UNIVERSIDAD DE BURGOS
Escuela Politécnica Superior

PROGRAMA DE DOCTORADO
«Ingeniería Civil e Industrial»

«*Minería de Datos* y combinación de regresores»

Carlos Pardo Aguilar
Burgos, diciembre de 2015

La investigación realizada en esta tesis doctoral ha sido parcialmente subvencionada por el Ministerio de Economía y Competitividad, proyecto TIN-2011-24046.

Agradecimientos

En primer lugar, mi principal agradecimiento es para mis directores de tesis, César y Juanjo sin vuestra paciencia y vuestro empuje esta tesis no habría podido llegar termino.

Este agradecimiento debo hacerlo extensivo a todos mis compañeros del grupo de investigación ADMIRABLE. Gracias además de a César y Juanjo, a Jesús, Raúl, Carlos, Andrés, José y Álvar, por vuestros ánimos y por asumir estos últimos años todas las tareas de ingratas de la universidad que me han facilitado el trabajo últimamente y, a los más veteranos gracias por ese compañerismo de todos estos años. Agradecer al grupo de la Universidad de Córdoba que coordina el profesor Nicolás Pedrajas por compartir el uso de su máquina de computo; y a toda la comunidad investigadora que difunde de forma libre el código y pone disponibles los conjunto de datos parar comparar los resultados. Un recuerdo a mis profesores que me formaron desde niño, en especial a dos que me marcaron especialmente en la Universidad de Valladolid, Pablo de la Fuente y Valentín Cardeñoso.

Y por último, agradecer a la familia, por una vida entera, por estar siempre allí, por apoyo continuo, por aguantar mis jornadas interminables de trabajo y no poderos hacer el caso que os merecéis. Clemente, Felisa, Nieves, Antonio y Carmen estáis en mi corazón siempre.

UNIVERSIDAD DE BURGOS

«*Minería de Datos* y combinación de regresores»

La tesis «*Minería de Datos* y combinación de regresores», que presenta Carlos Pardo Aguilar para optar al título de doctor, ha sido realizada dentro del programa de «Ingeniería Civil e Industrial», en el Área de Lenguajes y Sistemas Informáticos perteneciente al Departamento de Ingeniería Civil de la Universidad de Burgos bajo la dirección de los doctores D. César Ignacio García Osorio y D. Juan José Rodríguez Diez.

Los directores autorizan la presentación del presente documento como memoria para optar al grado de Doctor por la Universidad de Burgos.

En Burgos, a diciembre de 2015

Vº. Bº. del Director:	Vº. Bº. del Director:	El doctorando:
Dr. D. César Ignacio García Osorio	Dr. D. Juan José Rodríguez Diez	D. Carlos Pardo Aguilar

UNIVERSIDAD DE BURGOS
Escuela Politécnica Superior

Programa de doctorado «Ingeniería Civil e Industrial»

TESIS:
«*Minería de Datos* y combinación de regresores»

AUTOR: Carlos Pardo Aguilar

DIRECTORES: Dr. D. César Ignacio García Osorio y
Dr. D. Juan José Rodríguez Diez

RESUMEN:
La *Inteligencia Artificial* es el área de conocimiento que se dedica a la investigación en la mejora de algoritmos para añadir comportamiento más parecido al humano en los sistemas informáticos. La *Minería de Datos* es una de sus ramas, que está especializada en buscar y extraer información analizando conjuntos de datos. Su objeto de estudio son los sistemas que aprenden por sí mismos. Dentro de los sistemas que aprenden, se llaman *sistemas con aprendizaje supervisado* a aquellos a los que se les proporcionan tanto las entradas como las salidas esperadas de un conjunto de datos de entrenamiento. Cuando la salida esperada es una lista de categorías, el sistema se denomina clasificador, y cuando la salida es numérica, se denomina regresor. La combina-

ción de varios clasificadores o regresores para formar un sistema mejor se suele denominar con el término inglés *ensemble* y suele obtener mejores resultados que el de los métodos independientes que combina.

Esta tesis investiga la mejora que se consigue, al utilizar en tareas de regresión, la combinación de modelos, en vez de la utilización individual de éstos, y analiza que tipo de combinación consigue mayores mejoras. Además, también propone la adaptación a problemas de regresión de algunos algoritmos que estaban diseñados originariamente para clasificación. Más concretamente, se centra en analizar combinaciones homogéneas, es decir, cuando combinan regresores del mismo tipo, e intentar ver qué combinación es la que más mejora a cada tipo de regresor base.

A partir de los resultado de estos estudios se han redactado varias publicaciones: un artículo de revista del primer cuartil, un capítulo de libro, tres comunicaciones en congresos internacionales y otra en un congreso nacional.

PALABRAS CLAVE:
Minería de Datos, Regresión, Algoritmos de combinación de modelos, *Rotation Forest*, *Random Oracle*, Proyecciones lineales supervisadas

UNIVERSIDAD DE BURGOS
Escuela Politécnica Superior

PhD program «Ingeniería Civil e Industrial»

THESIS: «*Minería de Datos* y combinación de regresores»
«Data Mining *and ensembles for regression tasks.*»

AUTHOR: Carlos Pardo Aguilar

SUPERVISORS: Dr. D. César Ignacio García Osorio y
Dr. D. Juan José Rodríguez Diez

ABSTRACT:
Artificial Intelligence *is the knowledge area devoted to research on improving algorithms to add more humanlike behavior to computer systems.* Data mining *is one of its branches, which specializes in searching and extracting information by analyzing datasets. Its object of study are the systems able to learn by themselves. Within learning systems, they are called* supervised learning systems *those who will provide both entries, as the expected outputs of a training dataset. When the expected output is a list of categories, the system is called classifier, and when the output is numeric, it is called regressor. The combination of multiple classifiers, or regressors, to form a new system is often referred to with the term "ensemble", often these combinations give better results than the individual methods that are*

combined.

This thesis investigates the improvement that can be achieved when, in regression tasks, algorithms of model combination are used instead of single models, and analyzes which one is the combination that gives higher improvements. In addition, this thesis also proposes the adaptation to regression of some methods that were originally designed for classification. More specifically, it focuses on analyzing homogeneous ensembles, ie, when the combined regressors are of the same type, and try to determine which combination is the one that achieves the greatest improvement of the base regressor.

From the results of these studies, several publications have been written: a paper in a journal from first quartile of the JCR index, a book chapter, three papers presented in international conferences and another presented in a national conference.

KEYWORDS:
Data mining, *Regression*, *Ensembles*, *Rotation Forest*, *Random Oracle*, *Supervised linear projections*

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

La *Inteligencia Artificial* es el área de conocimiento que se dedica a la investigación en la mejora de algoritmos para añadir comportamiento más parecido al humano en los sistemas informáticos.

La *Minería de Datos* es una disciplina emergente, dentro de la Inteligencia Artificial, que trata de proveer distintos métodos para el análisis de la información que permitan la solución de diversos problemas. Para ello, se busca encontrar relaciones entre los datos, predecir valores de unos datos a partir de otros existentes, agrupar instancias de datos por su similitud, etc.

Dentro de la *Minería de Datos*, el campo de la *clasificación supervisada*, se ocupa del estudio de aquellos métodos que se utilizan para predecir a qué clase, de entre un posible conjunto de *clases* previamente definido, pertenece una instancia de un conjunto de datos. Los métodos de clasificación han ido evolucionando con el tiempo, y en la actualidad hay una familia de métodos cuyo estudio e investigación está en auge, son los *ensembles*. Los *ensembles* son aquellos algoritmos que obtienen su resultado como combinación del resultado de un conjunto de clasificadores individuales.

Cuando en vez de predecir una clase lo que hay que hacer es predecir un valor, es decir, se pasa de un conjunto discreto de

Tabla 1.1: Resultados de búsqueda en la red.

Resultados en Google	Resultados en G.académico	patrón de búsqueda
1 600 000	197 000	regresión
353 000	72 400	«regresión lineal»
1 990	12	«combinación de regresores»
58 200 000	4 340 000	*regression*
359 000 000	2 900 000	*ensemble*
9 460 000	2 840 000	«*linear regression*»
5 310	656	«*regression ensemble*»

valores a un conjunto no discreto, se habla de regresión.

El ámbito de esta tesis se va a concentrar en el estudio del funcionamiento de diferentes algoritmos para resolver problemas de regresión, más concretamente nos centraremos en los algoritmos que usan combinaciones homogéneas de regresores.

Dicho ámbito es bastante citado si observamos las ocurrencias en la búsquedas en la red. Tanto en el ámbito global, como si restringimos la búsqueda al ámbito académico. En la Tabla 1.1 aparecen los resultados de búsqueda en la red tanto el término en español «regresión» como de los términos ingleses «*regression*» y «*ensemble*». Los términos ingleses aparecen varios millones de veces en el ámbito académico y incluso muchas más en el global[1].

Los resultados se reducen si concretamos las búsquedas en las expresiones «*regresión lineal*», «combinación de regresores» y sus equivalentes inglesas, pero los resultados *«linear regression»* en el ámbito académico siguen siendo millones.

Esta tesis se ha desarrollado dentro del grupo de investigación *Advanced Data MIning Research And Bioinformatics LEarning* [2] (ADMIRABLE) de la Universidad de Burgos, que se dedica, principalmente, al desarrollo nuevos algoritmos de *Mine-*

[1] Google informa que el número de resultados es aproximado y alguno de los resultados pueden ser por otros usos de los términos.

[2] `http://www2.ubu.es/ginves/ing_const/admirable/`

ría de Datos y a estudiar el comportamiento y la aplicación de los algoritmos existentes, tanto en entornos de *benchmarking*, como en entornos de aplicaciones reales.

1.1. Organización de la tesis

Para empezar, la tesis repasará qué es la regresión en el Capítulo 2, para, a continuación pasar a ver los principales algoritmos básicos de regresión. El Capítulo 3 se dedicará a comentar algunos de los algoritmos que se pueden utilizar para combinar dichos regresores base.

El Capítulo 5 serán los resultados de unos estudios experimentales de dichos algoritmos, dentro del entorno de trabajo experimental que se comentará en el Capítulo 4.

En el Capítulo 6 trataremos la adaptación del algoritmo de combinación de clasificadores *Rotation Forest* a problemas de regresión. El Capítulo 7 presentará un estudio del efecto de diferentes proyecciones lineales en en problemas de regresión, con el objeto de en el futuro estudiar como afectan al *Rotation Forest*.

Para acabar la tesis, se presentarán unas conclusiones y unas líneas de futuros trabajos.

1.2. Aportaciones de esta tesis

Como resultado de esta tesis se han obtenido:

- Comparativas del funcionamiento de diferentes algoritmos de combinación de modelos en problemas de regresión usando diferentes métodos base para generar los modelos (perceptrón multicapa , *Random Oracle*, función de base radial, máquina de soporte vectorial).

- La adaptación de los algoritmos *Random Oracle* y *Rotation Forest* a problemas de regresión y su comparación con otros algoritmos.
- Una nomenclatura genérica común a las diferentes proyecciones lineales existentes, comparativas del funcionamiento de esas proyecciones lineales y nuevas variantes (LPHD, HDAr, SNDA).

Estos estudios se han documentado y se han redactado varias publicaciones: un artículo de revista del primer cuartil, un capítulo de libro, tres comunicaciones en congresos internacionales y otra en un congreso nacional.

- Artículo de revista
  - Pardo, C., Díez-Pastor, J.F., García-Osorio, C., Rodríguez, J.J.: Rotation forest for regression. Applied Mathematics and Computation 219(19), 9914–9924 (Jun 2013), doi: `10.1016/j.amc.2013.03.139`
    En 2013, *Applied Mathematics and Computation* tuvo un índice de impacto 1,6 situándose en la el primer cuartil de su área de conocimiento –en la posición 30 de las 241 de la categoría «*MATHEMATICS, APPLIED*».
- Capítulo de libro:
  - Pardo, C., Rodríguez, J.J., Díez-Pastor, J.F., García-Osorio, C.: Random oracles for regression ensembles. In: Okun, O., Valentini, G., Re, M. (eds.) Ensembles in Machine Learning Applications, Studies in Computational Intelligence, vol. 373, pp. 181–199. Springer Berlin / Heidelberg (2011).

- Artículos en congresos internacionales:
  - IEA/AIE 2010:
    Pardo, C., Rodríguez, J.J., García-Osorio, C., Maudes, J.: An empirical study of multilayer perceptron ensembles for regression tasks. In: García-Pedrajas, N., Herrera, F., Fyfe, C., Benítez, J., Ali, M. (eds.) Trends in Applied Intelligent Systems, Lecture Notes in Computer Science, vol. 6097, pp. 106–115. Springer Berlin Heidelberg (2010), `http://dx.doi.org/10.1007/978-3-642-13025-0_12`.
  - SUEMA 2010: Pardo, C., Rodríguez, J.J., Díez-Pastor, J.F., García-Osorio, C.: Random oracles for regression ensembles. In: Workshop on Supervised and Unsupervised Ensemble Methods and their Application. pp. 85–96 (2010).
  - ICPRAM 2012:
    Pardo-Aguilar, C., Díez-Pastor, J.F., García-Pedrajas, N., Rodríguez, J.J., García-Osorio, C.: Linear projection methods-an experimental study for regression problems. pp. 198–204 (2012)
- Artículo en congreso nacional:
  - CAEPIA 2015:
    Pardo-Aguilar, C., Rodríguez, J.J., García-Osorio, C., Díez-Pastor, J.F.: Un estudio experimental de combinaciones usando redes de funciones de base radial en tareas de regresión. In: admitido en (2015).

Capítulo 2

La regresión y los regresores base

La regresión es el conjunto de técnicas que se usan para poder predecir un valor numérico asociado a un dato de entrada.

Más formalmente, el problema a resolver se enuncia como sigue:

Dado un conjunto D de n datos, formado por pares $(\mathbf{x}_i, y_i)$, $i = 1, 2, \ldots, n$, donde $\mathbf{x}_i \in \mathbb{R}^d$ e $y_i \in \mathbb{R}$, hay que encontrar la función desconocida $f(\mathbf{x})$ que intente relacionar cada $\mathbf{x}_i$ con su correspondiente y_i.

Según [113] el término regresión se uso por primera vez en un trabajo de 1877 [38]. Galton[1] acuñó el término en este trabajo sobre el tamaño de los guisantes, y en otro posterior de 1885, que es más conocido, sobre la altura de los hijos con respecto de la de sus padres, llegando en esos estudios a la conclusión de que las medidas de la siguiente generación se acercaban, «regresaban», a la media de la población. Este término, inicialmente sólo con significado en la biología, se extendió a otros trabajos estadísticos.

Se denominan regresores base al conjunto de algoritmos simples que se han utilizado para resolver problemas de regresión.

[1] Sir Francis Galton (1822 – 1911)

Los principales regresores base que se han usado son: la *regresión lineal*, los *árboles de regresión*, los *perceptrones multicapa*, las *redes de funciones de base radial* y las *máquinas de soporte vectorial.*

2.1. La *regresión lineal*

La *regresión lineal* [27] consiste en asociar una recta a los datos de entrada para obtener, predecir, un valor numérico. En otras palabras, el problema es encontrar la relación entre los datos de entrada y salida y nos aproximamos a la solución suponiendo que esta relación es de tipo lineal. La validez de esta aproximación se determina por el error que comete en sus predicciones. En *regresión lineal*para determinar el error se usa la *media cuadrática de los errores*[2] de sus predicciones:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} e_i^2}{n}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - r_i)^2}$$

donde e_i representa el error de la predicción i-ésima, es decir, la diferencia entre el valor esperado (y_i) y el valor predicho (r_i), el uso de esta medida del error es la razón de que a esta técnica se la denomine también ajuste por mínimos cuadrados (en inglés *Linear Least Squares.*).

La predicción de r_i para un $\mathbf{x}_i$ dado, que hace mínimo el error cuadrático es [103]:

$$r_i = P(\mathbf{x}_i) = \bar{y} + \frac{\frac{1}{n}\left(\sum_{k=1}^{n} y_k \mathbf{x}_k\right) - \bar{y}\bar{\mathbf{x}}}{\frac{1}{n}\left(\sum_{k=1}^{n} \mathbf{x}_k^2\right) - (\bar{\mathbf{x}})^2} \cdot (\mathbf{x}_i - \bar{\mathbf{x}})$$

[2]La media cuadrática de los errores es la raíz cuadrada de la media de los cuadrados de los errores, es frecuente referirse a ella por las siglas inglesas RMSE (*Root of Mean Squared Error*)

donde:

- $\bar{y} \in \mathbb{R}$ representa el valor promedio de los valores esperados,
- $\bar{\mathbf{x}} \in \mathbb{R}^d$ representa los valores promedio de cada dimensión,
- la diferencia de dos valores de $\mathbb{R}^d$ da como resultado un valor de $\mathbb{R}^d$ con las diferencias en cada dimensión,
- los productos de dos valores de $\mathbb{R}^d$ son productos escalares que dan como resultado un valor de $\mathbb{R}$,
- mientras que los productos de un valor escalar $\mathbb{R}$ por otro de $\mathbb{R}^d$ dan como resultado un valor de $\mathbb{R}^d$ con el producto del escalar por cada dimensión.

También se puede representar la solución con esta otra expresión, equivalente:

$$r_i = \frac{\sum_{k=1}^{n} \mathbf{y}_k}{n} + \frac{n\left(\sum_{k=1}^{n} y_k \mathbf{x}_k\right) - \left(\sum_{k=1}^{n} y_k\right)\left(\sum_{k=1}^{n} \mathbf{x}_k\right)}{n\left(\sum_{k=1}^{n} \mathbf{x}_k^2\right) - \left(\sum_{k=1}^{n} \mathbf{x}_k\right)^2} \left(\mathbf{x}_i - \frac{\sum_{k=1}^{n} \mathbf{x}_k}{n}\right)$$

El uso más antiguo de estas técnicas de *regresión lineal*, son anteriores al acuño del término regresión. Según [113] los primeros ejemplos de uso de regresión lineal en la historia son los trabajos sobre trayectorias de objetos celestes, principalmente cometas, publicados por Legendre[3] [67] y Gauss[4] [41] en la primera década del siglo XIX. Estos trabajos ya usaron el ajuste por mínimos cuadrados.

Estos trabajos consideraron que los datos se distribuían en una gausiana, pero posteriormente los trabajos de Fisher en

[3]Adrien-Marie Legendre (1752 – 1833)
[4]Carl Friedrich Gauss (1777 – 1855)

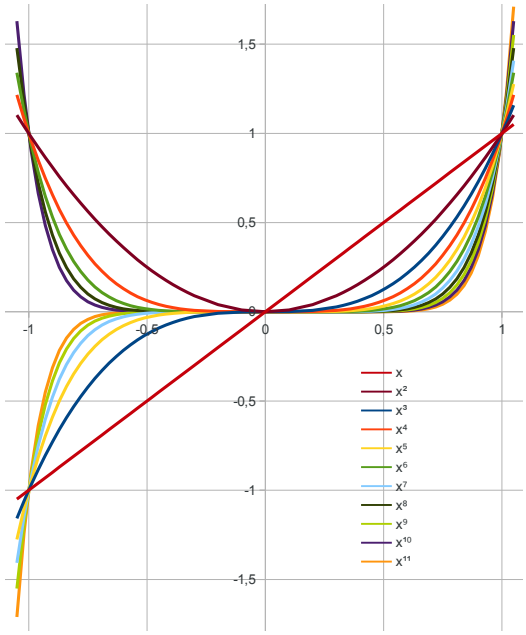

Figura 2.1: Forma de las funciones polinómicas monotérmino ($f(x) = x^n$) en la cercanía del origen de coordenadas.

1922 [28] y 1925 [29] buscaron hacer regresión para otro tipo distribuciones de los datos. Así se empezaron a estudiar regresiones con otros tipos de funciones y otras técnicas no lineales.

Una de las formas de generalizar es para ajustar no sólo usando líneas rectas, sino usando otro tipo de líneas polinómicas [26] de mayor grado que se curven para acercarse a los datos y así reducir el error, por ello podríamos definir que la *regresión lineal*genérica es asociar una función polinómica a los datos de entrada para obtener, predecir, un valor numérico.

Si el polinomio es de grado cero, es decir, sólo tiene una constante como término independiente; se tiene una función que predice un valor fijo, dependiendo del problema se suele elegir la moda, la mediana o un valor promedio. Si el polinomio es de grado uno se tiene una función que predice una línea recta, o un plano en el espacio. Si el polinomio es de grado dos se tiene una función que predice una parabólica. En la Figura 2.1 se

pueden observar la formas de las funciones de predicción de los polinomios desde grado uno hasta grado once.

2.2. Los *árboles de regresión*

Los árboles de decisión (*decision trees*, en inglés) se pueden usar tanto para clasificación, como para regresión [11].

Los árboles de decisión son aquellos que en cada nodo del árbol toman una decisión que separa el espacio de representación de los datos en tantas partes como ramas tiene. De tal forma que cuando llegamos a una hoja tengamos asignada una acción a realizar. La eficiencia de un árbol de decisión viene ligada a que la profundidad de una hoja, el número de ramas que hay desde la raíz del árbol hasta la hoja, sea inversamente proporcional a la probabilidad de que esa su acción sea la que se tenga que hacer. Para ellos cada rama divide el espacio de representación sean lo más iguales posibles.

En los problemas de clasificación la acción será asignar una clase a la entrada, y en cada nodo se intenta separar en ramas distintas los datos de cada clase, para lo que se busca la condición que deja menos ejemplos mal clasificados.

En los problemas de regresión la acción de cada hoja es asignar un valor. Los *árboles de regresión* más simples son aquellos en los que en cada nodo del árbol se divide el espacio de representación en dos ramas en función del valor de un único atributo. Hasta que en las hojas se toma la decisión de asignar un valor que será fijo en cada hoja. La condición que se busca en cada nodo, en regresión, es aquella que más reduce la diversidad de los subconjuntos de datos que van por cada rama. Una de las medidas usadas en regresión es la reducción de la suma de las varianzas de cada subconjunto de datos, otra de las medidas que se pueden usar para regresión es reducir la suma de los coeficientes Ginni de cada subconjunto de datos.

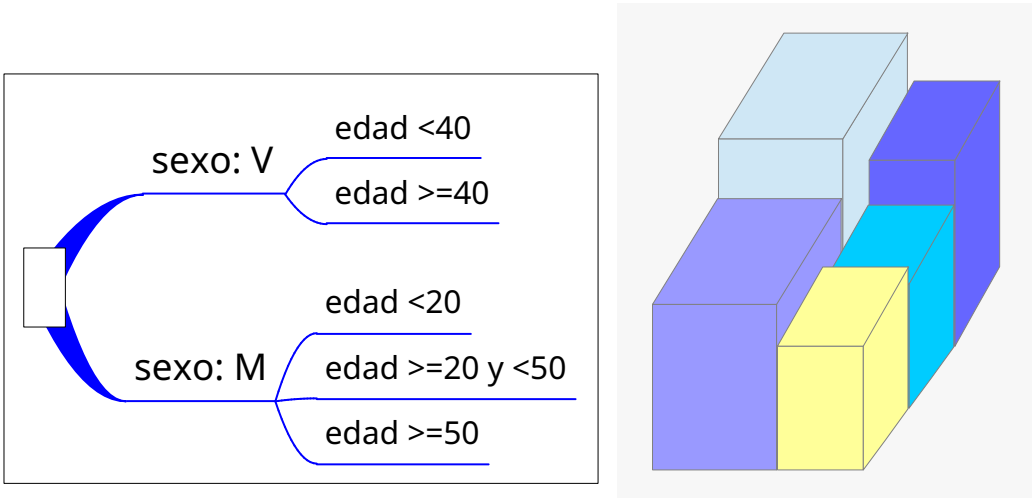

Figura 2.2: Ejemplo simple de predicción de un árbol de decisión.

El coeficiente definido por Gini [42] para medir la diversidad es la diferencia entre la distribución acumulada relativa de un conjunto de datos y aquella de pendiente constante y se calcula mediante la fórmula:

$$G_p = \frac{\sum_{i=1}^{n_p} \sum_{j=1}^{n_p} |y_i - y_j|}{2n_p \sum_{i=1}^{n_p} y_i} = \frac{\sum_{i=1}^{n_p-1} \sum_{j=i+1}^{n_p} |y_i - y_j|}{n_p \sum_{i=1}^{n_p} y_i}$$

donde G_p es el coeficiente Gini del subconjunto p que está formado por n_p elementos, y la salida esperada del elemento i-ésimo de ese subconjunto es y_i.

Como ejemplo, en la Figura 2.2 se muestra un posible árbol de decisión, para un caso en el que los datos tienen dimensión dos, y la representación de esas dos dimensiones en un plano y el valor a predecir como altura de las zonas en que se ha dividido el plano con el árbol de decisión.

A partir de esta configuración simple existen otras. En unas la función de decisión de cada nodo del árbol puede complicarse comparando más de un atributo, en otras la estructura del

árbol puede generalizarse partiendo más de dos ramas de un único nodo. Por otra parte, los árboles pueden construirse completamente para que representen todos los datos del conjunto de datos, o se pueden construir árboles que agrupen los datos mediante podas y que asignen el valor más representativo de un conjunto de datos, ese valor concreto depende del problema y la implementación, pudiendo usarse la moda (lo habitual en problemas de clasificación), la mediana, o una media. Además, existen otros *árboles de regresión* en los que en cada nodo hoja se tiene una función de *regresión lineal*. En estos algoritmos se consigue tener un menor número de ramas para un mismo nivel de error, a costa de aumentar la complejidad de la fase de construcción del árbol.

Una revisión detallada de los *árboles de regresión* y su justificación matemática puede consultarse en [11].

2.3. Los perceptrones multicapa

Un perceptrón multicapa [50] (*multilayer perceptron*, en inglés) es un tipo de red neuronal artificial.

La red neuronal está formada por elementos que denominamos neuronas artificiales, cada neurona tiene un estado de activación o no en función de unos valores de entrada, simulando la neurona biológica.

Las neuronas que forman un perceptrón multicapa se agrupan en capas (ver Figura 2.4). La primera capa recoge directamente los valores de las entradas al perceptrón multicapa , las capas posteriores tienen una función de activación que es función lineal de los valores de las neuronas de la capa anterior. La última capa es la que obtiene la salida. Como función de activación suele utilizarse un función sigmoidea o la tangente hiperbólica. Los coeficientes de las funciones lineales se inician de forma aleatoria, y posteriormente van variando en el proceso

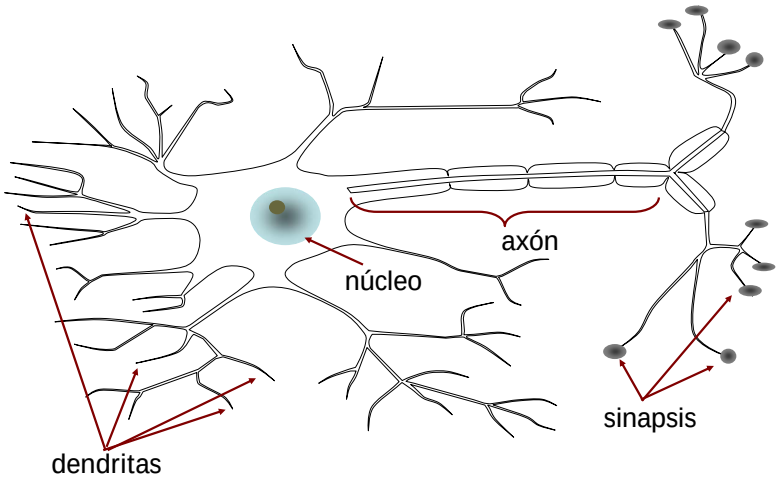

Figura 2.3: Modelo de neurona humana.
Se observan varias dendritas (entradas) y un axón (salida) que puede conectarse a varias neuronas

de entrenamiento para mejorar la salida. Este proceso se hace atrás-adelante, es decir modificando primero los pesos de la función de activación de la capa de salida y posteriormente de las capas anteriores, hasta llegar a la capa de entrada. Se suelen usar, habitualmente, sólo perceptrones multicapa con una única capa oculta, ya que el algoritmo de entrenamiento con varias capas es más lento al necesitar más cantidad de datos para converger a situaciones con buenas capacidades de generalización.

2.3.1. El algoritmo *backpropagation*

El algoritmo de *backpropagation* [55], es un algoritmo para determinar los pesos de las funciones de activación de cada neurona que tiene un entrenamiento en tres fases que se repiten varias veces.

En una primera fase llega un dato a la entrada y se calcula el estado de la salida de todas las neuronas, capa a capa, desde

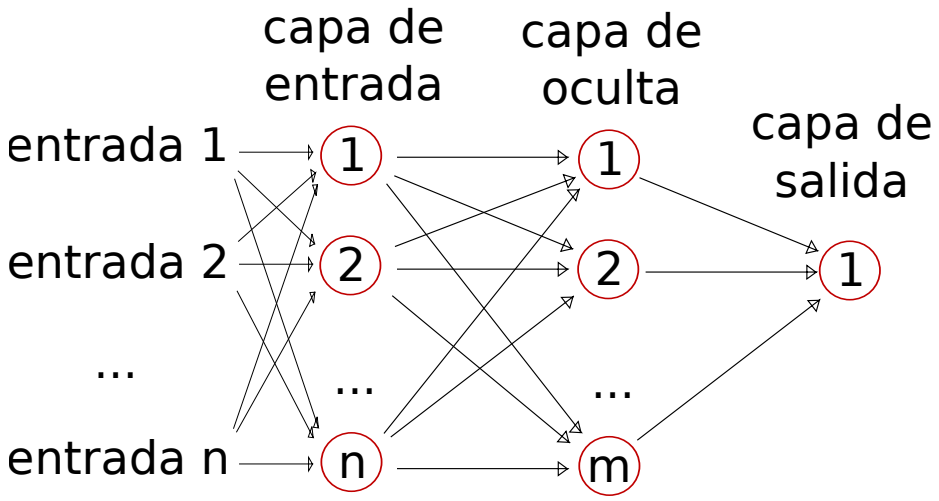

Figura 2.4: Modelo perceptrón multicapa .

la capa de entrada hasta la de salida de la red:

$$y_k^c = f_k^c\left(e\right) \mid e = \begin{cases} \sum\limits_{i \in entradas} w_{ik}^0 x_i & \text{si } c = 0 \\ \sum\limits_{i \in capa\ c-1} w_{ik}^c y_i^{c-1} & \text{si } c > 0 \end{cases}$$

donde:

- y_k^c es el valor de salida de la neurona k de la capa c,
- $f_k^c()$ es la función de activación de la neurona k de la capa c,
- w_{ik}^c es el peso de la entrada i-ésima en neurona k de la capa c, si c es capa de entrada (capa 0), la entrada se corresponde con la entrada i-ésima de la red (x_i), en el resto de las capas la entrada es la salida de la neurona i-ésima de la capa anterior (y_i^{c-1}).

En una segunda fase se calcula cuál ha sido el error cometido por la red y se propaga ese cálculo, desde la capa de salida hasta la capa de entrada, para saber la aportación de cada neurona a ese error global:

$$e_1^{salida} = z - y_k^{salida}$$

$$e_i^c = \sum_{k \in capa\ c+1} w_{ik}^{c+1} e_j^{c+1}$$

donde:

- z es el valor esperado,
- e_1^{salida} es el error de la red, que es el el de la neurona de salida
- e_i^c es la aportación al error de la neurona i de la capa c.

En una tercera fase se recalculan los pesos de entrada de las funciones de activación de todas las neuronas:

$$w_{ik}^c = w_{ik}^c + \nu e_k \frac{\partial f_k^c(w_{ik}^c)}{\partial w_{ik}^c} y_i$$

donde:

- $\partial f_k^c(w_{ik}^c)/\partial w_{ik}^c$ es la derivada de la función de activación de la neurona k de la capa c con respecto a la variación del peso de la entrada i-esima en neurona k de la capa c, y
- ν es la función de aprendizaje de la red que puede variar durante el proceso de entrenamiento.

Este proceso se repite para cada dato de entrenamiento. En muchos problemas es necesario que los datos de entrenamiento se entrenen varias veces, hasta que se determina que la red ha aprendido adecuadamente.

El algoritmo de *backpropagation* es el más conocido y utilizado para el entrenamiento de redes neuronales, pero se han propuesto alternativas, por ejemplo los enfoques basados en algoritmos genéticos [40].

Las redes neuronales son un paradigma ampliamente estudiado [32, 6, 51, 96] que ha sido extensivamente utilizado en diversos ámbitos [99], entre ellos la *Minería de Datos* [76, 16], y por supuesto también para la construcción de *ensembles* [58]

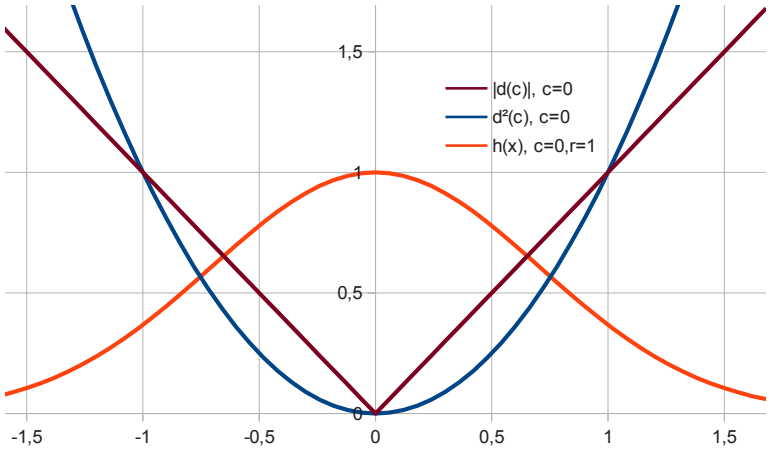

Figura 2.5: Ejemplos de funciones de base radial.
Distancia al origen, cuadrado de la distancia al origen y gausiana centrada en el origen y radio 1.

y la solución de problemas de regresión [64]. Además, en los últimos años se ha dado un resurgir de las redes neuronales con las «extreme learning machines» [54, 22] y el «deep learning» [52, 3, 65].

2.4. Las redes de funciones de base radial

Las redes de funciones de base radial (*radial basis function network*, en inglés) [7, 117] son otro caso de redes neuronales artificiales. En este caso, las neuronas, que la forman, también se agrupan en capas, como en el perceptrón multicapa de la Figura 2.4.

La primera capa recoge directamente los valores de las entradas a la red, pero la capa oculta está formada por neuronas que tienen una función de base radial como función de activación, y la última capa es la que obtiene la salida.

Las funciones de base radial son un caso de funciones lineales monótonas crecientes (o decrecientes) con la distancia de la entrada a un valor de referencia denominado centro. Para cada

neurona se fija el centro, la escala en la que se calcula la distancia y el tipo de función. En la Figura 2.5 se pueden ver los ejemplos típicos de funciones de base radial monótonas crecientes como son el valor absoluto de la distancia, o el cuadrado de la distancia, el ejemplo típico de función de base radial monótona decreciente con la distancia es la gausiana de centro en $\mathbf{c}$:

$$h(\mathbf{x}) = e^{-\left(\frac{d(\mathbf{x},\mathbf{c})}{r}\right)^2}$$

El proceso de entrenamiento, normalmente se hace por fases, inicialmente se entrenan las neuronas de la capa oculta para que cada una de ellas *aprenda* con una función de base radial a identificar una zona del espacio de entrada, un centro y un radio, para posteriormente, pasar a que la capa de salida *aprenda* el peso que cada una de esas zonas tiene en la salida. Cuantas más neuronas hay en la capa oculta, en más zonas se puede dividir en espacio de entrada.

Existen trabajos recientes que buscan optimizar el uso de redes de funciones de base radial como [15, 43, 44] algunos usando algoritmos bioinspirados [5, 92]. También se encuentran trabajos que las aplican para resolver problemas tanto industriales [97, 89] como de rendimiento académico [75].

2.5. Las máquinas de soporte vectorial

Las máquinas de soporte vectorial (*support vector machines*, en inglés) [101] buscan una frontera que separe el espacio en dos zonas. Más concretamente, buscan el hiperplano frontera que intente minimizar el riesgo de error cuando se generalice a nuevos ejemplos.

Inicialmente se definieron para clasificación. En los problemas de clasificar ejemplos separables, (ejemplos en negro en la Figura 2.6) las máquinas de soporte vectorial buscan encontrar

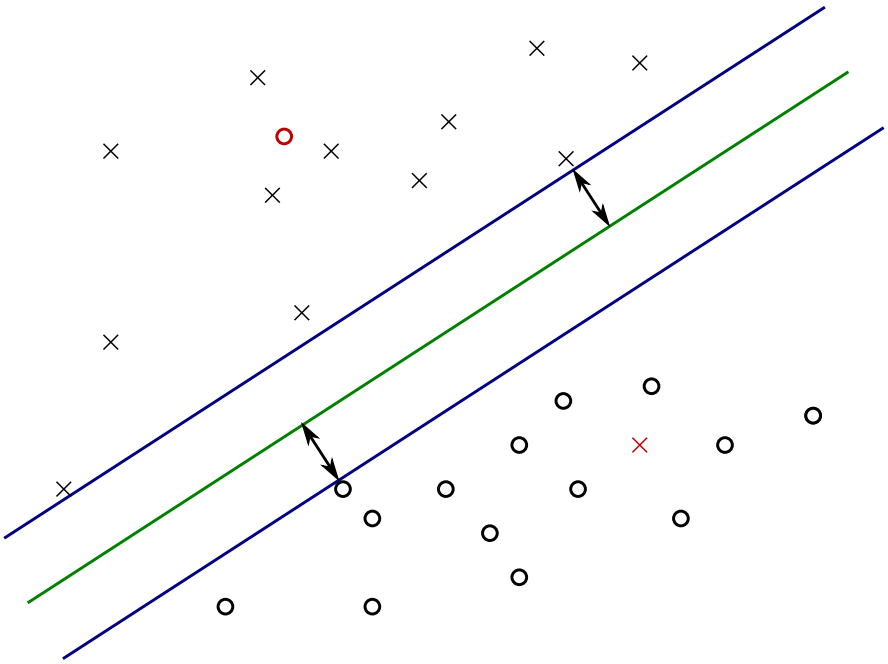

Figura 2.6: Ejemplo de separación lineal que busca una SVM para resolver un problema de clasificación.

la frontera (línea central verde) entre clases que maximice el margen de separación, es decir, se busca que todos los elementos de cada clase queden en lados distintos de la frontera y que la distancia a la frontera, de los elementos más cercanos a ella, sea la mayor posible (líneas azules equidistantes de la frontera).

El algoritmo se resuelve buscando el vector $\mathbf{w}$, perpendicular al hiperplano frontera. Si al proyectar con ese vector (usando el producto escalar) el resultado es mayor que un valor b estamos en una clase y si no en otra. Todo esto se puede expresar como:

$$\mathbf{w} \cdot \mathbf{x}_i - b \begin{cases} > 0 \Longrightarrow \textbf{clase x} \\ < 0 \Longrightarrow \textbf{clase o} \end{cases} \tag{2.1}$$

Pero se busca que no haya ejemplos en una franja, alrededor de la frontera:

$$\mathbf{w} \cdot \mathbf{x}_i - b \begin{cases} \geq 1 \Longrightarrow \textbf{clase x} \\ \leq -1 \Longrightarrow \textbf{clase o} \end{cases}$$

o si representamos la salida con los valores ± 1:

$$y_i(\mathbf{w}\cdot\mathbf{x}_i - b) - 1 \geq 0 \mid y_i = \begin{cases} +1 & \forall i \in \textbf{clase x} \\ -1 & \forall i \in \textbf{clase o} \end{cases} \tag{2.2}$$

La anchura de la franja fronteriza se puede calcular como la proyección sobre un vector unitario perpendicular a la frontera, del vector diferencia entre dos de los ejemplos en el borde de la franja (los que en la Ecuación 2.2 igualan a cero), uno de la clase x ($\mathbf{x}_x$) y otro de la clase o ($\mathbf{x}_o$). El vector $\mathbf{w}$ es perpendicular a la frontera, dividiendo este vector entre su módulo, se obtiene un vector unitario, $\mathbf{w}/\|\mathbf{w}\|$. La proyección del vector diferencia sobre este vector unitario será la distancia que hay que maximizar:

$$\begin{aligned} \text{máx}\left((\mathbf{x}_x - \mathbf{x}_o)\cdot\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) &= \text{máx}\left(\frac{(b+1)-(b-1)}{\|\mathbf{w}\|}\right) \\ = \text{máx}\left(\frac{2}{\|\mathbf{w}\|}\right) \Rightarrow \text{mín}\left(\|\mathbf{w}\|\right) &\Rightarrow \text{mín}\left(\frac{\|\mathbf{w}\|^2}{2}\right) \end{aligned}$$

encontrar mínimo de la mitad del cuadrado es equivalente y simplifica operaciones posteriores.

Aplicando multiplicadores de Lagrange α_i para incorporar las restricciones:

$$L = \frac{\|\mathbf{w}\|^2}{2} - \sum_i^n \alpha_i \left[y_i(\mathbf{w}\cdot\mathbf{x}_i - b) - 1\right] \tag{2.3}$$

el mínimo está donde se anulan las derivadas:

$$\frac{\partial L}{\partial \mathbf{w}} = \frac{2\mathbf{w}}{2} - \sum_i^n \alpha_i y_i \mathbf{x}_i = 0 \;\Rightarrow\; \mathbf{w} = \sum_i^n \alpha_i y_i \mathbf{x}_i \tag{2.4}$$

$$\frac{\partial L}{\partial b} = \sum_i^n \alpha_i y_i = 0 \tag{2.5}$$

y sustituyéndolo $\mathbf{w}$ en la Ecuación 2.3 queda:

$$L = \frac{\left(\sum_i^n \alpha_i y_i \mathbf{x}_i\right)\cdot\left(\sum_j^n \alpha_j y_j \mathbf{x}_j\right)}{2} - \sum_i^n \alpha_i \left\{ y_i \left[\mathbf{x}_i \cdot \left(\sum_j^n \alpha_j y_j \mathbf{x}_j \right) - b \right] - 1 \right\}$$

$$L = \sum_{i}^{n} \alpha_i - \frac{1}{2} \sum_{i}^{n} \sum_{j}^{n} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{2.6}$$

de donde se deduce que todo depende del producto escalar de los ejemplos que no se anulan por los multiplicadores α_i.

El hecho de que en la Ecuación 2.6 el problema a minimizar dependa de un producto escalar permite una interesante generalización de las máquinas de soporte vectorial a través de lo que se conoce como *the kernel trick*. El producto escalar $(\mathbf{x}_i \cdot \mathbf{x}_j)$ se puede sustituir por una función del tipo *kernel* que nos de el resultado de dicho producto $K(\mathbf{x}_i, \mathbf{x}_j)$ que nos transforma la ecuación en

$$L = \sum_{i}^{n} \alpha_i - \frac{1}{2} \sum_{i}^{n} \sum_{j}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{2.7}$$

Lo interesante del *kernel trick* está en que para determinados *kernels* devuelven, directamente, el producto escalar de una transformación de los datos de entrada, aunque desconozcamos la naturaleza de esta transformación, con lo que al sustituir el producto escalar por una función de *kernel* estamos encontrando la frontera de división en una transformación no lineal del problema de partida, que puede ser de cualquier dimensión, incluso de dimensión infinita.

La solución de la Ecuación 2.7 se puede obtener utilizando técnicas estándar de optimización cuadrática, o también aplicando métodos iterativos como el de descenso más rápido (*gradient descent*, en inglés)

Cuando no se encuentra la separación entre las clases (elementos en rojo en la Figura 2.6), para los ejemplos que no han podido ser separados, se definen variables de holgura que miden qué lejos de su borde de la franja fronteriza (por el lado opuesto) están estos ejemplos. Estas variables de holgura se introducen en el problema de minimización junto con un factor de penalización, representado típicamente con la letra C, de modo que

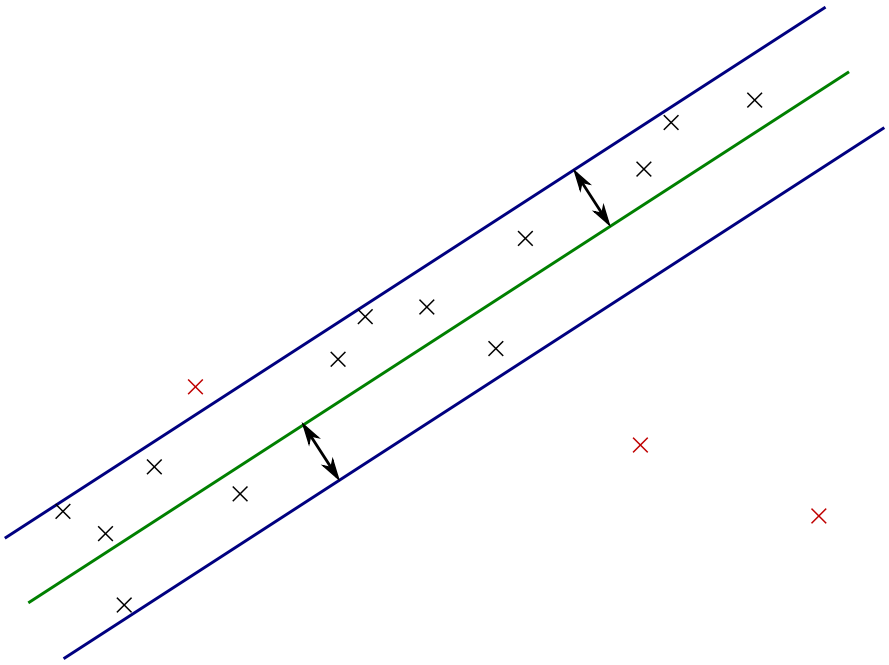

Figura 2.7: Ejemplos de recta de regresión que busca una SVM.

ahora además de minimizar $\|\vec{w}\|^2$ (que como hemos visto es lo mismo que maximizar el margen) obliga a minimizar el valor de las variables de holgura. Es decir, cuando las clases no son separables, al menos que los errores que se cometan sean los menores posibles

En 1997, Druker et al. [24] ampliaron su uso para regresión denominándolas SVRM[5].

En los problemas de regresión, como el de la Figura 2.7, las máquinas de soporte vectorial buscan que todos los ejemplos, o casi todos, estén dentro de una frontera de la anchura mínima Expresándonos de forma no precisa, podríamos decir que lo que se busca es la *famosa «recta astuta»*[6] de menor anchura que pase por todos los puntos.

[5]Existen bibliotecas con los algoritmos ya programados, por ejemplo LIBSVM disponible en `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[6]Falso teorema que dice permitir hacer pasar una recta por tres puntos no alineados, normalmente usado como broma para esconder imprecisiones en un dibujo técnico.

Buscamos, de igual forma que en clasificación, un vector $\mathbf{w}$ perpendicular al hiperplano de regresión, pero cambiando el objetivo, esta vez buscamos una franja entorno al hiperplano en la que queden todo los ejemplos, o casi todos:

$$|y_i - \mathbf{w}\mathbf{x}_i - b| \leq 1 \tag{2.8}$$

lo que vuelve a ser resolver el mismo tipo de problema, encontrar la distancia proyectada mínima entre los dos puntos, uno en cada borde de la franja alrededor del hiperplano, $\mathbf{x}_+$, el que hace 1 la parte izquierda de la Ecuación 2.8 antes de aplicar el valor absoluto y $\mathbf{x}_-$, el que hace -1 esa parte:

$$\text{mín}\left((\mathbf{x}_+ - \mathbf{x}_-)\frac{\mathbf{w}}{\|\mathbf{w}\|}\right) \Rightarrow \text{mín}\left(\frac{\|\mathbf{w}\|^2}{2}\right)$$

Para los punto que caen fuera de la franja, se introduce en el problema, como en clasificación,una penalización para esos puntos externos.

2.6. Los *vecinos más cercanos*

El algoritmo de los *vecinos más cercanos* (*k-nearest neighbors*, en inglés) se usó para problemas de regresión en 1965 por Loftsgaarden y Quesemberry [74]. Trabajo posteriormente ampliado por [48, 108].

El algoritmo del *vecino más cercano* decide que la predicción de un elemento es la misma que la del elemento que se encuentra a menor distancia en el espacio de características. Es como si el espacio se dividiera en celdas que tuvieran sus paredes equidistantes de dos ejemplos y todos los valores de cada celda tienen el mismo valor (ver Figura 2.8).

Las variantes con *k vecinos más cercanos*, en lugar de buscar un único vecino, determinan los k elementos de entrenamiento más próximos al que se desea predecir y le asignan un valor

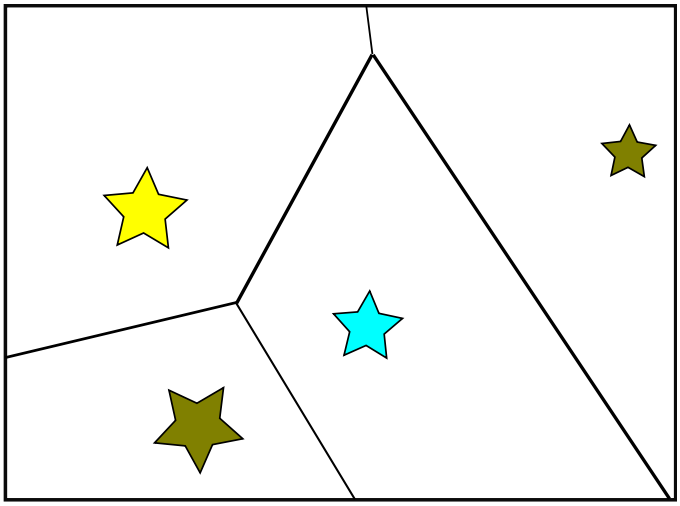

Figura 2.8: Ejemplo de zonas del plano que resuelve cada vecino más cercano.

que es función de los valores de sus vecinos, típicamente el valor promedio.

Existen implementaciones que permiten variar la función que se usa para el cálculo de la distancia entre elementos, cambiando algún parámetro del algoritmo, es posible pasar de usar la distancia euclidiana a la distancia de Manhattan o a la del coseno, o a una gran variedad de medidas de la distancia existentes como la distancia de Minkowsky [1], la de Mahalanobis [82]; u otras como la de Camberra, la de Chebychev, la cuadrática, la de correlación, y la χ-cuadrada [18, 79]; también se pueden usar medidas de similitud del contexto [4], o medidas heterogéneas (para atributos numéricos y nominales) [114]. En la Figura 2.9 se muestran las curvas que unen los puntos equidistantes a un centro según algunas definiciones de la distancia y en la Figura 2.10 están las definiciones matemáticas.

Además, existen otras implementaciones en las que distancia a cada uno de los vecinos se usa para ponderar su influencia en el resultado de la predicción [111, 31]

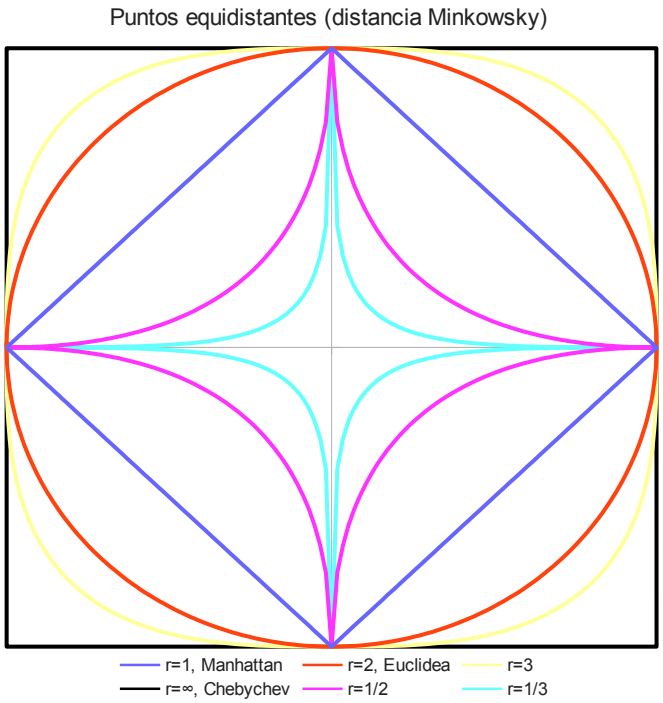

Figura 2.9: Curvas de los puntos equidistantes al centro según distintas medidas de la distancia.

Minkowsky

$$D(\mathbf{x},\mathbf{y}) = \left(\sum_{i=1}^{d} |x_i - y_i|^r\right)^{\frac{1}{r}}$$

Euclidiana

$$D(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2}$$

Minkowsky con $r = 2$. Mahalanobis con $S = I$

Manhattan

$$D(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d} |x_i - y_i|$$

Minkowsky con $r = 1$.

Chebychev

$$D(\mathbf{x},\mathbf{y}) = \max_{i=1}^{d} |x_i - y_i|$$

Minkowsky con $r = \infty$

χ-cuadrada

$$D(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d} \frac{1}{\xi_i}\left(\frac{x_i}{\varepsilon_{\mathbf{x}}} - \frac{y_i}{\varepsilon_{\mathbf{y}}}\right)^2$$

Donde ξ_i es la suma de los valores del atributo i, y $\varepsilon_{\mathbf{x}}$ es la suma de todos los valores del vector $\mathbf{x}$.

Cuadrática

$$D(\mathbf{x},\mathbf{y}) = (\mathbf{x}-\mathbf{y})^T Q\,(\mathbf{x}-\mathbf{y}) = \sum_{j=1}^{d}\left(\sum_{i=1}^{d}(x_i - y_i)q_{ji}\right)(x_j - y_j)$$

Donde la matriz Q es una matriz $m \times m$ definida positiva de pesos dependientes del problema concreto.

Mahalanobis

$$D(\mathbf{x},\mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})^T S^{-1}(\mathbf{x}-\mathbf{y})}$$

Donde S es la matriz de covarianzas entre los dos vectores.

Euclidiana normalizada

$$D(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{d}\frac{(x_i - y_i)^2}{\sigma_i^2}}$$

donde σ_i^2 es la varianza de x_i. Mahalanobis con S diagonal

Camberra

$$D(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{d}\frac{|x_i - y_i|}{|x_i + y_i|}$$

versión con pesos de la Manhattan.

Coseno del ángulo

$$D(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{d} x_i y_i}{\sqrt{\sum_{i=1}^{d} x_i^2}\sqrt{\sum_{i=1}^{d} y_i^2}}$$

Bray Curtis o **Sorensen**

$$D(\mathbf{x},\mathbf{y}) = \frac{\sum_{i=1}^{d}|x_i - y_i|}{\sum_{i=1}^{d}(x_i + y_i)}$$

método de normalización habitual en botánica, ecología y otras ciencias medioambientales.

Figura 2.10: Varias medidas de la distancia.

2.7. Otros regresores base

Además de los regresores mostrados en este capítulo, existen multitud de otros regresores, los que siguen son sólo algunos ejemplos de otros métodos que también se han utilizado para obtener una solución al problema de regresión:

- Los núcleos [80, 110]
- Distintos tipos de proyecciones para regresión [36],
- Modelos de lógica difusa [105, 91, 14],
- Alternancia condicional de expectativas ACE [12],
- Estimadores con series ortogonales [90],
- Estimación de la media de la derivada ADE [47],
- Otros tipos de redes neuronales [102],
- Curvas spline adaptativas [35],
- Modelos adaptativos generalizados [49].

Capítulo 3

Las combinaciones de regresores

Se plantea la cuestión de si son mejorables las predicciones obtenidas por los métodos mencionados en el capítulo anterior. Si se puede confiar en la predicción de un único modelo, o, en su lugar, como haríamos con un dictamen de un experto médico o pericial, sería conveniente compararlo con una «segunda opinión», o incluso una tercera, o ulteriores opiniones de otros expertos, modelos en nuestro problema.

El planteamiento que se seguirá es como si se organizara un concilio para comparar y discutir diferentes puntos de vista de multitud de expertos, y obtener de esta forma una solución que se pueda afirmar que comete menos error. Se va a buscar, por lo tanto, algoritmos que combinen las predicciones individuales de varios de los modelos de regresión.

La combinación de clasificadores o regresores para formar otro, con la intención de que sea más preciso, recibe el nombre inglés de *ensemble* [59].

La construcción de una combinación supone tener en cuenta cuestiones relativas a cómo aprenden, o se entrenan, los miembros que forman la combinación, qué entradas van a llegar a cada uno de ellos, y qué salidas se van a producir en cada uno, así como diseñar cómo se va a computar la salida final de la

combinación (p.e., votación o votación ponderada de los clasificadores/regresores base, etc.).

En la bibliografía [60] se puede comprobar que los algoritmos que usan combinaciones de métodos mejoran a los métodos simples en muchos de los problemas. Si nos centramos en el caso de las combinaciones de regresores, es fácil ver la razón por la que estos funcionan mejor que los regresores individuales que lo componen. Tal como hacen notar Krogh y Vedelsgy [57], el error de generalización de la combinación es menor que la media de los errores de generalización de cada uno de los miembros de la combinación (ver la Figura 3.1). Si uno se fija en esta descomposición del error, se puede concluir que:

- El error de generalización de la combinación, $e(\mathbf{x})$, siempre es menor que el error medio de los regresores de la combinación, $\overline{\epsilon}(\mathbf{x})$.
- Una combinación debería estar compuesta de regresores precisos y bien entrenados, pero que difieran entre ellos para incrementar la diversidad de la combinación, que aumenta la ambigüedad media de la combinación, $\overline{a}(\mathbf{x})$.

Por lo tanto, como estos algoritmos para mejorar el resultado necesitan que los métodos base aporten comportamientos diferentes «diversidad», se han propuesto varias estrategias para asegurar la diversidad. Una de las estrategias para obtener diversidad sería el usar métodos base diferentes, otra sería usar el mismo método base, pero variando diferentes parámetros del mismo, y una tercera estrategia para obtener diversidad usando el mismo método base con los valores de sus parámetros idénticos, es entrenar cada miembro con diferentes datos de entrenamiento, generados por selección o proyección del conjunto de datos de entrenamiento.

En los casos de estudio del presente trabajo se realizarán comparaciones de combinaciones homogéneas de métodos, es decir

Si se define la salida del *ensemble* como una media ponderada de sus miembros

$$\overline{f}(\mathbf{x}) = \sum_{k=1}^{K} w_k f_k(\mathbf{x})$$

donde $f_k(\mathbf{x})$ representa el k-ésimo regresor de la combinación y los pesos w_k suman 1, $\sum_k w_k = 1$. El error de generalización de la combinación

$$e(\mathbf{x}) = \Big(y(\mathbf{x}) - \overline{f}(\mathbf{x})\Big)^2$$

Puede ser descompuesto como

$$e(\mathbf{x}) = \overline{\epsilon}(\mathbf{x}) - \overline{a}(\mathbf{x})$$

Donde el primer sumando, $\overline{\epsilon}(\mathbf{x})$, es el error medio de los regresores miembros de la combinación

$$\overline{\epsilon}(\mathbf{x}) = \sum_{k=1}^{K} w_k \Big(y(\mathbf{x}) - f_k(\mathbf{x})\Big)^2$$

Y el segundo, $\overline{a}(\mathbf{x})$, es la ambigüedad media o diversidad de la combinación

$$\overline{a}(\mathbf{x}) = \sum_{k=1}^{K} w_k \left(f_k(\mathbf{x}) - \overline{f}(\mathbf{x})\right)^2$$

Figura 3.1: Descomposición del error de generalización de una combinación de regresores como combinación del error medio de los regresores de la combinación y de la diversidad de la combinación.

se usa con la última estrategia, la de variar que parte de los datos de entrenamiento se muestran a cada modelo, para determinar los métodos que mejor se comportan y cuáles son sus parámetros que hacen variar más su comportamiento.

Aunque existen muchos algoritmos de combinación de modelos, en el presente trabajo se van a usar, principalmente, los algoritmos *Bagging*, *AdaBoost*, *Random Subspaces*, *Iterated Bagging*, *Rotation Forest* y *Random Oracle*.

3.1. *Bagging*

Ya en 1996, Breiman [8] define un método en el que se entrenan diferentes modelos y se toma como predicción una agregación de las predicciones de los diferentes modelos, al modelo lo llama *Bagging* como contracción de «***b****ootstrap* ***agg****regat****ing***».

Decide entrenar cada modelo con un conjunto de datos diferente, para lo que se crea, para cada modelo, una combinación con repetición con tantos elementos como el conjunto de datos de entrenamiento original, es decir, los elementos del conjunto de datos de entrenamiento para cada modelo se eligen siempre del conjunto de datos de entrenamiento original, sin tener en cuenta si ya se había elegido previamente, esto permite que algunos datos estén repetidos y que otros no figuren en la muestra elegida. En el Apartado 3.1.1 tenemos la fórmula que determina que con un conjunto de datos de n elementos se podrían entrenar CR_n^n modelos diferentes. Hay implementaciones que tienen opciones que permiten modificar la cardinalidad del conjunto de datos de entrenamiento reduciéndolo a un porcentaje del total. Se pueden usar variantes de *Bagging* que trabajen con conjuntos de entrenamiento más pequeños, con un 50 % o un 75 % del tamaño del conjunto de datos.

En los problemas de clasificación la predicción del algoritmo *Bagging* será la moda de las predicciones (clase más veces pre-

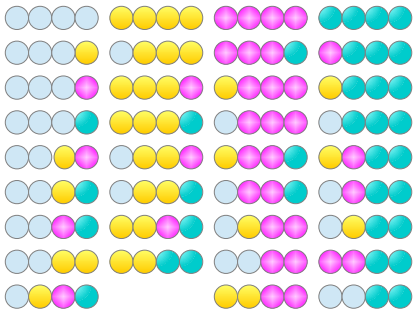

Figura 3.2: Ejemplo de las 35 combinaciones con repetición de 4 elementos de orden 4.

dicha), mientras que en los problemas de regresión, se calculará el promedio de las predicciones. Breiman estima que se necesita tener la «opinión» de, al menos, 50 miembros en los problemas de clasificación y 25 en los de regresión para tener suficiente confianza en que las combinación*Bagging* mejoran la predicción original.

3.1.1. Combinaciones con repetición

La mayor parte de estos algoritmos se basan en la '*explosión combinatoria*', en generar distintos conjuntos de datos aleatorios a partir de otros dados.

Se dice que una combinación con repetición es el conjunto de elementos que se forma si se selecciona dichos elementos de un conjunto inicial de elementos, de tal forma que se pueda elegir varias veces el mismo y que no importe el orden en el que aparecen.

El número el conjunto original tenía m elementos y seleccionamos n elementos, podríamos elegir una de las posibles combinaciones con repetición de de m elementos de orden n.

$$CR_m^n = \binom{m+n-1}{n} = \frac{(m-1+n)!}{(m-1)! \cdot n!}$$

En el caso planteado por varios de los algoritmos, se intentará

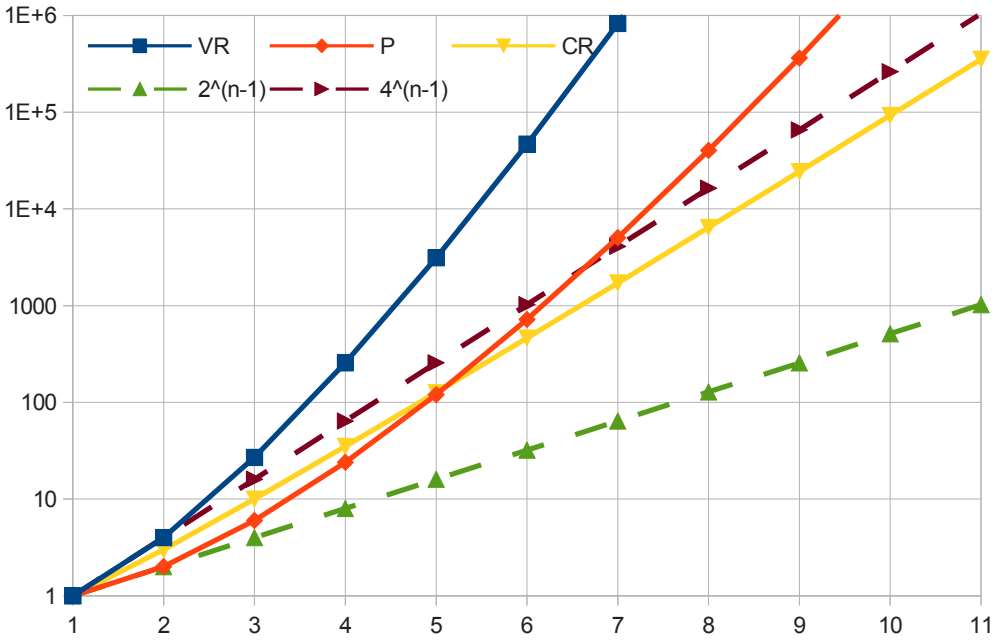

Figura 3.3: Crecimiento del número de VR_n^n, P^n y CR_n^n en función de n. Con referencia a las curvas exponenciales 2^{n-1} y 4^{n-1}.

que la combinación elegida siga teniendo el mismo número de elementos que la original, por lo que habría tantas combinaciones diferentes como combinaciones con repetición de n elementos de orden n existen.

$$CR_n^n = \binom{n+n-1}{n} = \frac{(n+n-1)!}{(n-1)! \cdot n!} = \frac{(2 \cdot n)!}{2 \cdot (n!)^2}$$

en la Figura 3.2 se puede observar las combinaciones con repetición para el caso de 4 elementos tomados de 4 en 4.

El número de combinaciones crece de forma cercana a una curva exponencial (4^{n-1}), por lo que con sólo 5 ejemplos ya se pueden formar más de cien combinaciones diferentes (126), con 7 más de mil (1716) y con 12 más de un millón (1 352 078). En la Figura 3.3 se puede ver ese crecimiento del número de combinaciones con repetición de n elementos tomados de n en n en función del n y compararlo con el de otras funciones de

crecimiento similar como son el número de permutaciones de n elementos y el número de variaciones con repetición de n elementos tomados de n en n y las curvas exponenciales de base 2 y base 4.

3.2. *AdaBoost*

También en 1996, Freund y Schapire [33] proponen una variante de *boosting* para construir una combinación de modelos y la denomina *AdaBoost*. Inicialmente, *AdaBoost* se usa para clasificación, pero ya Druker en [23] usa variantes para regresión.

La idea básica del algoritmo de *boosting* y sus diferentes variantes es generar modelos de forma iterativa, cada modelo se entrena con ejemplos que han sido mal predichos por los modelos anteriores.

En el planteamiento de *AdaBoost*, los modelos de la combinación también se van construyendo de uno en uno. Se asocia un peso a cada ejemplo de entrenamiento, inicialmente todos los ejemplos empiezan con el mismo peso. Para construir los modelos miembros de la combinación se tiene en cuenta el valor de los pesos. Una vez construido un modelo, se recalculan los pesos de los ejemplos antes de construir el siguiente. El objetivo es que el peso asociado a cada instancia se incremente en función del error obtenido en la iteración anterior para esa instancia. Por lo que los ejemplos con mayor error acumulado son más importantes en la construcción del siguiente modelo. Por otro lado, en función del error de cada modelo, se le asocia un peso para la predicción final de la combinación.

En el caso de regresión hay varias implementaciones de *AdaBoost*. Se ha trabajado con una versión implementada para trabajar en Weka (ver explicación del entorno para *Minería de Datos* Weka en el Apartado 4.5) que se denominó *AdaBoost.R2*, en la que la predicción de la combinación es el promedio ponde-

rado de las predicciones de sus miembros. Esta implementación permite dos estrategias de entrenamiento en función de los pesos, por un lado, tiene una opción que tiene en cuenta todos los ejemplos, cada uno con su peso (*reweighting*, en inglés), o por otro lado, tiene otra opción que selecciona una muestra de los ejemplos (una combinación con repetición del conjunto de datos) en la que los elementos con más peso tienen más probabilidad de aparecer (*resampling*, en inglés). Esta implementación permite también variar la función con la que se recalculan los pesos, permitiendo que sea lineal, cuadrática o exponencial.

Según un estudio de Zhang et al. [118], este método es el que obtiene uno de los mejores resultados promedio dentro de los métodos comparados.

3.3. *Random Subspaces*

Ho, en 1998 [53], plantea un método de combinación que denomina *Random Subspaces*.

En este modelo plantea entrenar a todos los modelos con el total de ejemplos del conjunto de datos de entrenamiento, pero, para obtener la diversidad, a cada modelo no le pasa los ejemplos completos con todos sus atributos, cada modelo es entrenado con un subespacio del dominio de los datos de origen. Para formar los subespacios se elige una combinación de los atributos, y sólo se pasan los datos de esos atributos. El número de atributos que se pasa a cada miembro se denomina dimensión del subespacio, y es un parámetro del método. Es habitual que la dimensión se establezca en función del número de atributos del conjunto de datos de partida en forma de un porcentaje. Por ejemplo si tenemos un conjunto de datos con d atributos y los subespacios son del 50 % de los atributos se pueden entrenar tantos modelos diferentes como combinaciones de d atributos de orden $\lfloor \frac{d}{2} \rfloor$.

La predicción final de la combinación*Random Subspaces* es el

promedio de las predicciones de sus miembros.

3.4. *Negative Correlation Learning*

En 1999, Liu and X. Yao proponen *Negative Correlation Learning* [73] en la que los miembros de la combinación se construyen en paralelo. Cada miembro es estrenado con una penalización, que es función de la similitud con el resto de los miembros de la combinación. En 2005, Brown et al. estudian como afecta a la diversidad de los miembros de la combinación [13].

3.5. *Random Forest*

Breiman, también en 2001 [9] propone otro método que denomina *Random Forest*.

Este algoritmo construye una combinación de árboles, a los que pasa todos los datos, la diversidad de los modelos la consigue con el tipo de árbol que utiliza para generar cada modelo. Para ellos usa el *Random tree*.

Random tree es un árbol de búsqueda que se entrena con todos los datos de entrenamiento pero en lugar de entrenarse buscando, en cada nodo del árbol, entre todos los atributos del conjunto de datos cual es en que mejor divide el conjunto en dos subconjuntos con menos entropía, lo hace evaluando sólo un subconjunto de atributos que elige al azar en cada nodo, esta característica aleatoria hace que puedan ser construidos muchos árboles distintos.

3.6. *Iterated Bagging*

Breiman, también en 2001 [10], vuelve a definir un nuevo método que denomina *Iterated Bagging*.

Este algoritmo se basa en generar varios *Bagging* de forma consecutiva, y combinar el resultado de todos ellos. Primero se construye un *Bagging* de forma normal, como se describió en el Apartado 3.1. En función de las predicciones de ese primer *Bagging*, denominado primera iteración, se cambia la variable a predecir por la siguiente iteración. Los valores de la variable a predecir en la siguiente iteración se denominan *residuos*, y se calculan como la diferencia entre el valor real y el predicho por las iteraciones previas. Pero para estas predicciones no se utilizan las predicciones de todos los miembros de los modelos *Bagging* completamente obtenidos, se descartan aquellos modelos que han usado el ejemplo evaluado durante el entrenamiento.

A partir de esa segunda iteración el valor de la variable a predecir en las siguientes se definen en función del conjunto de todas las predicciones previas. En problemas de regresión, la predicción de una combinación*Iterated Bagging* es la suma de las predicciones de los *Bagging* de cada iteración.

Según el trabajo de Suen et al. [104], *Iterated Bagging* es el método más efectivo en la mayor parte de las ocasiones.

3.7. *Rotation Forest*

Rodríguez et al., en 2006 [94], propusieron un algoritmo de combinación para clasificación que denominaron *Rotation Forest*.

El algoritmo se basa en entrenar todos los modelos del clasificador con el conjunto de datos de entrenamiento completo, pero previamente se ha aplicado una transformación de rotación diferente a los datos de cada modelo. El cálculo de las matrices de rotación a aplicar para en modelo es obtenido mediante la fusión de proyecciones resultantes de hacer un análisis de componentes principales a K muestras de datos diferentes creadas a partir de K subconjuntos de entrenamiento que se han generado

al repartir las diferentes características del conjunto de datos de entrenamiento.

Como predicción de la combinación se toma el promedio de las predicciones de los modelos obtenidos.

De la adaptación de este algoritmo para regresión se tratará en el Capítulo 6.

3.8. *Random Oracle*

En 2007, Kuncheva y Rodríguez [62] proponen un clasificador formado por una minicombinación con solo un par de clasificadores base y un «oráculo aleatorio» que elige entre ellos.

El oráculo se puede describir como una función discriminante aleatoria, que divide los datos en dos particiones sin tener en cuenta las características, las etiquetas o la distribución del conjunto de datos.

Las bases del algoritmo de entrenamiento de un *Random Oracle* son:

- Elegir una función al azar, el oráculo.
- Partir el conjunto de datos de entrenamiento en dos usando el oráculo.
- Construir un modelo para cada subconjunto de entrenamiento.

El modelo *Random Oracle* está formado por ese par de modelos y el oráculo. Para hacer la predicción final de la combinación se realizan los siguientes pasos:

- Se pregunta al oráculo que elija uno de los dos modelos.
- Se devuelve la predicción de ese modelo.

Un Random Oracle es una combinación de modelos y se puede usar como tal, pero la complejidad computacional del oráculo es baja, tanto en entrenamiento como en predicción, por lo que la complejidad del *Random Oracle* es comparable a la complejidad de un modelo base, En la fase de predicción solo se usa uno de los dos modelos, y en la fase de entrenamiento cada modelo entrena con una partición de los datos de entrenamiento, como el tiempo de entrenamiento de los métodos base crece, al menos de forma lineal, con el número de ejemplos de entrenamiento, el tiempo de entrenamiento de un *Random Oracle* será igual al del método base en el peor de los casos. Por todo lo anterior, en el Apartado 3.8 será tratado como un modelo base.

Capítulo 4

El entorno experimental

Uno de los problemas de elegir un algoritmo o de proponer uno nuevo, es cómo asegurar que el rendimiento de un algoritmo es mejor que otro, y tener evidencias experimentales de tal aseveración. En el Apartado 4.1 se tratará las medidas usadas en los experimentos para obtener el rendimiento de un algoritmo para un conjunto de datos dado. En el siguiente apartado (Apartado 4.2) veremos que para esa medida tenga cierta fiabilidad es necesario que el experimento se repita varias veces usando la técnica de validación cruzada y para que la medida no dependa de los datos utilizados se utilizan varios conjuntos de datos con diferentes características. En el Apartado 4.4 se describirán los datos con los que se ha trabajado y sus características.

Para analizar si las diferencias de rendimiento obtenidas son importantes (significativas) o no, se utilizan medidas de comparación (ver Apartado 4.3). Para automatizar los experimentos nos hemos ayudado de una herramienta informática de *Minería de Datos* (ver Apartado 4.5).

4.1. Rendimiento de un algoritmo

En los problemas de clasificación, el rendimiento de un algoritmo, para un conjunto de datos, se mide calculando la tasa

$$tasa\_Acierto = \frac{n_a}{n}$$

$$SAE = \sum_{i=1}^{n} |a_i - p_i| \qquad MAE = \sum_{i=1}^{n} \frac{|a_i - p_i|}{n}$$

$$SSE = \sum_{i=1}^{n} (a_i - p_i)^2 \qquad RMSE = \sqrt{\sum_{i=1}^{n} \frac{(a_i - p_i)^2}{n}}$$

Figura 4.1: Medidas rendimiento de un modelo.
Donde n es el nº de ejemplos, n_a es el nº de ejemplos correctamente clasificados, a_i es el valor objetivo a predecir del ejemplo i-ésimo y p_i el valor predicho.

de acierto (ver Figura 4.1). Mientras que lo más habitual en los problemas de regresión, es medir la falta de rendimiento calculando la media cuadrática del error (RMSE), considerando error la diferencia entre el valor predicho y el objetivo a predecir. En la Figura 4.1 se pueden ver las fórmulas de esta y otras medidas del error como la suma del valor absoluto de los errores (SAE), la suma de los cuadrados de los errores (SSE), la media aritmética del valor absoluto de los errores (MAE).

4.2. Validación cruzada

La validación cruzada $\alpha \times \beta$ (*cross-validation*, en inglés) es una técnica que divide el conjunto de datos de entrenamiento en β partes 'iguales', que denominaremos particiones (*folds*). El modelo es entrenado con los datos de $\beta - 1$ de esas particiones y luego se determina la tasa de acierto (o error) con los datos de la partición que no ha usado para entrenar. Este proceso se repite β veces, cambiando cuál es la partición que se deja fuera del proceso de entrenamiento y se reserva para la fase de test. El proceso de dividir el conjunto de datos se repite α veces, por lo que al final se ha repetido el proceso $\alpha \times \beta$ veces. Con las $\alpha \times \beta$

tasa de acierto (o error) obtenidas se calculará un promedio de error para ese conjunto de datos.

Para validar un algoritmo se pueden encontrar experimentos que, buscando tener una buena estimación del rendimiento, repiten 100 veces el proceso usando validación cruzada 10×10 particiones, y otros que repiten 10 veces el proceso usando validación cruzada 5×2 particiones. Dietterich propone en [19] que los resultados de esta última configuración pueden tener, estadísticamente hablando, suficiente valor científico.

Cuando se fuerza a que las particiones tengan propiedades estadísticas cercanas al conjunto de datos original se habla de particiones estratificadas, éstas son normalmente necesarias para el correcto funcionamiento de algunos algoritmos de clasificación.

4.3. Métodos de comparación

Para poder comparar el rendimiento se suelen usar tablas de clasificaciones o algún test estadístico.

4.3.1. Tabla de clasificaciones promedio

Para poder comparar el rendimiento de varios algoritmos se suele crear una tabla de clasificación (en inglés *ranking*) [17] ordenándoles en función de tasa de acierto, el que más acierta primero, o de su error, el que menos yerra el primero.

Para cada conjunto de datos se construye una tabla de clasificación en la que se asigna a cada algoritmo una posición en función de su rendimiento obtenido en ese conjunto de datos. La primera posición de la tabla es para el mejor algoritmo, al siguiente algoritmo se le asigna la posición 2 y así de forma sucesiva. Si el rendimiento de varios algoritmos es el mismo, se les asigna a todos el valor promedio de las posiciones que en las están empatados –por ejemplo se asigna la posición 2,5 a cuatro

algoritmos que queden empatados con el mejor rendimiento, es decir, que ocupan indistintamente las posiciones entre la primera y la cuarta.

Una vez conocida la posición de cada algoritmo en todos los conjuntos de datos considerados, se hace una nueva tabla en la que se ordenan los algoritmos por el promedio de esas posiciones. Según Demšar [17] la tabla de clasificaciones promedio es una forma justa de comparar algoritmos («*average ranks by themselves provide a fair comparison of the algorithms*»).

4.3.2. t-test estadístico corregido remuestreado

El *t*-test estadístico corregido remuestreado (*resampled t-test*, en inglés) [81] es una prueba que determina cuando una diferencia entre el resultado en un conjunto de datos de dos algoritmos es significativa o, por el contrario, la ventaja es dudosa, al ser posible por azar.

Este test está basado en el test de *Student* que se emplea cuando se dispone de muestras de población de pequeño tamaño, y necesita estimar la varianza de una muestra considerando que el grado de libertad es el de muestras menos uno.

Este test estima que una medida es significativa en función de lo alejada que esté de la media de una distribución normal, el porcentaje de valores que se alejan más de n veces la varianza se puede ver en la Tabla 4.1.

Como población se tiene el conjunto de las diferencias entre las predicciones de dos regresores que han usado las mismas particiones de entrenamiento y test –un conjunto de 10 valores, si se ha usado validación cruzada 5×2 particiones, o de 100, si se ha usado validación cruzada 10×10 particiones– pero estas medidas no se pueden suponer independientes entre sí, por lo que para poder dar como significativas diferencias hay que meter un factor que aumente el cociente de repeticiones.

Tabla 4.1: Porcentaje de valores dentro de una distancia a la media en una distribución normal.

d	en el rango $[-d, d]$	en el rango $[d, \infty)$
σ	68,26 %	15,87 %
$1{,}28\sigma$	79,94 %	10,03 %
$1{,}29\sigma$	80,30 %	9,85 %
$1{,}64\sigma$	89,90 %	5,05 %
$1{,}65\sigma$	90,10 %	4,95 %
2σ	95,44 %	2,28 %
3σ	99,74 %	0,13 %

La medida de significación corregida se evalúa como:

$$t = \frac{\bar{d}}{\sigma_d \sqrt{\frac{1}{n_r} + \frac{n_t}{n_e}}}$$

donde

- $\bar{d}$ es el valor medio de la diferencia de predicciones de los dos modelos,
- σ_d es la desviación típica de esa muestra,
- n_r es el número de predicciones que tenemos,
- n_t es el número de datos de test,
- n_e es el número de datos de entrenamiento.

Si trabajamos con validación cruzada 5×2 particiones, tenemos 10 predicciones, y la ratio entre datos de test y datos de entrenamiento es de 1, y si trabajamos con validación cruzada 10×10 particiones, tenemos 100 predicciones, y la ratio de datos de test y datos de entrenamiento es de 1 por cada 9, con lo que la expresión de t, dependiendo de cada caso queda:

$$t = \begin{cases} \dfrac{\bar{d}}{1{,}0488 \cdot \sigma_d} & |\text{v.c.}5\times 2 \\ \dfrac{\bar{d}}{0{,}348 \cdot \sigma_d} & |\text{v.c.}10\times 10 \end{cases}$$

Tabla 4.2: Victorias necesarias para tener 0,05 de significación en el test de signo.

nº de conjuntos	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
min.de victorias	5	6	7	7	8	9	9	10	10	11	12	12	13	13	14	15	15	16	17	18	18

En la tesis usaremos un nivel de significación t de 0,05.

4.3.3. Test de signo

El test de signo es un método estadístico para comparar algoritmos de dos en dos. Es útil para comparar algoritmos cuando, en los experimentos, se usa una gran cantidad de conjuntos de datos. Este test propone que no puede ser sólo por azar que un algoritmo gane a otro en muchos conjuntos de datos, aunque no lo gane significativamente en ninguno.

Según Demšar [17], usando un test de signo, un método es significativamente mejor que otro, con un nivel de significación del 0,05, si el número de victorias que supera un umbral que es función del número de conjuntos de datos usados en los experimentos. Demšar enumera el umbral de victorias hasta 25 conjuntos de datos (ver Tabla 4.2) y si se trabaja con más conjuntos de datos propone la fórmula:

$$victorias \geq \frac{N + 1{,}96\sqrt{N}}{2}$$

En la Figura 4.2 se puede ver la evolución del tanto por ciento de conjuntos de datos en los que debe obtener victoria un algoritmo para asegurar que es mejor que otro, con una significación del 0,05.

4.4. Los conjuntos de datos utilizados

Para que el resultado de un estudio sea suficientemente significativo se ha de utilizar una amplia selección de conjuntos

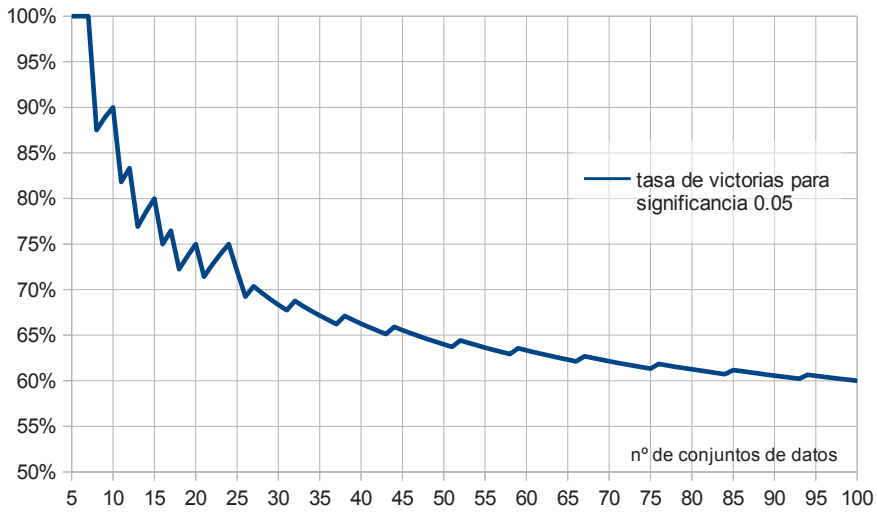

Figura 4.2: Tasa de victorias necesarias para tener 0,05 de significación.

de datos, en concreto en este trabajo se han utilizado los 61 conjuntos que se muestran en las tablas 4.3 y 4.4. Todos estos conjuntos de datos tienen la característica de estar disponibles para ser usados directamente en el formato de Weka[1], los 29 de la Tabla 4.3 han sido preparados por Luís Torgo[2].

La naturaleza de estos conjuntos de datos es bastante variada. Tienen un promedio de casi cinco mil trescientos ejemplos, hay desde conjuntos pequeños, con solo trece ejemplos, hasta otros grandes, con más de cuarenta mil ejemplos, la mediana está en 286 ejemplos. Ocupan un promedio de más de $\frac{1}{2}$ MB, hay conjuntos cuyo tamaño varía entre 1 KB y 4 MB, la mediana está en 17 KB. En cuanto al número de características, éstas varían entre dos y sesenta atributos (con un promedio de 13 atributos), la mayoría de estos atributos son numéricos ya que entre ninguno y once son atributos nominales mientras que entre ninguno y el máximo de sesenta son atributos numéricos. De estos conjuntos de datos 34 solo tienen atributos numéricos, dos de los conjuntos sólo tienen atributos nominales, el resto (25)

[1] `http://www.cs.waikato.ac.nz/ml/weka/index_datasets.html`
[2] `http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html`

Tabla 4.3: Los conjuntos de datos de la UCI usados en los experimentos, que fueron preparados por Luís Torgo.

Tamaño (bytes)	nº de ejemplos	nº atributos numéricos	nominales	conjunto de datos
1 454	43	2	–	diabetes
15 958	74	27	–	pyrim
16 532	159	14	1	autoPrice
5 393	167	–	4	servo
81 983	186	60	–	triazines
49 242	194	32	–	wisconsin
6 038	209	6	–	machineCpu
13 309	398	4	3	autoMpg
37 128	506	12	1	housing
57 008	950	9	–	stock
197 741	4 177	7	1	abalone
309 015	7 129	5	–	deltaAilerons
580 622	8 192	12	–	cpuSmall
690 435	8 192	8	–	bank8FM
701 353	8 192	8	–	puma8NH
1 058 076	8 192	21	–	cpuAct
1 131 186	8 192	8	–	kin8nm
2 505 882	8 192	32	–	bank32nh
2 806 314	8 192	32	–	puma32H
362 585	9 517	6	–	deltaElevators
3 740 675	13 750	40	–	ailerons
1 726 623	15 000	48	–	pol
2 202 688	16 599	18	–	elevators
2 073 848	20 460	8	–	calHousing
2 378 793	22 784	8	–	house8L
4 393 861	22 784	16	–	house16H
1 308 027	40 768	10	–	2dplanes
2 724 449	40 768	10	–	fried
3 746 008	40 768	7	3	mv

Tabla 4.4: Otros conjuntos de datos de la UCI usados en los experimentos.

Tamaño (bytes)	nº de ejemplos	nº atributos numéricos	nº atributos nominales	conjunto de datos
3 598	13	13	–	detroit
1 387	16	6	–	longley
1 182	27	4	–	gascons
1 622	38	4	1	schlvote
4 534	40	7	–	bolts
1 155	52	3	–	vineyard
1 056	55	1	1	elusage
6 413	60	15	–	pollution
1 009	61	1	1	mbagrade
3 826	62	7	–	sleep
13 617	93	16	6	auto93
2 968	96	4	–	baskball
4 550	108	4	2	cloud
8 722	125	2	2	fruitfly
10 532	130	6	3	echoMonths
3 473	137	3	4	veteran
9 050	158	5	2	fishcatch
9 038	189	2	7	lowbwt
13 447	195	1	10	pharynx
6 266	200	10	–	pwLinear
29 997	205	15	10	autoHorse
10 182	209	6	1	cpu
24 433	252	14	–	bodyfat
16 955	286	1	8	breastTumor
21 224	294	6	7	hungarian
21 923	303	6	7	cleveland
22 358	303	6	7	cholesterol
32 916	418	10	8	pbc
72 762	528	19	2	meta
18 879	576	–	11	sensory
18 628	625	5	1	strike
44 441	2 178	3	–	quake

tiene de los dos tipos.

En la Figura 4.3 se puede ver representada esta misma información de las tablas 4.3 y 4.4, pero resumida en un formato gráfico; cada burbuja representa uno de los conjunto de datos, la posición del centro de dicha burbuja indica el volumen del conjunto de datos, el número de ejemplos (eje X en escala logarítmica) y el tamaño (eje Y también en escala logarítmica); el tamaño de la burbuja es función del número de atributos y el color representa la relación entre atributos nominales (corazón rojo de la burbuja) con respecto a los numéricos (corona circular azul).

El dominio de conjuntos de datos también es muy variado, dado que provienen de diferentes orígenes, por ejemplo, hay datos creados artificialmente: 2dplanes, bank, fried, mv; datos de reacciones químicas: pyrim; y del clima: cloud, pollution; datos de distintas máquinas: ailerons, elevators, auto, bolts, cpu, kin8nm, puma, servo, pol; datos de la vida animal: abalone, fishcatch, fruitfly, sleep; datos sociales: baskball, detroit, elusage, gascons, housing, house, mbagrade, schlvote, strike; datos médicos: bodyfat, wisconsin, breast tumor[3], cholesterol, cleveland, hungarian[4], diabetes, echo months, lowbwt, pbc, pharynx.

4.5. Weka

Para la realización de los experimentos se ha utilizado Weka [46].

[3] Los datos del cáncer de mama, disponibles en la UCI [71], fueron originalmente recopilados por los equipos de M. Zwitter and M. Soklic en el *University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia.*

[4] Los datos de diagnostico de enfermedades del corazón, disponibles en la UCI [71], fueron originalmente recopilados por los equipos de Andras Janosi, M.D., William Steinbrunn, M.D., Matthias Pfisterer, M.D. y Robert Detrano, M.D., Ph.D. en el *Hungarian Institute of Cardiology, Budapest*, el *University Hospital, Zurich*, el *University Hospital, Basel* y el *V.A. Medical Center, Long Beach and Cleveland Clinic Foundation.*

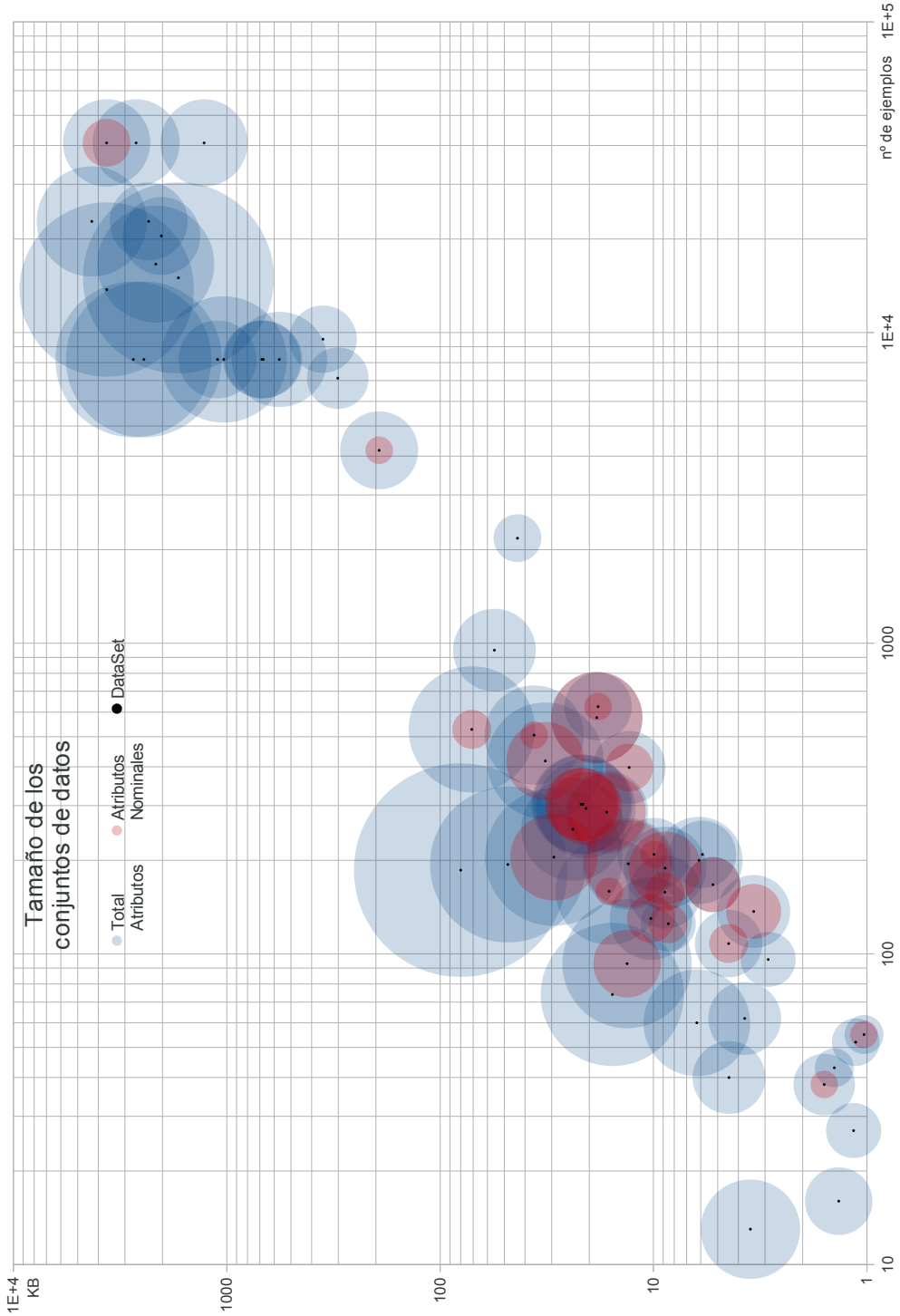

Figura 4.3: Gráfico de los tamaño de los conjuntos de datos

Weka es un programa de software libre para *Minería de Datos*, publicado con Licencia Pública General (GNU)[5].

Está desarrollado en lenguaje Java por el *Machine Learning Group* de la Universidad de Waikato[6], por lo que se puede usar múltiples plataformas. Weka tiene una colección de algoritmos de aprendizaje automático para tareas de *Minería de Datos* que hay ido aumentado desde la primera versión que se publicó en 1999. Los algoritmos de esa colección se pueden aplicar directamente a un conjunto de datos, también puede ejecutar algoritmos propios programados en Java. Todos estos algoritmos también pueden ser integrados dentro de programas externos.

Weka contiene herramientas para las distintas fases del proceso de *Minería de Datos*: el preprocesamiento de los datos, la clasificación de instancias, los cálculos de regresión, el *clustering* de datos, la clasificación por reglas de asociación, la validación cruzada y el análisis y la visualización de los resultados.

[5] Se agradece a los desarrolladores de Weka, y de las implementaciones de los diferentes algoritmos por distribuirlos.

[6] http://www.cs.waikato.ac.nz/ml/

Capítulo 5

Estudios experimentales del desempeño de las combinaciones de regresores base

En este capítulo se presentarán los resultados experimentales al comparar el funcionamientos de diversas combinaciones homogéneas de métodos, cuando todas usan el mismo método base.

En el Apartado 5.1 se comparan combinaciones de perceptrones multicapa, en el Apartado 5.2 se comparan combinaciones del *Random Oracle*, en el Apartado 5.3 se comparan combinaciones de redes de funciones de base radial y en el Apartado 5.4 se comparan combinaciones de máquinas de soporte vectorial; se dejará para un capítulo aparte las combinaciones de *árboles de regresión*, en que se se analizará el funcionamiento de la adaptación a regresión del *Rotation Forest*.

5.1. Estudio experimental de combinaciones de perceptrones multicapa para tareas de regresión

Los resultados del estudio experimental de este apartado han dado lugar a un artículo presentado en un congreso internacional [86].

En este apartado se compara el funcionamiento, para regresión, de varios algoritmos de combinaciones de métodos, usando perceptrones multicapa como método base y 61 conjuntos de datos. Se han comparado *Bagging* [8], *Random Subspaces* [53], *AdaBoost* [33, 23], *Iterated Bagging* [10]. Se añadieron algunos otros a la comparación por estar disponibles en Weka, pero no todos, por ejemplo, se descartó usar *Negative Correlation Learning* [13], porque se necesitaba modificar el método base para que este entrenara usando una penalización función de la similitud de la red actual con la combinación.

En el Apartado 5.1.1 se darán los datos de configuración del experimento, el Apartado 5.1.2 mostrará los resultados, el Apartado 5.1.3 se dedicará a presentar los diagramas del error y la diversidad. Para acabar, se expondrán unas conclusiones en el Apartado 5.1.4.

5.1.1. Diseño experimental

Los experimentos se han realizado usando una validación cruzada 5×2 particiones [19]. Los resultados de los diferentes métodos sobre distintos conjuntos de datos se han medido usando la *media cuadrática*[1] *de los errores* (RMSE[2]). Se ha utilizado como método base perceptrones multicapa y un tamaño de las

[1] La media cuadrática de un conjunto de valores es la raíz cuadrada de la media de los cuadrados de esos valores

[2] *Root of Mean Squared Error*, en inglés

combinaciones de 50 modelos.

Se han considerado varios métodos de combinación:

- *Aleatorización* (en inglés *Randomization*). Cuando el método base tiene un elemento aleatorio, cada modelo se obtiene con los mismos conjuntos de datos. En el caso de los perceptrones multicapa, se inician de forma aleatoria los pesos iniciales. La predicción de este método es la media de la predicción de los miembros de la combinación.
- *Bagging* (ver Apartado 3.1).
- *Random Subspaces* (ver Apartado 3.3). Se han considerado dos tamaños de subespacios. Subespacios con el 50 % del número de atributos y con el 75 %.
- *AdaBoost.R2* (ver Apartado 3.2). Los métodos basados en *AdaBoost* pueden usarse de dos formas [34]. La versión con repesado y la versión con remuestreo, que se diferencian con los sufijos «-W» and «-S», en las tablas y discusiones de este apartado. Además este método puede ser usado con diferentes funciones de pérdida. Se han usado las tres que se proponen en [23]: lineal, cuadrática y exponencial. Para diferenciar la función usada, en las tablas, se han usado los sufijos «-L», «-Q» and «-E».
- *Iterated Bagging* (ver Apartado 3.6). Se ha usado una configuración de 5×10: *Bagging* se itera 5 veces, el número de modelos de cada *Bagging* es 10.

Se han incluido, además, otros métodos en el estudio, como base de las comparaciones:

- Un único perceptrón multicapa (ver Apartado 2.3).
- La *regresión lineal*. Se han estudiado dos versiones: una usando todas las características y otra usando solo una selección de características con el método descrito en [109].

- Los *vecinos más cercanos* (ver Apartado 2.6). Se han usado dos versiones, en una de ella el número de vecinos es uno. En la otra, el número de vecinos se selecciona usando validación cruzada «dejando uno fuera».

Se ha usado Weka [115] para los experimentos. El método base (perceptrones multicapa) ya lo tiene implementado, así como *Bagging* y *Random Subspaces*. El resto de los métodos se han implementado.

Para perceptrones multicapa se ha usado la configuración por defecto de Weka. Con una capa oculta, y la mitad de neuronas que atributos, incluyendo el atributo a predecir. La tasa de aprendizaje es 0.3, el momento 0.2 y el número de iteraciones es 500.

Las tablas 4.3 y 4.4 muestran las características de los 61 conjuntos de datos considerados en el estudio.

5.1.2. Resultados

Para poder comparar todas las configuraciones consideradas se ha construido una tabla de clasificación (en inglés *ranking*) en función de los resultados del error [17].

Se han ordenado los algoritmos en función de sus resultados en cada conjunto de datos. La primera posición es para el mejor algoritmo, al siguiente algoritmo se le asigna la posición 2 y así de forma sucesiva. Si varios algoritmos quedan empatados con el mismo resultado, se les asigna a todos el valor promedio de las posiciones que están empatadas (p.ej. se asignará la posición 2,5 a cuatro algoritmos que queden empatados en el mejor resultado, es decir en las posiciones 1, 2, 3 y 4). Para cada algoritmo, se ha calculado el promedio de posiciones en todos los conjuntos de datos considerados. Según [17] es una forma justa de comparar algoritmos[3]. La Tabla 5.1 muestra los algoritmos ordenados por

[3] «*average ranks by themselves provide a fair comparison of the algorithms*»

Tabla 5.1: Algoritmos ordenados por su posición promedio.

#	Puesto promedio	Método
1	4,61	*Bagging* MLP
2	6,28	*Random Subspaces* 75 % MLP
3	6,74	*AdaBoost.R2-S*-L MLP
4	7,12	*AdaBoost.R2-S*-Q MLP
5	7,21	*AdaBoost.R2-S*-E MLP
6	7,30	*Random Subspaces* 50 % MLP
7	7,35	*k vecinos más cercanos*
8	7,37	*Iterated Bagging* MLP
9	8,42	*regresión lineal*(todos)
10	8,55	*regresión lineal*(selección)
11	8,63	*Aleatorización* MLP
12	9,48	*AdaBoost.R2-W*-E MLP
13	10,69	*AdaBoost.R2-W*-L MLP
14	11,82	un MLP
15	11,82	*AdaBoost.R2-W*-Q MLP
16	12,62	1 *vecino más cercano*

su promedio de posiciones.

Un único perceptrón multicapa y la solución de 1 *vecino más cercano* tienen peor puesto promedio que la solución mediante *regresión lineal*. Incluso, una combinación de perceptrones multicapa basada sólo en la inicialización aleatoria de los pesos es también peor que esos métodos básicos. Sin embargo, otras combinaciones de perceptrones multicapa tienen mejores puestos promedios que los métodos básicos.

En mejor puesto promedio es para *Bagging*. El segundo puesto es para *Random Subspaces*, usando 75 % de los atributos. La siguiente posición es para la versión de *AdaBoost.R2* que usa remuestreo. Por otro lado los resultados de la versión que repesa las instancias son peores que los métodos básicos directamente.

La Tabla 5.2 muestra la comparación directa de un único perceptrón multicapa y *Bagging* con los otros algoritmos En cada fila se muestra el número de victorias, empates y derrotas

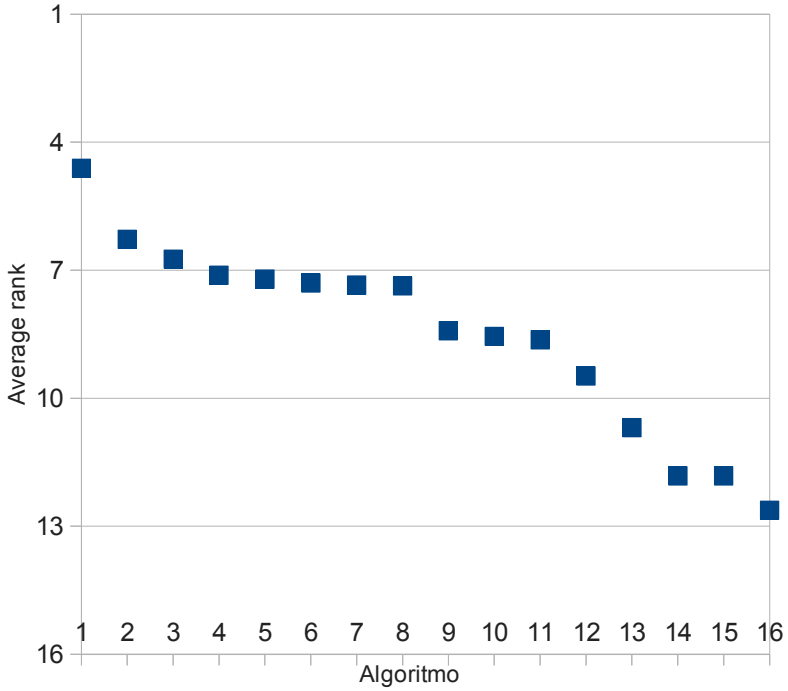

Figura 5.1: Gráfico de las posiciones promedio de la Tabla 5.1.

de cada uno de estos dos algoritmos al enfrentase con otro.

Cuando se comparan dos algoritmos, se calcula el número de conjuntos de datos en los que se es mejor, igual o peor. Según [17], usando un test de signo, un método es significativamente mejor que otro, con una seguridad del 0,05, si el número de victorias más la mitad de los empates es mayor que $(N + 1{,}96\sqrt{N})/2$. Para nuestro caso con $N = 61$ conjuntos de datos, el número de victorias es 39.

En número de victorias, empates, derrotas y el puesto promedio se han calculado usando una comparación directa de los resultados de los diferentes algoritmos. Sin embargo, esto no tiene en cuenta la magnitud de las diferencias. Por ello, se ha usado *quantitative scoring* [100, 118] para analizar las diferencias cuantitativas. Dados los resultados de dos algoritmos i y j para un

Tabla 5.2: Comparación entre un único perceptrón multicapa y *Bagging* con el resto de los algoritmos.

un MLP			*Bagging*			
G	E	P	G	E	P	al algoritmo
54	0	7	4	0	**57**	*Iterated Bagging* MLP
0	**61**	0	3	0	**58**	un MLP
60	0	1	5	0	**56**	*Aleatorización* MLP
23	0	**38**	5	0	**56**	1 *vecino más cercano*
39	0	22	5	0	**56**	*AdaBoost.R2-W*-L MLP
35	0	26	7	0	**54**	*AdaBoost.R2-W*-Q MLP
41	0	20	14	0	**47**	*AdaBoost.R2-W*-E MLP
34	0	27	19	0	**42**	*regresión lineal*
50	0	11	20	0	**41**	*AdaBoost.R2-S*-E MLP
36	0	25	21	0	**40**	*regresión lineal*(selection)
48	1	12	22	0	**39**	*AdaBoost.R2-S*-L MLP
48	0	13	22	0	**39**	*AdaBoost.R2-S*-Q MLP
52	0	9	23	0	**38**	*Random Subspaces* 75 % MLP
42	0	19	23	0	**38**	*k vecinos más cercanos*
39	1	21	27	0	**34**	*Random Subspaces* 50 % MLP
58	0	3	0	**61**	0	*Bagging* MLP

conjunto de datos, el resultado entre ellos se define como:

$$S_{i,j} = \frac{RMSE_j - RMSE_i}{\text{máx}(RMSE_i, RMSE_j)}$$

Donde $RMSE_i$ es la media cuadrática del error del método i. Salvo que ambos métodos tengan un error 0, esta medida se encuentra en el intervalo entre ± 1, por lo que también puede expresarse como un porcentaje, en el que el signo indica qué algoritmo es mejor.

La Figura 5.2 muestra estos resultados en escala porcentajes, comparando con *Bagging* con el resto de los algoritmos considerados. Se ha calculado el resultado para cada conjunto de datos y luego se se han ordenado los conjuntos de datos por su resultado. El número de valores positivos, por encima del eje de abscisas, se corresponden con el número de victorias, y los valores negativos, por debajo del eje, con las derrotas de la Tabla 5.2.

Cuando se está desarrollando o eligiendo un algoritmo y se compara con otros algoritmos, usando estos gráficos, se busca que haya más valores positivos que negativos, y también se busca que el valor absoluto de las diferencias positivas sea mayor que el de los negativas. En este caso, se cumplen las dos condiciones, hay más casos positivos y los valores absolutos de las diferencias son superiores.

El resultado de *Iterated Bagging* es claramente peor que el de *Bagging*. Esto se puede achacar a la configuración elegida, 5 iteraciones de *Bagging* con 10 modelos. Esta configuración se ha elegido por que se deseaba comparar combinaciones con el mismo número de modelos bases, 50. Pero el resultado de iterar 5 veces *Bagging* con sólo 10 modelos, no es lo mimo que una configuración de *Bagging* con 50 modelos base. Se puede suponer que *Iterated Bagging* podría mejorar los resultados de *Bagging* si se le permite usar más modelos base en cada iteración, por ejemplo usando 50 modelos.

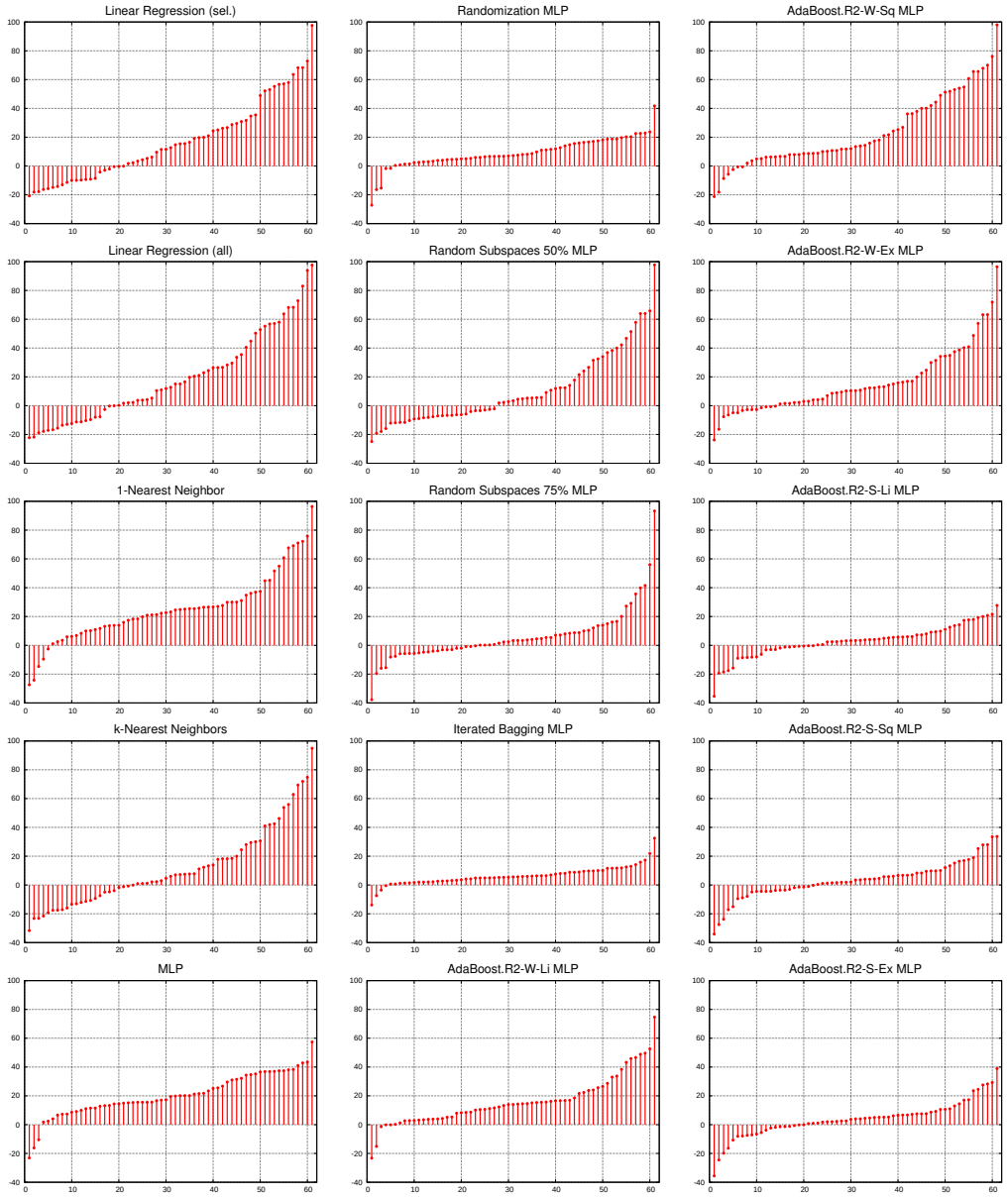

Figura 5.2: Comparación de resultados entre los algoritmos y *Bagging* de MLP.

5.1.3. Diagramas de diversidad-error

El éxito de los algoritmos de combinaciones de modelos es estar formados por modelos con poco error y que sean diferentes entre ellos, para compensarse. Estos objetivos suelen estar en conflicto, por que si dos modelos tienen poco error, no pueden ser muy diferentes. Se han propuestos diferentes medidas de la diversidad para analizar el comportamiento de algoritmos de combinaciones [61].

Una de las técnicas usadas son los diagramas de diversidad-error [77]. Son gráficos de dispersión en los que hay un punto para cada par de modelos. El eje horizontal representa la diversidad entre los dos modelos, para la clasificación, se utiliza por lo general κ (kappa). El eje vertical representa el error medio de los dos modelos.

En regresión, se pueden usar muchas medidas, en este momento usaremos la media cuadrática del error (RMSE) entre el valor predicho y el objetivo a predecir:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(a_i - p_i)^2}{n}}$$

Donde a_i es el valor objetivo a predecir y p_i el valor predicho.

Para medir la diversidad, se usará la RMSE entre los valores predichos por un modelo con respecto a los del otro:

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(q_i - p_i)^2}{n}}$$

Donde p_i y q_i son las predicciones de los dos modelos. Se hace notar que esta medida es creciente con la diversidad, mientras que el κ decrece con la diversidad.

En la figuras 5.3 y 5.4 se pueden observar los diagramas error-diversidad de los tres métodos mejores de la tabla de clasificación de puestos promedio, para los conjuntos de datos con más

ejemplos. En general, las combinaciones que usan *Bagging* tienen menos error, pero también menos diversidad. Parece que, en este caso, el mayor error de los miembros generados por las combinaciones*Random Subspaces* y *AdaBoost.R2* no puede ser compensado por el incremento de la diversidad que añaden.

5.1.4. Conclusiones

Se ha estudiado el rendimiento de los algoritmos de combinaciones de métodos aplicadas a regresión, usando como algoritmo base el perceptrón multicapa . Se ha comparado la combinación de algoritmos mediante *Aleatorización*, *Random Subspaces*, *Bagging*, *Iterated Bagging* y *AdaBoost.R2*.

El método con mejor posición promedio es *Bagging*, seguido de *Random Subspaces* (con subespacios de tamaño del 75 % del espacio original). Estos resultados difieren de estudios previos [104, 118, 95], donde resultaban ganadores *Iterated Bagging* o *AdaBoost.R2*. Una posible causa de estas discrepancias puede ser haber usado conjuntos de datos diferentes, pero [95] usa los mismos conjuntos de datos que este trabajo. Otra de las causas puede ser el modelo base usado, perceptrones multicapa en lugar de los *árboles de regresión*.

Estos resultados se han obtenido con los valores predefinidos. El tamaño de la combinación ha sido la misma en todos los casos (50), el perceptrón multicapa con los parámetros por defecto, los tamaños de los subespacios (50 % or 75 %) para el *Random Subspaces*, ... Los valores de algunos parámetros pueden modificarse, algunos incluso incrementado el tiempo de cálculo de forma significativa. Sin embargo, la comparación se puede considerar justa ya que se ha utilizado en todos los casos el mismo modelo base y se han utilizado combinaciones con el mismo número de modelos base.

Se ha visualizado el comportamiento de los miembros de

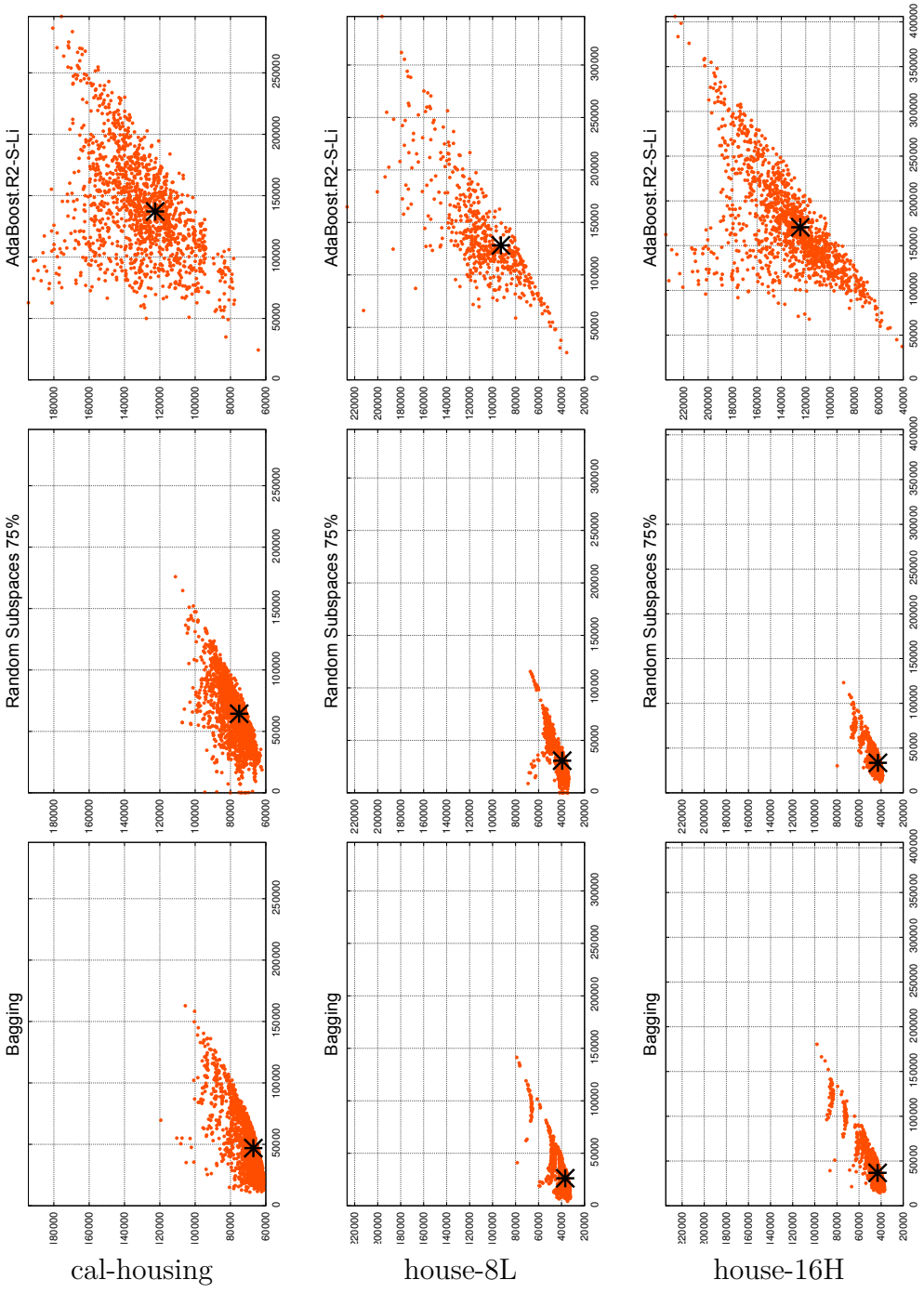

Figura 5.3: Diagramas de error-diversidad de combinaciones de MLP (conjuntos con alrededor de $2 \cdot 10^4$ ejemplos)

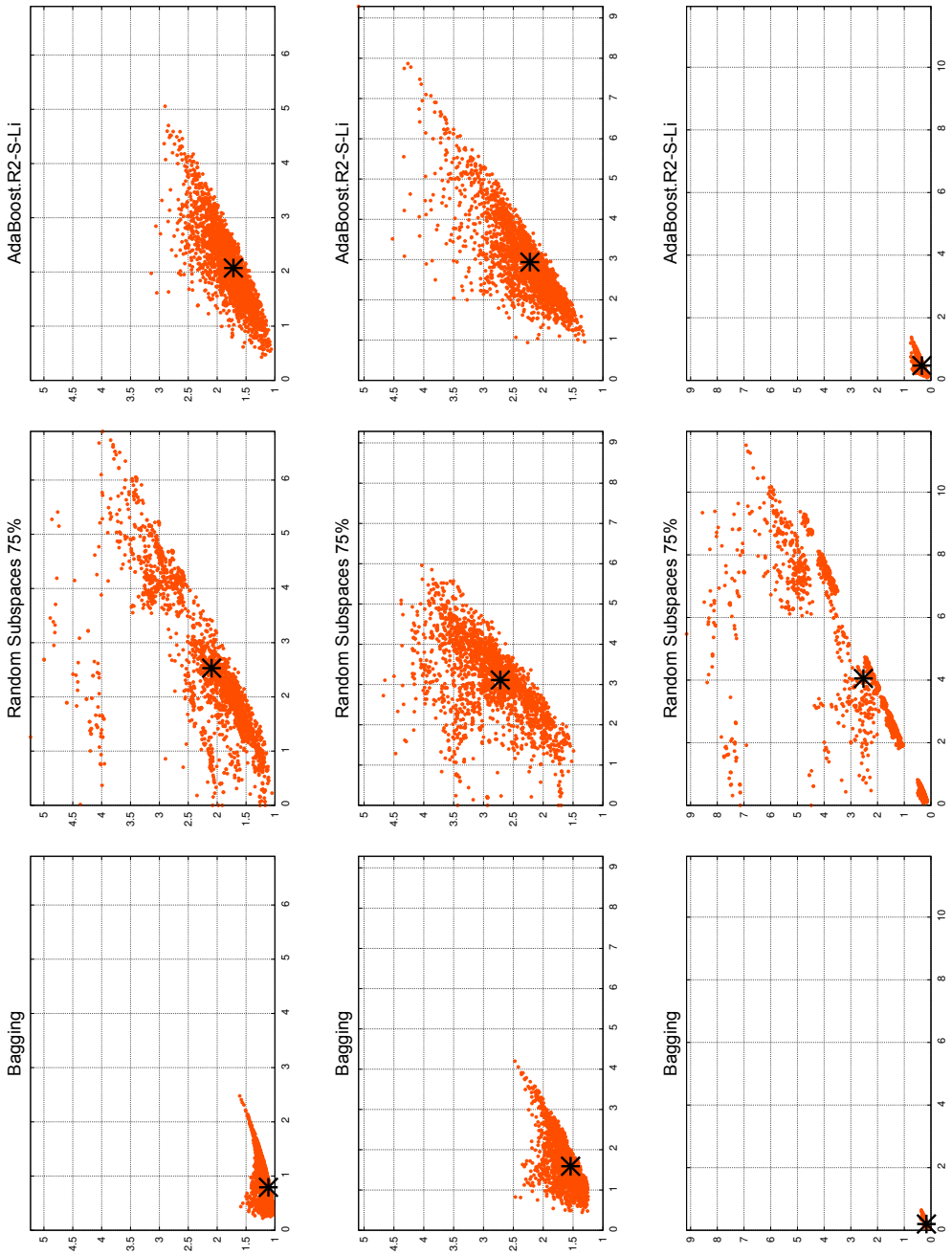

Figura 5.4: Diagramas de diversidad-error de combinaciones de MLP (conjuntos con alrededor de $4 \cdot 10^4$ ejemplos)

las combinaciones usando diagramas de diversidad-error. Parece que es más importante tener miembros con pocos errores que mucha diversidad, ya que es el caso del método con mejores resultados, *Bagging*, y coincide con los resultados publicados por otros investigadores, como por ejemplo [20, 21].

Bagging, a pesar de ser uno de los tipos de combinaciones más simples, es el que mejor resultado ha obtenido, por lo tanto, se plantea la pregunta de si hay otros métodos de combinaciones más adecuados para combinar perceptrones multicapa para tareas de regresión.

Dada la disparidad de los resultados para los algoritmos de combinaciones de métodos al utilizar árboles y perceptrones multicapa como modelos base, parece interesante estudiar el comportamiento de las combinaciones de otros modelos base, especialmente otros tipos de redes neuronales, como redes de funciones de base radial (RBF), en el Apartado 5.3 se presenta un estudio que utiliza este tipo de redes neuronales.

5.2. Estudio experimental de combinaciones usando *Random Oracle* para tareas de regresión

Los resultados del estudio experimental de este apartado han dado lugar a un trabajo que se presentó en un congreso internacional en 2010 [84] cuya ampliación dio lugar a un capítulo de libro en 2011 [85].

En este apartado se compara el funcionamiento para regresión con 61 conjuntos de datos, de la adaptación que se ha hecho para regresión del *Random Oracle*. Y se compara su funcionamiento tanto usándolo como algoritmo de combinaciones modelos, como su influencia en el comportamiento de otros algoritmos de combinaciones de modelos si se usan el *Random Oracle* como

método base (intermedio).

Se han comparado *Bagging* [8], *Random Subspaces* [53], *AdaBoost* [33, 23], *Iterated Bagging* [10]. Se añadieron a la comparación algunos otros, como *Aleatorización*, por estar disponibles en Weka.

En el Apartado 5.2.1 se darán los datos de configuración del experimento. El Apartado 5.2.2 mostrará los resultados, el Apartado 5.2.3 se dedicará a los diagramas del error y la diversidad. Para acabar, se expondrán con unas conclusiones en el Apartado 5.2.4.

5.2.1. Diseño experimental

Los experimentos se han realizado usando una validación cruzada 5×2 particiones [19]. Los resultados de los diferentes métodos sobre distintos conjuntos de datos se han medido usando la *media cuadrática de los errores* (RMSE[4]).

Los modelos base utilizados han sido *árboles de regresión*. Se han usado dos versiones, una sin podar (U) y otra podada (P), para lo que se ha usado la técnica de poda propuesta en [25].

Se han considerado varios métodos de combinación con 100 modelos cada una:

- *Aleatorización* (en inglés *Randomization*). Esta combinación necesita que el método base tenga un elemento aleatorio, para que cada modelo que se obtenga con el mismo conjunto de datos pueda ser diferente. La predicción de este algoritmo es la media de la predicción de los miembros de la combinación. Este algoritmo se ha usado sólo con los *árboles de regresión* podados, dado que permiten distribuir los datos de entrenamiento en dos particiones aleatorias, una para crear el árbol y otra para podar; en el caso de

[4] *Root of Mean Squared Error*, en inglés

los *árboles de regresión* sin poda no es posible al no haber elemento aleatorio.

- *Bagging* (ver Apartado 3.1).
- *Random Subspaces* (ver Apartado 3.3). Se han considerado dos tamaños de subespacios. Subespacios con el 50 % del número de atributos y con el 75 %.
- *AdaBoost.R2* (ver Apartado 3.2). Los métodos basados en *AdaBoost* pueden usarse de dos formas [34]. La versión con repesado y la versión con remuestreo, que se diferencian con los sufijos «-W» and «-S». Además, este método puede ser usado con diferentes funciones de pérdida. Se han usado las tres que se proponen en [23]: lineal, cuadrática y exponencial. Se ha usado los sufijos «-Li», «-Sq» and «-Ex» para diferenciar la función usada.
- *Iterated Bagging* (ver Apartado 3.6). Se han usado dos configuraciones con 100 modelos, una de 10×10 (*Bagging* se itera 10 veces, el número de modelos de cada *Bagging* es 10), y otra de 5×20 (*Bagging* se itera 5 veces, el número de modelos de cada *Bagging* es 20).

Para todos estos algoritmos se ha configurados dos versiones, una usando como método base un *Random Oracle* con *árboles de regresión*, y otra usando los *árboles de regresión* directamente. Como se pueden usar diferentes tipos de oráculos, se decidió usar un *Linear Random Oracle*, este oráculo divide el espacio en dos usando un hiperplano. Para construir el oráculo, el hiperplano está formado por los puntos equidistantes a los dos ejemplos elegidos al azar. Para este trabajo se ha usado la distancia euclidiana, se han normalizado los atributos numéricos en la escala $[0, 1]$ y en los atributos nominales se ha considerado que la distancia era 1 si sus valores eran diferentes y era 0 cuando coincidían.

Además, como base del experimento se han considerado otros métodos.

- Dos versiones de *árboles de regresión* directamente, una con poda y otra sin poda.
- Dos versiones de *regresión lineal*. Usando todos los atributos o sólo una selección de los mismos descrita en [109].
- Tres versiones de *vecinos más cercanos*. En una de ellas el número de vecinos es uno. En las otras, el número de vecinos se selecciona usando validación cruzada «dejando uno fuera» en función de la media absoluta del error, o de la media cuadrática.

Se ha usado Weka [115] para los experimentos. El método base (*árboles de regresión*) ya lo tiene implementado, así como *Bagging* y *Random Subspaces*. El resto de los métodos se han implementado.

Las tablas 4.3 y 4.4 muestran las características de los 61 conjuntos de datos considerados en el estudio.

5.2.2. Resultados

Para poder comparar todas las configuraciones consideradas, se ha usado una media de clasificaciones [17]. Para cada conjunto de datos, los algoritmos se han ordenado en función de su error. Al mejor método se le ha asignado la posición 1, al segundo mejor la 2 y se ha seguido así. Si ha habido empates se ha asignado la posición media a todos los métodos. Para cada método se ha calculado la media de las posiciones en cada conjunto de datos.

La Tabla 5.3 muestra los métodos ordenados en función de su posición promedio. El prefijo «*O-*» marca los métodos que usan *Random Oracle* y su puesto está en una columna separada de los que no. Las 12 mejores posiciones son para métodos que usan *Random Oracle*.

Tabla 5.3: Clasificación de posiciones promedio.

Puesto promedio		Método	Benef.	V	E	D
13.76	-	*O-AdaBoost.R2-S*-Ex *(P)*	8.74	•51	0	10
15.02	-	*O-Bagging (U)*	9.01	•54	0	7
15.91	-	*O-Iterated Bagging* 5x20 *(P)*	5.66	•46	1	14
16.86	-	*O-AdaBoost.R2-S*-Li *(P)*	8.73	•49	1	11
18.30	-	*O-Iterated Bagging* 5x20 *(U)*	8.31	•52	1	8
18.53	-	*O-AdaBoost.R2-S*-Sq *(P)*	6.70	•47	2	12
19.35	-	*O-AdaBoost.R2-S*-Li *(U)*	10.25	•56	0	5
19.48	-	*O-AdaBoost.R2-S*-Ex *(U)*	11.06	•54	0	7
19.60	-	*O-AdaBoost.R2-W*-Sq *(U)*	10.21	•55	0	6
21.03	-	*O-AdaBoost.R2-W*-Ex *(U)*	14.98	•60	0	1
21.30	-	*O-Random Subspaces* 75 % *(P)*	13.70	•60	0	1
21.57	-	*O-AdaBoost.R2-S*-Sq *(U)*	6.70	•48	1	12
-	21.57	*Iterated Bagging* 5x20 *(P)*	-	-	-	-
21.95	-	*O-AdaBoost.R2-W*-Li *(U)*	13.07	•56	0	5
21.98	-	*O-Bagging (P)*	4.15	•43	1	17
-	22.50	*AdaBoost.R2-S*-Ex *(P)*	-	-	-	-
22.54	-	*O-Aleatorización (U)*	24.76	•61	0	0
-	24.03	*Bagging (U)*	-	-	-	-

Continua en la siguiente página ...

Tabla 5.3 (continuación): Clasificación de posiciones promedio.

Puesto promedio		Método	Benef.	V	E	D
24.07	-	*O-Iterated Bagging* 10x10 *(P)*	2.96	•38	2	21
24.32	-	*O-Aleatorización (P)*	6.57	•51	1	9
-	25.24	*AdaBoost.R2-S*-Sq *(P)*	-	-	-	-
25.43	-	*O-AdaBoost.R2-W*-Ex *(P)*	5.29	•44	0	17
-	25.59	*AdaBoost.R2-S*-Li *(P)*	-	-	-	-
-	26.12	*Bagging (P)*	-	-	-	-
-	26.61	*Iterated Bagging* 5x20 *(U)*	-	-	-	-
26.76	-	*O-Iterated Bagging* 10x10 *(U)*	5.79	•47	1	13
26.98	-	*O-Random Subspaces* 50 % *(U)*	4.56	•47	2	12
-	27.02	*Iterated Bagging* 10x10 *(P)*	-	-	-	-
-	28.26	*AdaBoost.R2-S*-Sq *(U)*	-	-	-	-
28.69	-	*O-AdaBoost.R2-W*-Li *(P)*	3.04	•39	1	21
28.89	-	*O-Random Subspaces* 75 % *(P)*	3.32	35	2	24
29.49	-	*O-AdaBoost.R2-W*-Sq *(P)*	3.18	38	0	23
-	29.61	*AdaBoost.R2-S*-Li *(U)*	-	-	-	-
-	29.81	*AdaBoost.R2-W*-Sq *(U)*	-	-	-	-
-	30.53	*AdaBoost.R2-S*-Ex *(U)*	-	-	-	-
-	30.72	*AdaBoost.R2-W*-Ex *(P)*	-	-	-	-
-	30.89	*Aleatorización (P)*	-	-	-	-

Continua en la siguiente página ...

Tabla 5.3 (continuación): Clasificación de posiciones promedio.

Puesto promedio		Método	Benef.	V	E	D
-	31.54	*Random Subspaces* 50 % *(U)*	-	-	-	-
-	31.73	*AdaBoost.R2-W*-Li *(P)*	-	-	-	-
-	31.79	*regresión lineal*(todos)	-	-	-	-
-	31.79	k-vecinos (cuadrática)	-	-	-	-
-	31.90	*regresión lineal*(selección)	-	-	-	-
-	32.21	*Random Subspaces* 75 % *(P)*	-	-	-	-
-	32.55	*Iterated Bagging* 10x10 *(U)*	-	-	-	-
-	32.67	*AdaBoost.R2-W*-Sq *(P)*	-	-	-	-
-	33.53	k-vecinos (absoluta)	-	-	-	-
-	34.99	*Random Subspaces* 75 % *(U)*	-	-	-	-
-	35.02	*AdaBoost.R2-W*-Li *(U)*	-	-	-	-
35.50	-	*O-Random Subspaces* 50 % *(P)*	0.80	30	0	31
-	36.02	*AdaBoost.R2-W*-Ex *(U)*	-	-	-	-
-	36.30	*Random Subspaces* 50 % *(P)*	-	-	-	-
-	43.43	Árbol *(P)*	-	-	-	-
-	46.39	1-vecino	-	-	-	-
-	47.30	Árbol *(U)*	-	-	-	-

Se define beneficio como la diferencia de posición promedio de un método que usa *Random Oracle* con su correspondiente método sin *Random Oracle*. La columna que muestra los beneficios en la Tabla 5.3 siempre es positiva.

Cuando se comparan dos métodos, se calcula el número de conjuntos de datos en los que obtiene un resultado mejor, igual o peor que el otro. Según [17], usando un test de signo, un método es significativamente mejor que otro, con un nivel de confianza del 0,05 si el número de victorias más la mide los empates es al menos $N/2 + 1{,}96\sqrt{N}/2$ (para $N = 61$ conjuntos de datos, este número es 39). La Tabla 5.3 muestra, en las columnas V/E/D, La comparación entre un método que usa *Random Oracle* con su correspondiente método sin *Random Oracle*. El símbolo • marca los 21 casos en que la diferencia es significativa.

El número de victorias, empates y derrotas y las posiciones promedio se han calculado usando comparación directa entre los resultados de los diferentes métodos. En ningún de estos casos se ha tenido en cuenta el tamaño de las diferencias. Para tener en cuenta el tamaño de las diferencias, se ha usado *quantitative scoring* [100, 118]. Dado el resultado de dos métodos i and j en un conjunto de datos, este resultado se define como:

$$S_{i,j} = \frac{RMSE_j - RMSE_i}{\text{máx}(RMSE_i, RMSE_j)}$$

donde $RMSE_i$ es la media cuadrática del error del método i. Salvo que ambos métodos tengan error 0, esta medida estará en el rango $[-1, 1]$, por lo que habitualmente se espresa en porcentaje. El signo indica el mejor método.

La Figura 5.5 muestra estos resultados comparando los métodos estudiados con y sin *Random Oracle*. El resultado se ha calculado para cada conjunto de datos y ordenado de menor a mayor. El número de valores positivos se corresponden con las victorias de la Tabla 5.3 y los valores negativos con las derrotas.

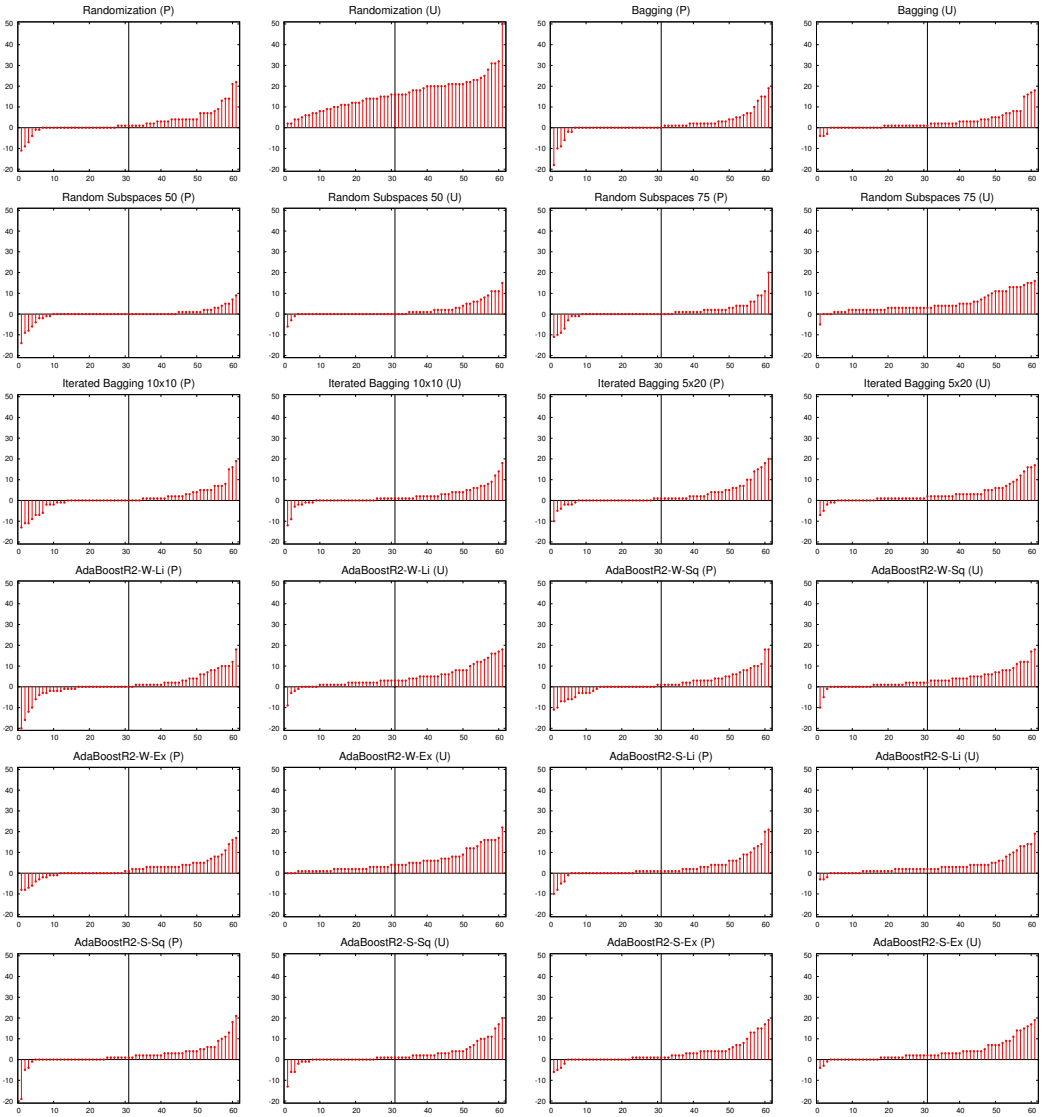

Figura 5.5: Comparación de la mejora de resultados de *Random Oracle*.

Cuando se compara métodos con estos gráficos, Se espera tener más valores positivos que negativos pero también se espera que los valores absolutos positivos sean mayores que los menores. Para este caso, los métodos con *Random Oracle* consiguen más valores positivos y sus mayores valores absolutos positivos.

El resultado de una combinación depende del número de modelos que se combinan. Las figuras 5.6 y 5.7 muestran el error en función del número de repeticiones (de 1 a 100) para cada conjunto de datos. El gráfico muestra el error de *AdaBoost.R2-S*-Ex *(P)* con y sin *Random Oracle*. Por lo general, los resultados con *Random Oracle* son mejores. Llama la atención que para algunos conjuntos de datos (p.ej. *echo-months*, *fruitfly*, *strike*, *veteran*), el error se incrementa con más iteraciones. Esto indica que *AdaBoost.R2* no es robusto con respecto al tamaño de la combinación. Sin embargo, el uso de *Random Oracle* mejora su robustez en todos los casos, y en algunos casos es capaz, incluso, de cambiar la tendencia del error, reduciéndose de forma considerable en los lugar de incrementarse con el incremento de repeticiones, como puede observarse en los conjuntos *pharynx* y *schlvote*.

La Figura 5.8 muestra una comparativa de la mejora entre la versión con *Random Oracle* y la versión sin él en función del tamaño de la combinación, para las configuraciones en las mejores posiciones: *AdaBoost.R2-S*-Ex *(P)* y *Bagging (U)*. Se muestra el porcentaje de conjuntos de datos en los que las versiones con *Random Oracle* son mejores que las que no los usan. El caso de empate se considera como media victoria. Salvo para combinaciones de tamaño muy pequeño –5 o menos–, en las que las versiones que no usan *Random Oracle* pueden ser mejores, en el resto las versiones con *Random Oracle* son mejores.

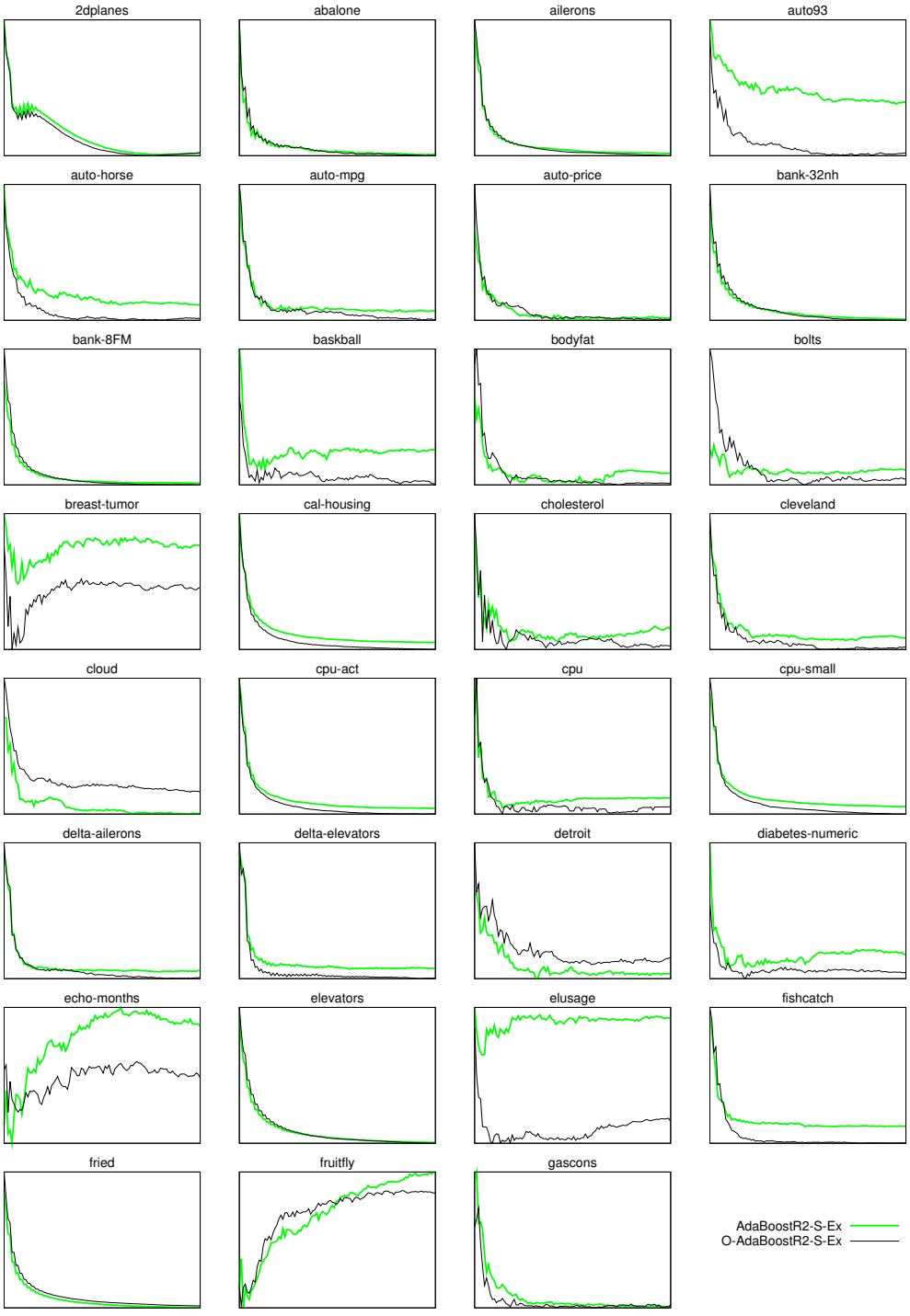

Figura 5.6: Error, en los diferentes conjuntos de datos, en función de tamaño de la combinación (1ª parte).
En verde el *AdaBoost.R2-S*-E, en negro la versión que usa *Random Oracle*.

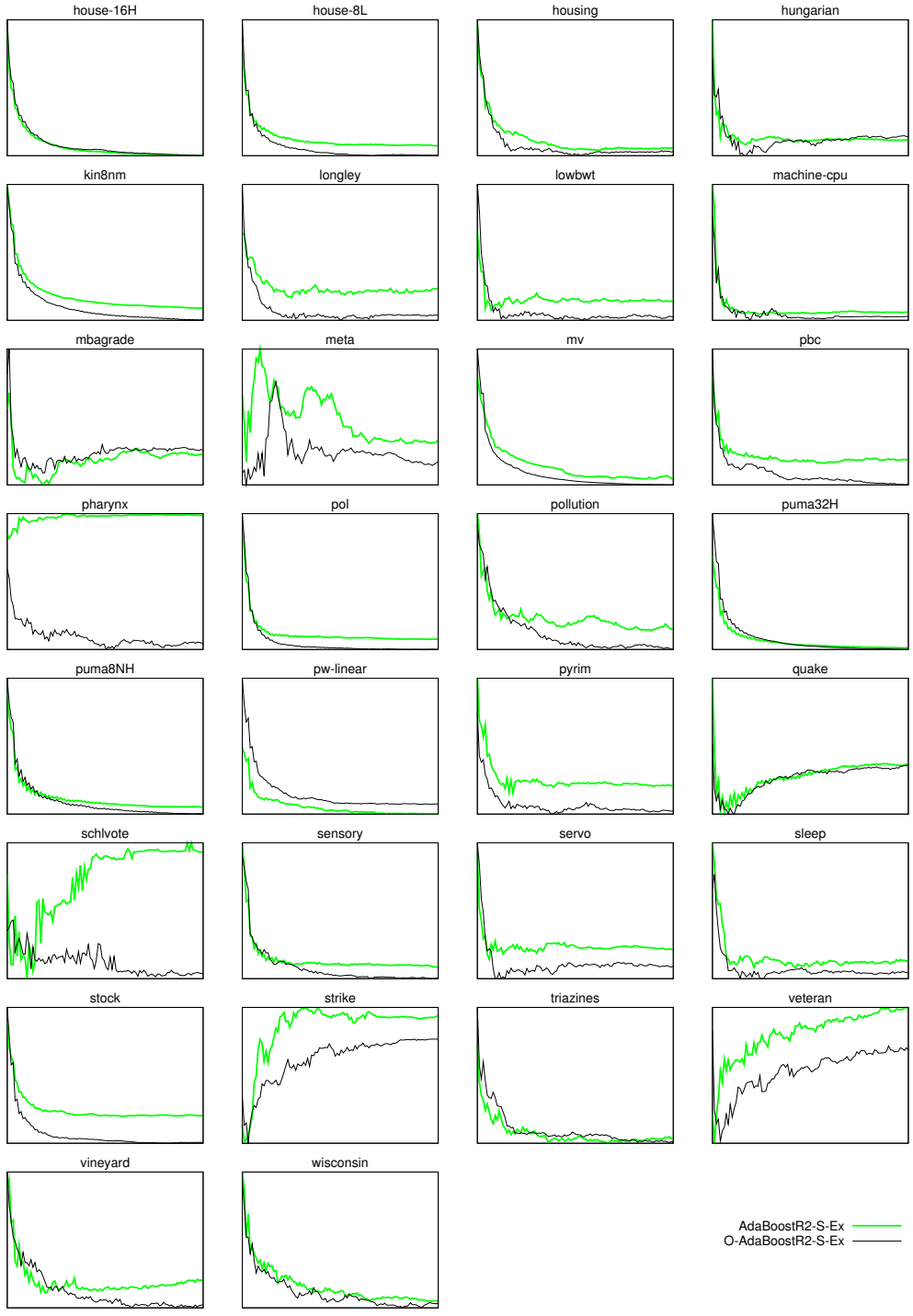

Figura 5.7: Error, en los diferentes conjuntos de datos, en función de tamaño de la combinación (2ª parte).
En verde el *AdaBoost.R2-S*-E, en negro la versión que usa *Random Oracle*.

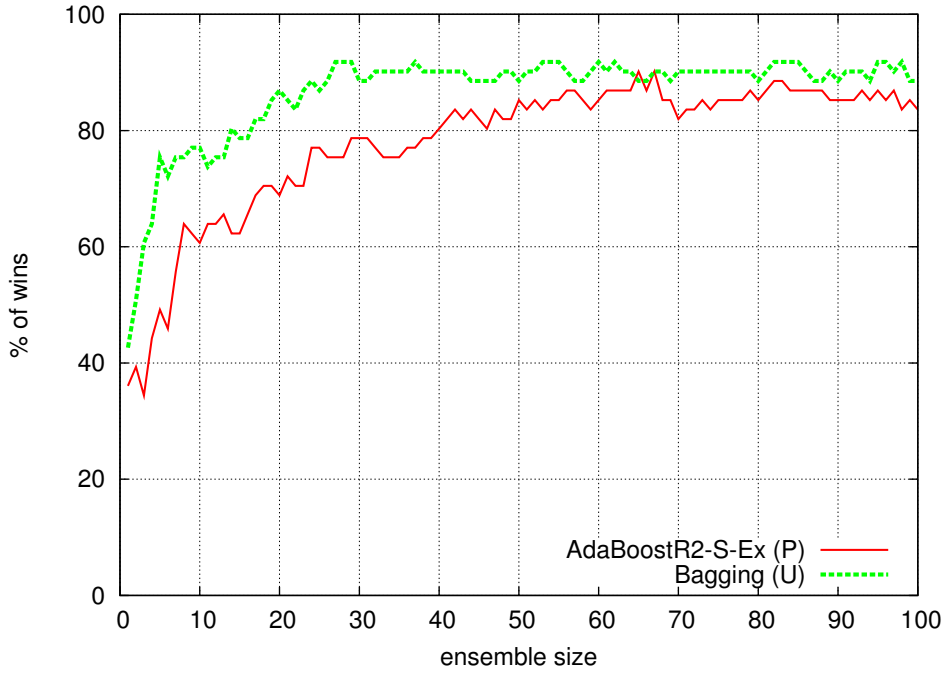

Figura 5.8: Evolución del porcentaje de victorias cuando se comparan versiones con y sin *Random Oracle*.

5.2.3. Diagramas de Diversidad-error

El éxito de las combinaciones están formadas por modelos con poco error, y que sean diversas. Estos dos objetivos son contradictorios, porque si el error de dos modelos es pequeño, estos no pueden diferir mucho. Se han propuesto varias medidas de la diversidad para analizar el comportamiento de las combinación de métodos [61].

Los *diagramas diversidad-error* [77] son una de las técnicas usadas. Son gráficos de dispersión, donde hay un punto para cada par de modelos. El eje horizontal representa la diversidad entre dos modelos, en clasificación se usa habitualmente κ (kappa). En este trabajo, para medir la diversidad se ha usado el opuesto a media cuadrática de la diferencia entre las predicciones de los modelos entre ellos.

$$RMSE \text{ opuesta} = -\sqrt{\sum_{i=1}^{n} \frac{(q_i - p_i)^2}{n}}$$

donde p_i y q_i son las predicciones de los dos modelos. Se hace notar que el cambio de signo es para tener una medida que al igual que el κ disminuya con la diversidad.

El eje vertical representa el error promedio de los dos modelos. Para regresión se pueden usar varias medidas del error, se ha usado la media cuadrática del error

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(a_i - p_i)^2}{n}}$$

donde a_i es el valor actual y p_i es el predicho.

La Figura 5.9 muestra estos diagramas, para 3 de los conjuntos de datos, de los las versiones de *AdaBoost.R2-S*-Ex *(P)* y *Bagging (U)* con *Random Oracle* y si ellos. Para resumir los diagramas de todos los conjuntos de datos, se ha usado el *diagrama de movimiento de la diversidad y el error* [93]. En este diagrama la relación entre la diversidad y el error de dos métodos, para el mismo conjunto de datos, se representa por una flecha, en el que los extremos de la flecha son los puntos del diagrama diversidad-error. La Figura 5.10 muestra este diagrama, para todos los conjuntos de datos del experimento. El origen de las flechas son las versiones de los métodos sin *Random Oracle*, mientras que las puntas representan las versiones de los métodos con *Random Oracle*. Casi todas las flechas apuntan a la izquierda, lo que indica que los *Random Oracle* aumentan la diversidad. Pero la mayoría de las flechas apuntan hacia arriba, lo que indica que los modelos base tienen más error. El coste de aumentar la diversidad es aumentar el error, pero no es cierto en este caso ya que el error de las combinaciones con *Random Oracle* era menor, en la mayor parte de los casos.

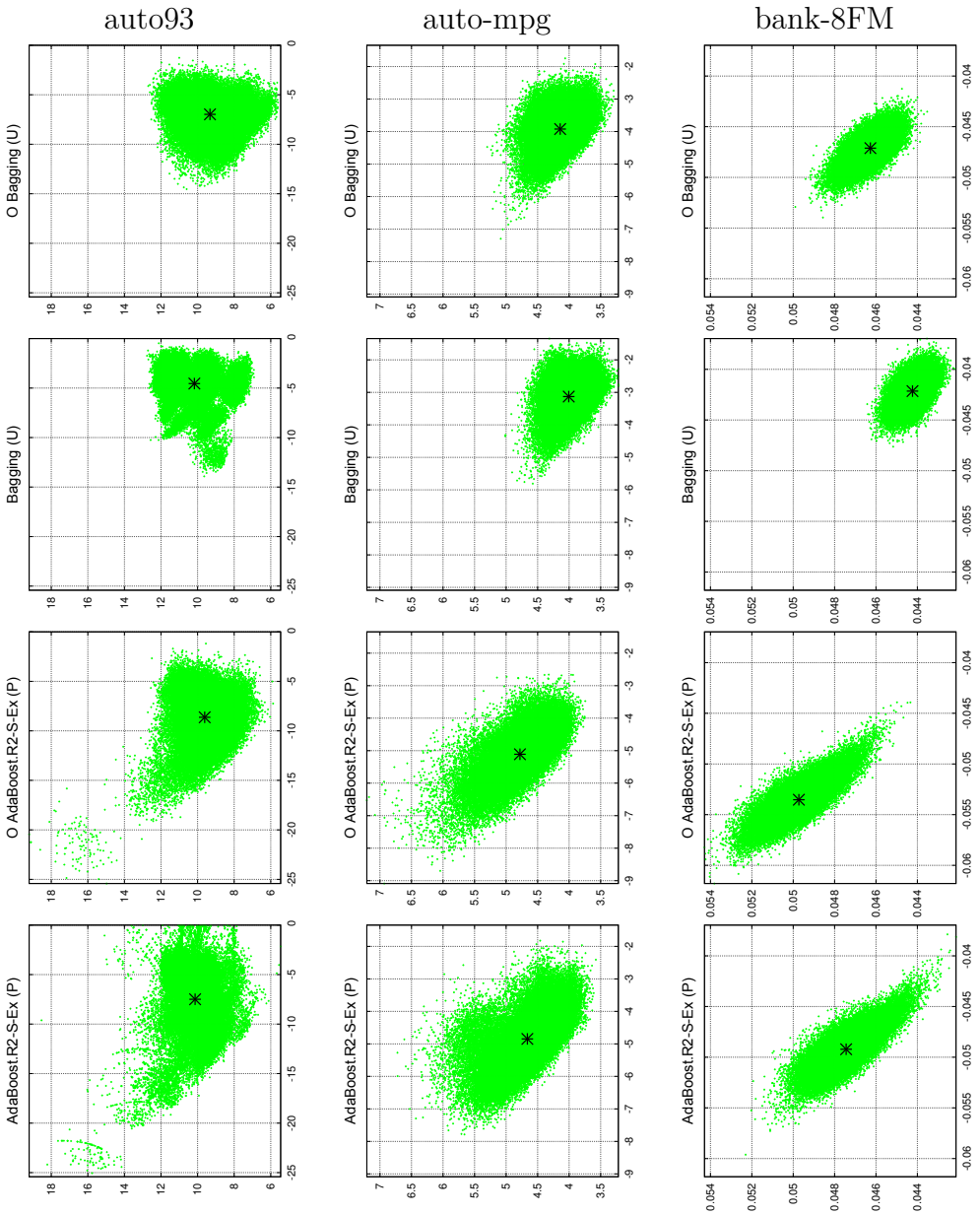

Figura 5.9: Diagramas diversidad-error.
El punto medio del cada diagrama está marcado con el símbolo '$\ast$'.

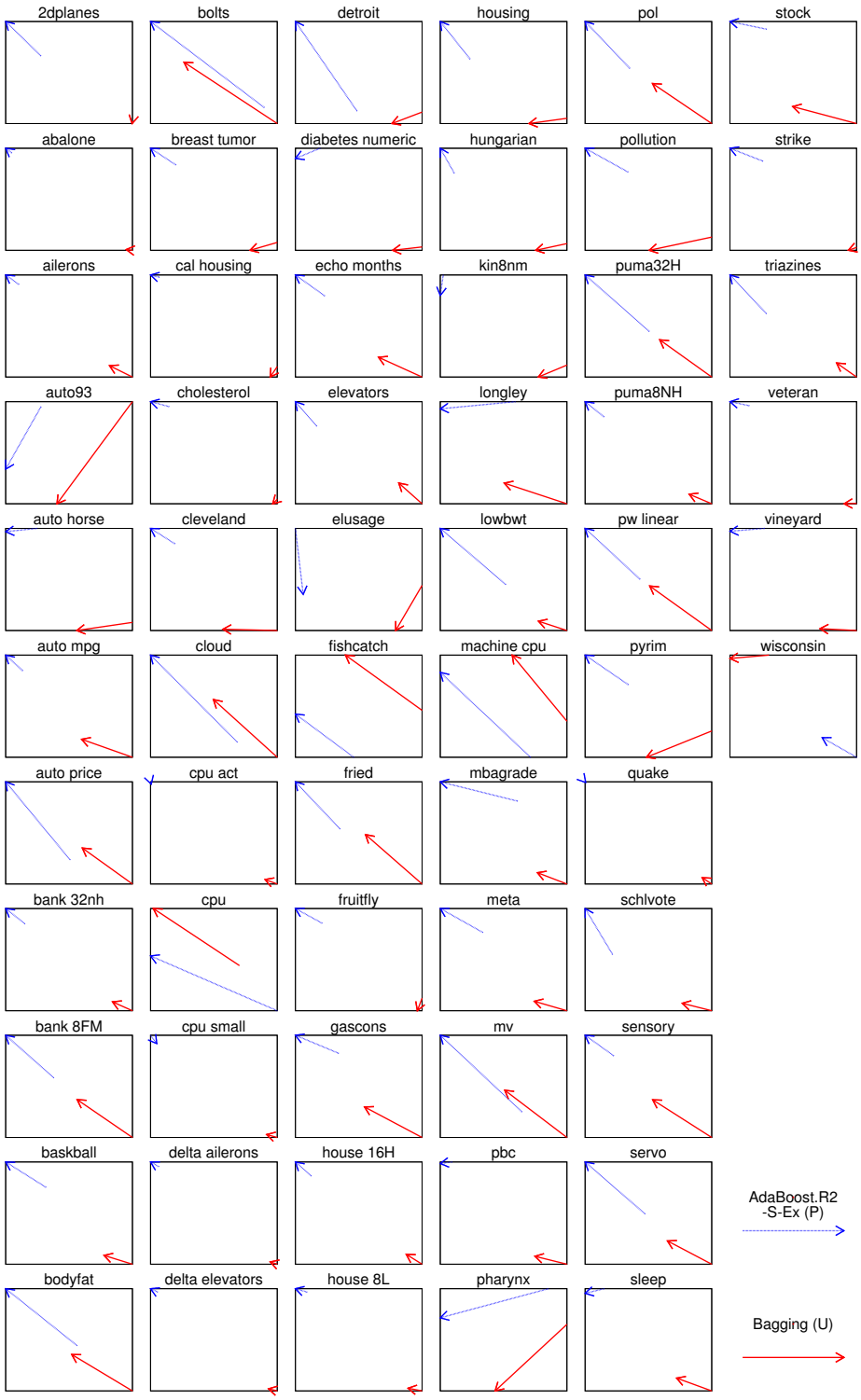

Figura 5.10: Diagrama de movimiento de la diversidad y el error.

5.2.4. Conclusiones

Se ha estudiado el funcionamiento de combinaciones de *Random Oracle* para regresión, usando *árboles de regresión* como modelos base y 61 conjuntos de datos. Se han combinado *Bagging*, *Iterated Bagging*, *Random Subspaces* y *AdaBoost.R2*. En todas las configuraciones estudiadas *Random Oracle* mejoraba los resultados. Como se ha mostrado para *AdaBoost.R2*, la robustez del método aumenta al usar *Random Oracle* como modelo base. Probablemente, esto sea debido a que incrementan la diversidad de los modelos, como se ha mostrado en los diagramas de divesidad-error.

5.3. Un estudio experimental de combinaciones usando redes de funciones de base radial para tareas de regresión

Los resultados del estudio de este apartado se presentan en la CAEPIA'2015 [88].

En este apartado se compara el funcionamiento, para regresión, de algoritmos de combinaciones de métodos, usando redes de funciones de base radial (RBFs) como método base y 61 conjuntos de datos. Se han comparado *Bagging* [8], *Iterated Bagging* [10], *Random Subspaces* [53], *AdaBoost* [33, 23] y *Rotation Forest* [94] usando tres configuraciones distintas de las funciones de base radial.

Las redes de funciones de base radial se comentaron en el Apartado 2.4.

El resto de este apartado se describirá el diseño del experimento realizado, para pasar a mostrar y comentar los resultados, y acabará con un apartado de conclusiones.

5.3.1. Diseño experimental

Para que el resultado del estudio sea suficientemente significativo se ha utilizado una amplia selección de conjuntos de datos, en concreto, se han usado los 61 conjuntos que se muestran en las tablas 4.4 y 4.3.

5.3.1.1. Algoritmos a comparar

Se han comparado diferentes algoritmos de combinación de métodos y se ha decidido usar redes de funciones de base radial (RBF) como método base. De entre los regresores que usan redes de funciones de base radial se ha elegido el algoritmo RBFNetwork que viene implementado en la distribución de Weka.

Se han utilizado tres configuraciones de este método base cambiando distinto número de funciones de base radial:

- Con un número fijo de funciones de base radial, en concreto con dos (es la opción por defecto en Weka).
- Con tantas funciones de base radial como la raíz cuadrada de la mitad del número de ejemplos de entrenamiento, número que se ha elegido apoyándonos en la aproximación recomendada por Kanti Mardia [112] para determinar el número de cluster de un conjunto de datos.
- Con tantas funciones de base radial como la raíz cuadrada del número de datos de entrenamiento, para como se comporta al aumentar el número.

Y para cada uno de estos tres métodos base se han probado configuraciones, todas ellas con 50 modelos, de las siguientes combinaciones:

- *Bagging* (ver Apartado 3.1). Se han utilizado dos tamaños de conjuntos para entrenar cada modelo, el 100 % y el 50 % del tamaño del conjunto de datos de entrenamiento.

- *Iterated Bagging* (ver Apartado 3.6). Se ha utilizado una configuración de 10 iteraciones, cada una con 5 modelos miembro para tener el todal de 50 modelos.

- *Random Subspaces* (ver Apartado 3.3). Se han utilizado subespacios con el 50 % y el 75 % del tamaño del espacio original.

- *AdaBoost.R2* (ver Apartado 3.2). Se han utilizado 6 configuraciones, por un lado se ha utilizado la estrategia de cambiar los pesos (*reweighting*) de cada ejemplo, y por otro lado se ha utilizado un sistema de selección de los ejemplos (*resampling*), estas dos estrategias se denotan con los sufijos «-W» y «-S» y se han combinado con tres funciones de perdida: lineal, cuadrática y exponencial, que se denotan con los sufijos «-L», «-Q» «-E».

- *Rotation Forest* (ver Apartado 3.7). Se han utilizado las opciones por defecto.

En el experimento se han añadido además las tres configuraciones del método base directamente, para comparar la mejora del uso de combinaciones para cada uno de ellas.

El resultado del experimento se obtuvo usando *validación cruzada* 5×2 particiones [19]. Los datos se dividen en dos mitades, una para comprobar la predicción obtenida por la combinación entrenada con la otra mitad. Y luego se cambian los papeles de esas mitades. Este proceso se repite cinco veces. Por lo que el resultado del informe representa la media de esas diez ejecuciones.

5.3.2. Resultados

Se analizan los resultados de los experimentos dos veces. Primero se comparan las combinaciones que usan el mismo tipo de

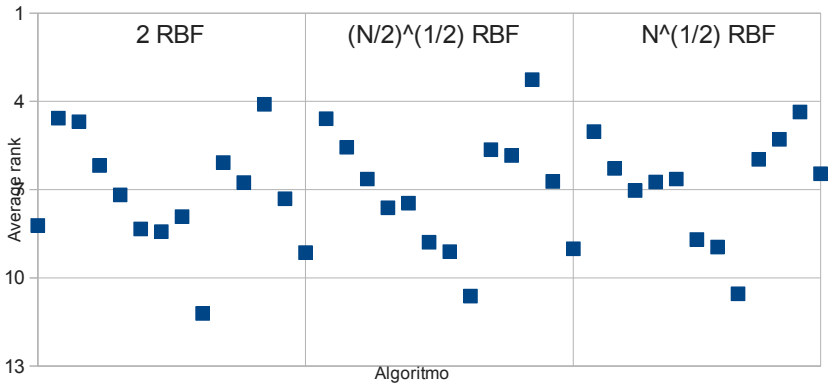

Figura 5.11: Gráfico de las posiciones promedio de la Tabla 5.4.

método base entre ellas, y luego se analizan todas las combinaciones en su conjunto.

5.3.2.1. Resultados con el mismo método base

En la Tabla 5.4 y en el gráfico de la Figura 5.11 se puede comparar el promedio de las posiciones en que queda clasificado cada algoritmo para cada conjunto de datos, tal como propone Demšar [17]. La tabla muestra de forma independiente las combinaciones que usan el mismo método base, ordenadas por posición, mientras que el gráfico de la figura muestra la posición promedio de los algoritmos en el orden que marca el número entre paréntesis en la tabla, los métodos base por si sólos están representado sobre las líneas de la cuadrícula.

Se puede observar que, en los tres casos, *Iterated Bagging* queda el mejor cuando se le compara con el resto de los algoritmos. La versión de *Bagging* que utiliza conjuntos del mismo tamaño que el conjunto de datos de entrenamiento es siempre la segunda mejor configuración.

También se puede observar que, al aumentar el número de funciones de base radial del método base, los *Random Subspaces*, sobre todo su versión con tamaño del subespacios del 75 %, me-

Tabla 5.4: *Ranking* promedio de los algoritmos utilizando el mismo tipo de modelo base de RBF.

2 funciones de base radial

#posic.	algoritmo
1	4,10 (12) *Iterated Bagging*
2	4,57 (2) *Bagging* 100 %
3	4,69 (3) *Bagging* 50 %
4	6,08 (10) *Random Subspaces* 50 %
5	6,18 (4) *AdaBoost.R2-S*-L
6	6,76 (11) *Random Subspaces* 75 %
7	7,18 (5) *AdaBoost.R2-S*-Q
8	7,31 (13) *Rotation Forest*
9	8,22 (1) *Red RBF*
10	8,34 (6) *AdaBoost.R2-S*-E
11	7,92 (8) *AdaBoost.R2-W*-Q
12	8,43 (7) *AdaBoost.R2-W*-L
13	11,21 (9) *AdaBoost.R2-W*-E

$\sqrt{N_T/2}$ funciones de base radial

#posic.	algoritmo
1	3,26 (25) *Iterated Bagging*
2	4,59 (15) *Bagging* 100 %
3	5,56 (16) *Bagging* 50 %
4	5,64 (23) *Random Subspaces* 50 %
5	5,84 (24) *Random Subspaces* 75 %
6	6,64 (17) *AdaBoost.R2-S*-L
7	6,72 (26) *Rotation Forest*
8	7,46 (19) *AdaBoost.R2-S*-E
9	7,62 (18) *AdaBoost.R2-S*-Q
10	8,79 (20) *AdaBoost.R2-W*-L
11	9,11 (21) *AdaBoost.R2-W*-Q
12	9,14 (14) *Red RBF*
13	10,62 (22) *AdaBoost.R2-W*-E

$\sqrt{N_T}$ funciones de base radial

#posic.	algoritmo
1	4,36 (38) *Iterated Bagging*
2	5,03 (28) *Bagging* 100 %
3	5,29 (37) *Random Subspaces* 75 %
4	5,97 (36) *Random Subspaces* 50 %
5	6,28 (29) *Bagging* 50 %
6	6,46 (39) *Rotation Forest*
7	6,64 (32) *AdaBoost.R2-S*-E
8	6,74 (31) *AdaBoost.R2-S*-Q
9	7,03 (30) *AdaBoost.R2-S*-L
10	8,70 (33) *AdaBoost.R2-W*-L
11	8,95 (34) *AdaBoost.R2-W*-Q
12	9,01 (27) *Red RBF*
13	10,54 (35) *AdaBoost.R2-W*-E

joran sus posiciones con respecto a los demás algoritmos, mientras que la versión de *Bagging* con tamaño 50 % del tamaño del conjunto de datos de entrenamiento sigue la tendencia opuesta.

Llama la atención que la configuración de *AdaBoost.R2* con repesado y función de perdida exponencial quede siempre como la peor configuración, incluso peor que el método base que utiliza.

5.3.2.2. Resultados globales

La tabla de clasificación con los resultados globales de todas las configuraciones, que se pueden consultar en la Tabla 5.5, las configuraciones están ordenadas por la posición promedio. Se han separado las posiciones en tres columnas en función del método base.

Los resultados sitúan como mejor algoritmo a la configuración de *Iterated Bagging* que usa $\sqrt{N_T}$ funciones de base radial, donde N_T en el número de datos del conjunto de datos de entrenamiento.

También se puede observar que, como norma general, aumentar el número de funciones de base radial es más importante en la reducción del error que el algoritmo de combinación elegido, dado que doce de las trece configuraciones de los algoritmos comparados que sólo usan dos funciones de base radial han quedado en las posiciones del último tercio de la tabla y ninguna en el primer tercio; mientras que de las configuraciones que usan $\sqrt{N_T/2}$ funciones de base radial cinco quedan en el primer tercio de la tabla de posiciones, siete en el tercio central y solo una en el último tercio; y de las trece configuraciones que usan más funciones de base radial, ocho quedan en el primer tercio de la tabla y cinco en el tercio central, no quedando ninguna en el tercio final.

Tabla 5.5: *Ranking* promedio de todas las configuraciones usando RBF.

#	posición con RBFs 2	$\sqrt{N_T/2}$	$\sqrt{N_T}$	algoritmo	vs 2 RBF P / E / G	victorias en conjunto de datos
1	-	-	8,16	*Iterated Bagging*	0 / 35 / 26	6
2	-	8,20	-	*Iterated Bagging*	0 / 34 / 27	5
3	-	-	9,85	*Bagging* 100 %	0 / 31 / 30	3
4	-	-	10,40	*Random Subspaces* 75 %	0 / 35 / 26	3
5	-	10,95	-	*Bagging* 100 %	0 / 31 / 30	-
6	-	-	11,77	*Random Subspaces* 50 %	0 / 34 / 27	5
7	-	-	12,66	*Bagging* 50 %	0 / 33 / 28	1
8	-	13,55	-	*Random Subspaces* 75 %	0 / 36 / 25	-
9	-	13,59	-	*Bagging* 50 %	0 / 35 / 26	1
10	-	-	13,74	*Rotation Forest*	0 / 34 / 27	10
11	-	13,85	-	*Random Subspaces* 50 %	0 / 35 / 26	1
12	-	-	15,43	*AdaBoost.R2-S*-E	7 / 34 / 20	4
13	-	-	15,77	*AdaBoost.R2-S*-Q	6 / 36 / 19	13
14	-	-	16,02	*AdaBoost.R2-S*-L	5 / 39 / 17	-
15	-	16,21	-	*Rotation Forest*	0 / 37 / 24	2

Nº Derrotas (P), empates (E) y victorias (G) significativas al método base RBF (por defecto), y Nº victorias a todos.

Continua en la siguiente página ...

Tabla 5.5 (continuación): *Ranking* promedio de todas las configuraciones usando

#	posición con RBFs 2	$\sqrt{N_T/2}$	$\sqrt{N_T}$	algoritmo	vs 2 RBF P / E / G	victorias en conjunto de datos
16	-	17,25	-	*AdaBoost.R2-S*-L	7 / 33 / 21	1
17	-	-	18,66	*AdaBoost.R2-W*-L	2 / 39 / 20	-
18	-	-	18,97	*AdaBoost.R2-W*-Q	1 / 45 / 15	-
19	-	-	19,11	Red RBF	0 / 36 / 25	-
20	-	19,28	-	*AdaBoost.R2-S*-E	7 / 33 / 21	2
21	-	20,23	-	*AdaBoost.R2-S*-Q	9 / 31 / 21	-
22	-	20,86	-	Red RBF	0 / 42 / 19	1
23	-	20,89	-	*AdaBoost.R2-W*-L	1 / 44 / 16	-
24	-	22,08	-	*AdaBoost.R2-W*-Q	3 / 44 / 14	-
25	22,64	-	-	*Iterated Bagging*	0 / 52 / 9	-
26	-	-	24,08	*AdaBoost.R2-W*-E	4 / 42 / 15	-
27	24,28	-	-	*Bagging* 100 %	0 / 54 / 7	-
28	24,54	-	-	*Bagging* 50 %	0 / 52 / 9	1
29	26,54	-	-	*AdaBoost.R2-S*-L	5 / 50 / 6	-
30	26,59	-	-	*Random Subspaces* 50 %	2 / 50 / 9	-
31	-	26,75	-	*AdaBoost.R2-W*-E	5 / 38 / 18	-

Nº Derrotas (P), empates (E) y victorias (G) significativas al método base RBF (por defecto), y Nº victorias a todos.

Continua en la siguiente página ...

Tabla 5.5 (continuación): *Ranking* promedio de todas las configuraciones usando

#	posición con RBFs 2	$\sqrt{N_T/2}$	$\sqrt{N_T}$	algoritmo	vs 2 RBF P / E / G	victorias en conjunto de datos
32	27,42	-	-	*Random Subspaces* 75 %	5 / 46 / 10	-
33	27,98	-	-	*AdaBoost.R2-S*-Q	1 / 56 / 4	-
34	28,23	-	-	*Rotation Forest*	2 / 52 / 7	-
35	29,52	-	-	*AdaBoost.R2-W*-Q	2 / 51 / 8	1
36	29,51	-	-	*AdaBoost.R2-S*-E	10 / 46 / 5	1
37	29,55	-	-	Red RBF	- / 61 / -	-
38	29,92	-	-	*AdaBoost.R2-W*-L	7 / 48 / 6	-
39	34,98	-	-	*AdaBoost.R2-W*-E	12 / 47 / 2	-
	27,82	17,21	14,97	promedio de la columna		

También se puede defender el aumento del número de funciones de base radial mirando el promedio de posiciones de todos las combinaciones que usan la misma configuración de método base (promedio de la columna en la Tabla 5.5). Los algoritmos con más funciones de base radial, tiene mejor promedio que el resto.

También se observa que algunos algoritmos han cambiado ligeramente su posición relativa, con respecto al resto de los que utilizan su mismo método base, si lo comparamos con la Tabla 5.4, como, por ejemplo, la versión con 2 funciones de base radial del método base, que estaba el noveno en la tabla anterior, ha sido superado por dos de las configuraciones de *AdaBoost.R2*.

Se ha comparado el número de veces que los métodos ganaban para cada conjunto de datos al método base por defecto, usando *t-test estadístico corregido remuestreado* [81] (con nivel de significación: 0,05).

En lo que se refiere a victorias significativas, *Bagging* 100 % es el que gana más veces al método base por defecto, un total de 30. Pero llama la atención que este método base, pese a quedar en la posición 37, la antepenúltima de la tabla, es capaz de ganar significativamente hasta en 103 ocasiones a 21 configuraciones de diferentes combinaciones de métodos.

Las configuraciones del algoritmo *AdaBoost.R2* tiene un comportamiento muy variable, el caso más extremo es su configuración *AdaBoost.R2-S*-Q con $\sqrt{N_T}$ funciones de base radial que pese a ser capaz de ganar en 13 de los 61 conjunto de datos a todos los demás algoritmos, ganando de forma significativa 19 veces al algoritmo base por defecto, en la tabla de clasificación cae al puesto 13 y pierde de forma significativa 6 veces con ese método base.

La configuración del algoritmo *Rotation Forest* con $\sqrt{N_T}$ funciones de base radial es la segunda que más veces es capaz de ganar a todos los demás algoritmos en un mismo conjunto de

datos, en 10, pese a ello queda en el décimo puesto de la tabla, aunque en este caso no pierde significativamente con el algoritmo base en ningún conjunto de datos y lo bate en 27 de ellos.

5.3.3. Conclusiones

Se ha analizado el comportamiento de diferentes combinaciones de métodos usando redes de funciones de base radial como método base. Se ha visto que la combinaciones que mejores resultados obtiene es *Iterated Bagging*, confirmando los resultados de [104] para otros métodos base.

Se han analizado los comportamientos de las redes con un número fijo de funciones de base radial (2) y con un número variable en función del tamaño del conjunto de datos de entrenamiento (N_T). El tener redes con más funciones de base radial mejora, en el promedio de los 61 conjunto de datos del experimento, los resultados para todas las combinaciones de métodos.

5.4. Un estudio experimental de combinaciones de máquinas de soporte vectorial para tareas de regresión

Los resultados del estudio de este apartado están pendientes de publicar.

Este apartado es un trabajo con un estudio experimental del funcionamiento de las combinaciones de métodos para regresión usando máquinas de soporte vectorial (SVMs) como método base, para lo que se compararán los resultados en 61 conjuntos de datos.

Se han comparado *Bagging* [8], *Iterated Bagging* [10], *Random Subspaces* [53], *AdaBoost* [33, 23] y *Rotation Forest* [94] usando cuatro configuraciones distintas de las máquinas de soporte vectorial.

Las máquinas de soporte vectorial se comentaron en el Apartado 2.5.

El resto este apartado describirá el diseño del experimento realizado, para pasar a mostrar y comentar los resultados, y acabará con un apartado de conclusiones

5.4.1. Diseño experimental

Con el objetivo de tener un estudio lo suficientemente significativo, se han utilizado los 61 conjuntos de datos que se muestran en las tablas 4.4 y 4.3.

5.4.1.1. Algoritmos a comparar

El cálculo de máquina de soporte vectorial es un método iterativo y el tiempo de proceso depende varios parámetros que hacen converger al método. En Weka existe una implementación de LIBSVM que tiene para elegir entre 5 algoritmos, tres son para problemas de clasificación y dos para problemas regresión, se ha decidido utilizar el último de las algoritmos (nu-SVR). Como el núcleo puede usar, entre otras, funciones lineales o funciones de base radial; se pueden configurar el coste y la tolerancia, además se puede fijar el parámetro γ (gamma)[5] de las funciones de base radial para todos los conjunto de datos o se puede dejar por defecto (será función del número de los atributos de cada conjunto de datos). También se puede optar por normalizar los datos o no.

Se ha decidido comparar el funcionamiento de las diferentes combinaciones con 4 configuraciones distintas de máquina de soporte vectorial. Se van a usar dos configuraciones que normalicen los conjunto de datos, y otras dos que no, dos que usen núcleos con función lineal y otras dos que usen núcleos gausianos, en el primero de estos el γ tendrá valor por defecto y en el otro caso se

[5] $\gamma = 1/r^2$ en la fórmula de $h(x)$ vista en el Apartado 2.4

establecerá un valor fijo. Se usará siempre una tolerancia 10^{-3}, y en un caso se reducirá el coste por defecto y en otros dos se aumentará.

Con estas decisiones, la primera de las configuraciones es la que viene por defecto para LIBSVM.nu-SVR, con núcleo función de base radial, γ por defecto, coste 1, tolerancia 10^{-3} y sin normalizar. La segunda de las configuraciones será también sin normalizar pero con núcleo función lineal, coste 0,1, tolerancia 10^{-3}. La tercera de las configuraciones será también con núcleo función lineal, coste 100, tolerancia 10^{-3} y normalizando. La cuarta de las configuraciones también será normalizando, pero con núcleo función de base radial, y fijando el coste 100, el gamma 0,1 y la tolerancia en 10^{-3}.

Se han probado configuraciones de las siguientes combinaciones, todas ellas con 50 modelos, para cada uno de estos cuatro modelos base:

- *Bagging* (ver Apartado 3.1). Se han utilizado dos tamaños de conjuntos para entrenar cada modelo, el 100 % y el 75 % del tamaño del conjunto de datos de entrenamiento.

- *Iterated Bagging* (ver Apartado 3.6). Se ha utilizado una configuración de 10 iteraciones, cada una con 5 modelos miembro, para tener el total de 50 modelos.

- *Random Subspaces* (ver Apartado 3.3). Se han utilizado subespacios con el 50 % y el 75 % del tamaño del espacio original.

- *AdaBoost.R2* (ver Apartado 3.2). Se han utilizado 3 configuraciones, se han configurado tres funciones de perdida posibles: lineal, cuadrática y exponencial, que se denotan con los sufijos «-L», «-Q» «-E». Se ha intentado doblar en número de configuraciones al utilizar dos estrategias para crear los conjunto de datos de entrenamiento, por un lado

cambiar los pesos (*reweighting*) de cada ejemplo, y por otro lado se ha utilizado un sistema de selección de los ejemplos (*resampling*), pero solo se se ha obtenido los datos de la segunda estrategia denotados con «-S», por qué al analizar los datos se vio que los resultados eran siempre iguales ya la implementación de la máquina de soporte vectorial sólo aceptaba esa estrategia, independientemente de la configuración.

- *Rotation Forest* (ver Apartado 3.7). Se han utilizado las opciones por defecto.

En el experimento se han añadido además las cuatro configuraciones del método base directamente, para comparar si había mejoras al usar las combinaciones.

El resultado del experimento se obtuvo usando *validación cruzada* 5×2 particiones [19]. Los datos se dividen en dos mitades, una para comprobar la predicción obtenida por la combinación entrenada con la otra mitad. Y luego se cambian los papeles de esas mitades. Este proceso se repite cinco veces. Por lo que al final se obtienen los resultados de diez ejecuciones y el informe representa la media Por lo que el resultado del de esas diez ejecuciones.

5.4.2. Resultados

Se analizan los resultados de los experimentos dos veces. Primero se comparan las combinaciones que usan el mismo tipo de método base entre ellas, y luego se analizan todas las combinaciones en su conjunto.

5.4.2.1. Resultados con el mismo método base

En la Tabla 5.6 y en el gráfico de la Figura 5.12 podemos comparar el promedio de las posiciones en que queda clasificado

Tabla 5.6: *Ranking* promedio de los algoritmos utilizando el mismo tipo de modelo base de SVM.

SVM por defecto

#posic.	algoritmo
1	2,37 $_{(2)}$ *Rotation Forest*
2	4,40 $_{(6)}$ *Random Subspaces* 50 %
3	4,64 $_{(5)}$ *Iterated Bagging*
4	4,72 $_{(7)}$ *Random Subspaces* 75 %
5	4,74 $_{(4)}$ *Bagging* 100 %
6	5,31 $_{(1)}$ SVM 0
7	5,59 $_{(3)}$ *Bagging* 75 %
8	7,14 $_{(11)}$ *AdaBoost.R2-S*-L
10	7,86 $_{(12)}$ *AdaBoost.R2-S*-Q
12	8,23 $_{(13)}$ *AdaBoost.R2-S*-E

SVM lineal, C:0.1

#posic.	algoritmo
1	3,74 $_{(17)}$ *Bagging* 100 %
2	4,13 $_{(16)}$ *Bagging* 75 %
3	4,15 $_{(15)}$ *Rotation Forest*
4	4,23 $_{(18)}$ *Iterated Bagging*
5	4,68 $_{(20)}$ *Random Subspaces* 75 %
6	4,76 $_{(14)}$ SVM 1
7	6,57 $_{(19)}$ *Random Subspaces* 50 %
8	6,82 $_{(24)}$ *AdaBoost.R2-S*-L
9	7,87 $_{(26)}$ *AdaBoost.R2-S*-E
10	8,05 $_{(25)}$ *AdaBoost.R2-S*-Q

SVM RBF, γ:0.1, C:100, Normaliza

#posic.	algoritmo
1	3,43 $_{(44)}$ *Iterated Bagging*
2	3,44 $_{(43)}$ *Bagging* 100 %
3	4,70 $_{(42)}$ *Bagging* 75 %
4	5,01 $_{(40)}$ SVM 3
6	5,03 $_{(41)}$ *Rotation Forest*
5	5,70 $_{(46)}$ *Random Subspaces* 75 %
7	5,90 $_{(50)}$ *AdaBoost.R2-S*-L
8	6,84 $_{(45)}$ *Random Subspaces* 50 %
9	7,44 $_{(52)}$ *AdaBoost.R2-S*-E
10	7,49 $_{(51)}$ *AdaBoost.R2-S*-Q

SVM lineal, C:100, Normaliza

#posic.	algoritmo
1	3,30 $_{(30)}$ *Bagging* 100 %
2	3,80 $_{(29)}$ *Bagging* 75 %
3	4,07 $_{(31)}$ *Iterated Bagging*
4	4,54 $_{(27)}$ SVM 2
5	4,88 $_{(33)}$ *Random Subspaces* 75 %
6	5,27 $_{(28)}$ *Rotation Forest*
7	6,26 $_{(32)}$ *Random Subspaces* 50 %
8	6,79 $_{(37)}$ *AdaBoost.R2-S*-L
9	7,82 $_{(39)}$ *AdaBoost.R2-S*-E
10	8,28 $_{(38)}$ *AdaBoost.R2-S*-Q

cada algoritmo para cada conjunto de datos, tal como propone Demšar [17]. La tabla muestra de forma independiente las combinaciones que usan el mismo métodos base, ordenadas por la posición promedio, mientras que el gráfico de la figura muestra la posición promedio de los algoritmos en el orden que marca el número entre paréntesis en la tabla, los métodos base por si sólos están representado sobre las líneas de la cuadrícula.

Lo primero que podemos observar son las mejores combinaciones. La variante *Bagging* 100 % es la mejor en los dos casos en

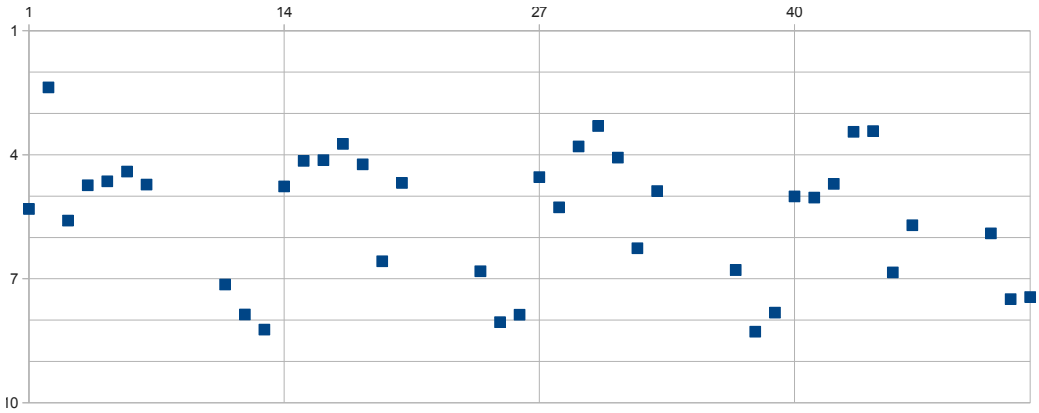

Figura 5.12: Gráfico de las posiciones promedio de la Tabla 5.6.

los que la función de la máquina de soporte vectorial es lineal. Mientras que cuando la máquina de soporte vectorial usa una función de base radial, el mejor algoritmo es *Rotation Forest* para la configuración por defecto e *Iterated Bagging* en la otra, la que fija el coste a 100, el γ a 0,1 y normaliza los datos.

Si analizamos por algoritmos, las variantes *AdaBoost.R2-W* como ya se han comentado, se han eliminado de la tabla al analizar que las opción de repesado quedaban siembre empatada con *AdaBoost.R2-S*, al hacer siempre remuestreo. Las variantes *AdaBoost.R2-S* nunca mejoran a la máquina de soporte vectorial por sí sola, ocupando los tres últimos puestos en tres de las clasificaciones y los dos últimos en la cuarta.

La variante *Random Subspaces* 50 % sólo gana a la máquina de soporte vectorial cuando se usan las opciones por defecto de ésta, en dos casos queda como séptima variante y en un caso queda entre las variantes de *AdaBoost.R2*. Mientras que la variante *Random Subspaces* 75 % y el *Rotation Forest* no mejoran a la máquina de soporte vectorial cuando ésta normaliza los conjunto de datos, pero sí cuando no lo hace; Además, la ventaja, en la tabla de clasificación, de la configuración del *Rotation Forest* con la máquina de soporte vectorial por defecto es notable.

Tabla 5.7: *Ranking* promedio global de los algoritmos con SVM.

#	posición				algoritmo
	SVM0	SVM1	SVM2	SVM3	
1	-	-	-	10,03	(44) *Iterated Bagging*
2	-	-	-	11,02	(43) *Bagging* 100 %
3	-	-	-	12,39	(42) *Bagging* 75 %
4	-	-	12,45	-	(30) *Bagging* 100 %
5	-	-	13,10	-	(29) *Bagging* 75 %
6	-	-	-	13,39	(41) *Rotation Forest*
7	-	-	13,74	-	(31) *Iterated Bagging*
8	-	-	-	14,02	(40) SVM 3 (RBF γ0.1 c100 Normal.)
9	-	-	14,20	-	(33) *Random Subspaces* 75 %
10	-	-	14,34	-	(27) SVM 2 (lineal c100 Normaliza)
11	-	-	-	14,41	(46) *Random Subspaces* 75 %
12	-	-	-	16,39	(50) *AdaBoost.R2-S*-L
13	-	-	17,11	-	(32) *Random Subspaces* 50 %
14	-	17,78	-	-	(17) *Bagging* 100 %
15	-	-	-	17,89	(45) *Random Subspaces* 50 %
16	-	18,22	-	-	(16) *Bagging* 75 %
17	-	18,43	-	-	(15) *Rotation Forest*

Continua en la siguiente página ...

Tabla 5.7 (continuación): *Ranking* promedio global de los algoritmos con SVM.

#	posición				algoritmo
	SVM0	SVM1	SVM2	SVM3	
18	-	-	18,55	-	(28) *Rotation Forest*
19	-	18,59	-	-	(18) *Iterated Bagging*
20	-	19,53	-	-	(14) SVM 1 (lineal -C0.1)
21	-	19,61	-	-	(20) *Random Subspaces* 75 %
22	20, 12	-	-	-	(2) *Rotation Forest*
23	-	-	-	20,30	(51) *AdaBoost.R2-S*-Q
24	-	-	-	20,61	(52) *AdaBoost.R2-S*-E
25	-	-	20,97	-	(37) *AdaBoost.R2-S*-L
26	-	24,02	-	-	(19) *Random Subspaces* 50 %
27	-	24,38	-	-	(24) *AdaBoost.R2-S*-L
28	25,69	-	-	-	(5) *Iterated Bagging*
29	-	-	26,08	-	(39) *AdaBoost.R2-S*-E
30	-	-	26,13	-	(38) *AdaBoost.R2-S*-Q
31	26,66	-	-	-	(4) *Bagging* 100 %
32	27,38	-	-	-	(7) *Random Subspaces* 75 %
33	27,46	-	-	-	(1) SVM 0 (def)
34	27,64	-	-	-	(3) *Bagging* 75 %
35	-	27,72	-	-	(25) *AdaBoost.R2-S*-Q

Continua en la siguiente página ...

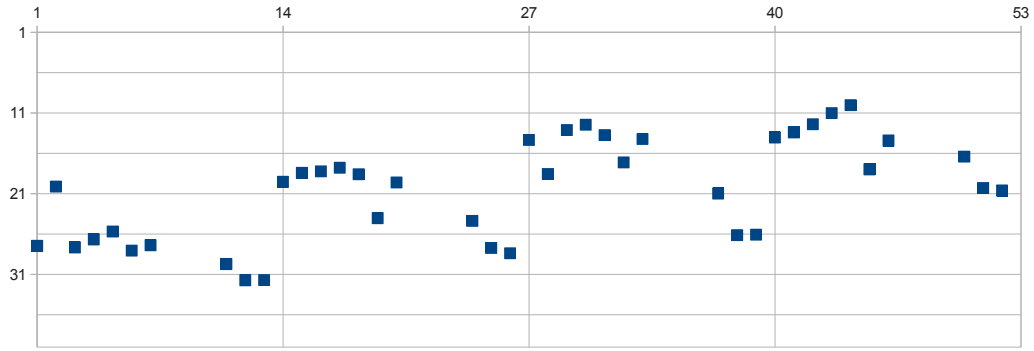

Figura 5.13: Gráfico de las posiciones promedio de la Tabla 5.7. Los algoritmos están en el orden que pone en la tabla entre ().

Tabla 5.7 (continuación): *Ranking* promedio global de los algoritmos con SVM.

#	posición				algoritmo
	SVM0	SVM1	SVM2	SVM3	
36	28,06	-	-	-	(6) *Random Subspaces* 50 %
37	-	28,38	-	-	(26) *AdaBoost.R2-S*-E
38	29,71	-	-	-	(11) *AdaBoost.R2-S*-L
39	31,70	-	-	-	(13) *AdaBoost.R2-S*-E
40	31,71	-	-	-	(12) *AdaBoost.R2-S*-Q

Las variantes de *Bagging* 75 % baten a la máquina de soporte vectorial salvo en las opciones por defecto de ésta. Las variantes de *Bagging* 100 % e *Iterated Bagging* baten a la máquina de soporte vectorial en todos los casos estudiados.

5.4.2.2. Resultados globales

La tabla de clasificación con los resultados globales de todas las configuraciones, se puede consultar en la Tabla 5.7, y en el gráfico de la Figura 5.13. En la tabla las configuraciones están ordenadas por la posición promedio, mientras que en la figura están ordenadas por el número que hay entre paréntesis delante del nombre del algoritmo.

Se puede observar que las dos máquina de soporte vectorial que normalizan los datos quedan en el primer cuartil de la tabla de clasificación, y la máquina de soporte vectorial con las opciones por defecto está en el cuarto. Por lo que las combinaciones que usan la máquina de soporte vectorial con las opciones por defecto parten en desventaja y solo el *Rotation Forest* logra escalar a casi la mitad de la tabla de clasificación (posición 22). Estando las otras ocho combinaciones están en el cuarto tercio de la tabla de clasificación.

Las combinaciones que usan las máquinas de soporte vectorial con una función lineal para el *kernel* ocupan la zona de la tabla de clasificación, quedando mejor las que normalizan los datos que tienen cinco combinaciones en el primer tercio y dos en cada uno de los otros tercios, mientras que la que no normaliza los datos tiene siete combinaciones en el tercio central y dos en el de cola.

También se observa que las mejores combinaciones son las que usan máquinas de soporte vectorial con funciones de base radial y normalizando el conjunto de datos dado que ocupa las tres primeras posiciones de la tabla de clasificación y tres pues-

tos más en el primer tercio de la tabla, con otras tres en el tercio central y ninguna en el último tercio. La mejor posición es para la combinaciones*Iterated Bagging* con máquinas de soporte vectorial que tienen esa combinación de parámetros, seguida por las dos versiones de *Bagging* con esa configuración de las máquinas de soporte vectorial.

5.4.3. Conclusiones

Se ha analizado el comportamiento de diferentes combinaciones de métodos usando máquinas de soporte vectorial como método base.

Se ha visto que no siempre es mejor usar combinaciones ya que algunas no mejoran a las máquinas de soporte vectorial por sí solas, solo *Bagging* 100 % e *Iterated Bagging* mejoran a la máquina de soporte vectorial que usan para construir sus modelos base en todas las configuraciones.

Se ha visto que la mejor combinación depende de la configuración de parámetros de máquinas de soporte vectorial, y que estos influyen mucho en el rendimiento final, siendo *Bagging* 100 %, *Iterated Bagging* y *Rotation Forest* los mejores en cada caso contemplado.

En los resultados globales se ha observado que las configuraciones que partían de máquinas de soporte vectorial que normalizaban los datos partían con ventaja. *Iterated Bagging* ha sido la combinación con mejores resultados globales, cuando usaba la variante de máquinas de soporte vectorial que funcionaba mejor que las otrasmáquinas de soporte vectorial.

Capítulo 6

Adaptación del *Rotation Forest* a problemas de regresión

Los resultados presentados en el presente capítulo han dado lugar a un artículo que ha sido publicado en un revista del JCR situada dentro del primer cuartil de su área de conocimiento [83].

El *Rotation Forest* es un algoritmo de combinación inicialmente propuesto para clasificación [94]. En ese contexto ha demostrado ser muy competitivo. En este capítulo se adapta el método a problemas de regresión y se analiza su desempeño. La conclusión es que en este tipo de problemas el *Rotation Forest* también ofrece muy buenos resultados.

6.1. El algoritmo

En el Algoritmo 6.1 se puede leer el pseudocódigo del *Rotation Forest* adaptado para regresión, en todo el capítulo se usa la notación de la Figura 6.1

Cada regresor D_i de la combinación se construye con un conjunto de datos transformado. Para transformar cada conjunto de datos se construye una matriz de rotación R_i^a, para lo que previamente se divide el conjunto de todas las características en K subconjuntos $\mathbf{F}_{i,j}$.

Algoritmo 6.1: Pseudocódigo del algoritmo *Rotation Forest*.

1 FASE DE ENTRENAMIENTO
Entrada: $X_{N\times n}$; N; n; $\mathbf{F}$; $Y_{N\times 1}$; L; K
Datos: Conjunto X (matriz $N\times n$) de N datos de entrenamiento, con un conjunto $\mathbf{F}$ de n características; valores de salida deseados Y (matriz $N\times 1$); valor L del tamaño de la combinación; número K de subconjuntos
Salida: **D**
Resultado: conjunto de L regresores $\{D_1, D_2 \dots D_L\}$
2 **Para** $i \leftarrow 1$ *hasta* L **haz**
3 Prepara una matriz de rotación R_i^a
4 **inicio**
5 Divide aleatoriamente el conjunto $\mathbf{F}$ en K subconjuntos $\mathbf{F}_{i,j}$, $\forall j = 1 \dots K$
6 **Para** $j \leftarrow 1$ *hasta* K **haz**
7 $X_{i,j} \leftarrow$ es la submatriz $(N\times K)$ de X con las características en $\mathbf{F}_{i,j}$
8 $X'_{i,j} \leftarrow$ es un muestreo (con repetición) de elementos de $X_{i,j}$
9 $C_{i,j} \leftarrow$ es la matriz de rotación que obtenemos aplicando `PCA`$(X'_{i,j})$
10 $R_i \leftarrow$ combinación de las matrices $C_{i,j}$ en una única matriz de rotación simple `// ver ec. 6.1`
11 $P_i \leftarrow$ es la matriz de permutación que ordena las características de $\mathbf{F}$ `// ver ec. 6.2`
12 $R_i^a \leftarrow P_i \cdot R_i$
13 $D_i \leftarrow$ `BuildRegressor`$(X \cdot R_i^a, Y)$ //Construir un regresor

14 FASE DE PREDICCIÓN
Entrada: $\mathbf{x}$; **D**
Datos: dato $\mathbf{x}$; conjunto de L regresores $D_1, D_2 \dots D_L$
Salida: r
Resultado: Valor $r \in \mathbb{R}$
15 $r \leftarrow \frac{1}{L} \cdot \sum_{i=1}^{L} D_i(\mathbf{x} \cdot R_i^a)$

$\mathbf{x} = [x_1, \ldots, x_n]$ es un punto de los datos con n atributos o características,
$y \in \mathbb{R}$ es el valor a predecir en ese punto,
X es el conjunto de datos de entrenamiento con N objetos o instancias,
$Y = [y_1, \ldots, y_N]^\top$ es el vector de las salidas de entrenamiento,
$\mathbf{F}$ es el conjunto de características,
L es un parámetro del algoritmo que nos determina el tamaño de la combinación,
K es otro parámetro del algoritmo que nos determina en cuantos subconjuntos dividir el conjunto $\mathbf{F}$ (otra alternativa sería el decir el tamaño M de los subconjuntos).

Figura 6.1: Notación utilizada en el capítulo.

Para cada subconjunto $\mathbf{F}_{i,j}$ se tiene un conjunto de datos $X_{i,j}$ con sólo la características de $\mathbf{F}_{i,j}$. En el método de clasificación *Rotation Forest* los datos de la clase propia se eliminaban de $X_{i,j}$, al no haber clases en regresión no se realiza este paso. Entonces se crea al azar una variación con repetición (*bootstrap*) del conjunto de datos $X_{i,j}$ obteniéndose $X'_{i,j}$.

Se aplica el análisis de componentes principales (PCA) a cada $X'_{i,j}$ para obtener las matrices de rotación $C_{i,j}$. Cada componente principal es una columna de esta matriz. El tamaño de esta matriz es $M \times M_{i,j}$. Donde M es de tamaño $\mathbf{F}_{i,j}$ y $M_{i,j}$ es de tamaño menor o igual a M; aunque suele ser igual porque con M características se pueden calcular hasta M componentes, salvo que alguno de los valores propios sea cero.

Las matrices $C_{i,j}$ se organizan en una matriz diagonal por bloques, creando una matriz de rotación R_i.

$$R_i = \begin{bmatrix} C_{i,1} & [\mathbf{0}] & \ldots & [\mathbf{0}] \\ [\mathbf{0}] & C_{i,2} & \ldots & [\mathbf{0}] \\ \vdots & \vdots & \ddots & \vdots \\ [\mathbf{0}] & [\mathbf{0}] & \ldots & C_{i,K} \end{bmatrix} \tag{6.1}$$

Como las características de X, seguramente, no están en orden, en la matriz R_i, ésta no se puede usar directamente como

matriz de rotación para el algoritmo; se necesita una segunda matriz de rotación que permuté las características en el orden correcto. Para K grupos de M características el orden es

$$\mathbf{F}_{i,1,1}, \mathbf{F}_{i,1,2} \ldots \mathbf{F}_{i,1,M}, \quad \mathbf{F}_{i,2,1}, \ldots \mathbf{F}_{i,2,M}, \quad \ldots \quad \mathbf{F}_{i,K,1}, \ldots \mathbf{F}_{i,K,M}$$

Se crea una matriz $n \times n$ de permutación P_i. Una matriz de permutación es aquella matriz cuadrada que está casi completa de ceros, salvo que tiene un uno, y solo uno, tanto por cada fila como por cada columna[1]. Para obtener el orden deseado para cada $\mathbf{F}_{i,j,k}$ ($j \in [1, K], k \in [1, M], \mathbf{F}_{i,j,k} \in [1, n]$):

$$P_i[\mathbf{F}_{i,j,k}\ , l] = \begin{cases} 1 & \text{si } l = (j-1)K + k + 1 \\ 0 & \text{si } l \neq (j-1)K + k + 1 \end{cases} \tag{6.2}$$

Entonces XP_i es el conjunto de datos ordenado. Y $R_i^a = P_i R_i$ es la matriz de rotación ordenada y el regresor D_i se entrena con XR_i^a y Y.

En la fase de predicción cada dato $\mathbf{x}$ se multiplica por R_i^a para rotarlo antes de aplicarle cada regresor D_i y la predicción sería $D_i(\mathbf{x}R_i^a)$. Finalmente la predicción de la combinación es el promedio de las predicciones de los regresores[2].

6.2. Los experimentos

Para los experimentos se usaran los 61 conjuntos de datos de la UCI [71] que se muestran en las tablas 4.3 y 4.4.

6.2.1. Configuraciones de los experimentos

Se ha usado Weka (ver Apartado 4.5) para ejecutar los experimentos. Weka suministra casi todos los métodos considerados

[1] La matriz identidad sería un ejemplo de matriz de permutación, es la que no mueve los datos.

[2] En clasificación, la probabilidad de la clase era la media de las probabilidades asignadas por cada clasificador de la combinación.

para los experimentos, con las excepciones de *Iterated Bagging* y *AdaBoost.R2* que han sido implementados para Weka.

El número de modelos en cada combinación se estableció en 100 y los métodos se evaluaron para minimizar la *media cuadrática de los errores* (RMSE[3]).

El resultado se obtuvo usando *validación cruzada* 5×2 particiones [19]. Los datos se dividen en dos mitades, una para comprobar la predicción que se ha entrenado con la otra mitad. Y luego se cambian los papeles de esas mitades. Este proceso se repite cinco veces. El resultado del informe representa la media de esas diez ejecuciones.

6.2.2. Algoritmos a comparar y opciones

Se han usado *árboles de regresión* tanto con poda *(P)* como sin poda *(U)* como modelos base, el algoritmo de poda usado fue *Reduced Error Pruning* (REP) [25].

Se han comparado seis familias de algoritmos de combinación. Para cada una de las familias se han considerado varias configuraciones de parámetros. Las familias y los parámetros usados son:

1. *Rotation Forest*. El tamaño de los subconjuntos de características fue de tres (M, el valor por defecto en Weka).

2. *Bagging* (ver Apartado 3.1). Se usaron dos tamaños de conjuntos: 100 % y 75 % del tamaño del conjunto de entrenamiento original.

3. *Random Subspaces* (ver Apartado 3.3). Se usaron dos tamaños de subespacios: 50 % y 75 % del tamaño del espacio original.

[3] *Root of Mean Squared Error*, en inglés.

4. *Iterated Bagging* (ver Apartado 3.6). Se usaron dos configuraciones: 10×10 y 5×20. En el primer caso se repite diez veces el proceso de construir diez árboles; y en el segundo caso se repite cinco veces el proceso de construir veinte árboles.

5. *AdaBoost.R2-W* (ver Apartado 3.2). El método utilizado para la construcción de *árboles de regresión* tiene en cuenta un peso de cada instancia (*reweighting* [34]). Existen tres propuestas de funciones de pérdida para este algoritmo: lineal, cuadrada y exponencial.

6. *AdaBoost.R2-S* (ver Apartado 3.2). Los *árboles de regresión* se construyeron sin tener en cuenta el valor de peso de las instancias. Sin embargo, el árbol se construye utilizando una muestra de los datos de entrenamiento y esa muestra se obtiene usando los pesos de la instancia como probabilidades (*resampling* [34]).

Todas las configuraciones anteriores se han combinado con la opción de construir los árboles con poda o sin poda. Por lo tanto se han probado un total de veintiséis configuraciones: dos configuraciones de *Rotation Forest*; cuatro configuraciones de *Bagging*, *Random Subspaces*, *Iterated Bagging* y seis configuraciones de *AdaBoost.R2-W*, *AdaBoost.R2-S*.

Al inicio de cada una de las ejecuciones de la validación cruzada 5×2 particiones se ha seleccionado la mejor configuración de cada familia de algoritmos calculando la media de validación de esos parámetros con 5 particiones.

6.2.3. Resultados

6.2.3.1. Resultados por familias de algoritmos

En la Tabla 6.1 se presentan los resultados agrupados por familias de algoritmos. El algoritmo *Rotation Forest* obtuvo el me-

jor resultado casi en la mitad de los conjuntos de datos, en 26 de los 61. La segunda posición fue para el algoritmo *AdaBoost.R2-S*, que ganó en casi la mitad de los restantes conjuntos de datos, en 14.

Se ha comparado *Rotation Forest* con los otros métodos para cada conjunto de datos usando *t-test estadístico corregido remuestreado* [81] (con un nivel de significación: 0,05). Cuando el *Rotation Forest* tiene una diferencia significativamente mejor se han marcado con el símbolo $\oplus$, y con el símbolo $\ominus$ cuando la diferencia es significativamente peor.

La Tabla 6.2 resume los resultados. Muestra el promedio de las posiciones [17] en que ha quedado cada una de las seis familias de algoritmos. Es decir, siguiendo el procedimiento [17], para cada conjunto de datos se han ordenado los algoritmos en función de sus resultados. La primera posición es para el mejor el siguiente en la posición 2 y así de forma sucesiva hasta la posición 6. Si varios algoritmos quedaban empatados en la misma posición de rendimiento, se les asignaba a todos el valor promedio de las posiciones que estaban empatadas (p.ej. se asignará posición 2,5 a cuatro algoritmos que queden empatados en el mejor resultado, es decir en las posiciones 1, 2, 3 y 4). Se ha calculado la posición promedio de cada familia como la media de las posiciones en todos los conjuntos de datos. *Rotation Forest* es la familia que mejor posición promedio ha obtenido.

En la Tabla 6.2 también se muestra el número de conjuntos de datos en los que el algoritmo *Rotation Forest* es mejor o peor que otros, y cuantas veces esa distancia entre los algoritmos es significativa. Del conjunto de métodos considerados *Rotation Forest* es el que tiene mejor balance de victorias contra derrotas.

Tabla 6.1: Resultados de los diferentes algoritmos y conjuntos de datos.

Rotation forest	*Bagging*	Random subspaces	Iterated bagging	*AdaBoost* R2-W	*AdaBoost* R2-S	Dataset
0,320895	0,333475	0,324957	0,330072	0,330939	0,340325	mbagrade
13,24999	17,05742 $\oplus$	13,83023	17,11714	16,67509	17,88310	elusage
3,130831	3,122334	3,187350	3,110877	3,003016	**2,965895**	vineyard
16,56246	**13,36005**	16,31507	14,79114	15,23908	13,93028	gascons
1 724,547	1 562,199	1 715,160	1 582,158	1 879,681	**1 552,271**	longley
0,687361	0,642223	0,711550	0,664124	0,697451	**0,640930**	diabetes-numeric
1 130 664	1 165 550	1 155 455	1 388 034	1 261 144	1 448 975	schlvote
0,094110	0,094570	0,099885	0,095404	0,096403	0,098632	baskball
147,1761	146,1992	148,0241	**146,1895**	154,8604	160,9588	veteran
70,87960	57,64301	69,86852	**56,41864**	66,53573	71,14636	detroit
3,493071	3,571728	3,685573	3,715490	3,804602	3,669260	sleep
13,64687	13,68913	14,48961	13,65187	14,25322	14,44445	bolts
0,601441	0,596108	0,583049	**0,555846**	0,601196	0,566442	cloud
0,770093	0,799162	0,919120	0,837626	0,825762	**0,749353**	servo
81,93499	82,55282	85,97422	96,43588	80,76914	**75,22683**	machine-cpu

$\oplus$, $\ominus$: mejora o deterioro estadísticamente significativo.

En negrita: el mejor resultados para cada conjunto de datos.

Continua en la siguiente página ...

Tabla 6.1 (continuación): Resultados de los diferentes algoritmos y conjuntos de datos.

Rotat.F.	*Bagging*	R.Subsp.	Itera.B.	ABR2-W	ABR2-S	C.Datos
1,869700	**1,860341**	2,049747	1,880415	1,911546	1,877279	pw-linear
47,24494	50,43081	54,47696	52,37454	49,62775	51,06973	pollution
16,11356	16,92420	16,72875	16,69582	16,68082	19,52100	fruitfly
454,6150	**451,1449**	460,2108	453,8834	455,2991	463,3115	lowbwt
94,3579	122,4886	138,9906 $\oplus$	134,9826	92,9600	**88,0561**	fishcatch
2,492983	2,673092	2,695393	2,523115	2,501045	2,553252	cpu-act
11,80740	11,24551	11,36964	11,36834	**11,17899**	12,38867	echo-months
2 544,270	2 811,593	2 657,883	2 805,742	2 654,938	2 715,811	auto-price
314,4732	426,2455 $\oplus$	370,6978 $\oplus$	430,3829 $\oplus$	449,7603 $\oplus$	433,7802 $\oplus$	pharynx
14,24495	21,48258	17,21359	19,61038	19,03034	18,84850	auto-horse
0,122045	0,114359	0,120967	0,114620	0,114491	**0,104622**	pyrimidines
5,996746	9,184314 $\oplus$	6,832310	9,118953	7,999304	8,289641	auto93
10,18011	10,28000	**10,14816**	10,35958	10,34739	10,91544	breast-tumor
516,7889	**495,2925**	499,8378	496,7877	527,9213	610,8574	strike
0,720179	0,735805	0,731524	0,740966	0,742897 $\oplus$	0,749728	sensory
0,363813	0,372314	0,367038	0,377535	0,397682	0,382483	hungarian
0,896754	0,945510	0,925044	0,954390	0,957223	0,935334	cleveland

$\oplus$, $\ominus$: mejora o deterioro estadísticamente significativo.

En negrita: el mejor resultados para cada conjunto de datos.

Continua en la siguiente página ...

Tabla 6.1 (continuación): Resultados de los diferentes algoritmos y conjuntos de datos.

Rotat.F.	*Bagging*	R.Subsp.	Itera.B.	ABR2-W	ABR2-S	C.Datos
51,36594	**50,81945**	51,09384	51,42633	51,84079	52,26418	cholesterol
2,147761	1,605292	1,951744	**1,585451**	1,619724	1,591565	bodyfat
2,833731	3,384313 ⊕	3,169765 ⊕	3,468779 ⊕	3,302314	3,270033	auto-mpg
880,6622	923,2950	925,9117	938,2374 ⊕	953,8157 ⊕	951,5385 ⊕	pbc
3,757041	3,934902	4,160726	3,970550	3,702022	**3,663753**	housing
0,188969	**0,188308**	0,188875	0,189023	0,202141	0,197127	quake
33,82990	**33,75127**	34,15153	33,94249	34,15160	34,57672	wisconsin
0,850263	1,061872	1,005375 ⊕	1,089851 ⊕	0,931931	0,858515	stock
697,6843	719,3639	751,6000	726,2079	944,4525	803,5071	meta
0,138358	**0,136369**	0,139854	0,136497	0,143053	0,137396	triazines
2,118008	2,181329 ⊕	2,212120 ⊕	2,192436 ⊕	2,257864 ⊕	2,224059 ⊕	abalone
0,000164	**0,000163**	0,000167 ⊕	0,000165	0,000172 ⊕	0,000168	delta-ailerons
0,001428	0,001437	0,001449 ⊕	0,001443 ⊕	0,001475 ⊕	0,001486 ⊕	delta-elevators
2,929343	3,084035	3,061580	2,973163	**2,928926**	2,969627	cpu-small
0,032778	0,032592	0,049778 ⊕	**0,031695**	0,032575	0,032268	bank-8FM
3,282299	**3,215566** ⊖	3,602815 ⊕	3,234148	3,366146	3,294971	puma8NH
67,97903	70,99767	76,63036	67,12399	67,22130	**63,19950**	cpu

⊕, ⊖: mejora o deterioro estadísticamente significativo.

En negrita: el mejor resultados para cada conjunto de datos.

Continua en la siguiente página ...

Tabla 6.1 (continuación): Resultados de los diferentes algoritmos y conjuntos de datos.

Rotat.F.	*Bagging*	R.Subsp.	Itera.B.	ABR2-W	ABR2-S	C.Datos
0,128836	0,152289 ⊕	0,173363 ⊕	0,134293	0,141330 ⊕	0,136040 ⊕	kin8nm
1,004859	1,013039 ⊕	1,486609 ⊕	1,026160 ⊕	1,019278 ⊕	1,036699 ⊕	2dplanes
5,239366	5,454475	7,210854 ⊕	5,318227	4,433465 ⊖	**4,347380** ⊖	pole
53 113,98	51 302,62 ⊖	52 227,55	50 681,83 ⊖	50 064,89 ⊖	**49 666,49** ⊖	cal-housing
0,002648	0,002932 ⊕	0,003226 ⊕	**0,002604**	0,002758	0,002679	elevators
30 579,55	**30 245,43**	31 476,34 ⊕	30 832,55	30 578,16	30 426,31	house-8L
0,085394	0,086097	0,090950 ⊕	0,086874	0,087752 ⊕	0,086358	bank-32nh
1,436373	1,361611 ⊖	1,750733 ⊕	1,230336 ⊖	**1,223865** ⊖	1,274957 ⊖	friedman
0,012832	0,007967 ⊖	0,014567	0,008041 ⊖	0,007974 ⊖	**0,007936** ⊖	puma32H
0,000163	0,000166 ⊕	0,000181 ⊕	0,000166	0,000167 ⊕	0,000166 ⊕	ailerons
0,229403	0,183499 ⊖	1,533746 ⊕	0,135598 ⊖	0,135096 ⊖	**0,126266** ⊖	mv
34 229,92	33 043,98	33 924,82	33 382,21	33 372,85	**33 002,01**	house-16H

⊕, ⊖: mejora o deterioro estadísticamente significativo.

En negrita: el mejor resultados para cada conjunto de datos.

Tabla 6.2: Posición promedio de la familia de algoritmos; victorias y derrotas significativas y totales.

Posición	Familia de algoritmos	Victorias RF		Derrotas RF	
		sign.	tot.	tot.	sign.
2,6721	*Rotation Forest*				
3,0492	*Bagging*	9	35	26	5
3,4754	*Iterated Bagging*	7	39	22	4
3,5410	*AdaBoost.R2-S*	6	40	21	5
3,8689	*AdaBoost.R2-W*	10	41	20	5
4,3934	*Random Subspaces*	17	48	13	0

Según el test de signo [17] (con 0,05 de nivel de significación), hay una diferencia significativa entre un algoritmo frente a otro, si éste gana en 39 o más conjuntos de datos de un total de 61 conjuntos. Por lo que, con la única excepción del *Bagging*, se puede afirmar que *Rotation Forest* es mejor que las otras alternativas, al serlo para al menos 39 conjuntos de datos.

La posición promedio y el número de victorias contra derrotas no tienen en cuenta las magnitudes de las diferencias de los resultados entre los algoritmos. La *puntuación cuantitativa* (QS, siglas de la expresión inglesa *quantitative qcore*) [100, 118] entre dos algoritmos, i y j, para un conjunto de datos k se define en función de la raíz de la de la media de los errores al cuadrado de esos algoritmos para ese conjunto de datos de la siguiente forma:

$$\mathrm{QS}_{i,j,k} = \frac{\mathrm{RMSE}_{k,i} - \mathrm{RMSE}_{k,j}}{\mathrm{máx}(\mathrm{RMSE}_{k,i}, \mathrm{RMSE}_{k,j})}$$

La Figura 6.2 muestra las puntuaciones cuantitativas (expresadas en %) de la comparación entre el *Rotation Forest* y las otras familias. En estos gráficos hay 61 barras (una por cada conjunto de datos). Los conjuntos de datos se han ordenado por la puntuación cuantitativa. Si la barra aparece por encima del eje de abscisas (signo positivo), indica que el algoritmo ganador en un conjunto de datos es el *Rotation Forest*; El mayor número

de barras por encima del eje de abscisas indican que el *Rotation Forest* es mejor que el otro algoritmo, se mayor número de barras por debajo del eje de abscisas indicarían el caso contrario. La longitud de la barras indican el porcentaje de mejora (o deterioro).

Con las puntuaciones cuantitativas se construye la *matriz de puntuaciones* [100, 118] como la media de las puntuaciones para todos los conjuntos de datos (C).

$$\mathrm{SM}_{i,j} = \sum_{k=1}^{C} \frac{1}{C} \mathrm{QS}_{i,j,k}$$

La Tabla 6.3 muestra esta matriz de puntuaciones. *Rotation Forest* tiene mejores puntuaciones comparado con cualquiera de los otros algoritmos. La última columna de la tabla muestra la suma de cada fila, lo cual, es una medida de la eficiencia de cada método. Se observa que *Rotation Forest* es el mejor con diferencia.

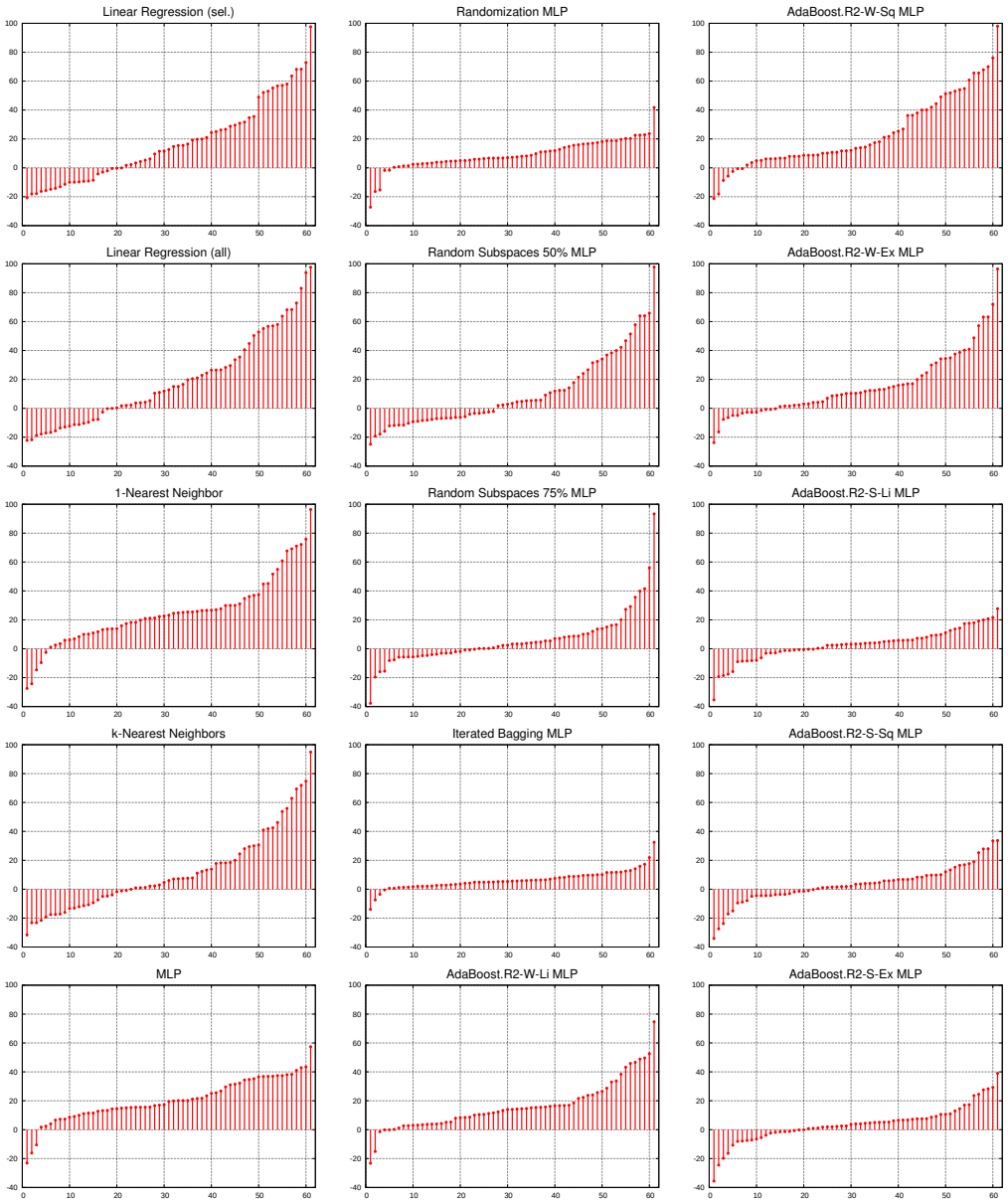

Figura 6.2: Puntuaciones cuantitativas.

Tabla 6.3: Matriz de resultados.

Rotation Forest	*AdaBoost* R2-S	Iterated *Bagging*	*Bagging*	*AdaBoost* R2-W	Random Subspaces	Algoritmo	Total
0,0000	1,0771	1,7373	1,8133	1,8600	8,1671	*Rotation Forest*	14,6548
−1,0771	0,0000	0,7088	0,8853	0,8333	6,1555	*AdaBoost.R2-S*	7,5059
−1,7373	−0,7088	0,0000	0,1301	0,0587	5,6440	*Iterated Bagging*	3,3867
−1,8133	−0,8853	−0,1301	0,0000	−0,1183	5,9714	*Bagging*	3,0245
−1,8600	−0,8333	−0,0587	0,1183	0,0000	5,4909	*AdaBoost.R2-W*	2,8572
−8,1671	−6,1555	−5,6440	−5,9714	−5,4909	0,0000	*Random Subspaces*	−31,4291

Los valores mostrados por las tablas 6.2 y 6.3 coinciden en señalar que el mejor algoritmo es el *Rotation Forest*, pero difieren en el orden de los demás algoritmos. Las diferencias radican en que la matriz de puntuaciones tiene en cuenta la magnitud de las diferencias de los errores, no sólo las posiciones en que han quedado ordenados los algoritmos.

6.2.3.2. Resultados individuales

Los resultados anteriores eran comparando el mejor algoritmo de cada familia. Pero se puede estudiar el rendimiento de cada uno de los diferentes algoritmos por separado, sin agruparlos en familias, para de ese modo determinar las opciones que dan los mejores resultados.

La Tabla 6.4 muestra la posición promedio de cada algoritmo. El uso de árboles podados o sin podar se han marcado con *(P)* y *(U)*, respectivamente. La función de pérdida utilizada por *AdaBoost* se ha marcado con los sufijos $-L$ (lineal), $-Q$ (cuadrada) y $-E$ (exponencial). Para hacer la comparación más completa se han añadido algunos de los algoritmos básicos: árboles simple, combinación de árboles (promedio de la predicción de un conjunto de árboles en el que cada uno se entrena con una partición aleatoria diferente de la partición de entrenamiento y se poda con el resto de los datos de entrenamiento), vecinos más cercanos (usando sólo un vecino o seleccionando el número óptimo de vecinos con validación cruzada) y regresión lineal (usando todas las características o sólo una selección de ellas). El algoritmo con mejor posición promedio es el *Rotation Forest* con árboles sin podar como método base.

6.2.4. Evolución del error

Se han usando 100 árboles en la combinación para calcular el error anteriormente mostrado. El error varía en función del

Tabla 6.4: Posición promedio.

Posición	Algoritmo
8,64	*Rotation Forest (U)*
11,21	*Iterated Bagging* 5x20 *(P)*
11,58	*AdaBoost.R2-S*-E *(P)*
11,68	*Bagging* 75 % *(U)*
12,66	*Bagging* 100 % *(U)*
13,02	*AdaBoost.R2-S*-C *(P)*
13,49	*AdaBoost.R2-S*-L *(P)*
14,09	*Iterated Bagging* 5x20 *(U)*
14,19	*Bagging* 100 % *(P)*
14,40	*Iterated Bagging* 10x10 *(P)*
14,87	*AdaBoost.R2-S*-C *(U)*
14,90	*Rotation Forest (P)*
15,55	*AdaBoost.R2-S*-L *(U)*
15,89	*AdaBoost.R2-W*-C *(U)*
16,13	*AdaBoost.R2-S*-E *(U)*
16,22	*Bagging* 75 % *(P)*
16,76	*AdaBoost.R2-W*-E *(P)*
17,39	Combinación de árboles *(P)*
17,68	*AdaBoost.R2-W*-L *(P)*
17,77	*Iterated Bagging* 10x10 *(U)*
18,06	*AdaBoost.R2-W*-C *(P)*
18,23	*Random Subspaces* 50 % *(U)*
18,30	*Random Subspaces* 75 % *(P)*
18,78	Regresión lineal (todos)
18,80	K-vecinos
18,99	Regresión lineal (selección)
19,31	*AdaBoost.R2-W*-L *(U)*
19,88	*Random Subspaces* 75 % *(U)*
19,98	*AdaBoost.R2-W*-E *(U)*
20,84	*Random Subspaces* 50 % *(P)*
25,94	Árboles *(P)*
27,52	1-vecino
28,23	Árboles *(U)*

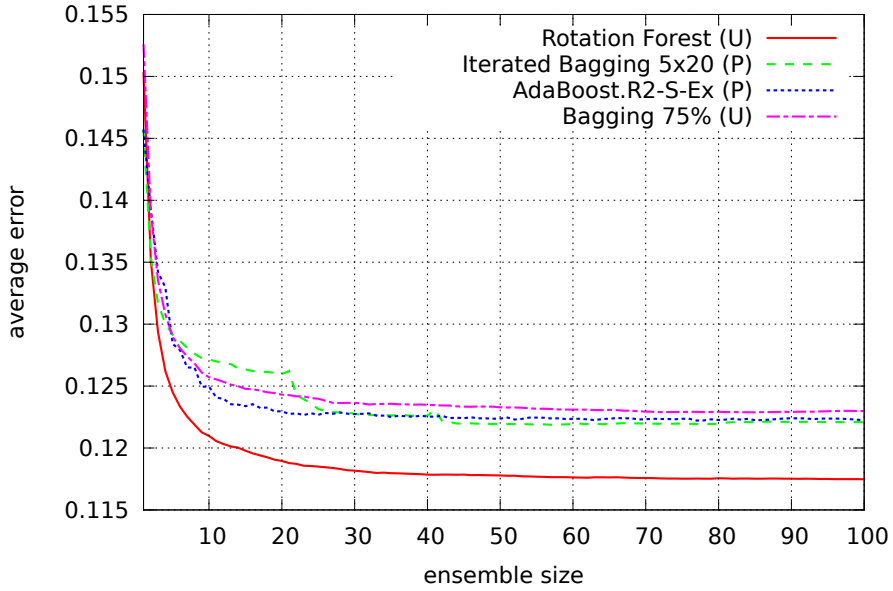

Figura 6.3: Media del error en función del tamaño de la combinación.

tamaño de la combinación, y el algoritmo mejor puede variar en función de ese tamaño. El algoritmo mejor lo podemos elegir en función del valor del error o del número de veces que queda mejor que los otros.

En la Figura 6.3 se muestra la variación de la media del error, para todos los conjuntos de datos, de los algoritmos con mejores posiciones en la Tabla 6.4. Como las variables a predecir en cada conjunto de datos tienen distintas magnitudes, para obtener estos resultados comparables se han normalizado todas las salidas al intervalo cerrado $[0; 1]$; y se han estimado los errores mediante una validación cruzada 5×2 particiones. Se puede observar que *Rotation Forest* tiene el menor error promedio incluso cuando la combinación tiene pocos elementos.

En la Figura 6.4, se muestra el porcentaje de conjuntos de datos en los que cada uno de estos cuatro algoritmos resulta la mejor combinación. También se puede observar que *Rotation Forest* tiene el mejor porcentaje, salvo cuando la combinación tiene menos de cinco clasificadores.

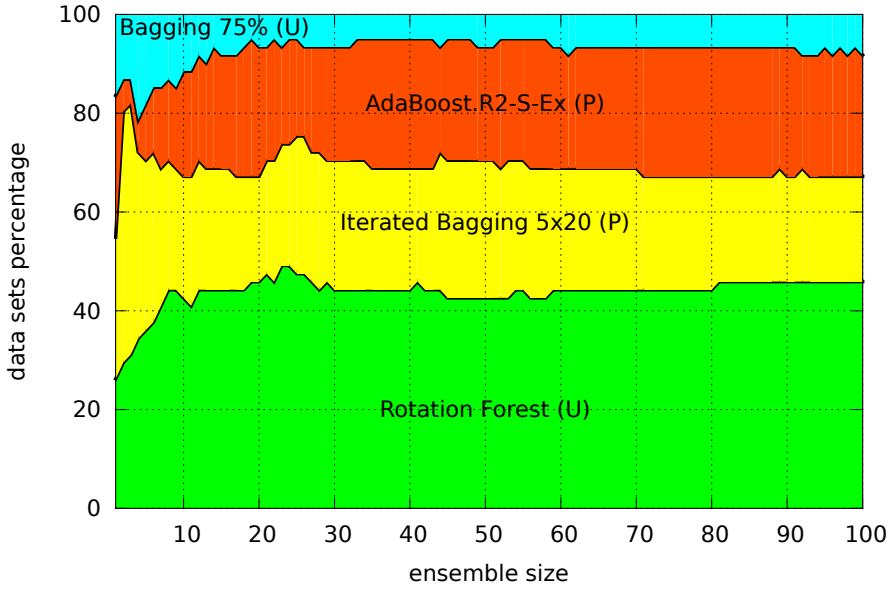

Figura 6.4: Diagrama porcentual de vencedores, en función del tamaño de la combinación.

6.2.4.1. Diagramas de error y diversidad de los clasificadores

El éxito de una combinación de modelos es estar formado por modelos con pocos errores, pero que sean diferentes. Estos dos objetivos suelen entrar en conflicto mutuo, porque si dos modelos tiene poco error, no pueden ser muy distintos. Para analizar el comportamiento de los algoritmos de combinación se proponen varias medidas de la diversidad [61].

Una de las técnicas es usar diagramas de diversidad-error [77]. Estos son gráficos de dispersión en el que hay un punto para cada par de modelos. El eje horizontal representa la diversidad entre dos modelos; en clasificación, se usa habitualmente la medida estadística κ (kappa). El eje vertical representa la media de error de los dos modelos.

En regresión, se pueden considerar varias medidas de error,

se ha decidido usar al RMSE

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(y_i - r_i)^2}{n}}$$

donde y_i representa el valor actual y r_i es el valor predicho.

Se usa como medida de diversidad el valor opuesto a la RMSE de la predicción de un modelo r respecto a otro r':

$$RMSE \text{ opuesta} = -\sqrt{\sum_{i=1}^{n} \frac{(r_i - r'_i)^2}{n}}$$

donde r_i y r'_i son las predicciones de los dos modelos. La razón de usar el valor opuesto al error en estos diagramas, cuando se usa κ, es que valores menores indican mayor diversidad.

El inconveniente de usar la RMSE para las medidas de error y de la diversidad es que sus valores no están acotados. En estos diagramas, para solucionar este problema, se transformaron los conjuntos de datos para que las salidas se escalen al intervalo cerrado $[0, 1]$. Los resultados de estos diagramas también se obtuvieron usando validación cruzada 5×2 particiones.

La Figura 6.5 muestra los diagramas de diversidad-error para uno de los conjuntos de datos de los tres métodos que están entre los 4 mejores según la Tabla 6.4. No se ha incluido el otro método, *Iterated Bagging*, por que al ser una combinación de combinaciones, sus diagramas no pueden ser comparados con otros métodos. Ya que en este algoritmo, sólo los métodos miembros de la primera combinación predicen la variable original, las predicciones del resto de métodos intentan compensar los errores de las combinaciones previas. Al no tener el mismo objetivo, no es comparable la diversidad de métodos de dos fases diferentes de la combinación.

Para resumir los diagramas diversidad-error de todos los conjuntos de datos se utilizan los *diagramas de movimiento relativo* [78]. Para comparar los diagramas obtenidos por dos métodos,

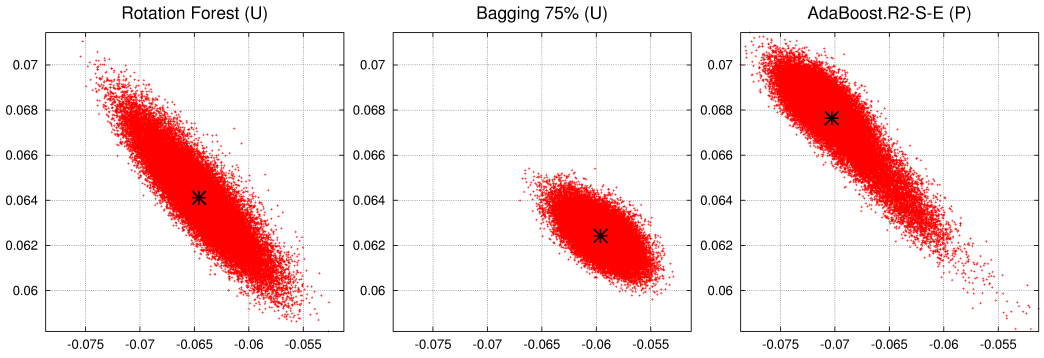

Figura 6.5: Diagramas de diversidad-error para el conjunto de datos *ailerons*. El símbolo '✳' marca el punto promedio.

primero, para cada uno de los conjuntos de datos, se calcula un vector que une el centro del diagrama del primer método con el centro del diagrama del segundo, a continuación se desplazan todos los vectores para que su origen sea el origen de coordenadas. Consiguiendo un diagrama que resume los resultados obtenidos para todos los conjuntos de datos. Se cuenta el número puntas de flecha que acaban en cada cuadrante del diagrama. Si el segundo método tiene menos error que el primero, habrá mas puntas de flecha en los cuadrantes inferiores que en los superiores; si el segundo método aumenta en diversidad, habrá mas puntas de flecha en los cuadrantes de la izquierda que en los de la derecha.

En la Figura 6.6 se pueden ver algunos de diagramas de movimiento relativo de diversidad-error. Por ejemplo, comparando *Bagging* y *Rotation Forest*, aparecen 24 flechas que apuntan hacia arriba y a la izquierda, esto significa que en 24 conjunto de datos los modelos del segundo algoritmo han aumentado la diversidad promedio, pero también su error promedio.

Comparando *Bagging* y *Rotation Forest* no se observan patrones claros: 25 flechas suben y 36 bajan; 32 flechas apuntan a la derecha y 29 a la izquierda. Además el que la mejora en un eje esté compensada con el deterioro en el otro $(24 + 28)$ es mucho más frecuente que la mejora o deterioro en ambos ejes $(8 + 1)$.

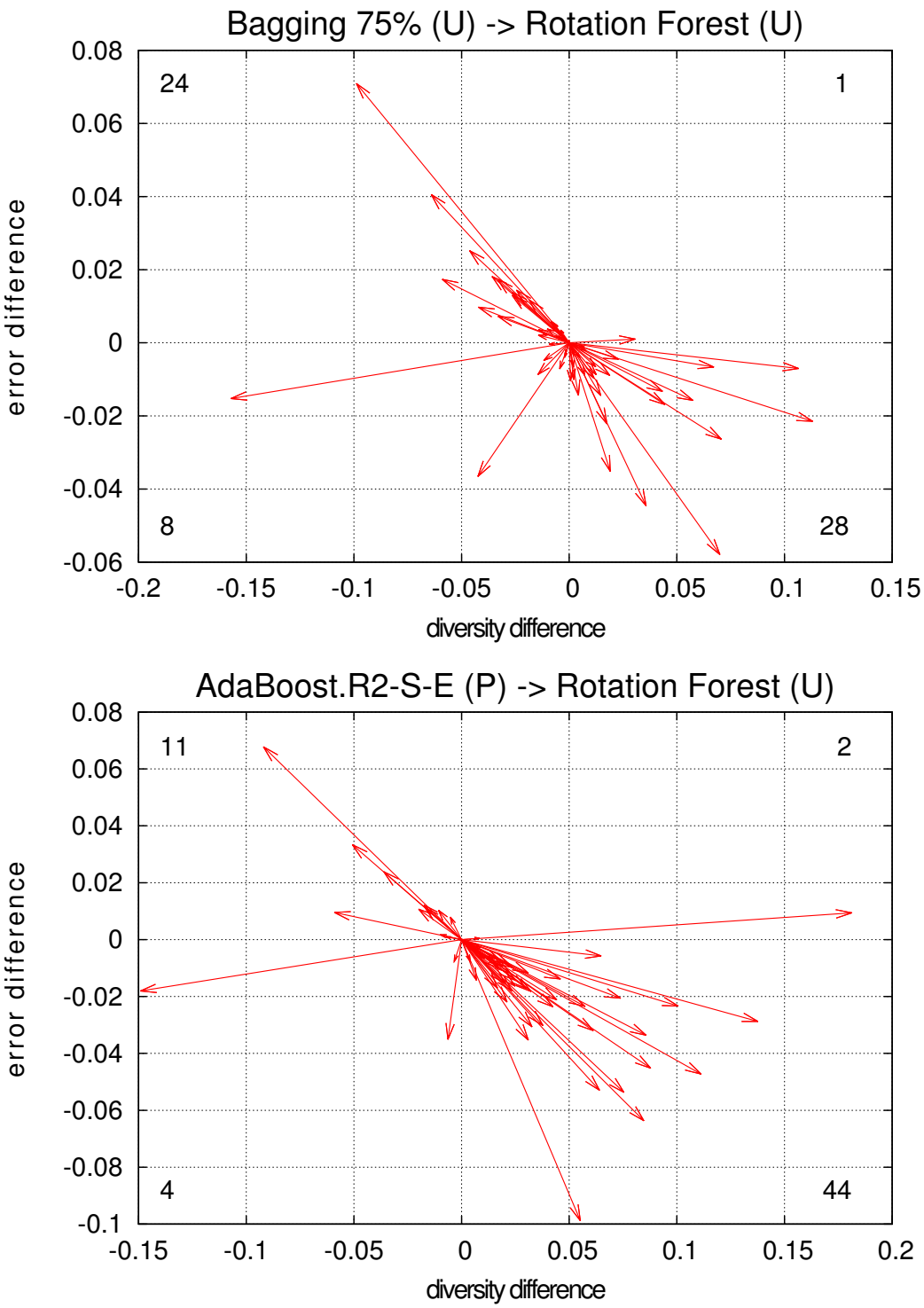

Figura 6.6: Diagramas de movimiento relativo de la diversidad y el error.

La comparación entre *AdaBoost.R2-S* y *Rotation Forest* es mucho más clara. La mayor parte de las flechas apuntan hacia abajo (48) y a la derecha (46). Esto significa que los modelos obtenidos con *Rotation Forest* tienen menos diversidad, pero mayor precisión que los obtenidos con *AdaBoost.R2-S*. Este comportamiento del *Rotation Forest* para regresión es similar al que tiene en clasificación [94].

6.3. Resultados por tamaño

Posteriormente se han repetido los experimentos con una validación cruzada 10×10 particiones para tener más seguridad en el experimento. La nueva posición promedio de cada algoritmo se muestra en Tabla 6.5. Se puede observar, en la tabla, que hay pocos cambios significativos en los primeros puestos, salvo la bajada de los algoritmos que usan *Iterated Bagging* 5×20.

Si se comparan los errores cometidos por los algoritmos con mejores puestos, relativizándolo entre ellos, se puede observar en la Figura 6.7 que el error cometido por *AdaBoost.R2-S*-E *(P)* es mayor que el cometido por *Rotation Forest (U)* en la mayor parte de los ejemplos (más puntos por encina del eje X), pero haciendo una la recta de regresión, se observa que el error cometido por *AdaBoost.R2-S*-E *(P)* mejora con los conjuntos con muchos ejemplos, siendo mejor en todos los conjunto de datos con más de mil ejemplos. De la misma forma, en las Figuras 6.8 y 6.9 se puede observar que el error de *Rotation Forest (U)* es menor que el de los métodos *Bagging* 100 % *(U)* y *Bagging* 75 % *(U)* en más casos, pero también estos conjuntos superan al *Rotation Forest (U)* en los conjuntos con muchos ejemplos, en estos casos a partir de dos mil ejemplos.

Tabla 6.5: Posición promedio con validación 10×10.

Posición	Algoritmo	variación
8,61	*Rotation Forest (U)*	⇌
10,67	*AdaBoost.R2-S*-E *(P)*	↑
10,81	*Bagging* 100 % *(U)*	⇑
11,62	*Bagging* 75 % *(U)*	⇌
12, 17	*AdaBoost.R2-S*-L *(P)*	⇑
12,64	*Bagging* 100 % *(P)*	↑⇑
13,72	*Rotation Forest (P)*	↑⇑⇑
13, 73	*AdaBoost.R2-S*-E *(U)*	↑⇑⇑⇑
13,78	*AdaBoost.R2-S*-L *(U)*	⇑⇑
14,25	*AdaBoost.R2-S*-C *(P)*	⇓⇓
15,03	*Bagging* 75 % *(P)*	↑⇑⇑
15,89	*Random Subspaces* 75 % *(P)*	↑⇑⇑⇑⇑⇑
16, 33	*AdaBoost.R2-S*-C *(U)*	⇓
16,38	*AdaBoost.R2-W*-L *(P)*	↑⇑⇑
16,54	*AdaBoost.R2-W*-C *(U)*	↓
16,69	*Iterated Bagging* 10x10 *(U)*	⇓⇓⇓
17,39	*AdaBoost.R2-W*-E *(P)*	⇌
17,66	*Iterated Bagging* 5x20 *(P)*	⇓⇓⇓⇓⇓⇓⇓⇓⇓
17,07	*Iterated Bagging* 10x10 *(P)*	↑
17,44	*AdaBoost.R2-W*-L *(U)*	↓⇓⇓⇓
17,70	*Random Subspaces* 50 % *(U)*	↑
18,24	K-vecinos *	↑⇑
18,51	Regresión lineal (selección*)	
18,55	Regresión lineal (*)	
18,71	*Random Subspaces* 75 % *(U)*	↑⇑
18,73	*AdaBoost.R2-W*-C *(P)*	↓⇓⇓
18,84	*AdaBoost.R2-W*-E *(U)*	⇑
19,56	*Random Subspaces* 50 % *(P)*	⇑
19,61	Regresión lineal (todos*)	
21,61	*Iterated Bagging* 5x20 *(U)*	⇓⇓⇓⇓⇓⇓⇓⇓⇓⇓
25,59	Árboles *(P)*	⇌
27,56	Árboles *(U)*	⇑
28,35	1-vecino	↓

Por cada flecha ⇑ un método ha adelantado, en la tabla, a otros dos, con respecto a la tabla con validación 5×2; por cada flecha ⇓ un método ha sido adelantado por otros dos; las fechas ↑ y ↓ indica variaciones de 1 puesto y ⇌ indica que no ha cambiado de posición relativa.

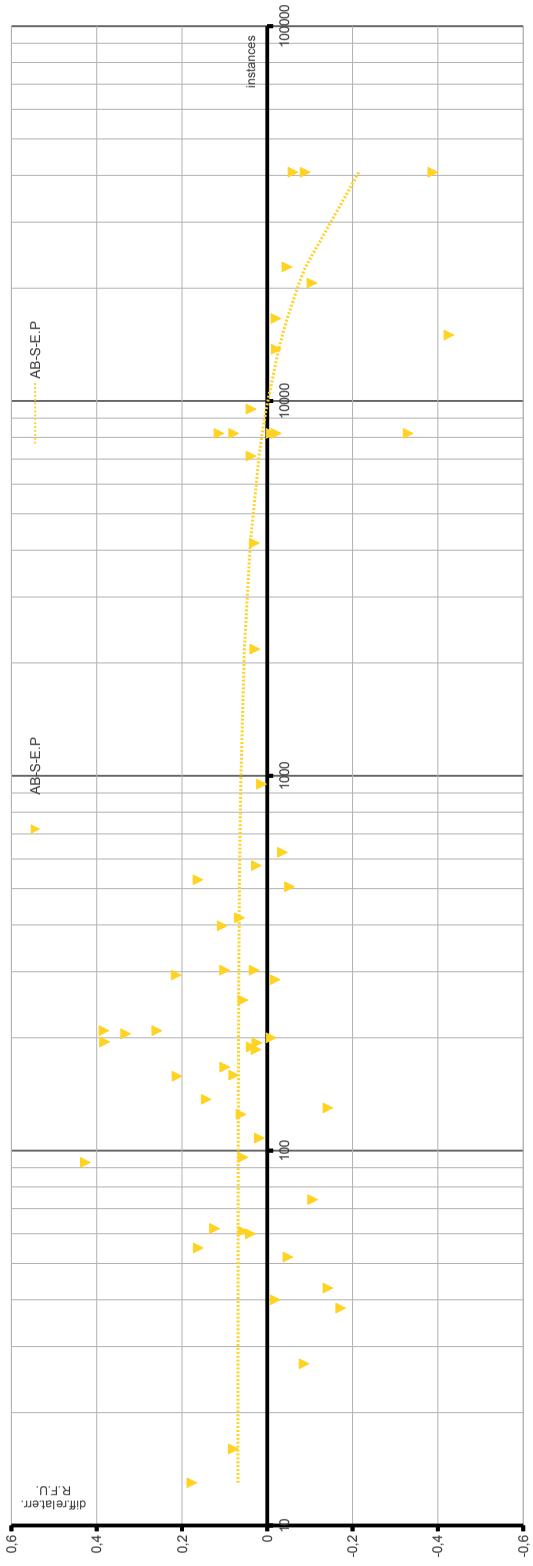

Figura 6.7: Línea de tendencia del error relativo de *AdaBoost.R2-S*-E *(P)*, en función del número de ejemplos del conjunto de datos, con referencia al error de *Rotation Forest (U)*.

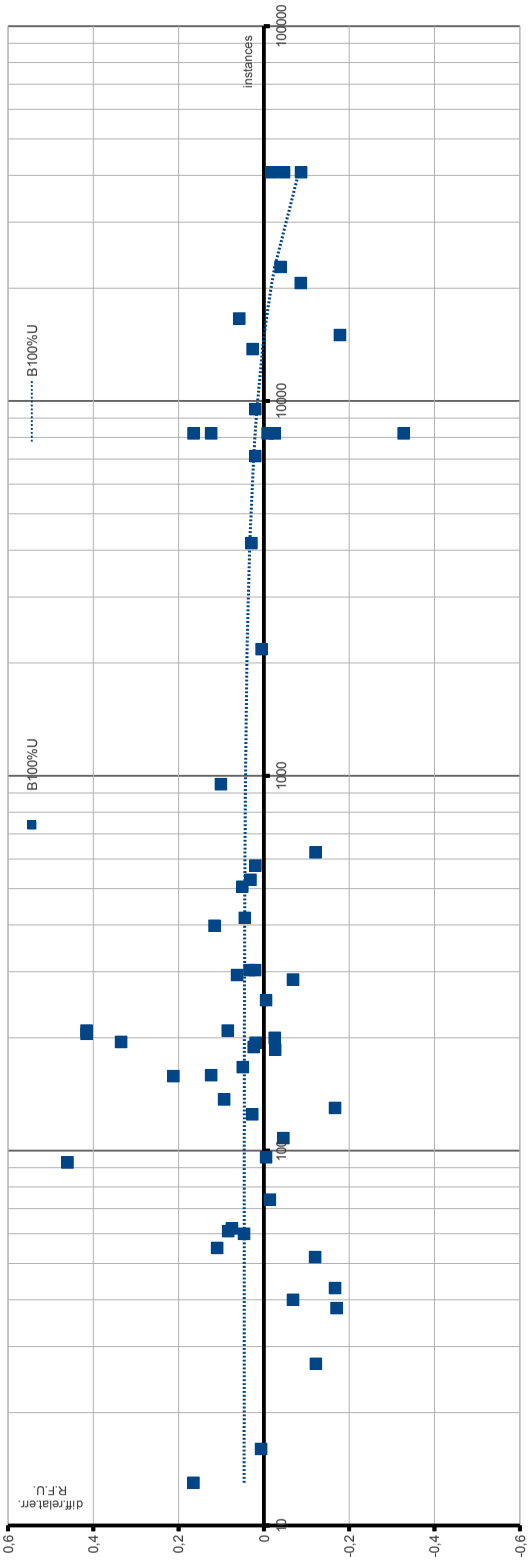

Figura 6.8: Línea de tendencia del error relativo de *Bagging* 100 % *(U)*, en función del número de ejemplos del conjunto de datos, con referencia al error de *Rotation Forest (U)*.

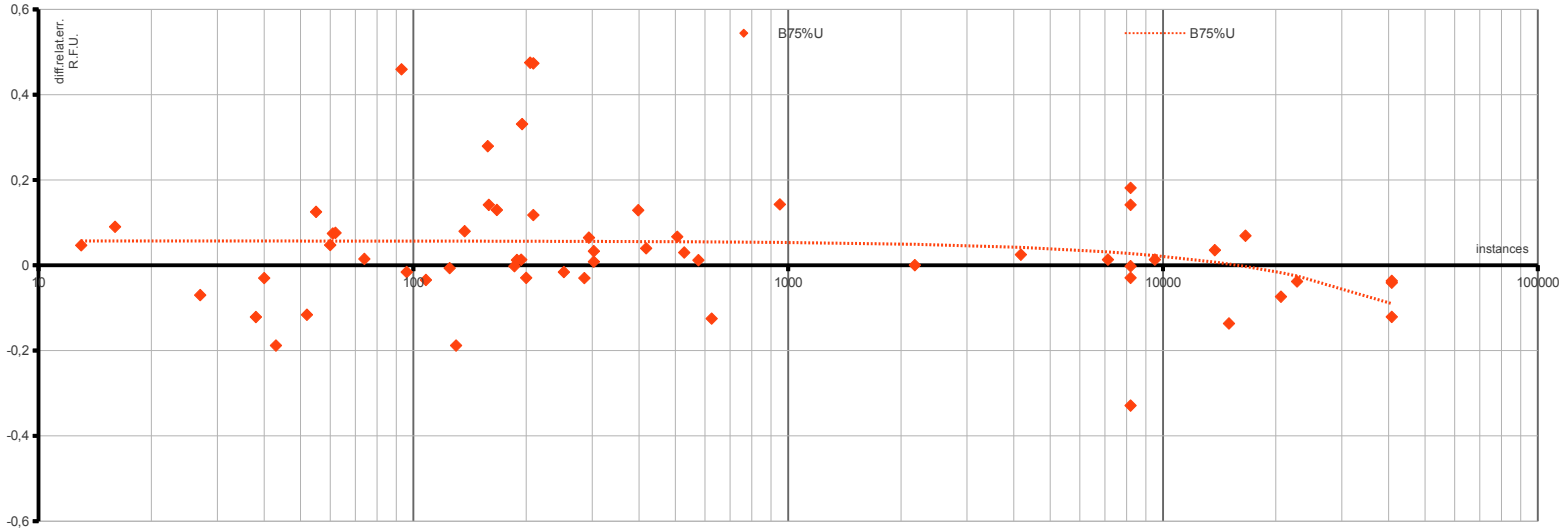

Figura 6.9: Línea de tendencia del error relativo de *Bagging* 75 % *(U)*, en función del número de ejemplos del conjunto de datos, con referencia al error de *Rotation Forest (U)*.

6.4. Conclusiones

Se ha estudiado el comportamiento de *Rotation Forest* para resolver problemas de regresión usando 61 conjuntos de datos. Se han utilizado como modelos base *árboles de regresión*. *Rotation Forest* obtiene resultados favorables cuando se le compara con *Bagging*, *Iterated Bagging*, *Random Subspaces* y *AdaBoost.R2*.

Se ha usado diagramas diversidad-error para comparar el comportamiento de *Rotation Forest* con los dos mejores contrincantes *Bagging* y *AdaBoost.R2*. No se ha podido observar un patrón claro en la comparación con *Bagging*. Pero en la comparación con *AdaBoost.R2*, *Rotation Forest* suele dar miembros con menos diversidad pero mayor precisión. En este caso, generalmente, la mejora del error compensa la perdida de diversidad.

El algoritmo *Rotation Forest* tiene un parámetro que es el tamaño del grupo de características. En los experimentos se ha usado este valor fijo en 3 características por grupo, tal como está definido por defecto en Weka. Lo que hace que sus resultados puedan ser mejorados si se ajusta el parámetro para cada conjunto de datos.

Capítulo 7

Un estudio experimental de métodos de proyección lineal para problemas de regresión

Los resultados presentados en este capítulo se han presentado en el congreso internacional ICPRAM[1] en el 2012 [87]

7.1. Introducción

Con frecuencia la dimensión intrínseca de un conjunto de datos, el número de variables o características necesarias para representarlo, es menor, o incluso mucho menor, que la dimensión real que exhibe el conjunto de datos. Un ejemplo que ilustra perfectamente esto es el de la Figura 7.1, que fue utilizado por Tenenbaun et al. en 2000 [106], en él tenemos un conjunto de datos formados por imágenes de manos se puede caracterizar por dos variables (dimensión intrínseca 2), la rotación de la muñeca y el ángulo de extensión de los dedos, a pesar de que su dimensión es 4096 (dado que se tratan de imágenes de 64×64 píxeles). Es decir, manteniendo constantes la distancia a la que se toma la foto y las condiciones de luz, todas las imágenes de manos tomadas con la misma rotación y extensión de dedos serán

[1]International Conference on Pattern Recognition Applications and Methods.

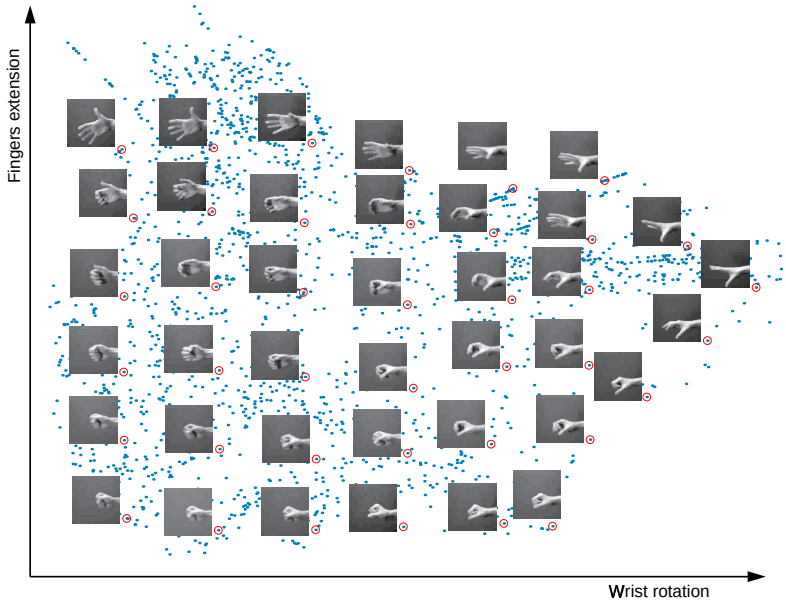

Figura 7.1: Ejemplo de imágenes utilizado por Tenenbaun et al. [106].

aproximadamente iguales, con lo que el valor de los 4096 píxeles se puede determinar bastante bien con tan solo conocer esos dos valores.

En el ámbito de la *Minería de Datos* existe gran interés en el estudio de métodos que permitan el descubrimiento de la dimensión intrínseca de los conjuntos de datos, lo que ha dado lugar a un área de investigación que ha sido denominada «aprendizaje de variedades » (en inglés «*manifold learning*») [106, 98, 66], habitualmente centrada en la determinación de relaciones no lineales, y a los métodos de selección o extracción de características (en inglés «*feature selection*» y «*feature extraction*») [45, 72], que se centran en las relaciones lineales por ser más interesantes al ser más fáciles de interpretar.

El interés por descubrir la dimensión intrínseca es doble. Por una parte, reducir la dimensión del conjunto de datos mitiga los efectos de la *maldición de la dimensionalidad*, término acuñado por Richard Bellman [2] para describir el hecho de que algunos problemas se conviertan en intratables al aumentar el número de

variables. En el caso de problemas de *Minería de Datos* tiene que ver con el hecho de que el número de instancias necesarias para resolver un problema de aprendizaje crezca de forma exponencial con el número de variables. Por otra parte, tener conocimiento de las variables de las que dependen de forma más directa los valores a predecir es de por sí muy valioso para el analista de datos.

Una forma muy sencilla de reducir la dimensión de un conjunto es encontrar una matriz de proyección que proyecte el conjunto de datos en un espacio de menor dimensión que el de partida. Es decir, en el nuevo conjunto de datos tenemos un número menor de variables que son combinación lineal de las de partida. La dificultad estará en encontrar una proyección que preserve algunas características interesantes del conjunto de partida.

Este capítulo se centra en este tipo de métodos de proyección lineal[2]. Entre los métodos no supervisados de esta categoría el más utilizado, sin duda, es el *análisis de componentes principales* [56] (PCA) que busca preservar la varianza del conjunto de datos. Entre los métodos supervisados orientados a la clasificación los dos más referenciados son el *análisis discriminante de forma lineal* [30] (LDA) y *análisis discriminante con técnicas no paramétricas* [37] (NDA), ambos buscan una proyección que maximice la separación entre clases y minimice la dispersión de las instancias dentro de su clase. Un tercer método supervisado con el mismo objetivo es el que se plantea combinar en el mismo método PCA y LDA y que se conoce como *análisis discriminante híbrido* [107] (HDA). Además, también existen métodos supervisados orientados a la regresión que buscan encontrar la relación lineal que aumenten la correlación con la variable dependiente. Entre estos cabe destacar la *regresión inversa por rebanadas* [68] (SIR) y las *direcciones principales de la matriz*

[2]El motivo de este estudio es el de eventualmente continuar la investigación sustituyendo el PCA del *Rotation Forest* por algunos de estos métodos.

hessiana [69] (PHD) y también los más recientes *SIR local* [116] (LSIR), *LDA para regresión* (LDAr) y *PCA ponderado* [63] (WPCA).

En este capítulo nos centramos en los métodos de proyección lineal para regresión. Se completa el estudio experimental de LSIR, LDAr y WPCA, dado que en los artículos donde se presentan sólo utilizan dos conjuntos reales, en el caso de LSIR, y tres conjuntos para los dos últimos métodos; y usando las ideas utilizadas para estos métodos, también se presentan nuevos métodos que se incluyen en el estudio comparativo.

El resto de la sección se estructura como sigue. El Apartado 7.2 da los detalles de los métodos, presentando además un enfoque unificado. El Apartado 7.3 presenta los nuevos métodos. El Apartado 7.4 explica los detalles de como se hizo el estudio y presenta los resultados. Por último, el Apartado 7.5 resume las conclusiones.

7.2. Revisión de antecedentes

Considérese un conjunto de n datos y valores $\{\mathbf{x}_i, y_i\}_{i=1}^n$ en los que $\mathbf{x}_i \in \mathbb{R}^{d\times 1}$ e $y_i \in \mathbb{R}$ –en un contexto más general, se podrían considerar pares $\{\mathbf{x}_i, \mathbf{y}_i\}$ con $\mathbf{y}_i \in \mathbb{R}^{t\times 1}$, pero en este capítulo sólo se consideran conjuntos de datos para los que $t = 1$. El objetivo es tratar de encontrar la combinación lineal de atributos $f_i = \mathbf{w}_i^T\mathbf{x}$ que de lugar a características f_i que mejor expliquen el valor a predecir y.

Todos los métodos que se presentan a continuación pueden plantearse como un problema de optimización en el que la función a maximizar es de la forma:

$$J(W) = \frac{|W^T A W|}{|W^T B W|} \tag{7.1}$$

donde las columnas de la solución óptima de W, se puede obtener

resolviendo el siguiente problema de valores propios generalizado

$$A\mathbf{w}_k = \lambda_k B\mathbf{w}_k, \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d \tag{7.2}$$

que puede resolverse como $B^{-1}Aw_k = \lambda_k \mathbf{w}_k$, un problema clásico de valores propios, pero que puede ser sensible al mal condicionamiento de B (cuando el determinante es próximo a cero).

Lo que va a cambiar de un método a otro es la forma de calcular las matrices A y B que aparecen en numerador y denominador.

7.2.1. PCA: Proyección lineal no supervisada

PCA son las siglas inglesas de *Principal Component Analysis*. En PCA [56] la matriz B en la Ecuación 7.2 es simplemente la matriz identidad y la matriz A es la matriz de covarianzas:

$$A = S_x = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^T$$
$$B = I \qquad (matriz\ identidad)$$

donde:
$\overline{\mathbf{x}}$ es la promedio de los valores $\mathbf{x}_i$

$$\overline{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$$

La Ecuación 7.2 queda reducida así a un problema clásico de valores propios

$$S_x\mathbf{w}_k = \lambda_k\mathbf{w}_k, \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$$

7.2.2. Métodos de proyección lineal supervisada para clasificación

7.2.2.1. LDA: Análisis discriminante de forma lineal

LDA son las siglas inglesas de *Linear Discriminant Analysis*. En LDA [30] la matriz del numerador es la denominada matriz

de covarianzas entre clases, en inglés *between-covariance matrix*, y la del denominador es la matriz de covarianzas interna de cada clase, en inglés *within-covariance matrix*, que se definen como:

$$A = S_b = \frac{1}{n} \sum_{c=1}^{N_c} n_c \, (\overline{\mathbf{x}}_c - \overline{\mathbf{x}})(\overline{\mathbf{x}}_c - \overline{\mathbf{x}})^T$$

$$B = S_w = \frac{1}{n} \sum_{c=1}^{N_c} \sum_{i \in \text{class } c} (\mathbf{x}_i - \overline{\mathbf{x}}_c)(\mathbf{x}_i - \overline{\mathbf{x}}_c)^T$$

donde:
N_c es el número de clases,
n_c el número de instancias en la clase c y
$\overline{\mathbf{x}}_c$ es la promedio de las instancias de clase c.

$$\overline{\mathbf{x}}_c = \frac{1}{n_c} \sum_{i \in \text{class } c} \mathbf{x}_i$$

La matriz S_w puede considerarse como la suma ponderada de las matrices de covarianza para cada una de las clases.

7.2.2.2. NDA: Análisis discriminante con formas no paramétricas

NDA son las siglas inglesas de *Nonparametric Discriminant Analysis*. En NDA [37] se sustituyen las matrices S_b y S_w de LDA por las siguientes:

$$A = S_b^{\text{NDA}} = \sum_{c=1}^{N_c} P_c \sum_{\substack{d=1 \\ d \neq c}}^{N_c} \sum_{i \in \text{class } c} \frac{w_i^{(c,d)}}{n_c} D_d(\mathbf{x}_i^{(c)}) \cdot D_d(\mathbf{x}_i^{(c)})^T$$

$$B = S_w^{\text{NDA}} = \sum_{c=1}^{N_c} P_c \sum_{i \in \text{class } c} \frac{w_i^{(c,c)}}{n_c} D_c(\mathbf{x}_i^{(c)}) \cdot D_c(\mathbf{x}_i^{(c)})^T$$

donde:
N_c es el número de clases,
n_c es el número de instancias en la clase c,
P_c es la probabilidad a priori de la clase c,

$\mathbf{x}_i^{(c)}$ es la instancia i-ésima de la clase c,
$D_d(\mathbf{x}_i^{(c)})$ es la media de la diferencia entre la instancia $\mathbf{x}_i^{(c)}$ y sus k vecinos más cercanos en la clase d que puede expresarse:

$$D_d(\mathbf{x}_i^{(c)}) = \mathbf{x}_i^{(c)} - M_d^k(\mathbf{x}_i^{(c)})$$

donde $M_d^k(\mathbf{x}_i^{(c)})$ es la media de los k vecinos más cercanos en la clase d a la instancia $\mathbf{x}_i^{(c)}$

$$M_d^k(\mathbf{x}_i^{(c)}) = \frac{1}{k}\sum_{t=1}^{k} \mathbf{x}_{\mathrm{tNN}}^{(d)}$$

y $w_i^{(c,d)}$ es un factor de ponderación que depende de un parámetro de control ρ (con un valor entre 0 e infinito) definido como:

$$w_i^{(c,d)} = \frac{\text{mín}\left\{\text{dist}(\mathbf{x}_i^{(c)}, \mathbf{x}_{\mathrm{kNN}}^{(c)})^\rho, \text{dist}(\mathbf{x}_i^{(c)}, \mathbf{x}_{\mathrm{kNN}}^{(d)})^\rho\right\}}{\text{dist}(\mathbf{x}_i^{(c)}, \mathbf{x}_{\mathrm{kNN}}^{(c)})^\rho + \text{dist}(\mathbf{x}_i^{(c)}, \mathbf{x}_{\mathrm{kNN}}^{(d)})^\rho}$$

donde $\text{dist}(\mathbf{x}_i^{(c)}, \mathbf{x}_{\mathrm{kNN}}^{(d)})$ es la distancia de $\mathbf{x}_i^{(c)}$ en la clase c a su k-ésimo vecino más cercano en la clase d.

7.2.2.3. HDA: Análisis discriminante híbrido

HDA son las siglas inglesas de *Hybrid Discriminant Analysis*. Este método es presentado en [107] como una combinación de PCA y LDA. La matrices del numerador y denominador de la Ecuación 7.1 se obtienen por combinación lineal de las correspondientes matrices en PCA y en LDA:

$$A = (1-\lambda)S_b + \lambda S_x$$
$$B = (1-\eta)S_w + \eta I$$

siendo I la matriz identidad. Cuando ambos parámetros valen la unidad ($\lambda = 1$ y $\eta = 1$), HDA se reduce a PCA; mientras que cuando ambos se anulan ($\lambda = 0$ y $\eta = 0$), HDA se corresponde con LDA puro, para otros valores podemos obtener proyecciones con características intermedias entre ambos métodos. Además, con $\eta > 0$ estamos obteniendo una regularización simple de S_w.

7.2.3. Métodos de proyección lineal supervisada para regresión

7.2.3.1. SIR: Regresión inversa por rebanadas

SIR son las siglas inglesas de *Sliced Inverse Regression*. En SIR [68] primero se ordena el conjunto de datos de acuerdo a los valores de y y se divide en L rebanadas, en inglés *slices*, este proceso puede verse como una discretización de los valores de y.

$$\begin{aligned} A = & S_\eta = \frac{1}{n}\sum_{l=1}^{L} n_l(\overline{\mathbf{x}}_l - \overline{\mathbf{x}})(\overline{\mathbf{x}}_l - \overline{\mathbf{x}})^T \\ B = & S_x \qquad (matriz\ de\ covarianzas) \end{aligned}$$

donde:
n_l sería el número de instancias en la rebanada l y
$\overline{\mathbf{x}}_l$ sería la media en cada rebanada.

$$\overline{\mathbf{x}}_l = \frac{1}{n_l}\sum_{i \in \text{slice } c} \mathbf{x}_i$$

Como matriz B se utiliza la matriz de covarianzas S_x. Si el cálculo de S_η se hace sobre el conjunto de datos una vez esferizado[3], la solución de la Ecuación 7.2 podría hacerse como un problema clásico de valores propios:

$$S_\eta \mathbf{w}_k = \lambda_k \mathbf{w}_k, \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$$

7.2.3.2. LSIR: SIR local

LSIR, en inglés *Localized SIR*,
es una variante de SIR [116] donde se sustituyen las medias de las rebanadas por medias locales.

$$\begin{aligned} A = & S_\eta^{\text{loc}} = \frac{1}{n}\sum_{i=1}^{n}(\overline{\mathbf{x}}_{i,\text{loc}} - \overline{\mathbf{x}})(\overline{\mathbf{x}}_{i,\text{loc}} - \overline{\mathbf{x}})^T \\ B = & S_x \qquad (matriz\ de\ covarianzas) \end{aligned}$$

[3]Es decir tras proyectarlo sobre las componentes principales y dividir cada variable por la raíz cuadrada del correspondiente valor propio.

donde $\overline{\mathbf{x}}_{i,\text{loc}}$ es la media de los vecinos más cercanos de la misma rebanada,

$$\overline{\mathbf{x}}_{i,\text{loc}} = \frac{1}{k}\sum_{j\in I_i}\mathbf{x}_j$$

donde I_i el conjunto de los índices de los k vecinos más cercanos de $\mathbf{x}_i$ en su misma rebanada, con lo que ahora el método depende de dos parámetros, el número de rebanadas, L, y el número de vecinos más cercanos, k.

7.2.3.3. PHD: Direcciones principales de la matriz hessiana

PHD son las siglas inglesas de *Principal Hessian Directions*. Este método [69, 70] se basa en resolver un problema de valores propios para el que es necesario calcular la media de las matrices hessianas $\overline{H}$, que está relacionada con la matriz de covarianzas ponderada

$$S_{yxx} = E\{(Y-\overline{y})(\mathbf{x}-\overline{\mathbf{x}})(\mathbf{x}-\overline{\mathbf{x}})^T\}$$

a través de la igualdad

$$\overline{H} = S_x^{-1}S_{yxx}S_x^{-1}$$

Desde el punto de vista del enfoque unificado que proponemos, este método equivaldría a resolver el problema de valores propios generalizado de la Ecuación 7.2 en el que las matrices A y B serían:

$$\begin{aligned} A = & S_{yxx} \quad = \frac{1}{n}\sum_{i=1}^{n}(y_i-\overline{y})(\mathbf{x}_i-\overline{\mathbf{x}})(\mathbf{x}_i-\overline{\mathbf{x}})^T \\ B = & S_x \qquad\qquad (\textit{matriz de covarianzas}) \end{aligned}$$

Como en el caso de SIR, si se esferiza [4] el conjunto de datos antes de calcular S_{yxx}, la solución podría obtenerse también resolviendo el problema de valores propios:

$$S_{yxx}\mathbf{w}_k = \lambda_k\mathbf{w}_k, \lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$$

[4] Esferizar es transformar un conjunto de datos para que tenga media 0 y varianza 1.

7.2.3.4. PCA ponderado

En el PCA ponderado [63], en inglés *Weighted PCA*, como en otros métodos, la matriz B es la matriz de covarianzas, S_x. La matriz A se define como:

$$A = S_{yx} = \frac{2}{n(n-1)} \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} g(y_i - y_j)(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T$$

$$B = S_x \qquad (\textit{matriz de covarianzas})$$

donde:
$g(\cdot)$ es una función positiva, simétrica y creciente.

Un posible ejemplo sería la función valor absoluto $g(x) = |x|$, esta expresión puede generalizarse como $g(x) = |x|^{-p}$, en la que p sería un parámetro del método, donde si p vale 1 tendríamos el ejemplo anterior o si p vale $\frac{1}{2}$ tendríamos $g(x) = \sqrt{|x|}$. Cuando p vale 0, la matriz S_{yx} sería equivalente a S_x, al anulase la dependencia de y.

7.2.3.5. LDAr: Análisis discriminante de forma lineal para regresión

LDAr, en inglés *LDA for regression*, es la adaptación para regresión de LDA propuesta en [63], que utiliza las siguientes variantes de las matrices S_b y S_w:

$$A = S_{br} = \frac{1}{n_b} \sum_{(i,j) \in I_{br}} f(y_i - y_j)(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T$$

$$B = S_{wr} = \frac{1}{n_w} \sum_{(i,j) \in I_{wr}} f(y_i - y_j)(\mathbf{x}_i - \mathbf{x}_j)(\mathbf{x}_i - \mathbf{x}_j)^T$$

donde los conjuntos de pares de índices I_{br} e I_{wr} se definen como:

$$I_{br} = \{(i,j) : |y_i - y_j| \geq \tau, i < j\}$$
$$I_{wr} = \{(i,j) : |y_i - y_j| < \tau, i < j\}$$

n_b y n_w son las cardinalidades de estos conjuntos,
$f(\cdot)$ podría ser cualquier función de las siguientes $f(x) = ||x| - \tau|$

o $f(x) = \sqrt{||x| - \tau|}$; como en WPCA, puede generalizarse en una función $f(x) = ||x| - \tau|^{-p}$, p sería un parámetro del método, y las anteriores serían casos especiales para $p = 1$ y $p = 0{,}5$.

7.3. Propuesta de nuevos métodos

Adaptando algunas de las ideas de los métodos anteriores, en este apartado se proponen nuevos métodos de proyección supervisada en problemas de regresión.

7.3.1. LPHD: Direcciones locales principales de la matriz hessiana

Este método se propone como una extensión de PHD, con la misma idea que utiliza Local SIR para extender SIR, se utiliza la información local en cada instancia. La nueva matriz para A sería:

$$\begin{aligned} A =& S_{yxx}^{\text{loc}} = \frac{1}{n}\sum_{i=1}^{n}(\overline{y}_{i,\text{loc}} - \overline{y})(\overline{\mathbf{x}}_{i,\text{loc}} - \overline{\mathbf{x}})(\overline{\mathbf{x}}_{i,\text{loc}} - \overline{\mathbf{x}})^T \\ B =& S_x \qquad\qquad\qquad\qquad (matriz\ de\ covarianzas) \end{aligned}$$

7.3.2. HDAr: Análisis discriminante híbrido para regresión

Este método propone utilizar la misma idea que en HDA, pero, en lugar de usar las matrices de PCA y LDA, utilizando las matrices de WPCA y LDAr,

$$\begin{aligned} A =& (1-\lambda)S_{br} + \lambda S_{yx} \\ B =& (1-\eta)S_{wr} + \eta I \end{aligned}$$

siendo I la matriz identidad. Por tanto, si ambos parámetros valen la unidad ($\lambda = 1$ y $\eta = 1$) HDAr se reduce a WPCA; mientras que si ambos parámetros se anulan ($\lambda = 0$ y $\eta = 0$)

HDAr se corresponde con LDAr puro; para otros valores podemos obtener proyecciones con características intermedias entre ambos métodos. Además, con $\eta > 0$ se consigue una regularización simple de S_{wr}.

7.3.3. SNDA: Análisis discriminante con formas no paramétricas para regresión

Esta propuesta consiste en primero hacer una fase previa de discretización de los valores de la variable dependiente en rebanadas, como se hace en SIR. Tras la discretización se considera la pertenencia a cada rebanada como una clase y esto permite aplicar NDA clásico.

7.4. Estudio comparativo

7.4.1. Validez de los nuevos métodos

Primero se comprobará la validez de las nuevas propuestas utilizando un par de conjuntos de datos artificiales para los que se conoce donde está la estructura, y por tanto es fácil validar que los métodos encuentran dicha estructura. Se han utilizado los mismos conjuntos de datos artificiales de [63], ambos de 1000 instancias y cinco dimensiones que siguen una distribución normal de media 0 y varianza 1. En uno de ellos la variable de salida depende linealmente de dos de los atributos $y = 2x_1 + 3x_3$, con lo que la dirección de proyección óptima sería $\mathbf{w}_1 = (2, 0, 3, 0, 0)^T$; en el otro la relación con la salida es no lineal $y = \sin(x_2 + 2x_4)$, y la proyección óptima es $\mathbf{w}_1 = (0, 1, 0, 2, 0)^T$.

Como referencia se calcularon también los resultados para los otros métodos. Se fijo el número de rebanadas en 12 (para SIR, LSIR y SNDA). El número de vecinos para los métodos localizados (LSIR y LPHD) ha sido establecido en 5. En WPCA, LDAr

Tabla 7.1: Valores de los parámetros del experimento.

	parámetros	
configuración	λ	η
HDAr55	0,5	0,5
HDAr83	0,8	0,3
HDAr38	0,3	0,8

Tabla 7.2: Valor absoluto del coseno del ángulo entre la dirección óptima y la encontrada por los métodos de proyección supervisada para regresión.

#	Problema lineal	#	Problema no lineal	Método
5	0,999 933	4	0,986 327	SIR
7	0,999 705	2	0,995 380	LSIR
3	0,999 958	5	0,952 026	WPCA
1	0,999 999	3	0,991 270	LDAr
10	0,376 449	9	0,800 947	PHD
9	0,446 795	10	0,343 918	LPHD
6	0,999 774	7	0,913 931	HDAr55
4	0,999 949	6	0,948 631	HDAr83
8	0,998 483	8	0,865 834	HDAr38
2	0,999 973	1	0,997 950	SNDA

y HDAr se ha utilizado 0,5 como valor del parámetro p. Como valor de τ, en LDAr y HDAr, se ha usado 0,3. Para el método HDAr se han probado tres configuraciones de los parámetros λ y η, que denominaremos HDAr55, HDAr83 y HDAr38 con los valores de la Tabla 7.1

En la Tabla 7.2 se muestra el valor absoluto del coseno del ángulo entre las direcciones óptimas y la dirección encontrada por cada método. Puede verse que tanto SNDA como las diversas configuraciones de HDAr consiguen buenas aproximaciones al óptimo tanto en el caso lineal como el no lineal. En el caso lineal la mejor aproximación la da LDAr seguido de cerca de SNDA. Además, la versión local de PHD consigue mejorar ligeramente a PHD para el caso lineal, aunque sus resultados son muy malos en el no lineal. La mejor aproximación en el problema no lineal

es la obtenida por SNDA.

7.4.2. Comparación experimental

Para el estudio de este capítulo se han utilizado 30 conjuntos de datos para regresión de la UCI [71], los se muestran en la Tabla 7.3.

Los resultados de predicción se han obtenido con un regresor ponderado de vecinos más cercanos (el mismo regresor usado en [63]), que normaliza los atributos al rango $[0, 1]$, utiliza 5 vecinos y la siguiente función de ponderación:

$$q(\mathbf{x}, \mathbf{x}_i) = \frac{1}{\left(1 + \sqrt{||\mathbf{x} - \mathbf{x}_i||}\right)}$$

Se ha comprobado el efecto del tamaño de la dimensión de la proyección, usando como dimensiones tanto los valores fijos (1, 2, 3) como valores que dependen de la dimensión original del conjunto de datos, d (50 % d, 75 % d y 100 % d), redondeando los valores no enteros al más próximo.

Se ha utilizado validación cruzada 1×10 particiones en estos experimentos, se ha divido aleatoriamente cada conjunto de datos en un 90 % para entrenamiento y un 10 % para test, y esto se ha repetido 10 veces calculando el error medio de cada repetición como la media cuadrática del error.

Para cada una de las dimensiones se han ordenado los métodos en función de los resultados del regresor, asignando el puesto 1 al mejor, el 2 al siguiente y así sucesivamente [17]. Los puestos obtenidos para todos los conjuntos de datos se utilizaron para calcular los puestos promedio, que se muestra en la Tabla 7.4. Uno de los métodos propuestos, SNDA, es el mejor método cuando se utiliza para proyectar el conjunto de datos sin reducir su dimensión y cuando se utiliza para reducir la dimensión al 75 % de la dimensión original. También queda entre los tres primeros en otros 3 casos. Además, se puede ver que la propuesta de

Tabla 7.3: Los conjuntos de datos usados en este experimento.

Conjunto de datos	Nº atributos numéricos	discretos	Instancias
abalone	7	1	4177
auto93	16	6	93
auto-horse	17	8	205
auto-mpg	4	3	398
auto-price	15	0	159
bodyfat	14	0	256
breast-tumor	1	8	286
cholesterol	6	7	303
cleveland	6	7	303
cloud	4	2	108
cpu	6	1	209
cpu-small	12	0	8192
delta-ailerons	5	0	7129
echo-months	6	3	130
fishcatch	5	2	158
housing	12	1	506
hungarian	6	7	294
lowbwt	2	7	189
machine-cpu	6	0	209
meta	19	2	528
pbc	10	8	418
pharynx	1	10	195
pw-linear	10	0	200
sensory	0	11	576
servo	0	4	167
stock	9	0	950
strike	5	1	625
triazines	60	0	186
veteran	3	4	137
wisconsin	32	0	194

Tabla 7.4: Clasificación de los métodos para cada una de las dimensiones considerada.

	1		2		3
4,27	WPCA	4,33	SIR	3,80	WPCA
4,53	SIR	4,47	WPCA	5,27	SIR
5,37	SNDA	4,90	SNDA	5,27	LSIR
5,47	LDAr	5,17	LDAr	5,30	LDAr
5,80	HDAr55	5,57	HDAr83	5,40	SNDA
5,83	HDAr83	6,13	HDAr55	5,60	HDAr83
6,33	LSIR	6,67	LSIR	6,27	HDAr55
6,40	HDAr38	6,67	HDAr38	6,57	HDAr38
7,17	PHD	6,67	PCA	7,03	PCA
7,37	PCA	7,63	PHD	7,73	LPHD
7,47	LPHD	7,80	LPHD	7,77	PHD

	$.5d$		$.75d$		d
4,30	WPCA	4,80	SNDA	4,70	SNDA
4,73	SNDA	4,87	WPCA	5,07	WPCA
4,97	LDAr	4,97	SIR	5,60	HDAr83
5,07	LSIR	5,10	LSIR	5,70	SIR
5,10	SIR	5,60	LDAr	5,73	LDAr
6,40	HDAr83	5,73	HDAr83	6,00	LSIR
6,47	PCA	6,37	HDAr55	6,07	PCA
6,60	HDAr55	6,57	PCA	6,20	HDAr55
7,37	PHD	7,23	HDAr38	6,37	HDAr38
7,37	LPHD	7,33	LPHD	7,17	PHD
7,63	HDAr38	7,43	PHD	7,40	LPHD

Tabla 7.5: Clasificación global de los métodos.

20,77	$_{3}$WPCA				
22,27	$_{,5d}$WPCA	28,97	$_{3}$LDAr	36,87	$_{1}$WPCA
23,07	$_{,75d}$SNDA	29,27	$_{2}$SNDA	37,13	$_{,75d}$LPHD
23,43	$_{,75d}$WPCA	29,60	$_{3}$LSIR	37,57	$_{1}$SNDA
23,53	$_{d}$SNDA	30,30	$_{d}$HDAr55	38,47	$_{3}$PCA
24,00	**ORI**	30,37	$_{,75d}$HDAr83	38,50	$_{,5d}$HDAr38
24,47	$_{d}$WPCA	31,07	$_{d}$HDAr38	38,67	$_{2}$LSIR
24,90	$_{,5d}$SNDA	31,83	$_{3}$HDAr83	38,83	$_{,5d}$LPHD
25,57	$_{,75d}$LSIR	31,90	$_{2}$LDAr	39,40	$_{2}$HDAr38
25,73	$_{,5d}$SIR	32,60	$_{,5d}$HDAr83	39,50	$_{,5d}$PHD
25,97	$_{,75d}$SIR	32,67	$_{,75d}$HDAr55	39,80	$_{2}$PCA
26,13	$_{,5d}$LDAr	33,07	$_{,75d}$PCA	40,37	$_{1}$LDAr
26,37	$_{,5d}$LSIR	33,63	$_{,5d}$PCA	43,00	$_{1}$HDAr55
27,03	$_{d}$SIR	33,73	$_{,5d}$HDAr55	43,30	$_{3}$PHD
27,40	$_{2}$WPCA	33,77	$_{3}$HDAr55	44,10	$_{1}$HDAr83
27,43	$_{d}$LDAr	34,30	$_{2}$HDAr83	44,67	$_{3}$LPHD
27,73	$_{3}$SIR	35,63	$_{d}$PHD	46,47	$_{1}$LSIR
27,77	$_{d}$HDAr83	35,67	$_{1}$SIR	46,63	$_{1}$HDAr38
28,00	$_{,75d}$LDAr	36,00	$_{,75d}$HDAr38	47,13	$_{2}$PHD
28,33	$_{2}$SIR	36,17	$_{3}$HDAr38	50,07	$_{2}$LPHD
28,63	$_{3}$SNDA	36,30	$_{d}$LPHD	53,10	$_{1}$PCA
28,70	$_{d}$LSIR	36,47	$_{,75d}$PHD	53,60	$_{1}$PHD
28,77	$_{d}$PCA	36,67	$_{2}$HDAr55	54,87	$_{1}$LPHD

HDAr no es muy buena idea, dado que en ningún caso se ha conseguido batir simultáneamente a los métodos de los que parte (WPCA y LDAr). La localización de PHD tampoco parece aportar mucho, aunque se ha superado al PHD en dos casos, en los restantes los resultados han sido peores. WPCA parece ser el mejor método quedando para todas las dimensiones entre los dos mejores métodos, siempre por encima de LDAr lo que contradice las conclusiones de los inventores del método, que establecían en [63] que LDAr era mejor que WPCA, tal vez por que solo utilizaron tres conjuntos de datos.

Por último, también se han calculado globalmente los rangos de las 66 combinaciones de métodos y posibles dimensiones (6

dimensiones de proyección diferentes $\times$ 11 métodos) junto con el resultado de aplicar el regresor base directamente al conjunto de datos (en las tablas denotado como ORI). Estos resultados se muestran en la Tabla 7.5. Aquí se nos presenta un resultado sorprendente, la mayoría de los métodos no logran superar los resultados de aplicar el regresor base directamente sobre el conjunto de partida sin ninguna reducción de dimensionalidad involucrada.

7.5. Conclusiones

En este capítulo se han descrito algunos de los métodos clásicos de obtención de proyecciones lineales supervisados para problemas de regresión, junto con algunas nuevas propuestas, utilizando el marco conceptual común de la resolución del problema de valor propio generalizado.

Tras comprobar la validez de las nuevas propuestas sobre un par de conjuntos de datos artificiales de estructura conocida, se ha hecho un estudio experimental de todos los métodos utilizando 30 conjuntos de datos. La conclusión más sorprendente de este estudio es que muchos de los métodos de proyección no consiguen mejorar los resultados de regresión del regresor utilizado como base del estudio, un regresor de vecinos más cercanos ponderado.

El SNDA, uno de los nuevo métodos propuestos en este capítulo, junto con el WPCA, son los únicos que mejoran al regresor original, con resultados comparables. También es destacable que WPCA sea mejor que LDAr, lo que contradice los resultados de [63], donde LDAr superaba a WPCA.

Un posible trabajo futuro sería poder determinar si las conclusiones aquí obtenidas pueden extenderse para casos en los que se usan otros regresores, así como considerar el efecto de los parámetros de los métodos. Otra línea de trabajo interesan-

te sería utilizar estos métodos como inductores de diversidad en los algoritmos de construcción de multiregresores. Esto vendría motivado por los resultados obtenidos por *Rotation Forest* utilizando PCA [94], o *Nonlinear Boosting Projection* utilizando NDA [39]. Es tentador pensar que para problemas de regresión la sustitución de PCA y NDA por alguno de los métodos propuestos en este capítulo podría mejorar los resultados.

Capítulo 8

Conclusiones de la tesis

En esta tesis se han estudiado los algoritmos básicos y las principales combinaciones de modelos para tareas de regresión.

Se han realizado experimentos analizando su comportamiento, validándolo con una gran colección de conjuntos de datos, los 61 publicados por la UCI [71].

De estos estudios no se puede concluir que una combinación sea siempre la mejor, ya que hay variaciones en función de si el método base utilizado es un perceptrón multicapa , un *Random Oracle*, una función de base radial o una máquina de soporte vectorial y también varía en función de la configuración de estos métodos base.

Además de la adaptación para poder usarlo en tareas de regresión del *Random Oracle*, también se ha adaptado a esas tareas otro algoritmo, el *Rotation Forest*. Se ha comparado su rendimiento y cómo es la diversidad de los modelos generados. *Random Oracle* mejora a otras combinaciones cuando se usa como método base y *Rotation Forest* funciona mejor que las otras combinaciones, para el caso estudiado, usando *árboles de regresión* como método base.

Por último, se han estudiado los métodos de proyección lineal existentes, se ha unificado su nomenclatura y se han propuestos variantes nuevas. Aplicando las proyecciones estudiadas al

regresor de *vecinos más cercanos*, sólo dos mejoraban los resultados de dicho regresor y una era una de las nuevas variantes propuestas, SNDA. Este estudio de proyecciones lineales es el inicio, inconcluso, de una nueva línea de investigación que busca probar alternativas a PCA en la obtención de árboles para el *Rotation Forest*.

Se han documentado estos estudios y a partir de los resultados se han redactado varias publicaciones:

- Un artículo de revista del primer cuartil de su área,
- Un capítulo de libro,
- Tres contribuciones en congresos internacionales y
- Una contribución en un congreso nacional.

Se deja como tarea para futuros trabajos

- Estudiar si se puede concluir que la mejora de funcionamiento de las combinaciones es función de alguna característica de los conjunto de datos.
- Analizar el comportamiento de otros algoritmos de regresión si se cambia el método de proyección, y en especial analizar el comportamiento del *Rotation Forest* si en lugar de usar análisis de componentes principales se usa otra proyección lineal.
- Estudiar el comportamiento de las combinaciones de modelos a otros conjuntos de datos, por ejemplo, a datos de interacción de los discentes en la educación en la red.
- Estudiar el uso de combinaciones de algoritmos en problemas de *big data*.

Bibliografía

[1] Batchelor, B.G.: Classification and data analysis in vector spaces. In: Pattern Recognition, pp. 65–116. Springer (1978)

[2] Bellmam, R.E.: Dynamic Programming. Princeton University Press (1957)

[3] Bengio, Y.: Learning deep architectures for ai. Foundations and trends in Machine Learning 2(1), 1–127 (2009)

[4] Biberman, Y.: A context similarity measure. In: Bergadano, F., De Raedt, L. (eds.) Machine Learning: ECML-94, Lecture Notes in Computer Science, vol. 784, pp. 49–63. Springer Berlin Heidelberg (1994), http://dx.doi.org/10.1007/3-540-57868-4_50

[5] Billings, S.A., Zheng, G.L.: Radial basis function network configuration using genetic algorithms. Neural Networks 8(6), 877–890 (1995)

[6] Bishop, C.M.: Neural networks for pattern recognition. Oxford university press (1995)

[7] Bishop, C.M.: Pattern recognition and machine learning. Springer, 2nd edn. (2006)

[8] Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)

[9] Breiman, L.: Random forests. Machine Learning 45(1), 5–32 (2001), citeseer.ist.psu.edu/breiman01random.html

[10] Breiman, L.: Using iterated bagging to debias regressions. Machine Learning 45(3), 261–277 (Dec 2001), http://dx.doi.org/10.1023/A:1017934522171

[11] Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Statistics/Probability Series, Wadsworth Publishing Company, Belmont, California, U.S.A. (1984)

[12] Breiman, L., Friedman, J.H.: Estimating optimal transformations for multiple regression and correlation. Journal of the American Statistical Association 80(391), 580–598 (1985)

[13] Brown, G., Wyatt, J.L., Tiňo, P.: Managing diversity in regression ensembles. Journal of Machine Learning Research 6, 1621–1650 (September 2005), http://www.jmlr.org/papers/volume6/brown05a/brown05a.pdf

[14] Chang, Y.H.O., Ayyub, B.M.: Fuzzy regression methods – a comparative assessment. Fuzzy Sets and Systems 119(2), 187 – 203 (2001)

[15] Chen, S., Cowan, C.F., Grant, P.M.: Orthogonal least squares learning algorithm for radial basis function networks. Neural Networks, IEEE Transactions on 2(2), 302–309 (1991)

[16] Craven, M.W., Shavlik, J.W.: Using neural networks for data mining. Future generation computer systems 13(2), 211–229 (1997)

[17] Demšar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research 7, 1–30 (2006)

[18] Diday, E.: Recent progress in distance and similarity measures in pattern recognition. In: Second International Joint Conference on Pattern Recognition. pp. 534–539 (1974)

[19] Dietterich, T.G.: Approximate statistical test for comparing supervised classification learning algorithms. Neural Computation 10(7), 1895–1923 (1998)

[20] Díez-Pastor, J.F., Rodríguez, J.J., García-Osorio, C.I., Kuncheva, L.I.: Diversity techniques improve the performance of the best imbalance learning ensembles. Information Sciences 325, 98 – 117 (2015), `http://www.sciencedirect.com/science/article/pii/S0020025515005186`

[21] Díez-Pastor, J.F.: Estudio de métodos de construcción de ensembles de clasificadores y aplicaciones. Master's thesis, Universidad de Burgos (2015)

[22] Ding, S., Zhao, H., Zhang, Y., Xu, X., Nie, R.: Extreme learning machine: algorithm, theory and applications. Artificial Intelligence Review pp. 1–13 (2013)

[23] Drucker, H.: Improving regressors using boosting techniques. In: ICML '97: Proceedings of the Fourteenth International Conference on Machine Learning. pp. 107–115. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1997), `http://portal.acm.org/citation.cfm?id=645526.657132`

[24] Drucker, H., Burges, C.J.C., Kaufman, L., Smola, A., Vapnik, V.: Support vector regression machines. In: Advances

in Neural Information Processing Systems 9. pp. 155–161. MIT Press (1997)

[25] Elomaa, T., Kääriäinen, M.: An analysis of reduced error pruning. Journal of Artificial Intelligence Research 15, 163–187 (2001)

[26] Fan, J., Gijbels, I.: Local Polynomial Modelling and Its Applications. Monographs on Statistics and Applied Probability, v. 66, Chapman and Hall, New York (1996)

[27] Faraway, J.J.: Linear Models with R. Texts in statistical science, v. 63, Chapman & Hall/CRC, Boca Raton, FL (2004), http://www.maths.bath.ac.uk/~jjf23/LMR/, iSBN 1-584-88425-8

[28] Fisher, R.A.: The goodness of fit of regression formulae, and the distribution of regression coefficients. Journal of the Royal Statistical Society (1922)

[29] Fisher, R.A.: Statistical methods for research workers. Genesis Publishing Pvt Ltd (1925)

[30] Fisher, R.A., et al.: The use of multiple measurements in taxonomic problems. Annals of eugenics 7(2), 179–188 (1936)

[31] Frazee, A.C., Hathcock, M.A., Prins, S.C.B.: Distance functions and attribute weighting in a k-nearest neighbors classifier with an ecological application. Electronic Proceedings of Undergraduate Mathematics Day 4(3), 1–13 (2010)

[32] Freeman, J.A., Skapura, D.M.: Algorithms, Applications, and Programming Techniques. Citeseer (1991)

[33] Freund, Y., Schapire, R.E.: Experiments with a new boosting algorithm. In: 13th International Conference on Machine Learning. pp. 148–156. Morgan Kaufmann, San Francisco (1996)

[34] Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences 55(1), 119–139 (1997)

[35] Friedman, J.H.: Multivariate adaptive regression splines. The Annals of Statistics 19(1), 1–141 (03 1991)

[36] Friedman, J.H., Stuetzle, W.: Projection pursuit regression. Journal of the American Statistical Association 76, 817–823 (1981)

[37] Fukunaga, K., Mantock, J.: Nonparametric discriminant analysis. IEEE Transaction on Pattern Analysis and Machine Intelligence 6(5), 671–678 (1983)

[38] Galton, F.: Typical laws of heredity. Nature 15 (1877)

[39] García-Pedrajas, N., García-Osorio, C.: Constructing ensembles of classifiers using supervised projection methods based on misclassified instances. Expert Systems with Applications 38(1), 343–359 (2011), http://www.sciencedirect.com/science/article/B6V03-50GJ2J0-7/2/6b1890282b8bfb900f1174dc7a027a9c, dOI: 10.1016/j.eswa.2010.06.072

[40] García-Pedrajas, N., Hervás-Martínez, C., Ortiz-Boyer, D.: Cooperative coevolution of artificial neural network ensembles for pattern classification. Evolutionary Computation, IEEE Transactions on 9(3), 271–302 (2005)

[41] Gauss, C.F.: Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientum. F. Didot Paris (1809)

[42] Gini, C.: Variability and mutability, contribution to the study of statistical distribution and relaitons. Studi Economico-Giuricici della R (1912)

[43] Gomm, J.B., Yu, D.L.: Selecting radial basis function network centers with recursive orthogonal least squares training. Neural Networks, IEEE Transactions on 11(2), 306–314 (2000)

[44] González, J.: Identificación y optimización de redes de funciones de base radiales para aproximación funcional. Ph.D. thesis, Ph. D. dissertation, Univ. Granada, Spain (2001)

[45] Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. Journal of Machine Learning Research 3, 1157–1182 (March 2003), http://portal.acm.org/citation.cfm?id=944919.944968

[46] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: An update. SIGKDD Explorations 11(1), 10–18 (2009)

[47] Härdle, W., Stoker, T.M.: Investigating smooth multiple regression by the method of average derivatives. Journal of the American Statistical Association 84(408), 986–995 (1989)

[48] Hastie, T., Tibshirani, R.: Discriminant adaptive nearest neihgbor classification and regression. In: Touretzky, D., Mozer, M.C., Hasselmo, M. (eds.) Advances in Neural Information Processing Systems, vol. 8, pp. 290–297. MIT Press (1996)

[49] Hastie, T., Tibshirani, R.: Generalized additive models. Statistical Science 1(3), 297–310 (08 1986)

[50] Haykin, S.: Neural Networks: A Comprehensive Foundation. Prentice Hall (1998)

[51] Haykin, S.S., Haykin, S.S., Haykin, S.S., Haykin, S.S.: Neural networks and learning machines, vol. 3. Pearson Education Upper Saddle River (2009)

[52] Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural computation 18(7), 1527–1554 (2006)

[53] Ho, T.K.: The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence 20(8), 832–844 (1998), `http://citeseer.ist.psu.edu/ho98random.html`

[54] Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: a new learning scheme of feedforward neural networks. In: Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on. vol. 2, pp. 985–990 vol.2 (July 2004)

[55] Jain, A., Mao, J., Mohiuddin, K.: Artificial neural networks: a tutorial. Computer 29(3), 31–44 (Mar 1996)

[56] Jolliffe, I.: Principal Component Analysis. Springer-Verlag (1986)

[57] Krogh, A., Vedelsby, J.: Neural network ensembles, cross validation, and active learning. In: Advances in Neural Information Processing Systems. vol. 7, pp. 231–238. MIT Press, Cambridge MA (1995)

[58] Krogh, A., Vedelsby, J., et al.: Neural network ensembles, cross validation, and active learning. Advances in neural information processing systems 7, 231–238 (1995)

[59] Kuncheva, L.I.: Combining Pattern Classifiers: Methods and Algorithms. Wiley-Interscience (July 2004)

[60] Kuncheva, L.I.: Combining Pattern Classifiers. Wiley, 2nd edition edn. (2014)

[61] Kuncheva, L.I., Whitaker, C.J.: Measures of diversity in classifier ensembles. Machine Learning 51(2), 181–207 (2003)

[62] Kuncheva, L., Rodriguez, J.: Classifier ensembles with a random linear oracle. Knowledge and Data Engineering, IEEE Transactions on 19(4), 500–508 (April 2007)

[63] Kwak, N., Lee, J.W.: Feature extraction based on subspace methods for regression problems. Neurocomputing 73(10-12), 1740–1751 (2010)

[64] Kwok, T.Y., Yeung, D.Y.: Constructive algorithms for structure learning in feedforward neural networks for regression problems. Neural Networks, IEEE Transactions on 8(3), 630–645 (1997)

[65] LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature 521(7553), 436–444 (2015)

[66] Lee, J.A., Verleysen, M.: Nonlinear Dimensionality Reduction. Springer (2007)

[67] Legendre, A.M.: Nouvelles methodes pour la determination des orbites des cometes. F. Didot Paris (1805)

[68] Li, K.C.: Sliced inverse regression for dimension reduction. Journal of the American Statistical Association 86(414), 316–327 (1991), http://www.jstor.org/stable/2290563

[69] Li, K.C.: On principal hessian directions for data visualization and dimension reduction: Another application of stein's lemma. Journal of the American Statistical Association 84(420), 1025–1039 (1992), stable URL: http://www.jstor.org/stable/229064

[70] Li, K.C.: High dimensional data analysis via the SIR/PHD approach. Available at http://www.stat.ucla.edu/~kcli/sir-PHD.pdf (2000)

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

[72] Liu, H., Yu, L.: Toward integrating feature selection algorithms for classification and clustering. IEEE Transanction on Knowledge and Data Engineering 17, 491–502 (2005)

[73] Liu, Y., Yao, X.: Ensemble learning via negative correlation. Neural Networks 12(10), 1399 – 1404 (1999), http://www.sciencedirect.com/science/article/pii/S0893608099000738

[74] Loftsgaarden, D.O., Quesenberry, C.P.: A nonparametric estimate of a multivariate density function. The Annals of Mathematical Statistics 36(3), 1049–1051 (06 1965)

[75] Longoni, M.G., Porcel, E., López, M.V., Dapozo, G.N.: Modelos de redes neuronales perceptrón multicapa y de base radial para la predicción del rendimiento académico de alumnos universitarios. In: XVI Congreso Argentino de Ciencias de la Computación (2010)

[76] Lu, H., Setiono, R., Liu, H.: Effective data mining using neural networks. Knowledge and Data Engineering, IEEE Transactions on 8(6), 957–961 (1996)

[77] Margineantu, D.D., Dietterich, T.G.: Pruning adaptive boosting. In: Proc. 14th International Conference on Machine Learning. pp. 211–218. Morgan Kaufmann (1997)

[78] Maudes, J., Rodríguez, J.J., Garciía-Osorio, C., García-Pedrajas, N.: Random feature weights for decision tree ensemble construction. Information Fusion 13(1), 20–30 (2012), http://dx.doi.org/10.1016/j.inffus.2010.11.004

[79] Michalski, R., Stepp, R.E., Diday, E.: A recent advance in data analysis: Clustering objects into classes characterized by conjunctive concepts. Progress in pattern recognition 1, 33–56 (1981)

[80] Nadaraya, E.: On estimating regression. Theory of Probability & Its Applications 9(1), 141–142 (1964)

[81] Nadeau, C., Bengio, Y.: Inference for the generalization error. Machine Learning 52(3) (2003)

[82] Nadler, M., Smith, E.P.: Pattern recognition engineering. Wiley-Interscience (1993)

[83] Pardo, C., Diez-Pastor, J.F., García-Osorio, C., Rodríguez, J.J.: Rotation forest for regression. Applied Mathematics and Computation 219(19), 9914–9924 (Jun 2013), http://www.sciencedirect.com/science/article/pii/S0096300313004128

[84] Pardo, C., Rodríguez, J.J., Díez-Pastor, J.F., García-Osorio, C.: Random oracles for regression ensembles. In: Workshop on Supervised and Unsupervised Ensemble

Methods and their Applications, SUEMA 2010. pp. 85–96 (2010)

[85] Pardo, C., Rodríguez, J.J., Díez-Pastor, J.F., García-Osorio, C.: Random oracles for regression ensembles. In: Okun, O., Valentini, G., Re, M. (eds.) Ensembles in Machine Learning Applications, Studies in Computational Intelligence, vol. 373, pp. 181–199. Springer Berlin / Heidelberg (2011)

[86] Pardo, C., Rodríguez, J.J., García-Osorio, C., Maudes, J.: An empirical study of multilayer perceptron ensembles for regression tasks. In: García-Pedrajas, N., Herrera, F., Fyfe, C., Benítez, J., Ali, M. (eds.) Trends in Applied Intelligent Systems, Lecture Notes in Computer Science, vol. 6097, pp. 106–115. Springer Berlin Heidelberg (2010), `http://dx.doi.org/10.1007/978-3-642-13025-0_12`

[87] Pardo-Aguilar, C., Díez-Pastor, J.F., García-Pedrajas, N., Rodríguez, J.J., García-Osorio, C.: Linear projection methods-an experimental study for regression problems. In: ICPRAM (1). pp. 198–204 (2012)

[88] Pardo-Aguilar, C., Rodríguez, J.J., García-Osorio, C., Díez-Pastor, J.F.: Un estudio experimental de combinaciones usando redes de funciones de base radial en tareas de regresión. In: admitido en CAEPIA (2015)

[89] Periche, F.M.F.: aproximación funcional mediante redes de funciones de base radial, una alternativa para la predicción en el proceso de reducción de mineral de la tecnología Caron de producción de níquel. Ph.D. thesis, Universidad de Granada (2008)

[90] Rafajlowicz, E.: Nonparametric orthogonal series estimators of regression: A class attaining the optimal convergen-

ce rate in {L2}. Statistics & Probability Letters 5(3), 219 – 224 (1987)

[91] Redden, D.T., Woodall, W.H.: Further examination of fuzzy linear regression. Fuzzy Sets and Systems 79(2), 203 – 211 (1996)

[92] Rivas, A.J.R.: Diseño y optimización de redes de funciones de base radial mediante técnicas bioinspiradas. Ph.D. thesis, Tesis Doctoral, universidad de Granada (2003)

[93] Rodríguez, J.J., García-Osorio, C., Maudes, J.: Forests of nested dichotomies. Pattern Recognition Letters 31(2), 125–132 (2010), http://dx.doi.org/10.1016/j.patrec.2009.09.015

[94] Rodríguez, J.J., Kuncheva, L.I., Alonso, C.J.: Rotation forest: A new classifier ensemble method. IEEE Transactions on Pattern Analysis and Machine Intelligence 28(10), 1619–1630 (Oct 2006), http://doi.ieeecomputersociety.org/10.1109/TPAMI.2006.211

[95] Rodríguez, J.J., Maudes, J., Pardo, C., , García-Osorio, C.: Disturbing neighbors ensembles for regression. In: XIII Conference of the Spanish Association for Artificial Intelligence, CAEPIA - TTIA 2009. pp. 369–378 (2009)

[96] Rojas, R.: Neural networks: a systematic introduction. Springer Science & Business Media (2013)

[97] Rossomando, F.G., Soria, C., Carelli, R.: Control de robots móviles con incertidumbres dinámicas usando redes de base radial. Revista Iberoamericana de Automática e Informática Industrial RIAI 7(4), 28–35 (2010)

[98] Roweis, S.T., Saul, L.K.: Nonlinear Dimensionality Reduction by Locally Linear Embedding. Science 290(5500), 2323–2326 (Dec 22 2000)

[99] Samarasinghe, S.: Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition. CRC Press (2006)

[100] Shrestha, D.L., Solomatine, D.P.: Experiments with AdaBoost.RT, an improved boosting scheme for regression. Neural Computation 18(7), 1678–1710 (2006), `http://dx.doi.org/10.1162/neco.2006.18.7.1678`

[101] Smola, Alex J.; Schölkopf, B.: A tutorial on support vector regression. Statistics and Computing 14(3), 199–222 (2004)

[102] Specht, D.F.: A general regression neural network. IEEE Transactions on Neural Networks 2(6), 568–576 (1991)

[103] Stephens, L.J., Spiegel, M.R.: Estadística. Schaum, McGraw-Hill, 4 edn. (2009)

[104] Suen, Y., Melville, P., Mooney, R.: Combining bias and variance reduction techniques for regression trees. In: Machine Learning: ECML 2005. pp. 741–749. Springer (2005), `http://dx.doi.org/10.1007/11564096_76`

[105] Tanaka, H., Uejima, S., Asai, K.: Linear regression analysis with fuzzy model. IEEE Transactions on Systems, Man and Cybernetics SMC-12(6), 903–907 (1982)

[106] Tenenbaum, J.B., de Silva, V., Langford, J.C.: A Global Geometric Framework for Nonlinear Dimensionality Reduction. Science 290(5500), 2319–2323 (Dec 22 2000)

[107] Tian, Q., Yu, J., Huang, T.S.: Boosting multiple classifiers constructed by hybrid discriminant analysis. In: Oza, N.C., Polikar, R., Kittler, J., Roli, F. (eds.) Multiple Classifier Systems. Lecture Notes in Computer Science, vol. 3541, pp. 42–52. Springer, Seaside, CA, USA (June 2005)

[108] Vijayakumar, S., Schaal, S.: Approximate nearest neighbor regression in very high dimensions. Nearest-Neighbor Methods in Learning and Vision: Theory and Practice pp. 103–142 (2006)

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

[110] Watson, G.S.: Smooth regression analysis. Sankhyā: The Indian Journal of Statistics 26, 359–372 (1964)

[111] Wettschereck, D., Aha, D.W., Mohri, T.: A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. Artificial Intelligence Review 11(1-5), 273–314 (1997)

[112] de Wikipedia, C.: Determining the number of clusters in a data set (2015), `http://en.wikipedia.org/w/index.php?title=Determining_the_number_of_clusters_in_a_data_set&oldid=661006527#Rule_of_thumb`, [Online; accessed 21-May-2015]

[113] de Wikipedia, C.: Regression analysis (2015), `https://en.wikipedia.org/w/index.php?title=Regression_analysis&oldid=672767562`, [Online; accessed 12-Aug-2015]

[114] Wilson, D.R., Martinez, T.R.: Improved heterogeneous distance functions. Journal of artificial intelligence research pp. 1–34 (1997)

[115] Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. Morgan Kaufmann, 2nd edn. (2005), http://www.cs.waikato.ac.nz/ml/weka

[116] Wu, Q., Mukherjee, S., Liang, F.: Localized sliced inverse regression. In: Koller, D., Schuurmans, D., Bengio, Y., Bottou, L. (eds.) NIPS. pp. 1785–1792. MIT Press (2008)

[117] Yee, P.V., Haykin, S.: Regularized Radial Basis Function Networks: Theory and Applications. Wiley-Interscience (2001)

[118] Zhang, C., Zhang, J., Wang, G.: An empirical study of using rotation forest to improve regressors. Applied Mathematics and Computation 195(2), 618–629 (Feb 2008), http://dx.doi.org/10.1016/j.amc.2007.05.010

Índice alfabético

D

E

F

G

H

I

K

L

M

N

O

P

Q

U

V

W


Advanced Data MIning Research And
Learning
Business intelligence
bioinformatics
big data
A!