))

  return(retornar)
}


# Funcion para obtener error de prediccion
obtenerPrecisionConPredict <- function(esClasificacion, model,
    test, prediccionMedia=NULL) {
  # Compare predictions with actual values

  test.data <- test[1:dim(test)[1],1:(dim(test)[2]-1)]
  # test.label <- train[0:dim(test)[1],dim(test)[2]]
  res <- predict(model, newdata = test.data)

  # print(res[1:5])
  lentest <- length(test[,1])
  nattr <- length(test[1,])
  # if (imprimir) {print(length(res))}
  # if (imprimir) {print(length(test[,nattr]))}
  cbind(res,test[nattr]) # cbind joins two matrices

  if (esClasificacion) {
    # Clasificacion
    hits <- 0
    for (i in 1:lentest) {
      if (res[i] == test[i,nattr]) {
        hits <- hits +1
      }
    }
    accuracy <- (hits/ lentest)
    precision1 <- trunc(accuracy*100)

    if (FALSE) {
      evaluacion <- evaluate_Weka_classifier(model, newdata =
          test, na.action = na.pass)
      precision2 <- trunc(evaluacion$details["pctCorrect"])
```

```
      if (precision1 != precision2) {
        print(paste("Las precisiones no coinciden:", precision1,
            precision2))
      }
    }
    precision<-(100-precision1)
    return(precision)
  } else {
    # Regresion
    # Calculamos MSE, Error cuadratico medio
    mse <- 0
    for (i in 1:lentest) {
      mseTmp<-(res[i] - test[i,nattr])
      # print(paste(lentest,")diferencia",mseTmp,":",
                                #    res[i],test[i,nattr]))
      if (mseTmp < 0) {
        mseTmp<-(mseTmp*-1)
      }
      # mseTmp<-mseTmp*mseTmp
      mse <- mse + mseTmp
    }
    mse1 <- trunc((mse / lentest)*100)
    if (is.null(prediccionMedia)) {
      return(mse1)
    } else {
      mse1 <- trunc((mse1 /prediccionMedia)*100)
      return(mse1)
    }
                  }
  }
}

# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# Inicio funcion: analizarRuidoInterna !!!!!!!

# Preparamos directorio temporal
if (file.exists(paste0(dirTmp,fichero))){
  # Borramos Ficheros previos
  unlink(file.path(dirTmp, fichero), recursive = TRUE, force =
      FALSE)
  for (ficheroBorrar in list.files(file.path(dirTmp, fichero)))
      {
    file.remove(paste0(dirTmp,fichero,"/",ficheroBorrar))
```

```
  }
}
# Creamos Directorio
dir.create(file.path(dirTmp, fichero))

# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# Ini Definicion de parametros !!!!!!!!!!!!!!!!
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

# Establecemos si separamos train y test en funcion de
    proporcionTrain
if (proporcionTrain <=0 || proporcionTrain >=1) {
  # ProporcionTrain es mayor que cero y menor que uno, por lo
      que usamos los mismos datos como train y test
  logger("Conf) ProporcionTrain es mayor que cero y menor que
      uno, por lo que usamos los mismos datos como train y
      test",
                                    paste0(dirTmp,fichero))
  separarTestTrain<-FALSE
} else {
  logger(paste("Conf) ProporcionTrain entre train y
      test",proporcionTrain), paste0(dirTmp,fichero))
  separarTestTrain<-TRUE
}

# Generamos ruido con jitter o con ruido gaussiano,
    distribucion normal
esRuidoNumericoRandom <- FALSE # Ruido random como en nominales
    o por distribucion normal
if (esRuidoNumericoRandom) {
  logger("Conf) Introducimos ruido en numericos, como en
      nominales, random de columna", paste0(dirTmp,fichero))
} else {
  logger("Conf) Introducimos ruido en numericos, con
      distribucion normal", paste0(dirTmp,fichero))
}
ruidoEnNominales <- TRUE # Aplicamos ruido a los nominales o no
if (ruidoEnNominales) {
  logger("Conf) Si introducimos ruido en nominales",
      paste0(dirTmp,fichero))
} else {
  logger("Conf) No introducimos ruido en nominales",
      paste0(dirTmp,fichero))
```

```
}

# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
# Fin Definicion de parametros !!!!!!!!!!!
# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

# Leemos datos
logger(paste("Leer Fichero:", fichero), paste0(dirTmp,fichero))
datosPrevios <- read.arff(paste0(dirDataset,fichero))

# Missing values: R<-NA, weka<-?
indices<-c(1:length(datosPrevios[,1]))
indicesCompletos<-complete.cases(datosPrevios)
indicesNoCompletos<-indices[!indicesCompletos]
logger(paste(length(indicesNoCompletos), " filas con valores no
    definidos"), paste0(dirTmp,fichero))
#     natr<-length(datosPrevios[1,])
#     for (indice in indicesNoCompletos) {
#       for (col in c(1:natr)) {
#         if (is.na(datosPrevios[indice,col])) {
#           print(datosPrevios[indice,col])
#           datosPrevios[indice,col]<-NaN
#         }
#       }
#     }

# Asumimos que el estudio se hace en base al ultimo campo
posParamEstudio<-length(datosPrevios[1,])
nomParamEstudio<-names(datosPrevios)[posParamEstudio]

# Solo los validos--- missing values NA, los ignora... [U+FFFD]?
if (is.numeric(datosPrevios[,posParamEstudio])){
  # En regresion quitamos nulos para ksvm que al no ser de
      rweka problema con los na
  datos <- datosPrevios[complete.cases(datosPrevios),]
} else {
  # No quitamos nulos
  datos <- datosPrevios
}

# Escribimos fichero con todos los registros
write.arff(datos, paste0(dirTmp,fichero,"/0_Inicial.arff"))
```

```
# Parametros

# Posicion parametros en los que introducir ruido, todos menos
    el ultimo
paramInterRuido<-c(1:(length(datos)-1))

# Iteraciones realizadas para afinar comportamiento ruido
repeticiones<-numRepeticiones
# Numero de veces que se hace la particion train/test
particiones<-numParticiones

# Valor requerido para calcular la amplitud
alfa<-2

# Num Registros totales
tam<-length(datos[,1]) #

filasRecortadas<-(length(datosPrevios[,1])-tam)
logger(paste(filasRecortadas, " filas no tratadas por valores
    no definidas"), paste0(dirTmp,fichero))

logger(paste(fichero,":", tam, "filas, ", posParamEstudio,
    "columnas"), paste0(dirTmp,fichero))

# es Clasificacion o Regresion
if (is.numeric(datos[,posParamEstudio])){
  # Regresion
  logger("El ultimo parametro es numerico, con lo que sera
      regresion", paste0(dirTmp,fichero))
  esClasificacion<-FALSE
} else {
  # Clasificacion
  logger("El ultimo parametro es nominal, con lo que sera
      clasificacion", paste0(dirTmp,fichero))
  esClasificacion<-TRUE
}

datosGraficaTotal<-list(Filas=c(),Columnas=c(),Total=c())
nombresRuido<-c("Filas","Columnas","Total")

# Realizamos n veces los calculos con distintas particiones
for (particion in seq(1, particiones, by = 1)) {
  datosGraficaParticion<-list(Filas=c(),Columnas=c(),Total=c())
```

```
# Obtenemos train y test
if (!separarTestTrain) {
  # TODOS para train y test
  train <- datos
  test <- datos
} else {
  # Separamos por strata de class
  indices<-c(1:tam) # indices ordenados de los datos
  indiceStrat<-stratificame(esClasificacion, datos,
     posParamEstudio, proporcionTrain)
  train<-datos[indiceStrat,]
  test<-datos[indices[!(indices %in% indiceStrat)],]
}

# Creamos Directorio
dirParticion <- paste0(dirTmp,fichero, "/particion",
    particion)
dir.create(file.path(dirParticion))

write.arff(train, paste0(dirParticion, "/1_Train.arff"))

# Entrenamos (J48, Logistic , IBK y NB) con los datos de
    entrenamiento
logger(paste0("Particion ",particion,"/",particiones),
    paste0(dirTmp,fichero))
logger(paste("Entrenar modelos:", fichero),
    paste0(dirTmp,fichero))

formulaModelo<-as.formula(paste(nomParamEstudio, "~."))

# WOW("J48")
if (esClasificacion) {
  # Clasificacion
  model_J48 <- J48(formulaModelo, data = train, na.action =
     na.pass) # Modelo J48

  model_Logistic <- Logistic(formulaModelo, data = train,
     na.action = na.pass) # Modelo Logistic

  NB <-
     make_Weka_classifier("weka/classifiers/bayes/NaiveBayes")
  model_NB <- NB(formulaModelo, data = train, na.action =
```

```
      na.pass) # Modelo NB

  model_IBk <- IBk(formulaModelo, data = train, na.action =
      na.pass) # Modelo IBk
} else {
  # Regresion

  model_M5P <- M5P(formulaModelo, data = train, na.action =
      na.pass) # Modelo M5P

  model_M5Rules <- M5Rules(formulaModelo, data = train,
      na.action = na.pass) # Modelo M5Rules

  model_LinearReg <- LinearRegression(formulaModelo, data =
      train, na.action = na.pass) # Linear Regression

  model_IBk <- IBk(formulaModelo, data = train, na.action =
      na.pass) # Modelo IBk

  ZeroR <-
      make_Weka_classifier("weka/classifiers/rules/ZeroR")
  model_ZeroR <- ZeroR(formulaModelo, data = train, na.action
      = na.pass) # Modelo ZeroR

  SMOreg <-
      make_Weka_classifier("weka/classifiers/functions/SMOreg")
  model_SMOreg <- SMOreg(formulaModelo, data = train,
      na.action = na.pass) # Modelo SMOreg

  svmmodel <- ksvm(formulaModelo, data= train, na.action =
      na.pass) # SVM
}

# Repetimos n veces para verificar comportamiento ruido
for (rep in seq(1, repeticiones, by = 1)) {

  # Iniciamos datos de la grafica
  datosGrafica<-list(Filas=c(),Columnas=c(),Total=c())

  # Recorremos el ruido desde 0 a 50, en pasos de 5
  for (ruido in saltoError) {

    # 3 tipos de Ruido: Filas, columnas y total
```

```
for (tipoRuido in ruidos) {

  descTipoRuido<-nombresRuido[tipoRuido]

  # Repetimos n veces para afinar los resultados
  if (ruido>0) {
    textoLog <- paste0("Ruido ",descTipoRuido," (", ruido,
        "%) ",rep, "/", repeticiones," ")
  } else if (rep==1) {
    textoLog <- paste0("Sin Ruido ",descTipoRuido," (0%)
        1/1 ")
  }
  logger(textoLog, dirParticion)

  # Si no hay ruido solo realizamos los calculos la
     primera vez
  if (ruido>0 || rep==1) {

    # Generamos ruido
    testError<-generarRuido(test,ruido,rep,tipoRuido,descTipoRuido)

    if (esClasificacion) {
      # Clasificacion

      # Obtenemos precision J48
      precision_J48 <-
          obtenerPrecisionConPredict(esClasificacion,
          model_J48, testError)
      datosGrafica[[tipoRuido]]<-rellenarGrafica(ruido,
          precision_J48, "J48", datosGrafica[[tipoRuido]],
          testError, textoLog)

      # Obtenemos precision Logistic
      precision_Logistic <-
          obtenerPrecisionConPredict(esClasificacion,
          model_Logistic, testError)
      datosGrafica[[tipoRuido]]<-rellenarGrafica(ruido,
          precision_Logistic, "Logistic",
          datosGrafica[[tipoRuido]], testError, textoLog)

      # Obtenemos precision NB
      precision_NB <-
          obtenerPrecisionConPredict(esClasificacion,
```

```
                    model_NB, testError)
                datosGrafica[[tipoRuido]]<-rellenarGrafica(ruido,
                    precision_NB, "NB", datosGrafica[[tipoRuido]],
                    testError, textoLog)

                # Obtenemos precision IBk
                precision_IBk <-
                    obtenerPrecisionConPredict(esClasificacion,
                    model_IBk, testError)
                datosGrafica[[tipoRuido]]<-rellenarGrafica(ruido,
                    precision_IBk, "IBk", datosGrafica[[tipoRuido]],
                    testError, textoLog)
              } else {
                # Regresion

                # Obtenemos ZeroR
#                 prediccionMedia <-
    obtenerPrediccionConPredict(esClasificacion, model_ZeroR,
    testError)
                precision_ZeroR <-
                    obtenerPrecisionConPredict(esClasificacion,
                    model_ZeroR, testError)
                datosGrafica[[tipoRuido]]<-rellenarGrafica(ruido,
                    100, "ZeroR", datosGrafica[[tipoRuido]],
                    testError, textoLog)

                # Obtenemos MSE M5P
                precision_M5P <-
                    obtenerPrecisionConPredict(esClasificacion,
                    model_M5P, testError, precision_ZeroR)
                datosGrafica[[tipoRuido]]<-rellenarGrafica(ruido,
                    precision_M5P, "M5P", datosGrafica[[tipoRuido]],
                    testError, textoLog)

                # Obtenemos MSE M5Rules
                precision_M5Rules <-
                    obtenerPrecisionConPredict(esClasificacion,
                    model_M5Rules, testError, precision_ZeroR)
                datosGrafica[[tipoRuido]]<-rellenarGrafica(ruido,
                    precision_M5Rules, "M5Rules",
                    datosGrafica[[tipoRuido]], testError, textoLog)

                # Obtenemos MSE IBk
```

```
        precision_LinearReg <-
            obtenerPrecisionConPredict(esClasificacion,
            model_LinearReg, testError, precision_ZeroR)
        datosGrafica[[tipoRuido]]<-rellenarGrafica(ruido,
            precision_LinearReg, "Linear Regression",
            datosGrafica[[tipoRuido]], testError, textoLog)

        # Obtenemos MSE IBk
        precision_IBk <-
            obtenerPrecisionConPredict(esClasificacion,
            model_IBk, testError, precision_ZeroR)
        datosGrafica[[tipoRuido]]<-rellenarGrafica(ruido,
            precision_IBk, "IBk", datosGrafica[[tipoRuido]],
            testError, textoLog)

        # Obtenemos MSE svmmodel
        precision_SVM <-
            obtenerPrecisionConPredict(esClasificacion,
            svmmodel, testError, precision_ZeroR)
        datosGrafica[[tipoRuido]]<-rellenarGrafica(ruido,
            precision_SVM, "SVM", datosGrafica[[tipoRuido]],
            testError, textoLog)

        # Obtenemos SMOreg
        precision_SMOreg <-
            obtenerPrecisionConPredict(esClasificacion,
            model_SMOreg, testError, precision_ZeroR)
        datosGrafica[[tipoRuido]]<-rellenarGrafica(ruido,
            precision_SMOreg, "SMOreg",
            datosGrafica[[tipoRuido]], testError, textoLog)
      }
    }
  }
}
if (rep == 10) {
  # Imprimimos grafica iteracion
  # imprimirGrafica(esClasificacion,
      data.frame(datosGrafica[[tipoRuido]]), fichero,
      dirParticion, paste0(descTipoRuido,"_",rep))
}

if (rep == repeticiones) {
  # La ultima grafica de la iteracion sera la de la
```

```
        particion
    # imprimirGrafica(esClasificacion,
        data.frame(datosGrafica[[tipoRuido]]), fichero,
        paste0(dirTmp,fichero),
        paste0(descTipoRuido,"_",particion))
  }

  # Al acabar cada repeticion almacenamos los resultados
  for (tipoRuido in ruidos) {
    matriz<-datosGrafica[[tipoRuido]]
    matriz<-c(matriz$Modelo,matriz$Ruido,matriz$Precision)
    matriz <- matrix(matriz, ncol=3, byrow=FALSE)
    write.matrix(matriz, file = paste(dirParticion,
      paste0("5_Resultado_",nombresRuido[tipoRuido],"_",
      particion,"_",rep,".csv"), sep="/"),sep=";")

    # Insertamos los datos de la grafica de la particion
        actual en la general
    datosGraficaParticion[[tipoRuido]] <- list(
      Ruido=c(datosGraficaParticion[[tipoRuido]]$Ruido,
      datosGrafica[[tipoRuido]]$Ruido),
      Precision=c(datosGraficaParticion[[tipoRuido]]$Precision,
      datosGrafica[[tipoRuido]]$Precision),
      Modelo=c(datosGraficaParticion[[tipoRuido]]$Modelo,
      datosGrafica[[tipoRuido]]$Modelo))
  }
}
# Imprimimos resultados de cada particion, media de las
    repeticiones
for (tipoRuido in ruidos) {
  datosGrafica<-list(Filas=c(),Columnas=c(),Total=c())
  matriz<-datosGraficaParticion[[tipoRuido]]
  matriz<-c(matriz$Modelo,matriz$Ruido,matriz$Precision)
  matriz <- matrix(matriz, ncol=3, byrow=FALSE)
  if (FALSE) {
    # print(matriz)
    matriz<-data.frame(matriz)
    ruidosModelo<-unique(matriz[,2])
    for (ruidoModelo in ruidosModelo) {
      # print(paste("Ruido",ruidoModelo))
      filasRuido<-matriz[matriz[,2]==ruidoModelo,]
      modelos<-unique(filasRuido[,1])
      for (modelo in modelos) {
```

```
            # print(paste("Modelo",modelo))
            precisionesModelo<-filasRuido[filasRuido[,1]==modelo,]
            precisionesModelo<-as.numeric(as.vector(precisionesModelo[,3]))
            # print(precisionesModelo)
            mediaPrecisionesModelo <-
                as.numeric(sum(precisionesModelo)/length(precisionesModelo))
            # mediaPrecisionesModelo <- precisionesModelo[1]
            datosGrafica <-
                list(Ruido=c(datosGrafica$Ruido,as.numeric(ruidoModelo)),
              Precision=c(datosGrafica$Precision,mediaPrecisionesModelo),
              Modelo=c(datosGrafica$Modelo,modelo))
            # print(datosGraficaTotal)
          }
        }
        matriz<-c(datosGrafica$Modelo,datosGrafica$Ruido,datosGrafica$Precision)
        matriz <- matrix(matriz, ncol=3, byrow=FALSE)
      } else {
        datosGrafica<-datosGraficaParticion[[tipoRuido]]
      }
      write.matrix(matriz, file = paste(paste0(dirTmp,fichero),
        paste0("5_Resultado_",nombresRuido[tipoRuido],"_",particion,".csv"),
            sep="/"),
        sep=";")
      # Insertamos los datos de la grafica de la particion actual
          en la general
      datosGraficaTotal[[tipoRuido]] <-
          list(Ruido=c(datosGraficaTotal[[tipoRuido]]$Ruido,
        datosGrafica$Ruido),
        Precision=c(datosGraficaTotal[[tipoRuido]]$Precision,
        datosGrafica$Precision),
        Modelo=c(datosGraficaTotal[[tipoRuido]]$Modelo,
        datosGrafica$Modelo))
    }
  }
  # Imprimimos grafica con los datos de todas las particiones
  for (tipoRuido in ruidos) {
    imprimirGrafica(esClasificacion,
        data.frame(datosGraficaTotal[[tipoRuido]]),
                    fichero, paste0(dirTmp,fichero),
                        paste0(nombresRuido[tipoRuido],"_",fichero))
  }
}
```

```
# Funcion para calcular y pintar la grafica
imprimirGrafica <- function(esClasificacion, datosGrafica,
    fichero, dirParticion, id) {
  logger("Imprimir Grafica Resultado", dirParticion)
  # windows(width = 15, height = 10, pointsize = 12)
  # CairoPNG por antialiasing, se puede usar el png y ya esta
  CairoPNG(file=paste0(dirParticion,"/2_Grafica_",id,".jpg"),
     width=1280,height=1024)
  if (is.null(datosGrafica)) {
    # No se ha podido realizar el analisis de ruido, mostramos
       una grafica aleatoria, por imprimir algo
    graf <- xyplot(lat ~ long, data = quakes,
                   main = paste("No se ha podido realizar el
                       Analisis de ruido para:",fichero)
    )
    # Imprimimos grafica
    print(graf)
  } else {
    # Pintamos grafica con los resultados ?panel.xyplot
    xMin<-min(datosGrafica$Ruido)
    xMax<-max(datosGrafica$Ruido)
    yMin<-min(datosGrafica$Precision)
    yMax<-max(datosGrafica$Precision)

    # Buscar colores: colors()[grep("green",colors())]
    if (esClasificacion) {
      desColumnaY<-"Error Precision"
    } else {
      desColumnaY<-"MSE"
    }

    # Pintamos primero una sin bigotes
    graf1 <- xyplot(
      Precision ~ Ruido, data = datosGrafica, groups=Modelo,
      xlab="Ruido",ylab=desColumnaY,
      xlim=c(xMin-1,xMax+1),ylim=c(yMin-5,yMax+5),
      type=c('g', 'a'),
      auto.key = list(border = TRUE, lines = TRUE),
      main = paste("Analisis de ruido para:",fichero)
    )
    # Pintamos luego la de con bigotes
    graf2 <- xyplot(
      Precision ~ Ruido|Modelo, data = datosGrafica,
```

```
            groups=Modelo,
          xlab="Ruido",ylab=desColumnaY,
          xlim=c(xMin-1,xMax+1),ylim=c(yMin-1,yMax+1),
          ,main = paste("Analisis de ruido para:",fichero)
          ,par.settings = list(
            superpose.line=list(col="greenyellow")
            ,reference.line = list(col="grey", lty ="dotted")
            ,plot.symbol=list(col="blue")
            ,box.rectangle = list(col="hotpink")
            ,box.dot = list(col="blue")
            ,box.umbrella=list(col="black")
          )
          ,panel=function(x, y, subscripts){
            panel.xyplot(x, y, type=c('g', 'a'),col="greenyellow")
            panel.bwplot(x,y,horizontal = FALSE, pch='|')
          }, as.table=T, subscripts=T
        )
        grid.arrange(graf1,graf2,nrow=2)
      }
      dev.off()
      # savePlot(filename=paste0(dirTmp,fichero,"/2_Grafica.jpg"),
      # type="jpg",device=dev.cur())
      # graphics.off()
    }

    # Inicio funcion: analizarRuido !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

    # Establecemos directorio por defecto
    setwd(dirDataset)

    # Obtenemos e imprimos Graficas, maximo 4 por pantalla
    numGraficas<-length(nombreDatos)

    for (i in seq(from = 1, to = numGraficas, by = 1)) {
      analizarRuidoInterna(nombreDatos[i])
    }
}
# datosGrafica<-
# Lanzamos Script
print(paste("Inicio Script:", Sys.time()))
print("_____________________________________________________________________")

# Regresion
```

```
nombresFiles<-c()
# REG_01_cholesterol.arff - 22KB
# nombresFiles<-c(nombresFiles,"REG_01_cholesterol.arff")
# REG_02_bodyfat.arff - 24KB
# nombresFiles<-c(nombresFiles,"REG_02_bodyfat.arff")
# REG_03_wisconsin.arff - 49KB
# nombresFiles<-c(nombresFiles,"REG_03_wisconsin.arff")
# REG_04_stock.arff - 56KB
nombresFiles<-c(nombresFiles,"REG_04_stock.arff")
# REG_05_meta.arff - 72KB
nombresFiles<-c(nombresFiles,"REG_05_meta.arff")
# REG_06_triazines.arff - 81KB
nombresFiles<-c(nombresFiles,"REG_06_triazines.arff")
# REG_07_abalone.arff - 194KB
# nombresFiles<-c(nombresFiles,"REG_07_abalone.arff")
# REG_08_cpu_act.arff - 1.034KB
# nombresFiles<-c(nombresFiles,"REG_08_cpu_act.arff")
# REG_09_cal_housing.arff - 2.026KB
# nombresFiles<-c(nombresFiles,"REG_09_cal_housing.arff")
# REG_10_house_8L.arff - 2.324KB
# nombresFiles<-c(nombresFiles,"REG_10_house_8L.arff")
analizarRuido(nombreDatos=nombresFiles,numRepeticiones=5,
    numParticiones=4,
    saltoError=c(0,5,10,15,20,30,40,50),ruidos=c(1,2,3))
# analizarRuido(nombreDatos=nombresFiles,numRepeticiones=1,
    numParticiones=1, saltoError=c(0,50),ruidos=c(3))

# Clasificacion
nombresFiles<-c()
# CLAS_01_iris.arff - 8KB
# nombresFiles<-c(nombresFiles,"CLAS_01_iris.arff")
# CLAS_02_white-clover.arff - 15KB
# nombresFiles<-c(nombresFiles,"CLAS_02_white-clover.arff")
# CLAS_03_hepatitis.arff - 17KB
# nombresFiles<-c(nombresFiles,"CLAS_03_hepatitis.arff")
# CLAS_04_breast-cancer.arff - 29KB
# nombresFiles<-c(nombresFiles,"CLAS_04_breast-cancer.arff")
# CLAS_05_vehicle.arff - 63KB
# nombresFiles<-c(nombresFiles,"CLAS_05_vehicle.arff")
# CLAS_06_credit-g.arff - 159KB
# nombresFiles<-c(nombresFiles,"CLAS_06_credit-g.arff")
# CLAS_07_splice.arff - 512KB
# nombresFiles<-c(nombresFiles,"CLAS_07_splice.arff")
```

```
# CLAS_08_letter.arff - 704KB
# nombresFiles<-c(nombresFiles,"CLAS_08_letter.arff")
# CLAS_09_waveform-5000.arff - 1.055KB
nombresFiles<-c(nombresFiles,"CLAS_09_waveform-5000.arff")
# CLAS_10_mfeat-fourier.arff - 1.642KB
nombresFiles<-c(nombresFiles,"CLAS_10_mfeat-fourier.arff")
# nombresFiles<-c("CLAS_02_white-clover.arff")
#
    analizarRuido(nombreDatos=c("CLAS_waveform-5000.arff"),numRepeticiones=2,
    numParticiones=5, saltoError=c(0,50),ruidos=c(1,2,3))
# analizarRuido(nombreDatos=nombresFiles,numRepeticiones=10,
    numParticiones=5,
    saltoError=c(0,5,10,15,20,30,40,50),ruidos=c(1,2,3))
# analizarRuido(nombreDatos=nombresFiles,numRepeticiones=5,
    numParticiones=4,
    saltoError=c(0,5,10,15,20,30,40,50),ruidos=c(3))
print("_____________________________________________________________________")
print(paste("Fin Script:", Sys.time()))
```

Codigo fuente para procesar los datasets artificiales, introduciendo ruido y generando graficas:

```
# Funcion global
analizarRuido <- function(quemodelo) {
  library("RWeka")
  library("Cairo")
  library("lattice")
  require("gridExtra")

  esClasificacion <- TRUE
  if (TRUE) {
    esClasificacion <- FALSE
    nomFile <- "otro"
    nomTrain <- "100"
  } else if (FALSE) {
    esClasificacion <- FALSE
    nomFile <- "atan"
    nomTrain <- "100"
  } else if (FALSE) {
    esClasificacion <- FALSE
```

```
  nomFile <- "seno"
  nomTrain <- "100"
} else if (FALSE) {
  esClasificacion <- FALSE
  nomFile <- "ZZ_REGRESION"
  nomTrain <- "100"
} else if (FALSE) {
  nomFile <- "polynomial"
  nomTrain <- "1000"
} else if (FALSE) {
  nomFile <- "parity"
  nomTrain <- "1000"
} else if (FALSE) {
  nomFile <- "checkerboard"
  nomTrain <- "1000"
} else if (FALSE) {
  nomFile <- "spirals"
  nomTrain <- "172"
}

if (esClasificacion) {
  dirDataset<-"C:/Users/Eclipse/Documents/Master/Tesis/SVN/python/test_patterns/"
} else {
  dirDataset<-"C:/Users/Eclipse/Documents/Master/Tesis/SVN/python/test_regresion/"
}
dirTmp<-"C:/Users/Eclipse/Documents/Master/Tesis/SVN/python/results/"

fileTest<-paste0(nomFile,"/patterns.exhaustive.arff")
fileTrain<-paste0(nomFile,"/",nomTrain,"/patterns.arff")

esRuidoNumericoRandom<-TRUE
alfa<-2

obtenerPrecisionConPredict <- function(esClasificacion,model,
    test, prediccionMedia=NULL) {
  # Compare predictions with actual values
  test.data <- test#test[1:dim(test)[1],1:(dim(test)[2]-1)]
  # test.label <- train[0:dim(test)[1],dim(test)[2]]
  res <- predict(model, newdata = test.data)
  # print(res[1:5])
  lentest <- length(test[,1])
  nattr <- length(test[1,])
  cbind(res,test[nattr]) # cbind joins two matrices
```

```
  if (esClasificacion) {
    # Clasificacion
    hits <- 0
    for (i in 1:lentest) {
      if (res[i] == test[i,nattr]) {
        hits <- hits +1
      }
    }
    accuracy <- (hits/ lentest)
    precision <- trunc(accuracy*100)
    # precision<-(100-precision1)
    return(precision)
  } else {
    # Regresion
    # Calculamos MSE, Error cuadratico medio
    mse <- 0
    for (i in 1:lentest) {
      # print(paste(res[i]," - ",test[i,nattr]))
      mseTmp<-(res[i] - test[i,nattr])
      if (mseTmp < 0) {
        mseTmp<-(mseTmp*-1)
      }
      mse <- mse + mseTmp
    }
    mse1 <- trunc((mse / lentest)*100)

    if (is.null(prediccionMedia)) {
      return(mse1)
    } else {
      mse1 <- trunc((mse1 /prediccionMedia)*100)
      return(mse1)
    }
  }
}

# Funcion interna analizarRuidoInterna; Para introducir ruido en
    los datos dados
generarRuido <- function(lista, ruido,tipoRuido) {

  retornar<-lista
  if (ruido<=0) {
    # No metemos ruido
```

```
  return(retornar)
}

# Metemos ruido para los diversos tipos de resultado
proporcionRuido<-ruido/100

generarRuidoCelda <- function(retornar, i, j) {
  # Controlamos tipo de valores
  if (is.numeric(retornar[,j])){
    # Metemos ruido en los numericos
    if (esRuidoNumericoRandom) {
      # Calculo error Random como en nominales
      retornar[i,j]<-sample(retornar[,j],1)
    } else {
      # Calculo error gausssiano distribucion normal
      minParam<-summary(lista[,j])["Min."]
      maxParam<-summary(lista[,j])["Max."]
      amplitud<-(maxParam-minParam)/alfa
      # amplitud<-(amplitud*amplitud)
      # print(paste0("error->",retornar[j,i],",", amplitud))
      if (!is.na(retornar[i,j])) {
        # Si no es un valor valido lo dejo como esta
        valorPosteriori<-rnorm(1, mean=retornar[i,j],
            sd=amplitud)
        # print(paste(i,")antes",retornar[j,i], "despues",
            valorPosteriori))
        retornar[i,j]<-valorPosteriori
      }
    }
  } else if (ruidoEnNominales) {
    # Metemos ruido en nominales
    retornar[i,j]<-sample(retornar[,j],1)
  }
  return(retornar)
}

numFilas<-length(retornar[,1])
numColumnas<-(length(retornar[1,])-1)
if (tipoRuido == 1) {
  # Introducimos Error en todas las filas, en el porcentaje de
      ruido indicado para sus columnas
  filas<-c(1:numFilas)
  # Recorro filas
```

```
  for (i in filas) {
    # Obtenemos indices en proporcion de ruido dicha
    indices<-sample(1:numColumnas) # indices desordenados de
       los datos
    mig<-trunc(proporcionRuido*numColumnas) # particion segun
       parametro proporcion
    indices<-sort(indices[1:mig])
    if (FALSE) {
      print(paste("Introduciendo ruido",ruido,"en fila",i,".
         ",length(indices),"/",length(retornar[1,])))
      print(paste(sort(indices)))
    }
    # Recorro columnas
    for (j in indices) {
      retornar<-generarRuidoCelda(retornar,i,j)
    }
  }
} else if (tipoRuido == 3) {
  # Introducimos Error en la matriz, en el porcentaje de ruido
     indicado
  tam<-(numFilas*numColumnas) #
  indices<-sample(1:tam) # indices desordenados de los datos
  mig<-trunc(proporcionRuido*tam) # particion segun parametro
     proporcion
  indices<-sort(indices[1:mig])
  if (FALSE) {
    print(paste("Introduciendo ruido",ruido,".
       ",length(indices),"/",tam,".",numFilas,"*",numColumnas))
    print(paste(sort(indices)))
  }
  # Recorro matriz
  for (indice in indices) {
    # Obtengo valor fila y columna para localizar celda
    tmp<-indice/numColumnas
    tmpTrun<-trunc(tmp)
    if (tmp>tmpTrun) {
      i<-(tmpTrun+1)
    } else {
      i<-tmpTrun
    }
    j<-(indice-((i-1)*numColumnas))
    if (FALSE) {
      print(paste("Indice",indice,"=(",i,",",j,")"))
```

```
      }
      retornar<-generarRuidoCelda(retornar,i,j)
    }
  } else if (tipoRuido == 2) {
    # Introducimos Error en todas las columnas, en el porcentaje
      de ruido indicado para sus filas

    # Recorro columnas
    for (j in paramInterRuido) {
      # Obtenemos indices en proporcion de ruido dicha
      indices<-sample(1:numFilas) # indices desordenados de los
         datos
      mig<-trunc(proporcionRuido*numFilas) # particion segun
         parametro proporcion
      indices<-sort(indices[1:mig])
      if (FALSE) {
        print(paste("Introduciendo ruido",ruido,"en columna",i,".
           ",length(indices),"/",length(retornar[,1])))
        print(paste(sort(indices)))
      }

      # Recorro filas
      for (i in indices) {
        retornar<-generarRuidoCelda(retornar,i,j)
      }
    }
  }

  return(retornar)
}

# Funcion para calcular y pintar la grafica
imprimirGrafica <- function(esClasificacion,datos1, datos2,
    datos3, datos4, datos5, datos6, nomGrafica) {

  imprimirGraficaInterna1 <- function(datos, titulo,
      cambiarColores = FALSE) {
    if (cambiarColores) {
      colors=c("green","grey")
    } else {
      colors=c("blue","red")
    }
    return(xyplot(y ~ x, data = datos,
```

```
                groups=class, type=c('p'),
                col=colors,
                #                  auto.key = list(border =
                    TRUE, lines = TRUE),
                #                  xlim=c(0,4),ylim=c(0,4),
                #                  panel=function(x, y,
                    subscripts){
                #                    panel.xyplot(x,
                    y,col=c("green","red"))
                #                    panel.lines(y ~ x, col =
                    "grey", lwd = 1)
                #                  },
                main = paste(titulo)
  ))
}

imprimirGraficaInterna2 <- function(datos, titulo) {
  colors <- colorRampPalette(c('green', 'grey'))(256)
  return(levelplot(class~x*y, datos,col.regions=colors,main =
      titulo,colorkey=FALSE))
}

imprimirGraficaInterna3 <- function(datos, titulo,
    cambiarColores = 0) {
  if (cambiarColores == 0) {
    colors=c("brown")
  } else if (cambiarColores == 1) {
    colors=c("blue")
  } else if (cambiarColores == 2) {
    colors=c("red")
  }
  return(xyplot(class ~ x, data = datos,
                type=c('p'),
                col=colors,
                #                  auto.key = list(border =
                    TRUE, lines = TRUE),
                #                  xlim=c(0,4),ylim=c(0,4),
                #                  panel=function(x, y,
                    subscripts){
                #                    panel.xyplot(x,
                    y,col=c("green","red"))
                #                    panel.lines(y ~ x, col =
                    "grey", lwd = 1)
```

```
                #                   },
                main = paste(titulo)
  ))
}

imprimirGraficaInterna4 <- function(datos, titulo,
    cambiarColores = FALSE) {
  if (cambiarColores) {
    colors <- colorRampPalette(c('green', 'blue'))(256)
  } else {
    colors <- colorRampPalette(c('grey', 'red'))(256)
  }
  return(levelplot(class~x*y, datos,col.regions=colors,main =
      titulo,colorkey=TRUE))
}

CairoPNG(file=paste0(dirTmp,"/2_Grafica",nomGrafica,".jpg"),
   width=1280,height=1024)
# No se ha podido realizar el analisis de ruido, mostramos una
    grafica aleatoria, por imprimir algo
if (esClasificacion) {
  graf1 <- imprimirGraficaInterna1(datos1$Datos, datos1$Titulo)
  graf2 <- imprimirGraficaInterna1(datos2$Datos, datos2$Titulo)
  graf3 <- imprimirGraficaInterna1(datos3$Datos, datos3$Titulo,
      TRUE)
  graf4 <- imprimirGraficaInterna1(datos4$Datos, datos4$Titulo)
  graf5 <- imprimirGraficaInterna1(datos5$Datos, datos5$Titulo)
  graf6 <- imprimirGraficaInterna1(datos6$Datos, datos6$Titulo)
} else {
  graf1 <- imprimirGraficaInterna3(datos1$Datos, datos1$Titulo)
  graf2 <- imprimirGraficaInterna3(datos2$Datos, datos2$Titulo)
  graf3 <- imprimirGraficaInterna3(datos3$Datos, datos3$Titulo,
      1)
  graf4 <- imprimirGraficaInterna3(datos4$Datos, datos4$Titulo,
      2)
  graf5 <- imprimirGraficaInterna3(datos5$Datos, datos5$Titulo,
      2)
  graf6 <- imprimirGraficaInterna3(datos6$Datos, datos6$Titulo,
      2)
}
grid.arrange(graf1,graf2,graf3,graf4,graf5,graf6,nrow=2)
# Imprimimos grafica
#     print(graf)
```

```
  dev.off()
}

train <- read.arff(paste0(dirDataset,fileTrain))
datosGrafica1<-list(Datos=train,Titulo="Entrenamiento")

test <- read.arff(paste0(dirDataset,fileTest))
datosGrafica2<-list(Datos=test,Titulo="Test")


formulaModelo<-as.formula("class~.")
if (esClasificacion) {
  precision_ZeroR <- NULL
  if (FALSE) {
    nomModelo<-"J48"
    model <- J48(formulaModelo, data = train, na.action =
        na.pass) # Modelo J48
  } else if (TRUE) {
    nomModelo<-"NB"
    NB <-
        make_Weka_classifier("weka/classifiers/bayes/NaiveBayes")
    model <- NB(formulaModelo, data = train, na.action = na.pass)
        # Modelo NB
  } else if (FALSE) {
    nomModelo<-"Logistic"
    model <- Logistic(formulaModelo, data = train, na.action =
        na.pass) # Modelo Logistic
  } else if (FALSE) {
    nomModelo<-"IBk"
    model <- IBk(formulaModelo, data = train, na.action =
        na.pass) # Modelo IBk
  }
} else {
    ZeroR <- make_Weka_classifier("weka/classifiers/rules/ZeroR")
    model_ZeroR <- ZeroR(formulaModelo, data = train, na.action =
        na.pass) # Modelo ZeroR
  if (quemodelo == 1) {
    nomModelo<-"IBk"
    model <- IBk(formulaModelo, data = train, na.action =
        na.pass) # Modelo IBk
  } else if (quemodelo == 2) {
    nomModelo<-"M5P"
    model <- M5P(formulaModelo, data = train, na.action =
```

```
        na.pass) # Modelo M5P
  } else if (quemodelo == 3) {
    nomModelo<-"ZeroR"
    ZeroR <- make_Weka_classifier("weka/classifiers/rules/ZeroR")
    model <- ZeroR(formulaModelo, data = train, na.action =
        na.pass) # Modelo ZeroR
  } else if (quemodelo == 4) {
    nomModelo<-"SMOreg"
    SMOreg <-
        make_Weka_classifier("weka/classifiers/functions/SMOreg")
    model <- SMOreg(formulaModelo, data = train, na.action =
        na.pass) # Modelo SMOreg
  } else if (quemodelo == 5) {
    nomModelo<-"M5Rules"
    model <- M5Rules(formulaModelo, data = train, na.action =
        na.pass) # Modelo M5Rules
  } else if (quemodelo == 6) {
    nomModelo<-"LinearRegression"
    model <- LinearRegression(formulaModelo, data = train,
        na.action = na.pass) # Linear Regression
  } else if (quemodelo == 7) {
    nomModelo<-"ksvm"
    library("kernlab")
    model <- ksvm(formulaModelo, data= train, na.action =
        na.pass) # SVM
  }
}

ruido<-0
if (!esClasificacion) {
  precision_ZeroR <- obtenerPrecisionConPredict(esClasificacion,
      model_ZeroR, test)
}
prec<-obtenerPrecisionConPredict(esClasificacion,model,
    test,precision_ZeroR)
res <- predict(model, newdata = test)
testError<-test
if (esClasificacion) {
  testError[,3]<-res
}else {
  testError[,2]<-res
}
datosGrafica3<-list(Datos=testError,Titulo=paste0(nomModelo,"
```

```
    Ruido ",ruido," precision (",prec,")"))

  ruido<-10
  testError <- generarRuido(test, ruido, 3)
  if (!esClasificacion) {
    precision_ZeroR <- obtenerPrecisionConPredict(esClasificacion,
        model_ZeroR, testError)
  }
  prec<-obtenerPrecisionConPredict(esClasificacion,model,
      testError,precision_ZeroR)
  datosGrafica4<-list(Datos=testError,Titulo=paste0(nomModelo,"
      Ruido ",ruido," precision (",prec,")"))

  ruido<-20
  testError <- generarRuido(test, ruido, 3)
  if (!esClasificacion) {
    precision_ZeroR <- obtenerPrecisionConPredict(esClasificacion,
        model_ZeroR, testError)
  }
  prec<-obtenerPrecisionConPredict(esClasificacion,model,
      testError,precision_ZeroR)
  datosGrafica5<-list(Datos=testError,Titulo=paste0(nomModelo,"
      Ruido ",ruido," precision (",prec,")"))

  ruido<-30
  testError <- generarRuido(test, ruido, 3)
  if (!esClasificacion) {
    precision_ZeroR <- obtenerPrecisionConPredict(esClasificacion,
        model_ZeroR, testError)
  }
  prec<-obtenerPrecisionConPredict(esClasificacion,model,
      testError,precision_ZeroR)
  datosGrafica6<-list(Datos=testError,Titulo=paste0(nomModelo,"
      Ruido ",ruido," precision (",prec,")"))

  imprimirGrafica(esClasificacion,datosGrafica1,datosGrafica2,datosGrafica3,
    datosGrafica4,datosGrafica5,datosGrafica6,
    paste0("_",nomFile,"_",nomModelo))
  print("FIN")
}

analizarRuido(1)
analizarRuido(2)
```

```
analizarRuido(3)
analizarRuido(4)
analizarRuido(5)
analizarRuido(6)
analizarRuido(7)
```

Codigo fuente con funciones extra para generar tablas y datasets artificiales:

```
dirDataset<-"C:/Users/Eclipse/Documents/Master/Tesis/SVN/tmp/"
setwd(dirDataset)

# Generamos fichero tabla con resultados intermedios experimentos
estadisticas <- function(nomFile="CLAS_01_iris.arff",
    numParticiones=5, tiposError=c("Total")) {

  for (tipoError in tiposError) {
    estadisticasTotales<-c()
    for (particion in c(1:numParticiones)) {
      datos <- data.frame(read.csv(paste(paste0(dirDataset,
          nomFile,"/"),
       paste0("5_Resultado_",tipoError,"_",particion,".csv"),
           sep="/"), sep=";",header=FALSE))
      ruidos<-sort(unique(datos[,2]),decreasing=FALSE)
      estadisticasParticion<-c("Modelo")
      for (ruido in ruidos) {
        estadisticasParticion<-c(estadisticasParticion,paste("Ruido",ruido))
      }

      modelos<-unique(datos[,1])
      for (modelo in modelos) {
        # print(paste("Modelo",modelo))
        filasModelo<-datos[datos[,1]==modelo,]
        precisiones<-c()
        for (ruidoModelo in ruidos) {
          # print(paste("Ruido",ruidoModelo))
          precisionesModelo<-filasModelo[filasModelo[,2]==ruidoModelo,]
          precisionesModelo<-as.numeric(as.vector(precisionesModelo[,3]))
          mediaPrecisionesModelo <-
              as.numeric(sum(precisionesModelo)/length(precisionesModelo))
```

```
      precisiones<-c(precisiones, mediaPrecisionesModelo)
    }
    estadisticasParticion<-c(estadisticasParticion,modelo,precisiones)
  }
  estadisticasParticion <- matrix(estadisticasParticion,
     ncol=9, byrow=TRUE)
  # print(paste("Particion",particion))
  # print(estadisticasParticion)
  if (particion == 1) {
    # Son iguales primera iteracion
    estadisticasTotales<-estadisticasParticion
  } else {
    # Sumamos particion
    for (i in c(2:(1+length(modelos)))) {
      for (j in c(2:(1+length(ruidos)))) {
        estadisticasTotales[i,j]=(as.numeric(estadisticasTotales[i,j])
          +as.numeric(estadisticasParticion[i,j]))
      }
    }
  }
}
if (numParticiones > 1) {
  for (i in c(2:(1+length(modelos)))) {
    for (j in c(2:(1+length(ruidos)))) {
      estadisticasTotales[i,j]=trunc(as.numeric(estadisticasTotales[i,j])
        /numParticiones)
    }
  }
}

# Conclusiones
fila<-(2+length(modelos))
# 2 espacios
estadisticasTotales<-rbind(estadisticasTotales,rep("",9))
estadisticasTotales<-rbind(estadisticasTotales,rep("",9))
fila<-(2+fila)

# Quitamos ruido 0 y se lo restamos a las demas
for (i in c(0:length(modelos))) {
  # A[U+FFFD]adimos fila
  if (i==1) {
    # No incluimos ZeroR
    estadisticasTotales<-rbind(estadisticasTotales,rep("",9))
```

```
  } else {
    estadisticasTotales<-rbind(estadisticasTotales,rep("",9))
    estadisticasTotales[fila+i,1]=estadisticasTotales[i+1,1]
    for (j in c(2:(length(ruidos)))) {
      #
        print(paste("fila",fila+i,",",j,","))#,estadisticasTotales[(i-1),j]))
      if (i == 0) {
        # Titulo
        estadisticasTotales[fila+i,j]=estadisticasTotales[i+1,j+1]
      } else {
        # Calculo
        estadisticasTotales[fila+i,j]=
          trunc(as.numeric(estadisticasTotales[i+1,j+1])
          - as.numeric(estadisticasTotales[i+1,2]))
      }
    }
  }
}

# 2 espacios
fila<-(length(estadisticasTotales[,1])+1)
estadisticasTotales<-rbind(estadisticasTotales,rep("",9))
estadisticasTotales<-rbind(estadisticasTotales,rep("",9))
fila<-(2+fila)

# Obtenemos puntuacion segun error
for (i in c(0:length(modelos))) {
  if (i==1) {
    # No incluimos ZeroR
    estadisticasTotales<-rbind(estadisticasTotales,rep("",9))
  } else {
    # A[U+FFFD]adimos fila
    estadisticasTotales<-rbind(estadisticasTotales,rep("",9))
    estadisticasTotales[fila+i,1]=estadisticasTotales[i+1,1]
    for (j in c(2:(length(ruidos)))) {
      # print(paste("fila",fila+i,",",j,","))#,
      # estadisticasTotales[(i-1),j]))
      if (i == 0) {
        # Titulo
        estadisticasTotales[fila+i,j]=estadisticasTotales[i+1,j+1]
      } else {
        # Calculo
        posAnterior <- 5+length(modelos)
```

```
            intervalo <- sort(unique(as.array(as.numeric(
            estadisticasTotales[posAnterior:
              (posAnterior-1+(length(modelos))),j]))),
              decreasing=TRUE)
            posAnterior <- (posAnterior-1)
            anterior <- estadisticasTotales[posAnterior+i,j]
            puntuacion <- (which(intervalo == anterior) - 1)
            # print(paste(intervalo))
            # print(anterior)
            # print(puntuacion)
            estadisticasTotales[fila+i,j] = puntuacion
          }
        }
      }
    }

    print("Total")
    print(estadisticasTotales)
    library("MASS")
    write.matrix(estadisticasTotales, file =
        paste(paste0(dirDataset, nomFile,"/"),
        paste0("5_Resultado_",tipoError,".csv"), sep="/"), sep=";")
  }
}
# estadisticas(nomFile="CLAS_01_iris.arff", numParticiones=5,
    tiposError=c("Total"))
# estadisticas(nomFile="CLAS_02_white-clover.arff",
    numParticiones=5, tiposError=c("Total"))
# estadisticas(nomFile="CLAS_03_hepatitis.arff", numParticiones=5,
    tiposError=c("Total"))
# estadisticas(nomFile="CLAS_04_breast-cancer.arff",
    numParticiones=5, tiposError=c("Total"))
# estadisticas(nomFile="CLAS_05_vehicle.arff", numParticiones=5,
    tiposError=c("Total"))
# estadisticas(nomFile="CLAS_06_credit-g.arff", numParticiones=5,
    tiposError=c("Total"))
# estadisticas(nomFile="CLAS_07_splice.arff", numParticiones=5,
    tiposError=c("Total"))
# estadisticas(nomFile="CLAS_08_letter.arff", numParticiones=4,
    tiposError=c("Total"))
# estadisticas(nomFile="CLAS_09_waveform-5000.arff",
    numParticiones=4, tiposError=c("Total"))
# estadisticas(nomFile="CLAS_10_mfeat-fourier.arff",
```

```
    numParticiones=4, tiposError=c("Total"))
# estadisticas(nomFile="REG_01_cholesterol.arff",
    numParticiones=4, tiposError=c("Total"))
# estadisticas(nomFile="REG_02_bodyfat.arff", numParticiones=4,
    tiposError=c("Total"))
# estadisticas(nomFile="REG_03_wisconsin.arff", numParticiones=4,
    tiposError=c("Total"))
# estadisticas(nomFile="REG_04_stock.arff", numParticiones=4,
    tiposError=c("Total"))
# estadisticas(nomFile="REG_05_meta.arff", numParticiones=4,
    tiposError=c("Total"))
# estadisticas(nomFile="REG_06_triazines.arff", numParticiones=4,
    tiposError=c("Total"))
# estadisticas(nomFile="REG_07_abalone.arff", numParticiones=4,
    tiposError=c("Total"))
# estadisticas(nomFile="REG_08_cpu_act.arff", numParticiones=4,
    tiposError=c("Total"))
# estadisticas(nomFile="REG_09_cal_housing.arff",
    numParticiones=4, tiposError=c("Total"))
# estadisticas(nomFile="REG_10_house_8L.arff", numParticiones=4,
    tiposError=c("Total"))

# Funci[U+FFFD]n para generar daset de regresion
generarDatasetRegresion <- function() {
  dirDataset<-
  "C:/Users/Eclipse/Documents/Master/Tesis/SVN/python/test_regresion/otro/"
  setwd(dirDataset)
  if (FALSE) {
    x <- seq(from=0, to=4, by=0.05)
    y <- seq(from=0, to=4, by=0.05)
    r <- seq(from=1, to=10, length=length(x))
    grid <- expand.grid(x=x, y=y)
    grid$z <- r
    grid<-as.matrix(grid)
    # print(grid)
    library("MASS")
    # write.matrix(grid, file = "patterns.exhaustive.arff", sep=",")

    x <- seq(from=0, to=4, by=0.5)
    y <- seq(from=0, to=4, by=0.5)
    r <- seq(from=1, to=10, length=length(x))
    grid <- expand.grid(x=x, y=y)
    grid$z <- r
```

```
    grid<-as.matrix(grid)
    library("MASS")
    write.matrix(grid, file = "100/patterns.arff", sep=",")
  } else {
    x <- seq(from=-4, to=4, by=0.05)
    grid <- c()
    for (valX in x) {
      grid<-c(grid,valX,((1/valX)*100))
    }
    # grid <- expand.grid(x=x, y=y)
    # grid<-as.matrix(grid)
    grid <- matrix(grid, ncol=2, byrow=TRUE)
    library("MASS")
    write.matrix(grid, file = "patterns.exhaustive.arff", sep=",")
    x <- seq(from=-4, to=4, by=0.5)
    grid <- c()
    for (valX in x) {
      grid<-c(grid,valX,((1/valX)*100))
    }
    #     grid <- expand.grid(x=x, y=y)
    #     grid<-as.matrix(grid)
    grid <- matrix(grid, ncol=2, byrow=TRUE)
    library("MASS")
    write.matrix(grid, file = "100/patterns.arff", sep=",")
  }

}
# generarDatasetRegresion()
```

Bibliografía

[1] Yudong Chen and Constantine Caramanis. Noisy and missing data regression: Distribution-oblivious support recovery. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 383–391. JMLR Workshop and Conference Proceedings, 2013.

[2] La Bitacora de AudieMan. Relacion entre datos, informacion, conocimiento e inteligencia. `http://audiemangt.blogspot.com.es/2010/05/datos-informacion-conocimiento-e.html`, 2010. [Online; accessed 25-July-2013].

[3] Bala Deshpande. Beware of 2 facts when using naive bayes classification for analytics. `http://www.simafore.com/blog/bid/100934/`, 2012. [Online; accessed 28-August-2012].

[4] L.C.M. Félix. *Data Mining no processo de extração de conhecimento de base de dados*. ICMC/USP, 1998.

[5] Marcelo R. Ferreyra. Powerhouse ¿qué es el ruido? `http://powerhousedm.blogspot.com.es/2007/10/qu-es-el-ruido.html`, 2007. [Online; accessed 25-July-2013].

[6] Luis Carlos Molina Félix. Data mining: torturando a los datos hasta que confiesen. `http://www.uoc.edu/web/esp/art/uoc/molina1102/molina1102.html`, 2002. [Online; accessed 25-July-2013].

[7] José Hernández Orallo, María José Ramírez Quintana, and César Ferri Ramírez. *Introducción a la Minería de Datos*. Pearson Educación, 2004.

[8] Chuck P. Lam and David G. Stork. Evaluating classifiers by means of test data with noisy labels. In *Proceedings of the 18th International*

Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico, 2003.

[9] J. Neville, D. Jensen, and B. Gallagher. Simple estimators for relational bayesian classifiers. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003), December 19-22, 2003, Melbourne, Florida, USA*, pages 609–612. IEEE Computer Society, Washington, DC, USA, 2003.

[10] Dorian Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann, 1999.

[11] Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[12] Ross J. Quinlan. Learning with continuous classes. In *5th Australian Joint Conference on Artificial Intelligence*, pages 343–348, Singapore, 1992. World Scientific.

[13] Matteo Re and Giorgio Valentini. Noise tolerance of multiple classifier systems in data integration-based gene function prediction. *J. Integrative Bioinformatics*, 7(3), 2010.

[14] S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, and K.R.K. Murthy. Improvements to the smo algorithm for svm regression. In *IEEE Transactions on Neural Networks*, 1999.

[15] Qun Sun, Xiuzhen Zhang, and Kotagiri Ramamohanarao. Noise tolerance of ep-based classifiers. In Tamás D. Gedeon and Lance Chun Che Fung, editors, *Australian Conference on Artificial Intelligence*, volume 2903 of *Lecture Notes in Computer Science*, pages 796–806. Springer, 2003.

[16] René Villeda-Ruz and Javier Garcia-Garcia. Meaningful error estimations for data analysis. In *Proceedings of the 2009 Mexican International Conference on Computer Science*, ENC '09, pages 280–288, Washington, DC, USA, 2009. IEEE Computer Society.

[17] Y. Wang and I. H. Witten. Induction of model trees for predicting continuous classes. In *Poster papers of the 9th European Conference on Machine Learning*. Springer, 1997.